DISK INCLUDED

Includes Fractint V18 and FDESIGN

CD-ROM INCLUDED

Includes 1,800 Fractal Images plus source code for DOS, Unix

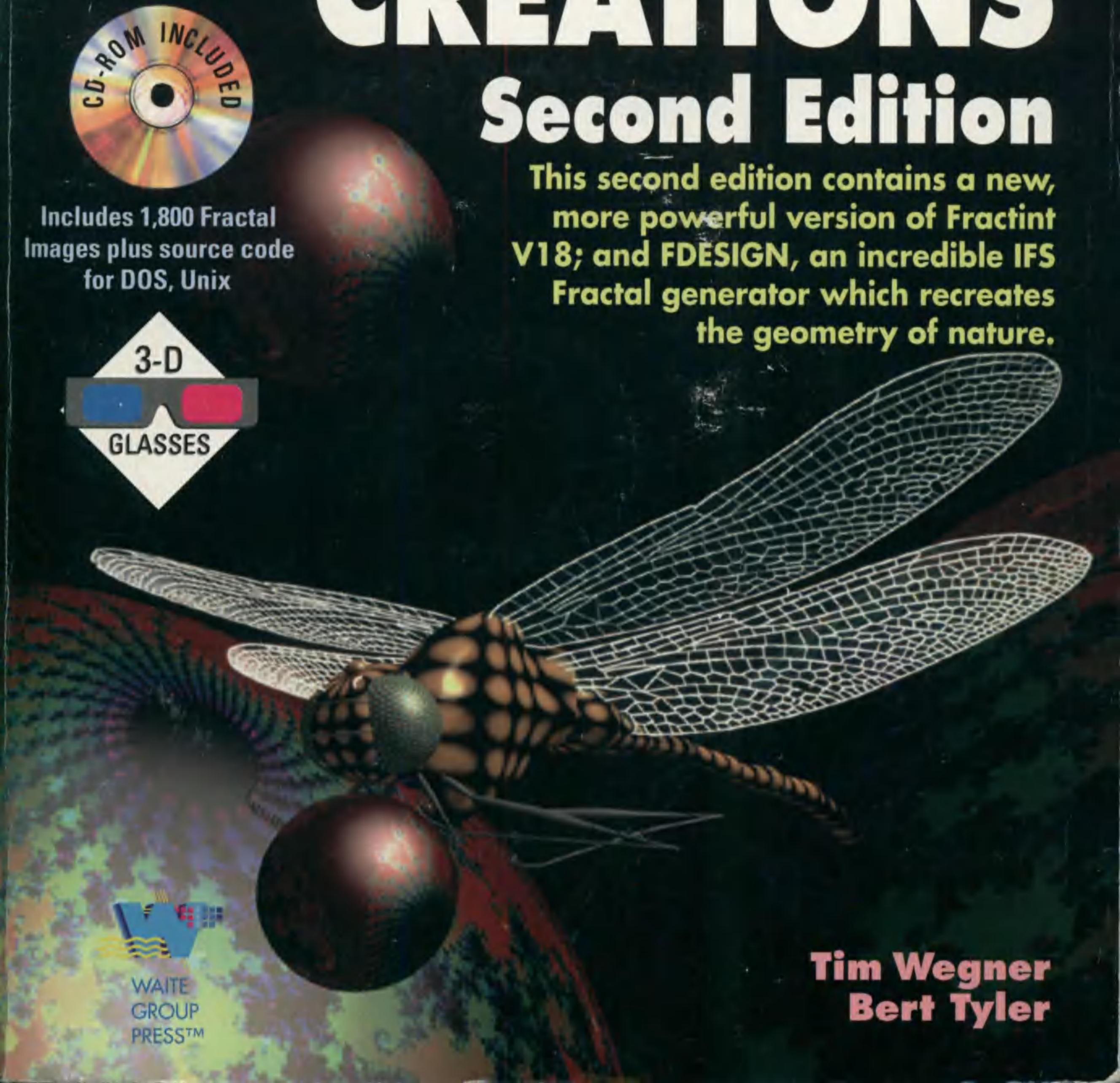
3-D GLASSES

THE WAITE GROUP®

FRACTAL CREATIONS

Second Edition

This second edition contains a new, more powerful version of Fractint V18; and FDESIGN, an incredible IFS Fractal generator which recreates the geometry of nature.



WAITE GROUP PRESS™

Tim Wegner
Bert Tyler

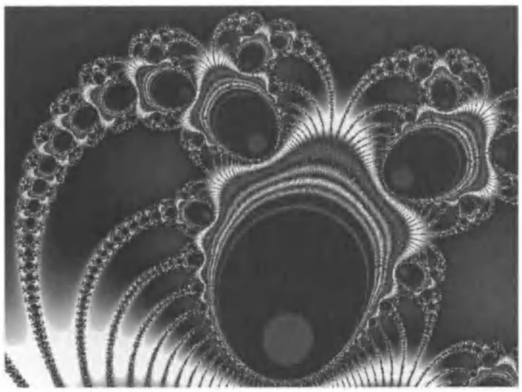
Waite Group Press™
FRACTAL CREATIONS
Second Edition
Contains over 1,900 fractal images, Fractint, and utilities
FRACTELS 2.E.
MASTERED BY NIMBUS
MS-DOS
COMPACT
disc
DATA STORAGE
See book for installation instructions
© 1993 The Waite Group, Inc.®

FRACTAL CREATIONS

SECOND EDITION

WAITE GROUP PRESS™
Corte Madera, CA

Tim Wegner
Bert Tyler



Publisher · *Mitchell Waite*
Editorial Director · *Scott Calamar*
Managing Editor · *Joel Fugazzotto*
Technical Coordinator· *David Middleton*
Content Editor · *Harry Henderson*
Production Director · *Julianne Ososke*
Cover Design · *Ted Mader*
Design · *Cecile Kaufman*
Production · *Cecelia Morales*
Illustration · *Pamela Watenmaker*

Printed in the United States of America
94 95 96 • 10 9 8 7 6 5 4 3

Wegner, Tim (Timothy I.)
Fractal creations / Tim Wegner. --2nd Ed.
p. cm.
Updated ed. of: The Waite Group's fractal creations. 1st. ed. c1991.
Includes index.
ISBN 1-878739-34-4 : $34.95
1. Fractals--Data processing. I. Wegner, Tim (Timothy I.). Waite Group's fractal creations. II. Title.
QA614.86.W44 1993
006.6'869--dc20 93-8903
CIP

DEDICATION

We dedicate this book to everyone who has ever lain belly down on a freshly cut lawn, examined the teeming microcosm of a few square inches of dirt and grass, and wondered about the mystery of the infinitely large and the infinitely small.

ACKNOWLEDGMENTS

The authors would like to acknowledge all those who have contributed to the Fractint program in one way or another. We have been blessed with a steady stream of ideas, bug reports, code contributions, and fractal recipes from the four corners of the globe. As a result of these contributions, Fractint can truly be called a community project, and the "Stone Soup Group" a lively international body of fractal enthusiasts that goes far beyond your authors. You will find their names in the scrolling credits screen and in the Parameter files on the book disk.

The programmers who contributed to Fractint version 18 include Dennis Bragg, Juan Bühler, Chuck Ebbert, Dan Farmer, Dan Goldwater, Wesley Loewer, Jonathan Osuch, Ken Shirriff, and Michael Snyder. Doug Nelson is the programmer of Fdesign.

The fractal artists whose images appear in these pages have made this book a special treat. They are Paul Dickins, Bob Dodson, Dan Farmer, Jon Horner, Ronald Lewen, Peter Moreland, Jonathan Osuch, Caren Park, Bill Potter, Richard Sherry, Lee Skinner, and Michael Snyder.

We'd also like to thank the extraordinarily talented users of Fractint for the over 1800 images they've contributed to the CD; Bob Dodson, Dan Farmer, Jon Horner, Ron Lewen, Peter Moreland, Jonathan Osuch, Caren Park, Bill Potter, Richard H. Sherry and Lee Skinner. The images are used by permission of the artists. They are for your personal use. If you wish to make commercial use of the images, you must contact the individual artists. You can write to them care of the publisher.

Thanks are due once again to Larry Wood, whose Graphics Developer's forum on CompuServe (GO GRAPHDEV) is the on-line home of Fractint.

Finally, we are deeply grateful for the encouragement of Mitchell Waite, who saw us through the creation of this book.

Tim Wegner
Bert Tyler

PREFACE TO THE SECOND EDITION

The first edition of *Fractal Creations* introduced PC owners to the fruits of the collective efforts of a loose-knit international fractal programming team known as Stone Soup. Readers enjoyed dazzling fractal graphics on their super VGA PCs, and spent hours of fascinated pleasure exploring countless fractal landscapes and discovering a visual cornucopia of images. The first edition came with version 15 of Fractint, the program providing all this excitement. The authors and publisher received appreciative feedback from readers about the software and book, which topped several computer book best-seller lists.

The very success of *Fractal Creations* provided an even wider audience for Fractint, which in turn accelerated the contributions of new ideas, code, and examples. Fractint version 16 followed rapidly on the heels of version 15, and introduced the compact PAR format for storing fractal parameters that has become a *defacto* standard for the exchange of images among fractal enthusiasts. After version 17, the handwriting was on the wall: a new edition of *Fractal Creations* was needed. This book introduces Fractint version 18.2, the third significant upgrade of Fractint in the less than two years.

FRACTINT VERSION 18.2

Fractal Creations, Second Edition provides several significant improvements over the first edition. Fractint version 18.2 has many major new features; there are more than 25 new fractal types. Fractint has always had a reputation for speed, but clever programmers contributed new code that significantly speeded up many functions. Fractint can now create images of virtually unlimited resolution; the 2048 x 2048 limit is now gone. New interactive exploration tools have been added—you can now see the Julia set metamorphose before your eyes as you move a cursor around the Mandelbrot set. Fractint also has a better facility for displaying the orbits underlying the generation of fractals, which have an eerie beauty in their own right.

Fractint version 18.2 has a greatly expanded capability to generate and display 4-D fractals in both two and three dimensions. A new kind of fractal using hypercomplex numbers is introduced for the first time in this book.

All the popular features of the earlier edition are still available; you can make fractals dance with color cycling, turn clouds into mountains or planets with the

3-D facility, view fractals in full stereo with the included red/blue glasses, zoom into any fractal to impossible depths, and design your very own fractal formulas.

OVER 1900 SPECTACULAR FRACTAL GIF IMAGES ON CD ROM

This book contains a CD ROM with over 1900 high resolution 1024 x 768 GIF images of stunning artistry and quality. These images required 6886 hours to generate on fast 33 and 66 mHz 486 PCs—that's over 280 *days* of continuous computer time—yours for the price of this book! Since the first edition, a number of talented fractal artists have emerged. Every single fractal type now has a new example contributed by a Fractint user. A new chapter *Fractal Recipes,* explores some of these glorious images in detail, so you can gain insight into how their artistry was achieved.

C AND ASM SOURCE CODE PROVIDED

You don't have to be a programmer to enjoy Fractint; but if you are a programmer, you will enjoy the source code. You can learn how to enhance the program and add your own features. After all, that is what the Stone Soup origins of Fractint are all about! The complete source code for Fractint version 18.2 is provided on the companion CD. A chapter describing the source code and demonstrating how to add a fractal type has been added. The Fractal Types chapter now documents the locations of the source modules for every fractal type.

NEW INTERACTIVE IFS DESIGN PROGRAM

The earlier Fractint version 15 could display Iterated Function Systems fractals, but it did not provide an easy way to design and edit them. *Fractal Creations, Second Edition* includes Fdesign, an interactive tool for designing IFS fractals. You can use a mouse to visually modify the IFS transformations and instantly see the results on the screen. Fdesign is another Stone Soup program, and can share files with Fractint.

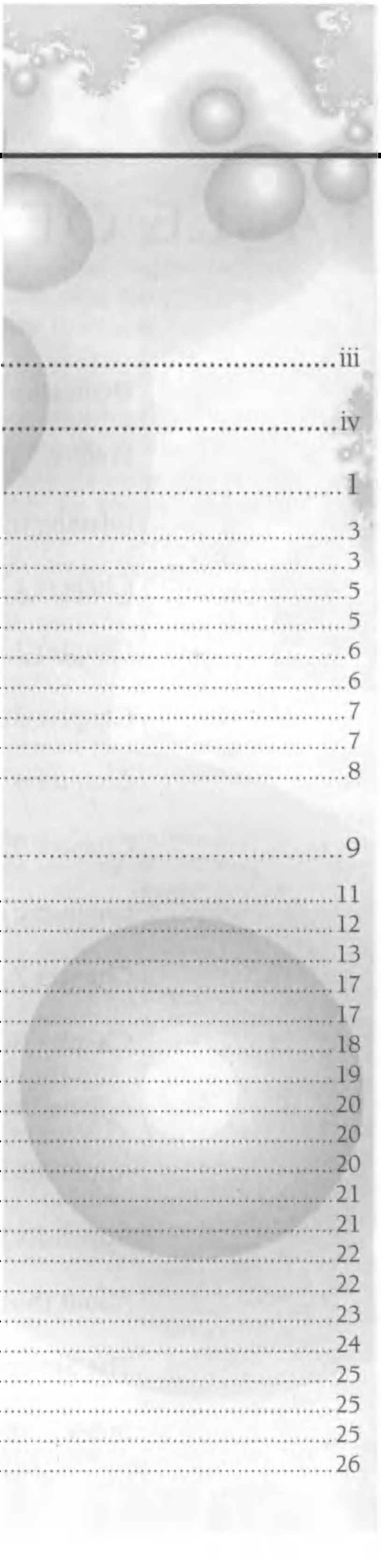
EXTRA FOR UNIX WORKSTATION FOLKS: XFRACTINT

Want to create fractals on a fast RISC workstation? Xfractint, the experimental port of Fractint to UNIX/X, is included on your companion CD. You can compile the Xfract source code yourself, or run the ready-to-use executables for Sun, Sparc, DEC Alpha, or MIPS workstations.

We hope that you gain as much enjoyment reading this book and using this software as the programmers have had developing the software!

TABLE OF CONTENTS

CONTENTS



INTRODUCTION

This book is about creating fractals—dazzling and colorful images of infinite detail—on your PC. This hands-on book comes bundled with Fractint, the preeminent fractal-generating program. All the images on the book jacket and in the color plate section of this book were created with this powerful program, which is the result of collaborative effort by an international team of volunteer fractal enthusiasts. With this software and a PC, you can quickly and easily begin creating your own fractals from any of the built-in fractal types. As you become more proficient, you will discover an inexhaustible number of options for coloring and transforming your images to suit your imagination. A beginner can create images of striking complexity and beauty. But even an expert will find more than enough controls and tools to challenge his or her adventurous creativity.

ORGANIZATION OF THE BOOK

Fractal Creations, Second Edition consists of the preface, this introduction, eight chapters, three appendices, a color plate section, and companion disk and CD containing the latest version 18.2 of the Fractint program and source code, the Fdesign program, and example files.

Chapter 1: Installation

The first chapter tells you how to get Fractint up and running on your PC. If you are an experienced computer user eager to begin creating fractals, you may find that the Quick Start in this section is all you need to begin using the program. A more detailed guided tour is provided in Chapter 3, *Fractint Tutorial*.

Chapter 2: Fractals: A Primer

The second chapter describes fractals, the different varieties of fractals, how they are generated, and their significance. The background provided here will enhance your enjoyment of creating fractals by providing insight into what fractals are and how they are generated.

Chapter 3: Fractint Tutorial

The Fractint tutorial is an extensive tour of the main features of Fractint. You will go on a step-by-step tour through Fractint's basic functions right up to some of the more advanced functions. By the end of the tour you will know how to make many of those spectacular color plate images.

Chapter 4: Fractal Recipes

In this chapter you can sample a gourmet feast of the very best fractal recipes created by devoted Fractint users. You can learn by example the hints and tricks that the experts use to make dazzling fractal images.

Chapter 5: Fractint Reference

Fractint is a multifeatured software program. This chapter tells you how to access all the basic functions and unlock the secrets of Fractint's advanced fractal-generation options. This chapter has been updated to reflect the many new features of Fractint version 18.2.

Chapter 6: Fractal Types

Fractint can generate the most extensive variety of fractals of any fractal program. At last count, the main fractal type screen, which lists the different kinds of fractals generated by Fractint, had 95 entries. As you will discover, the actual number of possible kinds of fractals you can create with Fractint is much larger than that. This chapter tells you about all of those different kinds of fractals, and is filled with dozens of all-new examples you can try. For each of Fractint's 95 fractal types, you will find an example, the mathematical algorithm used to generate that type, and a reference to the source code routines that implement the algorithm.

Chapter 7: Making IFS Fractals with Fdesign

This chapter is about Fdesign, a companion program for Fractint (included on your book disk) that lets you visually create and modify Iterated Functions Systems (IFS) fractals. You can rapidly create bushes, trees, ferns, Sierpinski gaskets, wheat kernels, telephone cords, and quilt patterns. The results of your creativity can be imported into Fractint and displayed with Fractint's powerful video support.

Chapter 8: Fractint's Source Code

If you are curious about the inner workings of Fractint, you will find what you are looking for in Chapter 8. The programmers tell you how to compile the program, and walk you through key sections of code with explanations of how the fractal magic is accomplished.

Appendices

There are three appendices, Appendix A, *Fractint and Video Adapters*, Appendix B, *Fractints and GIF Files,* and Appendix C, *Complex and Hypercomplex Numbers.* These appendices explain how Fractint's video drivers work, give background on the main file format that Fractint uses to store graphics, and introduce you to the mathematics of multidimensional spaces that Fractint uses to create Fractals.

HOW TO USE THIS BOOK

Learning About Fractals

You will find a discussion of what fractals are, how they are generated, and the ideas behind them in Chapter 2, *Fractals: A Primer.* This is the one chapter in the book that does not require a computer—the only prerequisite is a lively curiosity.

For Those New to Fractint

The first three chapters provide a logical sequence for those encountering Fractint for the first time. These chapters take you from Fractint installation and background about fractals to a guided tour of Fractint's many capabilities.

Users of Older Versions of Fractint

If you are already familiar with earlier versions of the Fractint program, you can start with the tutorial in Chapter 3, *Fractint Tutorial,* to brush up on Fractint's operation as well as learn about some of the newer features. Then try out the *Fractal Recipes* in Chapter 4. Consult Chapter 5, *Fractint Reference,* for a comprehensive review of all the features of Fractint 18.2.

Reference Information

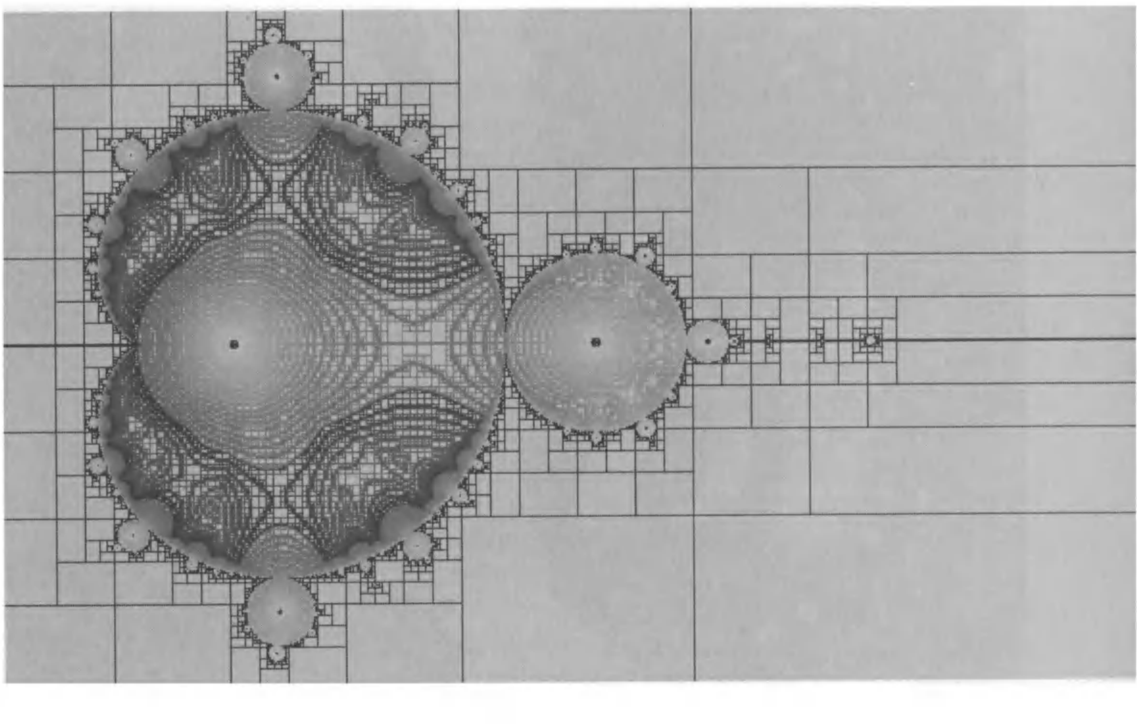
This book contains two chapters of useful reference information. Chapter 5, *Fractint Reference,* documents the Fractint's functions, commands, and menus. Chapter 6, *Fractal Types,* provides a comprehensive description of Fractint's fractal types.

For Programmers

You don't have to be a programmer to enjoy this book; but if you are, you will find a wealth of useful information about programming fractals. The complete source for Fractint is included on the distribution CD. The *Fractal Types* chapter (Chapter 6) tells you where to find the routines used to generate each kind of fractal. Chapter 8, *Fractint's Source Code,* provides an overview of the code, tells you how to compile Fractint, and describes the step-by-step process of adding a new fractal to Fractint.

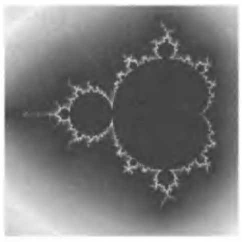
CHAPTER 1

INSTALLATION



CHAPTER 1

INSTALLATION



In this chapter we'll describe how to install the software that is bundled with this book. We've also included quick start instructions for those who just can't wait to start generating fractals. Chapter 2, *Fractals: A Primer,* provides a more thorough guided tour of Fractint.

HARDWARE AND SOFTWARE REQUIREMENTS

Fractint will run on any IBM-compatible PC with at least 512K of free memory. A hard disk is highly recommended, but not required. Although Fractint can run on text-only systems using its Disk/RAM video modes, displaying the fractals generated by Fractint requires some kind of graphics video support. Fractint supports the IBM CGA, EGA, VGA, MCGA, 8514/A, and XGA standards and the VESA VBE standard. It also works with most other super VGA boards, Targa boards, and Hercules-compatible monochrome graphics.

We have included a companion disk and a CD with this book. This disk contains the files you'll need to install Fractint and the Fdesign program.

The disk distributed with this book is a 3-1/2" diskette, so you'll need a high-density 3-1/2" disk drive to access it. If the machine on which you want to install Fractint uses high-density 5-1/4" disk drives and you can locate another machine with both types of disk drives, you can copy the files on your companion disk to a high-density 5-1/4" disk and perform the installation process from that disk. The disk is not copy protected in any way, and the files will fit onto a high-density 5-1/4" disk.

The CD contains the Fractint source code and hundreds of exciting fractal images. To use this disk you will need a CD ROM drive.

INSTALLING FRACTINT

Before installing any software package on your computer, it is a good idea to make a backup copy of the floppy installation disk and store the original away somewhere safely.

After you've made that backup copy, next insert the copy of the companion disk or your CD into the computer and view the README file that is on it. The mechanics of the book industry are such that the text of a book is finalized before the companion disks, and it is always possible that we changed something in either the Fractint program or its installation process after this text was written. If that's the case, you'll find everything you need to know about any such changes in that README file. With that disk in your floppy drive, view the README.COM file and see what it contains (we'll use the A: drive as an example) by typing

```
C:\>    A:README.
```

To view the README file on the CD, change to the CD drive and type:

```
D:\> TYPE READ.ME|MORE (ENTER)
```

The file containing the Fractint program itself is FINSTALL.EXE. It is a *self-extracting archive* file, which contains the "real" programs and their support files stored in compressed form inside it.

To install Fractint and Fdesign on a hard disk, first create a new directory for your Fractint files. Assuming you want to place the software in the \fractint directory on drive C: and the A: drive is the source, enter the following commands at the C:\> prompt to create a new directory and make it your current one. Then, with your companion disk or your CD in the computer, run FINSTALL to install Fractint and its related files on that new directory:

```
C:\> md \fractint (ENTER)
C:\> cd \fractint (ENTER)
C:\FRACTINT>  a:\finstall  (ENTER)
```

FINSTALL will display a startup message something like the one printed here and it will ask you if you want to proceed. Press (Y) and then (ENTER) to do so. FINSTALL will list the name of each file as it puts it on your hard disk.

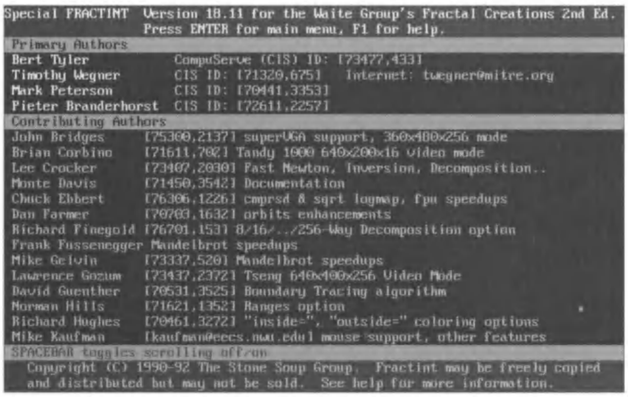
```
LHa's  SFX  2.10S  (c)1991,  Yoshi

This  program  will  put  FRACTINT.EXE  and  its  related  files  in  your  current  directory.

Do  you  wish  to  proceed  with  this  program?

[Y/N]   Y

FRACTINT.EXE
SIMPLGIF.EXE
FRACTINT.FRM
ALTERN.MAP   .
(etc)
C:\>
```



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.
                  Press ENTER for main menu, F1 for help.
 Primary Authors
 Bert Tyler             CompuServe (CIS) ID: [73477,433]
 Timothy Wegner         CIS ID: [71320,675]   Internet: twegner@mitre.org
 Mark Peterson          CIS ID: [70441,3353]
 Pieter Branderhorst    CIS ID: [72611,2257]
 Contributing Authors
 John Bridges       [75300,2137] superVGA support, 360x480x256 mode
 Brian Corbino      [71611,702] Tandy 1000 640x200x16 video mode
 Lee Crocker        [73407,2030] Fast Newton, Inversion, Decomposition..
 Monte Davis        [71450,3542] Documentation
 Chuck Ebbert       [76306,1226] cmprsd & sqrt logmap, fpu speedups
 Dan Farmer         [70703,1632] orbits enhancements
 Richard Finegold   [76701,153] 8/16/../256-Way Decomposition option
 Frank Fussenegger Mandelbrot speedups
 Mike Gelvin        [73337,520] Mandelbrot speedups
 Lawrence Gozum     [73437,2372] Tseng 640x400x256 Video Mode
 David Guenther     [70531,3525] Boundary Tracing algorithm
 Norman Hills       [71621,1352] Ranges option
 Richard Hughes     [70461,3272] "inside=", "outside=" coloring options
 Mike Kaufman       [kaufman@eecs.nwu.edu] mouse support, other features
 SPACEBAR toggles scrolling off/on
   Copyright (C) 1990-92 The Stone Soup Group.  Fractint may be freely copied
   and distributed but may not be sold.  See help for more information.
```

Figure 1-1 Fractint's initial scrolling credits screen

Installation from the CD-ROM will write 176 files to your hard disk. Installing from the floppy will write 177 files to your hard disk.

MODIFYING YOUR DOS PATH

If you'd like to be able to run Fractint from any directory, you'll have to change your DOS PATH to include the Fractint directory (otherwise, you'll still be able to run Fractint, but you'll have to be in the Fractint directory to do so). You probably set your DOS PATH with a PATH statement inside your AUTOEXEC.BAT file—if so, just add the Fractint directory to the end of that statement. Note that changing your PATH this way won't actually affect your DOS PATH until the next time you reboot your computer. Your PATH statement will end up looking something like the one here.

```
path c:\dos;c:\work;c:\games;...;c:\fractint
```

QUICK START

To start Fractint, simply enter its name at the DOS command prompt:

```
C:\ > fractint ENTER
```

Fractint takes a few seconds to start up, and then it presents you with its initial *scrolling credits* screen (see Figure 1-1). This screen lists the names of all the folks—and there are quite a few of them—who have contributed to the program and helped make it what it is today.

SELECTING A VIDEO MODE

Pressing ENTER with the scrolling credits screen active brings up Fractint's MAIN MENU screen, with its *highlight bar* on the SELECT VIDEO MODE menu item. Press ENTER again to select that highlighted menu item, and Fractint will bring up its SELECT VIDEO MODE screen. The highlight bar will be on a video mode appropriate for your hardware. You can scroll the highlight bar up and down the list using your cursor keys to choose an alternate video mode, if you want. If you have a standard adapter (CGA, EGA, MCGA, VGA, or Hercules), Fractint should have detected your video equipment and highlighted a reasonable starting choice. The choices are

F2—for EGA	(16 colors)
F3—for VGA, MCGA, and SVGA	(256 colors)
F5—for CGA	(4 colors)
F6—for monochrome CGA, EGA, VGA	(2 colors)
CONTROL-F6—Hercules Monochrome Graphics	(2 colors)

Press ENTER to select the video mode currently being highlighted. (You can also select video modes directly without going through Fractint's menu interface by pressing the indicated function keys, such as F2 for 16-color EGA mode.)

Fractint will immediately begin drawing its initial image—the full Mandelbrot set. For low resolution modes, this usually takes two passes. The first pass uses large rectangles of color. The second pass adds more detail by breaking up the large rectangles into smaller ones. For the full Mandelbrot set, this process is quite fast—less than a second for 320 x 200 x 256 mode on a high-speed 486 machine.

ZOOMING IN ON AN IMAGE

Let's bring up a *zoom box*, a device Fractint uses to control its zooming process. Press PAGE UP several times. A rectangular outline will appear on the screen and grow progressively smaller for each keypress until it reaches a minimum size. Use the arrow keys to move this zoom box to someplace interesting (the areas on the edge of the blue interior "lake" where there are lots of colors are the best) and press ENTER. Fractint will clear and redraw the screen using the area of the initial image that was within the zoom box.

If you have a mouse, you can also use it to control the zoom box. Holding down the left mouse button, move the mouse "up" and away from you to bring up and shrink the zoom box (go ahead—you don't have to wait for Fractint to

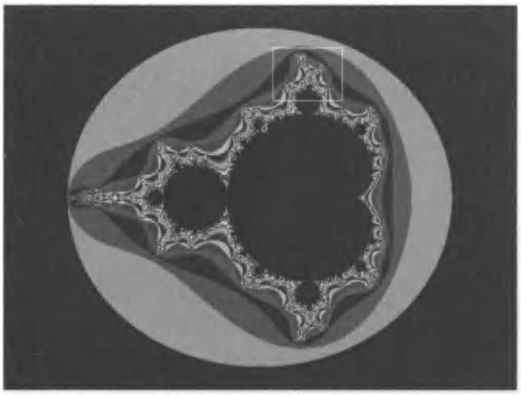


Figure 1-2a Using the zoom box

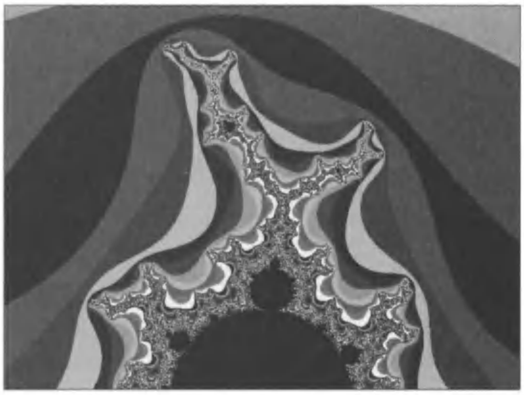


Figure 1-2b The zoomed-in results

finish generating an image before you select a new one). Moving the mouse toward you while holding down the left mouse button expands the zoom box. To move the zoom box, just move the mouse without holding down any mouse buttons. To perform the zoom, double-click the left mouse button.

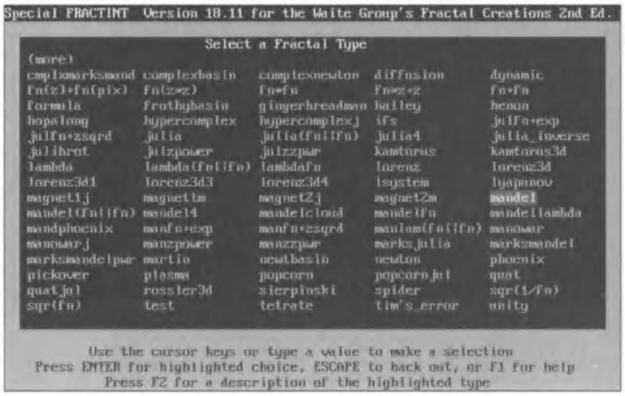
Figures 1-2a and 1-2b show a Mandelbrot image with a zoom box on an interesting area and the zoomed-in image that results when you press (ENTER).

SELECTING A NEW FRACTAL TYPE

Now let's select a new fractal type. You can do this either directly by pressing (T), or indirectly by pressing the (ESC) key to bring back Fractint's MAIN MENU, using the arrow keys to highlight the SELECT FRACTAL TYPE menu item, and pressing (ENTER). Fractint will bring up a rather formidable list of fractal types (see Figure 1-3), with the current one (MANDEL) highlighted. Select the *lambda* fractal type (either by using your cursor keys to move the highlight bar or by typing its name in directly) and press (ENTER). Press (ENTER) again at the parameter entry screen to accept the default values, and Fractint will begin generating the Lambda fractal. You can zoom in on interesting places in that fractal type just as you did with the Mandelbrot fractal.

COLOR CYCLING

If you have a VGA video adapter, press the (+) key to enter Fractint's *color-cycling* mode. While the colors are changing on your screen, press (ENTER) to randomly select new sets of colors and the frequency with which they mutate into new



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

                          Select a Fractal Type
 (more)
 cmplxmarksmand complexbasin    complexnewton  diffusion       dynamic
 fn(z)+fn(pix)  fn(z*z)         fn*fn          fn*z+z          fn+fn
 formula        frothybasin     gingerbreadman halley          henon
 hopalong       hypercomplex    hypercomplexj  ifs             julfn+exp
 julfn+zsqrd    julia           julia(fn||fn)  julia4          julia_inverse
 julibrot       julzpower       julzzpwr       kamtorus        kamtorus3d
 lambda         lambda(fn||fn)  lambdafn       lorenz          lorenz3d
 lorenz3d1      lorenz3d3       lorenz3d4      lsystem         lyapunov
 magnet1j       magnet1m        magnet2j       magnet2m        mandel
 mandel(fn||fn) mandel4         mandelcloud    mandelfn        mandellambda
 mandphoenix    manfn+exp       manfn+zsqrd    manlam(fn||fn)  manowar
 manowarj       manzpower       manzzpwr       marksjulia      marksmandel
 marksmandelpwr martin          newtbasin      newton          phoenix
 pickover       plasma          popcorn        popcornjul      quat
 quatjul        rossler3d       sierpinski     spider          sqr(1/fn)
 sqr(fn)        test            tetrate        tim's_error     unity

          Use the cursor keys or type a value to make a selection
     Press ENTER for highlighted choice, ESCAPE to back out, or F1 for help
              Press F2 for a description of the highlighted type
```

Figure 1-3 Fractint's fractal types menu

colors. The ← and → keys change the "direction" of the color cycling, and the ↑ and ↓ keys change the speed at which they change. (You can see why Fractint has been called the 90s version of the Lava Lamp.) You exit from Fractint's color-cycling mode by pressing ESC.

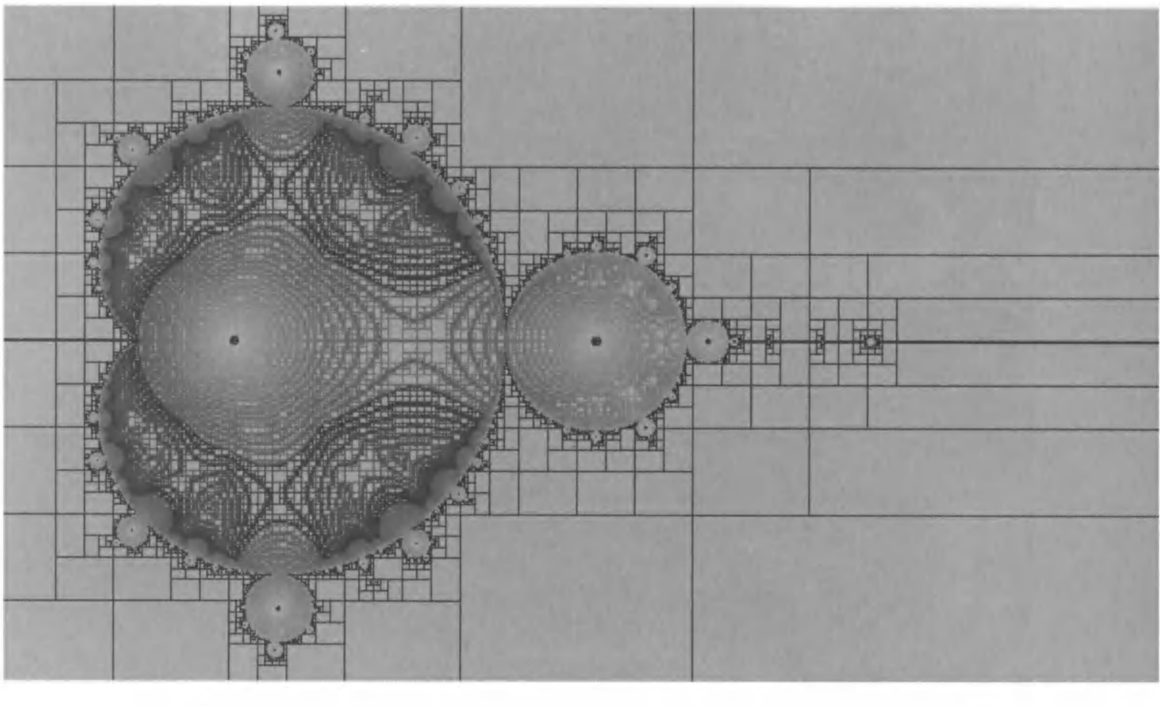
GETTING OUT OF FRACTINT

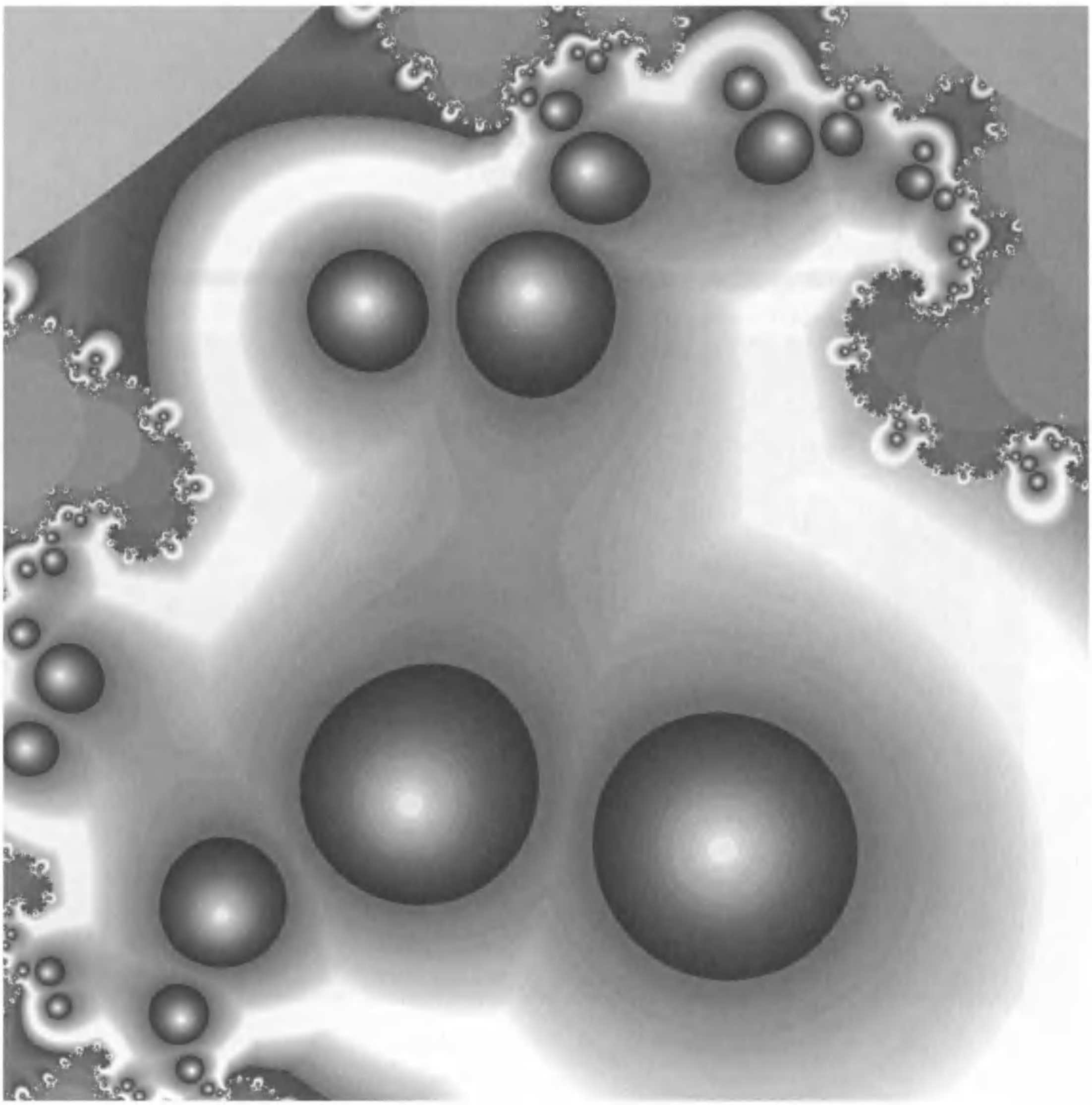
Pressing ESC a few times will back you out to the MAIN MENU and finally to a prompt asking if you want to exit Fractint. Pressing Y brings you back to the DOS prompt.

Now that you've gotten Fractint installed and running, you're ready to turn to Chapter 2, *Fractals: A Primer,* to learn what fractals are and to Chapter 3, *Fractint Tutorial,* to learn more about Fractint.

CHAPTER 2

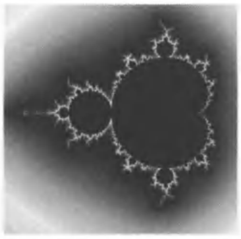
FRACTALS: A PRIMER





CHAPTER 2

FRACTALS: A PRIMER



Impossible patterns with dazzling color and mind-stretching detail—you've seen them on the covers of magazines, in calendars, on book jackets, and on personal computer screens. They are *fractals*, a product of the marriage between contemporary mathematics and the high-tech computer revolution, but nevertheless a phenomenon as close to home as the flowers in your garden or the pores on the back of your hand. You may have the impression that understanding or exploring fractals requires a mind-numbing amount of higher math. Fortunately, that's not true at all.

This chapter will teach you what fractals are and where they come from. You will see what properties fractals share in common, and explore the inner workings of the chaos that creates distinctive "families" of fractals. You will learn about interesting applications of fractals in a variety of different fields. Finally, you will learn how fractal images are created using today's personal computers. Having read this chapter you'll be in an excellent position to appreciate the power of the Fractint program that comes with this book. Fractint is described in the next chapter.

WHAT ARE FRACTALS?

Fractals are beautiful, fascinating designs of infinite structure and complexity—the sort of intricate patterns that capture attention and evoke a sense of childlike wonder. A fractal is a mathematical object that has detailed structure no matter how closely you look at it, no matter how great the magnification. Look at Figure 2-1, which is a famous fractal called a Julia set. This fractal was generated

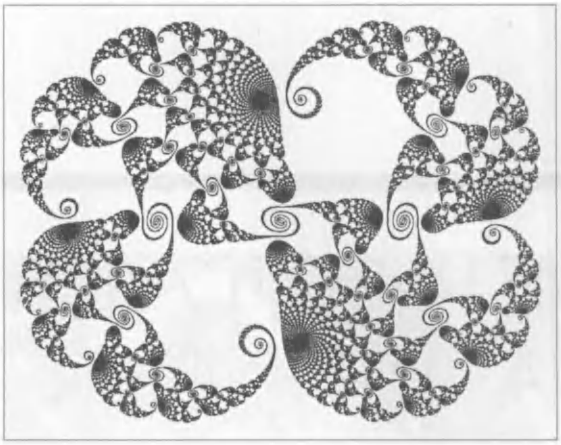


Figure 2-1 A computer-generated fractal

on a computer with the software enclosed with this book and then printed. If you hold the page at arm's length, you see spirals within spirals in repeating patterns, sequences of ever-shrinking structures vanishing into nothing. If you hold the page up close, your eyes will discover more detail right down to the limit of what the printer could record. What you see here is an infinite pattern somehow compressed into a finite space.

So what are fractals anyway? As you make your way through this book, we will present ample evidence of the diversity of the universe of fractals and the multiplicity of ways of answering that simple question.

THE TRUTH ABOUT FRACTALS

We could go on and on about beauty and complexity, but let's begin this discussion with a healthy dose of reality. Far from being esoteric abstractions, fractals are much closer to ~~home~~ than you realize. In fact, it is the *nonfractal* objects that are un~~real~~, abstract, and removed from our experience. Let's see why that is true.

From the beginnings of our education, formal and informal, we have been given simplified categories for organizing the world. The world is a sphere. Throw a baseball in the air, and its trajectory is a parabola. Nations are divided into the First World, the Second World, and the Third World. All of these statements have a strong element of truth, but none of them turns out to be accurate when you look closely. We have known since the Apollo days that the earth is really pear-shaped. After allowing for air resistance, the pear-shape of the earth, and even the gravitational field of the moon, the path of a baseball is *not exactly* a parabola. As is increasingly evident today, the elements of the First, Second, and Third Worlds are intertwined in a complex way in the economies and societies of every country.

This may sound like splitting hairs, but our everyday lives are full of clothes that don't fit exactly, lawns that are not all grass, and new cars with dents in their fenders. Yet we cannot do without our approximations and generalizations; we wouldn't make it through the day without simplifying assumptions. We say, "I'll meet you around 3:00," "enough to feed thirteen," or "about five people per car," instead of "meet me at 3:12:26," "enough food to feed five adults, two children, and six elderlies," or "exactly 4.67359 people per car." There is too much detail in the world to fully grasp. Indeed, there is too much detail in a single leaf for the mind to absorb.

It is irritating in the extreme to have one's simplified picture of the world shown to be inaccurate, but it happens to us all the time. Galileo faced the Inquisition for maintaining that the Earth was not the center of the universe, and Einstein (as an employee of the Trademark office) puzzled us with the idea that matter and energy are the same thing. The history of the investigation of fractals contains many stories of discoveries made by outsiders who collected the forgotten crumbs of different disciplines and prepared a feast of chaotic structures and theories. Many scientists are finding that "curious counter-examples" turn out to be the basis of a whole new field of inquiry, and worse yet, a field developed by others! But we are getting ahead of ourselves.

Fractals are about looking closely and ~~seeing~~ more. Fractals have to do with bumps that have bumps, cracks that have crookednesses within crookednesses, and atoms that turn out to be universes. Fractals have to do with the rich structure of our universe that spans all scales from the uncountable galaxies at unthinkable distances to the mysterious inner electric flashes and vibrations of the subatomic realm. Let's see how looking closer, results in fractals.

HOW LONG IS THE COASTLINE OF BRITAIN?

Benoit Mandelbrot, of IBM's Thomas J. Watson Research Center, did ground-breaking work in the theory of fractals and indeed, he coined the very word "fractal." Dr. Mandelbrot poses a simple question to introduce the notion of a fractal in his book *The Fractal Geometry of Nature* (Wilt Freeman and Company, 1977, 1982, 1983, ISBN 0-7167-1186-9): How long is the coastline of Britain?

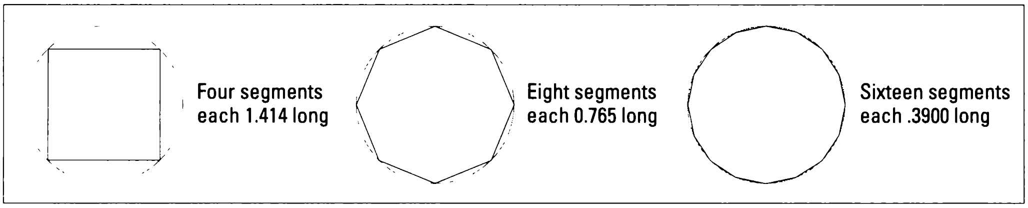

Four segments each 1.414 long
Eight segments each 0.765 long
Sixteen segments each .3900 long

Figure 2-2 Approximating a circle with polygons

This deceptively simple question turns out to expose a deep problem and give us insight into the question "What is a fractal?"

Consider how to approximate the length of the "coastline" of a circle of radius 1. Of course you probably remember the answer in advance from high school geometry: using the formula for the circumference of a circle; it is $2 \times \pi \times 1$ where $\pi = 3.14159...$, or approximately 6.28. As a way of arriving at a similar result, you could inscribe a square inside the circle, and estimate that the circumference of the circle is the sum of the sides of the square, as shown in Figure 2-2. Notice that if the results are not accurate enough, all you have to do is make a polygon with more sides. Table 2-1 shows how the circumference of an inscribed polygon gets closer and closer to a limiting value, which is the "real" circumference.

This procedure is both mathematically correct and intuitively clear, and it works in much more general settings than this example. Estimating distances of curves by approximating them with a series of straight segments is a tried and true procedure that surveyors use when mapping terrain. Think of the side of the polygon (or the length of a sighting with a surveyor's scope) as a giant measuring stick. If the curve being measured is "well behaved"—which is to say, continuous and smooth—the answer can be made as accurate as desired by making the

Sides	Length of One Side	Circumference
3	1.732	5.20
4	1.414	5.66
8	0.765	6.12
16	0.390	6.24
32	0.196	6.27
64	0.098	6.28

Table 2-1 The circumference of polygons inscribed in a unit circle

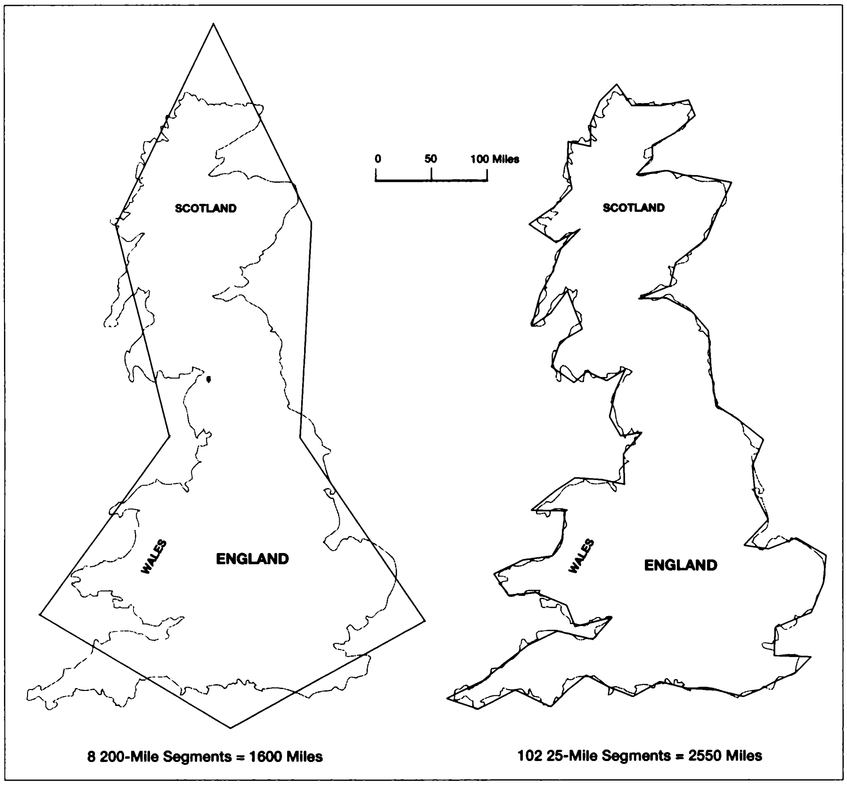

0 50 100 Miles
SCOTLAND
ENGLAND
WALES
SCOTLAND
ENGLAND
WALES
8 200-Mile Segments = 1600 Miles
102 25-Mile Segments = 2550 Miles

Figure 2-3 Approximating the length of the coastline of Britain

approximating measuring sticks smaller and smaller. Presumably this same logic can be used to find the length of the coastline of Britain. *Or can it?*

Let's try the same trick on a map of Britain, using measuring sticks 200 and 25 miles long. Figure 2-3 shows the measuring stick approximations overlaid on a map of Britain, and Table 2-2 shows the numerical results.

What is strange is that as the measuring stick gets smaller, the coastline estimation seems to grow larger—much larger than we would expect from the way the circumference approximation went! What is happening?

The difficulty is not too hard to see. The coastline of Britain is very, very irregular, full of large and small bays, inlets, tiny rivers, and complex, rocky shores. A long measuring stick does not bend with these many twists and turns, but cuts directly over them. A shorter measuring stick fits snugly inside these nooks and bays, thereby increasing the length estimate. Imagine doing this exercise crawling on your hands and knees, measuring the coastline of Britain

Length of Measuring Stick	Coastline
200 miles	1,600 miles
25 miles	2,550 miles

Table 2-2 Estimation of the length of the coastline of Britain

with a measuring stick an inch long. Every small rock that you traversed around would increase your coastline estimate. Your answer for estimating the coastline would be astronomical!

There is a fundamental difference between a curve like a circle and a curve like the coastline of Britain. This difference separates the shapes of classical geometry from the shapes of fractal geometry. So here's your first definition: the coastline of Britain is a fractal, and our difficulty in measuring its length suggests a definition of a fractal. For present purposes, we will use informal intuitive definitions, because the formal definitions are beyond the scope of this book.

DEFINITION: If the estimated length of a curve becomes arbitrarily large as the measuring stick becomes smaller and smaller, then the curve is called a *fractal* curve.

While you might not be impressed by this observation of increasing distances measured as we go from circles to coastlines, what is magic is that the idea behind the fractal definition can be generalized to cover many other kinds of shapes besides curves. In all cases the basic idea is the same—the difficulty of measuring is due to the irregularity of the object being measured, and it is an irregularity that continues to the most microscopic level. This difficulty of measuring is related to the idea of dimension. Lines and curves are one-dimensional, planes and surfaces are two-dimensional. It turns out that the idea of "dimension" can be broadened in such a way that these unusual curves have a dimension greater than 1. This leads us directly to an alternative way to define a fractal.

DEFINITION: The *fractal dimension* of an object is a measure of its degree of irregularity considered at all scales, and it can be a fractional amount greater than the classical geometrical dimension of the object. The fractal dimension is related to how fast the estimated measurement of the object increases as the measurement device becomes smaller. A higher fractal dimension means the fractal is more irregular, and the estimated measurement increases more rapidly. For objects of classical geometry (lines, curves), the dimension of the object and its fractal dimension are the same. A *fractal* is an object that has a fractal dimension that is strictly greater than its classical dimension.

Because the British coastline is, after all, a curved line, which is a one-dimensional geometric object, the fractal dimension of the coastline must be a little greater than 1. According to Mandelbrot, the mathematician Lewis Fry Richardson estimated it to be approximately 1.2. Indeed, mathematical "one-dimensional" curves can be defined which are so irregular that their fractal dimension approaches 2.0. One such "impossible" curve is the boundary of the Mandelbrot set, which was proven to have a fractal dimension of exactly 2.0 by Japanese mathematician Shishikura in 1991. (We'll introduce you to the Mandelbrot set a bit later in this chapter.) In the discussion that follows, we will use the term "fractal geometry" to refer loosely to the theory of these bumpy shapes, just as classical geometry is the theory about regular "well-behaved" shapes.

EXAMPLES OF FRACTALS OCCURRING IN NATURE

Now that we know the coastline of Britain is a fractal, where else are these fractals lurking? If you have begun to catch the gist of where this discussion is heading, you have probably already guessed the answer: nearly everywhere!

Mountains as Fractals

Have you ever noticed how difficult it is to estimate the distance to a far-off mountain? Nearby foothills and distant mountains have a very similar appearance. A mountain is, therefore, a fractal; its roughness is the same at different scales. Indeed, the fractal characteristic of hills and mountains quickly becomes a practical matter for a hiker; a mild two-hour dash to the top can turn out to be a full day of traversing up and down through ravines and canyons that were invisible from a distance. The fun of scrambling up rocky hillsides is in part due to the fact that the fractal dimension of a mountain applies at all scales, including the scale of a human being. Figure 2-4 shows a range of snow-capped mountains as seen from the Space Shuttle. The snow line traces the fractured boundaries of ravines, forming a fractal dimension and a pattern amazingly similar to some computer-generated fractals we will be discussing a bit later in the book.

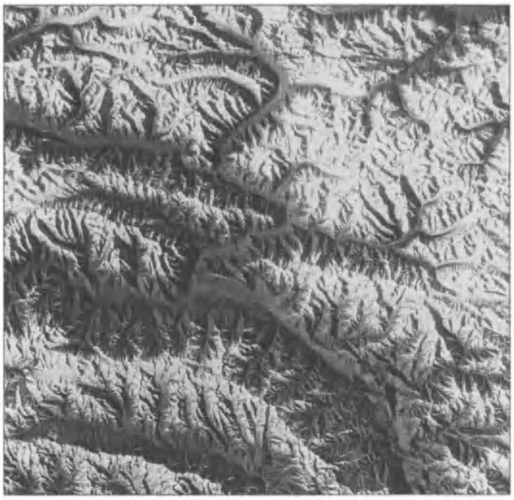


Figure 2-4 Snow-capped mountains from space are fractals

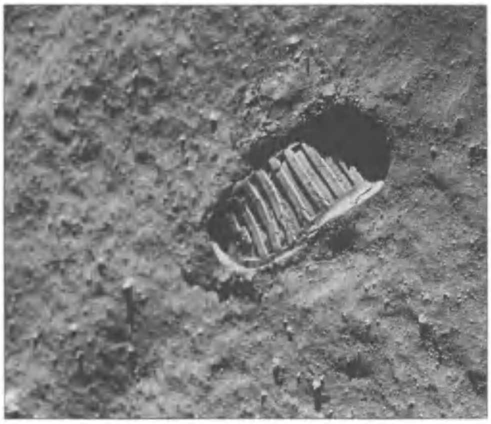


Figure 2-5 Footprint on the moon

A good example of a fractal is found in the famous picture of a footprint on the moon (see Figure 2-5). Near the footprint is the gravelly crust of the moon's surface. Consider now the "earthrise" view of the Earth and moon (see Figure 2-6). This picture is most famous for its beautiful view of the Earth, but look at the lunar landscape and compare it with the lunar surface in the footprint picture. Take the footprint out of the picture, and the surface of the moon seen from two feet away looks somewhat like the moonscape viewed from two hundred miles away. When a tiny piece of a fractal is similar to the whole, we say that the fractal is *self-similar*. Understand that a self-similar object is generally a fractal, but not all fractals are self-similar. A fractal is defined by the irregularity that must exist at all scales, but this irregularity need not look the same. Both views of the moon's surface show fractal irregularities, but the fractal dimension appears to be higher in the footprint picture than in the more distant moon surface.

Clouds

Clouds are wonderful examples of fractals. Sophisticated travelers are supposed to prefer aisle seats on airplanes, but real fractal lovers choose window seats so they can watch clouds. You may wonder how something as soft and fluffy as a cloud can be a fractal, which we have defined in terms of jagged but measurable bumps and rough irregularities. Clouds are indeed roughly irregular and jagged; it's just that

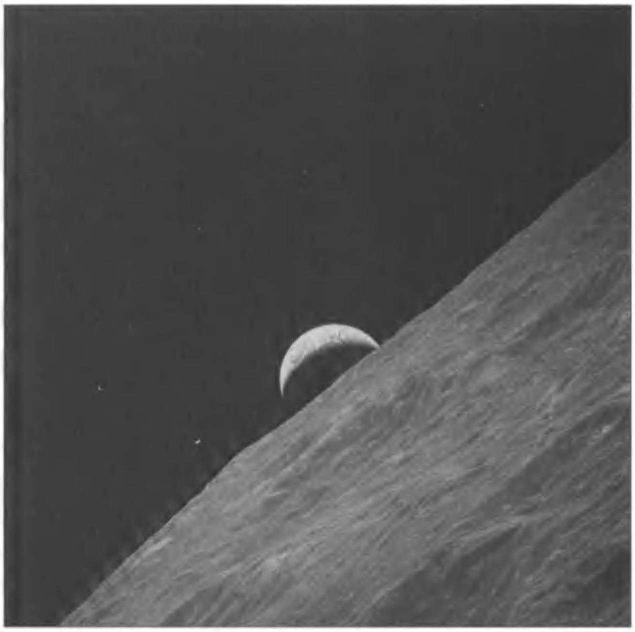


Figure 2-6 Earthrise

the colors reflected by the cloud blend smoothly into one another, giving the impression of smoothness. A little later in the book, we will try to convince you that clouds and mountains from a fractal perspective are virtually the same thing.

Waves as Fractals

Not too long ago, before the study of turbulence (the complex movements of air or fluids) had advanced, it was believed that ripples on the surface of a lake were uniformly distributed. You can verify for yourself that this is not true, and that the pattern of ripples is very nonuniform, by simply taking a closer look at a body of water on a windy day. Every lake surface has smooth patches. On a windy day they might be small, and on a calmer day larger, but they are always there. But if you look closely at the rough areas of the surface—the areas full of wavelets—you will see that the "rough" areas are not completely rough, but themselves contain little glassy smooth areas. The surface of a lake is complex in the extreme, consisting of a nested pattern of smooth and rough areas that continues as you look closer and closer. This kind of nested mixture of the smooth and rough is a trademark of fractals. We can say that the lake's surface has a fractal dimension.

The Human Circulatory System

Blood flows from the heart in arteries and back to the heart in veins, but what happens in between? The arteries and veins are connected by a network of smaller and smaller vessels successively branching and rebranching until they finally meet in microscopic capillaries. A wonderful article in the February 1990 *Scientific American* entitled "Chaos and Fractals in Human Physiology" describes and vividly pictures this phenomenon. Branching patterns are a characteristic quality of certain classes of fractals.

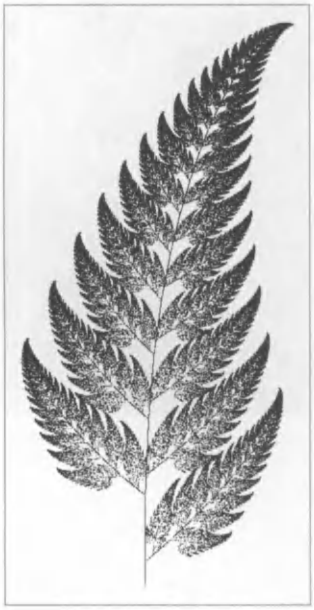


Figure 2-7 A fractal fern

Fractal Ferns

A more common example of fractal branching can be found in the plant kingdom. Trees, shrubs, and flowers all develop with a branching growth pattern that has a fractal character. Figure 2-7 shows a computer-generated fractal fern based on a deceptively simple scheme of symmetry and self-similarity. (The fern was made using Fractint.) Each frond of the fern is a miniature of the whole. A real fern is not self-similar to the same degree, yet it is amazing how realistic this idealized fern looks.

Weather: Chaotic Fractals

Some of the most powerful supercomputers run complex mathematical models in an attempt to improve weather forecasts, yet the success of this effort has been only moderate. A large investment in computational power purchases the ability to predict only a short time further ahead. The reason for this is not that the computers don't work or that the mathematical modelers are inept, but rather that the dynamics underlying the weather are chaotic. Weather is like the flow of water over Niagara Falls. If you launch a small leaf above the falls, where will it be a few minutes later after going over the falls? While a personal computer can easily project the orbit of the Voyager spacecraft far beyond the solar system, the largest supercomputer cannot with any accuracy predict the path of our ill-fated leaf. This is the difference between well-behaved and chaotic dynamic systems.

DEFINITION: A *dynamic system* is a collection of parts that interact with each other and change each other over time. A dynamic system is *chaotic* if small changes in the initial conditions of the system make large changes in the system at later times.

The weather is a great example of a dynamic system. There are periods of relative calm and predictability, like the calm patches on a disturbed lake. But as anyone knows who has watched the weather report on TV, there are always fronts on the way, low pressure areas with huge spiral arms slowly moving to the east, and hurricanes brewing in the Gulf.

Satellite pictures of weather patterns have become part of our cultural memory. They have a certain beauty to them and, from our present perspective, a definite fractal character. If the weather forecaster could zoom the satellite picture, the audience would be treated to a succession of equally detailed pictures as the nation-sized low pressure areas would give way to a picture of the wind eddies around their city. These satellite pictures can be thought of as a graphical representation of the chaotic weather dynamics. So now we have another route to fractals—pictures of chaos.

QUALITIES OF A FRACTAL

The different qualities of fractals that have come up in the discussion of these examples are summarized here. Note that not all of these qualities apply to every fractal.

Qualities of Fractals

- Fractional Dimension
- Complex Structure at All Scales
- Infinite Branching
- Self-Similarity
- Chaotic Dynamics

OF WHAT PRACTICAL USE ARE FRACTALS?

The second most common question about fractals after the question "What are they?" is some variation of "What earthly use do they have?" This is really a very reasonable question, but somehow we fractal fanatics are irritated by it. Imagine going to Paris to see the Mona Lisa in the Louvre and having someone ask you, "Fine, but what is it good for?" Let's see.

Mathematics Education

Fractals are educational because they visually illustrate many basic mathematical concepts and make an ideal vehicle for challenging visually oriented people with those concepts. While appreciation for graphic images is not a substitute for learning the abstract foundations of mathematics, being intrigued by dazzling fractal images can motivate a student to dig through math texts looking for abstract concepts that made possible the visual feast.

Mathematical subjects related to fractals include algebra, geometry, complex numbers, and calculus. Fractals are an excellent topic for high school or even junior high school mathematics projects. College level topics related to fractals include complex analysis, measure theory, and the study of dynamic systems. A recurring theme of this book is that one need not be an expert mathematician to appreciate fractals, so that if you have never had the opportunity to study any of these subjects, you can still understand and enjoy the fundamental concepts of fractals. However, those who do take on the discipline of learning mathematics will discover that the pleasures of fractal exploration will take on an added dimension. This is why educators are using fractals in the classroom: fractals are both accessible to beginning mathematics students and rewarding for mathematical experts.

Understanding Chaotic Dynamic Systems with Fractals

While we rarely think this way, the life of a person in our complex society is utterly dependent on both artificial and natural dynamic systems. As stated earlier, a dynamic system is a collection of parts that interact with each other and change each other over time. A few examples are power systems, the weather system, computer systems, the national and international economies, and even the planetary ecosystem. We say that dynamic systems can exhibit behavior that is stable or chaotic. You may feel the word "chaotic" has negative connotations, but it is not necessarily a bad thing. When you are roasting marshmallows in front of a campfire, eyes transfixed on the swirls of smoke twisting up to the sky, you are observing a chaotic dynamic system made up of the air, the fire, and the wood. That kind of chaos is a pleasure, not a problem. But when chaotic interactions in power systems cause blackouts, that is usually a bad thing (although certain criminals would disagree). Useful computer algorithms (equations) are sometimes stable for some numeric inputs but exhibit chaotic behavior for others. This is an important concept to understand—certain formulas "blow up" and act unpredictably at certain times. If such an algorithm is used to calculate the position of a spacecraft just before reentry, the experience of the chaotic region of the algorithm could have serious consequences.

As we have already seen in connection with our example of the weather system, fractals are intimately connected with chaos. In fact, many computer-generated fractals are created precisely by operating otherwise well-behaved algorithms in regions where they exhibit chaotic behavior. The study of fractals cannot help but increase our knowledge of the chaotic behavior of dynamic systems. Indeed, fractal theory may not only help us predict the weather, but it can also help us understand the limits of our ability to predict it.

Image Compression

Now let's move from chaos and weather to discuss an application of fractals for computers.

Most personal computer users have encountered compression utility programs like ARC and PKZIP that allow computer files to be stored in a very compact form. These compression programs take advantage of the redundancy in the pattern of bits that make up your file. Because graphic images consume so much disk space, the need for this kind of file compression becomes even more critical when storing graphics. For example, one of the typical new "super" VGA graphics adapters can display an image 1024 pixels wide and 768 pixels high (*pixels* are the small dots that make up a computer screen image). Because each of these pixels can be any of 256 colors, it takes 8 bits (or 1 byte) of storage to store the color of each pixel. Multiply that out, and you discover that storing one graphics image from your screen at that resolution on your disk takes 786,432 bytes. That is enough to take up the better part of a high density floppy disk. After compressing—with PKZIP, for example—the same image can often be stored in less than half the space.

Fractals are complex images, but what is amazing is that in many cases they can be represented by simple equations that consume little space. In some cases it is possible to identify patterns of self-similarity in a graphics image and compress the image storage by describing the self-similarity rather than drawing the image. Taking this concept one step further, consider attempting to identify fractal patterns in any graphics image, and compress storage by representing the images with the rules generating the fractals. Imagine how powerful a technique this could be, as it might allow huge amounts of information to be reduced to a simple formula made of five or six characters! Michael Barnsley, one of the originators of the Iterated Function Systems approach to generating fractals which we will discuss shortly, has started a company that is building a commercial venture on this idea of graphics image compression. Fractal compression techniques can reduce the size of an image as much as 100 times, reducing megabytes of files to tens of kilobytes.

The ability to compress and decompress images is one of the keys to new multimedia applications. If a single high-resolution image takes up nearly a megabyte of disk space, consider that a minute of high-definition full-motion video running at 30 frames per second requires 1800 times more, or nearly a gigabyte (a thousand million bytes!). Fractal compression techniques are right in the thick of the technological revolution that is bringing animation, video, and sound to your desktop.

Computer-Generated Simulation

Another application of fractals that you have almost certainly encountered is the computer-generated simulation. Movie special effects is a whole industry that uses many different technologies, ranging from animated artwork to miniature models. We have discussed how many natural objects from mountains to planets have a fractal nature. With the advent of high-resolution graphics workstations, it is possible with fractal formulas to generate realistic-looking computer images of mountains, trees, forests, and flowers. In the movie *Star Trek: The Wrath of Khan,* the entire Genesis planet was a computer-generated fractal landscape. The computer game *Starflight* also pioneered the use of fractal planets. In the popular mind, computer-generated images have a mechanistic quality, perhaps due to the fact that popular computer drawing and paint tools come equipped with a repertoire of regular shapes such as lines, circles, and squares. But if the computer artist can supplement those with tools that create fractal shapes with roughness, texture, branching, and cloudiness, then the mechanistic feel will be replaced by the earthiness of the natural world. Figure 2-8 shows a scene of a fractal planet as viewed from a fractal landscape. This example was generated with Fractint.

A New Artistic Medium

Fractals represent an opportunity for artists to utilize the computer as a new medium for their creativity. Fractals are appearing on book covers, wall paper, calendars, greeting cards, textile patterns, and gallery art. The use of fractals in art is not new. Artists from Van Gogh to Escher have incorporated fractal patterns and textures into their works. The difference is that earlier artists used the paint brush or the wood cutting knife, whereas today a new generation of artists have added the computer to their artistic tool kit.

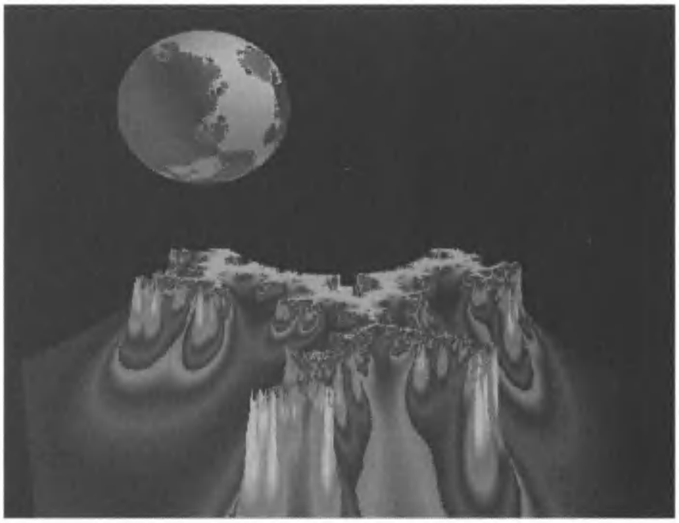


Figure 2-8 View of a fractal planet from a fractal landscape

Fractals Are Fun!

Despite the fact that it gets easier every day to argue the case that fractals are practically important, somehow all the evidence for the usefulness of fractals cited in the previous sections doesn't address the real truth. People who are visually oriented (who enjoy color, texture, and patterns) are naturally attracted to fractal images. People who in addition to visual imagination have a mathematical curiosity (no matter how little they may have actually studied math) are irresistibly attracted to fractals. If you also have a philosophical bent and an interest in computer graphics, then you and fractals are a match made in heaven!

The ultimate practical application of fractals is the sheer enjoyment of exploring, creating, coloring, designing, modifying, and contemplating fractal images. So enjoy!

AN EMERGING VIEW OF NATURE

If you have begun to feel that more is at stake with fractals than beautiful pictures, education, or image compression, you are on the right track. The universe of fractals is related to the larger issue of understanding the relationship between humanity and the natural world.

Ever since the Greek philosophers, our Western civilization has operated out of the idea that lines, circles, squares, and the other objects of classical geometry were somehow "more real" than nature itself, which contains few pure examples of these shapes. Plato postulated a world of ideal forms, where these perfect shapes resided unblemished. The world of human experience to Plato was but an imperfect and dim image of this ideal world. So, unable to live in this perfect world, people remake the natural world into a vision of imaginary perfection. Buildings must be square, shelves straight, and wheels round. Could it be that this deeply held world view is behind our impulse to bulldoze forests and build cities of rectangular skyscrapers laced with a gridwork of roads? Whatever the case, the irony is that classical geometry is used to model nature, and when the model doesn't fit, we blame nature rather than the model. What's worse, we then try to change nature to fit our preconceptions!

While this doesn't necessarily mean we should make buildings shaped like fractals, it does mean fractal geometry can often provide a much better "fit" for nature, and it can describe with great accuracy the structure of clouds, mountains, rivers, ferns, waterfalls, sunflower fields, and even weather. It may also tell us more about how the weather works, secrets of biochemistry, or insights about how people think. What is of critical importance is not the success of the theory but the reorientation of fundamental thinking. This emerging view of nature is more humble, less arrogant. The deepest wonder is for nature itself, not our attempts to model it and understand it.

THE COMPUTER AS A WINDOW TO CHAOS

Examples of chaotic phenomena occur in many disciplines, often as anomalous special cases. In many fields you will find the term "ill-behaved" used to describe chaotic phenomena. This is a very curious term indeed, reminiscent of the attitude that children are meant to be seen and not heard, and when they are heard, they are bad! But can the notion of "badness" be extended to a mathematical algorithm? That question will remain unanswered here, and this observation will have to suffice: where fractals are concerned, what is "bad" often turns out to be "good"!

For years, algorithms that exhibited chaotic behavior were ignored or relegated to footnotes for the curious. Chaotic behavior represented as numbers is very hard to understand. But make this chaotic behavior visual and it can be directly grasped. With the advent of low-cost video adapters, a personal computer can now be used as a tool to visualize such chaotic dynamics—a kind of window to chaos.

We've spent a good deal of time drawing parallels between nature and fractals and revealing ways in which fractals play a role in science. Now we are going to go into more depth and explain how a simple fractal is generated on a computer. You don't need to understand this to run the Fractint program that comes with this book, but knowing how the fractal is made can enhance your appreciation of its physical beauty. This section explores a whole category of fractals created by what are known as escape-time algorithms. The term *escape time* comes from the fact that the algorithm works by determining when an orbit "escapes" a circle, as will be explained shortly. The most famous fractal of them all, the Mandelbrot set, is an example of this kind of fractal. Let's have a look at how pictures of this fractal are created.

How the Escape-Time Mandelbrot Set Is Generated

To appreciate the Mandelbrot fractal, a few mathematical preliminaries are needed. We will be using these rules later, so it is important to understand them. A *set* is simply a collection of objects of some kind. In the case of the Mandelbrot set, those objects are the coordinates of locations on a mathematical map called a complex plane. These particular locations are unusual because they are made up not of the familiar real numbers we use every day for finances and measuring, but rather what are called complex numbers. You might think of this plane as being like the map of a city with rectangular streets and avenues. The horizontal x-axis might be considered a collection of avenues numbered from some large negative number to some large positive number. The vertical y-axis would be unusual in that it corresponded to complex numbers (from negative large to positive large) with names such as $2i$, $6.529i$, and so on.

Complex Numbers and the Complex Plane

What's so special about complex numbers? First, they are unusual in that they are composed of two parts, one a familiar real number, the other an imaginary number. The imaginary part is most interesting. With real numbers, you are not allowed to take the square root of a negative number, and this operation is not defined. With complex numbers this is allowed, and the result of taking the square root of -1 is a special number designated "i." Looking at this another way, the number i is defined to be the complex number such that $i^2 = -1$, which is another way of saying that i is the positive square root of -1. Every complex number is written as the sum of a real number and another real number times i, or $a + bi$, where a and b are real numbers.

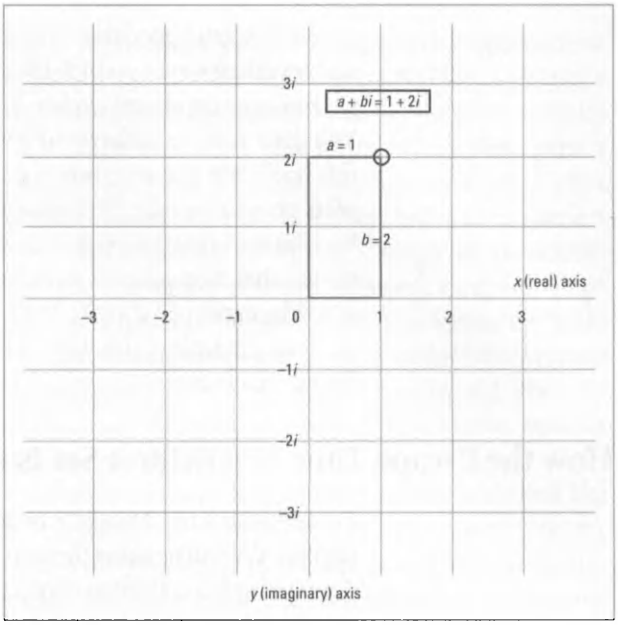


Figure 2-9 The complex plane

Complex numbers can be graphed using a "real" axis (for the "a" part), and an "imaginary" axis (for the "bi" part). Figure 2-9 shows how the complex number $a + bi$ can be graphed using the two axes on the complex plane. The place where the two axes meet is called the *origin*, and it is the graph of the complex number $0 + 0i$, which is the familiar zero from ordinary arithmetic.

Using the fact that $i^2 = -1$, and the ordinary rules of arithmetic, you can do arithmetic using complex numbers. For example, $(2 + 3i) + (-3 + 2i)$ is calculated by adding the real parts together and the imaginary parts together, so the answer is $((2 - 3) + (3 + 2)i)$ or $-1 + 5i$.

Multiplying is a little more complicated. The expression $(2 + 3i) \times (-3 + 2i)$ is multiplied out exactly as it would be in algebra if "i" were a variable, and then simplified using $i^2 = -1$. (See the Appendices for more on complex numbers.)

Distance Between Complex Numbers

The next concept we need to grasp is how to calculate the distance between complex numbers. Imagine our map is Manhattan, New York City, U.S.A., where the x-axis is avenue numbers and the y-axis is street numbers. Suppose you live in a high-rise apartment at 3rd Street and 4th Avenue, and a friend of

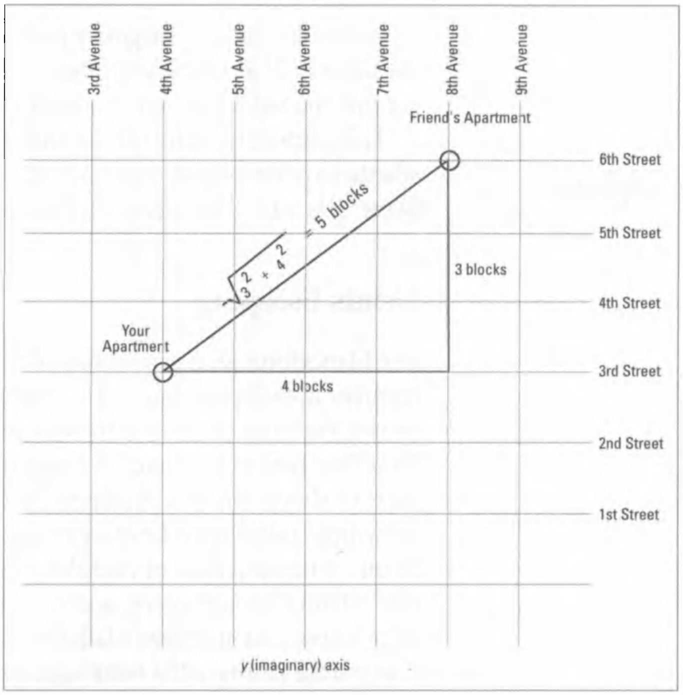


Figure 2-10 Distance between two apartments in Manhattan

yours lives in another high-rise apartment at 6th Street and 8th Avenue. You are peeking at your friend's apartment through a telescope, and you are curious about how far away it is. For the sake of this discussion, let's say that New York blocks are perfectly square, so a block along the avenues is the same distance as a block along the streets. Figure 2-10 shows the section of Manhattan where these two apartments are located.

You can see that the apartments are on the ends of the hypotenuse of a right triangle. One leg of the triangle, the leg that runs in the avenue direction, is three blocks long. The other leg is four blocks long. Using the Pythagorean theorem, we see that the distance is five blocks, because $5 = (3^2 + 4^2)$. The formula for the distance between two complex numbers is based on the same idea. The avenues are the real part of the complex number, and the streets are the imaginary part. The distance formula is just the Pythagorean theorem applied to the distance between the two complex numbers in the x-axis (real) direction and the y-axis (imaginary) direction.

To make a Mandelbrot set, we need the distance from the complex number $a + bi$ to the origin $0 + 0i$. Again, this distance is the square root of the sum of the squares of the real and imaginary parts, or ($a^2 + b^2$). A shorthand way of writing the distance of a complex number $a + bi$ to the origin is $|a + bi|$, and when you see this you will know that the real meaning is ($a^2 + b^2$).

The purpose of using this formula in generating a Mandelbrot set is to test whether a point is inside a circle of radius 2 centered on the origin of the complex plane. If $|a + bi|$ is less than 2.0, then the point is inside the circle.

Orbits Escaping

The Mandelbrot set is a collection of points "in" the complex plane. In order to calculate it, each point is tested to determine if it is in the set. Here is how the test works. Each test point determines a sequence of points in the complex plane (you'll see how in a minute). A sequence is just a list of numbers. A subscript is used to show which is the first, the second, and so forth. This sequence is sometimes called the *orbit* of a particular test point, such as the point $.37 + .4i$. Think of the sequence of complex numbers as the successive positions of an object flying through space, and you'll see why the term "orbit" is appropriate. Here is how a point passes or fails the test for membership in the Mandelbrot set. If any of the points in the orbit belonging to the test point are outside the circle of radius 2 about the origin, then that test point is *not* in the Mandelbrot set. If all of the orbit positions remain inside the circle of radius 2, then the test point is in the Mandelbrot set. Another way to put this is that the Mandelbrot set consists of all those test points whose orbits never escape the circle of radius 2, but whiz around forever inside it. A radius larger than 2 would work fine for this computation, but a smaller radius would not. A radius of 2 is the smallest radius centered on the origin that contains all of the Mandelbrot set, as you can see in Figure 2-15 later.

The Magic Formula

How are these orbits generated from the test point? Suppose the point to be tested is the one on the origin, $a + bi$, which we will call c. The sequence of points generated by c will be designated $z_0, z_1, z_2, z_3, \ldots, z_n, \ldots$ Here, z_n is the nth member of the sequence, counting up from zero, and the little dots are mathematics-ese for "and so forth." (By the way, mathematicians often use the letter "z" to represent complex numbers.) The first element of the sequence is the origin itself, so $z_0 = 0 + 0i$, that is, $z_0 = 0$. To get the next member of the sequence, the previous

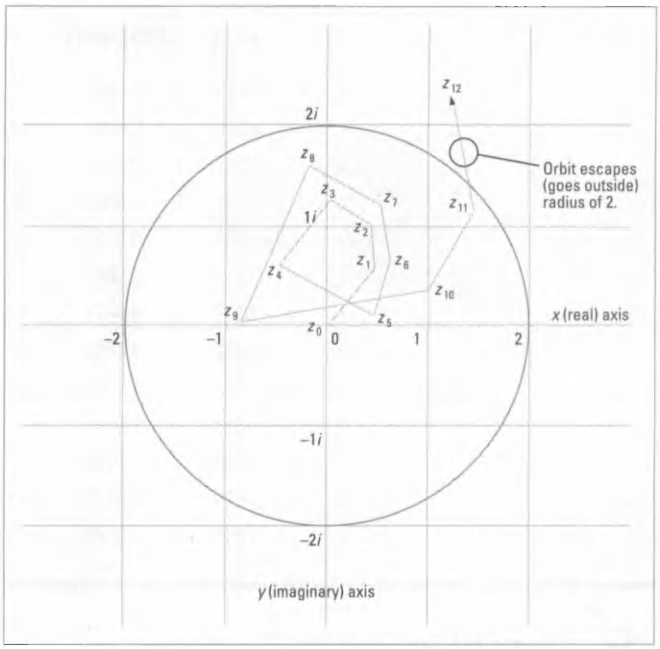


Figure 2-11 The escaping orbit of .37 + .4*i*

member is multiplied times itself and added to *c*. This sequence-building process is described by the equation:

$$z_0 = 0 + 0i$$
$$z_1 = z_0^2 + c$$

$$z_{n+1} = z_n^2 + c$$

Let's use a real point. Suppose the test point is the complex number .37 + .4*i*. Calculating z_1 is easy, because $z_1 = z_0^2 + (.37 + .4i)$, and $z_0^2 = 0 \times 0 = 0$, so $z_1 = .37 + .4i$. The distance of this point to the origin is $(.37^2 + .4^2)$, or about .545, which is well within the circle of radius 2. The orbit value z_2 is $(.37 + .4i)^2 + (.37 + .4i)$. To simplify all this, we used a computer, and Table 2-3 shows the orbit sequence values for the test point .37 + .4*i*, along with the distance from the origin of each sequence member. Figure 2-11 shows a plot for the orbit formed by this table of results.

The orbit starts to swing outward, comes back in to a minimum value at z_5, and swings around outward again. The orbit member z_{12} is the first one to wander outside the circle. Notice that the distance value for z_{12} is 3.950, almost

		Real		Imaginary			Distance
z_0	=	0.000	+	0.000*i*	$\lvert z_0\rvert$	=	0.000
z_1	=	0.370	+	0.400*i*	$\lvert z_1\rvert$	=	0.545
z_2	=	0.347	+	0.696*i*	$\lvert z_2\rvert$	=	0.778
z_3	=	0.006	+	0.883*i*	$\lvert z_3\rvert$	=	0.883
z_4	=	−0.409	+	0.410*i*	$\lvert z_4\rvert$	=	0.580
z_5	=	0.369	+	0.064*i*	$\lvert z_5\rvert$	=	0.375
z_6	=	0.502	+	0.447*i*	$\lvert z_6\rvert$	=	0.672
z_7	=	0.422	+	0.849*i*	$\lvert z_7\rvert$	=	0.948
z_8	=	−0.173	+	1.117*i*	$\lvert z_8\rvert$	=	1.130
z_9	=	−0.848	+	0.014*i*	$\lvert z_9\rvert$	=	0.848
z_{10}	=	1.089	+	0.376*i*	$\lvert z_{10}\rvert$	=	1.152
z_{11}	=	1.415	+	1.219*i*	$\lvert z_{11}\rvert$	=	1.868
z_{12}	=	0.885	+	3.8500*i*	$\lvert z_{12}\rvert$	=	3.950

Table 2-3 Test orbit for .37 + .4*i*

double the test circle radius. Figure 2-11 shows a plot of this escaping orbit in the complex plane.

This calculation shows that the test point .37 + .4*i* is not in the Mandelbrot set because its orbit escapes the circle.

Nonescaping Orbit

Now, changing this complex number just a little gives a different result. Table 2-4 shows the orbit of the point .37 + .2*i*. One hundred values were calculated, but not all are shown. Figure 2-12 shows a plot of these values. Note how nice and symmetrical this orbit is.

If we calculate the orbit sequence starting with .37 + .2*i*, we discover that the orbit values stay well inside the circle for the first 100 orbit calculations. This raises a difficult point. Just because the first 100 orbit values are within the circle doesn't mean some later values might not escape. So how do we ever know a test point is in the Mandelbrot set? The answer is that we don't really know. The Mandelbrot set has to be approximated by setting an arbitrary cutoff point for how many orbit values will be tested. So for practical purposes, we will say that the test value .37 + .2*i* is in the Mandelbrot set because for the 100 orbit sequence

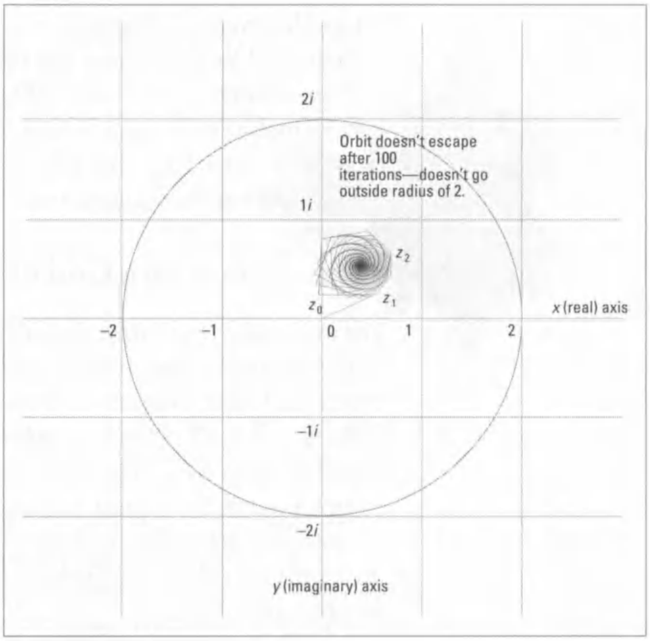

2i
Orbit doesn't escape after 100 iterations—doesn't go outside radius of 2.
1i
z_2
z_0
z_1
x (real) axis
−2
−1
0
1
2
−1i
−2i
y (imaginary) axis

Figure 2-12 The nonescaping orbit of .37 + .2*i*

		Real		Imaginary			Distance
z_0	=	0.000	+	0.000i	$\lvert z_0 \rvert$	=	0.000
z_1	=	0.370	+	0.200*i*	$\lvert z_1 \rvert$	=	0.421
z_2	=	0.467	+	0.348*i*	$\lvert z_2 \rvert$	=	0.582
z_3	=	0.467	+	0.525*i*	$\lvert z_3 \rvert$	=	0.703
z_4	=	0.312	+	0.690*i*	$\lvert z_4 \rvert$	=	0.758
z_5	=	−0.009	+	0.631*i*	$\lvert z_5 \rvert$	=	0.631
...							
z_{96}	=	0.352	+	0.479*i*	$\lvert z_{96} \rvert$	=	0.594
z_{97}	=	0.264	+	0.537*i*	$\lvert z_{97} \rvert$	=	0.598
z_{98}	=	0.152	+	0.484*i*	$\lvert z_{98} \rvert$	=	0.507
z_{99}	=	0.159	+	0.347*i*	$\lvert z_{99} \rvert$	=	0.382
z_{100}	=	0.275	+	0.310*i*	$\lvert z_{100} \rvert$	=	0.415

Table 2-4 Test Orbit for .37 + .2*i*

values that were checked, all were confined to the inside of the test circle. (Fractint will let you control this parameter.)

Even though only the first 100 values were checked, this orbit looks very convincingly nonescaping. It has a definite, regular inward spiral that appears to converge to a point. Fractint lets you watch these fascinating orbits come and go while fractals are being generated.

Testing Points on a Grid of Pixels

The next problem is how to test all the points of a given set in the complex plane. This is impossible, because there are an infinite number of points to test. But it isn't really necessary to test all the points. The end objective is to make a picture of the Mandelbrot set on a computer screen. The solution is to map the pixels (small dots) on the computer screen to the complex plane, and just test those complex points that correspond to a pixel. This is analogous to coloring just the street/avenue intersections of our Manhattan map. When this is accomplished, the pixels are colored one color if the test value is in the Mandelbrot set and another color if it isn't.

The Final Black-and-White Mandelbrot Algorithm

Let's summarize what has been said so far, and use a little different notation. For each pixel on the computer screen, the complex number z_{pixel} mapped to that pixel will be tested to see if it is in the Mandelbrot set or not. z_{pixel} is the test point we discussed in the above examples and therefore, it is the variable "c" in the Mandelbrot orbit formula $z_{n+1} = z_n^2 + c$, so $c = z_{pixel}$. We will define the sequence of complex numbers (called the *orbit sequence*) $z_0, z_1, z_2, \ldots, z_n, \ldots$. The first member of the orbit sequence is the origin, so $z_0 = 0 + 0i$. The second member of the sequence, z_1, is $z_0^2 + c$, or c itself, because z_0 is zero. If c is already outside the circle, we are done; we'll color the pixel white.

The next member of the sequence, z_2, is the first member squared plus c, so $z_2 = z_1^2 + c$. We must plug values into z and then, after checking to see if the new value is outside the circle, the process is continued. In general, each orbit value z_{n+1} is obtained from the previous orbit value z_n by the formula $z_{n+1} = z_n^2 + c$. Each time a new z_{n+1} is calculated, it is tested to see if it has gone outside the circle. The notation for the "in the circle test" for our Mandelbrot set is whether $|z_{n+1}| < 2$, where $|z_{n+1}|$ is the distance of z_{n+1} to the center point. If $|z_{n+1}| < 2$ is true, we calculate another iteration; otherwise, the orbit value has escaped, and it is colored white. Because we do not want the computer to calculate forever, we have a maximum

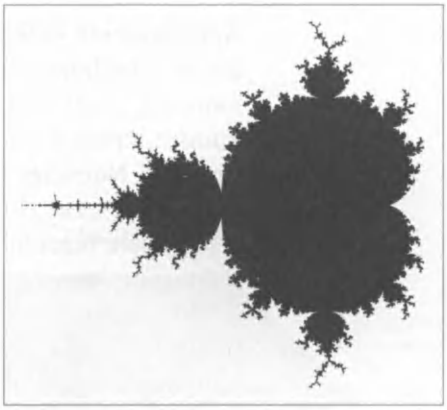


Figure 2-13 The Mandelbrot set

iteration cutoff, and if the orbit has not escaped by the time we reach the cutoff, we quit and declare the point to be colored black.

Figure 2-13 shows the result of this little exercise, after all the points are colored. The Mandelbrot set consists of all those points we colored black—points whose orbits always stayed inside the circle (or at least, stayed inside for as long as the computer had the patience to wait). The actual edge of what appears as a lake in the figure is a fractal in the sense of the definitions earlier in this chapter. Measured with a small enough "inch stick," the coastline of "Mandelbrot Lake" can be made as long as you want, and it has a fractal dimension greater than one. In fact, as mentioned earlier, the fractal dimension of the Mandelbrot set coastline has been proved to be exactly 2.0!

Where Did the Mandelbrot Fractal REALLY Come From?

In Figure 2-13, there are two big "bays" in the giant lake, with smaller baylets at the top and bottom. The whole "coastline" is an impossibly detailed nesting of bay within bay within bay, resulting in thin, jagged filaments shooting out like static electricity. This is a picture of a set that James Gleick called "the most complex object of mathematics."

By contrast, look at this formula, placed in a box in big, bold, type, so you can soak it in, meditate on it, and wonder about it.

$$\boxed{z_{n+1} = z_n^2 + c}$$

Appearances of simplicity CAN be deceiving. The innocent-looking formula $E = mc^2$ somehow encapsulates the whole theory of relativity. Not so here! The formula $z_{n+1} = z_n^2 + c$ is no more, and no less, than what it appears to be. Take a number, square it, and add a number. Nothing fancy, nothing tricky, nothing profound. No energy, no mass, no real-world stuff. Yet this is the formula which, given a few more details about repeating and checking for escaping orbits, generates the beautiful Mandelbrot set. How can such a wondrous and complex shape come from the absurdly simple formula $z_{n+1} = z_n^2 + c$?

Here is a hint of where to look for the mysterious source of fractals. The formula $z_{n+1} = z_n^2 + c$ may be simple, but it is repeated over and over a very large number of times. At the very beginning of this chapter, a fractal was described as an infinite pattern somehow compressed into a finite space. There are many different kinds of fractals, but however different they are, and however diverse their methods of generation, all of them have some kind of iterative scheme at their heart. The secret: formulas play a less important role in a fractal compared to the powerful iterative powers at work. Yet this is not enough to explain fractals completely. And while the mathematics and iterative method are logical, perhaps limitations of the human mind will never allow us to fully understand fractals. For some of us, therein lies their appeal!

Fractals Come Alive: Escape-Time Colors

Our black-and-white coloring scheme for each test point works well and provides a beautiful picture. But there is one more refinement we can make to an escape-time fractal that gives an additional and wondrous level of beauty: color.

As we have seen, the Mandelbrot set is defined as the set of points that do not escape a circle of radius 2 under iteration of the formula $z_{n+1} = z_n^2 + c$. And we have seen that a picture of the Mandelbrot set can be made with two colors, one for the points in the set, one for the points out of the set.

A brightly colored variation of this picture can be created by coloring the points *not* in the Mandelbrot set—the ones that escaped the circle—according to how long it takes for the orbit to escape, where "how long" means "how many orbits." We can use the number of iterations to control the final color of the test-point pixel. So if the test point escapes in a few iterations, the color might be red, but if it takes many iterations it might be colored blue.

Figure 2-14 shows a more graphic view of how escape-time coloring works. The bottom of the diagram shows the familiar two dimensions of the complex plane, with two points, *a* and *b*, selected for testing and coloring. The vertical

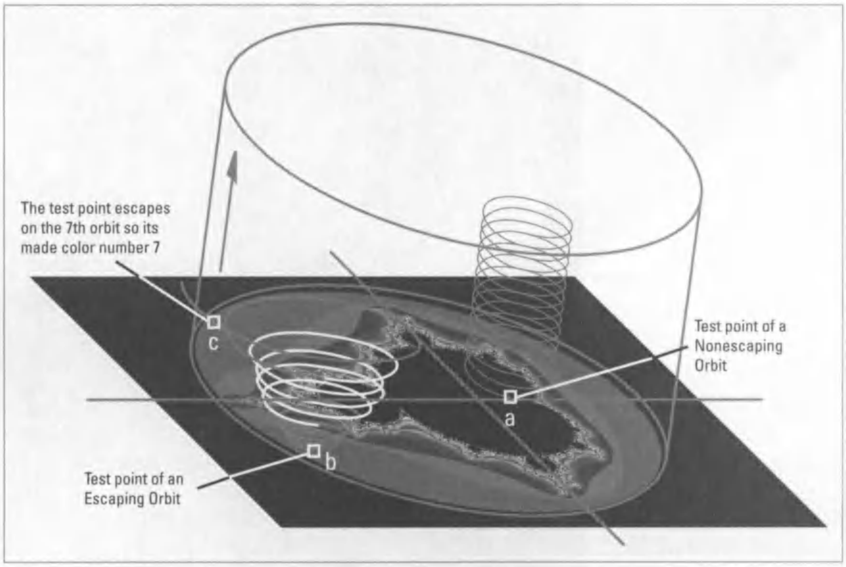

The test point escapes on the 7th orbit so its made color number 7
c
Test point of a Nonescaping Orbit
a
b
Test point of an Escaping Orbit

Figure 2-14 Escape-time coloring of the Mandelbrot set

axis represents the number of times the formula is iterated. You can imagine the 2" radius escape circle as a cylinder that is stretched into the third dimension, with the iteration values on the vertical scale color-coded. Therefore, the vertical level reached when the orbit escapes the cylinder is used to color the test pixel according to the color for that level. In our figure we show that test-point *a* forms a spiral that never escapes, so it is colored the "inside color" (blue in Fractint). Test-point *b* forms an orbit that escapes on the seventh iteration, so it is made of color number 7. The overall effect of this coloring scheme divides the Mandelbrot fractal into bands reminiscent of terraced rice paddies on a Chinese mountainside. Each band represents an area where the orbits began with points in these bands escaping at the same iteration. Near the "lake edge" of the Mandelbrot set, these bands become more and more irregular and bent. You can see these bands in Figure 2-15.

The spectacular stripes of the Mandelbrot set rendered with escape-time coloring should not be confused with the set itself. Mathematically, the Mandelbrot set consists of the solidly colored lake area. The colorful stripes are points *near* the Mandelbrot set. However, this distinction is not always made, and in popular fractal parlance the Mandelbrot set often refers to the whole colorful image, lake, stripes, and all.

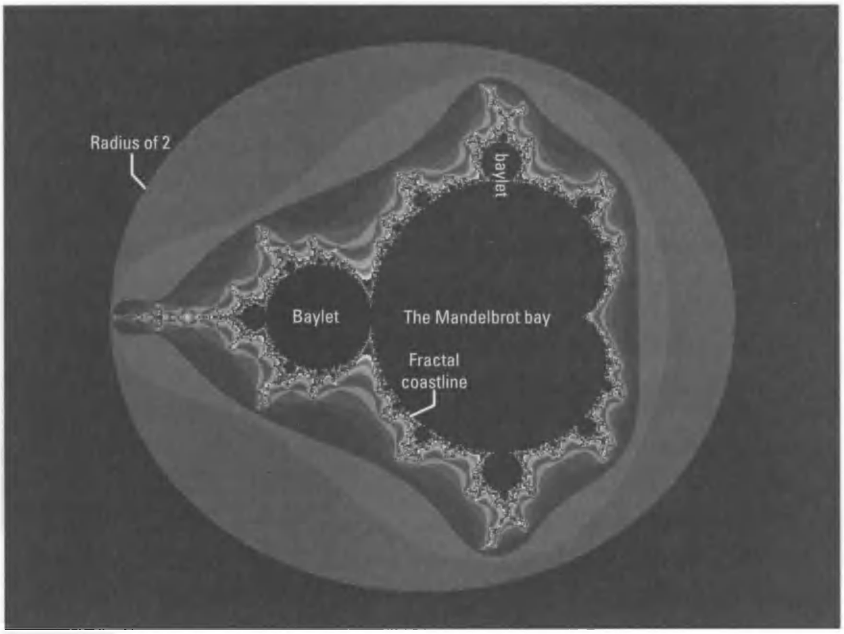


Figure 2-15 Colorized Mandelbrot set (in grayscale with the parts identified)

Zooming In, or How Big Is a Fractal?

Because there are too many possible points to calculate—infinitely many, to be exact—the complete Mandelbrot set cannot be rendered in a picture. In common computer practice, a rectangular grid of numbers is used for the values of *c*, using as fine a mesh as can be resolved by the particular graphics hardware. To show the complete Mandelbrot set, these numbers must span a range of approximately –2 to 2 in the *x* and *y* dimensions. However, there is no law that says that the entire Mandelbrot set from –2 to 2 must be included in the view. By picking a very small piece of the complex plane as the corners for the calculation grid, a small area of the fractal can be blown up with a zoom effect. For example, you can look at the fractal between –.2 and +.2 or –.02 and +.02. From what we have said so far, you are undoubtedly prepared for the fact that the Mandelbrot set, being a fractal, is just as interesting in these microscopic views as it is in the large view. That is indeed the case. Even a modestly powered personal computer can reveal staggering patterns in the Mandelbrot set. Let us do a quick calculation to see just how staggering.

Fractint allows zooming in successively on a fractal ten times, magnifying the image a maximum of about twenty-five times for each zoom. The limit of ten

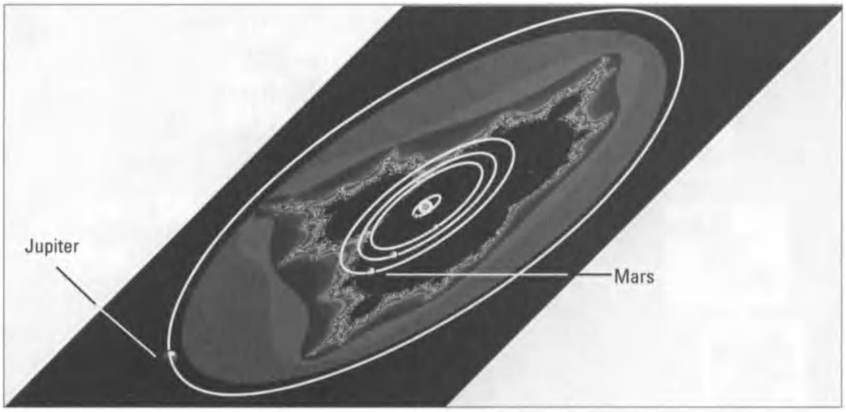


Figure 2-16 A giant Mandelbrot set swallows the orbit of Mars

zooms is not mathematical but is due to the computer's representation of numbers. At the most extreme magnification, a small patch of the complex plane about .000000000001 units (1.0×10^{-12}) wide fills the screen. Using the width of the Mandelbrot set of 4.0, and the width of the physical screen of about a foot, we can calculate how big the complete Mandelbrot set would be at the same scale.

Don't peek at the answer—guess! You're probably thinking that the giant Mandelbrot set would be pretty big, or we wouldn't be making much of a fuss about it, so maybe the answer is...ahhh...as big as a football field? Maybe a mile or two? Well, that's a brave answer. Indeed, if the giant Mandelbrot set were a mile wide, and because there are about twenty-five *million* different one-foot-wide patches in a square mile, you could be pretty busy charting them all.

But a mile wide is the wrong answer. A Mandelbrot set blown up to the scale of the most extreme zoomed view you can see on your PC screen with Fractint would be *one billion miles wide.* That is *ten times* the distance from the Earth to the sun; a bit greater than the diameter of the orbit of Jupiter. Figure 2-16 shows the relative sizes of this giant Mandelbrot set and the solar system.

What are the chances, then, that in your fractal explorations you will find a piece of the Mandelbrot set never before seen with human eyes? Not only pretty good, but virtually certain, as a matter of fact. You may have heard of a company that for a fee will name a star after you and record it in a book? Maybe the same thing will soon be done with the Mandelbrot set!

Mandelbrot and Julia Sets

Although the magnitude of exploration possibilities so far discussed is already of an astronomical size, you should be warned that the parade of endless fractal vistas has not even begun! The Mandelbrot set can be viewed not only as a fascinating fractal in its own right, but as an infinite "catalog" of a related class of fractals, called Julia sets. Each *point* of the Mandelbrot set may be considered an index pointing to a specific Julia set. These Julia sets are named after the French mathematician Gaston Julia, who discovered them.

Here is how Julias are formed. Consider a point c in a picture of the Mandelbrot set, and let it be inside or outside the "lake" that is the Mandelbrot set proper. Given this fixed point c, let's apply a slight modification of the escape-time algorithm for calculating the Mandelbrot set. In the calculation of the Mandelbrot set, the c in the formula $z^2 + c$ was set to the value $z_{pixel,}$ which changes for each pixel being colored. In the Julia set calculation, by way of contrast, the value of c is kept *fixed* for the entire image and just z changes. This little trick results in a new type of fractal. Changing the value of c changes the entire Julia set to another Julia set. Thus, there is no one Julia set, but rather an infinity of them, one for each value of c. That same number c corresponds to *one point* of the Mandelbrot set, so that one point may be considered as the index of the Julia set.

Figure 2-17 shows a picture of the Mandelbrot set surrounded by smaller pictures of Julia sets, with numbers connecting the Julia sets with the corresponding index points on the Mandelbrot set.

Note that Julia sets whose Mandelbrot index is inside the Mandelbrot lake have a lake themselves, whereas index points well outside the Mandelbrot lake do not have a lake. Some of the most interesting Julia sets have an index near the shore of the Mandelbrot lake. As the index approaches the shore from within the Mandelbrot lake, the Julia set lake's shoreline becomes more and more convoluted, until it explodes into fragments just as the index "hits the shore." In fact, this phenomenon can be used as the definition of the Mandelbrot set (which is, you recall, just the lake part of the escape-time picture of the Mandelbrot set). The Mandelbrot set consists of exactly those Julia indices of Julia sets with lakes in one connected piece.

This idea of one fractal being a catalog for a whole family of other fractals is a quite general idea. Later on in the book, when we are discussing other kinds of fractals, we will refer to the catalog fractal as the Mandelbrot form, and the family of fractals that correspond to the indices as the Julia form. This relationship makes sense even though the iterated formulas used to calculate the fractals are very different than the familiar $z^2 + c$ formula. When we want to make it clear that we mean the original Mandelbrot or Julias, we will speak of the "classic" Mandelbrot/Julia.

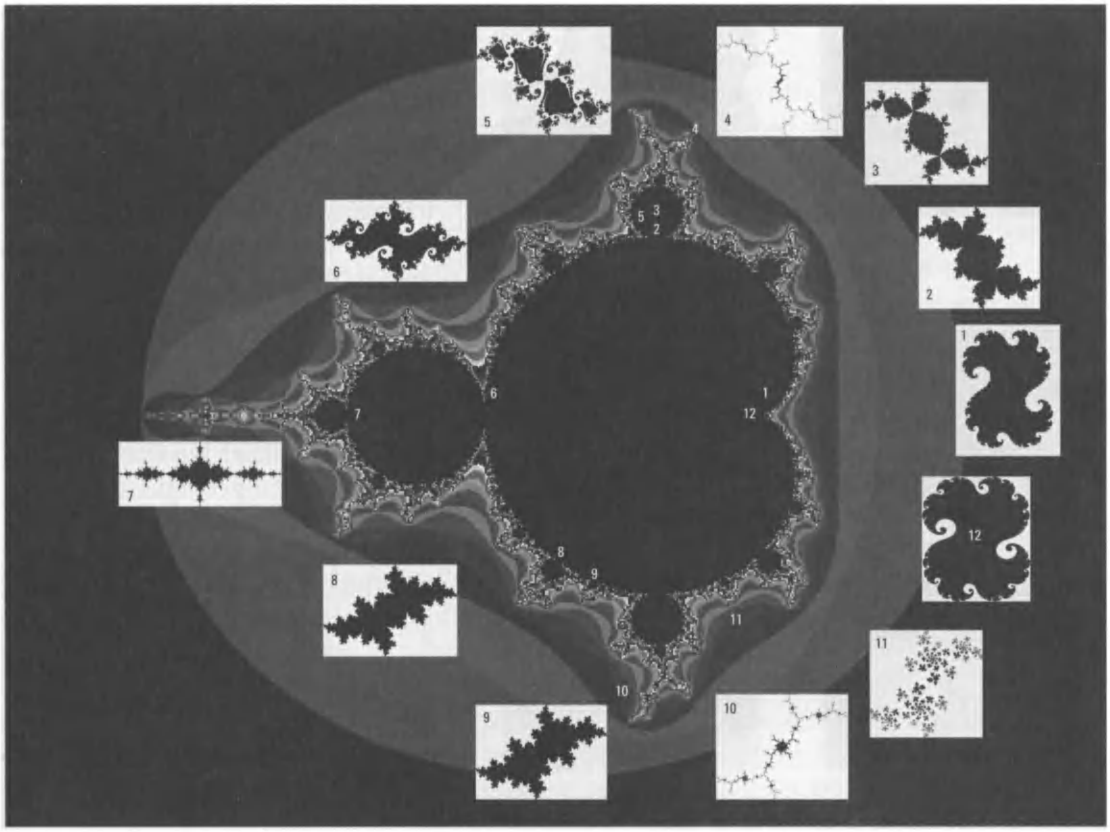


Figure 2-17 Julia family

The Ubiquitous Mandelbrot Set

In physics and mathematics, there are certain numbers that appear over and over again, sometimes in completely different contexts. A good example is the number π. The definition of π comes from geometry; it is simply the ratio between the circumference and diameter of a circle. But π is ubiquitous: it pops up again and again in connection with waves, power systems, complex numbers, exponentials, and logarithms.

In a similar way, you will find the familiar bulging shape of the Mandelbrot set reappearing over and over in miniature form, both within itself and as a detail

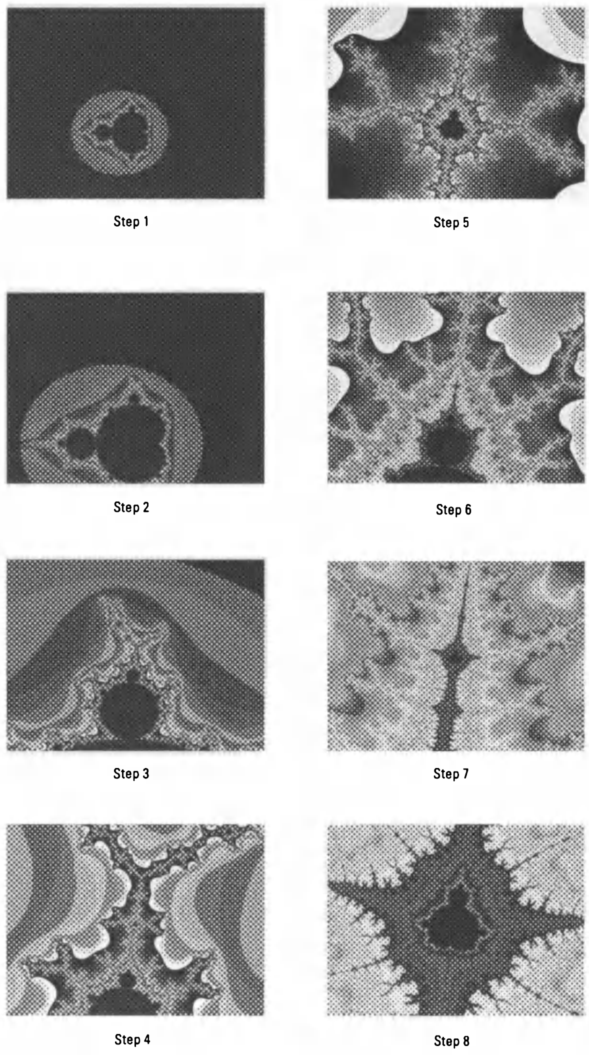


Figure 2-18 Fractal zoom in steps

within totally different fractals. Figure 2-18 shows several "baby Mandelbrots" within a sequence of successively greater magnification zooms.

Given the fundamental nature of fractals, which has to do with the existence of infinite detail, at greater and greater magnification, it is not too surprising to find baby Mandelbrots inside the original Mandelbrot fractal. But suppose we use the same approach to fractal generation (coloring pixels by iterating a formula), but change the formula to something completely different, say, $z_{n+1} = c \times \text{cosine}(z_n)$. This formula doesn't look anything like the Mandelbrot formula, and neither does the generated fractal. Yet buried within the fractal is the shape shown in Figure 2-13. Another baby Mandelbrot! This is not an isolated example—it happens again and again. The ubiquitous Mandelbrot set shape is to fractal theory what the number is to mathematics and engineering. Indeed, the plaque on the Pioneer spacecraft should have contained a Mandelbrot set engraving!

Now that we've covered the Mandelbrot in great detail, let's take a look at some other kinds of fractals.

Higher Dimension Mandelbrot and Julia Sets

Because the idea of Einstein's relativity theory has permeated popular consciousness, most people have heard of the idea of spaces with more than three dimensions. So you are probably not surprised to discover that fractals can be defined with four dimensions.

Four-Dimensional Number Systems

The classical Mandelbrot and Julia sets use complex numbers, which have the interesting property that they are two-dimensional (can be represented naturally as points on the plane) but still have all the algebraic properties of familiar real numbers. You can add, subtract, multiply, and divide them and all the usual rules of arithmetics apply. Why not use higher dimensional numbers instead of complex numbers? If a complex number may be represented as $x + iy$, why not define $x + iy + jz + kt$, for some suitable i, j, and k, where x, y, z, and t are real numbers? Extending our analogy mentioned earlier connecting complex numbers with New York addresses, x and y might refer to streets and avenues, and z might be the floor of a building where someone lives. The fourth dimension is a bit harder, perhaps we could think of t as referring to the date and time.

As it turns out, there are several alternative ways to mathematically define four-dimensional extensions of the complex numbers along these lines. None of these extensions quite satisfy all the arithmetic properties of real numbers, but they can be used to define fractals. These alternatives were studied extensively by mathematicians at the turn of the century. (See Appendix C, *Complex and Hypercomplex Numbers*, for more details about two- and four-dimensional numbers.)

The most famous four-dimensional extensions of the complex numbers are the quaternions, which are very useful in physics. Because no four-dimensional number system can satisfy all the algebraic properties of real numbers, some property must fail. For quaternions, $q_1 \times q_2$ and $q_2 \times q_1$ are not always the same (the commutative law of multiplication fails). Alan Norton, an associate of Benoit Mandelbrot at the IBM's Thomas J. Watson Research Center, introduced the world to quaternion Julia sets in 1982, and John Hart extended this work in 1989.

In the Fractint program that comes with this book, we resurrect a forgotten four-dimensional number system from the dustbin of mathematical history, the hypercomplex number system, and use it to generate four-dimensional fractals. Hypercomplex numbers satisfy the commutative law, but you cannot always divide by nonzero numbers (the existence of multiplicative inverses of nonzero elements sometimes fails). However, hypercomplex numbers have the advantage over quaternions that familiar mathematical functions such as sine, cosine, and the exponential function that work with complex numbers can easily be extended to hypercomplex numbers. Four-dimensional fractals use the fact that the fundamental formula used to generate Mandelbrot and Julia sets, $z^2 + c$, works just as well using quaternions or hypercomplex numbers as regular complex numbers. But the resulting Mandelbrot or Julia fractal is four-dimensional instead of two dimensional!

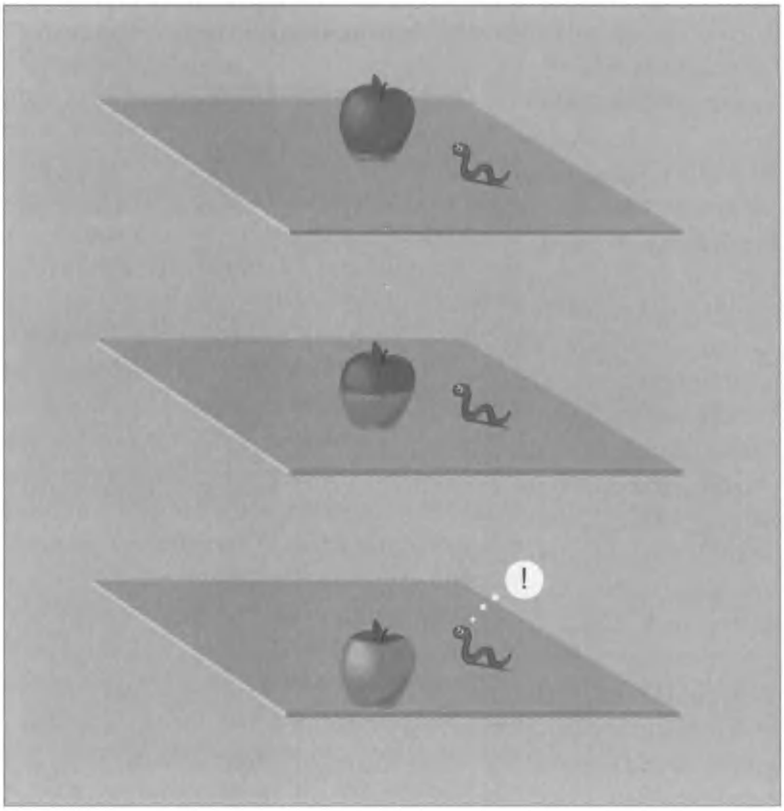


Figure 2-19 Explaining an apple to a flatlander

Visualizing Four-Dimensional Fractals

Given that you and I live in three-dimensional space, it is hard to imagine a four-dimensional fractal. The classic satirical book *Flatland* (Edwin A. Abbot, Dover Publications, NY, 1952) discusses problems imaginary creatures living in two or three dimensions might have understanding higher dimensions. Our solution is the same as the solution of the *Flatland* characters. As three-dimensional creatures we can understand four-dimensional objects in terms of three-dimensional slices. If you wanted to explain an apple to a two-dimensional creature living somehow in a plane, you could show him several different slices (cross-sectional views) of the apple. Figure 2-19 shows a flatland creature contemplating apple slices. Now think of yourself as a three-dimensional "flatland" creature and look at the three-dimensional "slices" of a hypercomplex Julia set shown in the Figure 2-20.

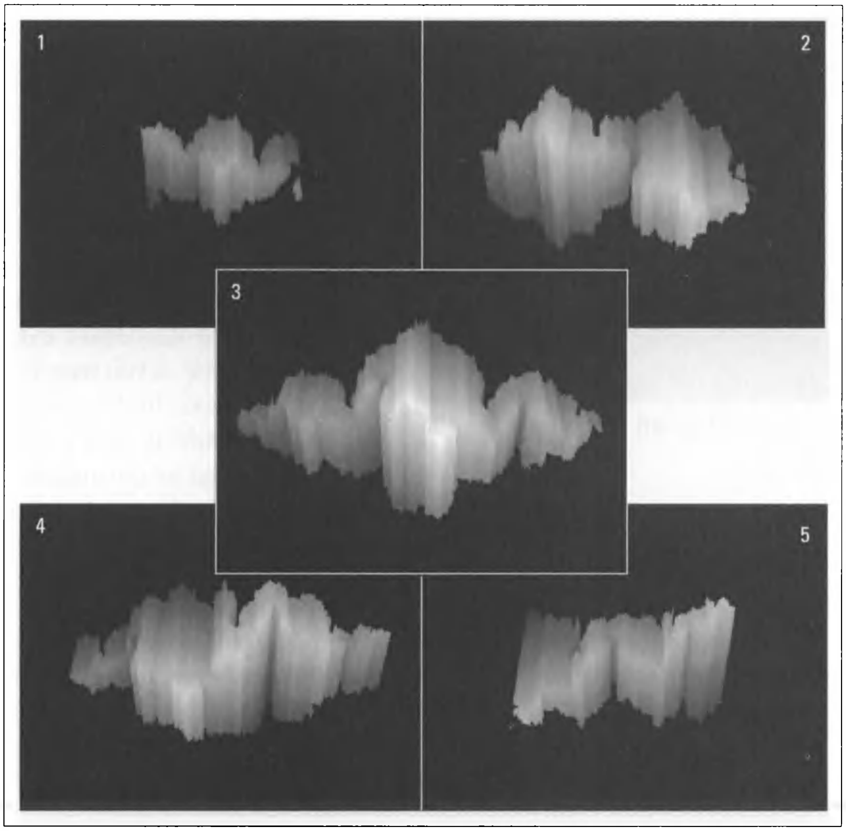


Figure 2-20 3-D slices of a 4-D Julia set printed on a 2-D page

One way to visualize a four-dimensional object is to think of the fourth dimension as time. You could imagine that the hypercomplex Julia fractals shown in Figure 2-20 are snapshots of a single mutating fractal taken at different times. Our imaginary flatlander could use the same trick to visualize an apple. We could create a "mutating apple slice" animation for our flat friend by filming a sequence of horizontal slices a frame at a time, moving from the bottom to the top of the apple. We could then project the movie onto the flatlander's plane. He would see a circular apple slice grow in size as the plane moved to the fat part of the apple, then become smaller and smaller until it vanished entirely. But because you and I know that the apple is "really" three-dimensional, this approach is not too satisfying. Fortunately, you can just create and enjoy 3-D "slices" of hypercomplex and quaternion fractals without solving this knotty philosophical problem, or for that matter even really understanding what these four-dimensional numbers "really" are! You'll learn how to do this with the Fractint program later in this book.

Newton's Method-Escape to Finite Attractor

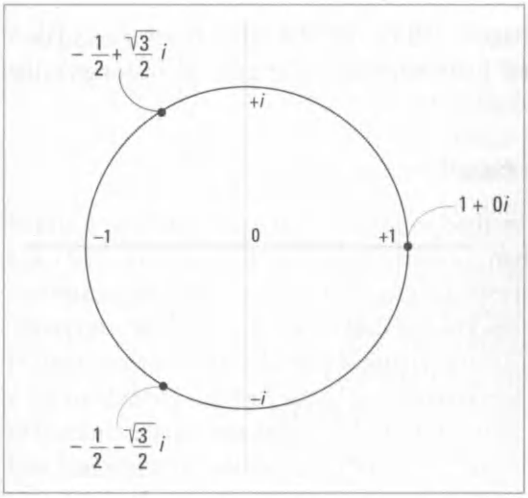
After this brief excursion to higher dimensions, we return to the more familiar world of two-dimensional fractals. The escape-time method of generating fractals we have discussed so far might be called "escape to infinity." The test for when an orbit has escaped (strayed outside a circle of radius 2) is really a test for escaping to infinity. In the case of the Mandelbrot and Julia orbit formulas, once the orbit value gets outside that circle, if you were to continue to calculate the orbit it would spiral outward forever. In this case we say that "infinity is an attractor" for the orbit. It is as if infinity were a magnet trying to attract the Mandelbrot orbit values to itself. And we can imagine that the orbit test point is trying to keep the orbit values in check.

Escape to a Finite Attractor

A similar kind of fractal image is generated by measuring the escape time to a finite value rather than infinity. One example of this creates fractals using what is called Newton's method. (Newton, as you probably recall, was a famous physicist who invented—that is, discovered—a great many truths about moving objects and gravity. He also discovered some clever math techniques.) For example, every time you press the square root button on a calculator, you are using Newton's method. Newton's method is a way of doing a calculation by beginning with a guess for the answer, and repeatedly applying a formula that transforms the guess into a better guess. The series of answers so generated converges rapidly to the correct answer.

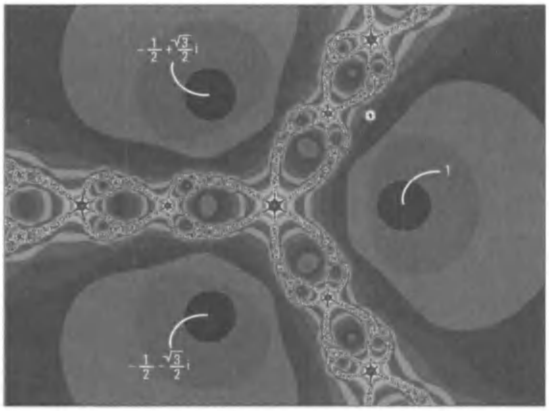
Consider the problem of finding the cube root of 1. This is the same problem as finding the solution to the equation $z^3 - 1 = 0$. The solutions to this equation are the numbers that when multiplied by themselves two times ($z \times z \times z$) give 1 as an answer. You might think that this is a silly problem, because the answer is clearly the number 1, because $1^3 = 1$. What makes the problem interesting is that when complex numbers are considered (the same kind of numbers we just discussed in connection with the Mandelbrot calculation), there are actually *three* solutions to the equation. These three answers are three equally spaced points on a circle of radius 1. They are the complex numbers $1 + i0$, $-1/2 + i\,3/2$, $-1/2 - i\,3/2$

Figure 2-21 shows the three cube roots of 1 distributed on the unit circle in the complex plane, and what happens to several initial guesses when fed into the Newton's method formula.



$-\frac{1}{2}+\frac{\sqrt{3}}{2}i$
$+i$
$1+0i$
-1
0
$+1$
$-i$
$-\frac{1}{2}-\frac{\sqrt{3}}{2}i$

Figure 2-21 The three complex cube roots of 1



$-\frac{1}{2}+\frac{\sqrt{3}}{2}i$
1
$-\frac{1}{2}-\frac{\sqrt{3}}{2}i$

Figure 2-22 Newton's method fractal for the cube root of 1

The Newton's method approach is very similar to the Mandelbrot set calculation. The pixels on the screen are mapped to complex numbers in the same way. For each complex number z_{pixel} corresponding to a pixel, an orbit sequence $z_0, z_1, z_2,\ldots,z_n,\ldots$ is generated. This time the orbit sequence is generated by a slightly more complicated formula, $z_{n+1} = (2z^3 + 1)/3z^2$. But the main difference is that with Newton's method the criterion for "escape" is different. For the Mandelbrot set, escaping meant that the orbit got outside a circle of radius 2 centered on the origin. The orbits that got too close to the "magnet" at infinity were attracted to it. But in the case of Newton's method, there are three magnets, one located at each of the cube roots of 1 around the unit circle. Orbits escape (or perhaps we should say they die) when they are irreversibly attracted to these magnets. Each test-point pixel is colored according to the magnet that captures its orbit.

But what happens when the test point guess is *between* two of the three possible attracting values? The answer is chaos! Areas colored according to the ultimate destination of the orbit become intertwined in an infinitely complex pattern, as Figure 2-22 reveals.

Newton's method is an example of where fractals turn up in situations that engineers want to avoid. That square root button on your calculator has a purpose—to find the square root. The first guess your calculator makes before applying Newton's method is designed to be close enough to the final answer so that the algorithm will work effectively to find the square root. If the algorithm doesn't work for bad initial guesses, then it is the job of the calculator designer to avoid those values. The designer will be out of a job if he or she builds a calculator where an "ill-behaved" initial guess is used and the calculator gives the wrong answer.

Generating a Newton Fractal

Here is how to use Newton's method to generate a fractal. Start with a grid of complex numbers that more than covers the unit circle and our three cube roots of one. The corner values might extend from –2 to 2 in both the x and y direction. Assign colors to the three answers. Fractint uses dark blue, light blue, and green. Each number, z_{pixel} in the grid is used as an initial guess for the Newton's method calculation. Set $z_0 = z_{pixel}$ and successively apply the Newton formula to get a sequence $z_0, z_1, z_2, \ldots$. Each time the formula is iterated, the orbit is checked to see if it has come near one of the roots. If it does, the calculation is finished, and z_{pixel} is assigned the color of the root that captured it. The areas near the three roots end up being solidly colored with the color for that root. In between the roots, the three colors twist together in an intricate braided pattern. These solid areas are called *basins of attraction*, because they show all the starting points that end up converging to a particular attractor. Figure 2-22 shows this intriguing fractal, which might be said to be based on the applied mathematician's nightmare—the indecision of Newton's method!

Chaotic Orbits and the Lorenz Attractor

The discussion of escape-time fractals introduced the idea of an orbit as a series of points that can be imagined to be the path of a flying object. The only concern for the orbit was the time required to escape outside some radius, or the time required to be captured by an attractor (that is, the number of iterations required). The orbit itself was not the main concern, but was simply a step in the calculation of a color of a single point. However, orbits can be interesting in themselves.

The idea of plotting orbits from the equations describing dynamic systems is as old as physics itself. One of the first triumphs of theoretical physics was the demonstration that the elliptic orbit of a small moon around a large planet is a consequence of the inverse square law of gravitation. The problem of determining the orbits of two objects revolving around each other is known as the "two-body problem." It has a simple and elegant solution. But adding a third body to the dynamic system greatly complicates the orbits. Three-body orbits can be complex beyond imagination.

Why is it, then, that every high school science student has the idea that planetary orbits are ellipses, when there are, to make a slight understatement, more than two objects in the universe? *No* orbit in the physical world is exactly an ellipse. If the three-body problem has a complicated solution, how about the trillion-body problem, the one that exists today in our universe!

There is, of course, a perfectly reasonable answer to this question. An ellipse is a simple geometric shape that has simple mathematical properties that make it very suitable for computational purposes, not to mention educational purposes. In science and engineering, careful simplifications and approximations can make intractable problems manageable, and they are a very important tool in the engineering tool kit.

Yet this eminently reasonable answer is unsatisfying. This propensity to imagine orbits in the simplest possible geometric terms is probably yet another manifestation of a deep cultural bias toward a classically geometric way of imagining the world. What do we find when we abandon the simple beauty of the ellipse and contemplate chaotic orbits—which is to say, virtually every *real* orbit? Fractals, of course!

Before launching into an example of a chaotic orbit, let us review a few properties of the well-behaved orbits of classical mechanics. The elliptic orbit is periodic. That is, the orbiting object describes a single path over and over. Alternatively, under different conditions the orbit might be a parabola or a hyperbola, in which case the orbit is not periodic, but the object traverses the orbit exactly once. In all of these cases, the orbit is a well-defined smooth curve.

In late 1963, Edward Lorenz published a paper on deterministic chaos that included some plots of an unusual orbit. Like the Mandelbrot set, his "monster curve" had a very simple mathematical description. But the behavior of this orbit, which we will refer to as the Lorenz attractor, is far from simple.

The plot of the Lorenz attractor orbit consists of two connected spirals, in two different planes at an angle to each other (see Figure 2-23). The orbit path would swirl around inside one of these spiral areas, and then at random intervals it would switch allegiances to the other, and so on back and forth. This orbit has some bizarre properties. It is bounded, like the ellipse, and contained forever within a delimited region of space. But unlike the ellipse, the Lorenz orbit is not periodic; in fact, it never crosses itself or repeats. Its path is, therefore, an infinitely long thread wound around in a finite space. The combination of these three factors—bounded, infinitely long, never crossing itself or repeating—implies a complex interweaving of arbitrarily close near misses of different strands of the orbits like an air traffic controller's worst nightmare! From all that we have discussed so far, you will not be surprised to learn that such an orbit is a fractal. Figure 2-23 is a plot of the first few thousand or so turns of this chaotic orbit. You

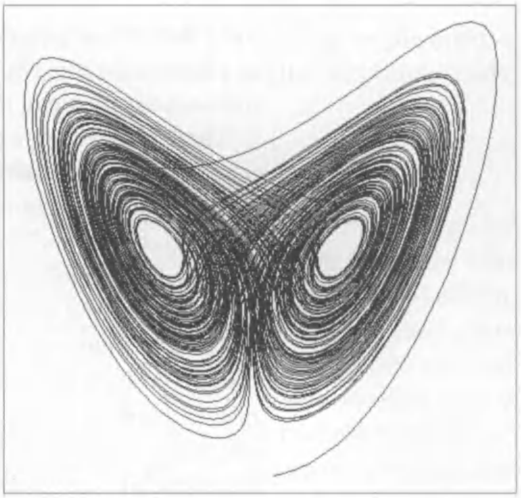


Figure 2-23 The Lorenz attractor

can generate the Lorenz attractor in stereo 3-D using Fractint and even have it generate tones as it's being made.

Gaskets and Ferns—Iterated Function Systems

The essence of a fractal is to have detail at all scales, including the most extreme magnifications. One way to achieve this characteristic is through self-similarity. An object is self-similar if small pieces of itself are identically shaped versions of the complete object, only on a smaller scale. One method of generating fractals is to directly exploit this idea. A fractal can be defined by exactly specifying the relationship between itself and its self-similar parts.

Michael Barnsley has developed this approach and named it Iterated Function Systems, or IFS for short. An endless variety of fractals can be created in this way, some of them eerily lifelike. Fractint and Fdesign can create a variety of bushes, trees, and ferns using the IFS fractal type.

The Sierpinski Gasket

The Sierpinski (pronounced "sear-pin-ski") gasket is a fractal that looks as if it is made of Swiss cheese because it has so many holes. It's called a gasket because it seems to offer the structure you might find in a gasket—lots of passages surrounding each other.

The Sierpinski gasket can be exactly specified by stating the rule governing its self-similarity: it's a geometric object built within a triangle, with the property that each of the three subtriangles formed from one of its corners and the midpoints of the adjacent two sides is an exact self-similar replica of the whole triangle.

Another way to define this fractal is to use what are called affine maps. An *affine map* is a transformation of an object that preserves its shape. The transformation can rotate it, move it, enlarge it, or shrink it, but it must not distort the shape of the object. Therefore, you must be sure your transformations do the same operation to each point in the same way. Such a map is said to be *contractive* if it always shrinks objects. The notion of a contractive affine transformation is a formal way of saying "self-similar." In other words, if there is a contractive affine map between an object and itself, then the object contains a miniature image of itself and is self-similar.

We can easily describe the Sierpinski gasket with three affine maps. Draw the Sierpinski gasket on a graph, so that two of the sides are nestled against the x- and y- axes. The corners of the triangle are the points (0,0), (1,0), and (0,1). Here are three affine maps defining this Sierpinski gasket:

1. Map every point (x,y) to the point $(x/2,y/2)$. This maps the whole triangle to the lower left triangle by shrinking the scale by a factor of a half.

2. Map every point (x,y) to the point $(x/2,y/2 + 1/2)$. This maps the whole triangle to the upper left corner subtriangle by shrinking the scale by a factor of a half and shifting up half a unit.

3. Map every point (x,y) to the point $(x/2 + 1/2,y/2)$. This maps the whole triangle to the lower right subtriangle by shrinking the scale by a factor of a half and shifting to the right half a unit.

These three affine maps are shown in Figure 2-24.

In this particular example, the transformations are particularly simple because no rotation was involved, only shifting and shrinking. The key insight into the relationship between affine transformations and the Sierpinski gasket is to notice that there are four possible triangles with sides equal to half the sides of the original. The missing triangle is the center one, formed from the midpoints of the three sides. Why is there no transformation mapping the whole triangle to the center? Because there is nothing in the center—that is the "hole"! If the fourth affine transformation were added, mapping the whole triangle to the middle, the result would be rather boring—simply a filled-in triangle! Leaving out the center is what creates the "Swiss cheese" effect with the missing centers of the triangles.

Barnsley suggests a method of generating the fractal from these affine transformations that he calls the "chaos game." Start with any arbitrary point whatsoever. Pick one of the transformations at random and apply it to the point,

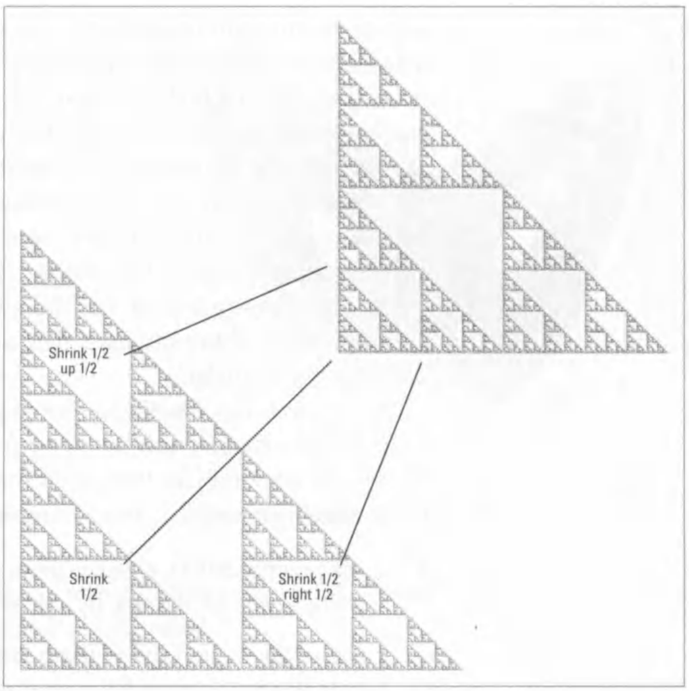


Figure 2-24 Sierpinski gasket

plotting the result. Continue by applying a new randomly chosen transformation each time to the last point, again plotting the result. But as the process is repeated, the points generated will produce the shape of the Sierpinski gasket. The name Iterated Function Systems for this kind of fractal comes from the repeated, or iterated, application of these affine maps, or function systems.

The Sierpinski gasket is used here to illustrate IFS fractals, but it can also be generated in two other ways by Fractint: by using L-systems (see Chapter 5, *Fractint Reference*) and by using the escape-time methods we have discussed. The Sierpinski gasket holds the record for the number of different ways that Fractint can create it.

A Fractal Fern

The Sierpinski gasket is a very unnatural-looking object, but it is just one of an endless variety of images possible with the IFS approach. Another fractal that has become almost a trademark of Barnsley's work is the fractal fern.

Many plants have several levels of self-similarity because of their branching structure. Some kinds of ferns have wide fronds at the base and narrower fronds

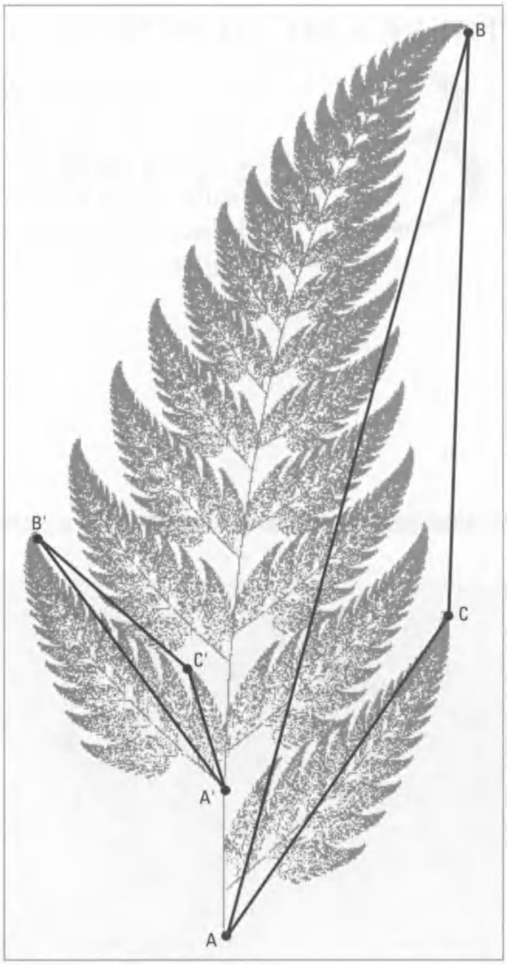


Figure 2-25 The self-similarity of a fractal fern

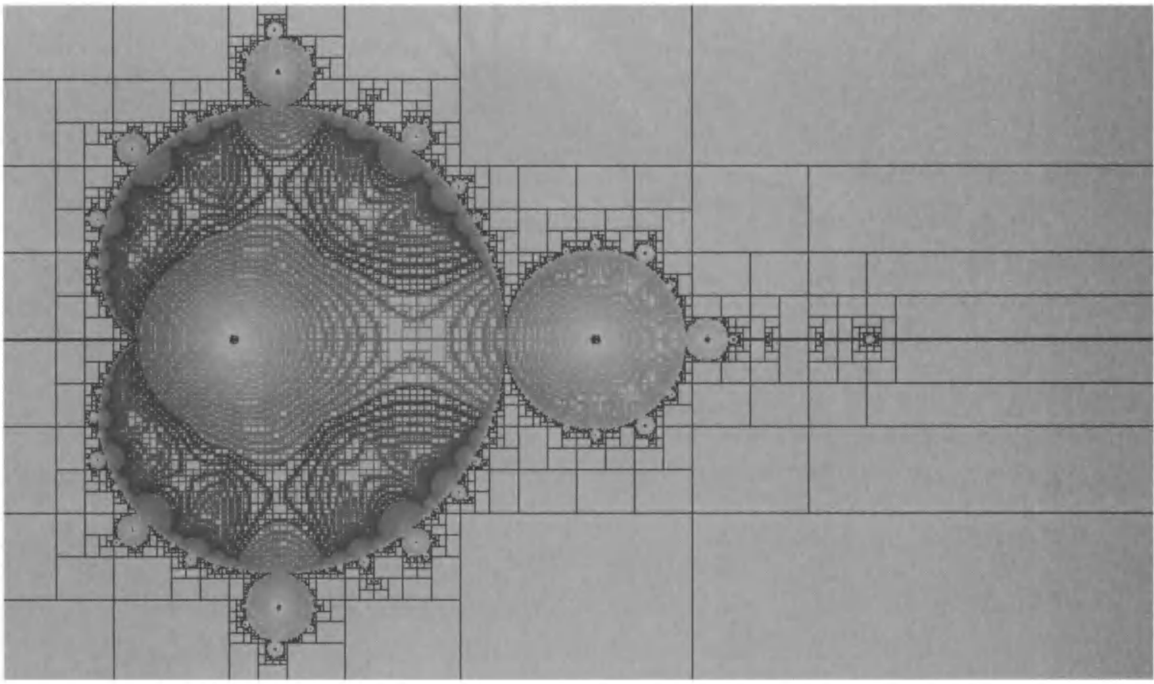
in the center, tapering to a pointed tip. If you broke off the very bottom fronds, you would end up with a smaller but still similar fern. Four iterated functions can be used to define a very natural-looking fern. Two functions define the self-similarity between the left and right fronds with the whole. One function defines the stem. Finally, another function defines the relationship between the whole fern and the fern less the bottom fronds. Figure 2-25 shows these self-similarities. As with the Sierpinski gasket, probabilities are assigned to these functions, and their repeated application seeded with an arbitrary starting point generates the image, just as we described earlier for the Sierpinski gasket.

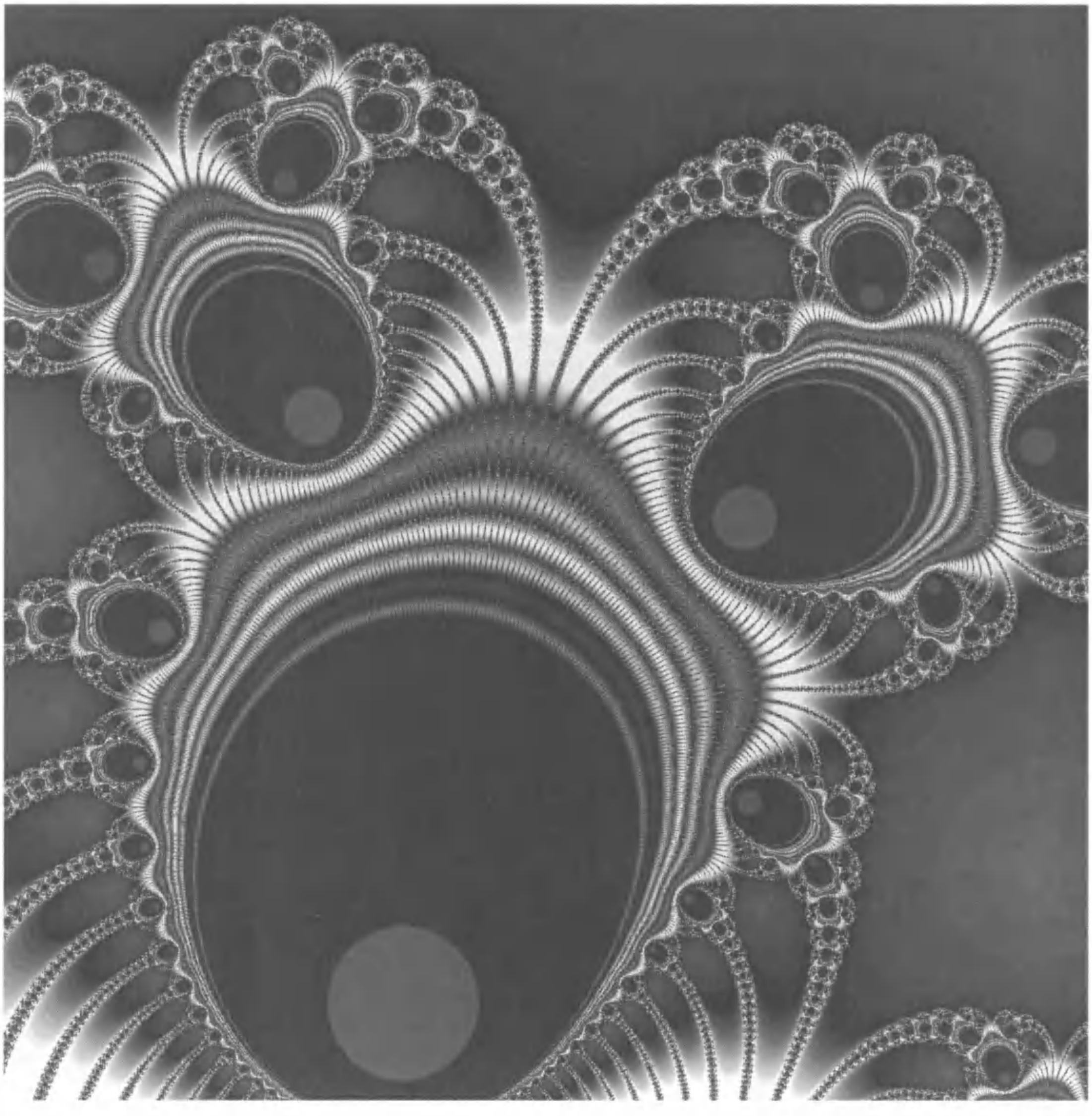
THERE'S MUCH MORE

No introduction to fractals can completely cover the subject. This fractal primer was designed to give you a taste and feel of what fractals are all about, as well as a slight touch of the mathematics behind them. But because this is really more a book about exploring and creating fractals, the next chapter begins that exploration with a guided tour of the Fractint program that comes with this book.

CHAPTER 3

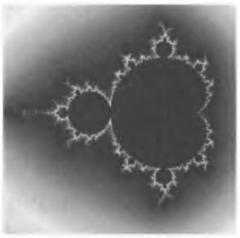
FRACTINT TUTORIAL





CHAPTER 3

FRACTINT TUTORIAL



Are you ready for a wild ride into the mysterious world of fractals? You have come to the right place. This chapter is a guided tour of the Fractint program. For a complete reference to the keystrokes and commands, see Chapter 5, *Fractint Reference.*

Fractint generates fractals based on any of its 95 different built-in formulas. It can save and retrieve fractals in CompuServe's Graphics Interchange Format (GIF) format. Fractint has additional capabilities for generating 3-D transformations of fractals, making stereo red/blue images, doing color-cycle animation, changing color palettes, letting you experiment with your own fractal formulas (with no programming needed), and much more. In this chapter you will learn how to access some of these features and how to fine-tune the way the program operates.

This chapter will take you through a hands-on demonstration of Fractint's most basic functions. It is for readers who have never used Fractint before, but it will also show you nooks and crannies of the program that even experienced users may not have discovered. Don't feel restricted by our tour, however. You may want to explore on your own at various points along the way. But do come back. Fractint is the kind of program that grows on you because it has more possibilities than you can absorb all at once.

UP AND RUNNING

We assume you have read Chapter 1, *Installation*, earlier in the book and that Fractint is installed and ready to run in a directory that is included in your DOS path. If not, go back and read the installation instructions, make sure you have

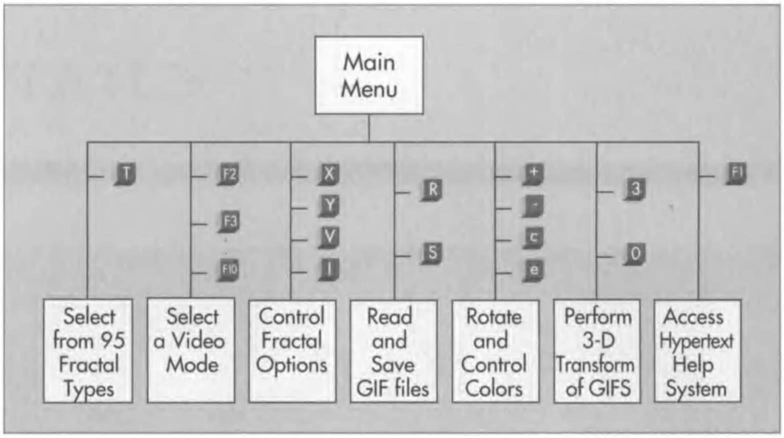

Main Menu
T
F2
F3
F10
X
Y
V
I
R
S
+
-
c
e
3
0
F1
Select from 95 Fractal Types
Select a Video Mode
Control Fractal Options
Read and Save GIF files
Rotate and Control Colors
Perform 3-D Transform of GIFS
Access Hypertext Help System

Figure 3-1 The Fractint road map

the correct files in your Fractal Creations directory, and come back here as soon as you have Fractint running.

Fractint has a keystroke record and playback capability which you can use to demonstrate anything the program can do. When in playback mode, Fractint appears to be under the control of an invisible guide entering keystrokes at the keyboard. For a demo of many of Fractint's features using the keystroke playback mode, change to your Fractint directory and run DEMO.BAT. Assuming that you have placed your files in \FRACTINT, type

```
cd \FRACTINT (ENTER)
DEMO (ENTER)
```

This demo will show you some of Fractint's fractal types, how to control Fractint with menus, and how to create special effects using color cycling. You can exit the demo mode at any time by pressing (ESC). Fractint will continue to run, but will now respond to your keystrokes rather than the keystrokes stored in the demo file.

A FRACTINT GUIDED TOUR

To make your tour easier, a road map is shown in Figure 3-1. This figure is a simplified flowchart of the different functions of Fractint. As we go through the tour, we will traverse the routes shown on the road map.

Here are a few conventions about how we will describe what you type to invoke the various commands. When we want you to type in something literally,

Key	Action
F1	Go to the main help index.
PAGE DOWN/PAGE UP	Go to the next/previous page.
BACKSPACE	Go to the previous topic.
ESC	Exit help mode.
ENTER	Select (go to) highlighted hot-link.
TAB/SHIFT-TAB	Move to the next/previous hot-link.
↑ ↓ ← →	Move to a hot-link.
HOME/END	Move to the first/last hot-link.

Table 3-1 Help navigation keys

we'll show it in monospace and bold. For example, when you see "type in the file name ALTERN.MAP," you should type: **`ALTERN.MAP`**.

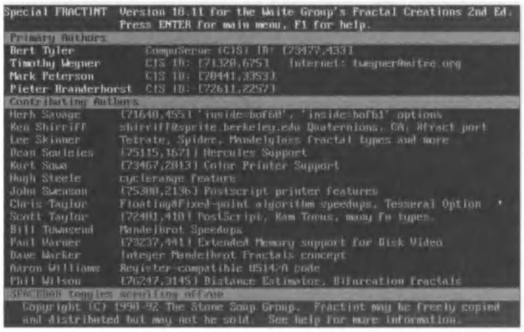
Some keys or other items will be referred to using key caps characters, as when we write "press ESC to return to the MAIN MENU." You should press the ESC key.

We will surround variable names that you should *not* type in literally with the "<" and ">" characters. When we write: "Type in at the DOS prompt FRACTINT SAVENAME=<FILENAME>," for example, you supply a file name to replace <FILENAME>, so you would actually type in something like: **`FRACTINT SAVENAME=MYFRACTAL.GIF`**.

Quitting Fractint

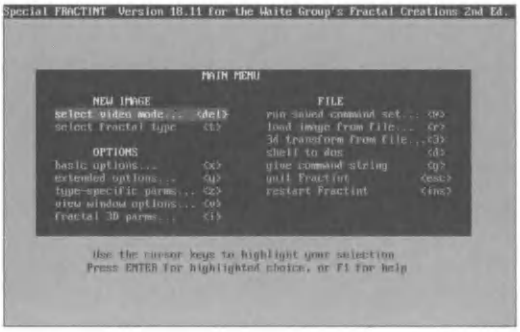
Before we start, let's talk about the two most important commands in Fractint, the Quit command and the Help command. The Quit command key is ESC, which is used in every context to back out of whatever mode you are in and move toward the MAIN MENU. In fact, repeatedly pressing ESC will get you to the EXIT FROM FRACTINT? (Y/N) prompt, which then requires pressing only Y to return to the DOS prompt.

The Help function is accessed by pressing the F1 key from any place in the program. The first Help screen you see is context-sensitive; it depends on where you were in the program when you pressed F1. Pressing F1 a second time will take you to the MAIN HELP INDEX, from which you can access all the Help screens. Pressing ESC exits the help mode. Table 3-1 summarizes the keystrokes used to navigate through the help system.



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.
                  Press ENTER for main menu, F1 for help.
 Primary Authors
 Bert Tyler             CompuServe (CIS) ID: [73477,433]
 Timothy Wegner         CIS ID: [71320,675]   Internet: twegner@mitre.org
 Mark Peterson          CIS ID: [70441,3353]
 Pieter Branderhorst    CIS ID: [72611,2257]
 Contributing Authors
 Herb Savage        [71640,455] 'inside=bof60', 'inside=bof61' options
 Ken Shirriff       shirriff@sprite.berkeley.edu Quaternions, CA, Xfract port
 Lee Skinner        Tetrate, Spider, Mandelglass fractal types and more
 Dean Souleles      [75115,1671] Hercules Support
 Kurt Sowa          [73467,2013] Color Printer Support
 Hugh Steele        cyclerange feature
 John Swenson       [75300,2136] Postscript printer features
 Chris Taylor       Floating&Fixed-point algorithm speedups, Tesseral Option
 Scott Taylor       [72401,410] PostScript, Kam Torus, many fn types.
 Bill Townsend      Mandelbrot Speedups
 Paul Varner        [73237,441] Extended Memory support for Disk Video
 Dave Warker        Integer Mandelbrot Fractals concept
 Aaron Williams     Register-compatible 8514/A code
 Phil Wilson        [76247,3145] Distance Estimator, Bifurcation fractals
 SPACEBAR toggles scrolling off/on
   Copyright (C) 1990-92 The Stone Soup Group.  Fractint may be freely copied
   and distributed but may not be sold.  See help for more information.
```

Figure 3-2 The famous Fractint credits screen



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

                                MAIN MENU

           NEW IMAGE                              FILE
      select video mode...   <del>     run saved command set...  <@>
      select fractal type     <t>      load image from file...   <r>
                                       3d transform from file...<3>
           OPTIONS                     shell to dos              <d>
      basic options...        <x>      give command string       <g>
      extended options...     <y>      quit Fractint           <esc>
      type-specific parms...  <z>      restart Fractint        <ins>
      view window options...  <v>
      fractal 3D parms...     <i>

            Use the cursor keys to highlight your selection
           Press ENTER for highlighted choice, or F1 for help
```

Figure 3-3 The Fractint Main Menu

The Credits Screen

Fractint's opening screen shown in Figure 3-2 is decidedly unconventional and comes closer than anything else to symbolizing the participatory nature of Fractint. We "Stone Soupers" (the creators of Fractint) considered giving Fractint a more fashionable face for this book, but after a little thought we realized that the opening screen has become indispensable. To our eyes, it's beautiful!

When you fire up Fractint you are presented with a scrolling list of the names of people who have made the "stone soup" tasty by contributing the odds and ends from their programming "cupboards." (Notice that the list shows the Stone Soupers' CompuServe Information Service (CIS) numbers, so you can contact many of us by electronic mail.) Fractint is truly a community project driven by the excitement and imagination of an international network of kindred souls. These people have two things in common: their fascination with fractals and their desire to share their excitement with others.

Once you have used Fractint more than a few times, you will probably get in the habit of immediately pressing (ENTER) to bypass the credit screen and move straight to the MAIN MENU when you start Fractint.

The Main Menu

Having paid homage to the legions who have contributed to the program, go ahead and press (ENTER) to move to the MAIN MENU, which should look like Figure 3-3.

Using Fractint Menus

Fractint was originally a command-driven program, which means that various keystrokes caused Fractint to execute different commands. All commands are now also accessible from screens we call "menus," which you can control with the cursor keys. Throughout Fractint, you can select items from the menu by moving the highlight to different menu items using the ↑, ↓, ←, and → keys, and then pressing ENTER to execute the commands you have highlighted. Note that the menus also tell you how to select the menu items by using direct command keystrokes. For example, you can leave Fractint either by selecting QUIT FRACTINT with the arrow keys and pressing ENTER, or by pressing ESC. Note that none of the Fractint menus use the mouse.

Most commands can also be given at Fractint startup time using command-line options. You can, for example, specify which video mode you want Fractint to use by typing the following at the DOS prompt: `fractint video=F3`ENTER. This method can be useful when you have specialized commands to execute and want to simply type them instead of using the menus. You can also execute Fractint with command-line parameters from a batch file.

Fractint Modes

Fractint has five modes, each with its own set of commands. These five modes are the display mode, the color-cycling mode, the orbits window mode, the Julia window mode, and the palette editing mode. Of these five modes, the display mode is the most important, because the main functions of Fractint are accessible from within this mode. Color cycling, the orbits window, and the Julia window are simple but secondary modes, while palette editing is a more advanced function. Each of the five modes has its own set of commands, so it is important for you to be clear at all times which mode is currently active. You can press F1 at any time to get help with the current mode. You do not need to memorize keystrokes because menus and help screens can be easily invoked, although you will find yourself quickly learning the most important keystrokes. At first Fractint will be in the display mode. The other four modes will be described a little later.

The Fractint MAIN MENU is divided into three sections, described in the following paragraphs.

New Image

The NEW IMAGE menu category contains commands that let you select a fractal type and select the desired video mode. Fractint can calculate its images from 95 built-in formulas. But in order to create a new image, Fractint needs to know two things: what video mode you want to display and what fractal type you want to calculate (see Chapter 6, *Fractal Types,* for an explanation of fractal types). The fractal type defaults to the Mandelbrot set. If Fractint is able to detect what exact video hardware your PC has, a suggested video mode will be highlighted in the list of video modes when you invoke the SELECT VIDEO MODE function. (A video mode is the screen resolution and number of colors supported by a particular video adapter.) However, there is no universal default for the video mode; you must choose a mode in order to generate an image. But before you go ahead and choose a mode, let's discuss the other menu headings.

Options

The OPTIONS menu items give you access to a variety of settings and special effects. You won't need these right away, except possibly the VIEW WINDOW option. Because the time it takes to create a fractal is directly proportional to how many screen pixels have to be calculated, the VIEW WINDOW capability allows you to specify a very small image that can be calculated very rapidly. This capability is wonderful for the intrepid fractal explorer, especially one who does not have the world's fastest computer!

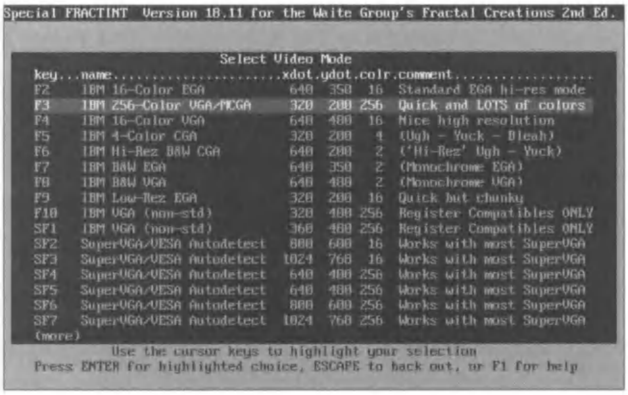
File

The FILE menu category includes the ability to restore to the screen, or read in, previously calculated images that were saved as GIF files. (See Appendix B, *Fractint and GIF Files* for more about the GIF standard.) There are two ways to restore files and duplicate them on-screen: either as they were originally calculated, or by doing a 3-D transformation on the file. There are also useful commands on the FILE menu that let you enter DOS without exiting Fractint (you return to Fractint by typing `exit`), quit Fractint altogether with (ESC), and restart Fractint with (INSERT).

Let's continue this tour through each of the MAIN MENU commands in detail.

Selecting a Video Mode

Using the arrow keys, choose SELECT VIDEO MODE from the NEW IMAGE section of the MAIN MENU, and press (ENTER). You will be presented with a list of video modes, with a choice highlighted, that should look like Figure 3-4. This list shows all the



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

                        Select Video Mode
 key...name....................xdot.ydot.colr.comment..................
 F2    IBM 16-Color EGA          640  350  16  Standard EGA hi-res mode
 F3    IBM 256-Color VGA/MCGA    320  200 256  Quick and LOTS of colors
 F4    IBM 16-Color VGA          640  480  16  Nice high resolution
 F5    IBM 4-Color CGA           320  200   4  (Ugh - Yuck - Bleah)
 F6    IBM Hi-Rez B&W CGA        640  200   2  ('Hi-Rez' Ugh - Yuck)
 F7    IBM B&W EGA               640  350   2  (Monochrome EGA)
 F8    IBM B&W VGA               640  480   2  (Monochrome VGA)
 F9    IBM Low-Rez EGA           320  200  16  Quick but chunky
 F10   IBM VGA (non-std)         320  400 256  Register Compatibles ONLY
 SF1   IBM VGA (non-std)         360  480 256  Register Compatibles ONLY
 SF2   SuperVGA/VESA Autodetect  800  600  16  Works with most SuperVGA
 SF3   SuperVGA/VESA Autodetect 1024  768  16  Works with most SuperVGA
 SF4   SuperVGA/VESA Autodetect  640  400 256  Works with most SuperVGA
 SF5   SuperVGA/VESA Autodetect  640  480 256  Works with most SuperVGA
 SF6   SuperVGA/VESA Autodetect  800  600 256  Works with most SuperVGA
 SF7   SuperVGA/VESA Autodetect 1024  768 256  Works with most SuperVGA
 (more)
            Use the cursor keys to highlight your selection
 Press ENTER for highlighted choice, ESCAPE to back out, or F1 for help
```

Figure 3-4 The Select Video Mode menu

various drivers built into Fractint for a variety of video hardware, along with comments about each video mode. (A video driver is a specialized routine for accessing the features of your video hardware.) The list includes not only standard IBM-compatible modes such as the 320 x 200 VGA 256-color mode, but some highly unusual "tweaked" modes that can squeeze extra resolution out of a plain VGA board.

The F3 option means that pressing function key F3 will directly select that video mode. Each video driver has a key combination associated with it, and pressing the key combination selects that video mode. You can also select different modes by using the arrow keys to move the highlight to the mode you want and then pressing ENTER. Fractint has room for up to 100 different video modes, each with a different key combination. Table 3-2 lists some examples to show how the key-naming scheme works.

Video Mode Label	Equivalent Keystrokes
F2	Press the function key F2
SF2	Press SHIFT and F2 together
CF2	Press CONTROL and F2 together
AF2	Press ALT and F2 together
Alt-1	Press ALT and 1 together
Ctl-1	Press CONTROL and 1 together

Table 3-2 The Fractint key naming scheme

Graphics Adapter	Function Key	Pixels Across	Pixels Down	Colors
CGA	F5	320	200	4
EGA	F2	640	350	16
VGA	F3	320	200	256
Hercules	ALT-G	720	348	2
SVGA	SHIFT-F5	640	480	256

Table 3-3 Video mode choices for different hardware

As an example, the mode labeled SF1 has a resolution of 360 pixels wide and 480 pixels high, with 256 possible colors. This mode results from Fractint directly programming the VGA registers, and it should work on any VGA that is fully register-compatible with the IBM VGA. It is accessed by pressing SHIFT-F1.

For right now, however, our main concern is to find a good video mode for getting started. If you have a CGA, EGA, or VGA adapter, Fractint will have detected your adapter and chosen a mode for you, which you will see highlighted. If you have a VGA, for example, a good mode to use is the one labeled F3 IBM 256-COLOR VGA/MCGA, because it has relatively low resolution for fast results, and because it has more colors—a decided plus. You won't immediately notice the extra colors of a 256-color mode in the default Mandelbrot image that we'll be generating shortly, but once you zoom deep inside a fractal and try cycling the colors, you'll see the value of having more colors.

Once you select a video mode, it will remain current until you change modes. Table 3-3 shows several good initial choices of video mode for different kinds of video hardware. After you have created a really spectacular image, you may want to regenerate it using a higher resolution mode.

If you have a super VGA board, try some of the SuperVGA/VESA Autodetect modes. It's a good idea to get out your video board documentation, find a chart of video modes, and see which modes your board supports. For this tour, though, any of the modes in Table 3-3 will be just fine.

Generating a Fractal

Go ahead and select a video mode. If Fractint's highlighted choice looks reasonable for your computer hardware, just press ENTER to select it. If you don't

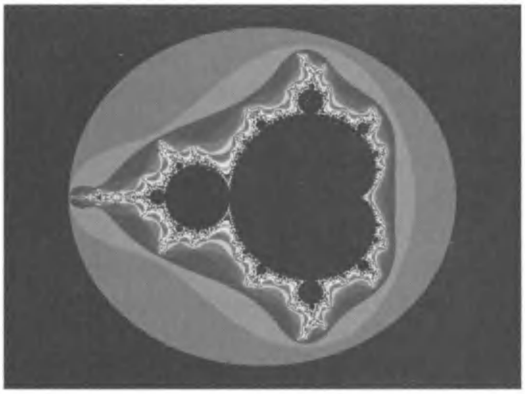


Figure 3-5 Your first Mandelbrot set

like the mode Fractint chose and you have a VGA board, use F3; for an EGA, try F2; and for a CGA, try F5.

Now for the trigger—press (ENTER)! Pressing (ENTER) to select a video mode begins generating a fractal image. The default image for Fractint is type mandel, so you should now see a fractal being generated on your screen—the famous Mandelbrot set. Your screen should look like Figure 3-5. Congratulations, you have created your first fractal!

If you didn't get Fractint to display the Mandelbrot set, you probably selected a video mode that is not supported by your hardware. This can be disconcerting, but it does no harm. Even if your screen is black, press (ESC) to get back to the MAIN MENU. (If the computer "locks up"—freezes the screen and keyboard—you could have to reboot, but this is unlikely.) Try again, this time selecting an appropriate video mode. If you have a color system, try (F5). That is the old IBM 4-color CGA mode, which should work on most systems.

EGA and VGA Colors

Once you get things working, try experimenting with other modes. If you have a super VGA and you started off with (F3), try the EGA (F2) mode. This will give you a feel for the way resolution affects speed. Notice that the default Mandelbrot image (the one you get when you first start Fractint and select a video mode) looks almost the same with 16 colors as it does with 256 colors. In fact, the authors have had phone calls from people complaining that the 256-color modes didn't appear to work! The reason is that *the colors correspond to the number of iterations of the formula used to calculate the Mandelbrot set.* The outer colored area

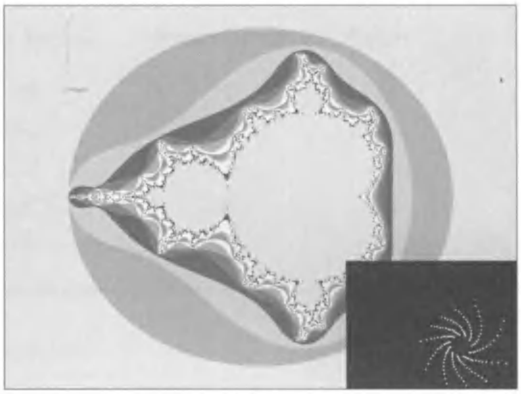


Figure 3-6 The orbits window

corresponds to color 1, the next stripe to color 2, and so forth. Mathematically it turns out that the vast majority of visible pixels in the Mandelbrot image have colors with values less than 16—which is the number of colors on the EGA, too. But after you begin zooming in, the 256-color images will begin looking very different.

To allow quick evaluation, Fractint plots its images, such as the Mandelbrot set, using multiple passes. First it draws the entire fractal image using large chunky blocks, then it goes over the image again and subdivides the blocks into smaller blocks. The number of passes depends on the video mode; the higher the resolution, the more passes. For example, 320 x 200 modes have two passes, while 640 x480 modes have three. What's nice about this approach is that you don't have to wait for the image to be completed before continuing your explorations. You can generally tell what the fractal will look like soon after the coarse pixels of the first pass are colored. If you don't like the way the fractal is developing, you can press (ESC) to return to the MAIN MENU and change some of the settings.

The Orbits Window

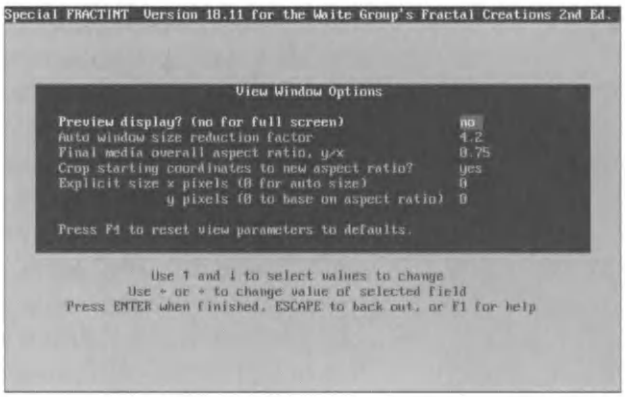
Try the following experiment: After your Mandelbrot image is complete, press the (O) (the letter O) key. You will see a black window appear in the lower right corner of the screen, and a cross-shaped cursor appear in the center of the screen. As you move the cursor with the mouse or the arrow keys, you will see a spiral-like pattern of points appear in the window, as shown in Figure 3-6. This pattern makes visible the orbit values generated in the Mandelbrot calculation. (See Chapter 2, *Fractals: A Primer* for an explanation of orbits.) Pressing ESC turns off this feature and returns to the normal display mode.

The Mandelbrot image is calculated by generating a sequence of points and testing whether they have escaped a circle of radius 2. The O window plots this sequence of points on the screen, and colors them according to which iteration of the Mandelbrot formula they belong. Notice that the orbits have the most structure when the "lake" points are being plotted and that the orbit path does not quickly escape to large values. The orbits get more complex as you move the cursor toward the "lake" shoreline. (Some people feel the orbit plots are more interesting than the fractal images themselves—and that is why in this latest version of Fractint we have added this new orbit window.)

When in the orbit mode, there are special keys that enhance its effect. The C key draws circles around each point, with radius inversely proportional to the iteration number, so the first orbit points have the largest circle. Because this key is a toggle, pressing C again turns off circles and returns to plotting individual points. The L key toggles a line mode, in which the orbit points are connected by lines. You can have the circle mode and line mode on at the same time! Pressing N toggles on and off the display of the coordinates of the point where the cross-hair cursor points.

An Orbit Window Trick

Here's a trick that will let you fill the whole screen with just the orbit image. Press V to open the VIEW WINDOW OPTIONS menu. (If you were in orbits mode, pressing any display mode command key such as V ends the orbits mode.) You will see the view windows screen, as shown in Figure 3-7. In the first field at the top of the menu, type Y at the PREVIEW DISPLAY? (NO FOR FULL SCREEN) prompt, and press ENTER. A small Mandelbrot image will then be generated in the center of your screen. (View windows is really handy for quickly getting a feel for the appearance of a fractal, because the little images generate very quickly.) Now turn on the orbits mode by pressing O. The baby Mandelbrot image will jump to the upper left corner, and the orbits display will fill the screen! You can hide the fractal image and allow the orbit image to fill the whole screen by using the H toggle. Pressing H again restores the Mandelbrot image. Don't forget to try C and L to see the orbits represented with circles or lines. When you are done with the orbits, press V again to turn off view windows by typing N at the first prompt and pressing ENTER.



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

                         View Window Options

     Preview display? (no for full screen)                no
     Auto window size reduction factor                    4.2
     Final media overall aspect ratio, y/x                0.75
     Crop starting coordinates to new aspect ratio?       yes
     Explicit size x pixels (0 for auto size)             0
                   y pixels (0 to base on aspect ratio)   0

     Press F4 to reset view parameters to defaults.

                 Use ↑ and ↓ to select values to change
               Use ← or → to change value of selected field
         Press ENTER when finished, ESCAPE to back out, or F1 for help
```

Figure 3-7 The view window options menu

Zooming In

Now that you have your first Mandelbrot image displayed, and you've played with the orbit display, what the heck can you do with it? Fractals are full of interesting details that unfold as you expand them. The "zoom" function of Fractint lets you dive inside a fractal on the screen and behold its inner beauty. This is also the main tool that enables you to explore fractals at different scales.

Pressing the (PAGE UP) key or clicking the left mouse button creates a dashed rectangle—called the zoom box—around the outer edge of the screen. Repeatedly pressing (PAGE UP) or holding the left button down while moving the mouse away from you shrinks the zoom box. You can move the zoom box around the screen by using either the arrow keys or the mouse. Moving the mouse with no buttons pressed moves the zoom box within the screen's *x-y* plane.

The opposite of (PAGE UP), the (PAGE DOWN) key is used to make the zoom box larger. Repeatedly pressing (PAGE DOWN) will make the zoom box disappear. To do the same thing with the mouse, hold the left button down and pull the mouse toward you. You can use keys and mouse together, too. On most recent machines, you can speed up the movement of the zoom box with the cursor keys by holding down the (CONTROL) key while pressing the cursor keys.

You can even rotate the zoom box! Try (CONTROL)-(+) and (CONTROL)-(–), where "+" and "–" are the gray keys on the numeric keypad. Moving the mouse left or right while holding the right button down performs the same function.

Zoom Box Exploration Technique

The real fun of using Fractint is locating some interesting detail in a fractal, placing the zoom box over it, and then pressing ENTER (or double-clicking the left mouse button) to cause Fractint to calculate and fill the entire screen with the smaller detail that was in the zoom box. The zoom box acts as a magnifier.

You can also zoom out by creating a small zoom box and then pressing CONTROL-ENTER. The effect of this is to zoom out so that the previous image is shrunk to the size of the zoom box and the surrounding area is filled in. The equivalent mouse command is to double-click the right mouse button.

Finding Baby Mandelbrots

Mandelbrot images live inside Mandelbrots. In fact, it is hard to avoid them. In order to try the zooming facility, try to locate some baby Mandelbrot sets in the left-hand spike of the Mandelbrot set. Figure 3-8 shows the results of two zooms which you can duplicate. The image on the left shows the full Mandelbrot set with a zoom box centered on a bulge in the left-hand spike of the Mandelbrot. The second image shows the resulting zoomed image, which contains a miniature copy of the whole Mandelbrot set, and another small zoom box centered on another small bulge in the fractal image. The third image shows the result of this deeper zoom, revealing yet another small Mandelbrot shape. Even a short investigation of the Mandelbrot spike should convince you that there are an infinite number of such baby Mandelbrots.

Now try it yourself. Starting with the full Mandelbrot set, press PAGE UP several times to make a small zoom box. Move the zoom box to the spot shown in Figure 3-8a. When you are satisfied that the zoom box is in the correct position, press ENTER. Your image should look something like the one in Figure 3-8b. Repeat this process, trying to duplicate the zoom box in the middle image to generate the third image, Figure 3-8c.

Increasing the Maximum Iterations

Try zooming in several more times, finding still smaller baby Mandelbrot sets. If you zoom far enough, you will find that the Mandelbrot shape degrades. The shape loses the sharp twists and turns of its convoluted coastline. The Mandelbrot set is

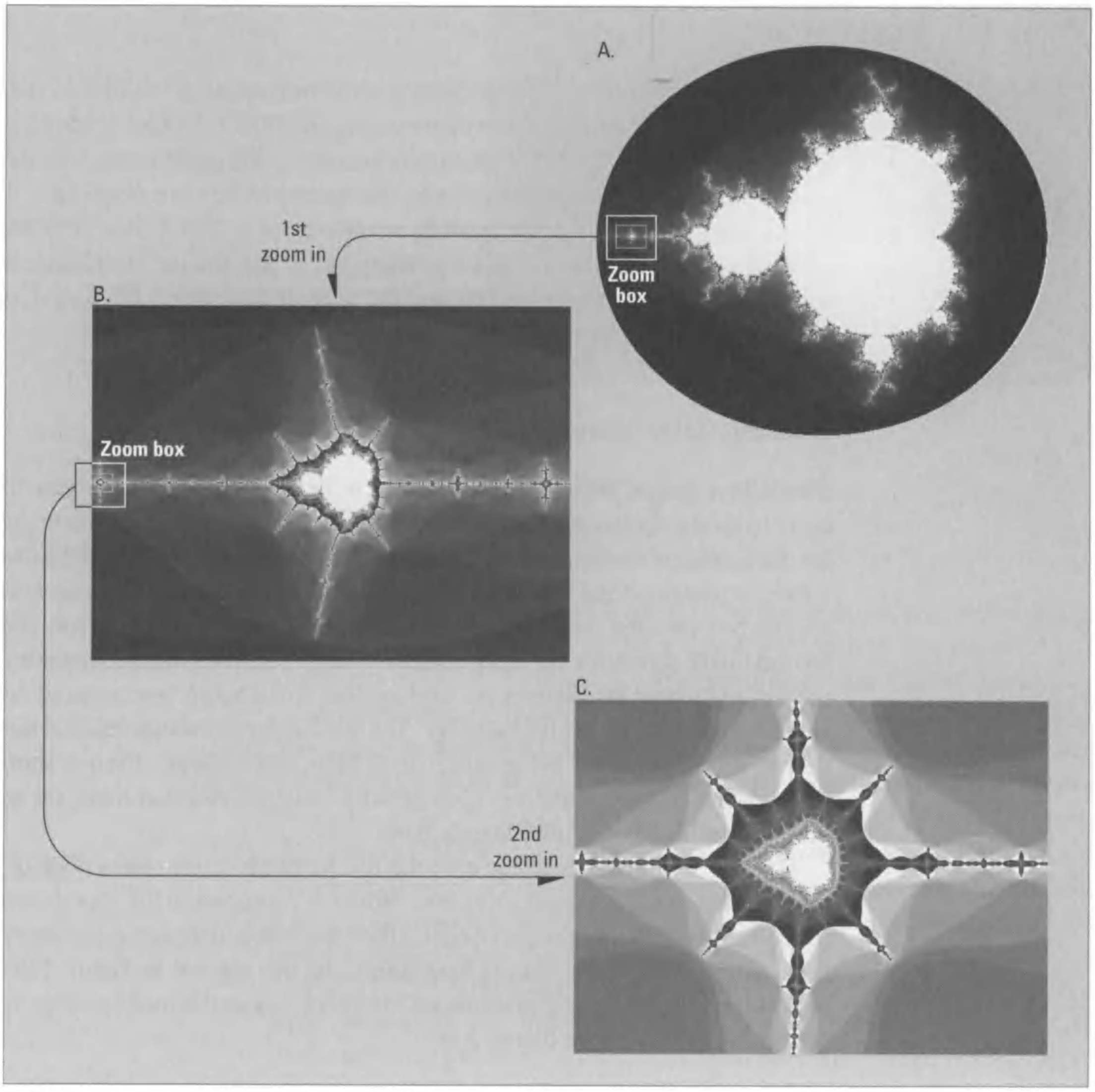


Figure 3-8 Zooming in on baby Mandelbrots

defined to be the set of points *c* whose orbits generated by iterating the formula $z_{n+1} = z_n^2 + c$ *never* escape a circle of radius 2 no matter how many iterations are calculated. (See Chapter 2, *Fractals: A Primer,* for a discussion of escape-time fractals.) Fractint approximates the value of "never" by waiting until some

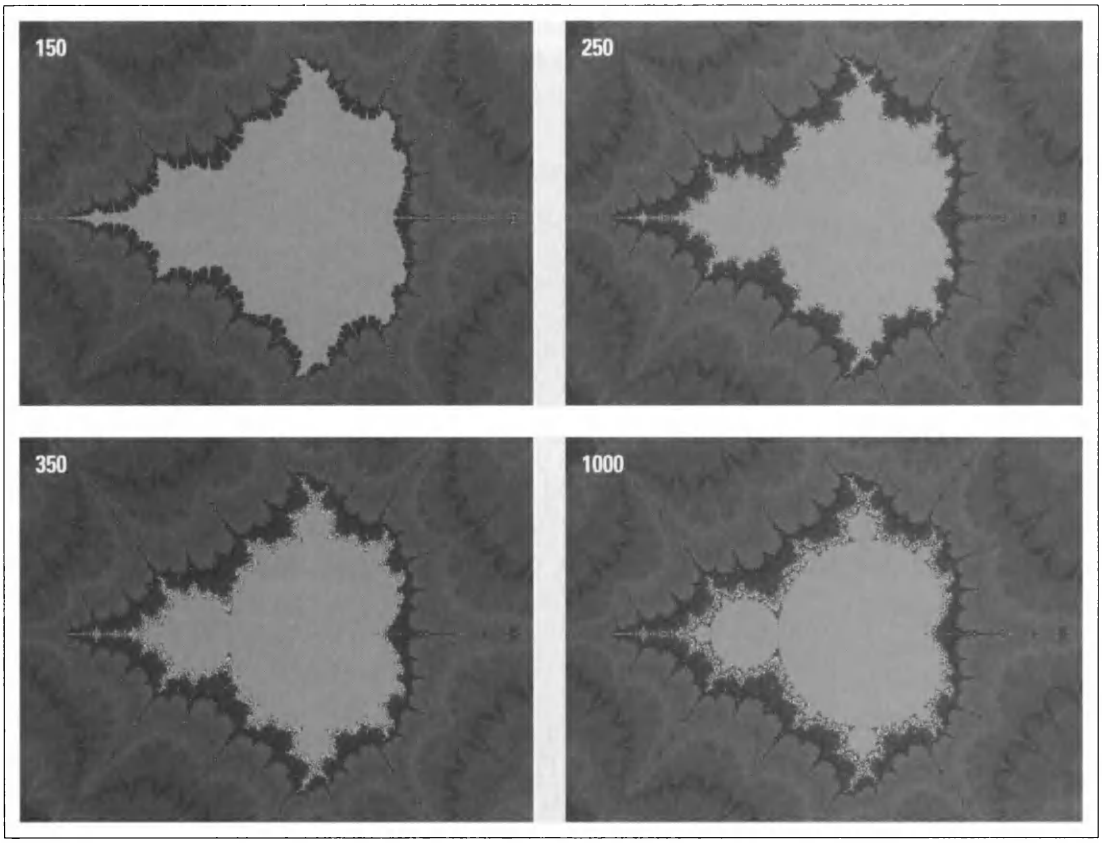


Figure 3-9 Effect of increasing Maximum Iterations

maximum number of iterations is reached, and then assuming that if the orbit hasn't escaped yet, it will never escape. This assumption is not completely accurate, and it is better if a higher maximum iteration cutoff is used as you zoom deeper. The default value in Fractint is 150 iterations, but you can set it as high as 32,768. The price you pay for accuracy is that the calculations will take longer. You can set the maximum iterations by pressing $\boxed{X}$ to access the BASIC OPTIONS screen (or select BASIC OPTIONS <x> under OPTIONS from the MAIN MENU, and filling in a value for MAXIMUM ITERATIONS. Figure 3-9 shows four versions of a baby Mandelbrot found after a number of zooms. The maximum iterations values used for the four images were 150, 250, 350, and 1000. You can see that the first image has begun to lose the characteristic Mandelbrot shape, but that as the maximum iterations value is increased, the shape gets more accurate. For moderate depth zooms, the default maximum iterations value of 150 works very well.

Color Cycling

The second Fractint mode is called color cycling. In this mode, Fractint rapidly alters displayed colors of an image, giving an effect of animation. This works because most graphics adapters create colors by using what are called color palettes. A color palette is like the color key for children's paint-by-numbers oil painting kits. Each area of the painting is assigned a number, and each number represents a color. The set of all the available colors assigned to numbers is the current color palette. There are as many palette entries as the number of colors your computer's hardware video adapter can display at one time. So, a palette for an EGA has 16 entries, and a VGA has as many as 256. But—and this is a big "but"—the colors assigned to the palette entries are drawn from a much larger selection. For example, the VGA 320 x 200 mode can display 256 colors on the screen at one time, but these 256 can be selected from 262,144 possible colors! (Understand that in Fractint, different shades of the same color are considered different colors. Some fractals, for example, may have only 2 colors with 128 shades.) Your screen is like the child's painting; each pixel is assigned not a color but a color palette entry number. The color of the pixel is the color assigned to its palette number. What Fractint does when it color cycles is rapidly change *which colors are assigned to which palette numbers.* As you'll see, this simple technique creates a magical effect.

Why Color Cycling "Animates" Images

Most fractal images have more information in them than the mind can comprehend. By assigning colors differently, you can make different details visible in the same image. By cycling the colors, areas that make up the fractal are revealed by color moving between them. Because the areas are connected in a highly organized fashion, there is a high degree of animation potential. Playing with the colors is at least half the fun of Fractint. Alas, this feature only works if your video hardware supports at least 16 colors (EGA), and works best in the VGA and super VGA 256-color modes. If you have a CGA or Hercules monochrome graphics adapter, we suggest you skip on to the next section, or go out and buy a super VGA with 1024K of RAM. A few years ago these adapters were the state-of-the-art, but now they are inexpensive commodity items.

Press the ⊕ key to see your fractal color cycle. Show time! The colors of your fractal will now start wildly gyrating! (If the cycling is too fast on your machine, you can slow it down with the ↓ key.)

Color-Cycling Features and Experimenting

Earlier in this chapter we discussed the five modes of Fractint—the display mode, the color-cycling mode, the orbits window mode, the Julia window mode, and the palette editing mode. As soon as you press ⊕ (or ⊖ or C), Fractint enters the color-cycling mode, and a whole new set of command keys takes effect. The ⊕ and ⊖ keys reverse the direction of the color rotation: the ⊕ key makes the colors radiate outward, the ⊖ key makes the colors move inward. To freeze the color scheme, press SPACEBAR. The outside border of the screen will now be white to remind you that Fractint is still in the color-cycling mode, even though the colors aren't moving.

When you first enter the color-cycling mode, Fractint rotates the existing colors in the current color palette. The original color scheme will repeat periodically as the colors rotate (every 256 colors if you have VGA, every 16 colors if you have EGA). Pressing any of the function keys F2 to F10 causes Fractint to randomly create new colors in the existing palette, so the color schemes never repeat (or at least not any time soon). Indeed, the number of color schemes obtained by pressing the function keys is astronomical! These function keys work by periodically adding random colors to the palette, and making the in-between colors ooze continuously between the random colors. The lower function keys (F2, F3, and F4) cause the colors to change abruptly. The higher function keys (F8, F9, and F10) cause the colors to change more smoothly and continuously between more widely spaced random colors. Table 3-4 shows how widely spaced the random colors are. For F2, every fourth color is randomly chosen, and the three in-between colors change smoothly between the two random colors. The effect is one of rapidly moving stripes. At the other extreme, the F10 key causes new colors to be randomly created every 100 colors. This means that the 99 intervening colors smoothly merge from one widely spaced random color to the next 100 colors later. The effect is one of oozing pastels.

Pressing ENTER while the colors are cycling causes the color scheme to be completely and randomly altered.

Slow-Motion Color Cycling

Often you'll want to slow down or speed up the color cycling. There are two ways to do this in Fractint. The first way is to use the ↓ and ↑ keys while colors are

Function Key	Random Color Interval
F2	4
F3	8
F4	12
F5	16
F6	24
F7	32
F8	40
F9	54
F10	100

Table 3-4 Random color interval for color-cycling function keys

cycling. This feature was originally added to control "flicker" on machines with slower graphics adapters. On these machines, slowing down the cycling with the ↓ cleans up the flicker. But if you have one of the new breed of faster computers (such as a 486, or Pentium), you may find the color cycling is just too fast for enjoyment; use the ↓ key to slow it down. The other way to change the color cycling speed is to press any of the number keys, 1 through 9. These keys cause certain colors in the palette to be skipped, effectively increasing the rotation speed. The higher-numbered keys cause colors to rotate faster. Fractint defaults to the speed of the 1 key.

Zoom in several times using PAGE UP—as you recall, most of the colors in the default Mandelbrot are concentrated near the coastline. By zooming in, you will spread them out and see the colors rotate more clearly. The Mandelbrot with the normal IBM palette has a markedly striped appearance. To see more smoothly changing colors, start color cycling with +. Then press F10. The new colors will be added to the end of the 256-color palette, and will take a little time to "flush out" the old colors. To speed things up, press 9, wait until you see that the smoother color changes have taken effect, then press 1 to slow things down again.

Try the function keys in reverse sequence, moving from F10 to F2, waiting long enough for the old color palette to rotate out so you can see the new colors. As you change to lower numbered function keys, the colors of the stripes will start to blend less, and your attention will be drawn away from the lake outline to the stripes.

When you see the image the way you want it, press (SPACEBAR) or (C) to freeze it, and (ESC) to exit the color-cycling mode and return to the regular display mode.

Saving a File

Between creative zooming and color cycling, by now you should have created a few beautiful fractals. Chances are very good that your creation is unlike any other. If you save it as a GIF, it can be opened again and experimented with, or uploaded to CompuServe. (See Appendix B, *Fractint and GIF Files*, for details about the GIF format.)

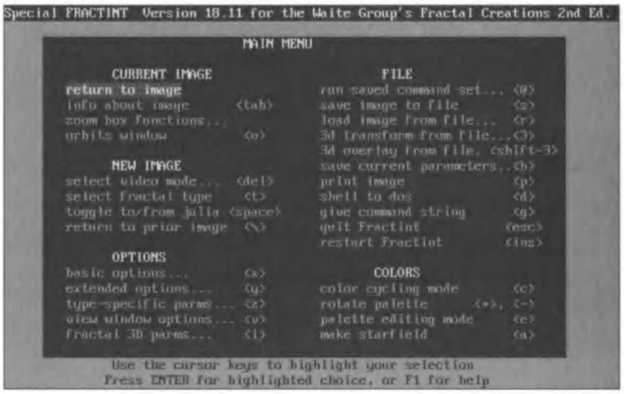
The command to save a fractal image to a file is (S). You must be viewing your fractal, and you must *not* be in the color-cycling mode. (Pressing (ESC) exits the color-cycling mode.) Press (S) to save. You will see two multicolored stripes moving down the right and left sides of the screen like a bar as the saving progresses. Fractint saves images as GIF files; so when done, a window will appear on the screen with the message "File saved as fract001.gif." The number "001" will increment as you save more images. These files will not be overwritten, so watch out for your disk filling up with too many images! A typical 640 x 480 256-color image can use 200 kilobytes or more of storage. If you already have a fractal saved as FRACT001.GIF, the next time you start Fractint and do a save it will increase the number and save it as FRACT002.GIF. You can exit Fractint or drop to DOS (the (D) command) and rename your best fractal GIFs using more meaningful names.

The Expanded Main Menu

Assuming that you have saved your fractal creation, press (ESC) to return to the MAIN MENU. You will notice that the MAIN MENU is no longer the same. Because you have created a fractal image, there are more functions Fractint can perform, so the menu is expanded. These additional menu functions will always show whenever there is a graphics image that has been calculated or read in from a disk. Figure 3-10 shows the expanded MAIN MENU. The next sections will tell you a little about the additional items in the MAIN MENU. The menu is organized according to the groups to which the menu functions belong.

Current Image

There are four additional functions listed that relate to the current image. Fractint can move back and forth nondestructively between menu or information screens



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.
                              MAIN MENU

        CURRENT IMAGE                          FILE
    return to image                  run saved command set... <@>
    info about image      <tab>      save image to file       <s>
    zoom box functions...            load image from file...  <r>
    orbits window          <o>       3d transform from file...<3>
                                     3d overlay from file. <shift-3>
        NEW IMAGE                    save current parameters..<b>
    select video mode...  <del>      print image              <p>
    select fractal type    <t>       shell to dos             <d>
    toggle to/from julia <space>     give command string      <g>
    return to prior image  <\>       quit Fractint          <esc>
                                     restart Fractint       <ins>
        OPTIONS
    basic options...       <x>                COLORS
    extended options...    <y>       color cycling mode       <c>
    type-specific parms... <z>       rotate palette      <+>, <->
    view window options... <v>       palette editing mode     <e>
    fractal 3D parms...    <i>       make starfield           <a>
           Use the cursor keys to highlight your selection
          Press ENTER for highlighted choice, or F1 for help
```

Figure 3-10 The expanded Main menu

and fractal graphics screens. The first menu item is different depending on whether the fractal calculation was complete when you pressed (ESC) to return to the menu. If the image was complete, you will see the menu item RETURN TO IMAGE. If the fractal calculation was interrupted when you returned to the MAIN MENU, you will see the menu item CONTINUE CALCULATION. Both of these will return you to the graphics image; with CONTINUE CALCULATION the fractal calculation will be resumed where you left off.

The INFO ABOUT IMAGE <TAB> selection gives you a screen of status information about the current image. This is particularly useful if you want to find out whether an image has been completed. The (TAB) key allows access to this status information screen directly from an image without needing this menu.

The ZOOM BOX FUNCTIONS item takes you to the Help screen describing keystrokes and mouse actions for manipulating the zoom box.

Finally, ORBITS WINDOW <O> takes you to the orbit window mode discussed earlier in this chapter.

New Image

There are two new items listed under NEW IMAGE, one allowing the recalculation of the previous image and one allowing toggling to and from Julia sets. The RETURN TO PRIOR IMAGE <\> command goes back and recalculates the previous image you created before you zoomed or changed fractal types. This is useful when you are exploring a fractal by zooming, reach a dead end, and want to back up. If you began with the Mandelbrot image, pressing (\) several times will get you back to the full image. Try this now.

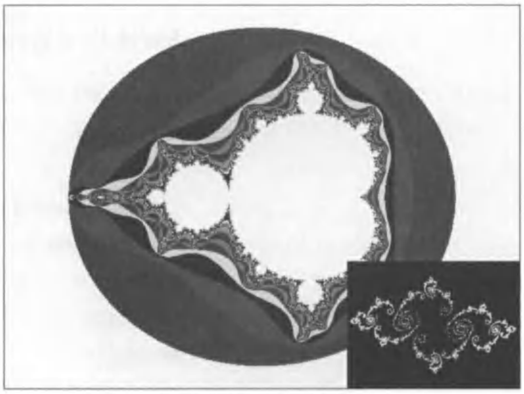


Figure 3-11 The Julia window

Julia Window

The TOGGLE TO/FROM JULIA <SPACE> command initiated by the (SPACEBAR) exploits the relationship between Mandelbrot and Julia fractals that was discussed in Chapter 2, *Fractals: A Primer.* (You will only see this menu item if the fractal type is a Mandelbrot or Julia type. More on this a bit later.) Recall that the Mandelbrot fractal is a "catalog" of Julia fractals; each point of a Mandelbrot fractal corresponds to the Julia fractal with its parameters equal to that point. The (SPACEBAR) lets you see the relationship between the Julia and Mandelbrot sets. The (SPACEBAR) command has been greatly enhanced in Fractint version 18. Let's try it now.

Once you have the Mandelbrot image back on the screen, press (SPACEBAR). A window will appear in the lower right corner and a cross-shaped cursor will appear in the middle of the screen. This window and cursor work in a similar way to the orbits window discussed earlier, with the difference that the window shows an outline of the Julia set rather than the orbit associated with the cursor position. As you move the cursor around the Mandelbrot, the Julia image changes shape, as shown in Figure 3-11. If you want to see the full Julia fractal at any point, just press (SPACEBAR) again, and Fractint will switch to the Julia fractal type and generate the image. Pressing (SPACEBAR) yet again regenerates the Mandelbrot set.

With a little experience, you will be able to predict the appearance of the Julia set from the characteristics of the Mandelbrot image at the point you selected. If the selected point is in the lake, the corresponding Julia set will have a lake. If the point is "on land," the Julia set will not have a single connected lake. Some of the most interesting Julia sets are created with parameter values that are right on the Mandelbrot shoreline.

Mandelbrot Variant	Julia Variant
barnsleym1	barnsleyj1
barnsleym2	barnsleyj2
barnsleym3	barnsleyj3
cmplxmarksmand	cmplxmarksjul
hypercomplex	hypercomplexj
magnet1m	magnet1j
magnet2m	magnet2j
mandel	julia
mandel(fn||fn)	julia(fn||fn)
mandel4	julia4
mandelfn	lambdafn
mandellambda	lambda
mandphoenix	phoenix
manfn+exp	julfn+exp
manfn+zsqrd	julfn+zsqrd
manlam(fn||fn)	lambda(fn||fn)
manowar	manowarj
manzpower	julzpower
manzzpwr	julzzpwr
marksmandel	marksjulia
quat	quatjul

Table 3-5 Fractint's Mandelbrot/Julia pairs

Mandelbrot/Julia Pairs

Many fractal types in Fractint have this Mandelbrot-Julia relationship. Table 3-5 shows all of these Mandelbrot-Julia pairs. The SPACEBAR toggle works with any of them, but it only shows the Julia outline in a window for the traditional Mandelbrot set (fractal type Mandel). For the other types, you can still explore the Julia-Mandelbrot relationship by pressing SPACEBAR, creating a cursor which you can move with the mouse or arrow keys, and pressing SPACEBAR again to generate the Julia.

Options

The OPTION part of the MAIN MENU is the same as it was before you generated an image. It allows you to set parameters that affect how images are calculated.

File

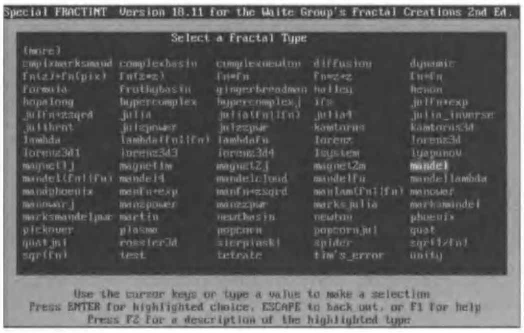
The FILE group of the MAIN MENU covers different ways of getting information into and out of Fractint. The SAVE IMAGE command (S key) creates a file in CompuServe's GIF format (see box).

THE SPIFFY GIF FORMAT: The GIF acronym (pronounced "Jiff") stands for Graphics Interchange Format, which is a device-independent way of representing images developed by CompuServe Information Service. GIF has the advantage that software to view images in this format is widely available on many different kinds of computers. The GIF89a has an additional advantage that makes it the format of choice for saving Fractint fractals. Fractint uses special areas in the GIF89a format to store all the fractal information needed to reproduce the file. If you open a GIF89a format file created by Fractint (or its sister programs Winfract or Xfract), Fractint can extract from the file not only the image but all the Fractint settings that were used to generate the file. Better yet, if the image was saved before the fractal calculation was completed, Fractint can open the partially completed GIF file, load the parameters, and resume the calculation. GIF87a files are an earlier version of GIF. This file format is provided only for compatibility with older software that cannot handle GIF89a. Fractal information is not stored with GIF87a files.

Another way to save all the Fractint settings is to use the SAVE CURRENT PARAMETERS <B> command, which saves these settings in a Fractint .PAR file. The information in this file is in the form of a named set of command-line options. You can modify the .PAR file with a text editor. For more information on the command controlled by the B key, see the reference section later in Chapter 5, .*Fractint Reference*.

Colors

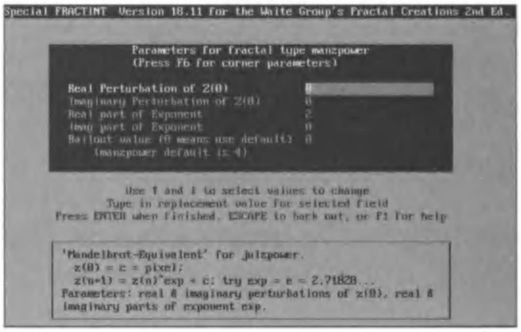
The COLORS group in the MAIN MENU allows you to color cycle. You will see this only if you are in a graphics video mode that allows color cycling, such as EGA or VGA 16- or 256-color modes. In addition to the color-cycling commands, you



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.
                         Select a Fractal Type
 (more)
 cmplxmarksmand complexbasin   complexnewton  diffusion      dynamic
 fn(z)+fn(pix)  fn(z*z)        fn*fn          fn*z+z         fn+fn
 formula        frothybasin    gingerbreadman halley         henon
 hopalong       hypercomplex   hypercomplexj  ifs            julfn+exp
 julfn+zsqrd    julia          julia(fn||fn)  julia4         julia_inverse
 julibrot       julzpower      julzzpwr       kamtorus       kamtorus3d
 lambda         lambda(fn||fn) lambdafn       lorenz         lorenz3d
 lorenz3d1      lorenz3d3      lorenz3d4      lsystem        lyapunov
 magnet1j       magnet1m       magnet2j       magnet2m       mandel
 mandel(fn||fn) mandel4        mandelcloud    mandelfn       mandellambda
 mandphoenix    manfn+exp      manfn+zsqrd    manlam(fn||fn) manowar
 manowarj       manzpower      manzzpwr       marksjulia     marksmandel
 marksmandelpwr martin         newtbasin      newton         phoenix
 pickover       plasma         popcorn        popcornjul     quat
 quatjul        rossler3d      sierpinski     spider         sqr(1/fn)
 sqr(fn)        test           tetrate        tim's_error    unity

          Use the cursor keys or type a value to make a selection
    Press ENTER for highlighted choice, ESCAPE to back out, or F1 for help
          Press F2 for a description of the highlighted type
```

Figure 3-12 Fractal type selection screen



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

              Parameters for fractal type manzpower
              (Press F6 for corner parameters)

      Real Perturbation of Z(0)               0
      Imaginary Perturbation of Z(0)          0
      Real part of Exponent                   2
      Imag part of Exponent                   0
      Bailout value (0 means use default)     0
          (manzpower default is 4)

             Use ↑ and ↓ to select values to change
            Type in replacement value for selected field
      Press ENTER when finished, ESCAPE to back out, or F1 for help

   'Mandelbrot-Equivalent' for julzpower.
     z(0) = c = pixel;
     z(n+1) = z(n)^exp + c; try exp = e = 2.71828...
   Parameters: real & imaginary perturbations of z(0), real &
   imaginary parts of exponent exp.
```

Figure 3-13 Fractal parameters for fractal type manzpower screen

can access Fractint's color editor and starfield functions from this menu. These are discussed in Chapter 5, *Fractint Reference*.

A Generalized Mandelbrot Set

Up to this point all the examples have been limited to the Mandelbrot set. This is actually not such a serious limitation—this one fractal type alone has a richness of shape and form that defies imagination. Fractint gives you such power that you can examine the details of the Mandelbrot at immense magnifications. (If you have not rotated or distorted your zoom box, you can see the magnification of your zooms on the TAB status screen.) But the Mandelbrot set is just the beginning of where you can explore using Fractint, so let's be adventurous and try another fractal type.

As we saw, you can select a fractal type by pressing the T key or choosing SELECT FRACTAL TYPE from the MAIN MENU. The SELECT A FRACTAL TYPE screen is shown in Figure 3-12. Use the arrow keys to select type manzpower. Fractint has a speed key feature, so when you start typing the name of the fractal, the highlight jumps to the first fractal type matching the name. In this case you only have to type **manz** to make the highlight jump to manzpower. Next press ENTER, and you will see a screen entitled PARAMETERS FOR FRACTAL TYPE MANZPOWER, as shown in Figure 3-13. (You can also reach this screen using the Z key from the MAIN MENU or while viewing a fractal.) Most fractal types have parameters that you can change to alter the appearance of the fractal, and this screen lets you control these parameters. At the bottom of the PARAMETERS screen is a window frame showing the formula used to generate the fractal and showing how the parameters are used

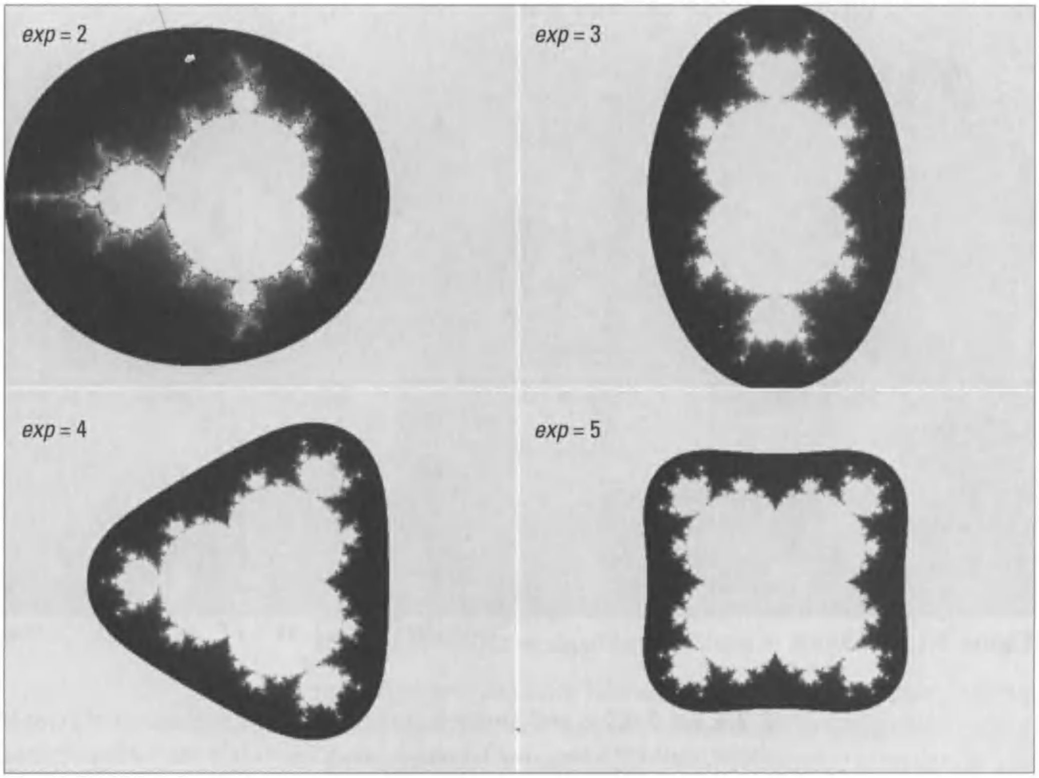


Figure 3-14 Manzpower images using parameters 2, 3, 4, and 5

in the fractal calculation. In the case of manzpower, the iterated formula is $z_{n+1} = z_n^{exp} + c$. In the case $exp = 2$, this is the same as the familiar Mandelbrot formula. Press (ENTER), and if you haven't changed the parameters from the values shown in Figure 3-13, Fractint will calculate the familiar Mandelbrot set.

Let's find out what happens if the exponent parameter *exp* is not the usual 2 which results in the Mandelbrot image, but another value such as 3. Press (Z) to return to the PARAMETERS screen, and change the parameter labeled REAL PART OF EXPONENT to 3 and press (ENTER) to see the result. Then repeat the experiment using the values 4, 5, and 6. The resulting images are shown in Figure 3-14. It is easy to see a pattern. For the value $exp = 3$. the fractal has two lobes that look somewhat like one end of the Mandelbrot. When $exp = 4$, the fractal becomes a triangular shape with three lobes, and so forth.

You can not only use integral values such as 2, 3, 4, or 5 for the *exp* parameter, but fractional values as well. Try making a series of images using values such as

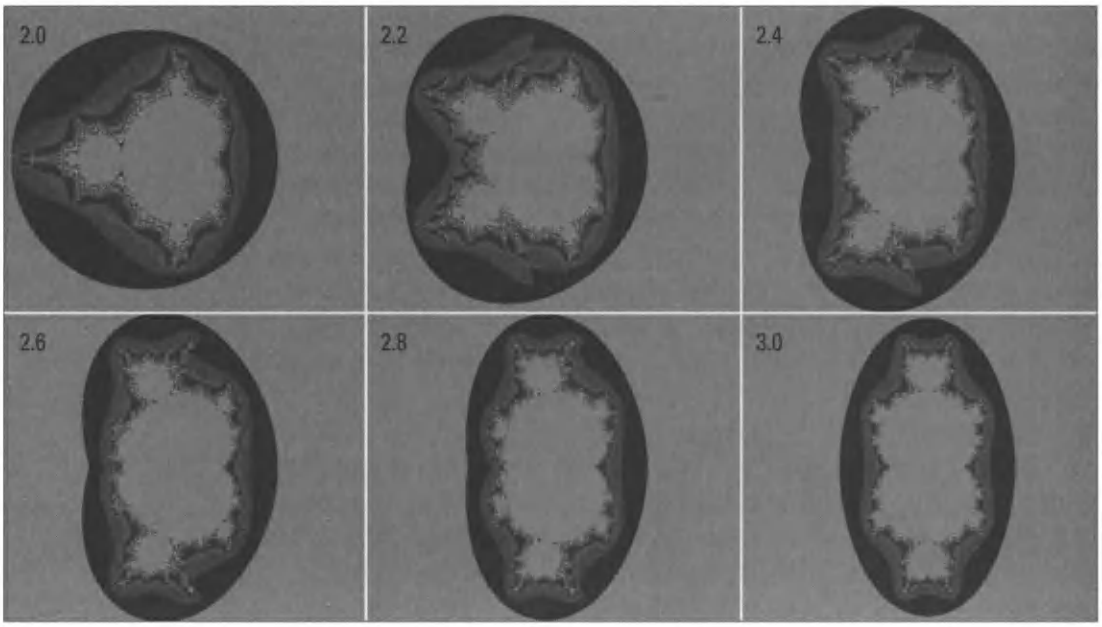


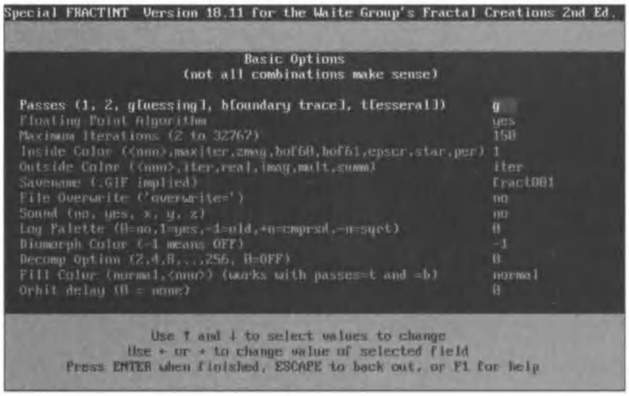
Figure 3-15 Evolution of manzpower images as exponent changes

2, 2.2, 2.4, 2.6, 2.8, and 3 to see how the Mandelbrot evolves into the two-headed shape resulting from *exp* = 3. The left-hand "bay" of the Mandelbrot splits in two, and the two halves migrate to the top and bottom. This series of images is shown in Figure 3-15.

One variation of this fractal, using the value *exp* = 2.71828182845905 (the number known to mathematicians as *e*), has been extensively studied by Lee Skinner, which he calls the Zexpe fractal. (You can find many examples of images using this formula in The Waite Group Press book *Image Lab* by Tim Wegner, © 1992, and on the companion CD.) To explore this fractal, return to the PARAMETERS screen for manzpower by pressing Z, use the arrow keys to move the highlight to REAL PART OF EXPONENT, and enter **e**.

HINT: When entering parameters, you can type the number directly or indirectly. Typing **e** inserts 2.71828182845905 (the number *e*), and typing **p** inserts 3.14159265358979 (the number).

Do not judge a fractal from its "outer" appearance. The Zexpe fractal doesn't look much different from the Mandelbrot set at low zoom rations, but when you zoom inside you will discover that they are completely different. Try this now!



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

                              Basic Options
                     (not all combinations make sense)

 Passes (1, 2, g[uessing], b[oundary trace], t[esseral])      g
 Floating Point Algorithm                                     yes
 Maximum Iterations (2 to 32767)                              150
 Inside Color (<nnn>,maxiter,zmag,bof60,bof61,epscr,star,per) 1
 Outside Color (<nnn>,iter,real,imag,mult,summ)               iter
 Savename (.GIF implied)                                      fract001
 File Overwrite ('overwrite=')                                no
 Sound (no, yes, x, y, z)                                     no
 Log Palette (0=no,1=yes,-1=old,+n=cmprsd,-n=sqrt)            0
 Biomorph Color (-1 means OFF)                                -1
 Decomp Option (2,4,8,...,256, 0=OFF)                         0
 Fill Color (normal,<nnn>) (works with passes=t and =b)       normal
 Orbit delay (0 = none)                                       0

                  Use ↑ and ↓ to select values to change
               Use ← or → to change value of selected field
       Press ENTER when finished, ESCAPE to back out, or F1 for help
```

Figure 3-16 The Basic Options screen

In this discussion we have discussed only *one* of the four manzpower parameters. Try entering small values for the other parameters and gradually increasing them to see what they do.

Options, Options, We've Got Options!

From this tutorial so far, you have learned three different methods to create beautiful fractal images. The first is to play with the colors using color cycling. The second is to explore the "terrain" of the fractal by zooming in on details. The third is to experiment with different parameter values. Now you'll learn a fourth method, experimenting with Fractint's basic and extended options.

Just so we are all together, press (INSERT) to return Fractint to its startup defaults. Press (T) to get the SELECT A FRACTAL TYPE menu. Move the highlight to the manzpower fractal type using the arrow keys or by typing **manz**. Press (ENTER) to get to the PARAMETERS FOR FRACTAL TYPE MANZPOWER screen. Enter the value 4 for the REAL PART OF EXPONENT parameter and press (ENTER). You should now see the MAIN MENU. Press a function key for a quick, low resolution mode such as the (F3) 320 x 200 x 256 mode. You should see the triangular manzpower image we discussed earlier. Press (X) (or (ESC) and select BASIC OPTIONS from the MAIN MENU) to access the BASIC OPTIONS screen, shown in Figure 3-16. Have a look at the odd assortment of things the (X) key (BASIC OPTIONS command) allows you to set. Don't worry—you don't have to understand them all at once. In fact, you can rarely use most of these settings and still get a tremendous amount of enjoyment out of Fractint. Using these options is a lot of fun though, because you can transform your images in intriguing often unpredictable ways. In this section we'll give you some tips on using some of these options.

Two of the most powerful controls on the Basic Options screen are the inside and outside options. The "inside" points are those where the maximum iteration limit is reached before the bailout criterion is met. For example, the Mandelbrot "inside" points form the blue lake region in the center of the fractal image. The "outside" points are those points where the orbit eventually satisfies the bailout criterion. The outside points of the Mandelbrot form its stylish stripes.

The inside options affect the points whose orbits never escape, but are eternally drawn to points within the fractal set known as "attractors." Fractint's default action is to color the inside points with the solid color blue. If you set the inside parameter to a number, Fractint will use that color value. (Technically, Fractint uses the number as an index to look up a color from the current color palette.) Alternatively, you can use the other inside values to reveal the orbital dynamics of the inside points in various ways. The possibilities are listed on the screen—Maxiter, Zmag, Bof60, Bof61, Epscr, Star, and Per.

These options are documented in detail in Chapter 5, *Fractint Reference*. You can also learn about them using Fractint's context-sensitive hypertext help system. While the Basic Options screen is showing, press the help key (F1). You can use the arrow keys to select highlighted topics and jump to them with (ENTER). (This is what is meant by hypertext.) Pressing the (BACKSPACE) allows you to back up to previous help screens, and the (ESC) key returns you to the Basic Options screen.

The outside option works in a way similar to the inside. It affects points whose orbits meet the bailout criterion before the maximum iterations limit is reached. Fractint's default action is to color the outside points according to the iteration when the orbit met the bailout criterion (outside set to "iter"), resulting in the familiar "escape time" stripes. As with the inside option, you can color all the outside points with a solid color by entering a color number. The remaining outside options render the areas where orbits escape in different ways. The possibilities include Iter, Real, Imag, Mult, and Summ. Let's try a few combinations of these options and see what happens.

Simple Is Beautiful—Solid Inside and Outside Colors

A mathematics teacher sent the Stone Soup Group a letter saying that she wanted to show her students just the Mandelbrot set by itself, without any distracting escape-time bands that really aren't part of the fractal. As a result of this request, we added the ability to set the outside colors to a solid color. For example, to make a white-on-black fractal, use the arrow keys on the Basic Options menu to select the Inside Color option, and enter the value 15 (white in the default IBM PC color palette). Then move to the next line below with the arrow keys to the Outside

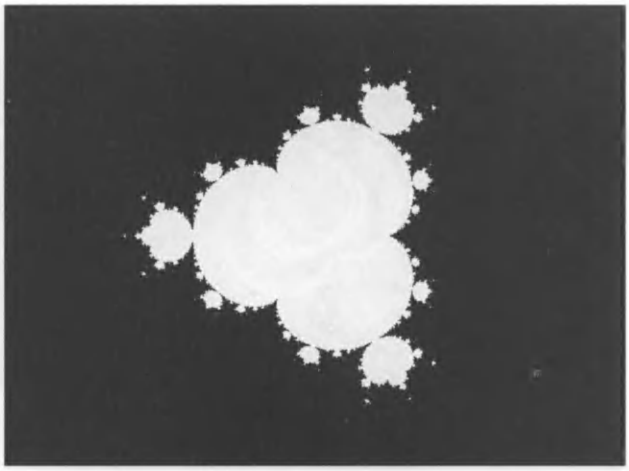


Figure 3-17 Manzpower outline with outside=0 and inside=15

COLOR line and enter the number 0 (black). Press (ENTER). You should now see the familiar triangular shape of the manzpower fractal with exponent four, only this time as a solid white shape, as shown in Figure 3-17.

Inside Mysteries—Zmag and BOF60

Press (X) again to return to the BASIC OPTIONS screen, move to INSIDE COLOR with the (↓) key, and type **zmag**. Press (ENTER) to recalculate the manzpower fractal. Each lobe of the fractal is the center of a basin with lower color numbers. Try pressing (+) to cycle the colors. The colors seem to move toward the centers of the lobes of the fractal. This effect is accomplished by coloring according to the magnitude of the last orbit value when the maximum iterations were reached, using the formula color = $(x^2 + y^2)$ maxiter/2 + 1.

For a subtly different effect, press (X) (after pressing (ESC) if you are still cycling colors) to return to the BASIC OPTIONS screen, and change INSIDE COLOR to BOF60. This time the coloring is according to the closest distance the orbit comes to the origin. The result of both of these inside options is shown Figure 3-18.

Outside Options—Real

Revisit the BASIC OPTIONS screen once again by pressing (X). Change FLOATING POINT ALGORITHM to Yes and set INSIDE COLOR back to the original value of 1 (blue). (If your

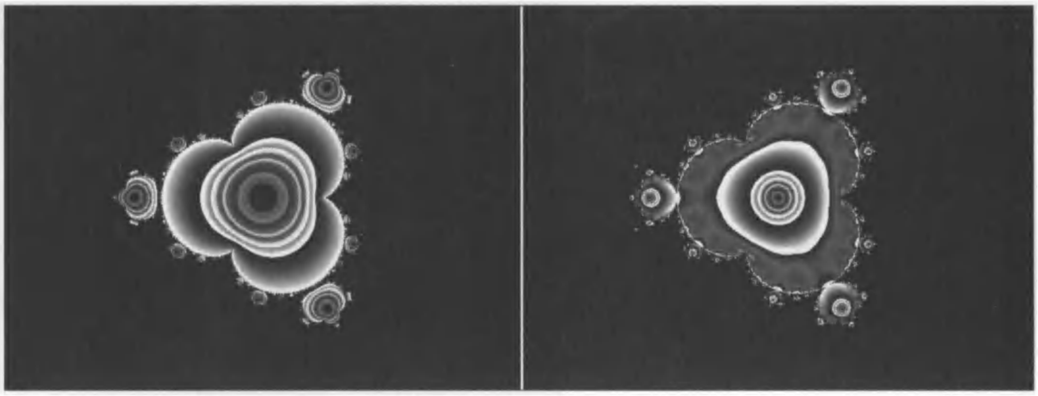


Figure 3-18 Results inside=zmag and inside=bof60

computer does not have a math coprocessor and the fractals generate too slowly, go ahead and set floating point back to No. The results will be a bit different, but you can still follow along.) You can mix inside and outside options; but while you are learning, it is a good idea to use a solid color for the inside option while experimenting with outside, so that you can see clearly the effect of your experiments. Move the highlight one row lower to OUTSIDE COLOR using the ↓ key, and type in `real`. This option colors pixels according to the sum of iteration when the orbit escaped and the real part of the last orbit value. Press ENTER to see the result, shown in Figure 3-19. If you look carefully, you can see the original escape-time bands, but a repeating pattern of curves is now superimposed onto the bands.

Now press the PAGE UP key to create a zoom box, and keep pressing PAGE UP, shrinking the zoom box until it just fits inside the central blue "lake" area of the fractal. While holding down CONTROL, press ENTER. This causes Fractint to "zoom out" for a more distant view. You can now see an eight-pointed star made up of two hyperbolas.

If you are using Fractint's integer mode for speed, you will see the star inscribed in a polygonal shape with a background of stripes. The striped background is due to the limitation of fast integer math, which cannot handle large corners values. You can toggle back and forth between floating-point and Fractint's fast integer math using the F key. If you are not sure which mode you are presently using, press TAB. If you are using floating-point math, you will see the message "Floating-point flag is activated."

There is an interesting relationship between this star and the exponent used in the manzpower fractal. Press Z to return to the PARAMETERS FOR FRACTAL TYPE MANZPOWER screen. Use the ↓ key to move the highlight to the third parameter,

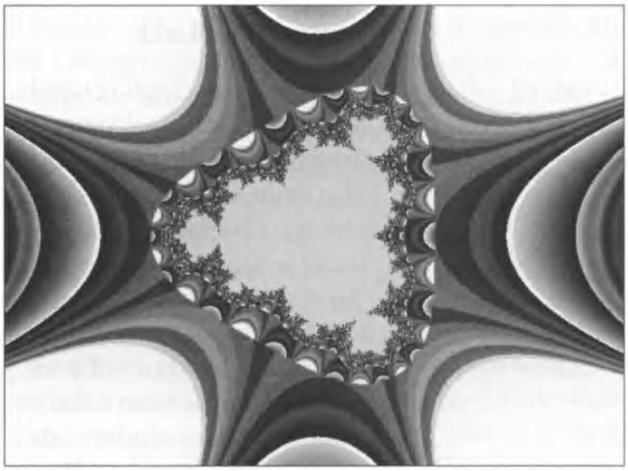


Figure 3-19 Outside=real option

labeled Real Part Of Exponent, and change the value from 4 to 1 and press (ENTER). You will see an image consisting of concentric circles, with the circles broken into vertical stripes. Press (Z) again and change Real Part Of Exponent to 2. As mentioned earlier, the manzpower fractal with exponent 2 is the Mandelbrot set. This time, however, the vertical stripes have been bent to form two hyperbolas, forming a four-pointed star. (You might recall from geometry that "one" hyperbola is really two curves.) Repeat this experiment once again by pressing (Z) and setting Real Part Of Exponent to 3. Now—guess what—three hyperbolas form a six-pointed star. Just for fun, try one more time, bumping the exponent up to a larger number, say 6. Voila! A twelve-pointed star!

A 3-D Mandelbrot Set

Let's continue the Fractint tour. Next, we are going to perform some 3-D transformations on the Mandelbrot fractal you created earlier. For the mathematically curious, the result will be a 3-D plot of the escape times of the Mandelbrot formula. Be sure to try color cycling using the (+) key with these 3-D examples. Also, note that the zooming feature does not work in the 3-D mode.

From the Main Menu, select Restart Fractint or press the (INSERT) key, which accomplishes the same thing. This command reinitializes almost all settings to their default values, with the same result as exiting and restarting Fractint.

Press the (X) key, and set Inside Color to Maxiter. To do this, use the (↓) key to move the highlight down to Inside Color and type `maxiter`.

Moving the Lake

The "inside color" option sets the palette number of the color used to color the lake areas of fractals—locations where the orbit never escapes and the maximum iterations limit is reached. This inside color value defaults to 1, which is the color blue in the standard IBM color palette on both EGA and VGA adapters. This setting is a Fractint tradition begun by the original author, Bert Tyler. Many other fractal programs favor the color 0, which is black, but Bert preferred to see a blue "lake". Setting the inside color to maxiter has the effect of setting the inside color to the maximum iteration value, which defaults to 150.

Normally, the choice of inside color is purely aesthetic, but not for what we are about to do. The reason is that for interpreting fractals for 3-D purposes, Fractint treats the color as a number, and the color number is interpreted as the height above the plane. A color number of 2 means a low point, while a high color number means a mountain or high place. The setting of the inside color to maxiter has the effect of making the lake float at the top of the 3-D surface. This makes mathematical sense because the resulting image is a graph of the iteration count of the escape-time calculation, and when the lake occurs the iteration value is at the maximum, or 150, unless you change it with the (X) command. The important point is that setting the inside color affects the height of the lake when you are doing a 3-D transformation, and we have jammed it to the maximum height so it hangs above the plane of our fractal. Set the inside color to maxiter now.

When you are done with the (X) command screen, press (ENTER) to accept the values and return to the screen where you'll generate the fractal image. If you have a VGA, press the (F3) key to generate a Mandelbrot image in the 320 x 200 256-color mode. Note that the lake area is no longer blue but rather gray. If you don't have a VGA or other adapter with a 320 x 200 x 256 mode, then use the Disk/RAM Video 320 200 256 mode. This mode is buried way down the list. Cursor down to it, highlight the mode, and press (ENTER). (The Disk/RAM video isn't really "video" at all. Rather, it is a way of creating images using your disk, or, if you have enough memory, your extended or expanded memory.) Later we'll be able to display the disk video file.

When the image is complete (you'll hear a little whistle), save the image by pressing the (S) key. Make a mental note of the file name that was reported at the top of the screen—it was probably something like FRACT003.GIF, depending on how many images you have already saved. Return to the MAIN MENU with (ESC). Select 3D TRANSFORM FROM FILE from the FILE menu, or press the (3) key. You will now be presented with a list of files. Use the arrow keys to move the highlight to the file you just created with the save operation, and press (ENTER). Next you will see a list of video modes. The video mode you used to generate the Mandelbrot image should be highlighted. If you have a VGA, you can use the same (F3) mode. Press (ENTER) to select it. If you have a CGA, use the F5 IBM 4-color CGA 320 200 4 mode, and if you have an EGA, use the F9 Low-Rez EGA 320 200 16 mode. In all cases, try to use a mode as close as possible to 320 pixels wide and 200,pixels high.

Setting the 3-D Parameters

Fractint is now going to lead you through some screens that allow setting all manner of parameters and effects for 3-D. The good news is that the default values almost always make sense—you do not have to understand what all of them mean. The screens are documented in detail in the Chapter 5, *Fractint Reference*.

The first screen is entitled 3D MODE SELECTION. Here is where you can turn on what's known as funny-glasses stereo (stereo using red/blue glasses) or use the sphere mode to create a fractal planet. But not yet! This time around, press (ENTER) to accept all the values. The next screen is entitled SELECT 3D FILL TYPE. Here you will find the Fractint light source options, which let you illuminate your fractal and create shadows. The JUST DRAW THE POINTS option will be highlighted, which is just fine for now, so press (ENTER).

The next somewhat imposing screen, entitled PLANAR 3D PARAMETERS, presents numerous options for these three-dimensional rotations and scale factors. You can view your fractal from different angles, spin it in space, stretch it, shrink it, and move the viewer's perspective right into the middle of it! When making fractal landscapes, you can control the roughness of mountain ranges and the height of floods in the valleys. For this tour, the default values are all okay, so press (ENTER) to accept them. After the tour, you can come back and experiment.

If you have a slower XT- or AT-compatible PC, you may want to get up from your chair, stretch your legs, and grab a quick cup of coffee or beverage of choice. The 3-D transformation takes a few minutes. You will see the blue background of the Mandelbrot image appear just as you saw it on the screen a few moments before, but laid at an angle like a piece of paper on a desk. As the image develops, you will see that the colored stripes of the Mandelbrot image are raised like Chinese terraces on a mountainside. Floating above everything is the dark blue Mandelbrot lake, raining a sparkling mist down to the terrain below. You should now understand why we set `inside=maxiter`. If we had left the default `inside=2`, the lake would have been at the same level as the blue background, instead of floating mysteriously above it.

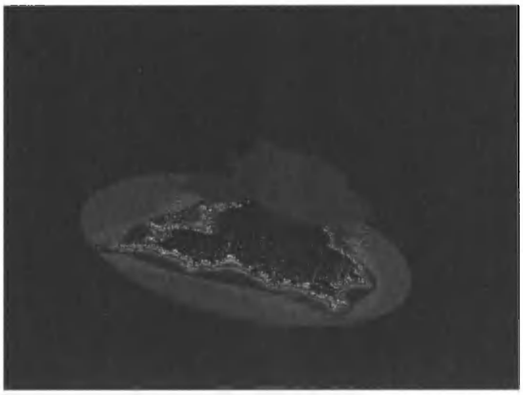


Figure 3-20 3-D Fractal

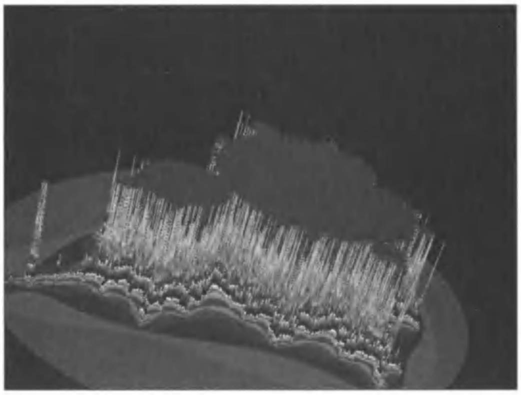


Figure 3-21 Mandelbrot cliffs in perspective

Figure 3-20 shows what your 3-D fractal should look like. For the final touch, press (+) to launch color cycling, and try the higher function keys to create some smoothly changing colors. When you are done playing with the colors, exit the color-cycling mode with (ESC).

Variations on a Theme

Let's try a few variations. For each variation, start with the (3) command. Fractint will remember your previous settings, and you can move from screen to screen by pressing (ENTER), pausing only to make the indicated changes. If you overshoot a screen, you can back up with (ESC). The one setting that will *not* be remembered is the video mode for you CGA and EGA owners who used Disk/RAM video. Each time you are asked for the video mode, you should press (F5) (CGA) or (F9) (EGA). If you have a VGA and used (F3) to generate the original image, you will not have this minor complication, because in your case the video list will come up with the (F3) mode highlighted. You can accept it by pressing (ENTER) just as you do for the other screens that do not require changes.

3-D Variation #1: Make solid cliffs. Start with the (3) command, and move through the screens with (ENTER). When you come to the SELECT 3D FILL TYPE screen, select SURFACE FILL (COLORS INTERPOLATED), but otherwise leave the settings unchanged, pressing (ENTER) until the image regenerates. This option definitely slows things up, so take another break! This time the floating Mandelbrot image will become the top of a mountain with precipitous cliffs hanging under it.

3-D Variation #2: Add a perspective viewpoint. Start with (3), and move through the screens with (ENTER). When you come to the PLANAR 3D PARAMETERS screen, look for Perspective Distance [1–999, 0 For No Persp] about halfway down. Type in 150. Smaller numbers provide the more extreme perspective of a closer viewpoint, while higher numbers create a flatter perspective such as photographers obtain through a telephoto lens. Press (ENTER) to regenerate the image. As a side effect the image edges will look a little bit rougher. But you are now closer to the scene, with closer features expanded! Figure 3-21 shows the "Mandelbrot cliffs" in perspective.

3-D Variation #3: Make the mountain into a lake. Throughout this book we have referred to the classic Mandelbrot shape in the center of the Mandelbrot fractal as a lake, but then we turned around and made it into a mountain top in 3-D. We'll show you how to remedy that. Start with (3), and move through the screens with (ENTER). When you come to the Planar 3D Parameters screen, look for Surface Roughness Scaling Factor In Pct, which should have the default value of 30. We want you to depress the mountain top and make it a lake, so change the surface roughness value to –5. That's right, the new value is –5, which means that the *z*-coordinate will be scaled by –5 percent— depressing the mountain top below the surrounding plain. Press (ENTER) to regenerate the image. You have turned the mountain into a lake-bottomed canyon!

And Now, Images in Stereo!

Let's try one more bit of magic and plot this Mandelbrot image in red/blue stereoscopic 3-D. For this you'll need your trusty red-and-blue glasses—the ones that came with this book. Here's how these images work: A number of different cues tell you that a scene has depth. Distant objects appear smaller than nearer objects. As you move your head, nearer objects get in the way of farther objects. Mist obscures distant objects. Because of these cues, a person with one eye can still perceive depth. Those of us with two good eyes have another depth cue: binocular vision. Our brain fuses the images coming from our two eyes and gives a sense of depth.

Red/Blue Glasses

Fractint is capable of performing the perspective transformations necessary to simulate the viewpoints of two eyes. The problem is how to get the left image to the left eye and the right image to the right eye. One solution would be to rapidly alternate the left and right images on your screen and have the user wear special glasses with liquid crystal shutters synchronized to the monitor. High-end workstations can be purchased with this capability, but it is expensive, costing thousands of dollars. For Fractint we have opted for a simpler approach that has cost you just the price of this book: red/blue glasses. These are the same kind of glasses that kids of a bygone era eagerly retrieved from cereal packages in order to view stereo scenes on the box.

The idea is simple. Fractint puts a left view of a fractal on the screen in red and a right view of the image in blue. You put on the glasses, which have filters that block the incorrect view from reaching your eyes: blue blocks red, and red blocks blue. (Note that some 30 percent of the population cannot see these binocular effects for one reason or another, and we hope you are not one of them!) And alas, even if your eyes are perfect, you are still going to need a color monitor that can display red and blue. That rules out all monochrome setups as well as CGA, which cannot show red and blue. If you have EGA or VGA with color, you are in business.

Press (ESC) to return to the MAIN MENU, and then press (3). Select the same GIF file from the SELECT FILE FOR 3D TRANSFORM screen that you created before (follow the previous instructions to make it if you haven't already). Set the video mode to (F3) for VGA or (F9) for EGA. At the 3D MODE SELECTION screen, cursor down to the bottom option labeled STEREO (R/B 3D)? (0=NO,1=ALTERNATE 2=SUPERIMPOSE,3=PHOTO) and press (2) and then (ENTER). This is the superimpose option, which describes how the red and blue colors will be combined on the screen. The superimpose method combines colors red and blue to make magenta and pink, giving sharper results but fewer color shades. The alternate option alternates red and blue dots on the screen, sacrificing resolution but allowing more color shades. The photo mode is for photographing the screen and making stereo slides. The reference section of this book explains all this in more detail.

Under SELECT 3D FILL TYPE, select the top option MAKE A SURFACE GRID. Press (ENTER). Note that (ESC) lets you back up to previous screens if you want to change something. You will then come to the FUNNY GLASSES PARAMETERS screen, which you did not see in the previous examples. The defaults are OK, so press (ENTER). If you changed the surface roughness parameter on the PLANAR 3D PARAMETERS screen in the previous examples, change it back to 30 and press (ENTER).

This time you will see a grid approximating the solid Mandelbrot image you generated a moment before. First a red image is generated, then a blue image. These images should look like the Mandelbrot cliffs image of the previous example, shown in Figure 3-21. Put on the red/blue glasses that came with this book and view the image, making sure the red lens is over the left eye. There you are in living 3-D—a Mandelbrot mountain outlined in a grid!

Clouds, Mountains, and Plasma

A fractal that is particularly interesting for 3-D experiments is the plasma type. This fractal allows the creation of both cloud and mountain range images. Who would have guessed that mountains and clouds are so closely related? The plasma type makes a random pattern of smoothly changing colors that look like clouds. This type works best with a 256-color mode. If you have an EGA, you can follow along, but the results will be somewhat different.

Select Plasma

Just to make sure we are on the same track, press the INSERT key to reinitialize Fractint, and then select a video mode. The F3 mode is a good choice for VGA, F9 for EGA. This will generate the Mandelbrot image again! (The plasma type will not work with modes with fewer than 16 colors, which rules out CGA and Hercules adapters.) After you have selected a video mode (and pressed ENTER if you selected the mode from the mode list), press T to display the type screen. You don't have wait for the Mandelbrot image to finish before pressing T.

You can select "plasma" by moving the highlight with the cursor keys, or you can just start typing the word `plasma` and use the speed key feature mentioned earlier. Because plasma is the only type that begins with "pl," as soon as you have typed these two letters, the highlight will jump to the plasma type. Now press ENTER to select the highlighted fractal type. After selecting a type, Fractint prompts you for any parameters that affect the appearance of that particular type. In the case of the plasma type, the "graininess" factor parameter affects how gradually the colors on the screen merge with one another. You have a choice of two different algorithms, one that winds recursively around smaller and smaller blocks, and the other which systematically covers the screen with multiple passes. Both give the same result. Because the shape of the plasma pattern is normally randomly determined, the random seed value allows you to specify if you want each plasma fractal to be newly created, or if you want to repeat the last pattern. For now, press ENTER to accept all the default values. You might want to come back later and experiment. To do so, start with the T command and reselect type plasma, which will get you back to the plasma parameter screen, or move directly to the parameter screen while viewing a plasma fractal using Z.

As soon as you have pressed ENTER to accept your parameters choice, the plasma calculation will begin. What you are seeing on the screen is a fascinating algorithm that recursively subdivides the screen, randomly choosing colors with values between surrounding colors. No two plasma images are quite the same because of the random element of the calculation. When the image is complete, start color cycling with the + command. Now you understand why this type is called "plasma"! The screen colors ooze and writhe in graceful undulations of ethereal plasma waves. Be sure to try the function keys while color cycling. The lower-numbered function keys (F2, F3,...) give detailed paisley patterns, while the higher ones (F8, F9, and F10) result in larger, flowing patterns.

Turning a Cloud into a Mountain

Take a good, long time playing with the plasma type, which is certainly one of the more colorful and dramatic fractals that you can create with Fractint. Be sure to press the ENTER key occasionally while it's color cycling; this instantly changes the colors. We should have convinced you that Fractint can make clouds, but what about mountains? To create mountains you have to first save a plasma image. If you are still in the color-cycling mode (as visually indicated by either moving colors or a white screen boundary) then you should exit to the display mode by pressing ESC. Press S to save the plasma screen, once again making a note of the file name that is reported on the screen. You are probably up to file FRACT022.GIF by now, right?

We can turn a cloud into a mountain by doing a 3-D transformation on the colors of the cloud. A cloud image can be considered a color-coded contour map of a mountain, where areas of equal color are the same height. By performing a 3-D transformation, we are transforming the contour map back into the mountain it represents.

Press the 3 key to invoke the 3-D function, and select the just-saved file from the file list. At the VIDEO MODE SELECTION screen, select the same video mode used to generate the plasma in the first place (F3 for VGA, F9 for EGA). Accept the default values of the 3D MODE SELECTION by pressing ENTER. At the SELECT 3D FILL TYPE screen, select SURFACE FILL (COLORS INTERPOLATED) and press ENTER. This will bring you to the PLANAR 3D PARAMETERS screen. If you have an EGA and are reading in a 16-color plasma file, set surface roughness to 500. VGA users reading in a 256-color plasma file can leave the default value of 30 unchanged. Press ENTER, and watch a mountain emerge before your eyes!

Plasma Mountain Variations

Plasma Mountain Variation #1: Make the mountain emerge from water. Repeat all the instructions for making a mountain from a saved plasma cloud in the previous paragraph, until you reach the PLANAR 3D PARAMETERS screen. Then set

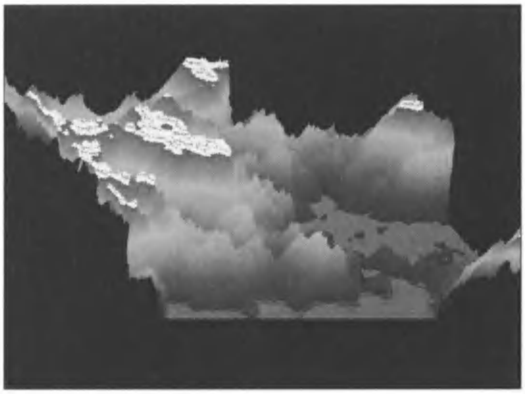


Figure 3-22 Plasma mountains

the WATER LEVEL (MINIMUM COLOR VALUE) item to 47. This will cause all color values less than 47 to be mapped to a flat lake surface. You can begin to see why George Lucas developed Pixar to use computers to simulate real terrain. After the mountain landscape has been created, enter the color-cycling mode by pressing (C). You will be in color-cycling mode, but the colors will not be moving. Then press (l) (for "load map"). This color-cycling command allows you to load various color maps.

There is a special map called TOPO.MAP that has color values tailored to plasma mountains, complete with water, rocks, greenery, and snow. Select TOPO.MAP from the file list. (If it is not in the list, type in the drive letter and directory where you put the Fractint files. For example, if your Fractint files are in c:\fractint type `c:\fractint\`(ENTER). Then the map file screen will be refreshed and you should see TOPO.MAP.) Select it, and press (ENTER). Exit the color-cycling mode by pressing (ESC). You should then see a plasma mountain with more realistic landscape coloring—blue water, green hillsides, brown fields, and snow-capped mountains. Figure 3-22 shows an example.

Variation #2: Make a red/blue 3-D glasses plasma mountain. Repeat the plasma mountain instructions, beginning with the (3) command, selecting the same plasma file, and using the (F3) video mode. At the 3D MODE SELECTION screen, cursor down to the bottom option labeled STEREO (R/B 3D)? (0=NO,1=ALTERNATE,2=SUPERIMPOSE,3=PHOTO), type `2` , and press (ENTER). Under SELECT 3D FILL TYPE, select the top option, MAKE A SURFACE GRID. Press (ENTER). Continue to press (ENTER), accepting all the defaults for the remaining screens. The result is a wire-frame mountain, which makes an excellent red/blue 3-D glasses stereo image. The "grid" fill type has the virtue of being very fast, so it is an excellent means to

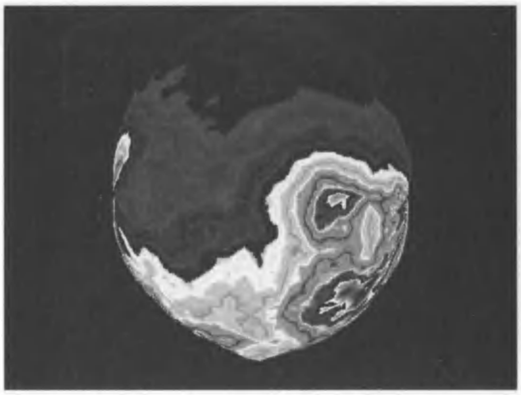


Figure 3-23 Plasma planet

play with 3-D parameters. When you have an image the way you want it, you can apply a slower fill type.

Variation #3: Make a plasma planet. Repeat the plasma mountain instructions, beginning with the (3) command, selecting the same plasma file, and using the (F3) video mode. At the 3D Mode Selection screen, cursor down to the SPHERICAL PROJECTION item and type **yes**. Set the option STEREO (R/B 3D)? (0=NO, 1=ALTERNATE, 2=SUPERIMPOSE, 3=PHOTO) back to 0, and press (ENTER). Continue to press (ENTER), accepting all the defaults for the next screens. The plasma image will be projected onto the surface of a sphere, making a plasma planet, as shown in Figure 3-23. You can project any GIF image onto a sphere in this way, whether or not it originated in Fractint.

At this point we shall leave you to your own devices. You can press (3) again and try some of the other 3D options—a good strategy for learning. One piece of advice, though: just change one or two things at a time so you get an idea of what you are doing! For instance, try combining your plasma landscape with your planet by first creating the landscape, and then adding the planet, using the (#) key ((SHIFT)-(3) on most keyboards) instead of the (3) . The (#) key is just like (3) except the previous image is not erased. You can use (#) to superimpose moons over your plasma landscapes.

COLOR PLATES REFERENCE

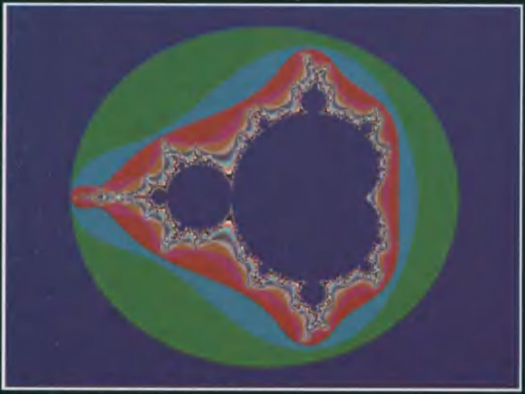
All of the images shown in the Color Plates section are used elsewhere in the book. This reference will enable you to find out how the images were made and even duplicate them on your PC. Color Plates 1 through 25 are discussed in Chapter 4, *Fractal Recipes*. You will find the parameter file entries in the file RECIPES2.PAR on your companion disk. The remaining images are used as examples in Chapter 6, *Fractal Types*. The parameter file entries are located in EXAMPLES.PAR on your disk.

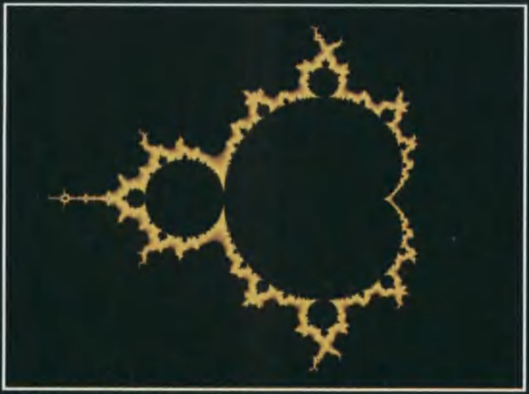
#	Name	PAR Name	Fractal Type
1	Default Mandelbrot	Default_Mandelbrot	mandel
2	Potential Mandelbrot	man001	mandel
3	BOF60 Mandelbrot	man004	mandel
4	BOF60 Mandelbrot II	man012	mandel
5	Silky Mandelbrot	man026	mandel
6	Baby Mandelbrot	man032	mandel
7	Meditating Hermit	Smile_2	mandel
8	Smiling Face	rhs144	fn(z)+fn(pix)
9	Mixed Up Smiling Face	RHS143	fn(z)+fn(pix)
10	Alien Owl	rhs148	fn(z)+fn(pix)
11	Snail Shell	rhs205	lambda
12	Hi Earthling!	HI Earthling..	magnet1m
13	Tentacles	Tentacles	magnet1j
14	Smokie Watches You	Smokie_Watches_you	manzzpwr
15	Aeolis and Janus	AeolisAndJanus	barnsleym3
16	Asteroid Ship	AsteroidShip	fn*z+z
17	Bad Dream	BadDream	newton
18	Bulls Eye Star	BullsEyeStar	fn*z+z
19	Blue Strands On Green	BlueStrandsOnGreen	complexnewton
20	Chains	Chains	julia
21	Egg On Leaf	EggsOnLeaf	fn(z)+fn(pix)
22	Who Pulled the Plug?	Who_pulled_the	frothybasin

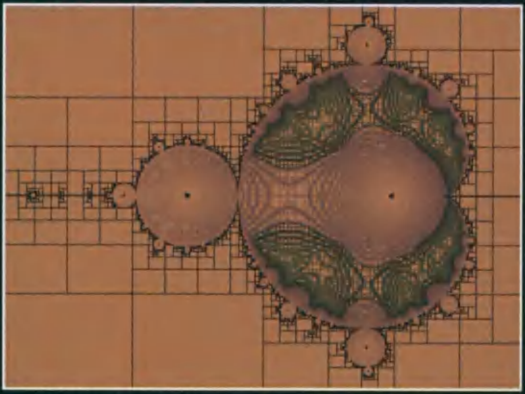
continued on next page

continued from previous page

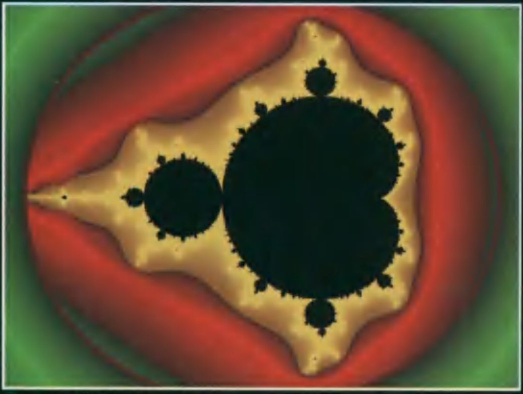
#	Name	PAR Name	Fractal Type
23	Meta Cold!	Meta_Cold!	fn*z+z
24	Magnetic Hole	MagneticHole	formula/halleysin
25	Map Compass	MapCompass	lambdafn
26	Bird and Waves	BirdAndWaves	magnet2m
27	Bracelet	BRACLET	fn+fn
28	CET16104	cet16104	fn*fn
29	CET16109	cet16109	lambdafn
30	Complex Newton One	CMPNWT1.GIF	complexnewton
31	Coral Atolls	CoralAtolls	mandellambda
32	TI6097	t16097	manowar
33	Octopus Breast	OctopusBreast	magnetj1
34	Decomposition 32	decomp32	julzpower
35	Diffusion	diffusion	diffusion
36	Eggzactly!	Eggzactly!	tim's_error
37	Evil Frog	EvilFrog	frothybasin
38	Food Chain	food_chain	barnsleyj2
39	Fractal Angel	Fractal_Angel???	barnsleyj3
40	Genesis Wave	Genesis_Wave	newtbasin
41	Gingerbread Man	gingerbreadman	gingerbreadman
42	Julia's Jewels	Julias Jewels	formula/Jm_14
43	Low Iteration Unity	Lo_Iteration_Unity	unity
44	Lyapunov One	Lyapunov_one	lorenz3d1
45	Mirror, Mirror!	Mirror, Mirror!	barnsleyj1
46	Not Your Usual Fractal 13	NYUF013	kamtorus
47	PAR of the Day 34	pod034	kamtorus3d
48	Purple Kaleido Ring	Purple_KaleidoRing	sqr(1/fn)
49	Quaternion Julia One	QuatJ1	quatjul
50	Really?	Really?	lambda
51	Seashell	SeaShell	julia_inverse
52	Sine Waves	Sin_Waves	dynamic
53	Snowflake	Snowflake	lsystem
54	T16142	t16142	manowarj
55	T17064	t17064	newton
56	The Great Divide	The_Great_Divide	mandel
57	Lorenz Three Lobe	lorenz_three_lobe	lorenz3d3
58	Lorenz Two Lobe	Lorenz_two_lobe	lorenz3d
59	Barnsley Sponge	Barnsley_Sponge	julibrot/barnsleyj1
60	Sand Dollar 3D II	Sand_Dollar_3d_II	icons3d


1—Default Mandelbrot

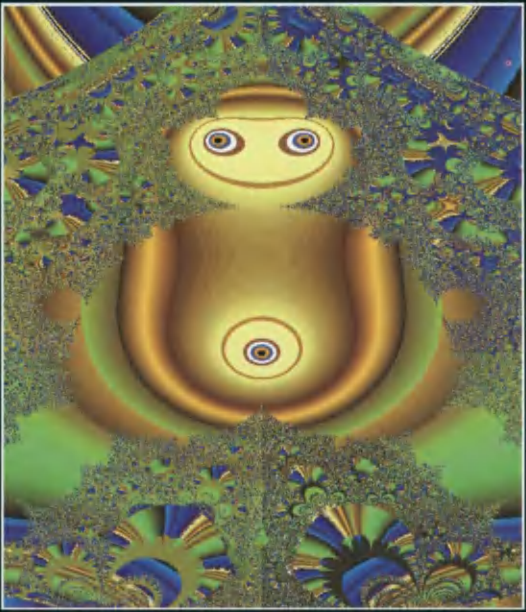

2—Potential Mandelbrot


3—BOF60 Mandelbrot

4—BOF60 Mandelbrot II


5—Silky Mandelbrot

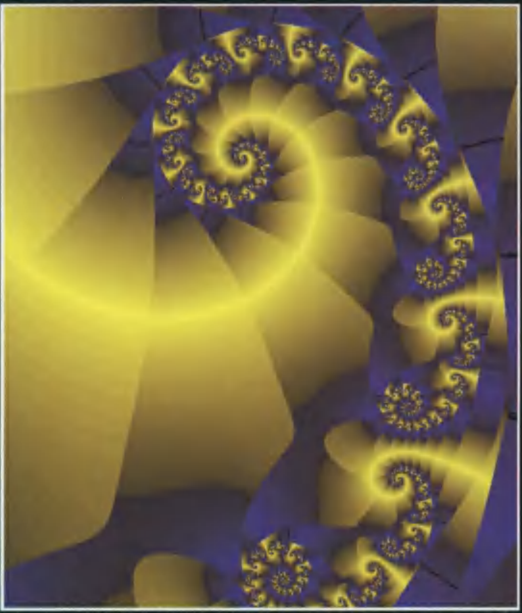
6—Baby Mandelbrot

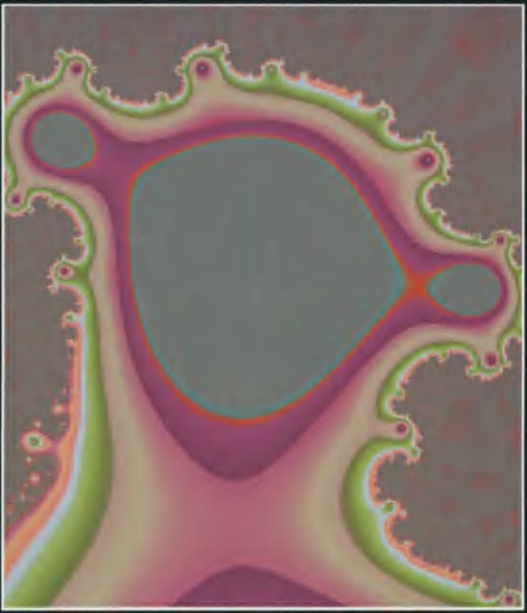

7—Meditating Hermit

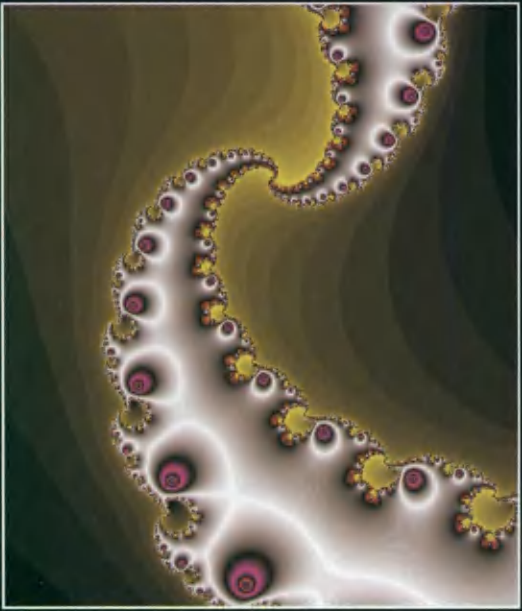
8—Smiling Face

9—Mixed Up Smiling Face

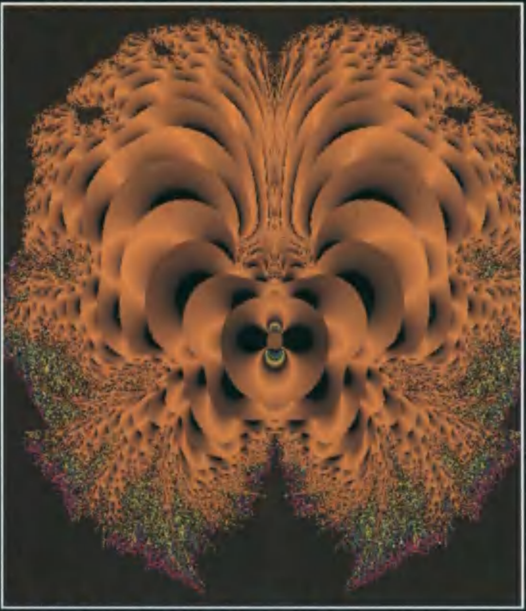
10—Alien Owl


11—Snail Shell

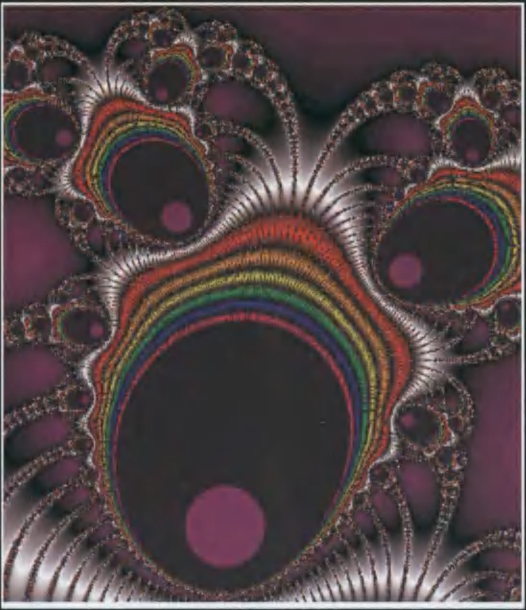

12—Hi Earthling!

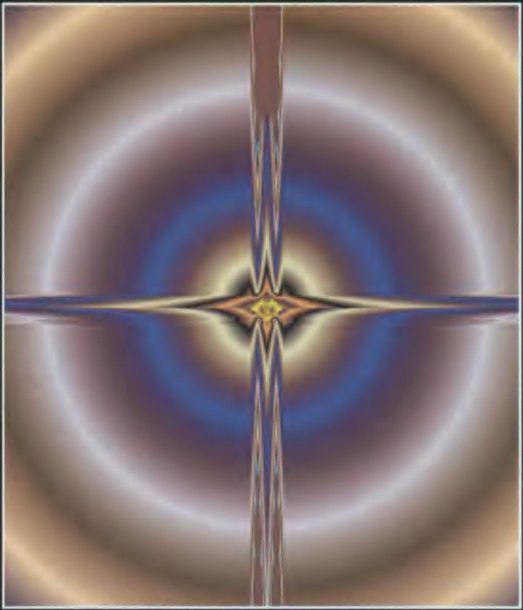

13—Tentacles

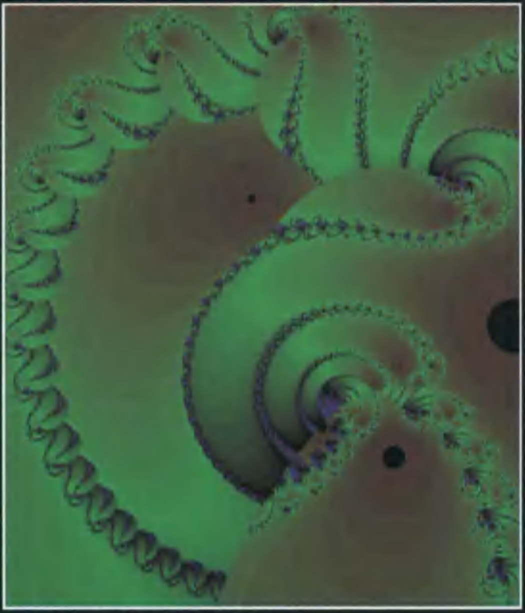
14—Smokie Watches You

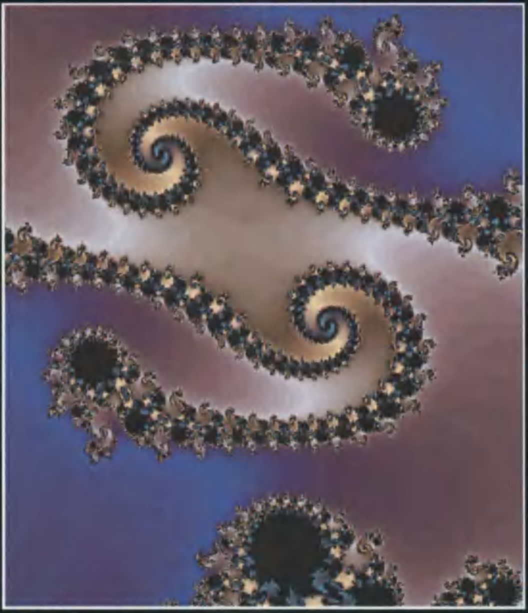

15—Aeolis and Janus

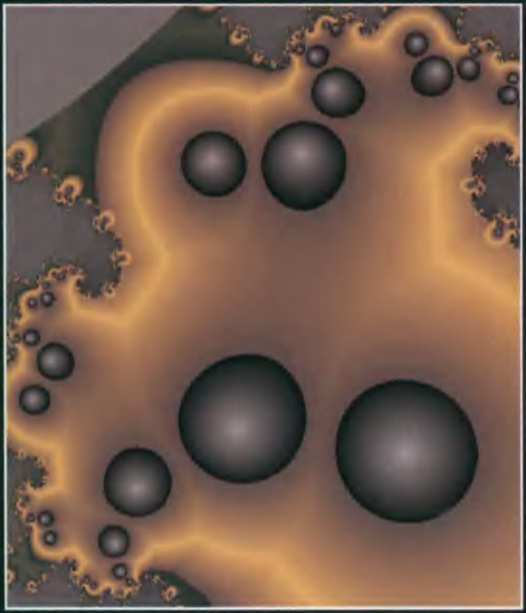
16—Asteroid Ship

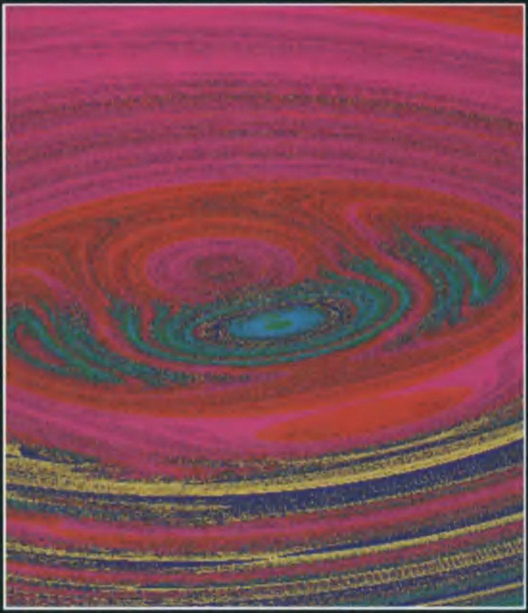

17—Bad Dream

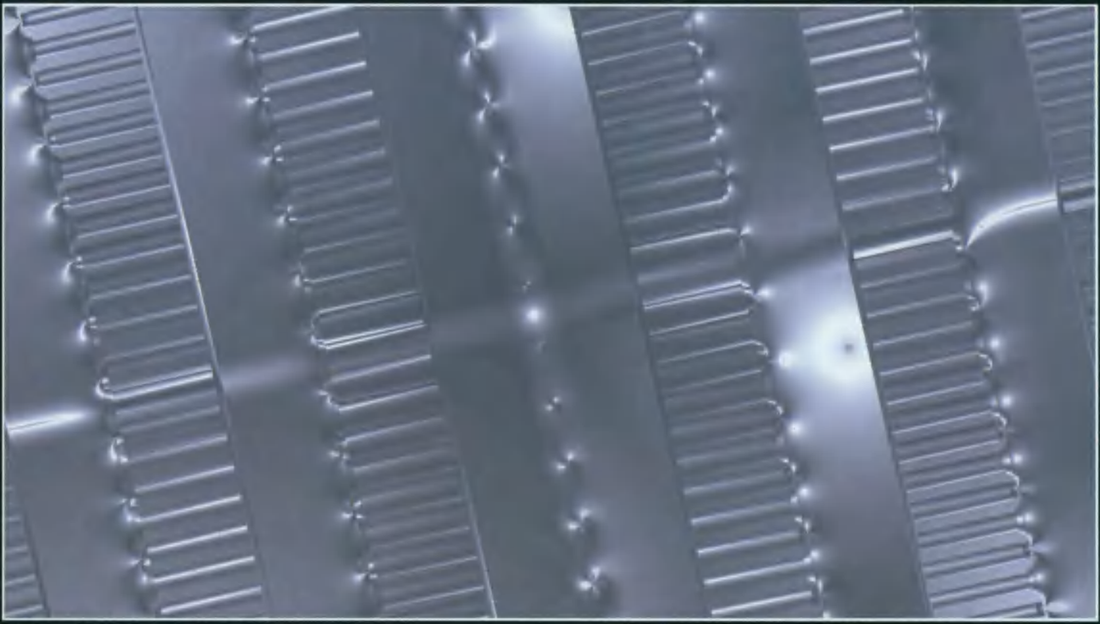

18—Bulls Eye Star

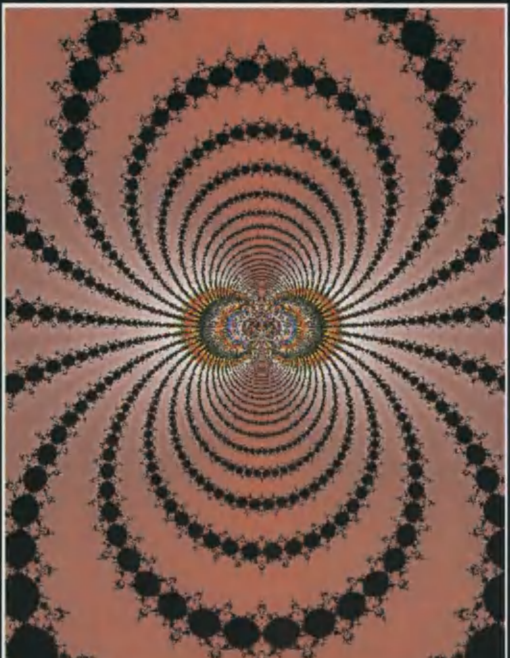

19—Blue Strands On Green

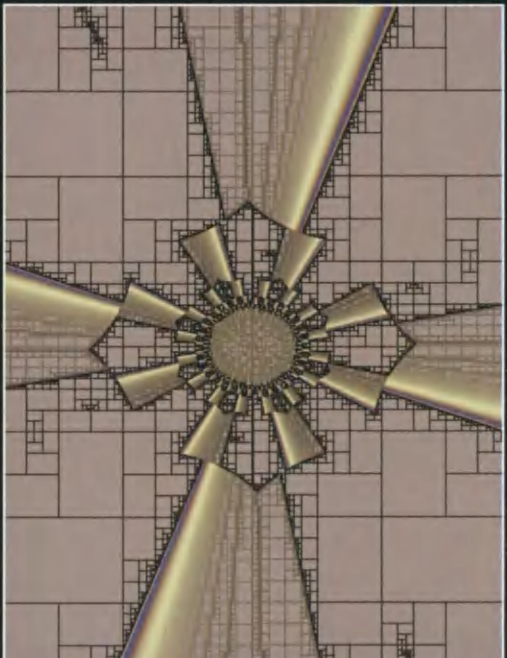

20—Chains


21—Eggs On Leaf

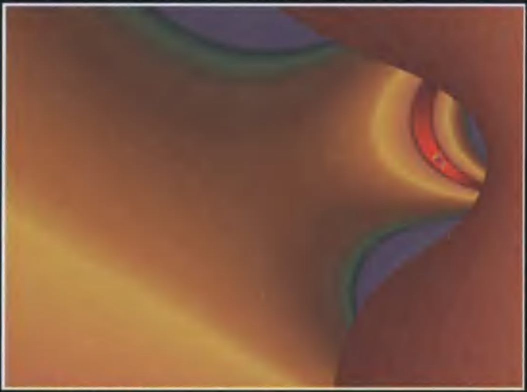

22—Who Pulled the Plug

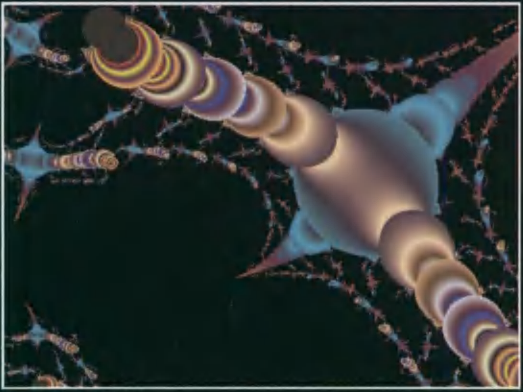

23—Meta Cold

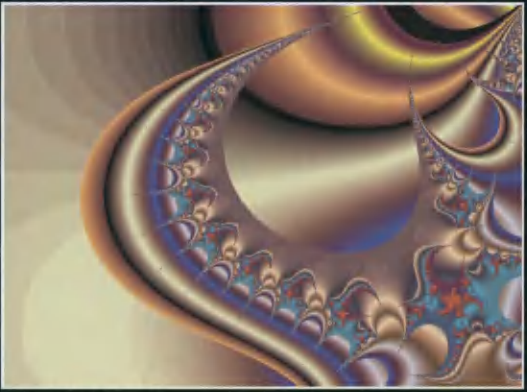

24—Magnetic Hole


25—Map Compass

26—Bird and Waves

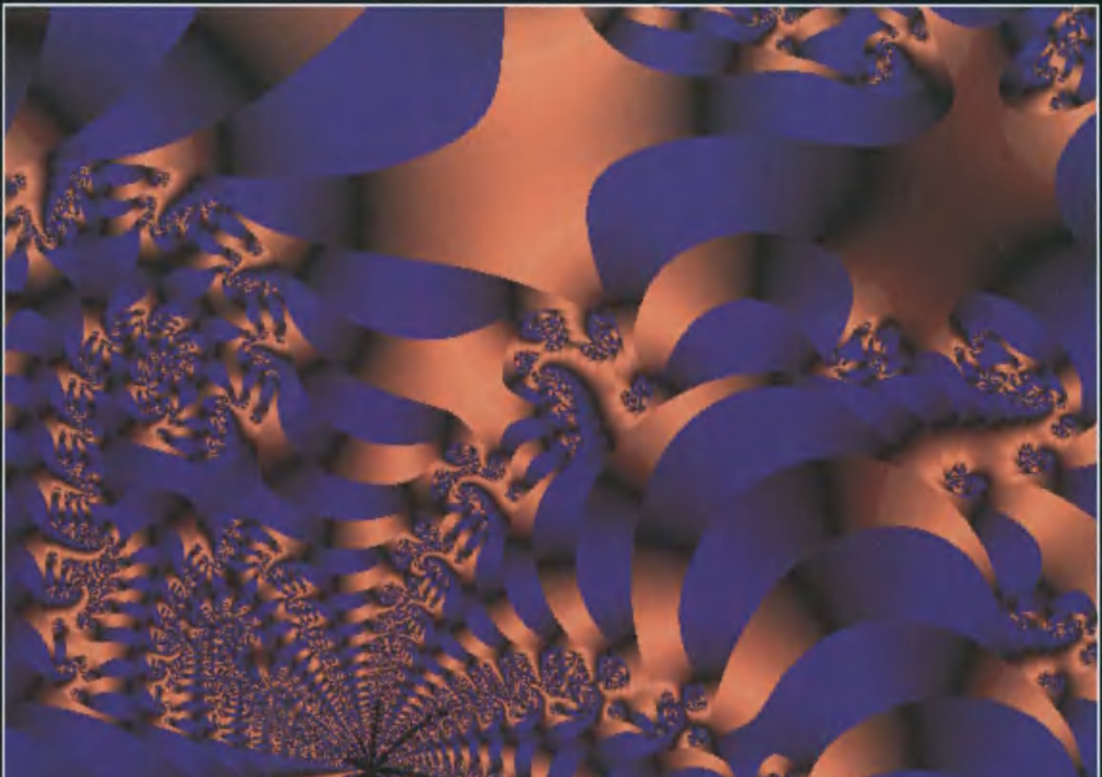

27—Bracelet

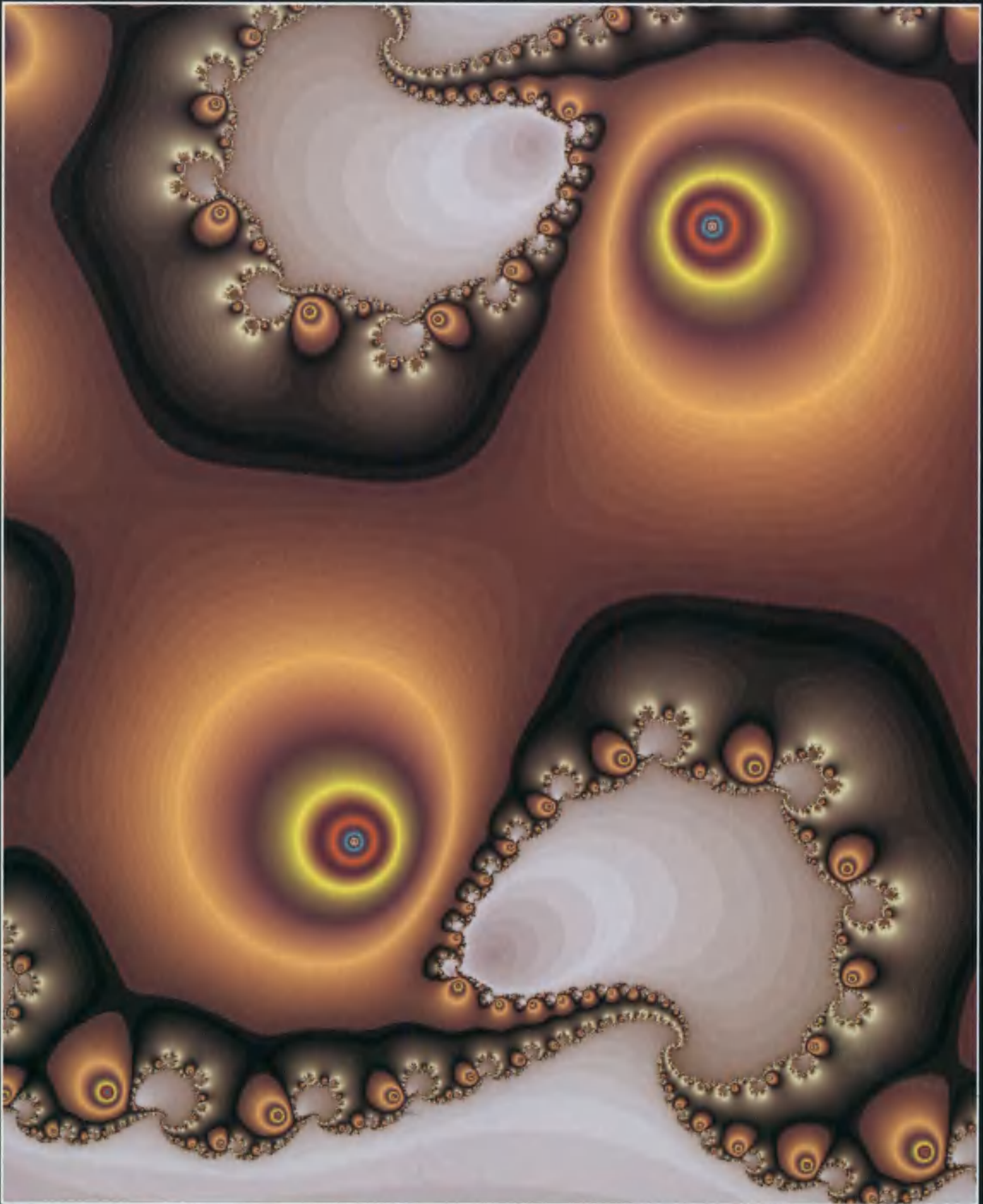

28—CET16104


29—CET16109

30—Complex Newton One

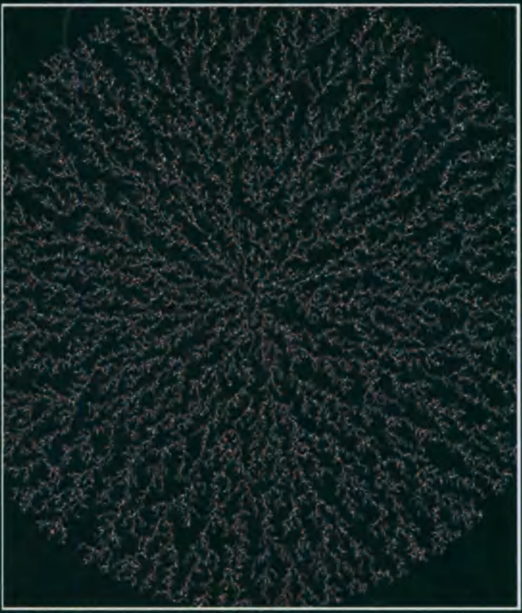
31—Coral Atolls



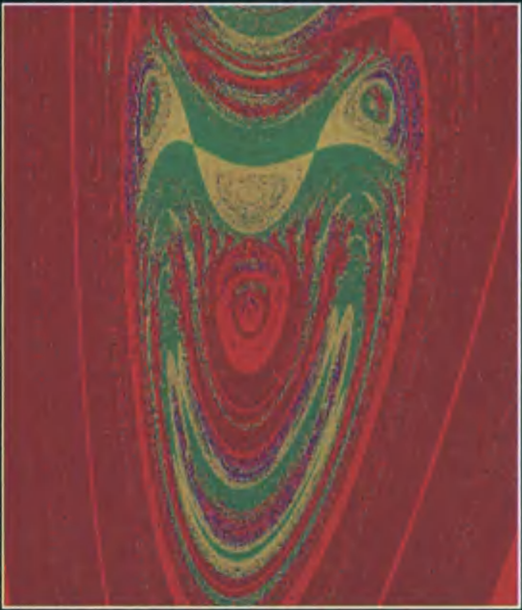


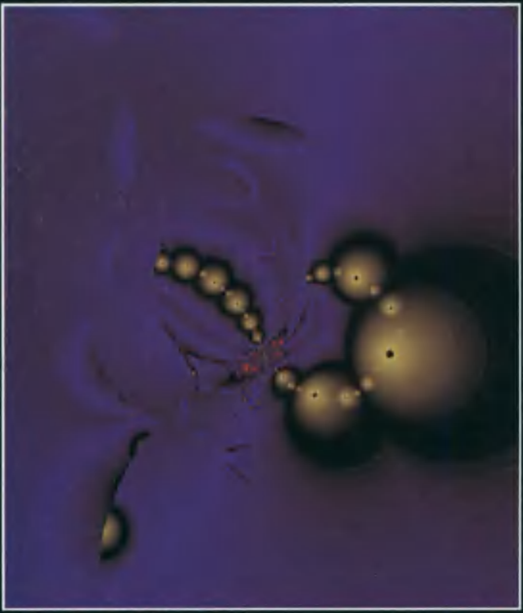
33—Octopus Breast

34—Decomposition 32


35—Diffusion

36—Eggzactly!

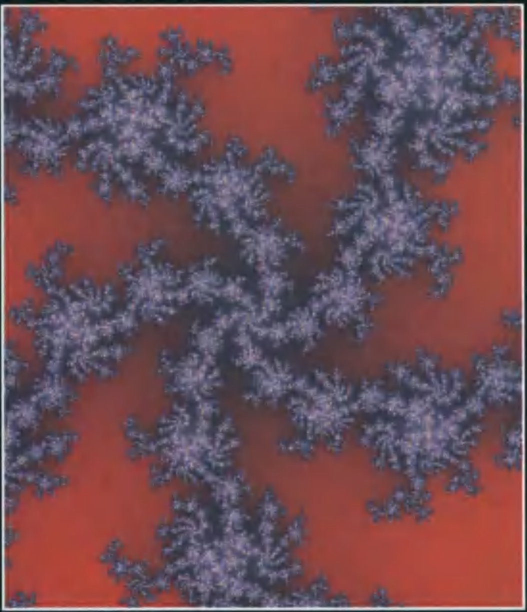

37—Evil Frog

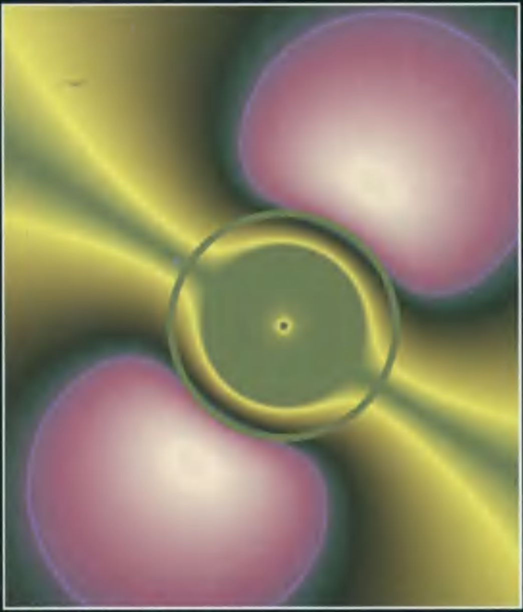

38—Food Chain

39—Fractal Angel

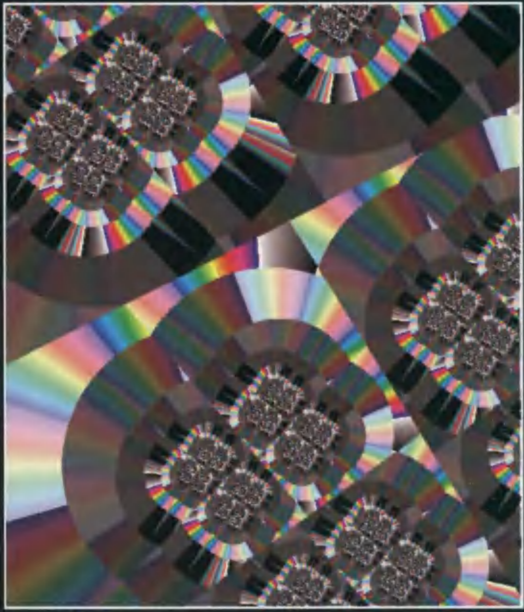
40—Genesis Wave

41—Gingerbread Man


42—Julia's Jewels

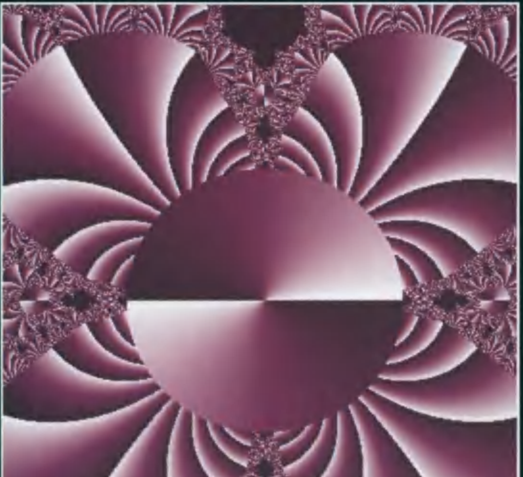

43—Low Iteration Unity

44—Lyapunov One


45—Mirror, Mirror!

46—Not Your Usual Fractal 13

47—Par of the Day 34

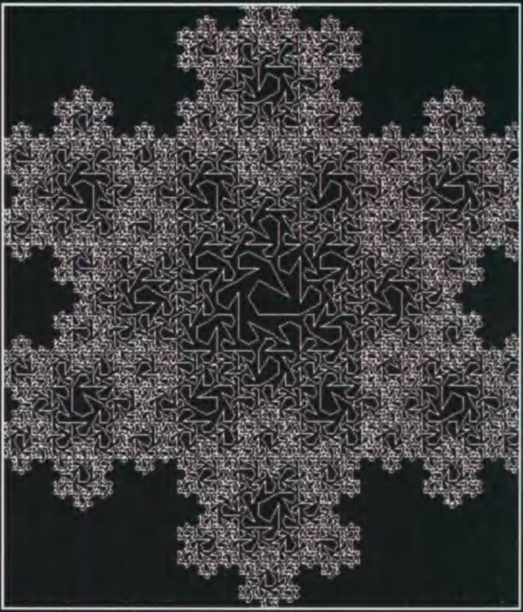

48—Purple Kaleido Ring

49—Quaternion Julia One

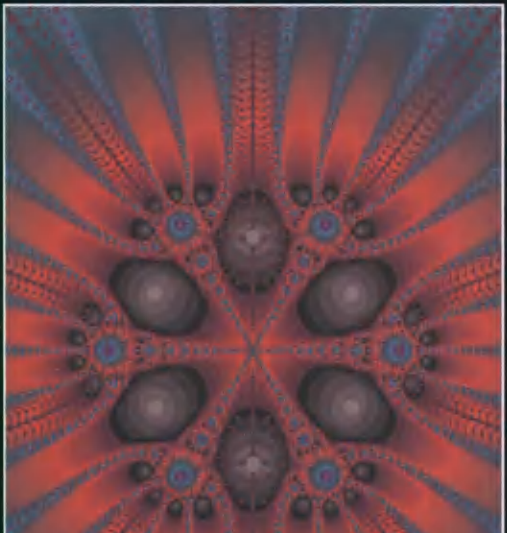
50—Really?

51—Seashell

52—Sine Waves

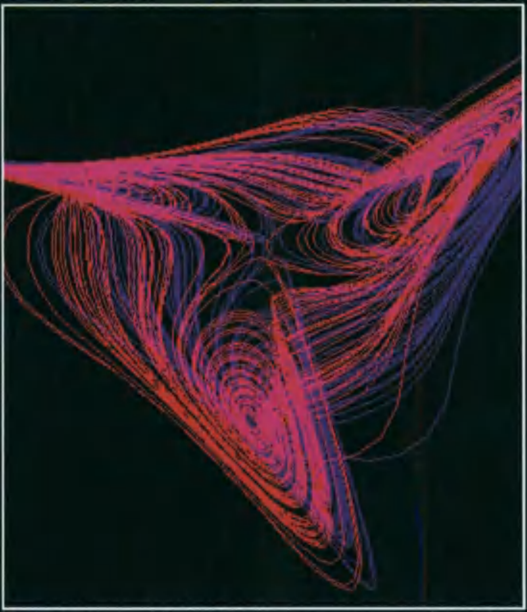

53—Snowflake

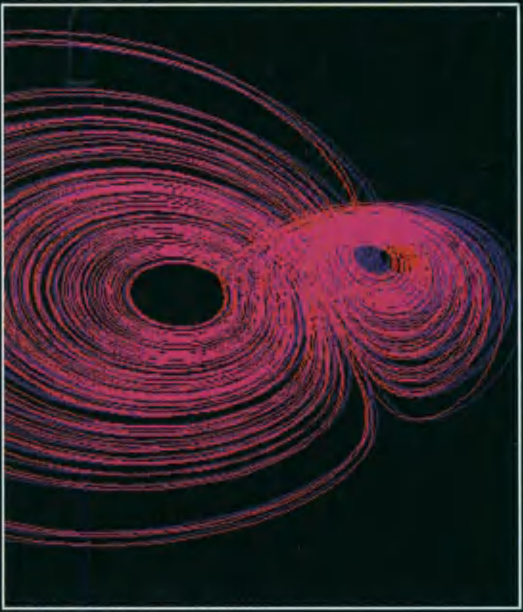
54—T16142



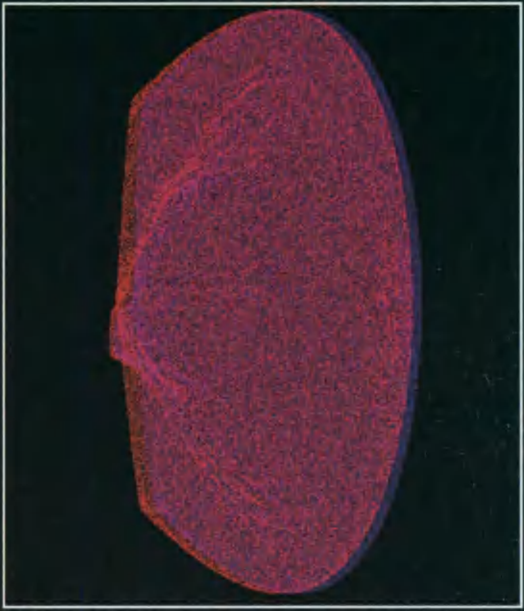
55—T17064

56—The Great Divide


57—Lorenz Three Lobe

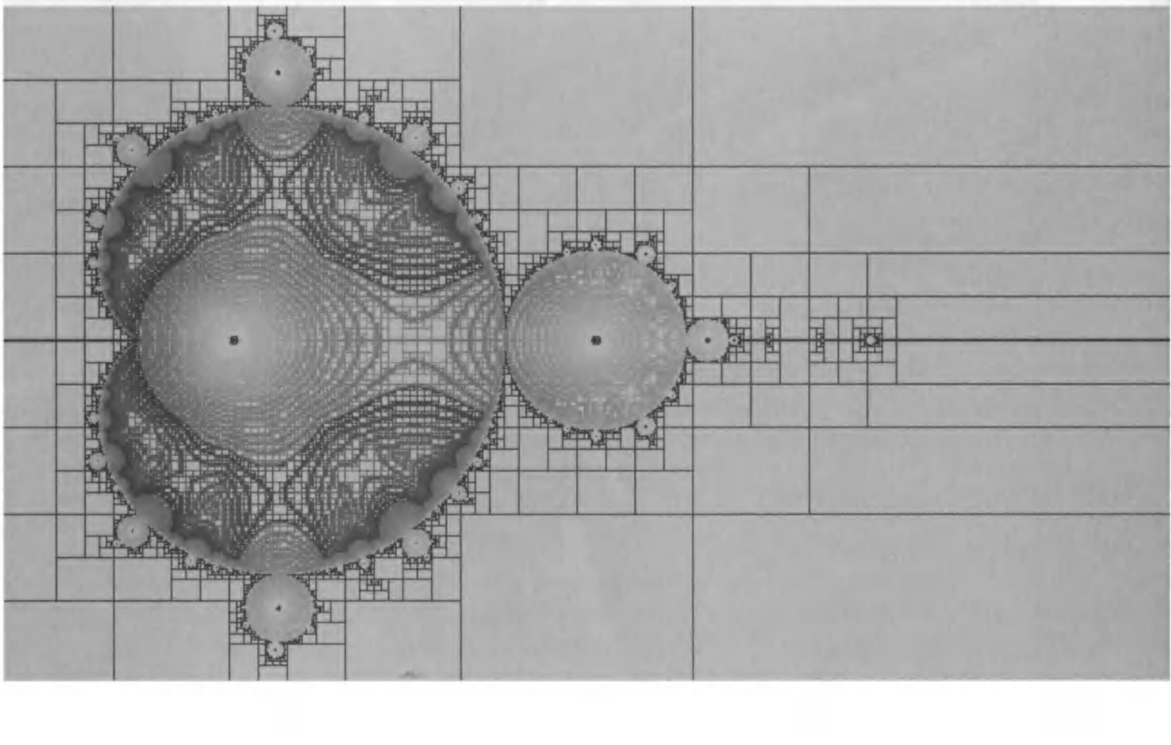

58—Lorenz Two Lobe

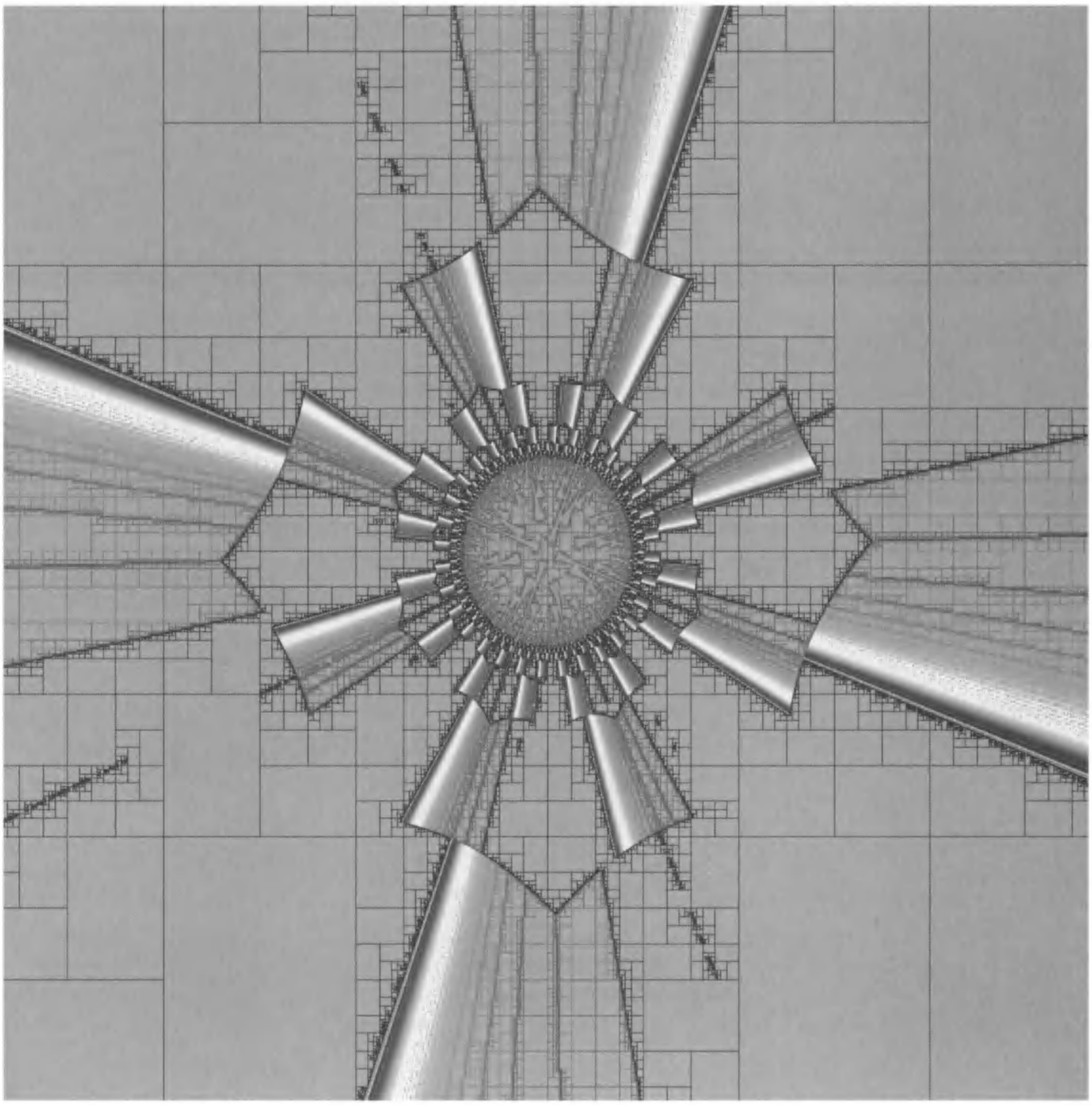
59—Barnsley Sponge


60—Sand Dollar 3D II

CHAPTER 4

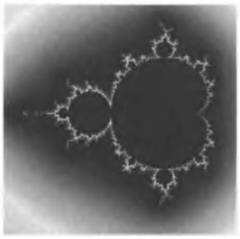
FRACTAL RECIPES





CHAPTER 4

FRACTAL RECIPES



Creating fractals is a little like cooking. You can have a lot of fun using your creativity to make a tasty dish out of whatever ingredients are at hand. Sometimes, though, you yearn for the exquisite taste of a gourmet dish, so you search through your favorite recipe book for a treat invented by some extraordinary culinary genius. Seeing the fractals made by a Fractint expert is a lot like tasting a prize-winning chef's favorite dish. Despite Fractint's multitude of options and possibilities, you may think you have seen everything after many hours of experimenting. Then you try a new fractal recipe and discover a whole new universe waiting to be explored. Some very talented people have used Fractint to make images in their own inimitable style. In this chapter you'll find their best recipes, and you can try them out for yourself!

Your humble authors consider themselves to be programmers more than artists. A synergy exists between the creators of the Fractint program and the artists whose work is featured in these pages. The programmer thinks of some twist of coding or modification of an algorithm, and wonders what visual effect that change would make. The source code is edited, the program recompiled, and a quick test made. Then the programmer's attention moves on to other concerns. Much later that selfsame programmer discovers images of seductive complexity and beauty made by some of these fractal artists and asks, "How on earth were those made?"—only to discover that the images were made with his or her very own code! Fractint's programmers live for the experience of seeing the fruits of their technical labors transmuted into art.

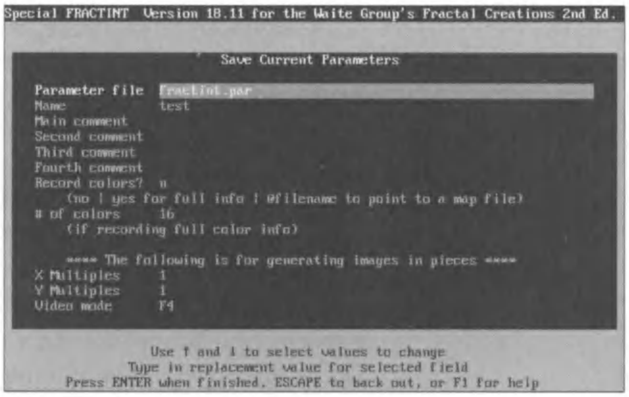
FRACTAL COOKING HINTS

When it comes to creating world-class fractals, there are no hard and fast rules. Each of the artists whose work you will find in this chapter has a different approach to how great fractals are created. Here are a few tips gleaned from the experts.

- Don't jump to conclusions about the initial appearance of a fractal. Fractals that look superficially the same from a "zoomed out" perspective may be extraordinarily different when viewed at high magnifications. Try different degrees of zooming. A fractal does not usually look the same at a magnification of a million as it does at a magnification of one. So do try zooming in to different levels when exploring a fractal.
- Spend time coloring your fractals. You can do this by turning on color cycling with the + key and trying the various function keys. You can also load different color maps by entering the color-cycling mode with C or + and then pressing L. Fractals typically contain far more detail than the eye or the mind can absorb. Assigning the colors differently can completely alter what you see in a fractal image. Later in this chapter you'll see examples of this.
- Experiment with Fractint's basic and extended options (X and Y). The inside and outside coloring options, or the use of features like continuous potential or binary decomposition, can completely alter the appearance of a fractal. Don't forget the fractal parameters (you will see the fractal parameters screen after selecting a fractal type with T or by using the Z key). Remember that you really don't have to fully understand these options in order to test them out! Experiment with them and see what happens!
- Try your hand at inventing your own formula types. You can use a text editor (such as the DOS EDIT application) to edit the file FRACTINT.FRM. The easiest way to invent a whole new fractal is to copy an existing formula type, and change the formula around. Once again, do not be deterred if you have no idea what a complex hyperbolic tangent is. We'll let you in on a deep secret: a bright mathematician having fun with Fractint probably has no more idea than you do of the effect of making a fractal from a formula like $z = tanh(z)/(z^2 + cos(z/2))$.

USING PARAMETER FILES

The best way to share fractal recipes is by using Fractint parameter files, better known as PAR files because their file name extension is .PAR. The PAR format



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

                        Save Current Parameters

  Parameter file    fractint.par
  Name              test
  Main comment
  Second comment
  Third comment
  Fourth comment
  Record colors?    n
      (no | yes for full info | @filename to point to a map file)
  # of colors       16
      (if recording full color info)

      **** The following is for generating images in pieces ****
  X Multiples       1
  Y Multiples       1
  Video mode        F4

                 Use ↑ and ↓ to select values to change
               Type in replacement value for selected field
         Press ENTER when finished, ESCAPE to back out, or F1 for help
```

Figure 4-1 Save @Batchfile dialog box

is a compact way of saving the fractal parameters used to make a fractal. This format is shared by Fractint and the various incarnations of Fractint running on PCs and workstations. PAR files are a great way to share your fractals with others because they are small and compact and still hold all the information needed to reproduce a fractal. You will find them on computer bulletin boards and conferencing systems wherever fractals are discussed. On CompuServe, look in library 4 of the GRAPHDEV forum for many examples.

Writing Parameter Files

When, in the midst of your fractal explorations, you have created a fractal you want to save in a parameter file, press (B) or select SAVE CURRENT PARAMETERS..<B> from the MAIN MENU. You will then see a menu that looks like Figure 4-1. Fill in the name of the parameter file (the extension .PAR is added automatically) and the name you want to give to the fractal. The name of the parameter file entry must be different than any of the names already used in the file; one parameter file can hold many entries, each with a unique name. You also have the option of having the colors written in compressed form in the parameter file, or of using the colors already saved in a separate color map file. Note that if you specify a color map file, Fractint will not automatically create the map file if it doesn't exist, but you can create it in a separate step using the (S) command when in the color-cycling mode. The parameters will be written in a file you can edit.

A parameter file entry looks like this:

```
Hypnoteyes2 { ; by Pieter Branderhorst
  reset type=julia corners=-0.162458/0.17487/0.734422/0.984935
  params=0.258919/1.76951e-007 decomp=256 colors=@blues.map
  }
```

The name of the image is Hypnoteyes2. This is the name the Fractint user sees when opening the parameter file. The curly braces contain all the fractal options discussed in Chapter 3, *Fractint Tutorial*. In this case, the options include the fractal type, the corners values defining the piece of the complex plane delimiting the fractal, the decomposition option, and a color map. You can edit this file by hand, or you can read it back into Fractint, make changes, and save it again. The ";" character (semicolon) indicates that the rest of the line is ignored, and may be used for comments. In this case "; by Pieter Branderhorst" is a comment.

Remember that Fractint saves this same fractal information with the GIF89a format images. You can convert your previously saved Fractint GIF files to PAR files by pressing R or by selecting LOAD IMAGE FROM FILE... <R> from the MAIN MENU to read in the GIF file and then pressing B or selecting SAVE CURRENT PARAMETERS from the MAIN MENU. to save it as a PAR file entry.

Reading Parameter Files

All of the fractal recipes in this chapter are stored on this book's companion disk in PAR files, and should have been installed in your \FRACTINT directory. The easiest way to try the recipes is to read in the PAR file entry. To do this, press @ or select RUN SAVED COMMAND SET <@> from the MAIN MENU and select the PAR file from the file list. (Fractint will automatically use the file called FRACTINT.PAR if it can find it. To change to a different PAR file from the named PAR entry screen, press F6.) You can navigate to different directories by selecting " " to go up a directory, or by selecting a subdirectory name to go down a directory. Once the PAR file is opened, you will be presented with a list of named PAR entries. Select a PAR entry with the arrow keys, press ENTER, and Fractint will go to work generating the image.

THE RECIPES

For each of these fractal recipe images, you'll find the following information:

1. **Name of the Image**: the image name as it is stored in the parameter file.
2. **PAR File**: where the image parameters are stored on your companion disk.
3. **Image Credits**: who is responsible for creating this fractal parameter entry.
4. **Parameter File Listing**: the listing of the parameter file entry.
5. **Generation Time**: seconds required to generate a 640 x 480 image on a 66 MHz 486DX2 machine. Your machine and resolution may be different,

but this will give you an idea how long the fractal takes. If you generate one of these fractals and note the time using the TAB display, you can figure that the other times will be proportionally related.

6. **Formula File Listing**: for type=formula fractals, the FRM file entry.
7. **About the Image**: notes about particularly interesting parameters used and how the parameters affect the appearance of the image.
8. **Variations**: other options to try out with this fractal image.

The Mandelbrot Set

The most famous fractal of all is the Mandelbrot set. Many fractal devotees spend their entire time exploring and playing with just this one fractal. There are two completely different ways of exploring a fractal. The first is to zoom into different parts of the fractal and explore the vast fractal terrain at different magnifications. This is the most common and most readily understandable method of exploration. A completely different approach is to experiment with different methods of rendering the colors of the fractal. Both of these techniques will be demonstrated in this section.

The Mandelbrot set is a set of points in the complex plane. The colorful default image of the Mandelbrot set that you get by starting Fractint and pressing F3 is not so much a picture of the Mandelbrot set itself as one way of visualizing the dynamic system used to define the set. The Mandelbrot set consists of the blue area in the middle of the image. The colorful stripes surrounding the "lake" are a graphical representation of the escape time of the iterated function $z^2 + c$ used to define the set (how many iterations it took for the orbit generated by that function to escape a circle of radius 2). There are many other schemes for assigning colors to point inside or outside of the Mandelbrot set. Most of these depend in some way on the dynamics of the whirling orbits determined by $z' = z^2 + c$. Other effects are the results of algorithms to do with the order in which the points are plotted.

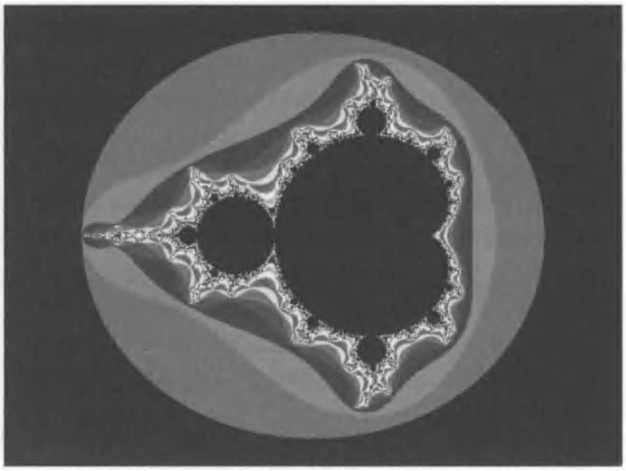


Figure 4-2 Default_Mandelbrot

Mandelbrot

PAR File: RECIPES2.PAR

Image Credits: Benoit Mandelbrot (who else?)

Parameter File Listing:

```
Default_Mandelbrot         { ;This is as simple as it gets
  reset type=mandel
  }
```

Generation Time: 4 seconds

About the Image: Figure 4-2 and color plate 1 show the classic Mandelbrot escape-time image. There are two minor differences between this image and other Mandelbrot images you may have seen. Both of these differences are due to decisions made early in the life of Fractint. The first is that the outer two escape-time bands have been combined into the blue background. Therefore, the green ring that appears to be the second escape-time ring (the points that escaped a circle of radius 2 after two iterations) is really the third. This difference is due to a programming speedup that was irresistible to the performance-oriented programmer. Since $0^2 + c = c$, one iteration of computation is saved by starting the orbit calculation $z' = z^2 + c$ with c instead of zero. The second difference is that the Mandelbrot lake is often assigned the color black, but Fractint's original programmer felt that blue was a more appropriate color for a lake. (You can change the color of the lake to black by pressing $\boxed{X}$ to get the BASIC OPTIONS menu and setting the INSIDE parameter to 0.)

The `reset` keyword in the parameter file tells Fractint to reset all parameters to the default values before reading the file. This is not absolutely necessary, but without it the results may vary if you have changed some of Fractint's settings. Beginning a PAR file with `reset` is standard practice. PAR files generated by Fractint with the (B) command always begin with `reset`.

Variations: You can see the missing outer escape-time ring by generating the Mandelbrot set using fractal type test. Press (@), select RECIPES2.PAR, and select TRUEMANDELBROT.

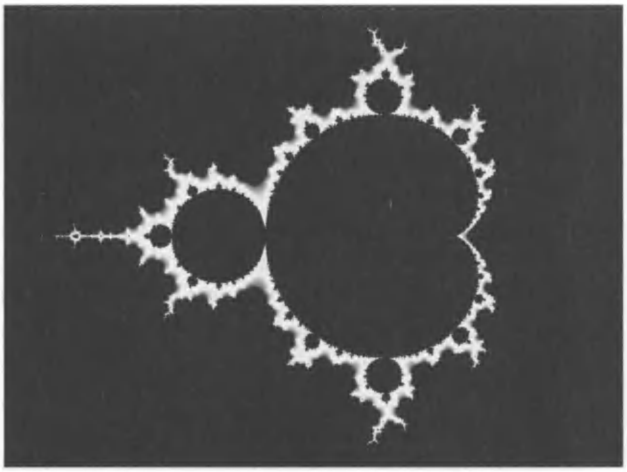


Figure 4-3 Potential Mandelbrot

Potential Mandelbrot

PAR File: RECIPES2.PAR

Image Credits: Richard H. Sherry

Parameter File Listing:

```
man001            { ; (c) 1993 Richard H. Sherry, 76264,752
                    ; Par series based on Mandelbrot classic lake
  reset type=mandel passes=1 corners=-2.14/1.14/-1.23/1.23 float=y
  maxiter=255 inside=255 potential=255/2000/1000
  colors=000<200>000ZK0<45>xpAxpAwoA<3>tl9000
  }
```

Generation Time: 16 seconds

About the Image: The escape-time method of rendering fractals results in an inherently discontinuous image. That's because the regions outside the fractal set are divided into regions of identical escape times. The continuous potential method is based on the idea of graphing the potential field that would be created around a charged Mandelbrot set. For our purposes, this physical interpretation is less important than the fact that the continuous potential method results in smoothly varying colors rather than striped bands.

The line of the PAR file that does this magic is the line `potential=255/2000/1000`, as Figure 4-3 and color plate 2 show. The first parameter is the maximum color value, the second parameter is the slope, and the third parameter is the orbit bailout value. Of these parameters, the one to experiment with is the slope. A higher value makes the colors change more rapidly. In this case, Richard Sherry has intentionally chosen a "too high" value so that the entire range of 256 colors is compressed into a band just outside the Mandelbrot set. In a more normal use of potential, the goal would be to spread out the colors to cover the entire image by using a lower slope. The result of using a higher value is an image that is deceptive; it looks very much like a distance estimator method that thickens the tiny filaments emanating from the Mandelbrot and makes them visible.

If you are new to Fractint, you might wonder about the mysterious `colors=` line of the PAR file. Fractint uses a method of encoding the 768 values of a VGA color palette into a few characters. This line is not meant to be understandable by humans, but Fractint understands it very well. You can convert this color palette to a form you can understand by generating the image, entering the color-cycling mode using Ⓒ, and saving the map file with the Ⓢ command.

Variations: Try color cycling this image by pressing ⊕. You will see the black outline of the Mandelbrot set grow larger and slowly lose its shape as it traces the outer escape-time contours.

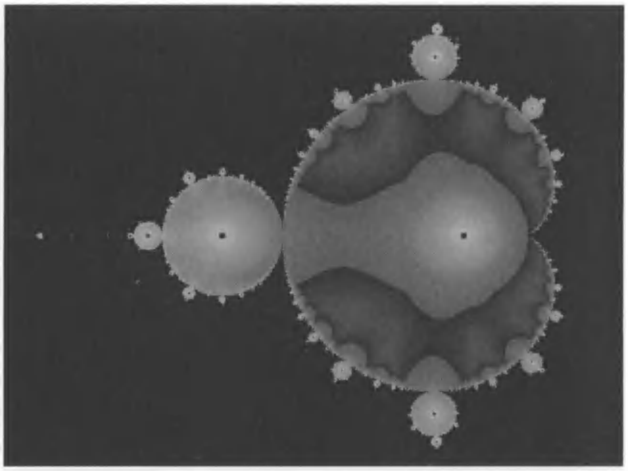


Figure 4-4 BOF60 Mandelbrot

BOF60 Mandelbrot

PAR File: RECIPES2.PAR

Image Credits: Richard H. Sherry

Parameter File Listing:

```
man004            { ; (c) 1993 Richard H. Sherry, 76264,752
                    ; Par series based on Mandelbrot classic lake
  reset type=mandel passes=1
  corners=-1.905194/0.654814/-0.959991/0.960002 maxiter=500
  inside=bof60 outside=0
  colors=000jVD<17>PFF0A0<15>Zc0<14>0A0<15>Zc0<15>0A0SNC<29>xo'zpaypa<29>U\
  PF000<29>nnnpppooo<30>000PFF<29>xeDzfCyfC<11>kWD
  }
```

Generation Time: 1 minute 20 seconds

About the Image: Until now all of the Mandelbrot images we have discussed have dealt with the area outside of the Mandelbrot set. The interior of the Mandelbrot set has been rendered with a solid color. The orbits used to generate the Mandelbrot set exhibit complex dynamics that vary within the set. Any method that can render these dynamics with colors that are dependent on the chaotic dynamics of the orbit will reveal structure within the Mandelbrot. One such method uses Fractint's `inside=bof60` option, so named because it is discussed on page 60 of our edition of the classic book *Beauty of Fractals*. This option is discussed in Chapter 5, *Fractint Reference*, in more detail. For now let this suffice: the interior

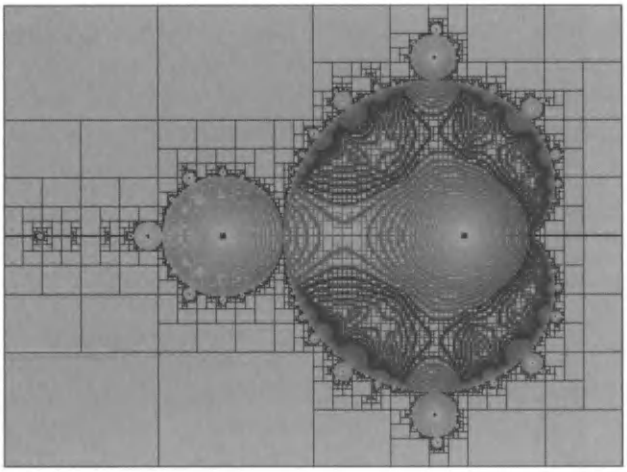


Figure 4-5 BOF60 Mandelbrot using passes=tesseral

of the set is colored according to how close to the origin the orbit associated with that point comes. The result is shown in Figure 4-4.

Note the `outside=0` option. This sets the entire exterior of the Mandelbrot set to the color black. Any fractal has too much information for the eye and the mind to grasp, so some method of focusing attention is needed. It is possible to combine methods that show structure in both the interior and the exterior of the Mandelbrot set. In this case simple is better, and the fact that the outside has been rendered black makes it easier to focus on the beauty revealed in the Mandelbrot interior.

Variations:

An interesting effect (not for mathematical purists!) is to set `passes=t` and `fillcolor=1`. The tesseral option is a divide-and-conquer algorithm that recursively subdivides an image into rectangles; and if the border is a solid color, the interior of the rectangle is filled in. Normally, this makes no difference to the final image, but setting `fillcolor=1` causes the interiors of the rectangles to be filled in with color number 1 rather than the color of the rectangle boundary. The effect is to leave a lattice of rectangles as an artifact of the computation method. This effect shows off the structure inherent in the BOF60 very well. Figure 4-5 and color plate 3 show the result. Be sure to try rotating the colors with ⊕.

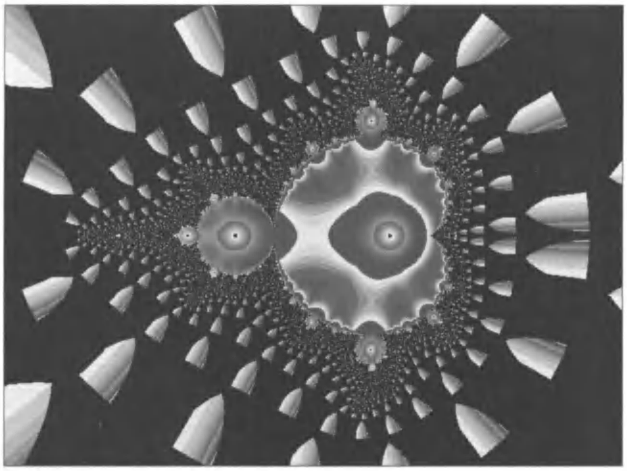


Figure 4-6 BOF60 Mandelbrot II

BOF60 Mandelbrot II

PAR File: RECIPES2.PAR

Image Credits: Richard H. Sherry

Parameter File Listing:

```
man012              { ; (c) 1993 Richard H. Sherry, 76264,752
                      ; Par series based on Mandelbrot classic lake
  reset type=mandel passes=1 corners=-2.5/1.500012/-1.499989/1.5
  maxiter=125 inside=bof60 outside=mult biomorph=0
  colors=000eib<3>JQNDLJ5_n<4>8JL1Cr<3>7GNxfd<3>ILLQCc<2>CFMVHH<4>EGGBGG9j\
  e8VTipUXcPKSKcRe<6>CHJxa4<5>FJFHZBJ44<3>AEEhMBWKDKIFhnV<4>ELIhG7<6>CGFiI\
  k<3>FGMhJk<2>HGOlpm<6>DKKyiRBmh<2>80NM_9<4>AJFlf6<4>EKFhW60cw4Savl3XWAul\
  FdaGhlEhq7c3v<6>8b5<6>mJ9<6>lHR<4>BWI<6>xMo<5>_BDkJ7vQ1<2>Isd<2>Ce0<6>QS\
  r<5>gcridrjYo<3>lAe<3>d7Ec77_A8<2>PJ9<3>U1tha9VaLIaX8GHEURBNLIuB<4>9NG7c\
  t8S_I53<3>AEEGkOCWKSk8<2>DOE80P<3>8DHhWE<4>EIGdNYOJPaz3NbAPVV<2>CJJ1CO5E\
  KLcN<6>9JGqtjknf
  }
```

Generation Time: 26 seconds

About the Image: This image uses three special options in tandem. The interior of the Mandelbrot set is colored using `inside=bof60` as before. The exterior is a combination of the `biomorph=0` and the `outside=mult`, which factor in the direction of the orbit and the real and imaginary components of the last orbit value along with the iteration number (see Chapter 5, *Fractint Reference,* for more details). The result is shown in Figure 4-6 and color plate 4.

Variations: Try different combinations of `outside=` and `biomorph=` options, such as `outside=iter` with `biomorph=0`.

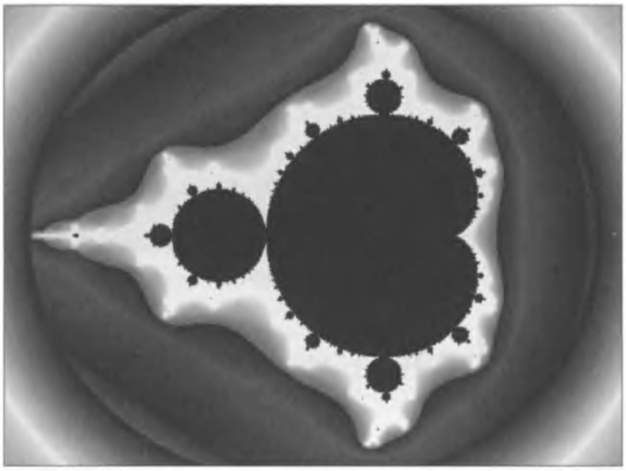


Figure 4-7 Silky Mandelbrot

Silky Mandelbrot

PAR File: RECIPES2.PAR

Image Credits: Richard H. Sherry

Parameter File Listing:

```
man026             { ; (c) 1993 Richard H. Sherry, 76264,752
                     ; Par series based on Mandelbrot classic lake
  reset type=mandel corners=-2.139999/1.140009/-1.229988/1.230001
  maxiter=500 inside=0 potential=255/300/0
  colors=000sl9<40>QK00000F0<44>FxOGzOGyO<31>0F0F00<45>x00z00y00<30>F00ZK0\
  <45>xpAxpAwoA<2>um9 cyclerange=0/255
  }
```

Generation Time: 32 seconds

About the Image: This image revisits the `potential=` option. This time a much lower slope value is used (300 compared to 2000). The Mandelbrot exterior has been transformed into smooth silky cushions with a pseudo 3-D effect shown in Figure 4-7 and color plate 5.

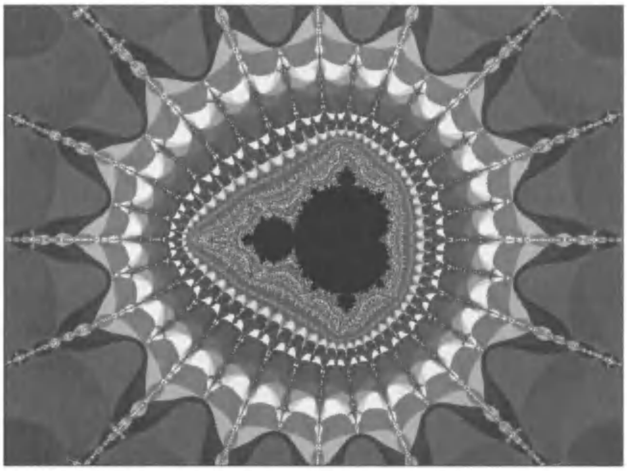


Figure 4-8 Baby Mandelbrot

Baby Mandelbrot

PAR File: RECIPES2.PAR

Image Credits: Richard H. Sherry

Parameter File Listing:

```
man032            { ; (c) 1993 Richard H. Sherry, 76264,752
                    ; Par series based on Mandelbrot classic lake
  reset type=mandel
  corners=-1.930068861596/-1.930068528796/-0.000000121619/0.000000125717
  float=y maxiter=500 inside=0 periodicity=0
  colors=000USQJNQSzg<2>DTUjK0<2>IIQ4Y1gn2QYEz6M7D0KhDEVKQCEr1UWASBGzAHgTK\
  ITF5MGCFHJpZqW4s0ae<2>CNU8YL9Q0CfYBYVAQSm9ETEK2g26VECF4AHFj1l<2>IEVb9'I3\
  ADBIlRmTMa'CpOLmill<2>IPVIN8FLECJK3Au6EeCunc2VT8TJDRD2p<2>AEW'dygRanQc3Q\
  '<2>8KSYSf<2>FKUv81<2>LGKRJ8IIHBqs<2>9RXBrCJLg<2>BIUxH0<2>MIQhUu<2>ILYP4\
  90prGZc3RB50G7LLN_SIURD0QjTQlFITHMKmW0vh<2>CSUB33A8B9DJ1M75KHKgSMEE3pB<2\
  >8QNV'kKR'NA4GEF7Rk<2>9KVtAnEIi<2>AIVSGK<2>DIPb760DGK9iEE_o8aUDW3Qq5Ng7K\
  Zpu1vzRYcQJWn<2>BLW4xvZo5QcCHTJ80v<2>9JY0tW<2>7RReKHPJMNa_GSVPDXDIZ1M'<2\
  >GpH<2>0_c<2>754PdM<3>dqHB0dF8XIGPLNH<3>SFp<3>l27JILd"UUXJ0TkNkZLcMJXQM\
  PKKQEJQvvtYad0Yr5Qc1cWRZT<2>DMQovx<2>JSY95waHlSIdIIXQ4TK9SEER0gvJZkEQ'En\
  B<2>AQNbNv0KevAd<2>LGTdYQ
  }
```

Generation Time: 20 seconds

About the Image: This baby Mandelbrot, shown in Figure 4-8 and color plate 6, lives in the depths of the Mandelbrot at a magnification of 8,086,166. It is one of an infinite number of such self-similar subsets of the Mandelbrot that exist along the left-hand spike of the Mandelbrot.

Variations: Try zooming farther, and see if you can find still more baby Mandelbrots at even deeper resolutions. A magnification of eight million is nowhere near Fractint's limit, which exceeds 4,000,000,000,000 (that's four *trillion!*).

Critters

Fractals could really be used for Rorschach tests. Different people see very different things in the same fractal, and undoubtedly a learned psychiatrist could attach much profound meaning to what you see. However, the next few examples are clearly critters, beyond a shadow of a doubt!

If you can, get an imaginative six or seven year old to help you name your fractals. At that tender age the obstacles preventing direct perception of what fractals *really* are has not yet developed!

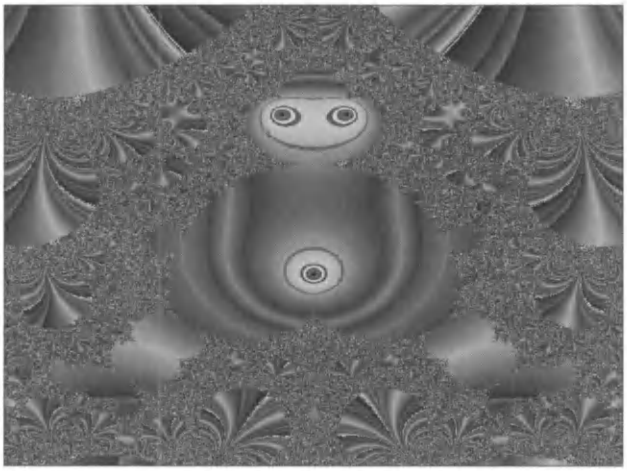


Figure 4-9 Meditating Hermit

Meditating Hermit

PAR File: RECIPES2.PAR

Image Credits: Richard H. Sherry (and the Compuserve Graphdev gang)

Parameter File Listing:

```
Smile_2           { ; (c) June 1992, Dick Sherry 76264,752
  reset type=formula formulafile=fractint.frm formulaname=Richard5
  corners=-1.750391/1.249622/1.496003/-1.495989/1.249622/-1.495989
  inside=bof60 decomp=256
  colors=000UM0AGRcccKF5'eI_dI<2>X'GPK5VXE<13>FA0<6>YR0<7>BA0GG0<11>KV0PM0<17>\
  FD0KI0<31>HiBHiBHhB<78>A506Pz<24>46K4Gz<39>00FWlGVkG
  }
```

Formula File Listing:

```
Richard5 (XAXIS) {; Jm Richard-Collard
   z = pixel:
   z=sin(z*sinh(z))+pixel,
    |z|<=50
  }
```

Generation Time: 10 minutes

About the Image: Who says fractals can't be funny? Figure 4-9 and color plate 7 show a strange creature with a blissful smile and a glowing navel. Who is this little guy? A meditating hermit? A sumo wrestler? A sun bather? And this from an arcane fractal made from a formula using the complex sine and complex hyperbolic sine functions!

If you are not familiar with Fractint's user-defined fractals, the Richard5 formula is a good example. The variable *z* is initialized to "pixel," which means the complex number associated with the pixel on the screen about to be colored. The formula to be iterated replaces *z* with sin(z×sinh(z)) + *pixel* each iteration, until the condition |z|<=50 is true. (Remember, the |z| operator is the "programmer's absolute value" rather than the mathematical absolute value. The expression |z| in the formula parser language means $|z^2|$ in mathematical terminology.)

The outline shape of the meditating figure is the region of the Richard5 fractal where maximum iterations are hit. The `decomp=256` option colors the surrounding background according to the pie slice where the just-escaped orbit value lands out of 256 pie slices. The `inside=bof60` is responsible for the details of the mysterious creature's face and body. Recall that this option colors pixels according to the closest distance the orbit swoops to the origin.

Variations: Try generating the default Richard5 fractal. To do this, press (T) to get the fractal types screen, select the formula fractal type, and then select the Richard5 fractal. In the center you will see the horizontal form of our friendly meditating critter, without any facial or body features.

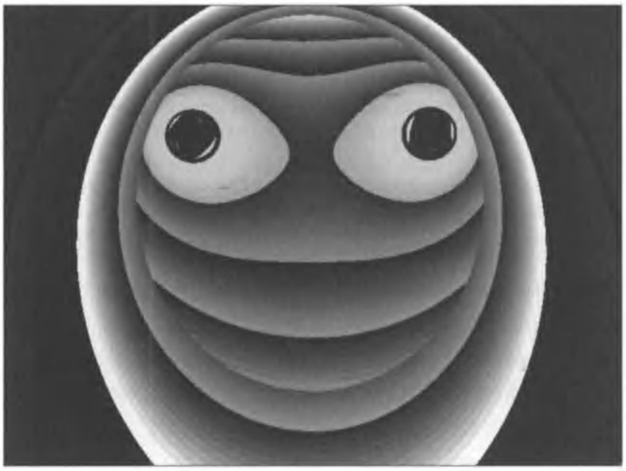


Figure 4-10 Smiling Face

Smiling Face

PAR File: RECIPES2.PAR

Image Credits: Richard H. Sherry (and the Compuserve Graphdev gang)

Parameter File Listing:

```
rhs144          { ; (c) June 1992 Dick Sherry 76264,752
                  ; no commercial use w/o permission
  reset type=fn(z)+fn(pix) function=sqr/cos
  corners=-1.066106/-0.517546/2.981645/3.122173/-0.936088/2.808271
  params=0/0/1 maxiter=32000 inside=0 potential=255/500/20
  colors=000B7C54693607n<13>503frT<10>842qP3<11>810Kd0<15>880MdU<9>9pc7re7re<1\
  9>5td000<60>0005Va<6>534JJN<2>845CoP<11>541H8G<19>500ja'<2>F99JGd<11>613qXY<\
  7>Ydw000<19>000kSq<5>GAI
  }
```

Generation Time: 1 minutes 8 seconds

About the Image: For another smiling critter, see Figure 4-10 and color plate 8. This one appears magically within the fn(*z*) + fn(*pix*) fractal type using sqr and cos as functions. This fractal type has been generalized using Fractint's function variables, which allow 16 different functions to replace different variables. Because fn(*z*) + fn(*pix*) has two such variables, it is really 256 different fractal types disguised as one.

Variations: So where did this smiling fellow come from? To find out, press @, select RECIPES2.PAR (use F6 to change PAR files if Fractint finds and opens

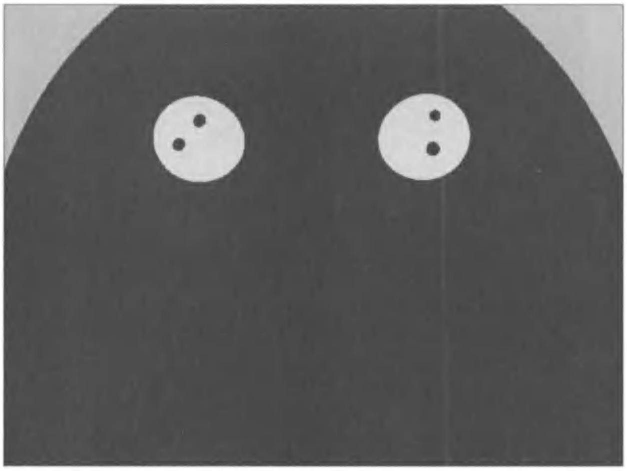


Figure 4-11 Smiling Face without potential

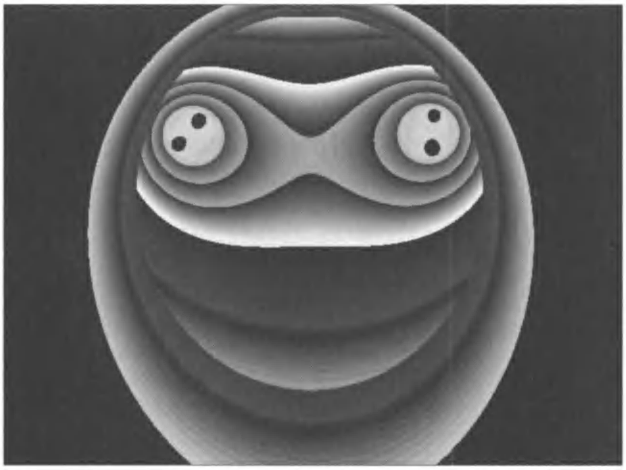


Figure 4-12 Mixed up smiling face

FRACTINT.PAR). Without waiting for the fractal to generate, press (Y) to access the EXTENDED OPTIONS,change potential to 0, and press (ENTER). Instead of the smiling face, you will see two blank eyes staring at you, shown in Figure 4-11. The structure of the face comes from the continuous potential color rendering that you just turned off. For another variation, try PAR entry RHS143. This is the identical fractal except for the color palette. This smiling face looks a bit confused, as you can see in Figure 4-12 and color plate 9.

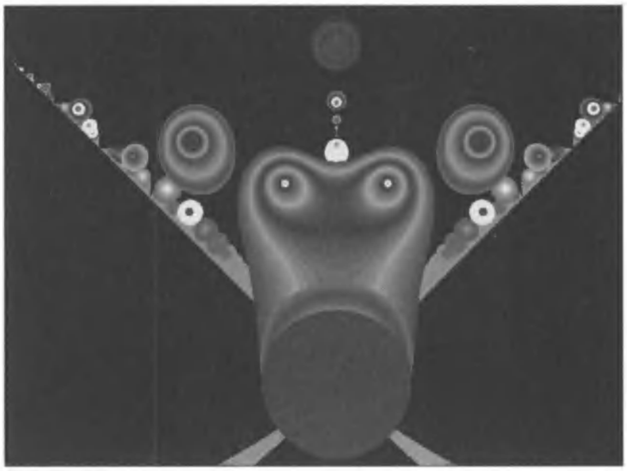


Figure 4-13 Alien Owl

Alien Owl

PAR File: RECIPES2.PAR

Image Credits: Richard H. Sherry (and the Compuserve Graphdev gang)

Parameter File Listing:

```
rhs148              { ; (c) June 1992 Dick Sherry 76264,752
                      ; no commercial use w/o permission
  reset type=fn(z)+fn(pix) function=recip/sqr float=yes
  corners=-1.133658/0.246355/0.852341/-0.98766/0.246355/-0.98766
  params=0/0/1 maxiter=32000 inside=0 potential=255/500/20
  colors=0004F164G<20>65z<11>06F0E3<17>HW8IY9HW9<7>4F1FA4<19>ZLC_MCZLB<8>HB0<2\
  2>ppp<8>C900A0<20>A'1Bb2A'2<6>2G10D00G0<6>0f0<9>0E0<30>Ln6Mp7Mo7<23>0E0kdh<9\
  >LDD<4>LIC_JB_KBLLBu9e
  }
```

Generation Time: 2 minutes 20 seconds

About the Image: Little creatures hiding in fractals are not always friendly and familiar. The fellow shown in Figure 4-13 and color plate 10 looks like an alien owl from a planet where two eyes are just not enough. The fractal formula is the same fn(z) + fn(pix) type, but using the functions recip and square. The iterated function in this case is $1/z + pixel^2$. Once again, the structure of the image comes from the continuous potential coloring.

Variations: Color cycling usually is very intriguing with fractals created using continuous potential. Try the ⊕ key and see.

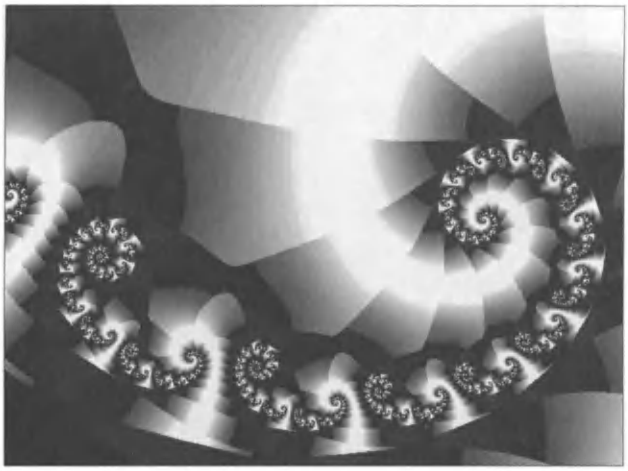


Figure 4-14 Snail Shell

Snail Shell

PAR File: RECIPES2.PAR

Image Credits: Richard H. Sherry

Parameter File Listing:

```
rhs205             { ; (c) Dick Sherry Aug 8, 1992 76264,752
  reset type=lambda
  corners=0.3536842/0.0873952/0.3558072/0.8289457/0.0307212/0.786423
  params=0.95/0.6 maxiter=500 inside=0 decomp=256
  colors=00000_<26>00FFF0<61>zz0zz0yy0<61>FF000F<62>00z00z00y<33>00_
  }
```

Generation Time: 1 minute 32 seconds

About the Image: Figure 4-14 and color plate 11 show a beautiful spiral very suggestive of a snail shell. The coloring scheme uses the `decomp=256` option which colors pixels according to the final position of a just-escaped orbit. The spiral itself is present in the fractal without the decomposition option, but the color rendering method and judicious selection of a color palette make all the difference for this image.

Variations: A good way to appreciate what the artist did with this fractal is to generate the same fractal with `decomp=0` and use the default IBM palette. To do this, load RHS205 from RECIPES2.PAR using the @ command. Press X to get BASIC OPTIONS (you don't have to wait for the fractal to finish). Set the decomposition parameter to 0 and press ENTER. Then press C to enter the color-cycling mode,

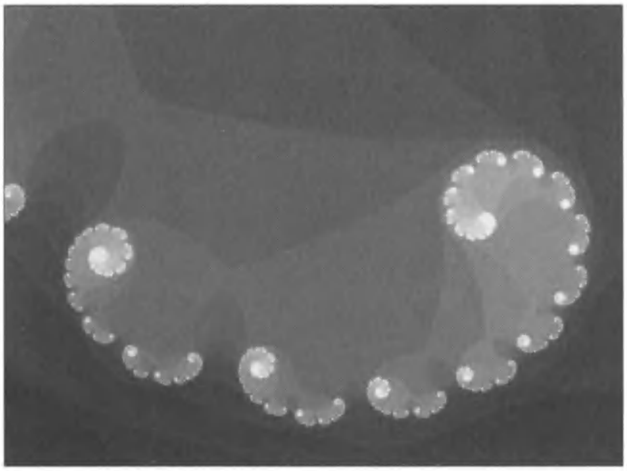


Figure 4-15 Snail Shell without decomposition

(D) to load the default IBM palette, and (ESC) to exit the color-cycling mode. The result should look like Figure 4-15. Notice that the spiral is still visible, but it is less structured because regular escape-time coloring is being used instead of decomposition coloring.

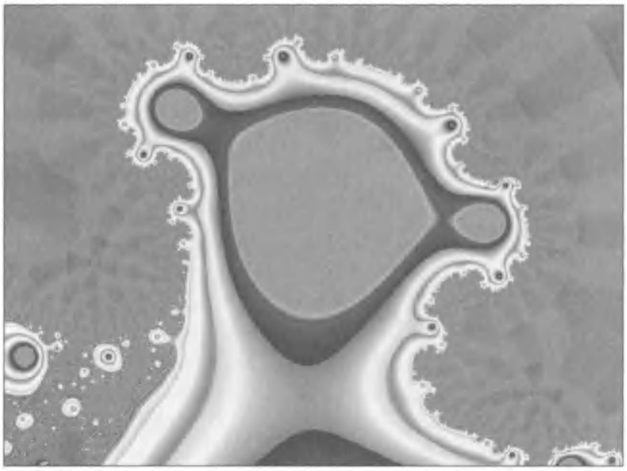


Figure 4-16 Hi Earthling!

Hi Earthling!

PAR File: RECIPES2.PAR

Image Credits: Peter Moreland

Parameter File Listing:

```
HI Earthling..     { ; Well where is your leader?
                     ; Peter Moreland 100012,3213
  reset type=magnet1m
  corners=7.414884/3.717069/-6.804988/3.803521/1.034958/1.774657
  params=1.76/4 float=y maxiter=32000 bailout=500 inside=0
  potential=255/511/0
  colors=LSUK'W<4>9kZ<2>s72<2>b9k<27>I2V<9>j5k<27>gjX<30>gUa<7>Hb1<13>Vq7Wr7Xs\
  9<30>esw<5>wUV<24>ubSubStaT<21>h9w<6>eHAeI3cL7<4>W_RVbUUcS<4>ThNk5Ln5K<12>MZ\
  V
  }
```

Generation Time: 38 seconds

About the Image: Figure 4-16 and color plate 12 show another alien creature brought to life with the continuous potential option. This fractal uses the magnetism formula—a complicated-looking rational function (division of two polynomials).

Variations: Try turning off potential by setting POTENTIAL MAX COLOR to 0 on the Y screen (EXTENDED OPTIONS). Then turn on decomposition by setting DECOMP OPTION to 255 using the X screen (BASIC OPTIONS). The same basic outline of the alien creature remains, but the appearance has a very different texture.

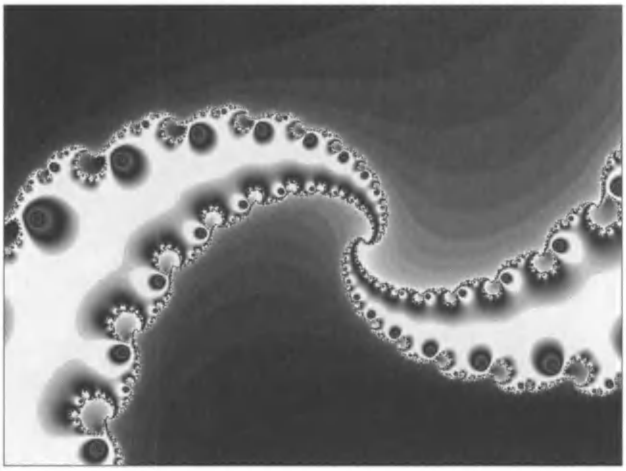


Figure 4-17 Tentacles

Tentacles

PAR File: RECIPES2.PAR

Image Credits: Peter Moreland

Parameter File Listing:

```
Tentacles          { ; 00:18:05.11
                     ; 10 jan 93 .. caren park
  reset type=magnet1j corners=1.265442/1.604642/0.775681/1.030081
  params=-0.2/0.4 float=y maxiter=500 inside=maxiter periodicity=0
  colors=000bL0<9>000<15>ut0<15>000GA4<12>2v10z00w0<14>000<15>00z<14>000<16>z0\
  X<15>000<15>p0w<15>000<15>zzz<15>000L00<13>z00<2>p00l00h00d00a00<9>000<15>zX\
  0<4>fN0
  }
```

Generation Time: 5 minutes 20 seconds

About the Image: Figure 4-17 and color plate 13 show the writhing tentacle of Captain Nemo's deep sea squid, complete with suction cups. This fractal uses the Julia variant of the same magnet formula used in "Hi Earthling!". The beauty of this fractal comes directly from the fractal algorithm and the colors—no unusual options were applied.

The `periodicity=0` parameter turns off one of Fractint's optimizations that occasionally fails. Fractint attempts to save work by noticing when an orbit is repeating itself. If you see an artificial-looking grid of dots on your fractal when using `passes=guessing`, try setting `periodicity=0` and see if that clears it up.

You have to set this on the command line or by using the (G) (for GIVE A COMMAND) facility. Press (G) and type in `periodicity=0`.

Variations: Try different color maps. You can load them by entering the color-cycling mode with (C) and then pressing (L) (load a color map).

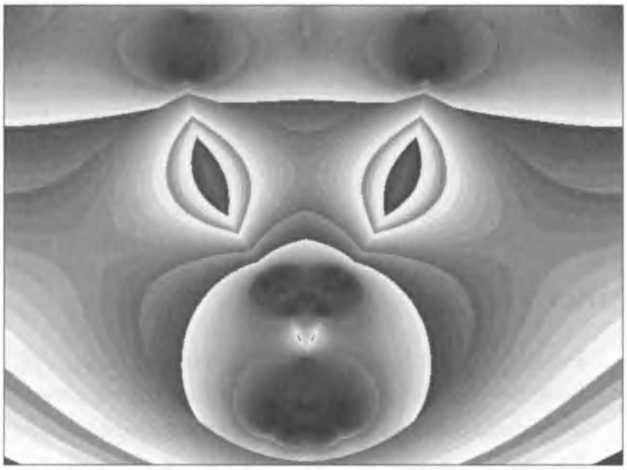


Figure 4-18 Smokie Watches You

Smokie Watches You

PAR File: RECIPES2.PAR

Image Credits: Peter Moreland, Richard Sherry

Parameter File Listing:

```
Smokie_Watches_you { ; From "Evil" by Dick Sherry ;) Credit where it's due <G>.
                     ; (c) 1992 Peter Moreland 100012,3213
  reset type=manzzpwr
  corners=-1.1681533/-1.1222362/0.0256899/-0.0246208/-1.1222362/-0.0246208
  params=1/0/2 float=y maxiter=500 inside=0 potential=255/511/0
  colors=000<178>000S00FZH<3>6H7ppp<3>UUUhhUhhT<10>NF4giT<11>TPBSOARM8PL70J5MH\
  3<2>LG3KG3JF3IF3IF2<24>050iN9hMA
  }
```

Generation Time: 1 minutes 20 seconds

About the Image: Not Smokie the Bear, no it can't be! Yes indeed, just to prove that fractal geometry provides a lifelike model of nature, we end this series of critter fractals with a genuine fractal bear! Check out Figure 4-18 and color plate 14. This image was the result of quite a humorous bit of group creativity in CompuServe's Graphdev forum. Some of the other images in this series, such as the smiling face and meditating figure, came from the same wild and woolly exchange of PAR files over the period of a few days.

The manzzpower function is an extension of the usual Mandelbrot formula. The iterated formula is $z' = z^z + z^{exp} + c$, with the parameter *exp* set to 2. All Fractint Mandelbrot-type fractal types allow the user to create "warped" images by perturbing the initial value of the orbit sequence. The first parameter in this example is set to 1, which means that the real component of the orbit initial value has 1 added at the beginning of each orbit calculation. Continuous potential is used to smooth out the colors.

Variations: Try different color maps, and see if you can come up with evil-looking eyes!

Just Beautiful Patterns

Finding fractals that look like critters is a lot of fun, and provides a welcome break from the serious pursuit of the ultimate fractal image. However, the true fractal aficionado doesn't care if his or her fractal looks like anything familiar at all. The fascination is in the pattern. Look at the following fractals, and let your imagination play with the dance of colors and interplay of shapes and texture. See what you can see, but enjoy!

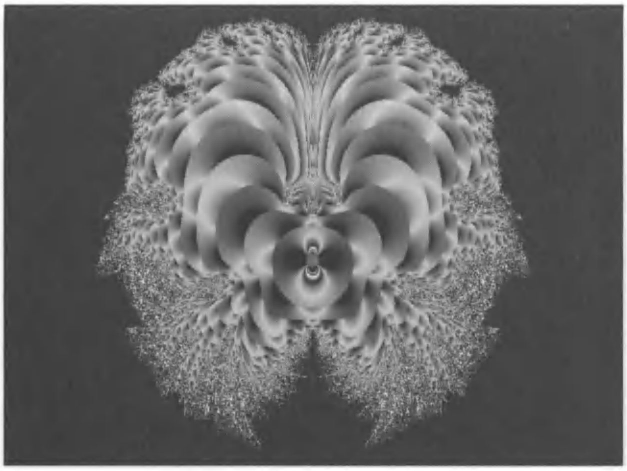


Figure 4-19 AeolisAndJanus

AeolisAndJanus

PAR File: RECIPES2.PAR

Image Credits: Caren Park

Parameter File Listing:

```
AeolisAndJanus      { ; 00:06:40.46
                      ; ... from JMS01:PigSnoutFace
  reset type=barnsleym3 corners=3.320626/-2.679374/-4.4/4.4/-2.679374/4.4
  maxiter=32767 bailout=16 outside=real logmap=yes invert=1/0/0
  colors=000840<13>zX0<15>000<15>ut0<15>000GA4<12>2v10z00w0<14>000<15>00z<14>0\
  00<16>z0X<15>000<15>p0w<15>000<15>zzz<15>000L00<13>z00<2>p00l00h00d00a00<9>0\
  00420
  }
```

Generation Time: 1 minutes 40 seconds

About the Image: The barnsleym3 fractal type is characterized by a conditional branch in the formula that creates discontinuities in the image. Fractals made with this type often have a mosaic quality. This image uses the `outside=real` option, which colors each pixel according to the iteration count plus the real value of the first escaping orbit. This has the effect of revealing additional structure, which takes on the appearance of nested petals that create a pine cone effect. Using logmap compresses colors and lets you see a wider range of color gradations before details are lost in the "gravel." The result is shown in Figure 4-19 and color plate 15.

Variations: Color cycling of this fractal is gentle and beautiful because of the many continuous gradations of color.

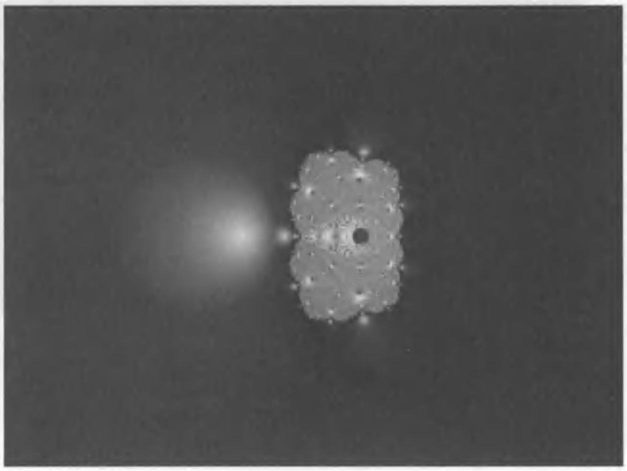


Figure 4-20 AsteroidShip

AsteroidShip

PAR File: RECIPES2.PAR

Image Credits: Caren Park

Parameter File Listing:

```
AsteroidShip { ; 00:38:19.91
               ; 02 jan 93 .. caren park
  reset type=fn*z+z function=cotan passes=t
  corners=-1.914244/1.413756/-1.248/1.248 params=1/0/1 maxiter=32767
  inside=bof60 outside=real logmap=yes invert=0.5/0/0
  colors=0tz<39>05z03z02z00z00y<59>003002000000000<29>00k00m01m<29>0ky0mz1mz<3\
  0>zzz<44>5zz3zz2zz0zz0yz<2>0uz cyclerange=0/255
  }
```

Generation Time: 16 minutes

About the Image: Figure 4-20 and color plate 16 show a fractal with delicate color shading that looks much like the interplay of light in a mist. The novel idea in this PAR file is to combine `inside=BOF` with inversion. The `invert=0.5/0/0` command turns the fractal inside out, and surrounds the fractal object by the luminous glow of the BOF60 effect.

Variations: Yes! Color cycle this fractal! Try turning off the various options (inside, outside, and logmap) to get a feel for what they do.

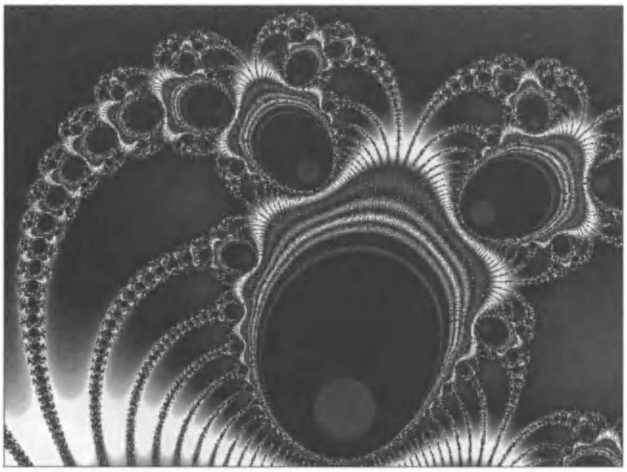


Figure 4-21 BadDream

BadDream

PAR File: RECIPES2.PAR

Image Credits: Caren Park

Parameter File Listing:

```
BadDream          { ; 01:30:10.15
                    ; 05 dec 92 .. caren park
  reset type=newton
  corners=0.813797527/0.814047167/-0.001023803/-0.000836573 params=47
  float=y maxiter=500 inside=maxiter periodicity=-1
  colors=000f0l<2>p0w<15>000<15>zzz<15>000L00<13>z00<2>p00l00h00d00a00<9>000<1\
  5>zX0<15>000<15>ut0<15>000GA4<12>2v10z00w0<14>000<15>00z<14>000<16>z0X<15>00\
  0<11>c0h
  }
```

Generation Time: 44 seconds

About the Image: Newton fractals graphically display the life-and-death struggle between different attractors battling to capture orbits. In this case, there are 47 such attractors because of `params=47`. The braids in this image (see Figure 4-21 and color plate 17) show the twisting together of microscopically separated regions that launch orbits ending up being captured by different attractors.

This isn't the critter section so we won't comment on the scary monster that Caren must have seen in this image!

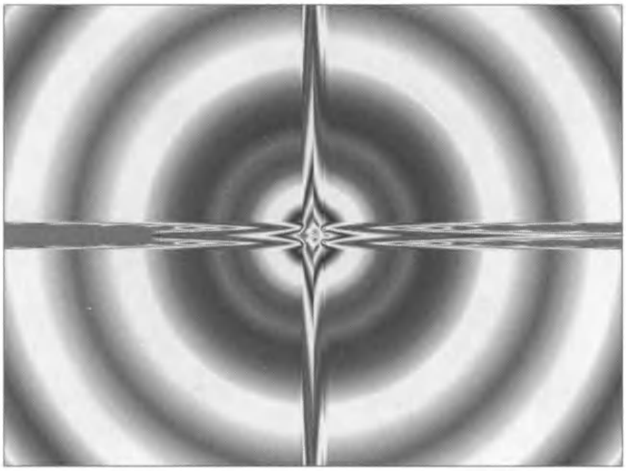


Figure 4-22 BullsEyeStar

BullsEyeStar

PAR File: RECIPES2.PAR

Image Credits: Caren Park

Parameter File Listing:

```
BullsEyeStar       { ; 00:01:46.94
                     ; 31 dec 92 .. caren park
  reset type=fn*z+z function=recip
  corners=31.99/-31.99/23.992505/-23.992505 params=-300/-200/5100/-21072
  maxiter=32000 inside=0 potential=255/200/32000 invert=0.5/0/0
  colors=WPJbHE<5>zH0<8>SDF<3>N3JLOKL1KM2KN3K04J<12>zz0<7>"9YYAXVB<6>KAJMAKOA\
  M<12>ziF<13>K8C413000<15>svcsvcsvcsvc<4>jiWhfUfdTebS<3>ZUMYRKUMHSJF<6>EAb<6>\
  7Nu5Px5Nu<5>A5cB5'<3>G8SH9QIAQ<4>SKP<3>_VZ'Y'b'cdcf<4>nru<15>0I5<6>eYJg_LiaN\
  kcPneR<4>zpa<4>jdWgaVcZT'WSYUR<6>AAK<3>ALQAOSARUBUWBXY<6>DqlDtnCql<5>8Xc7Ta9\
  Q'<5>L4R<8>VHIZHG
  }
```

Generation Time: 40 seconds

About the Image: The authors would like to politely request that all mathematicians and fractal purists quietly move on to the next recipe and skip this one. The BullsEyeStar fractal is not for you!

The BullsEyeStar fractal shown in Figure 4-22 and color plate 18 appears to be impossible. Consider this: the iterated formula is $z' = \text{fn}(z)z + z$, where $\text{fn}(z)$ is the recip function, $1/z$. But $(1/z)z$ is 1, so this formula is really $z' = 1+z$. Not much of a formula, and it shouldn't generate much of an image. For any starting point, the orbit just heads straight east at the rate of one unit per iteration. This should result in a very dull image. However, there are a few wrinkles. The first is that continuous potential is applied, and the second is that the fractal is inverted.

Fractint uses several kinds of math internally. Fixed-point integer math runs very fast and doesn't need a math coprocessor. The disadvantage is that fixed-point numbers have a limited dynamic range, and are not suitable for deep zooms or certain functions that have a wide range of values. The BullsEyeStar fractal works only with integer math. Try turning float on with the (F) toggle, and the result is zilch—an almost blank screen. (You can tell if floating point is used with the (TAB) status command. Integer math is never mentioned on this screen, but if floating point is used, this screen announces it.)

Perhaps the rolling circular waves of light come from the continuous potential calculation and the choice of color map. But where does the star come from? Remember that inversion is turned on, so the inside and outside are reversed. The complex numbers further from the origin are the ones where fixed-point numbers run out of precision. These numbers have been reflected to the inside, so the star is most likely an artifact of the failure of integer math.

Such an ephemeral fractal, and so beautiful! Is a fractal a fractal if it is an accidental consequence of a program's internals? Surely our failure to understand the basis of a fractal does not make it less a fractal.

Speaking for themselves, the authors are quite fond of this anomalous image, and we thank Ms. Park for discovering it!

Variations: Turn on floating point, turn off continuous potential, or turn off inversion, and watch this fractal slip away like sand through your fingers! To get insight into this image, try performing a 3-D transformation on it. You can use the (3) command or use the following PAR file:

```
bullseye          {
  3d=yes filename=bullseye.gif scalexyz=90/90 roughness=30 waterline=0
  ambient=20 rotation=60/30/0 perspective=0 xyshift=0/0
  colors=WPJbHE<5>zH0<8>SDF<3>N3JLOKL1KM2KN3K04J<12>zz0<7>"9YYAXVB<6>KAJM\
  AKOAM<12>ziF<13>K8C413000<15>svcsvcsvcsvc<4>jiWhfUfdTebS<3>ZUMYRKUMHSJF<\
  6>EAb<6>7Nu5Px5Nu<5>A5c<5>H9Q<4>QIPSKPUNS<2>_VZ'Y'b'cdcf<4>nru<15>0I5<6>\
  eYJg_LiaNkcPneR<4>zpa<4>jdWgaVcZT'WSYUR<6>AAK<3>ALQAOSARUBUWBXY<6>DqlDtn\
  Cql<6>7Ta<6>L4R<8>VHIZHG
  }
```

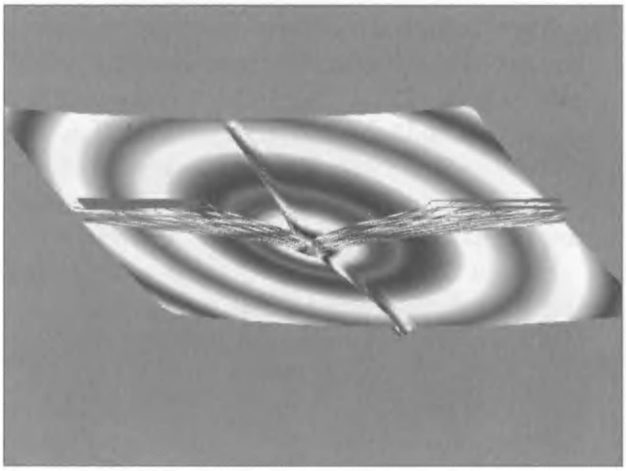


Figure 4-23 BullsEyeStar in 3-D

A 3-D transformation treats the color numbers as a third dimension and projects the resulting surface to two dimensions. You can see the result in Figure 4-23. This "fractal" is in fact a smooth basin with a cross-shaped discontinuity in the center.

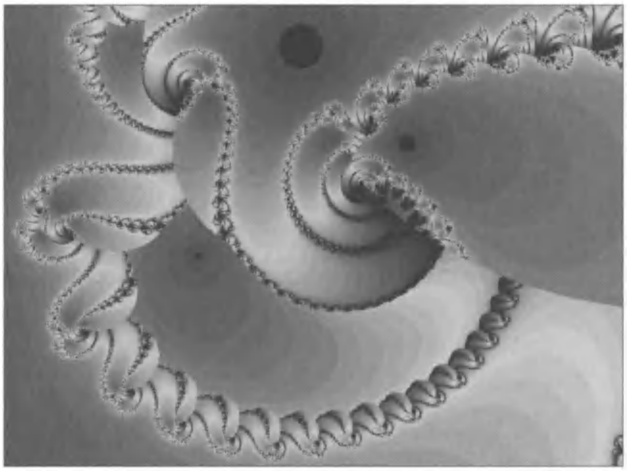


Figure 4-24 BlueStrandsOnGreen

BlueStrandsOnGreen

PAR File: RECIPES2.PAR

Image Credits: Caren Park

Parameter File Listing:

```
BlueStrandsOnGreen { ; 01:05:38.22
                     ; 06 dec 92 .. caren park
  reset type=complexnewton passes=t
  corners=-0.000008636234/-0.000008034234/0.0000004819/0.0000009334
  params=3/2/4/1 float=y maxiter=400 bailout=2500 inside=maxiter
  periodicity=0
  colors=000ZZZ<6>zzz<15>000L00<13>z00<2>p00l00h00d00a00<9>000<15>zX0<15>000<1\
  5>ut0<15>000GA4<12>2v10z00w0<14>000<15>00z<14>000<16>z0X<15>000<15>p0w<15>00\
  0<7>WWW
  }
```

Generation Time: 28 minutes

About the Image: The ComplexNewton fractal type is similar to the regular Newton fractal. Newton's method is used to determine the roots of the simple polynomial $z^n - r = 0$. For the regular Newton fractal type, *n* must be an integer and *r* is 1. For the ComplexNewton both *n* and *r* can be complex numbers. The very same Newton formula works in both cases, but with a big difference. The calculation of raising a complex number to a complex power involves computing a logarithm, and the complex logarithm has infinitely many values. One is arbitrarily selected in order to complete the calculation. The fractal resulting from this arbitrariness exhibits a discontinuous tear. This tear is propagated throughout the image at all magnifications. These tears look as though you are viewing a three-dimensional object, and parts of the object are blocking other parts behind.

The BlueStrandsonGreen fractal shown in Figure 4-24 and color plate 19 display this characteristic very clearly. Compare it to the regular Newton image BadDream earlier in this chapter. You can see the same braided strands in both images, but in the BlueStrandsonGreen image, these braids abruptly begin and end.

Part of the appeal of this fractal is the delicate color scheme that highlights one intertwined strand of the braid.

Variations: This image is zoomed in very deeply. You can zoom out by making a small zoom box with the PAGE UP key or mouse and pressing CONTROL-ENTER

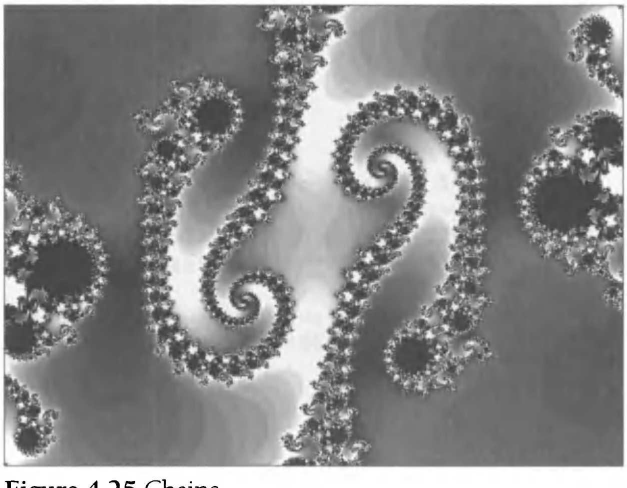


Figure 4-25 Chains

Chains

PAR File: RECIPES2.PAR

Image Credits: Caren Park

Parameter File Listing:

```
Chains             { ; 00:01:15.68
                     ; 03 jan 93 .. caren park
  reset type=julia corners=-0.1/0.1/-0.075/0.075
  params=-0.784469886/0.133089005
  colors=WPJ775<13>svcsvcsvcsvc<4>jiWhfUfdTebS<3>ZUMYRKUMHSJF<6>EAb<6>7Nu5Px5N\
  u<5>A5cB5'<3>G8SH9QIAQ<4>SKP<3>_VZ'Y'b'cdcf<4>nru<15>0I5<6>eYJg_LiaNkcPneR<4\
  >zpa<4>jdWgaVcZT'WSYUR<6>AAK<3>ALQAOSARUBUWBXY<6>DqlDtnCql<5>8Xc7Ta9Q'<5>L4R\
  <8>VHI<7>zHO<8>SDF<3>N3JLOKL1KM2KN3K04J<12>zz0<7>"9YYAXVB<6>KAJMAKOAM<12>zi\
  F<13>K8C413000443
  }
```

Generation Time: 28 minutes

About the Image: The Chains image, shown in Figure 4-25 and color plate 20, is just a simple unadorned Julia set with no special options and very skillful coloring.

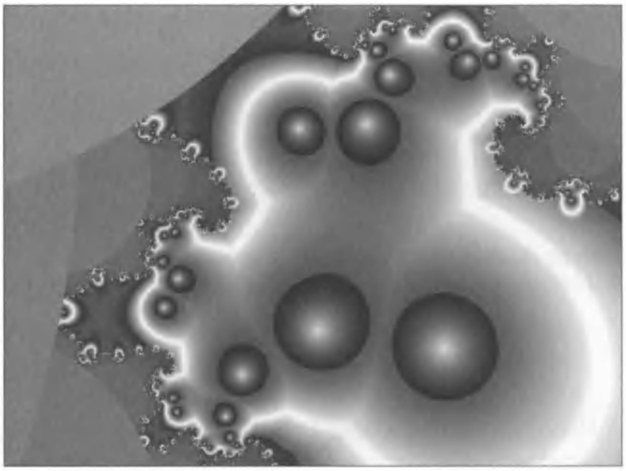


Figure 4-26 EggsOnLeaf

EggsOnLeaf

PAR File: RECIPES2.PAR

Image Credits: Caren Park

Parameter File Listing:

```
EggsOnLeaf          { ; 00:16:32.89
                      ; 28 dec 92 .. caren park
  reset type=fn(z)+fn(pix) function=sin/cosh passes=t
  corners=-0.571705/-0.403833/0.808401/0.934305 params=0.7/0.333/0.5/0.5
  float=y maxiter=100 bailout=50 inside=bof60
  colors=ddd<16>000PFF<29>xeDzfCyfC<30>PFF0A0<70>_UI'UIaVJbWK<19>xo'zpaypa<29>\
  UPF000<29>nnnpppooo<13>TTT
  }
```

Generation Time: 8 minutes

About the Image: The name EggsOnLeaf tells it all, as you can see in Figure 4-26 and color plate 21. The "eggs" are created by the `inside=bof60` option and the selection of a palette that highlights the BOF60 basins.

Variations: The artist elected to use the `passes=tesseral` algorithm. This option generally does not affect the resulting image, although if any image details can be completely surrounded by a box of constant color, they may disappear. You can incorporate an artifact of the tesseral recursive boxes method by setting FILL COLOR to 1 from the (X) BASIC OPTIONS screen. The outer areas are affected the most, and are turned into a latticework of boxes.

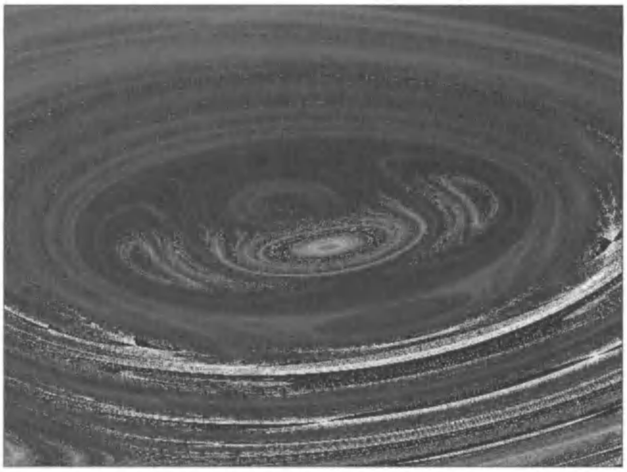


Figure 4-27 Who Pulled the Plug?

Who Pulled the Plug?

PAR File: RECIPES2.PAR

Image Credits: Peter Moreland

Parameter File Listing:

```
Who_pulled_the.... { ; Plug!  Oh well, there goes the galaxy...
                     ; (C)1993 Peter Moreland 100012,3213
                     ;
  reset type=frothybasin
  corners=0.4968056/0.6382582/0.3680705/0.5064456/0.4813085/0.3887334
  params=6 float=y maxiter=5000 decomp=256 distest=5/71 finattract=y
  colors=@froth6.map
  }
```

Generation Time: 1 minute and 40 seconds

About the Image: Figure 4-27 and color plate 22 show a giant whirlpool that looks as though it is emptying the universe through a cosmic bathtub drain. This image was created using the Frothy Basins fractal type, discovered by James C. Alexander of the University of Maryland.

One of the things that makes the Frothy Basins fractal so interesting is the shape of the dynamical system's attractors (shapes that orbits tend to move toward). It is not at all uncommon for a dynamical system to have nonpoint attractors. Shapes such as circles are very common. Strange attractors are attractors which are themselves fractal. What is unusual about this system, however, is that the attractors intersect, giving the fractal the appearance of a frothy liquid that has been stirred up. This is the first case in which such a phenomenon has been observed. The three attractors for this system are made up of line segments that overlap to form an equilateral triangle. This attractor triangle can be seen by pressing the O key while the fractal is being generated, turning on the SHOW ORBITS option.

Variations: An interesting variation on this fractal can be generated by applying the previous mapping twice per iteration. The result is that each of the three attractors is split into two parts, giving the system six attractors.

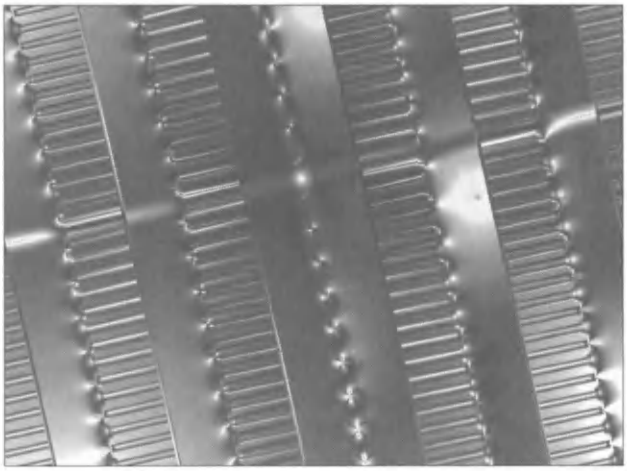


Figure 4-28 Meta_Cold

Meta_Cold

PAR File: RECIPES2.PAR

Image Credits: Peter Moreland

Parameter File Listing:

```
Meta_Cold!  { ; (c)1992 Peter Moreland 100012,3213
  reset type=fn*z+z function=tan
  corners=21.315763/-31.989983/-22.868427/22.868451/-19.828226/31.989997
  params=5/5/5/100 maxiter=32000 inside=0 potential=255/200/32000
  colors=LSUJPU<11>GMUFMUFMUFLUFLUEKUEKUDJV<2>BLSAMR9MQ8MP6KM<58>inujovinu<164\
  >JQUQW'JQU
  }
```

Generation Time: 72 seconds

About the Image: Figure 4-28 shows a fractal that looks like an aluminum lattice. The smooth coloring comes from the use of continuous potential, and the repeating pattern from the use of the trigonometric tangent function which is periodic (repeats itself at regular intervals).

Variations: You can turn the cold metal latticework red-hot by manipulating the colors. Try color cycling and using the more continuous palettes available with the higher-numbered function keys F5 through F9.

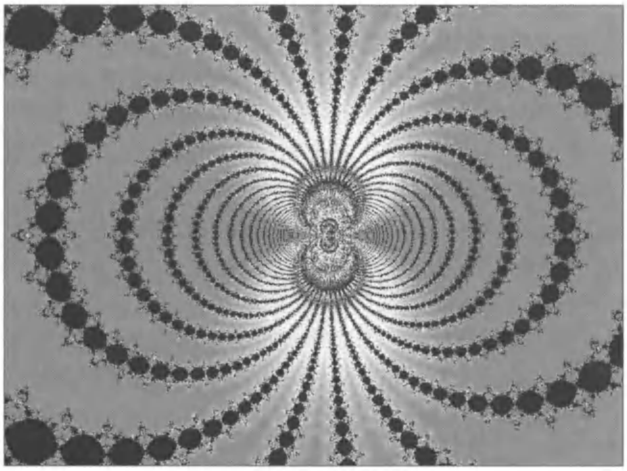


Figure 4-29 MagneticHole

MagneticHole

PAR File: RECIPES2.PAR

Image Credits: Peter Moreland

Parameter File Listing:

```
MagneticHole       { ; 00:23:23.30
                     ; 19 dec 92 .. caren park
  reset type=formula formulafile=fractint.frm formulaname=halleysin
  passes=2 corners=0.883445/1.020085/-0.05124/0.05124 params=1.6 float=y
  maxiter=500 bailout=1000 inside=maxiter
  colors=000sIE<6>lKKkLLjLMiMNhMO<5>aPT<20>zzz0z0LLLz00<5>zz0000555<2>EEE0z0KK\
  K<3>___ccchhhmmmssszzzzz0<6>z0Gz00<3>zz0nx6buCP0ZFpP<3>0zz<2>0GzVVz<3>zVz<3>\
  zVV<3>zzV<3>VzV<3>Vzz<2>Vbzhhz<3>zhz<3>zhh<3>zzh<3>hzh<3>hzz<2>hlzz00w03000o\
  0B<12>00z<15>zz0<8>EfH<2>zzz<5>jkqzzzdfm0zlZaj<5>IN'FKZAE057C000<6>zkk<7>z00\
  G00G40G8000z<10>zc0<4>zY2000zV3<5>zN5000zL6zJ7zI7zG8<5>tID
  }
```

Formula File Listing:

```
halleySin (XYAXIS) {; Chris Green. Halley's formula applied to sin(x)=0.
  ; Use floating point.
  ; P1 real = 0.1 will create the picture from page 281 of Pickover's book.
  z=pixel:
   s=sin(z), c=cos(z)
   z=z-p1*(s/(c-(s*s)/(c+c))),
    0.0001 <= |s|
  }
```

Generation Time: 4 minutes 4 seconds

About the Image: Figure 4-29 and color plate 24 show a fractal that looks like magnetic lines of force made from the Halley formula for the sine function. Halley's formula is an alternative to Newton's formula for finding the roots of functions (values where the function gives the value 0). This version of Halley's formula is not built into Fractint, but uses a formula stored in FRACTINT.FRM.

Variations: Yet another fractal that begs to be color cycled! Once you start the cycling, notice the difference between F2 (rapidly changing color bands) and F10 (slowly oozing colors).

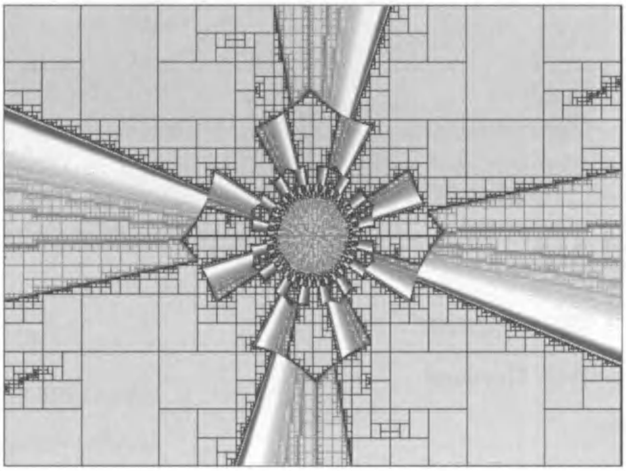


Figure 4-30 MapCompass

MapCompass

PAR File: RECIPES2.PAR

Image Credits: Caren Park

Parameter File Listing:

```
MapCompass         { ; 00:00:21.20
                     ; ... from POD047
  reset type=lambdafn function=sqr passes=t
  corners=-6.81694/6.810364/-4.549606/4.544006 params=1/0.9 maxiter=128
  fillcolor=255 inside=epsiloncross outside=mult logmap=yes
  colors=000LMF<9>svcsvcsvcsvc<13>UMHSJFQIJ<5>EAb<6>7Nu5Px5Nu<5>A5c<2>E7WF8UG8\
  SH9QJANKBLMCI<6>_VZ'Y'b'cdcf<4>nru<15>0I5<6>eYJg_LiaNkcPneR<4>zpa<4>jdWgaVcZ\
  T'WSYUR<6>AAK<4>AOS
  }
```

Generation Time: 11 seconds

About the Image: Figure 4-30 and color plate 25 show a radially symmetric image that could be a special compass or a windmill. The gridwork in the background is an artifact of the tesseral strategy for computing the image by recursively subdividing into smaller and smaller rectangles. The `fillcolor=255` option makes this grid visible. Both the `inside=epsiloncross` and `outside=mult` options add structure to the fractal.

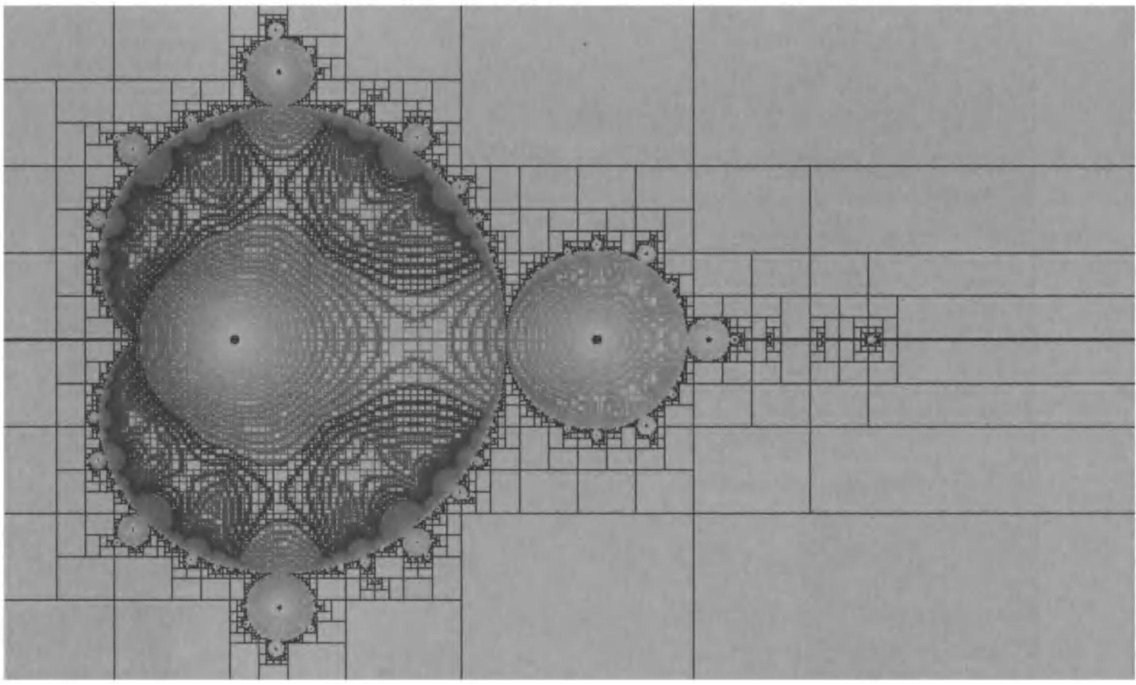
Variations: To appreciate how useful Fractint options are for revealing fractal structure, try converting MapCompass to a regular escape-time fractal. Invoke the BASIC OPTIONS screen by pressing X and change `passes=t` to `passes=g`, which removes the background grid. Then change `inside=epsiloncross` to `inside=0` and `outside=mult` to `outside=iter` to remove much of the remaining fractal structure. The resulting image looks quite ordinary. There is simply too much to see in a fractal in a glance; the options provide alternative views of the same unfathomable dynamic system, each one revealing one aspect.

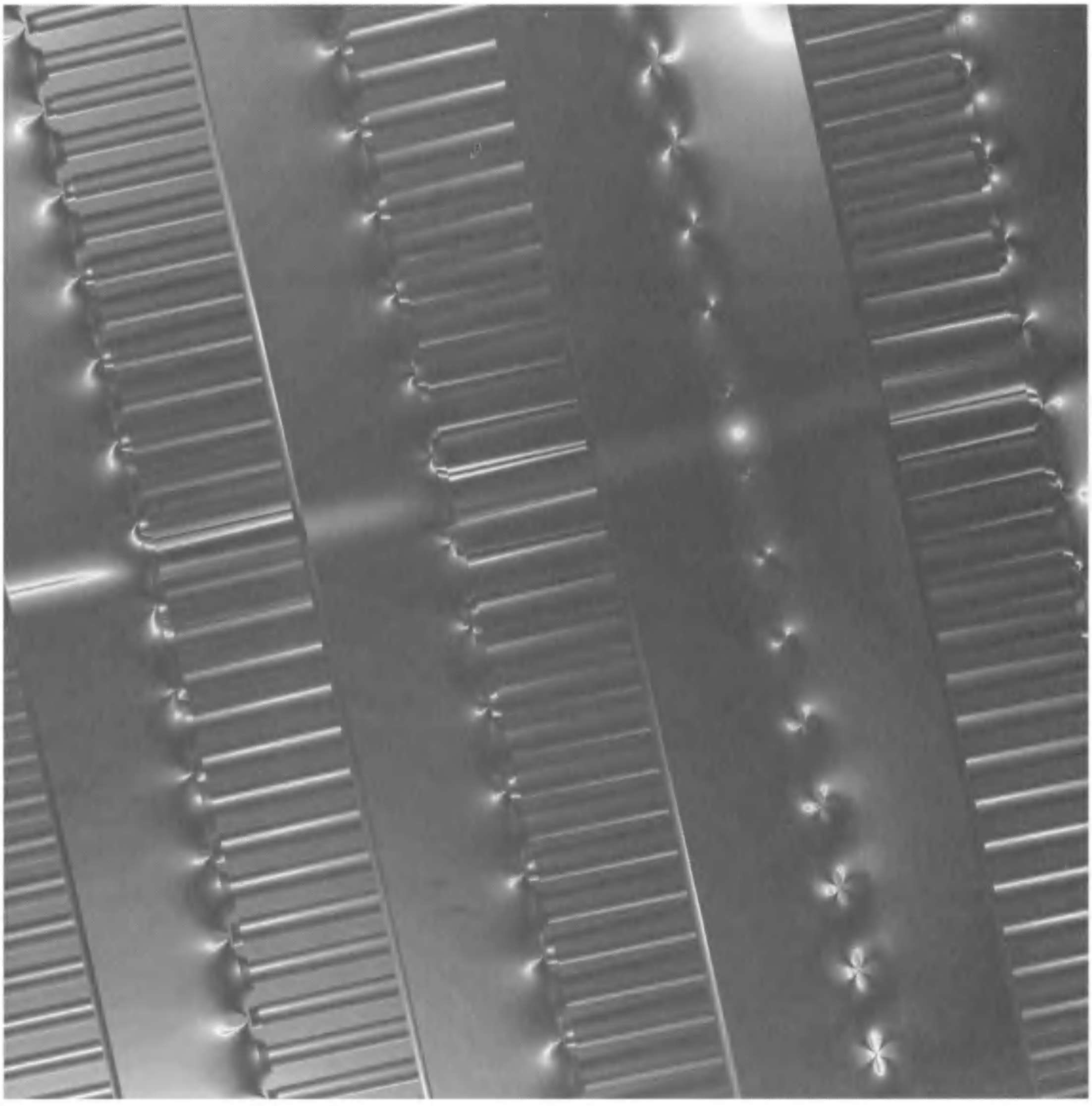
THERE'S MORE

These recipes are only the beginning of the tasty fractal treats you can try. For many more gourmet concoctions, look for other .PAR files on your distribution disk, and check out the 1800 images on the companion CD. Have fun!

CHAPTER 5

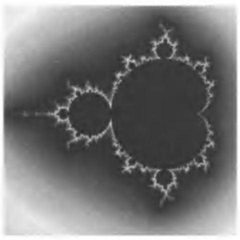
FRACTINT REFERENCE





CHAPTER 5

FRACTINT REFERENCE



Fractint gives you the capability to do a *lot* more than just generate fractal images. You can save them, restore them, contort them using various 3-D transformations, print them, and adjust their colors in all kinds of ways. You can even use Fractint to perform all of these functions on ordinary GIF files.

The other chapters in this book cover specific Fractint commands when they are appropriate for the topic at hand. In this chapter, Fractint's commands and how you access them *are* the topics at hand. If Fractint can do something, this chapter will tell you how to tell Fractint to do it.

First, we'll give you a general overview of Fractint's command structure and several alternative ways by which you can tell Fractint what you want it to do. Then we'll briefly describe Fractint's context-sensitive, on-line help system. Then, because nearly every Fractint command can optionally be issued as a *command-line argument*, we'll cover the basics of Fractint's command-line argument syntax. Finally, we'll cover Fractint's individual commands one by one.

FRACTINT COMMANDS

There are several different methods of telling Fractint what to do. Which method you use in a particular situation is often a matter of convenience and personal style. After becoming familiar with the program, you will probably end up using a mixture of these methods that you find to be the most effective for you. The four basic mechanisms are the mouse, the keystroke commands, the arrow-key-controlled menu interface, and command-line options. Let's look briefly at each of these.

The Mouse

Fractint uses the mouse for only two purposes: bringing up and controlling the zoom box while in its main display mode, and moving the pixel-selection cursor in the optional palette editing and orbits/Julia display modes. Even if you are not an enthusiastic fan of the mouse, we recommend using it with the zoom box if you have one.

If you don't have a mouse, don't worry. A mouse is not required for any Fractint operation, and you can perform all operations using only the keyboard.

Keystroke Commands

Using keystroke commands is often the fastest way to interactively control the operation of Fractint. If you have just generated the perfect fractal image and want to save it for posterity, for example, all you have to do is press the S key. Keystroke commands are not case-sensitive: you can save that fractal image using either the s or S key.

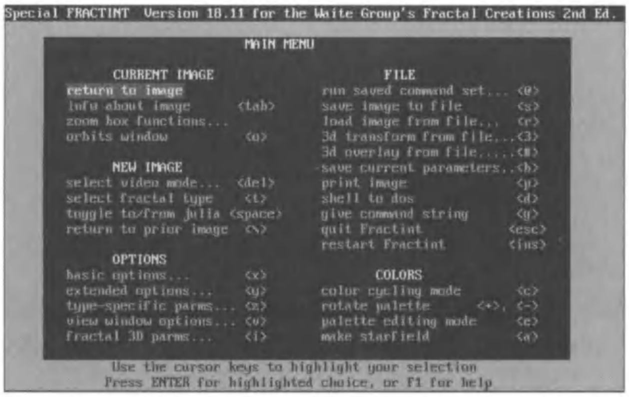
Note that some of Fractint's keystroke commands (such as the S command) perform their actions immediately, while others (such as the T command, which you use to select a new fractal type) take you to a menu screen for further action.

Because Fractint has several modes of operation, the effect of a keystroke sometimes depends on which mode Fractint is in. We'll discuss this in more depth as we cover the individual command functions.

The Arrow-Key-Controlled Menu Interface

The arrow-key or cursor-key interface uses a series of full-screen menus that display options. You can display the main arrow-key menu by pressing the ENTER key when you're looking at Fractint's initial credits screen or by pressing the ESC key when you're looking at a fractal image. Figure 5-1 shows the MAIN MENU as it appears once you've generated a fractal image (the menu that appears prior to that point is somewhat shorter, as several of the items on the full menu aren't applicable yet).

When using any of Fractint's menus, you select an option by moving the highlighted area to the desired option using the arrow keys and then pressing ENTER. At the MAIN MENU level, those menu selection items which are also reachable via a keystroke command show that keystroke inside angle brackets (<>)—this can help you learn Fractint's basic commands rapidly.



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.
                              MAIN MENU

        CURRENT IMAGE                          FILE
    return to image                  run saved command set... <@>
    info about image       <tab>     save image to file        <s>
    zoom box functions...            load image from file...   <r>
    orbits window            <o>     3d transform from file...<3>
                                     3d overlay from file.....<#>
        NEW IMAGE                    save current parameters..<b>
    select video mode...   <del>     print image               <p>
    select fractal type      <t>     shell to dos              <d>
    toggle to/from julia <space>     give command string       <g>
    return to prior image    <\>     quit Fractint           <esc>
                                     restart Fractint        <ins>
        OPTIONS
    basic options...         <x>               COLORS
    extended options...      <y>     color cycling mode        <c>
    type-specific parms...   <z>     rotate palette       <+>, <->
    view window options...   <v>     palette editing mode      <e>
    fractal 3D parms...      <i>     make starfield            <a>
            Use the cursor keys to highlight your selection
           Press ENTER for highlighted choice, or F1 for help
```

Figure 5-1 Fractint's Main menu screen

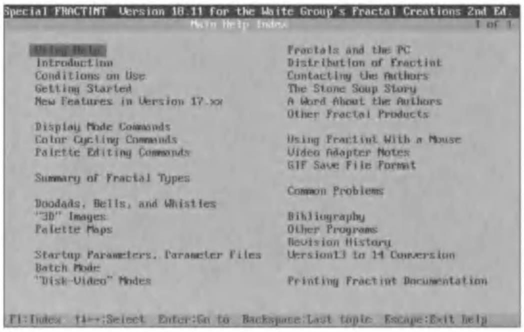
Some screens have input fields for entering various parameters that control how the program operates. Screens used to select files also have directory navigation capabilities; by selecting subdirectories, the directory displayed is changed. On these screens, selecting the directory ".." moves the listed directory up the directory tree. The special capabilities of some of these screens are documented later in this chapter.

Command-Line Arguments

A *command-line argument* is an option that you give a program "on the command line" as you start it up. When you give MS-DOS the command **TYPE AUTOEXEC.BAT**, for instance, you are giving the TYPE command the command-line argument AUTOEXEC.BAT. Fractint accepts command-line arguments that allow you to run it with your choice of video mode, starting coordinates, and just about every other parameter and option known to Fractint.

Fractint has several other ways to use these command-line arguments besides putting them in the command-line. In fact, referring to them as "command-line" arguments is a bit of an anachronism—the command line is probably the place where they are used the least.

By whatever name they are called, command-line arguments are extremely useful. Command-line arguments are used inside startup files (such as the SSTOOLS.INI file). They can also be invoked "on the fly" using Fractint's GIVE COMMAND STRING (G) command or the menu interface. Finally, Fractint can also load and save fractal images using *parameter files*, which are files containing the instructions for generating fractal images rather than their actual *bitmaps*—

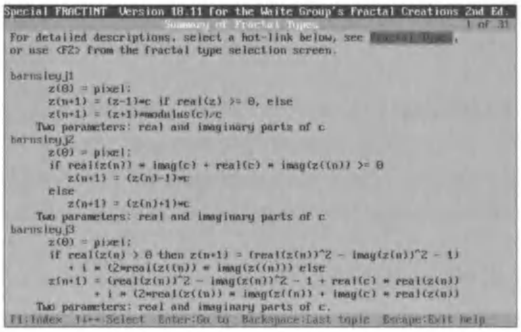


```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.
                              Main Help Index                            1 of 1

   Using Help                               Fractals and the PC
   Introduction                             Distribution of Fractint
   Conditions on Use                        Contacting the Authors
   Getting Started                          The Stone Soup Story
   New Features in Version 17.xx            A Word About the Authors
                                            Other Fractal Products
   Display Mode Commands
   Color Cycling Commands                   Using Fractint With a Mouse
   Palette Editing Commands                 Video Adapter Notes
                                            GIF Save File Format
   Summary of Fractal Types
                                            Common Problems
   Doodads, Bells, and Whistles
   "3D" Images                              Bibliography
   Palette Maps                             Other Programs
                                            Revision History
   Startup Parameters, Parameter Files      Version13 to 14 Conversion
   Batch Mode
   "Disk-Video" Modes                       Printing Fractint Documentation

 F1:Index  ↑↓←→:Select  Enter:Go to  Backspace:Last topic  Escape:Exit help
```

Figure 5-2 The Main Help Index menu



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.
                          Summary of Fractal Types                      1 of 31
For detailed descriptions, select a hot-link below, see Fractal Types,
or use <F2> from the fractal type selection screen.

barnsleyj1
     z(0) = pixel;
     z(n+1) = (z-1)*c if real(z) >= 0, else
     z(n+1) = (z+1)*modulus(c)/c
   Two parameters: real and imaginary parts of c
barnsleyj2
     z(0) = pixel;
     if real(z(n)) * imag(c) + real(c) * imag(z((n)) >= 0
        z(n+1) = (z(n)-1)*c
     else
        z(n+1) = (z(n)+1)*c
   Two parameters: real and imaginary parts of c
barnsleyj3
     z(0) = pixel;
     if real(z(n) > 0 then z(n+1) = (real(z(n))^2 - imag(z(n))^2 - 1)
        + i * (2*real(z((n)) * imag(z((n))) else
     z(n+1) = (real(z(n))^2 - imag(z(n))^2 - 1 + real(c) * real(z(n))
           + i * (2*real(z((n)) * imag(z((n)) + imag(c) * real(z(n))
   Two parameters: real and imaginary parts of c.
F1:Index  ↑↓←→:Select  Enter:Go to  Backspace:Last topic  Escape:Exit help
```

Figure 5-3 The Fractal Types Help screen

instructions stored in the form of command-line arguments. Each of these techniques, and the syntax of Fractint's command-line arguments in general, will be discussed in more detail later in the chapter.

FRACTINT'S HELP SYSTEM

Before getting too far into this reference chapter, it seems appropriate to mention one tool that may reduce your need to read it on occasion—Fractint's on-line, context-sensitive, hypertext-format Help system.

Fractint's built-in, on-line Help system can help you learn to use the program's many features. To get help at any time, simply press (F1). Fractint's Help system is context-sensitive, so the Help screen you see when you press (F1) depends on the particular mode you were in and which command function you were processing when you pressed that key. Figure 5-2 shows the main Help menu screen that is displayed if you press (F1) immediately upon starting Fractint. Figure 5-3 shows the Help screen that is displayed if you press (F1) immediately after you have pressed the (T) key (while staring at the screen showing Fractint's rather formidable list of fractal types).

The Help system uses a hypertext format—while using the Help system, you can select any topic (*hot-link*) displayed in blue by using the arrow or (TAB) keys to highlight it and then pressing (ENTER) to select it. For instance, on the fractal types help screen shown in Figure 5-3, the terms Fractal Types, Barnsleyj1, Barnsleyj2, and BarnsleyJ3 are hot-links and are displayed in blue. You can get more information on the Barnsleyj2 fractal type by pressing the down arrow key twice to highlight it and then pressing (ENTER). In the main Help screen shown in Figure 5-2, *every* item on the screen is displayed in blue, as that screen consists entirely of a list of help topics.

You can wander down through as many levels of help topics as you want. The BACKSPACE key backs you out one level at a time. Pressing F1 when you are already in the Help system always sends you to the main Help menu shown in Figure 5-2.

There is frequently more than one screen's worth of information on a particular help topic. When that happens, use the PAGE UP and PAGE DOWN keys to move through the available Help screens for that topic. The SUMMARY OF FRACTAL TYPES Help screen shown in Figure 5-3 is probably the most extreme example of this situation—note the 1 OF 33 displayed in the upper right-hand corner of that screen.

You can exit from Fractint's on-line Help system at any time by pressing the ESC key. You will be returned back to Fractint and whatever mode and command you left it in.

Finally, we should point out that the Help system gives you the most up-to-date information available about Fractint. The mechanics of book publishing are such that the companion disk is generated sometime *after* the book text is completed. If one of Fractint's entry screens or options looks a bit different than this reference chapter indicates, pressing the F1 key to check out the on-line help is a good idea. It's possible that the Fractint authors managed to relax a program limitation or even add a new feature during the interval between the book text being finalized and the master disk being sent to the duplicators.

USING COMMAND-LINE ARGUMENTS

As we proceed through this chapter describing various Fractint commands, we will list the command-line arguments that apply to those commands. Because of this, we should describe the syntax of command-line arguments before we go any further.

When used "on the command-line," the syntax for command-line arguments is as follows:

```
fractint argument=value argument=value argument=value...
```

where the individual arguments are Fractint settings and are separated by one or more spaces (an individual argument may **not** include spaces). Either upper- or lowercase may be used, and arguments can be in any order. A typical sequence of arguments might be

```
type=julia video=F3 inside=10
```

Command	Meaning
COMMAND=<nnn>	Enter a number in place of "nnn."
COMMAND=<filename>	You supply the file name.
COMMAND=yes|no|whatever	Type in one of the options (in this case, the options are "yes," or "no," or "whatever"). The "|" here means "or."
COMMAND=1st[/2nd[/3rd]]	You supply the slash-separated parameters to replace 1st, 2nd, and 3rd.The brackets [] mean that 2nd and 3rd, are optional. You *do* type in the slashes.

Table 5-1 Command-Line argument syntax

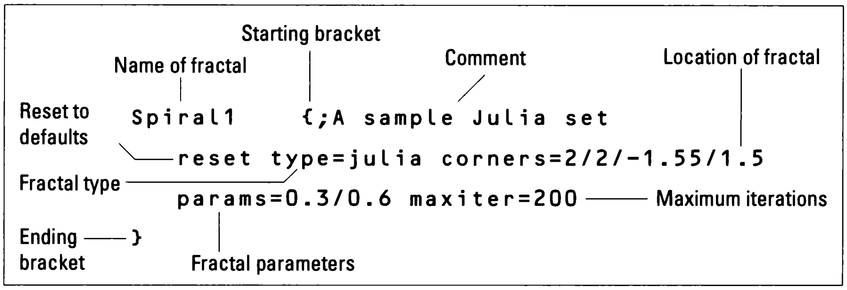
This example selects fractal type Julia, sets the video mode to F3 (VGA 320 x 200 256-color mode) and sets the inside to color number 10 in the palette. All of these settings can also be made using the menu interface and its various submenus.

Table 5-1 lists terminology we will use throughout the rest of this chapter as the commands are documented.

Commands in the SSTOOLS.INI File

When Fractint is first started, it always looks along the DOS path for any file called SSTOOLS.INI (SSTOOLS stands for Stone Soup Tools) and reads startup commands from that file if it exists. Then it looks at its own command line; arguments there will override those from the .INI file. The SSTOOLS.INI command file is used in the same way as Microsoft's TOOLS.INI or WINDOWS.INI configuration files. Sister Stone Soup Group programs, such as the Windows port of Fractint (Winfract) and Lee Crocker's Piclab, also use the SSTOOLS.INI file. You designate a section of SSTOOLS.INI as belonging to a particular program by beginning the section with its label in brackets. Fractint looks for the label [fractint] and ignores any lines it finds in the file belonging to any other labels.

Command-line parameters always appear in the [fractint] section. The commands do not have to be all on the same line; in fact, you may prefer to put each command on its own line for clarity. Comments can be added to



```
Spiral1    {;A sample Julia set
    reset type=julia corners=2/2/-1.55/1.5
    params=0.3/0.6 maxiter=200
}
```

Figure 5-4 An annotated parameter file entry

command-line files by preceding the comment with a semicolon. For example, if an SSTOOLS.INI file looks like this:

```
[fractint]
type=julia     ;start up with a Julia set
inside=0       ;using traditional black
printer=hp     ;my printer is a LaserJet
[startrek]
Aye, captain, but I dinna think the engines can take it!
```

Fractint will read only the second, third, and fourth lines. The last line is for a fictitious program called Startrek.

You can place any sort of Fractint command you like in SSTOOLS.INI, but the normal case is to place commands there that you want to *always* take effect (the `printer=hp` entry, for instance, is a perfect example).

Specifying Command-Line Arguments Interactively

Fractint also provides a way to enter command-line arguments interactively. Whenever you invoke Fractint's G (GIVE COMMAND STRING) command, which is explained in detail later in the chapter, you can enter a text string containing one or more command-line arguments.

Commands in Parameter Files

A powerful extension of the command-line concept is the parameter file. Parameter files contain lists of named fractal images, called parameter entries, and all the command-line arguments needed to generate them. Parameter files have a .PAR file name extension. Many parameter files, each containing a number of different entries, are supplied on the companion disk. A parameter file entry is shown in Figure 5-4 along with labels for the component parts. The name of the parameter entry, in this case SPIRAL1, is followed by a list of commands contained within curly brackets ({ }).

You can display and use parameter file entries using the @ (RUN SAVED COMMAND SET) command, described later in this chapter. You can create parameter file entries by using a text editor, but you will find it easier to generate them automatically using the B (SAVE CURRENT PARAMETERS) command.

Commands in Indirect Files

There is one final method for running commands that is available only from the MS-DOS command line. Command-line arguments can be put in an *indirect* file. If @filename appears in the command line, it causes Fractint to read the file name for any arguments it contains. When it finishes, it resumes reading its own command line. For example, the command line:

```
fractint maxiter=250 @myfile passes=1
```

sets the maximum iterations to 250, opens the file MYFILE, reads and executes the commands in it, and then sets the number of passes to 1. The indirect file option is valid only on the MS-DOS command line, as Fractint cannot deal with multiple indirection (putting the indirect file @filename commands within other indirect files).

For example, if the contents of MYFILE is

```
corners=-4/4/-2/2    ;set the image boundary
type=manowar         ;use this fractal type
biomorph=yes         ;and the Biomorph option
```

then the effect of starting Fractint with:

```
fractint @filename
```

is exactly the same as starting it with:

```
fractint corners=-4/4/-2/2 type=manowar biomorph=yes
```

Fractint can be told to take its commands indirectly from a parameter file entry as well, using the convention @filename/entryname. For example, the command line:

```
fractint @myfile/myentry
```

tells Fractint to open the parameter file MYFILE.PAR, locate the parameter entry in that file named MYENTRY, and take its initial commands contained in that parameter entry.

THE FRACTINT OPERATING MODES

Fractint has several different operating modes, each with its own set of commands. Fractint starts up in the main display mode. The main display mode is the most important because the main functions of Fractint are accessible from within this mode, and it's the one most folks end up using most of the time.

Fractint has several other operating modes, all entered from the main display mode and designed to handle special functions (such as palette editing and orbits displays). When in these modes, Fractint responds to the mouse and keyboard commands differently.

The rest of this chapter documents all the Fractint commands. It is organized using the order in which the available commands are displayed in Fractint's MAIN MENU. Commands are listed with the function they perform first. The alternative means of accessing that function using the menus, keystroke commands, command-line options, and the mouse are then given.

FRACTINT'S MAIN DISPLAY MODE

You are in the main display mode as soon as Fractint is started up, although some of the commands (the SAVE IMAGE command, for example) are not accessible until after you have created an image. The most basic commands may be executed either by using an arrow-key menu or by entering a keystroke. The mouse is used only for manipulating the zoom box in the display mode. Most commands can also be entered as a command-line option.

The Fractint MAIN MENU deals exclusively with display mode commands. The following section is organized according to the major headings of the main arrow-key menu. Remember that you can display the main arrow-key menu by pressing the ENTER key when you're looking at Fractint's initial credits screen or by pressing the ESC key when you're looking at a fractal image.

Note that the MAIN MENU that is shown before you've generated any fractal images is a subset of the one that displays after you have done so, as some of Fractint's commands (such as saving the image) aren't applicable until you have generated an image.

Current Image Commands

The Current Image commands all deal with the current paused or completed graphics images. These commands include returning from the menu back to the graphics image, getting information about the image, controlling the zoom box, and special display modes such as the orbits display.

Continue Calculation

Command Function: Continue calculation—resume a fractal calculation that was interrupted by pressing ESCAPE to access the main menu display.

Menu Access: CONTINUE CALCULATION under the CURRENT IMAGE section of the MAIN MENU.

Command-Line Access: none

Comments: This command switches Fractint from the MAIN MENU back to the current image, resuming the calculation if it was not complete, continuing where it left off. If the image was complete, this command is displayed as RETURN TO IMAGE rather than CONTINUE CALCULATION.

Info About Image TAB

Command Function: Find information about the status of your current image.

Menu Access: INFO ABOUT IMAGE under the CURRENT IMAGE section of the MAIN MENU.

Command-Line Access: none

Comments: This command displays an information screen about the current image, including fractal type, whether the image is complete or not, its corner parameters, time of calculation, parameters values, maximum iterations, and current bailout value used to test when an orbit has escaped. Pressing any key returns to the displayed image and resumes the calculation. The exact content of this screen varies with the options in effect at the time. This command is particularly useful for checking the completion status of an "all-nighter" 1024 x 768 image by telling you whether the image is complete and, if not, which of the multiple passes has been reached.

Zoom Box Functions

The zoom box is the mechanism in Fractint for selecting pieces of fractal images and recalculating them so that they expand to fill the screen. These commands apply only when an image is on the screen. You do not need to wait for Fractint to complete its process of generating an image before bringing up and manipulating the zoom box. There are no menu equivalents for the zoom box commands.

Fractint allows extraordinary control of the zoom box, including such functions as rotating and skewing. If your screen does not have a 4:3 aspect ratio (that is, if the visible display area on it is not 1.333 times as wide as it is high), rotating and zooming will have some odd effects—angles will change, including the zoom box's shape itself, circles (if you are lucky enough to see any with a nonstandard aspect ratio) become noncircular, and so on. The vast majority of PC screens do have a 4:3 aspect ratio.

Zooming is not implemented for some fractal types for which it does not apply, such as the plasma and diffusion fractal types, nor for overlaid and 3-D images. A few fractal types support zooming but do not support rotation and skewing—nothing happens when you try it.

The effect of manipulating the zoom box is the same as resetting the `corners=` value from the command line.

Define Zoom Region (PAGE UP)

Command Function: Define the region in the complex plane within which to carry out a fractal calculation.

Mouse Access: Clicking the left mouse button creates a zoom box. See the mouse zoom box functions in the following sections.

Command-Line Access:
```
corners=xmin/xmax/ymin/ymax[/x3rd/y3rd]
center-mag=[Xctr/Yctr/Mag]
```

Comments: When the command-line option is used and four values are specfied (the usual case), a rectangle is define as follows: *x*-coordinates are mapped to the screen, left to right, from xmin to xmax, and *y*-coordinates are mapped to the screen, bottom to top, from ymin to ymax. Six parameters can be used to describe any rotated or stretched parallelogram: (xmin,ymax) are the coordinates used for the top-left corner of the screen, (xmax,ymin) for the bottom-right corner, and (x3rd,y3rd) for the bottom-left corner. Figure 5-5 shows the relationship of the `corners=` parameters and the zoom box.

Including `center-mag=` on the command line, indirect file, or in SSTOOLS.INI is an alternative way to enter corners as a center point and a magnification. This approach is popular with some fractal programs and publications. Entering just `fractint center-mag=` tells Fractint to use this form rather than corners when saving a parameter file entry using the (B) command. The (TAB) status display shows the corners in both forms. Note that an aspect ratio of 1.3333 is assumed. If you have altered the zoom box proportions or rotated the zoom box, this form can no longer be used. The magnification is relative to a zoom box of width 2. The center-mag form of specifying the zoom box is particularly useful for creating

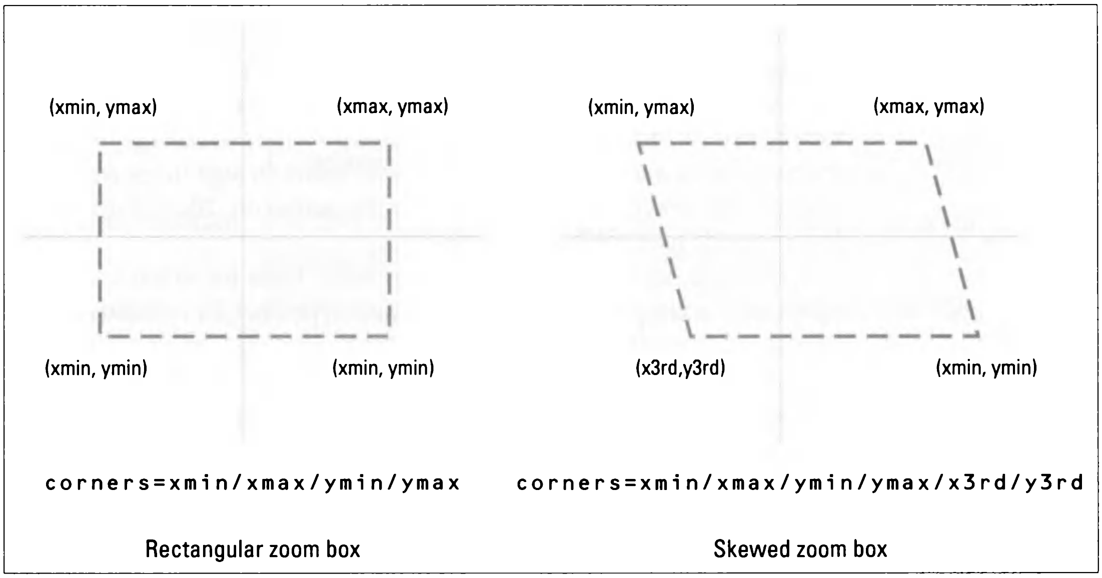


Figure 5-5 The corners of the zoom box mapped to the complex plane

a series of zooms. For example, make a file called ZOOM.BAT with the following lines, replacing F3 with an appropriate video mode for your setup:

```
fractint type=mandel center-mag=-0.1049/0.9278/.12 maxiter=1000 savename=zoom1 batch=yes video=f3
fractint type=mandel center-mag=-0.1049/0.9278/.63 maxiter=1000 savename=zoom2 batch=yes video=f3
fractint type=mandel center-mag=-0.1049/0.9278/3.17 maxiter=1000 savename=zoom3 batch=yes video=f3
fractint type=mandel center-mag=-0.1049/0.9278/15.8 maxiter=1000 savename=zoom4 batch=yes video=f3
fractint type=mandel center-mag=-0.1049/0.9278/79.2 maxiter=1000 savename=zoom5 batch=yes video=f3
fractint type=mandel center-mag=-0.1049/0.9278/396 maxiter=1000 savename=zoom6 batch=yes video=f3
fractint type=mandel center-mag=-0.1049/0.9278/1980 maxiter=1000 savename=zoom7 batch=yes video=f3
fractint type=mandel center-mag=-0.1049/0.9278/9900 maxiter=1000 savename=zoom9 batch=yes video=f3
```

Zoom In (PAGE UP)

Command Function: Resize the zoom box (zoom in).

Mouse Access: Click the left mouse button to create a zoom box. To make the zoom box smaller, hold the left button down and move the mouse up (away from you). To make it larger, hold the left button down and move the mouse down (toward you).

Comments: This action both creates and changes the size of the zoom box. Each time you press (PAGE UP) the box shrinks in size. (PAGE DOWN) increases the size of the zoom box. Figure 5-6 shows a full Mandelbrot image with a zoom box sized and moved to what should be an interesting area.

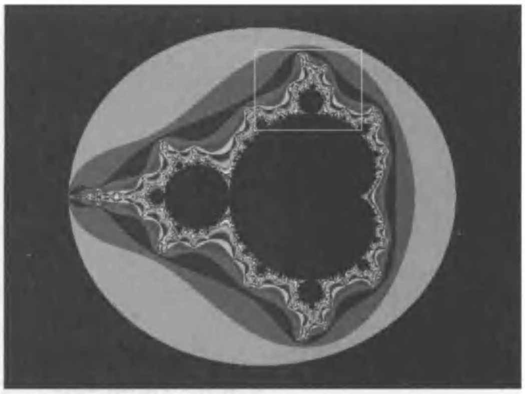


Figure 5-6 A zoom box at an interesting area on the Mandelbrot set

Zoom Out (PAGE DOWN)

Command Function:	Expand the zoom box (zoom out).
Mouse Access:	Hold the left button down and move the mouse down (toward you).
Comments:	If the zoom box is expanded to fill the whole screen, the zoom box settings are reset (useful if you have rotated the zoom box or altered the aspect ratio accidentally). Each time you press (PAGE DOWN) the box enlarges in size.

Move Zoom Box (↑), (↓), (←), (→)

Command Function:	Move ("pan") the zoom box to various screen locations.
Mouse Access:	Move the mouse without pushing either button.
Comments:	It is possible to move the zoom box partially off the screen, so the redrawn image includes points not in the original image. If you are using the keyboard to move the zoom box, holding down the (CONTROL) key while pressing the arrow keys causes the zoom box to move five times farther each time you press an arrow key—a useful feature when you want to move your zoom box a significant distance.

Draw Zoom Box Area (ENTER)

Command Function:	Redraw the area inside the zoom box as a full-screen image.
Mouse Access:	Double-click the left mouse button.
Comments:	Do this when you have the zoom box framing the exact area you want to recalculate as a full-screen image.

Zoom Out and Redraw CONTROL-ENTER

Command Function: Zoom out, so that the screen fills the current zoom box, and redraw the image.

Menu Access: none

Mouse Access: Double-click the right mouse button.

Comments: This function can be thought of as the opposite of the above *draw zoom box area* command—it's a way of zooming *out* rather than zooming *in*. When you perform a zoom out, your new fractal image is generated so that the previous image will be displayed inside the area you have currently outlined with the zoom box.

Rotate Zoom Box CONTROL-− CONTROL-+

Command Function: Rotate the zoom box.

Mouse Access: Move the mouse left or right while holding down the right button.

Comments: CONTROL-− means holding down CONTROL and pressing the numeric keypad's − key. Rotating the zoom box does not work with some fractal types, such as bif+sinpi, bif=sinpi, biflambda, bifurcation, diffusion, julibrot, and L-system—if you try to rotate the zoom box while displaying one of these fractal types, nothing happens.

Zoom Box Aspect Ratio CONTROL-PAGE UP, CONTROL-PAGE DOWN

Command Function: Alters the zoom box aspect ratio by shrinking or expanding its vertical size.

Mouse Access: Move the mouse away from you or toward you while holding both buttons (or the middle button on a three-button mouse).

Comments: The *aspect ratio* of the zoom box is the width of the zoom box in pixels relative to its height in pixels. By default, this aspect ratio matches that of the entire image on your screen. There are no commands to directly stretch or shrink the zoom box horizontally—the same effect can be achieved by combining vertical stretching and resizing. Figure 5-7 shows a zoom box whose aspect ratio has been modified with the CONTROL-PAGE UP key.

Zoom Box Skew CONTROL-HOME, CONTROL-END

Command Function: "Skew" the zoom box, moving the top and bottom edges in opposite directions so it forms a parallelogram rather than a rectangle. CONTROL-HOME moves the top of the zoom box to the right, CONTROL-END moves it to the left.

Mouse Access: Move the mouse left or right while holding both buttons (or the middle button on a three-button mouse).

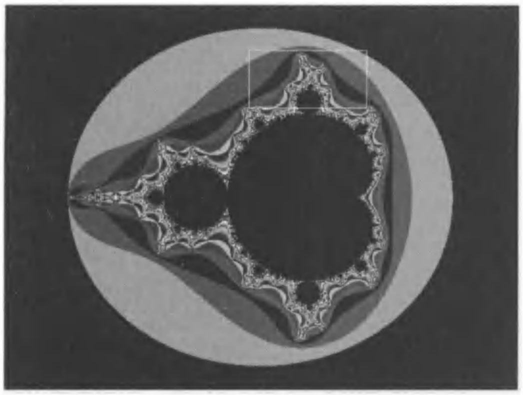


Figure 5-7 A zoom box with a modified aspect ratio

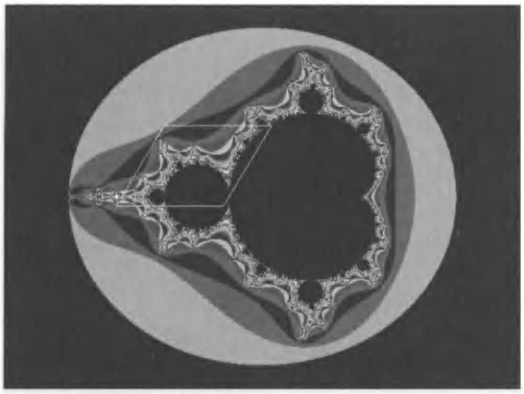


Figure 5-8 A skewed zoom box

Comments: There are no commands to directly skew the left and right sides of the zoom box—the same effect can be achieved by using the available skew functions combined with rotation. Figure 5-8 shows a zoom box that has been skewed with the CONTROL-END key.

Zoom Box Color CONTROL-INSERT, CONTROL-DELETE

Command Function: Change zoom box color. Each time you press CONTROL-INSERT the box will change to the next higher color number. The CONTROL-DELETE command changes the color to the next lower color number.

Mouse Access: Move the mouse away from you or toward you while holding the right button down.

Comments: Changing the zoom box color is useful when you're having trouble seeing the zoom box against the colors around it. For example, if you zoom into an area with very light colors, the white zoom box will blend into the image. Changing the zoom box color to any dark color will make it visible again.

Orbits Window O

Command Function: Show the orbit paths of fractal computations as the fractal is drawn.

Command-Line Access: `orbitdelay=<nnn>`

Comments: There are two separate and completely different display mode commands for the O key. Which one takes effect depends on whether a calculation is currently progressing. Pressing O while an image is still being generated toggles the orbit feature discussed in the next few paragraphs on and off. Pressing the O key after the calculation has completed (or pressing the CONTROL-O keystroke combination whether or not the image is still being calculated) activates an alternate Orbits display, described in the "Completed Image Orbits Display Mode" section that immediately follows.

Displaying Orbits While Generating an Image This version of the Orbits display shows the trajectories of the calculation orbits of each pixel used to create a fractal as the fractal is being drawn. Escape-time fractals work by repeatedly iterating a formula and generating a sequence of complex numbers, while testing whether each number has exceeded a threshold as the sequence is generated. Normally, the results of the interim iteration calculations (the *orbit values*) are not displayed. The O command shows you these orbit values, which can be fascinating to see. To best see this effect, first press X and set passes to 1. The multiple-pass guessing mode that Fractint normally uses makes it hard to see what pixel is being calculated because it fills the screen's black areas with color very quickly; orbits show up best on a black background and so are much easier to see in one-pass mode.

Next, press a function key to start a Mandelbrot fractal calculation. While the image is still being generated, press the O key to see the orbits. In the beginning, you will see white pixels flitting somewhat randomly around the screen. But when the Mandelbrot calculation reaches the lake area, lovely spiral patterns will emerge. If the orbit display is too fast, use the X key to bring up the BASIC OPTIONS menu and, increase the value in the ORBIT DELAY entry to slow it down. Note that the orbits display toggle is disabled whenever you switch to a text space mode display, so that you will have to press the O key again when you return to your graphics image after using that BASIC OPTIONS screen. Figure 5-9 shows a partially completed Mandelbrot image showing an interesting orbit spiral from somewhere in the main lake region.

Completed Image Orbits Display Mode Pressing the O key after your fractal image calculation has completed (or pressing the CONTROL-O keystroke combination whether or not it has completed) activates an alternate Orbits display—actually, an alternate operating mode with its own set of function key operations. Two things happen to your display right away: a small cross-hair cursor shows up in the middle of your screen, and a pop-up window displays the orbit (iteration) values of the formula at the location of the cursor. Moving the cross-hair cursor using either the mouse or the arrow keys causes the orbits display to change based on the new location of the cursor. If you have generated your fractal image with the VIEW WINDOW option active (described later in the chapter), the roles of the

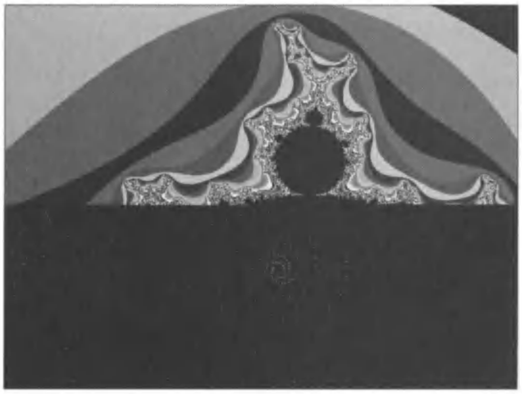


Figure 5-9 Orbits display while an image is being generated

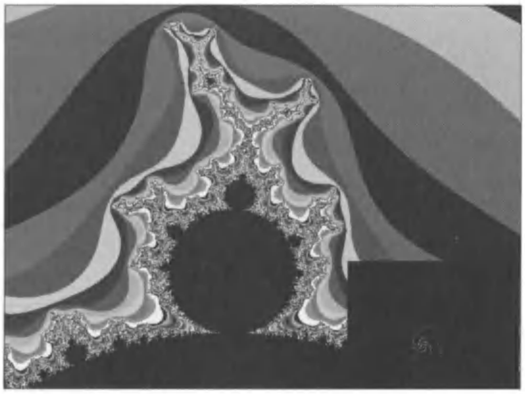


Figure 5-10 An orbits display in a window

two viewing windows are reversed: the fractal image is displayed in a window and the orbits display is displayed full-screen. Figure 5-10 shows an orbit display using this method.

The Orbits display can be modified in many ways:

C	Toggles circle mode on and off. In circle mode, the orbits are displayed not as individual pixels, but as circles with radii inversely proportional to the iteration count (higher iterations produce smaller circles).
L	Toggles line mode on and off. In line mode, consecutive orbit values are connected using lines.
N	Toggles number mode on and off. In number mode, a line of text at the top of the image displays the current pixel location.
P	When you press P, Fractint brings up an entry screen letting you enter a pixel location manually. The cursor moves to that new location and the orbits display follows it.
H	Toggles hide mode on and off. Hide mode only works if you have entered orbits mode with the VIEW WINDOW option enabled (described later in the chapter). When hide mode is on, the original image disappears and the orbits display is the only image shown on your screen.
S	Saves the Fractal, Cursor, Orbits, and Numeric displays as they currently appear on your screen to a file.

< or , (comma)	Makes the scale of the Orbits display smaller.
> or . (period)	Makes the scale of the Orbits display larger.
Z	Restores the scale of the Orbits display to its default value.

Any other keystroke returns you to Fractint's main command level.

New Image Commands

These commands are accessible under the NEW IMAGE section of the MAIN MENU. From this collection of commands, you can select a new video mode, change the current fractal type, toggle between Mandelbrot fractals and the equivalent Julia fractals, and regenerate images you previously made in the same session. All of these commands result in the calculation of a new image.

Select Video Mode (F2 through F10, CONTROL/SHIFT/ALT Combinations)

Command Function: Select the video mode in which the fractal will be displayed.

Menu Access: SELECT VIDEO MODE under the NEW IMAGE section of the MAIN MENU.

Command-Line Access: `video=<mode>`

where <mode> is the keystroke exactly as listed in the VIDEO menu. For example, the popular 320 x 200 256-color VGA mode can be accessed with the command-line option `video=F3`.

Comments: In Fractint the selection of a video mode triggers the recalculation of an image and changes the image resolution. Higher resolutions show more detail but take longer to calculate. A good strategy is to explore using a low-resolution mode such as 320 x 200 (F3) or the VIEW WINDOW option (described later in the chapter) and, if you like the image, recalculate it later using a higher resolution mode. Only certain modes work for any particular video adapter; you should check your graphics adapter documentation or just try different modes and see what works. Don't be dismayed if you try a video mode and get a blank screen; pressing the ESC key will get you back to the MAIN MENU.

You can change the keystrokes that invoke particular video modes if you don't like the defaults Fractint offers. To do this, simply modify Fractint's FRACTINT.CFG file using your favorite text editor.

Video Mode Label	Equivalent Keystrokes
F2	The function key F2
SF2	Press SHIFT and F2 together
CF2	Press CONTROL and F2 together
AF2	Press ALT and F2 together

Table 5-2 Video mode labels and their equivalent keystrokes

FRACTINT.CFG contains a list of all the modes built into Fractint, the same modes that you see when you go to the SELECT VIDEO MODE MAIN MENU item. When you start Fractint, it reads FRACTINT.CFG and builds its internal list of video modes and assigns them function keys based on its contents. Table 5-2 explains the scheme for labeling video modes by keystroke combinations.

Video modes listed without corresponding function keys are accessible only from Fractint's menu interface.

Disk/RAM Video Modes Certain video modes are labeled Disk/RAM Video. These are simulated video modes that do not display on your screen but use extended, expanded, or disk memory, as it is available, to store your image. These modes allow you to create images at higher resolutions than your video equipment supports (up to Fractint's internal limitation of 2048 x 2048 x 256 colors) and then view them in a lower resolution. Keep in mind that these modes need memory—the 2048 x 2048 mode needs 4 megabytes of expanded, extended, or disk memory! These disk video modes are also useful because they allow you to produce fractals in the background under multitasking environments such as Windows or DesQView. In the regular high-resolution video modes, the calculations stop if you switch to another program. Disk video modes have disadvantages. Besides the fact that you can't see a fractal while it is being calculated, disk video modes are slower than normal video modes (especially if Fractint can't find extended or expanded memory to use and must use disk space). Fractint's expanded memory options work with any memory manager supporting EMS 3.2 or later, and its extended memory options work with any memory manager supporting XMS 2.0 or later. Virtually every memory manager in use today supports both of these standards.

If you have one of today's popular super VGA boards, Fractint's SuperVGA/VESA Autodetect modes should work with it. Note that many of the super VGA resolution modes require video boards with more than the standard 256K memory and, therefore, not every board supports all the modes. The 640 x 400 256-color mode is the highest resolution using 256 colors that works with 256K of memory. Resolutions higher than 640 x 480 require that you have a monitor capable of higher resolutions as well. It is quite possible that you have a video adapter that can handle some high-resolution video modes—but a monitor that can't.

If you're not sure if you have a super VGA adapter or which modes it and your monitor support, the easiest approach to take is to try a few. One of three things will happen: Fractint may be able to use that video mode with your hardware, it may detect that it cannot use that video mode with your hardware and bring up a message box telling you so, or it may attempt to generate an image using that video mode and fail miserably. In the latter case, you can always press the (ESC) key to get back to the text-based menu system and try a different video mode.

Following the Autodetect modes are some less standard resolutions that work on various adapters. You should try the ones for your brand of graphics adapter. Don't overlook various "tweaked" modes if you have a VGA. You won't find these listed in your graphics board documentation, because they are achieved by Fractint accessing and directly programming the VGA registers. A favorite mode for many VGA users is the (F10) 320 x 400 256-color mode. This mode has better resolution than the (F3) 320 x 200 mode, and despite being nonstandard, will work on virtually any VGA-equipped system.

If your computer seems to be having problems with Fractint and any of its available video modes, Appendix A, *Fractint and Video Adapters,* describes how Fractint detects and works with different video adapters, problems that can occur, and how to work around them.

Select Fractal Type (T)

Command Function: Presents a list of fractal types from which a fractal can be generated.

Menu Access: SELECT FRACTAL TYPE under the NEW IMAGE section of the MAIN MENU.

Command-Line Access: `type=<type>`

where <type> is the fractal type exactly as listed under the interactive type list.

```
params=xxx[/xxx[/xxx[/xxx]]]
bailout=nnnn
```

where the xxx fields represent fractalspecific entry parameters and the nnnn field represents the fractalspecific bailout value.

Comments: After selecting a type, you will be prompted for any needed parameters and bailout values. Each fractal type uses its own unique fractal parameters. In all cases, default parameter values are provided which generate interesting images—you can press (ENTER) to accept these default vaules, if you want. The types formula, ifs, and lsystem read in a complete list of subtypes from files you can edit, allowing you to create new types. See Chapter 6, *Fractal Types*, for more about the fractal types available in Fractint.

Toggle To/From Julia (SPACEBAR)

Command Function: Toggle between Mandelbrot and Julia fractal types.

Menu Access: TOGGLE TO/FROM JULIA under the NEW IMAGE section of the MAIN MENU.

Command-Line Access: none

Comments: As explained in Chapter 2, *Fractals: A Primer,* each point of a Mandelbrot set corresponds to a Julia set. Fractint allows you to see this connection very clearly. Create an image of the classic Mandelbrot set (`type=mandel`, the default fractal type when you first start up Fractint). With the image on the screen, press the (SPACEBAR) key. Two things will happen: a cross-hair cursor will appear in the center of your image, and a small window will open up showing the Julia set corresponding to the location of that cross-hair cursor. The Julia set is generated using the julia_inverse algorithm, which generates basic outlines of Julia sets quickly and then fills them in later. As you move the cross-hair cursor around over the Mandelbrot set, the corresponding Julia set changes accordingly. Pressing the (SPACEBAR) key again causes Fractint to switch over to a full-screen display of the current Julia set. Pressing the (SPACEBAR) key a third time brings back the original, full-screen Mandelbrot display. If you have generated that Mandelbrot set with the VIEW WINDOW option active (described later in the chapter), the roles of the two viewing windows are reversed: the Mandelbrot set is displayed in a window and the Julia set is displayed full-screen. Figure 5-11 shows a full Mandelbrot image with a Julia set displayed in a window.

The Julia display can be modified in several ways:

(N) Toggles number mode on and off. In number mode, a line of text at the top of the image displays the current pixel location.

(P) When you press (P), Fractint brings up an entry screen letting you enter a pixel location manually. The cursor moves to that new location and the Julia display follows it.

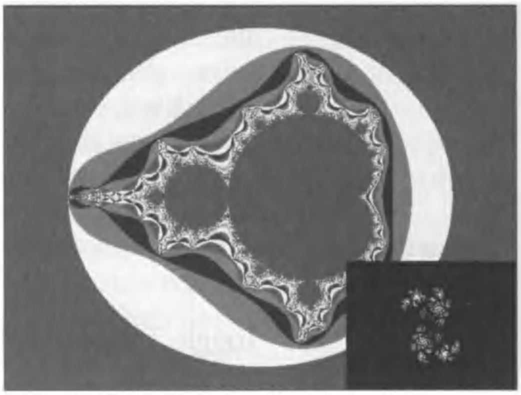


Figure 5-11 Mandelbrot image with a Julia set window

H	Toggles hide mode on and off. Hide mode only works if you have entered orbits mode with the View Window option enabled (described later on in the chapter). When hide mode is on, the original Mandelbrot image disappears and the Julia display is the only image shown on your screen.
S	Saves the fractal, cursor, and numeric display as they currently appear on your screen to a file.
SPACEBAR	Switches to full-screen display of the current Julia set using the standard `type=julia` algorithm. Pressing the SPACEBAR key a second time returns to your original Mandelbrot display.
< or , (comma)	Makes the scale of the Julia display smaller.
> or . (period)	Makes the scale of the Julia display larger.
Z	Restores the scale of the Julia display to its default value.

Any other keystroke returns you to Fractint's main command level.

There are many other fractal types that share this Mandelbrot/Julia relationship, and Fractint can be used to show those relationships, although perhaps not quite as dramatically as it does with the classic Mandelbrot/Julia fractals. (The presence of the quick julia_inverse algorithm lets Fractint generate its Julia sets "on the fly".) Select any of the fractal types that begin with the letters `man...` and then press the SPACEBAR key with that image on the screen. Fractint switches to the Julia set display mode and brings up the cross-hair cursor, but without the matching "Julia set" formula display. The command keys work as previously described, although some of them (such as the <, >, and Z keys) have no effect. Pressing the SPACEBAR key a second time causes Fractint to switch over to display the "julia" fractal type corresponding to the location of the cross-hair cursor. Pressing the SPACEBAR key a third time brings back the original "man..." image.

Remember that each Julia type is in fact an infinite collection of quite different fractals, depending on the values of the parameters. The characteristics of each Julia set can be inferred from the appearance of the Mandelbrot set near the point that generates the Julia set. For example, if the cursor is pointing somewhere deep inside a Mandelbrot lake, the corresponding Julia will have a large lake. Conversely, if the cursor is residing somewhere on land, the Julia will not have a large lake. The most interesting Julia sets may be found from points near the lake edge of the corresponding Mandelbrot set, where the chaos is the greatest.

Return to Prior Image \

Command Function: Redraw the previous image.

Menu Access: Return To Prior Image under the New Image section of the Main Menu.

Command-Line Access: none

Comments: As you make a series of images, Fractint remembers the zoom coordinates and fractal types of the last 25 images. The X command causes the zoom parameters and type to be set to the previously generated fractal. Repeatedly pressing \ causes Fractint to back through the list and recalculate your previous images. Use this when you have made an ill-advised zoom into a boring area and you don't want to start over, but just want to retreat to an image previously generated.

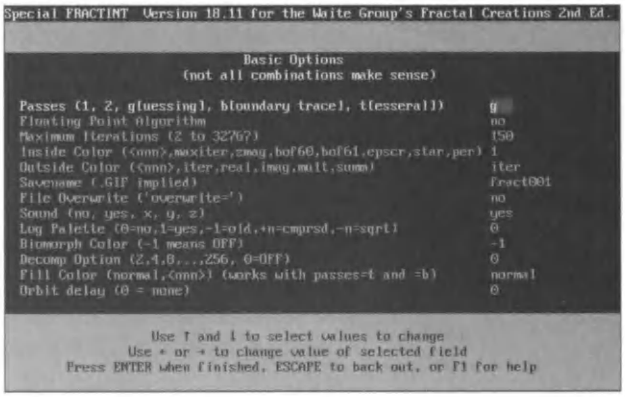
Options

The Options section of the Main Menu allows many of the features of Fractint to be accessed interactively. These cover a wide range of different effects and alternatives. We'll cover them here in the order they are encountered in the menus.

Basic Options X

Command Function: Access basic options.

Menu Access: Basic Options under the Options section of the Main Menu.



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

                              Basic Options
                    (not all combinations make sense)

 Passes (1, 2, g[uessing], b[oundary trace], t[esseral])       g
 Floating Point Algorithm                                      no
 Maximum Iterations (2 to 32767)                               150
 Inside Color (<nnn>,maxiter,zmag,bof60,bof61,epscr,star,per)  1
 Outside Color (<nnn>,iter,real,imag,mult,summ)                iter
 Savename (.GIF implied)                                       fract001
 File Overwrite ('overwrite=')                                 no
 Sound (no, yes, x, y, z)                                      yes
 Log Palette (0=no,1=yes,-1=old,+n=cmprsd,-n=sqrt)             0
 Biomorph Color (-1 means OFF)                                 -1
 Decomp Option (2,4,8,...,256, 0=OFF)                          0
 Fill Color (normal,<nnn>) (works with passes=t and =b)        normal
 Orbit delay (0 = none)                                        0

                  Use ↑ and ↓ to select values to change
               Use ← or → to change value of selected field
        Press ENTER when finished, ESCAPE to back out, or F1 for help
```

Figure 5-12 Fractint's Basic Options (X) Input screen

Command-Line Access: various (see the following text)

Comments: The distinction in Fractint between "basic" and "extended" options is somewhat arbitrary. The (X) Basic Options input screen is shown in Figure 5-12, with the default values showing.

Set Passes Options (X)

Command Function: Set the passes algorithm options.

Menu Access: Passes (1, 2, G[uessing], B[oundary Tracing], T[esseral])" on the Basic Options <x> menu.

Command-Line Access: `passes=1|2|guess|btm|tesseral`

Remember that 1|2|Guess|Btm|Tesseral means to use one of those five possibilities. Example:

```
passes=btm
```

Comments: The Passes option selects single-pass, dual-pass, solid-guessing, boundary tracing, or the tesseral algorithm. The single-pass mode draws the screen pixel by pixel and is the slowest. The dual-pass mode generates a "coarse" screen first as a preview using 2 x 2-pixel boxes and, when the screen is filled, generates the rest of the dots with a second pass. The effect is to quickly get a coarse view of the fractal so you can exit early if you decide you don't like it.

Solid-guessing is the fastest mode, because it attempts to avoid calculations by guessing the color of pixels surrounded by pixels of one color. It performs from two to four visible passes—more in higher-resolution video modes. Its first visible pass is actually two passes—one pixel per 4 x 4, 8 x 8, or 16 x 16 pixel box (depending on number of passes) is generated, and the guessing logic is applied to fill in the blocks at the next level (2 x 2, 4 x 4, or 8 x 8). Subsequent passes fill in the display at the next-finer resolution, skipping blocks that are surrounded by the same color. The multiple passes are for two reasons. The first is to give you a quick preview of the image in case you don't want to wait for it to complete. The second reason is that the guessing algorithm works in stages, starting with a rough approximation. Solid-guessing can guess wrong, but it guesses wrong quickly.

The Tesseral algorithm is a variant of the super-solid guessing algorithm that works by continually dividing the screen into quarters, calculating all of the pixel values of the rectangular border of each quadrant, and filling in an entire rectangle if its boundary is all the same color.

Boundary tracing is a completely different approach from the others. It only works accurately with fractal types that do not contain "islands" of colors (such as the Mandelbrot set), but not those that do (such as the Newton type). Boundary tracing works by finding a color boundary, tracing it around the screen, and then filling in the enclosed area. The idea of this algorithm is to speed up calculations, but in Fractint solid-guessing is almost always faster. We have included boundary tracing anyway because it is so much fun to watch! Boundary tracing does not work when the inside color is set to 0 (black), because it uses 0 to determine whether a color has been written to the screen already.

To select one of these options from the (X) screen, at the PASSES (1, 2, G[UESSING], B[OUNDARY TRACE], or T[ESSERAL]) prompt, type in `1` for one pass, `2` for two passes, `g` or `guess` for guessing, `b` or `btm` for boundary-tracing method, or `t` for the tesseral algorithm.

Understand that the single-pass and dual-pass modes result in exactly the same image and take the same amount of time. They work best for fractal purists who do not want to risk the occasional inaccuracies of the default guessing mode. Most of the time, the solid-guessing mode is the one to use, and it is usually the default. If you are the type who is fascinated by watching intriguing algorithms at work, by all means try the boundary tracing and tesseral options.

Floating-Point/Integer Toggle (F)

Command Function: Change between floating-point and integer math for calculating fractals.

Menu Access: FLOATING-POINT ALGORITHM under the BASIC OPTIONS <x> menu.

Command-Line Access: `float=yes`

Comments: Most fractal types have both a fast integer math and a floating-point version. The faster, but sometimes less accurate, integer version is the default. If you have an

Intel 80486-based PC or other fast machine, such as an 80286 or 80386 with a math coprocessor such as the 80287 or 80387, or if you are using the continuous potential option (which looks best with high bailout values not possible with Fractint's integer math implementation), you may prefer to use floating-point instead of integer math.

To force Fractint to use floating-point, you can add `float=yes` at the command line, use the (F) key to toggle between integer math and floating-point math, or specify floating point at its entry on the Basic Options menu. Fractint also automatically changes to floating-point math when you zoom deeply into an image beyond the limited range of its faster integer math routines. This will be seen by the sudden slowness of regeneration when zooming if your computer does not have a floating-point math unit (FPU).

If you want to run some comparison speed tests, the TAB status key adds a line (in its upper right-hand corner) that mentions when floating-point is being used and also reports the time taken to generate the current fractal. On a 33-MHz 80386-based PC, the default Mandelbrot set (type mandel) using the (F3) video mode takes 1.98 seconds. After you have pressed (F) to use floating-point instead of integer math, the time increases to 3.46 seconds using an 80387 floating-point coprocessor, and 57.5 seconds without a coprocessor. These results will vary a lot depending on which CPU chip your machine is using, and their clock speed—in fact, when run on a PC using an 486DX with its on-chip floating-point unit, Fractint's integer and floating-point algorithms run at nearly the same speed.

Set Maximum Iteration (X)

Command Function: Set the maximum iteration at which an escape-time fractal formula considers a point to have "escaped its orbit."

Menu Access: Maximum Iterations (2 to 32767) under the Basic Options <x> menu.

Command-Line Access: `maxiter=<nnn>`

where nnn is a number from 2 to 32767.

Comments: Recall that the escape-time algorithm creates fractal images by repeatedly iterating a formula and testing whether the orbit wanders outside the bailout threshold. Because many orbits never escape the bailout radius, Fractint must have a limit to how many iterations it will try before giving up, or the computation will go on forever. That limit is the maximum iterations value, and it has a default of 150. The limit causes some inaccuracy in the final fractal. For example, there are points near the lake shore of the Mandelbrot set whose orbits have not escaped after 150 iterations, but which would have escaped after a few more iterations if the calculation had been extended. These points might be plotted as part of the lake, when they really belong on the shore. The higher the maximum iterations cutoff, the more accurate the final image, but the slower the calculation. As a practical matter, the default Mandelbrot image looks fine with 150 iterations. As you zoom in further, however, you may need to increase the iteration limit as the inaccuracies become visible.

To see the effect of setting the maximum iteration limit, press X from the MAIN MENU or while viewing a fractal, and set the MAXIMUM ITERATIONS (2 TO 32767) value to 3. (The value 2 creates a solid blue image unless the inside value is set to something other than 2.) You will see a single oval band surrounding the lake, which consists of all the points whose orbits did not escape after 3 iterations. Now, press X again and set maximum iterations to 4. You will see one more band, and the lake will be a little smaller. After trying a few higher values, you will see why the value 150 is fine for the default Mandelbrot. A higher value makes no visual difference at that magnification.

Set Maximum Iteration (Inside) Color X

Command Function: Set the color assigned to points that pass the maximum iterations limit (the lake color).

Menu Access: INSIDE COLOR (NNNN, MAXITER, ZMAG, BOF60, BOF61, EPSCROSS, STARTRAIL, PERIOD) item on the BASIC OPTIONS <X> menu.

Command-Line Access: `inside=<nnn>|maxiter|zmag|bof60|bof61|epscross|startrail|period`

Comments: The *inside* option lets you set the color of the lake area of a fractal, which consists of the points whose orbits had still not escaped when the maximum iteration cutoff was reached (see the earlier discussion of maximum iterations). For example, setting inside to 0 makes the Mandelbrot fractal interior lake black, because color 0 is black in the standard IBM palette. (If you change the palette by cycling colors, 0 might be a different color.) Setting inside to maxiter makes the inside color the same as the maximum iteration value you are using, which is useful for 3-D purposes.

Other options reveal hidden structure inside the lake. `Inside=bof60` and `inside=bof62` are named after the page numbers in our copy of *The Beauty of Fractals* where we first saw these plotted. If you set `inside=bof60`, the lake area will be broken into colored areas where the iteration number of the closest orbit approach to the origin is the same. If you set `inside=bof61`, you will see the lake broken into colored areas where the closest value of the orbit to the origin is the same. Setting `inside=zmag` colors the inside pixels based on the magnitude of their order point when maxiter was reached. Setting `inside=period` colors the inside pixels based on their periodicity as detected using Fractint's periodicity-detection logic—pixels with a three-cycle periodicity are shown using color 3, for example.

Finally, there is the EPSCROSS option, which colors the inside pixels based on whether their orbits swung close to the *x* or *y* axis, and there is the STARCROSS option, which colors them based on clusters of points in the orbits.

Don't worry if you don't understand all these options; just try them to see what they look like!

Set Outside Color ⓧ

Command Function: Set the color of escape-time points with iterations less than the maximum iterations.

Menu Access: OUTSIDE COLOR (<NNN>, ITER, REAL, IMAG, MULT, SUMM) item on the BASIC OPTIONS <x> menu.

Command-Line Access: `outside=<nnn>|iter|real|imag|mult|summm`

where <nnn> is a number from 0 up to the number of colors of the current video mode.

Comments: As you might guess, this function is the opposite of the inside option. The inside option sets the color of the lake, which is to say the points of the Mandelbrot set. The outside option concerns itself with all the areas outside the lake.

Throughout this book we have discussed the Mandelbrot fractal or image instead of the Mandelbrot set. The reason is that the Mandelbrot set consists of just the interior lake; all the striped colors of the usual fractal image of the Mandelbrot are not part of the set at all! The first outside option was born when the authors received a letter from a high school math teacher who wanted to see just the Mandelbrot set (the part colored with the inside option), and not the distracting stripes outside the set.

The classic method of coloring outside the fractal is to color according to how many iterations were required before *z* reached the bailout value, usually 4. This is the method used when outside is set to Iter.

Setting the outside color to a numeric value *nnn* sets the color of the exterior to some number of your choosing. For example, if OUTSIDE COLOR is 1, all points not inside the fractal set are displayed as color 1 (blue). Note that defining an outside color forces any image to be a two-color one: either a point is inside the set, or it is outside it.

However, when z reaches bailout, the real and imaginary components can be at very different values. Setting the outside value to real or imag colors the outside pixels using the iteration value plus the real or imaginary values. If outside is summ, Fractint uses the sum of all these values to determine the pixel color. These options can give a startling 3-D quality to otherwise flat images and can change some boring images to wonderful ones. The mult option colors outside pixels by multiplying the iteration by real divided by imaginary. There was no mathematical reason for this coloring scheme—it just seemed like a good idea at the time.

Set Default Saving File Name ⓧ

Command Function: Set the default file name for saving images with the ⓢ command.

Menu Access: SAVENAME (.GIF implied) item on the BASIC OPTIONS <x> menu.

Command-Line Access: `savename=<filename>`

Comments: When you save an image with the ⓢ command, Fractint creates file names like FRACT001.GIF and increments the number automatically as more files are saved. You can change the default file name with the savename option. This is particularly useful when you are creating a collection of files at once using the batch mode and you want the file name to remind you what the fractal is. For example, make a file called SAVENAME.BAT with these lines:

```
fractint type=mandel savename=mandel video=f3 batch=yes
fractint type=manowar savename=manowar video=f3 batch=yes
```

If you don't have a VGA, use a different video mode that works with your adapter, such as F2 for EGA or F5 for CGA. Running this batch file will create two files, MANDEL.GIF and MANOWAR.GIF. Fractint will replace the last letter in your savename with a number if you save several images in a single Fractint session after setting the savename.

Note that even when you specify a savename, if you save more than one image during a session an incrementing number will appear at the end of the file name. For example, if you start Fractint with:

```
fractint savename=test
```

the successive names used for saving will be TEST.GIF, TES1.GIF, TES2.GIF and so forth.

Set File Overwrite Flag ⓧ

Command Function: Set the file overwrite flag.

Menu Access: FILE OVERWRITE (OVERWRITE=) item on the BASIC OPTIONS <x> menu.

Command-Line Access: `overwrite=no|yes`

The default value is "no."

Comments: If `overwrite=yes`, the file names used in a Fractint session will overwrite existing files from previous sessions with the same names. Files created during the same session will still not be overwritten because the file names will contain incrementing numbers. If `overwrite=no`, files will not be overwritten.

Set Sound Effects ⓧ

Command Function: Disable sound effects or attach sound to an orbit coordinate to make fractal music.

Menu Access: Sound (No, Yes, x, y, z) item on the Basic Options <x> menu.

Command-Line Access: `sound=off|x|y|z`

Use one of off, x, y, or z. The default is on.

Comments: The off option disables the beeps that tell you your fractal is done or that you have made an error. The `sound=x|y|z` options are for the "attractor" fractals, like the Lorenz fractals, control the frequency of the sound on your PC speaker as they are generating an image, based on the *x*-, *y*-, or *z*-coordinate the fractal is displaying at the moment. In other words, `sound=y` means the *y*-axis pixel values will control the frequency of the tone generator. The effect depends on the speed. If the sound changes too fast for your taste, try using the Ⓕ key to toggle to floating-point math and slow the "music" down.

Fractint's sound routines support the standard PC speaker only, and cheerfully ignore any fancy sound equipment (such as Sound Blaster boards) that might also reside on your system.

Use Log Map ⓧ

Command Function: Map iterations to colors with a logarithmic mapping.

Menu Access: Log Palette (0=No, 1=Yes, -1=Old, +n=cmprsd, -n=sqrt) item on the Basic Options <x> menu.

Command-Line Access: `logmap=yes|old|<nnn>`

Comments: Normally, escape-time iterations are mapped one-to-one to palette colors, which causes areas with a high iteration count to lose detail because the colors change so rapidly that the "stripes" are too close together for you to see any pattern. Turning this option on causes colors to be mapped to the logarithm of the iteration, revealing structure in the featureless areas of more chaotic coloring. Entering a positive number causes a variable degree of logarithmic compression to be used; a negative number causes quadratic compression. When using a logarithmic palette in a 256-color mode, we suggest changing your colors from the usual defaults. For example, the last few colors in the default IBM VGA color map are black, which results in points nearest the lake smearing into a single dark band, with little contrast from the blue lake.

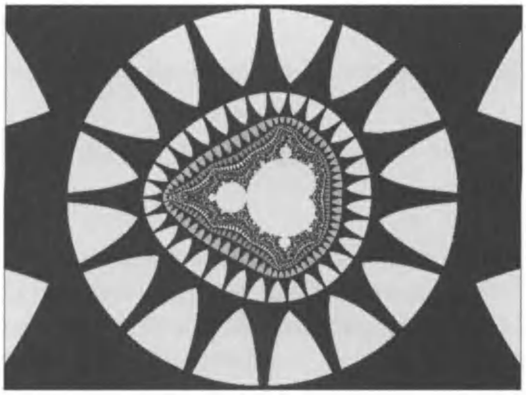


Figure 5-13 A Pickover Biomorph

Entering a value of +2 or −2 causes Fractint to first scan the borders of the image looking for the minimum iteration value (under the assumption that this is the minimum iteration value that will exist on the final image) and then set the `logmap=` value to that minimum value. This tends to cause Fractint to use all of the existing palette values in its logarithmic palette mapping.

Enable Biomorph Rendering $\boxed{X}$

Command Function: Turn on biomorph rendering of escape-time fractals.

Menu Access: Biomorph Color (−1 Means Off) item on the Basic Options <x> menu.

Command-Line Access: `biomorph=<nnn>`

Comments: Related to binary decomposition (see following description) are the biomorphs invented by Clifford Pickover and discussed by A. K. Dewdney in his "Computer Recreations" column in the July 1989 *Scientific American*, page 110. These are so named because this coloring scheme makes many fractals look like one-celled animals. Figure 5-13 shows an example of what appears to be a giant biomorph.

To create biomorphs, the normal escape-time coloring is modified so that if either the real *or* the imaginary component is less than the bailout, then the pixel is set to the biomorph color. The effect is a bit better with higher bailout values: the bailout is automatically set to 100 when this option is in effect. You can try other values with the `bailout=nnn` option. The biomorph option is turned on by setting the biomorph value to a positive number, taken to be the color to use on the affected pixels. When toggling to Julia sets, the default corners are three times bigger than normal to allow the biomorph appendages to be seen. This option

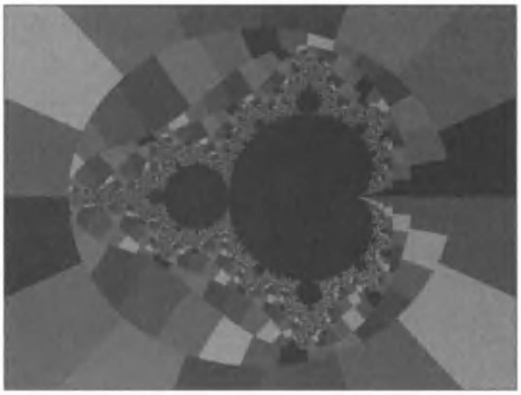


Figure 5-14 The Mandelbrot fractal using decomposition

does not work with all types. In particular it fails with any of the mandelsine family. However, if you are stuck with monochrome graphics, you should try it, as it works very well in two-color modes. Try it with the marksmandel and marksjulia types.

Use Binary Decomposition ⓧ

Command Function: Use the binary decomposition method when rendering escape-time fractals.

Menu Access: DECOMP OPTION (2,4,8,..,256, 0=OFF) item on the BASIC OPTIONS <x> menu.

Command-Line Access: `decomp=0|2|4|8|16|32|64|128|256`

Pick one of these values; the default is 0, which means decomposition is off.

Comments: Most escape-time fractal types are calculated by iterating a simple function of a complex number, producing another complex number, until either the number exceeds some predefined bailout value, or the iteration limit is reached. The pixel corresponding to the starting point is then colored based on the result of that calculation.

The decomposition command turns on another coloring method. Here the points are colored according to which section of the complex plane the final value is in. The decomposition parameter determines how many sections the plane is divided into for this purpose. The result is a kind of warped checkerboard coloring, even in areas that would ordinarily be part of a single contour. Figure 5-14 shows what the default Mandelbrot fractal looks like with `decomp=8`.

Fill Color ⓧ

Command Function: Select the color used as filler when using the boundary-tracing and tesseral algorithms.

Menu Access: FILL COLOR (NORMAL, <NNN>) item on the BASIC OPTIONS <X> menu

Command-Line Access: `fillcolor=<nnn>`

Comments: The Fill Color option only affects the boundary-tracing and tesseral algorithms (`passes=b` or `t`). It causes them to use a fixed color instead of the boundary color whenever they fill in an area. This gives you a pretty clear idea of just how much of the fractal image these algorithms avoided calculating

Orbit Delay ⓧ

Command Function: Slow down the optional orbits display.

Menu Access: ORBIT DELAY (0=NONE) item on the BASIC OPTIONS <X> menu.

Command-Line Access: `orbitdelay=<nnn>`

Comments: The ORBIT DELAY option affects only the speed of the display of the "orbit" pixels shown when you are using the orbits option during image generation. The default setting of zero does not delay the display at all. Higher values delay the display more dramatically.

Extended Options ⓨ

Command Function: Access the EXTENDED OPTIONS menu, which contains a loose collection of fractal options.

Menu Access: EXTENDED OPTIONS under the OPTIONS section of the MAIN MENU.

Command-Line Access: various (see the following text)

Comments: The ⓨ option input screen is shown in Figure 5-15 with the default values showing.

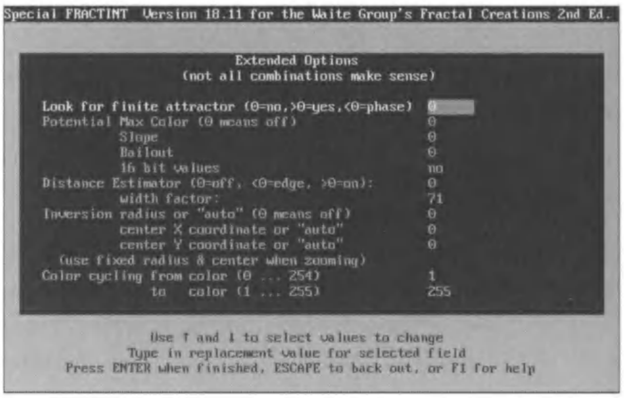
Look for Finite Attractor ⓨ

Command Function: Invoke the basins of finite attractor option for coloring Julia lakes.

Menu Access: LOOK FOR FINITE ATTRACTOR item on the EXTENDED OPTIONS <Y> menu.

Command-Line Access: `finattract=no|yes|phase`

The default value is "no."



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

                        Extended Options
                (not all combinations make sense)

  Look for finite attractor (0=no,>0=yes,<0=phase)   0
  Potential Max Color (0 means off)                  0
            Slope                                    0
            Bailout                                  0
            16 bit values                            no
  Distance Estimator (0=off, <0=edge, >0=on):        0
            width factor:                            71
  Inversion radius or "auto" (0 means off)           0
            center X coordinate or "auto"            0
            center Y coordinate or "auto"            0
    (use fixed radius & center when zooming)
  Color cycling from color (0 ... 254)               1
                to   color (1 ... 255)               255

              Use ↑ and ↓ to select values to change
             Type in replacement value for selected field
      Press ENTER when finished, ESCAPE to back out, or F1 for help
```

Figure 5-15 The Extended Options menu

Comments: This is another option that colors some Julia lakes, showing the escape time to finite attractors. It works with the lambda and magnet types and others.

A finite attractor is a point within a Julia set that captures the orbits of points that come near. By "capture" we mean that if this option is turned on, Fractint attempts to locate such a finite attractor, and then to color the inside of the lake according to the time of escape of that attractor. This is an exact analogy to the way the normal escape-time algorithm colors points according to escape time to infinity. Another way to put this is that this option graphs the level sets of the basin of attraction of a finite attractor.

For a quick demonstration, select a fractal type of lambda, with real and imaginary parts of the parameter both equal to .5. You will obtain an image with a large blue lake. Now set LOOK FOR FINITE ATTRACTOR to "yes" with the Y menu. The image will be redrawn from scratch, this time with a much more multicolored lake. A finite attractor lives in the center of one of the resulting ripple patterns in the lake—turn the Orbits display (O) on if you want to see where it is; the orbits of all initial points that are in the lake converge there. Figure 5-16 shows the result.

If the *phase* option is used, Fractint colors its attractor values using the phase of the convergence to a finite attractor rather than the time of its escape.

Use Distance Estimator Method Y

Command Function: Use the distance estimator algorithm when rendering Mandelbrot and Julia fractals.

Menu Access: DISTANCE ESTIMATOR METHOD (0 MEANS OFF) item on the EXTENDED OPTIONS <Y> menu.

Command-Line Access: `distest=<nnn>`

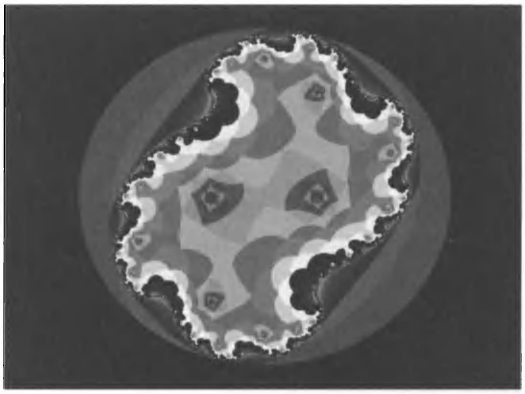


Figure 5-16 Finite Attractors in Lambda Lake

The default is 0 (distance estimator turned off).

Comments: This is Fractint's implementation of an alternative method for rendering the Mandelbrot and Julia sets, based on work by mathematician John Milnor and described in *The Science of Fractal Images*. While this alternative method takes full advantage of your color palette, one of its best uses is in preparing monochrome (single-color) images for a printer. Using the 1600 x 1200 2-color disk video mode and an HP LaserJet, you can generate fractals of quality equivalent to the black-and-white illustrations of the Mandelbrot set in *The Beauty of Fractals*.

The distance estimator method has the effect of widening the very thin strands which are part of the inside of the set. Instead of hiding invisibly between pixels, these strands are made one pixel wide. This method is designed to be used with the classic Mandelbrot and Julia types, and it may work with other escape-time fractals.

To turn on the distance estimator method, set the distest value on the Basic Options <x> screen to a nonzero value. If you set distest to 1, you should also set inside to something other than 1, or you will get a solid blue fractal. You should use the one-pass or two-pass mode—solid-guessing and boundary tracing can miss some of the thin strands made visible by the distance estimator method. For the highest-quality images, maxiter should also be set to a high value, say 1000 or so. You'll probably also want inside set to zero, to get a black interior.

In color modes, the distance estimator method also produces more evenly spaced contours. Set distest to a higher value for narrower color bands, a lower value for wider ones. A good value to start with is 1000. Setting distest automatically also toggles to floating-point mode. When you reset distest back to zero, remember to turn off floating-point mode if you want it off.

Unfortunately, images using the distance estimator method can take many hours to calculate even on a fast machine with a coprocessor. Therefore, you should not use the distest option for exploration, but use it only after you have found interesting fractals.

Set Continuous Potential Parameters Ⓨ

Command Function: Invoke the continuous potential option and control coloring (change stripes into continuously varying hues).

Menu Access: Use the following items on the EXTENDED OPTIONS <Y> menu:

POTENTIAL MAX COLOR (0 MEANS OFF)	(default 0)
SLOPE	(default 0)
BAILOUT	(default 0)
16-BIT VALUES	(default no)

Command-Line Access: `potential=<maxcolor>[/<slope>[/<bailout>[/16bit]]]`

Comments: Fractint's escape-time fractal images are usually calculated by the *level set* method, producing bands of color. Each of these bands consists of all points whose orbit exceeded the bailout threshold at the same iteration. The continuous potential option makes colors change continuously, rather than breaking the image into bands or stripes. A 256-color VGA video mode is mandatory to appreciate this effect, as it is impossible to show continuous variation with only 4 or 16 colors. Non-3-D continuous potential images sometimes have a 3-D appearance because of the smoothly changing colors. Color cycling a continuous potential image with the ⊕ command gives a totally different effect than you experience with a normal striped fractal. The colors ooze rather than flash.

MAX COLOR is the color corresponding to zero potential, which plots as the top of the mountain. Generally, this should be set to one less than the number of colors, for example, 255 for VGA. Remember that the last few colors of the default IBM VGA palette are black, so you won't see what you are really getting until you change to a different palette.

SLOPE is a number that determines how fast the colors change (try 2000 or so). If this value is too high, there will be large solid areas with the color 0; if it is too low, only a limited segment of possible colors will appear in the image. In 3-D transformations, this value determines the steepness of the mountain slopes.

BAILOUT is a number that replaces the normal escape-time bailout (set at 4). Larger values give more accurate and smoother potential—try 200.

16-Bit Values is a flag that makes Fractint save the file as a double-wide 16-bits-per-pixel GIF file. Use this flag if you want to try a 3-D transformation of the image. The 16 bits per pixel results in a smoother 3-D image. If you do not turn on this flag but save the file in the normal way, then the potential value will be truncated to an integer, resulting in a rougher 3-D image. When this flag is turned on, saved file names will have the extension .POT, short for "potential." You can load these files back into Fractint with the (R) command the same way normal GIF files are loaded back in. However, the .POT files will look strange when viewed with GIF decoders other than Fractint.

Creating 3-D Landscapes Continuous potential is particularly useful when creating 3-D landscape images from fractals. When viewed in 3-D, the stripes of a typical noncontinuous-potential image turn into something like Chinese terraces; most of the surface appears to be made up of colorful horizontal steps. This effect may be interesting, but it is not suitable for use with the illuminated 3-D fill options 5 and 6. Continuous potential smoothes the steplike terraces into a continuous surface, so that the illumination results in graduated shades of color.

Internally continuous potential is approximated in Fractint by calculating as follows:

$$\text{potential} = \frac{\log(\text{modulus})}{2^{\text{iterations}}}$$

where "modulus" is the magnitude of the iterations orbit value—the first orbit value that exceeded the bailout. The term "potential" comes from the fact that this value is related to the electrical potential field surrounding the lake that would result if it were electrically charged.

Here is a pointer for using continuous potential. Fractint's criterion for halting a fractal calculation, the bailout value, is generally set to 4, but continuous potential is inaccurate at such a low value. The integer math which makes the mandel and julia types so fast imposes a hard-wired maximum bailout value of 127. You can still make interesting images with these bailout values, such as ridges in the fractal hillsides. However, this bailout limitation can be avoided by turning on the floating-point algorithm option from the Basic Options <x> menu or by adding `float=yes` to the Fractint command line.

Creating Mt. Mandelbrot Using Continuous Potential Options The following commands can be used to re-create the image we call Mt. Mandelbrot. Type the following into a file called MTMAND. If you invoke Fractint from the DOS prompt as "fractint @mtmand," these options will take effect.

```
TYPE=mandel
CORNERS=-0.19920/-0.11/1.0/1.06707
INSIDE=maxiter
MAXITER=255
POTENTIAL=255/2000/1000/16bit
PASSES=1
FLOAT=yes
SAVENAME=mtmand
```

Use a 256-color video mode. (If you don't have a graphics adapter with a 256-color mode, use a disk video mode. You won't be able to see the file right away, but you can convert it to 3-D later and then see it.) See the 3-D section for how to generate a 3-D image from the resulting MTMAND.POT file.

Invert Image Ⓨ

Command Function: Invert an image for viewing in a cylindrical mirror

Menu Access: Use the following items on the EXTENDED OPTIONS <Y> menu:

INVERSION RADIUS OR "AUTO" (0 MEANS OFF)	(default 0)
CENTER X-COORDINATE OR "AUTO"	(default 0)
CENTER Y-COORDINATE OR "AUTO"	(default 0)

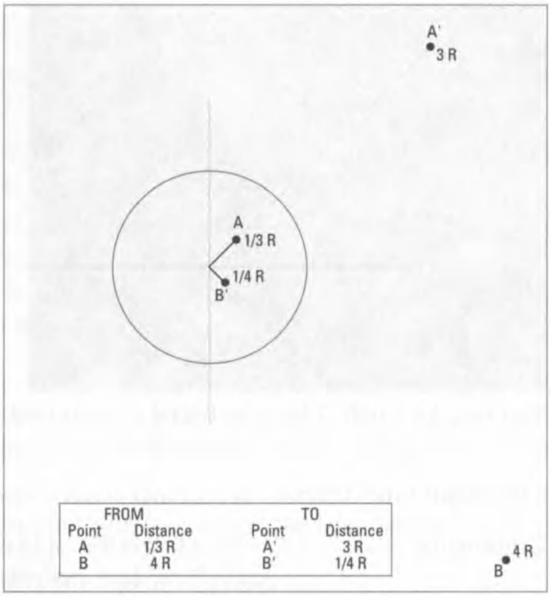
Command-Line Access: `invert=<radius>/<xcenter>/<ycenter>`

Comments: The invert image function has three parameters. The inversion radius must be set; the default 0 value means inversion is turned off. The center *x*- and *y*-coordinates default to 0 if not set.

Many years ago there was a brief craze for *anamorphic art* (images painted and viewed with the use of a cylindrical mirror, so that they looked weirdly distorted on the canvas but correct in the distorted reflection). In other words, you could see the paintings correctly if you looked at the image in the cylindrical mirror.

Fractint's inversion option performs a related transformation on most of the fractal types. You define the center point and radius of a circle on your fractal. Fractint maps each point inside the circle to a corresponding point outside, and vice versa. This is known to mathematicians as "everting" the plane. John Milnor made his name in the 1950s with a method for everting a seven-dimensional sphere, so Fractint still has a way to go in this particular area.

As an example, if a point A inside the circle is 1/3 of the way from the center to the radius, it is mapped to a point A' along the same radial line, but at a distance of (3 × radius) from the origin. An outside point B' at 4 times the radius is mapped



FROM		TO	
Point	Distance	Point	Distance
A	1/3 R	A'	3 R
B	4 R	B'	1/4 R

Figure 5-17 The inversion transformation

to a point B inside at ¼ the radius. Figure 5-17 shows the transformation that inversion accomplishes.

The EXTENDED OPTIONS menu prompts you for the radius and center coordinates of the inversion circle. Entering `Auto` sets the radius at ⅙ the smaller dimension of the image currently on the screen. The auto values for xcenter and ycenter use the coordinates currently mapped to the center of the screen.

The Newton fractal is a good one to try with the inversion option, because it has well-defined radial spokes that make it easy to visualize the before and after effects of inverting. Get the TYPE SELECTION menu by pressing (T), and then, selecting NEWTON. Then enter inversion parameters from the EXTENDED OPTIONS <Y> menu, and use a radius of 1 with the center coordinates set to 0. The center has "exploded" to the periphery. See Figure 5-18 for an example made with an order-3 Newton fractal showing the results before and after inversion. Inverting through a circle not centered on the origin produces bizarre effects that we're not even going to try to describe. Do this by entering nonzero values for xcenter and ycenter.

Set Color-Cycling Limits (Y)

Command Function: Limit the range of colors that are changed during color-cycling operations.

Menu Access: COLOR CYCLING FROM COLOR (0...254) item on the EXTENDED OPTIONS <Y> menu. COLOR CYCLING TO COLOR (1...255) item on the EXTENDED OPTIONS <Y> menu.

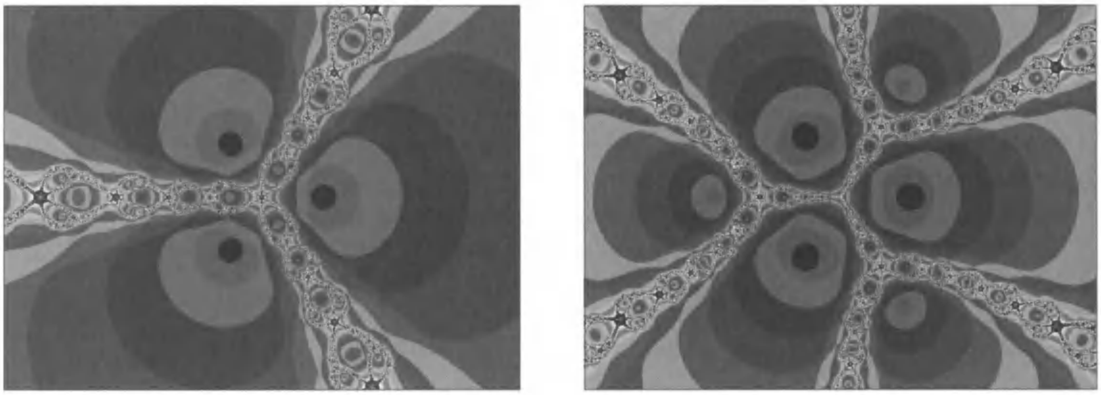


Figure 5-18 Order-3 Newton fractal with and without inversion

Command-Line Access: `cyclerange=<nnn>/<nnn>`

Comments: Color cycling is the rapid modification of the colors displayed on the screen by "cycling" through the color palette, a process described in detail later in the chapter. Normally, Fractint cycles palette entries 1 through 255 (color 0, used for the border background, is not normally cycled). You can modify this range to an alternate band of colors for some interesting effects.

Flipping the Image

Command Function: Flip an image around the screen's *x*-axis, *y*-axis, or origin.

Menu Access: none

Command-Line Access: none

Comments: Fractint can quickly flip (transpose) an image around the screen's *x*-axis, *y*-axis, or origin. With your image on the screen, a CONTROL-X (accomplished by holding down the CONTROL key and then pressing the X key) tells Fractint to flip the current image about the screen's *x*-axis. CONTROL-Y tells Fractint to flip the current image about the screen's *y*-axis, and CONTROL-Z tells Fractint to flip the current image about the screen's origin. There was no particular fractal-related reason to add these commands to Fractint, but someone asked for this capability so we added it!

Set Type-Specific Parameters Z

Command Function: Modify fractal type-specific parameters without selecting a new fractal type.

Menu Access: TYPE-SPECIFIC PARMS under the OPTIONS section of the MAIN MENU.

Command-Line Access: `params=<nnn>[/<nnn>[/<nnn>[/<nnn>]]]`

```
bailout=<nnn>
corners=xmin/xmax/ymin/ymax
center-mag=[Xctr/Yctr/Mag]
```

Comments: The Ⓩ option brings up type-specific input screens for your current fractal type—the same screens that the SELECT FRACTAL TYPE <T> command brings up after you have selected a fractal type. This option gives you the ability to modify either your current fractal parameters or your current image boundaries without affecting the other value. It also avoids wandering through the fractal types menu item when you're not interested in changing fractal types.

Set View Window Ⓥ

Command Function: Access view window settings so you can shrink the fractals image size for fractal calculation.

Menu Access: VIEW WINDOW OPTIONS under the OPTIONS section of the MAIN MENU.

Command-Line Access: `viewwindows=xx[/xx[/yes|no[/nn[/nn]]]]`

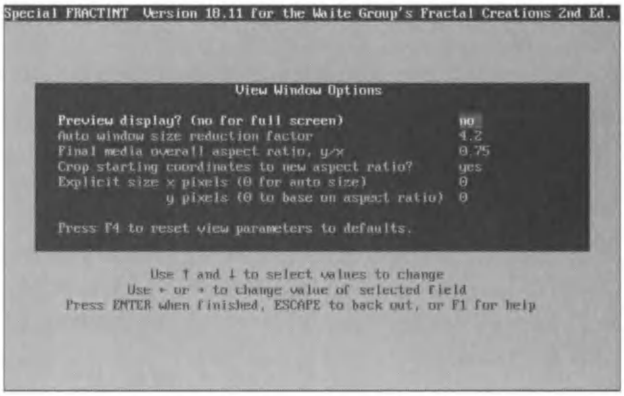
Sets the reduction factor, final media aspect ratio, crop starting coordinates (y/n), explicit x size, and explicit y size.

Comments: The view window is one of the fractal explorer's best friends. It allows smaller size images to be calculated and generated very rapidly because there are so few pixels. You set the size of the reduction from the menu (the default is a 4.2-times reduction of the normal full-screen image size). Thus, you can generate dozens of images in a fraction of the time that full-sized fractals would take. The calculation time is proportional to the number of pixels, so if you reduce the image dimensions by a factor of four, the number of pixels is reduced by a factor of sixteen, and the calculation time is reduced to one-sixteenth of the previous time! For experimental purposes, a small view window is just fine. When you find a promising effect, just turn off view windows, recalculate, and Fractint will make a full-screen image.

Figure 5-19 shows the input screen for the viewing area parameters accessed by the Ⓥ key. Note the message at the bottom indicating that the (F4) key will reset all parameters to the defaults. This reset feature is useful, because it is possible to get all tangled up in the settings!

Here is what each of these parameters does:

Preview display? (no for full screen) (default no) Answer "yes" to turn on the view window feature. Most of the time this is the only setting that will be needed—the other parameters have very reasonable default values.



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

                          View Window Options

    Preview display? (no for full screen)               no
    Auto window size reduction factor                   4.2
    Final media overall aspect ratio, y/x               0.75
    Crop starting coordinates to new aspect ratio?      yes
    Explicit size x pixels (0 for auto size)            0
                  y pixels (0 to base on aspect ratio)  0

    Press F4 to reset view parameters to defaults.

                 Use ↑ and ↓ to select values to change
               Use ← or → to change value of selected field
        Press ENTER when finished, ESCAPE to back out, or F1 for help
```

Figure 5-19 Viewing area parameters

Auto window size reduction factor (default 4.2) This factor is the amount by which the view window is scaled down. A larger value will make the view window smaller and it will calculate more quickly.

Final media overall aspect ratio, y/x (default 0.75) Aspect ratio is the overall height divided by overall width. The default value of .75 is almost universal, so you will rarely need to change this.

Crop starting coordinates to new aspect ratio? (default yes) If you answer "yes," and the corners parameters do not match the aspect ratio, the corners values will be changed to make the aspect ratio the value you specified. This generally will happen only if you either changed the view windows aspect ratio or altered the aspect ratio of the zoom box by stretching it in one of the dimensions.

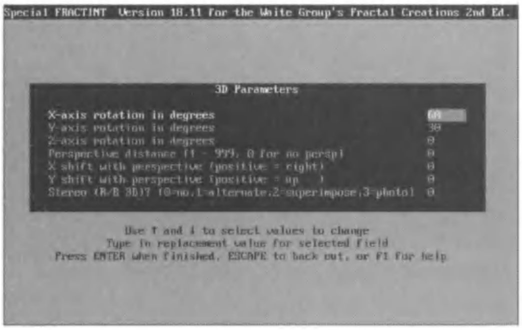
explicit size x pixels (0 for auto size) (default 0), y pixels (0 to base on aspect ratio) (default 0) You may specify the exact pixel dimensions of the view window. This overrides the autosize reduction factor and aspect ratio values.

Press ENTER to exit the View Area Parameters screen. To trigger the recalculation of the image to the new view window, reselect a video mode by pressing a function key.

Edit 3-D Transform Parameters I

Command Function: Edit the 3-D transform parameters.

Menu Access: Fractal 3D Parms from the Options section of the Main Menu (see Figure 5-20).



```
Special FRACTINT  Version 18.11 for the White Group's Fractal Creations 2nd Ed.

                              3D Parameters

   X-axis rotation in degrees                                      60
   Y-axis rotation in degrees                                      30
   Z-axis rotation in degrees                                      0
   Perspective distance [1 - 999, 0 for no persp]                  0
   X shift with perspective (positive = right)                     0
   Y shift with perspective (positive = up   )                     0
   Stereo (R/B 3D)? (0=no,1=alternate,2=superimpose,3=photo)       0

                  Use ↑ and ↓ to select values to change
                 Type in replacement value for selected field
         Press ENTER when finished, ESCAPE to back out, or F1 for help
```

Figure 5-20 3D Parameters input screen

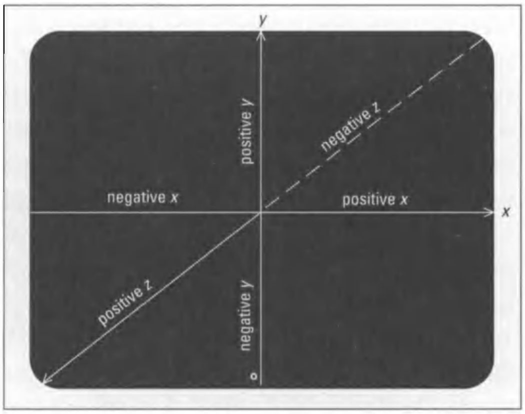


Figure 5-21 Fractint's 3-D coordinate system

Command-Line Access:

```
rotation=<xrot>[/<yrot>[/<zrot>]]
perspective=<nnn>
xyshift=<xshift>/<yshift>
stereo=0|1|2|3
```

Comments: All the 3-D capabilities in Fractint except Fractal type julibrot use the same variables. These 3-D parameters are also settable via the 3D Transform From File item on the Main Menu. The fractal types that use 3-D include ifs3D, lorenz3d, rossler3d, kamtorus3d, and henon.

Imagine that the x-axis runs horizontally across the middle of your computer screen, with zero in the middle. The y-axis runs vertically through the computer screen with zero in the middle. The z-axis is perpendicular to the plane of the screen with the positive end toward you. Figure 5-21 shows the coordinate system (in relation to a computer screen).

Here is what each of the 3-D parameters does. To see their effects in action, generate a lorenz3d fractal image by pressing the (T) key or selecting the Select Fractal Type entry from the Main Menu.

x-axis, y-axis, and z-axis rotation The first three parameters allow setting the rotations that cause the fractal objects to be viewed from different angles. Refer to Figure 5-22 as you follow this example. With a lorenz3d fractal on the screen, press the (I) key to bring up the 3D Parameters menu and change the x-, y-, and z-axis rotation values to 0, 0, 0. Press (ENTER) to accept these values, and press a video mode function key if you haven't already done so ((F2) is a good choice for EGA/VGA). You are now seeing the Lorenz orbit as it is with no 3-D rotations. The Lorenz orbit is the path of a wildly orbiting particle under the influence of two invisible attractors. It spirals around one, then the other, back and forth, forming two flat spirals in two different planes at an angle to each other.

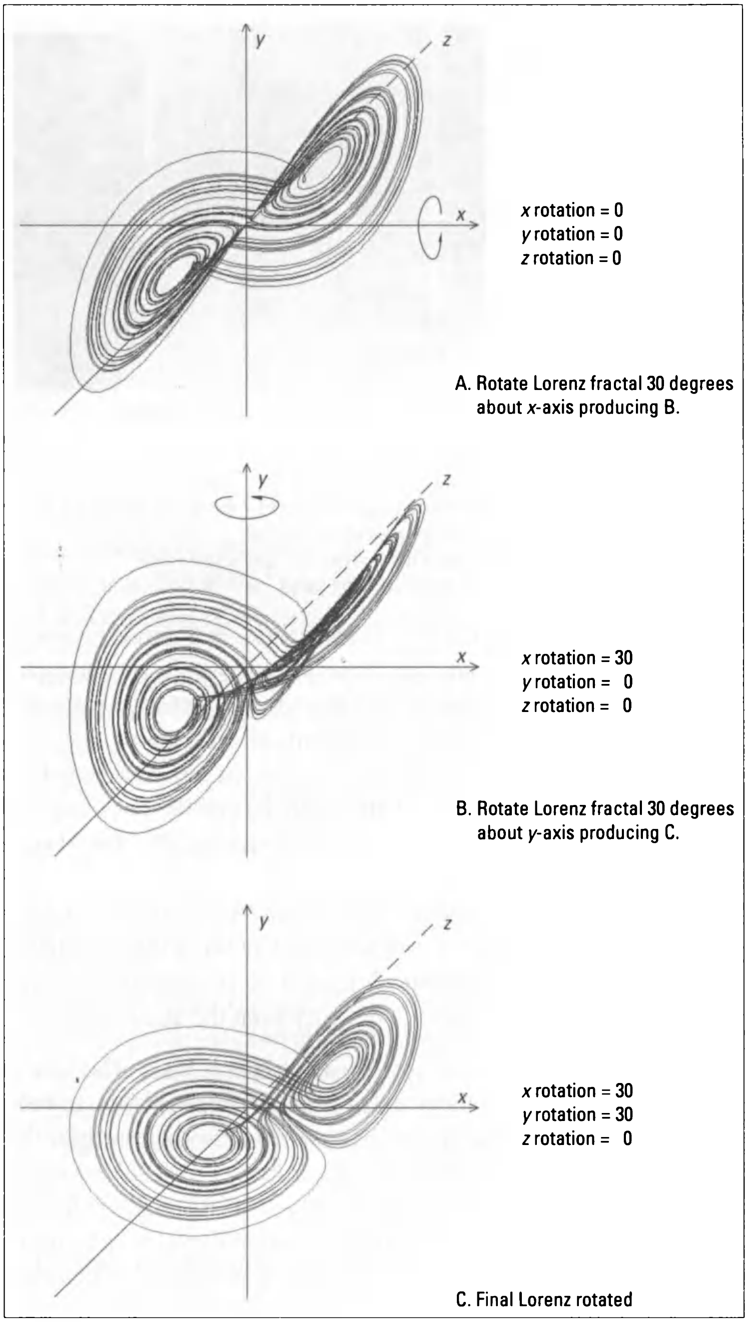


Figure 5-22 A Lorenz fractal rotated about the three axes

Repeat these steps starting with pressing (I), but change the *x*-axis rotation to 30. The image has rotated around the *x*-axis 30 degrees, with the top of the image coming toward you. One of the two spirals now looks very thin because you are viewing it end-on. Repeat these steps again, this time changing the *y*- axis rotation to 30, so both *x* and *y* rotations are now 30. The skinny spiral now looks fuller because the image has rotated around the *y*-axis and the right-hand side of the screen has moved away from you.

Repeat the steps one last time, setting all three rotation values to 30. The last rotation is the easiest to understand, because the *z*-axis is coming right out of the screen, and the rotation just moves the image clockwise around the screen. To get a little better feel, you might try repeating this whole experiment with red/blue glasses—just set the stereo option to 2. Figure 5-22 specifically shows the first three of these Lorenz images with superimposed axes, with arrows indicating the direction of rotation. The fourth image with all three rotations set to 30 is not shown.

Perspective distance [1–999, 0 for no persp] The perspective parameter causes the 3-D projection to use a viewpoint from different distances. The effect is to make closer parts of the fractal larger, and farther parts smaller, as seen by an imaginary observer. The value entered for perspective distance allows you to control how close this observer is to the fractal object. Imagine the 3-D object inside a box and just touching all the sides. A perspective value of 100 is an extreme perspective where your viewpoint is right on the near edge of the box, with parts of the object very close. This can be considered a closeup of the image. A value of 200 means that the near edge of the box is halfway between your eye and the far edge of the box. Figure 5-23 diagrams this situation. Try a value of about 120 with lorenz3d to see the effect.

X shift and Y shift with perspective The *x* and *y* shift move the position of the observer. If perspective is also turned on, the image is not just moved on the screen, but the point of view is also changed. Shifting to the left and then to the right changes the image in exactly the same way as what you see is changed when you close your right eye and look through your left, and then look at the same scene through your right eye.

Stereo (R/B 3D) Stereo viewing is a technique whereby two distinct views of a 3-D object are created, one as if seen by the right eye, the other a little offset and as if seen by the left eye. To reproduce stereo vision, a way is needed to get the left and right images to the correct eye. One method of doing that is to use red/blue funny glasses. The red filter blocks the blue image and lets the red image through, and

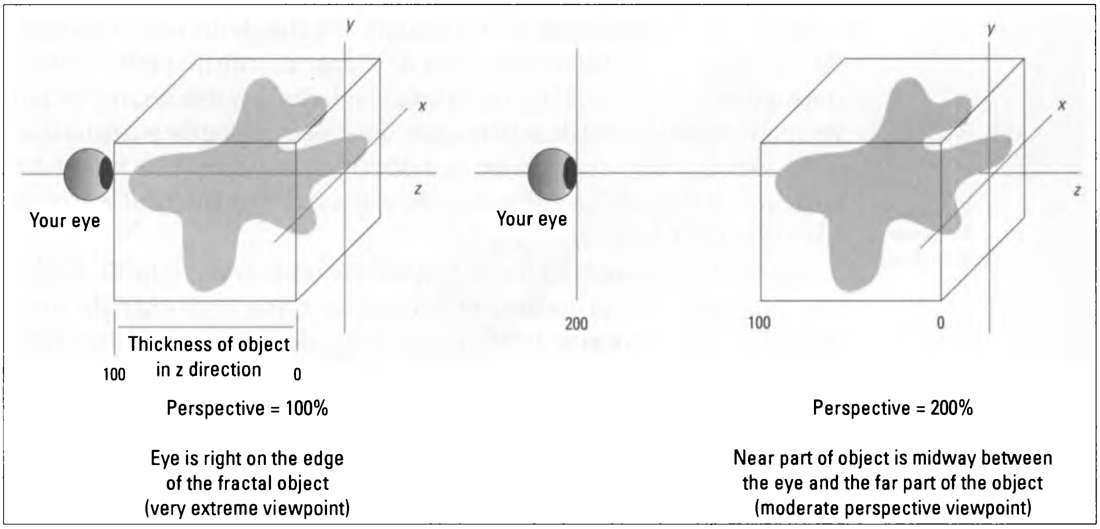
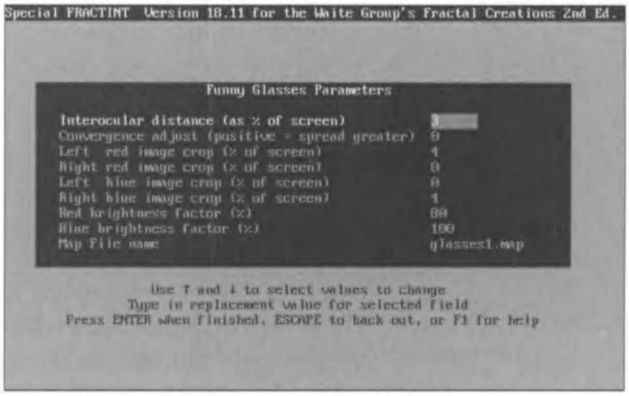

y
x
z
Your eye
Thickness of object in z direction
100
0
Perspective = 100%
Eye is right on the edge of the fractal object (very extreme viewpoint)
y
x
z
Your eye
200
100
0
Perspective = 200%
Near part of object is midway between the eye and the far part of the object (moderate perspective viewpoint)

Figure 5-23 Different perspective positions

the blue filter does the opposite. Fractint can put the left and right images on the screen at the same time, using red and blue colors. The two images overlap, so some method of combining the two colors is needed. Fractint provides two methods, each with its advantages. The alternate and superimpose options are the two different ways that the red and blue are combined. In the alternate approach, the screen is divided so that every other pixel in each row is designated to be either a red or blue pixel. The red and blue aren't really combined except in your mind. When your eye sees red and blue pixels close together, your mind "sees" the color magenta. The alternate approach does offer less resolution, because each image is formed from only half the screen pixels, but it allows more shades of red and blue—128 shades on a VGA in a 256-color mode.

The superimpose option combines overlapping red and blue pixels in a single magenta pixel, allowing higher effective resolution but fewer shades of red and blue. Even in a 256-color mode, there are only 16 visible shades of red and blue; all the colors are taken up with combinations of these shades. Use superimpose for lorenz3d. Put on your 3-D glasses and regenerate the lorenz3d example.

Stereo option 3 makes two separate images and pauses so that pictures can be taken of the screen. The two pictures can be mounted and viewed with a stereo slide viewer. This kind of stereo doesn't use red and blue; it provides full-color stereo. But of course, you need a camera to photograph the images or else you can save the left and right views as GIF files and have slides made by a slide service.



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

                         Funny Glasses Parameters

     Interocular distance (as % of screen)                3
     Convergence adjust (positive = spread greater)       0
     Left  red image crop (% of screen)                   4
     Right red image crop (% of screen)                   0
     Left  blue image crop (% of screen)                  0
     Right blue image crop (% of screen)                  4
     Red brightness factor (%)                            80
     Blue brightness factor (%)                           100
     Map File name                                        glasses1.map

                  Use ↑ and ↓ to select values to change
                Type in replacement value for selected field
         Press ENTER when finished, ESCAPE to back out, or F1 for help
```

Figure 5-24 Funny Glasses Parameters input screen

Edit Red/Blue Glasses Parameters (Y)

Command Function: Edit 3-D "funny glasses" parameters.

Menu Access: This menu is automatically selected when you have chosen either the ALTERNATE or SUPERIMPOSE STEREO 3D option during a 3D MODE selection. Figure 5-24 shows the FUNNY GLASSES PARAMETERS input screen.

Command-Line Access:

```
interocular=<distance>
converge=<distance>
crop=<red-left>/<red-right>/<blue-left>/<blue-right>
bright=<red>/<blue>
map=<mapfilename>
```

Comments: *Interocular distance* The interocular distance is the distance between the left and right viewpoints measured as a percent of the screen width. It should be set small enough so that your eyes can easily converge the two images, but large enough that there is an adequate stereo effect. The default of 3 usually works quite well if the perspective value is not extremely close.

Convergence adjust The convergence parameter adjusts the relative position of the two images in the horizontal dimension. The effect is to move the apparent image into or out from the screen. A larger value makes the image appear in front of the screen, while a smaller (possibly negative) value makes the 3-D object appear to be inside the monitor.

Left and right, red and blue image crop The red and blue images need to be clipped on the left and right differently to make a proper 3-D effect. The edge of the image should appear to be in the same place for both eyes. Setting this properly can be very tricky because it interacts strongly with the convergence parameter. The default is about right for images that appear near the screen surface. Cropping is only an issue when the screen edge clips the image. If a lorenz3d image is completely contained within the screen, for example, so the black background goes right up to the screen edge, then cropping is not needed.

Red and blue brightness factors The brightness parameters allow adjustment for differences in the red and blue screen color saturation and glasses filter properties. You can adjust these values so that both the red and blue images look equally bright. The correct settings are dependent on the color properties of your monitor, the quality of your funny glasses filters, and the color sensitivity of your eyes. Try to find values that minimize the ghost image caused by bleed-through of the left (red) image through the blue lens to the right eye, and the right (blue) image through the red lens to the left eye. If you can see a faint, green, "ghost" image when you place the blue lens over the red image, it means there is too much yellow in the red image—try turning down the monitor intensity until the ghost disappears.

Map file name The alternate and superimpose methods of displaying red and blue images on your screen each require special palette mappings. They are stored in the files GLASSES1.MAP and GLASSES2.MAP that come with Fractint. These files should be placed in a directory listed in your path. (If Fractint can't find them, it can generate the values on the fly.) The files GLASSES1.MAP and GLASSES2.MAP are designed to allow the greatest number of different shades of red and blue. If you are using superimpose, try substituting the file called GRID.MAP and see if it works any better (especially for wire frame images or other images that do not need shades of red and blue).

File Access

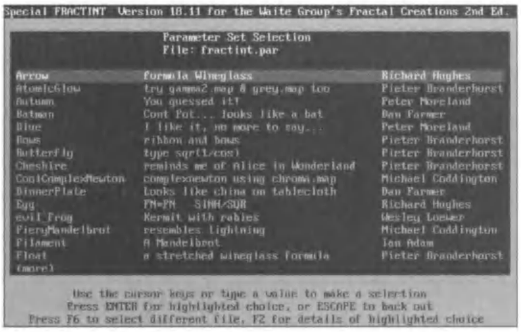
The FILE section of the MAIN MENU is where you go to save and load images, do 3-D transformations, create batch files, print, drop to DOS, quit Fractint, or restart. We will cover the menu items in order. Keep in mind that the MAIN MENU is modified to show more possibilities when an image has already been generated.

Select a Parameter File Entry (@)

Command Function: Select an entry from a parameter file and use it to generate a new image.

Menu Access: RUN SAVED COMMAND SET from the FILE section of the MAIN MENU.

Command-Line Access: `parmfile=filename`



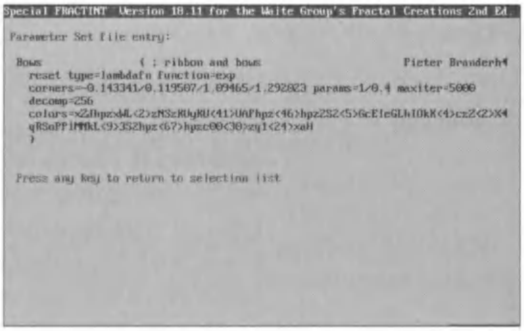
```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

                         Parameter Set Selection
                         File: fractint.par

Arrow              formula Wineglass                 Richard Hughes
AtomicGlow         try gamma2.map & grey.map too     Pieter Branderhorst
Autumn             You guessed it!                   Peter Moreland
Batman             Cont Pot... looks like a bat      Dan Farmer
Blue               I like it, no more to say...      Peter Moreland
Bows               ribbon and bows                   Pieter Branderhorst
Butterfly          type sqr(1/cos)                   Pieter Branderhorst
Cheshire           reminds me of Alice in Wonderland Pieter Branderhorst
CoolComplexNewton  complexnewton using chroma.map    Michael Coddington
DinnerPlate        Looks like china on tablecloth    Dan Farmer
Egg                FN*FN   SINH/SQR                  Richard Hughes
evil_frog          Kermit with rabies                Wesley Loewer
FieryMandelbrot    resembles lightning               Michael Coddington
Filament           A Mandelbrot                      Ian Adam
Float              a stretched wineglass formula     Pieter Branderhorst
(more)

         Use the cursor keys or type a value to make a selection
         Press ENTER for highlighted choice, or ESCAPE to back out
     Press F6 to select different file, F2 for details of highlighted choice
```

Figure 5-25 The (@) parameter entry selection screen



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

Parameter Set File entry:

 Bows                { ; ribbon and bows                       Pieter Branderh
   reset type=lambdafn function=exp
   corners=-0.143341/0.119507/1.09465/1.292823 params=1/0.4 maxiter=5000
   decomp=256
   colors=xZZhpzxWL<2>zNSzKUyKU<41>UAFhpz<46>hpz2S2<5>GcEIeGLhIOkK<4>czZ<2>X
   qRSoPPlMMkL<9>3S2hpz<67>hpzc00<30>zq1<24>xaH
   }

 Press any key to return to selection list
```

Figure 5-26 Details of a parameter entry

Changes Fractint's default parameter file name from FRACTINT.PAR to the file name specified.

`@filename/entryname`

Causes Fractint to start up using the parameter file filename and the parameter entry entryname.

Comments: Fractint has the ability to load and save sets of command-line entries describing images in the screen rather than the images themselves. These entries are stored in parameter files. Multiple entries can be saved in any one file. Each parameter file entry contains all the information Fractint needs to re-create a single fractal image. Parameter files are a very powerful feature, as they give you the ability to completely describe fractal images with just a few lines of text.

When you press the (@) key, Fractint reads the currently opened parameter file (by default, FRACTINT.PAR) and presents you with a selection screen displaying the name and first comment area of each entry (see Figure 5-25). If you want to select a different parameter entry file, you can press the (F6) key to bring up a file-selection menu of available parameter files. Otherwise, use the arrow keys to select a parameter entry (you can press (F2) at any time to see the entire text of the entry that is currently highlighted—see Figure 5-26) and press (ENTER) to select it. If you have not yet displayed any fractal images, you will be prompted to select a video mode; otherwise, Fractint will use the last graphics mode you selected to display the selected parameter entry.

Save an Image (S)

Command Function: Save a fractal image as a .GIF file.

Menu Access: SAVE IMAGE TO FILE under the FILE section of the MAIN MENU.

Command-Line Access: `batch=yes`

Causes an automatic save following the completion of a calculation.

Comments: Pressing (S) causes the current image to be saved as a CompuServe GIF file. (See Appendix B, *Fractint and GIF Files,* for details on the .GIF format.) Two small vertical bars representing the progress of the save operation will grow down the left- and right-hand sides of the screen. When they reach the bottom of the screen, the name of the saved file is reported at the top of your screen. The default name, which is settable in the BASIC OPTIONS menu, is FRACT001.GIF. If you save more than one time during a Fractint session, the last character of the name will be a number that is incremented, resulting in FRACT002.GIF, and so forth. The saved file appears in the directory that was current when Fractint was started. You can't set the path name from Fractint.

Normally, Fractint does not overwrite existing files. If you would like to reuse existing file names to conserve disk space, set overwrite to "yes" in the BASIC OPTIONS menu, or add the line `overwrite=yes` to the command line or in your SSTOOLS.INI file. Even with `overwrite=yes` the file names will have an incrementing number, so that saved images made during the same session will not overwrite each other.

Fractint's GIFs Store Partial States Fractint remembers the state of a partial calculation when saving in the GIF file format. Fractint is the world's fastest fractal program, but calculations with high maximum iterations in floating-point mode, extreme resolutions, or running on a slow PC can still take a long time. For example, the feather image on the cover of the original *Fractal Creations* book took a week to generate on a 25-MHz 80386 machine. (The image was created using a 2048 x 2048 disk video mode with a very high maximum iteration value.) Many times throughout that week the image was saved when the computer had to be used for other purposes—and when it was restored, the calculation picked right up where it left off!

Fractint stores GIFs using the GIF89a format. If you need to make a GIF file that is viewable with software that does not support the new GIF89a standard, start the program with the option

```
gif87a=yes
```

and then all saved images will be in the older GIF87a format. However, they will not contain any information about the Fractint parameter that created them. You can also convert GIF89a files to GIF87a files by reading them in and then saving them. The command line

```
fractint newformat.gif gif87a=yes savename=oldformat.gif batch=yes
```

reads in the file newformat.gif and saves it as the GIF87a format file oldformat.gif.

Load Images from a File (R)

Command Function: Load a fractal image from a GIF file for display.

Menu Access: LOAD IMAGE FROM FILE <R> under the FILE section of the MAIN MENU.

Command-Line Access: `[filename=]<filename>`

To load an image as you start Fractint so it's viewable, you can type in either `fractint myfile` or `fractint filename=myfile`. The GIF extension is assumed.

Comments: GIF files created by Fractint contain not just fractal images but also information about how the fractal was generated. Therefore, loading the file not only allows you to view the image, but resets Fractint to regenerate that image. Fractint is capable of being used as a GIF decoder to view files not created by Fractint, such as pictures. Fractint has no way of knowing how those GIFs were generated, but has to think of them as *some* sort of fractal, so it sets their fractal type to "plasma."

The (R) command takes you to a sophisticated file selection screen with several useful features. These features are as follows:

- **Point to file:** Just use the arrow keys to move the highlight to the file you want to select. Then press (ENTER). If you select an item that is a directory, the current directory changes to the selected directory. Selecting ".." takes you up one directory.
- **Speed key selection:** Begin typing the name of the file you want to select. The highlight will jump to the first file name that matches what you have typed so far.
- **Path search:** You can type the name of a file not in the current directory. If it is in one of the directories listed in your path statement, Fractint will find it.
- **Wild cards:** If you enter a wild card template, such as t*.gif, then the list will change to show just the files matching the template, in this case, all files starting with "t" with the extension ".gif." The wild cards work the same as DOS wild cards using "*" and "?".
- **Changing drive or directory:** You may enter a new drive by typing in the drive letter and a ":", or a new directory, or both at the same time. If you are entering a new directory, end with a "\" so that Fractint knows you want a directory rather than a file.

Once you have selected a file, the video mode list is presented and you will be prompted to select a video mode to display the selected fractal. If Fractint can locate a video mode on the list that matches the image being loaded (part of the information stored with a GIF is its resolution), that mode will be highlighted.

Disk Video and Higher-Resolution Modes It is possible to view an image at a lower resolution than the actual resolution of the image. For example, the image on the cover of the original edition of *Fractal Creations* was created at Fractint's highest resolution of 2048 x 2048 using the disk video mode. The same image can be viewed in Fractint at a low resolution mode such as 640 x 480 or 320 x 200. Fractint just throws out the extra pixels. This feature is extremely useful, because it allows you to create and view disk video images or other images at resolutions greater than those supported by your graphics equipment.

3-D Transformation from File (3)

Command Function: Perform a 3-D transformation of a GIF file.

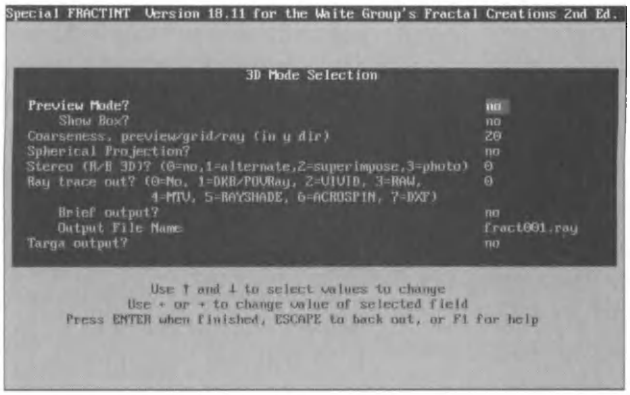
Menu Access: 3D TRANSFORM FROM FILE under the FILE section of the MAIN MENU.

Command-Line Access: `3D=yes`

Comments: Most of the fractals created by Fractint are inherently two-dimensional, meaning they are flat in the *x-y* plane. A 3-D mode allows you to transform any fractal into a three-dimensional image with depth and an *x-y-z*-axis. The 3-D function treats a fractal's colors as the third dimension and performs various 3-D and rendering transformations on the image, so it appears on the screen projected realistically. Another feature of Fractint is that the 3-D transformations are not limited to Fractint-generated files, but can also be performed on GIF files created by other software. Indeed some scientists use Fractint's 3-D capabilities to enhance electron microscope pictures!

Using 3-D involves several successive and somewhat complex-looking screens, but it is really quite easy to use. The (3) command leads you to the first of these screens for inputting all the parameters that affect 3-D. Do not be dismayed by the number of possibilities: usually the default values are something reasonable, and you press (ENTER) to move to the next screen. Follow the defaults at first, and then try changing the parameters a few at a time.

The (3) command begins with a file selection screen that works the same as the file selection for the (R) command. Select a GIF file, choose a video mode (generally the same as that of the GIF file), and then select a 3-D mode (see the following section).



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

                              3D Mode Selection

  Preview Mode?                                                      no
      Show Box?                                                      no
  Coarseness, preview/grid/ray (in y dir)                            20
  Spherical Projection?                                              no
  Stereo (R/B 3D)? (0=no,1=alternate,2=superimpose,3=photo)          0
  Ray trace out? (0=No, 1=DKB/POVRay, 2=VIVID, 3=RAW,                0
                 4=MTV, 5=RAYSHADE, 6=ACROSPIN, 7=DXF)
      Brief output?                                                  no
      Output File Name                                               fract001.ray
  Targa output?                                                      no

                  Use ↑ and ↓ to select values to change
                Use ← or → to change value of selected field
         Press ENTER when finished, ESCAPE to back out, or F1 for help
```

Figure 5-27 3D Mode Selection screen

Select a 3-D Mode ③

Command Function: After 3-D transformation has been selected, it is used to choose a specific 3-D mode.

Menu Access: 3D TRANSFORM FROM FILE under the FILE section of the MAIN MENU.

Command-Line Access:

```
3D=yes
preview=yes|no
coarse=<nnn>
showbox=yes|no
sphere=yes|no
stereo=0|1|2|3
ray=0|1|2|3|4|5|6|7
brief=yes|no
```

Comments: After the file name prompt and video mode check, Fractint presents the 3D Mode Selection screen shown in Figure 5-27. Each selection will have defaults entered. If you want to change any of the defaults, use the cursor keys to move through the menu. When you're satisfied, press (ENTER) to accept your choices and move to the next 3-D screen. (ESC) allows you to back up to the previous screen.

Here are the options and what they do.

Preview Mode? (yes or no) Preview mode provides a rapid look at your transformed 3-D image by skipping a lot of rows and filling in the image. It is good for quickly discovering the best parameters. Once the 3-D parameters look good, you can turn off the preview mode and generate the full image.

Show Box? If you have selected preview mode, you have another option to consider. This is the option to show a rectangular "image box" around the image boundaries in scaled and rotated coordinates *x*, *y*, and *z*. The bottom of this box is the original *x-y* plane of your fractal, and the height is the dimension where the colors in your fractal will be interpreted as elevations. The box appears only in rectangular transformations and shows how the final image will be oriented; it doesn't draw the actual transformation. If you select light source in the next screen, it will also show you the light source vector so you can tell where the light is coming from in relation to your image.

Coarseness, preview/grid/ray (in y dir) The coarseness parameter sets how many divisions the image will be divided into in the *y* direction. It is needed if you select preview mode as described, or grid fill in the SELECT FILL TYPE screen. The default is 20 divisions; a larger number makes a finer (and slower) grid.

Spherical Projection? The spherical projection parameter allows you to select a sphere projection of your fractal. This maps your image onto a plane as previously described if your answer is "no," or onto a sphere if you answer "yes." Therefore you can take your favorite fractal, wrap it around a sphere, and turn it into a planet, an asteroid, a moon, or whatever. Fractint allows you to use any GIF image and make a planet out of it—even a digitized photograph of your loved one! Planets can be smooth or rough, large or small, and they can be illuminated with the light from an imaginary sun.

Stereo Fractint allows you to create 3-D images for use with red/blue glasses like the ones found in 3-D comic books. Option 0 turns off the stereo effect. Options 1 and 2 require the special red/blue glasses. They are meant to be viewed right on the screen or on a color print of the screen. The image can be made to hover entirely or partially in front of the screen.

Stereo option 1 gives 64 shades of red and blue, but with half the spatial resolution you have selected. It works by writing the red and blue images on adjacent pixels, which is why it removes half the picture's resolution. In general, we recommend you use this with resolutions above 640 x 350 only. Use this mode for continuous potential landscapes where you need all those shades.

Stereo option 2 gives you full spatial resolution but with only 16 shades of gray. If the red and blue pixels overlap, the colors are mixed, and the pixel is colored magenta. This option is good for wire-frame images (we call them surface grids), lorenz3d, and ifs3d. It works fine in 16-color modes.

Stereo option 3 is for creating full-color stereo pair images for viewing with more specialized equipment. The left image is presented on the screen first. You may photograph it or save it as a GIF for later processing into a slide. Then the second image is presented, and you may do the same with it as you did with the first image. You can then take the two images and convert them to a stereo image pair.

Ray trace out?/Brief output?/Output File Name: Fractint can create files of its 3-D transformations that are compatible with many ray tracing programs. Currently, five are supported directly: POV-Ray, Vivid, MTV, Rayshade, and DXF. In addition, a Raw output is supported which can be relatively easily transformed into a format many other products can use. The DKB/POVray format is obsolete and not recommended. POV-Ray users should use RAW format and convert with the RAW2POV program, or directly read in GIF files with POV-Ray's height field feature. Acrospin is included as a ray tracing output option, even though it's not really a ray tracer, because the same Fractint options apply. All ray tracing files consist of triangles that follow the surface created by Fractint during the 3-D transform. Triangles that lie below the waterline are not created in order to avoid causing unnecessary work for the poor ray tracers which are already overworked. A simple plane can be substituted by the user at the waterline if needed.

The size (and, therefore, the number) of triangles created is determined by the PREVIEW FACTOR setting. While generating the ray tracing file, you will view the image from above and watch it partitioned into triangles.

The color of each triangle is the average of the color of its vertices in the original image, unless BRIEF OUTPUT is selected. If BRIEF OUTPUT is selected, a default color is assigned at the beginning of the file and is used for all triangles.

The RAY-TRACING OUTPUT FILE NAME is used to specify the name of the file to be written. The default name is FRACT001.RAY. Note that the ray tracing files generated by Fractint are not ready to be traced by themselves. For one thing, no light source is included. They are actually meant to be included within other ray tracing files. Because the intent is to produce an object that may be included in a larger ray tracing scene, it is expected that all rotations, shifts, and final scaling will be done by the ray tracer. Thus, in creating the images, no facilities for rotations or shifting is provided. Scaling is provided to achieve the correct aspect ratio.

The files created using the RAY TRACE option can be huge. Setting PREVIEW FACTOR to 40 will result in over 2000 triangles. Each triangle can take from 50 to 200 bytes each to describe, so your ray tracing files can rapidly approach or exceed 1Mb. Make sure you have enough disk space before you start.

Ray Tracing Files: Technical Information Each ray tracing file starts with a comment identifying the version of Fractint that generated it and ends with a comment giving the number of triangles in the file.

Fractint's coordinate system has the origin of the *x-y* plane at the upper left-hand corner of the screen, with positive *x* to the right and positive *y* down. The ray tracing files have the origin of the *x-y* plane moved to the center of the screen with positive *x* to the right and positive *y* up. Increasing values of the color index are out of the screen and in the $+z$ direction. The color index 0 will be found in the x-y plane at $z=-1$.

When x, y, and z scale are set to 100, the surface created by the triangles will fall within a box of +/– 1.0 in all three directions. Changing scale will change the size and/or aspect ratio of the enclosed object.

We will only describe the structure of the RAW format here. If you want to understand any of the ray tracing file formats besides Raw, please see your favorite ray tracer documentation.

The RAW format simply consists of a series of clockwise triangles. If BRIEF OUTPUT is checked, each line is a vertex with coordinates x, y, and z. Each triangle is separated by a couple of CRs from the next. If BRIEF OUTPUT is not checked, the first line in each triangle description is the red, green, blue value of the triangle.

Selecting BRIEF OUTPUT produces shorter files with the color of each triangle removed—all triangles will be the same color. These files are otherwise identical to normal files but will run faster than the non-BRIEF OUTPUT files. Also, with BRIEF OUTPUT selected, you may be able to get files to run with more triangles than otherwise.

For DKB, when BRIEF OUTPUT is selected and the WATER LEVEL value (specified on the next screen) is nonzero, you may get empty COMPOSITE/END_COMPOSITE pairs, (i.e., containing no triangle information). These are harmless, but may be edited out of the file if desired.

Targa output? If you want any of the 3-D transforms you select to be saved as Targa-24 files or overlaid onto one, rather than as GIF files, select this option.

Select 3-D Fill Type ③

Command Function: Select a 3-D fill type, determining if the image is drawn with all pixels, as a wire frame image, etc.

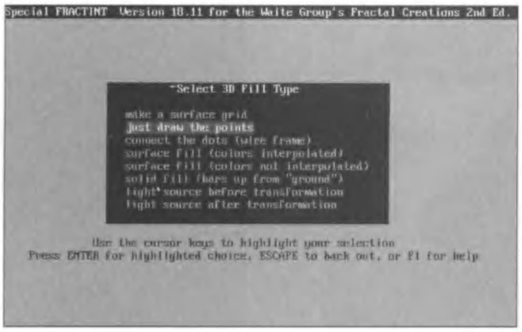
Menu Access: Automatically selected after you have entered your basic 3-D options during a 3-D file restore.

Figure 5-28 shows the SELECT 3D FILL TYPE screen as it appears in the nonsphere case. If you are doing a 3-D projection onto a sphere, the only difference is that there will be only one light source option.

Command-Line Access: `filltype=<nnn>`

where <nnn> is 0 through 7.

Comments: In the course of any 3-D projection, portions of the original image must be stretched to fit the new surface. Points of an image that formerly were right next



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

                    *Select 3D Fill Type

                     make a surface grid
                     just draw the points
                     connect the dots (wire frame)
                     surface fill (colors interpolated)
                     surface fill (colors not interpolated)
                     solid fill (bars up from "ground")
                     light source before transformation
                     light source after transformation

              Use the cursor keys to highlight your selection
     Press ENTER for highlighted choice, ESCAPE to back out, or F1 for help
```

Figure 5-28 Select 3D Fill Type screen

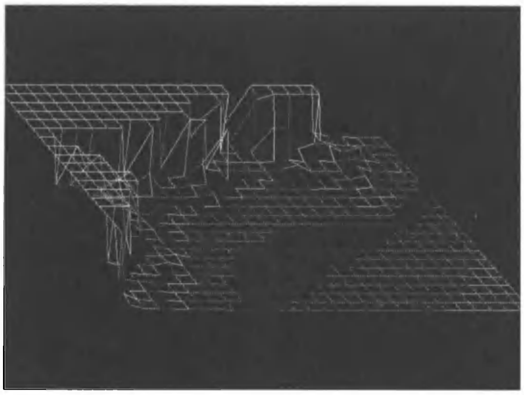


Figure 5-29 The 3-D surface grid fill option

to each other may have a space between them now. The SELECT FILL TYPE options generally determine what to do with the space between the mapped dots.

Make a surface grid If you select the MAKE A SURFACE GRID option, Fractint will make an unfilled wire-frame grid of the fractal surface that has as many divisions in the original *y* direction as were set in the COARSE option in the first screen. This wire-frame view of your image is generated very quickly and can reveal a quick approximation of what the final 3-D fractal will look like. Figure 5-29 shows a portion of the Mandelbrot set in planar 3-D drawn using this option.

Just draw the points The second option, JUST DRAW THE POINTS, means Fractint maps points in the 2-D image to corresponding points in the 3-D image. Generally, this will leave empty space between many of the points, and this space will appear black. Figure 5-30 shows the same portion of the Mandelbrot set drawn using this option.

Connect the dots (wire frame) This fill method simply connects the points in the hope that the connecting lines will fill in all the missing pixels. This option is rarely used because it has been supplanted by the superior surface-fill methods that were developed later.

Surface fill (colors interpolated), *Surface fill (colors not interpolated)* The surface-fill options fill in the areas between the 3-D dots with small triangles formed from the transformed points. If the corners of the triangles are different colors, the COLORS INTERPOLATED fill colors the interior of the triangle with colors that smoothly blend between the corner colors. The COLORS NOT INTERPOLATED fill simply colors the whole triangle the color of one of the corners. Interpolating the colors makes the

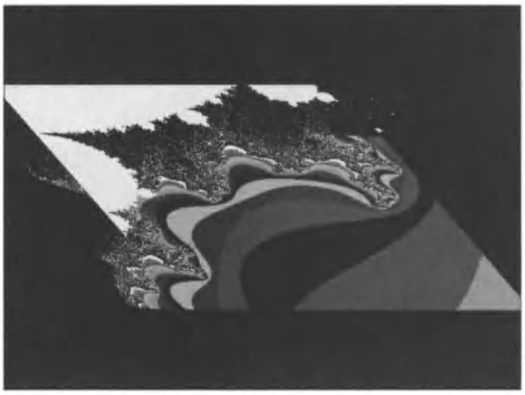


Figure 5-30 The 3-D just draw the points option

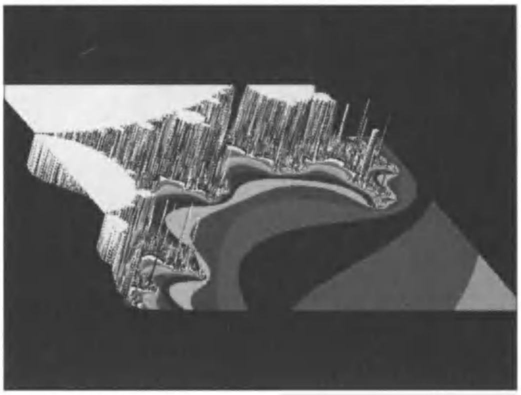


Figure 5-31 The 3-D surface fill option

little triangles blend better but only works if the color palette is continuous, meaning that colors with near color numbers are a similar color. If the results look strange, try the Colors Not Interpolated fill. Figure 5-31 shows the same portion of the Mandelbrot set drawn using this option.

Solid fill (bars up from "ground") The solid fill method works by using a kind of bar graph approach. A line is drawn from each point to its projection in the *x*-*y* plane. Figure 5-32 shows the same portion of the Mandelbrot set drawn using this option.

Light source before transformation, Light source after transformation The two light source fill options allow you to position an imaginary sun over your fractal landscape. Fractint colors each pixel of the landscape according to the angle the surface makes with an imaginary light source. This creates the appearance of shadows and can be used to create realistic mountains. You will be asked to enter the three coordinates of the vector pointing toward the light in one of the following screens.

The option called Light Source Before Transformation calculates the illumination before doing the coordinate transformations, and it is slightly faster. If you generate a sequence of images where one rotation is progressively changed, the effect is as if the image and the light source are fixed in relation to each other and you orbit around the image.

Light Source After Transformation applies the transformations first, then calculates the illumination. If you generate a sequence of images with progressive rotation as above, the effect is as if you and the light source are fixed and the object

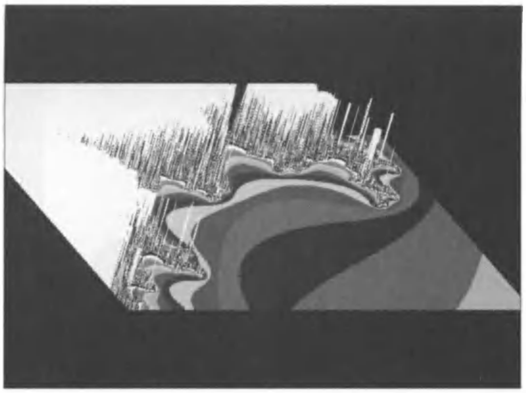


Figure 5-32 The 3-D solid filll option

is rotating. Figure 5-33 shows the relationship between the fractal object, the viewer, and the light source for these two options.

If you select either light source fill (before or after), you will be prompted for a color map, which is a file assigning colors to the color numbers. You can try ALTERN.MAP, which is a grayscale palette that represents the light source shading as shades of gray. However, any map that has continuous shades of color works well with the light source options, although they may not look as realistic as with the

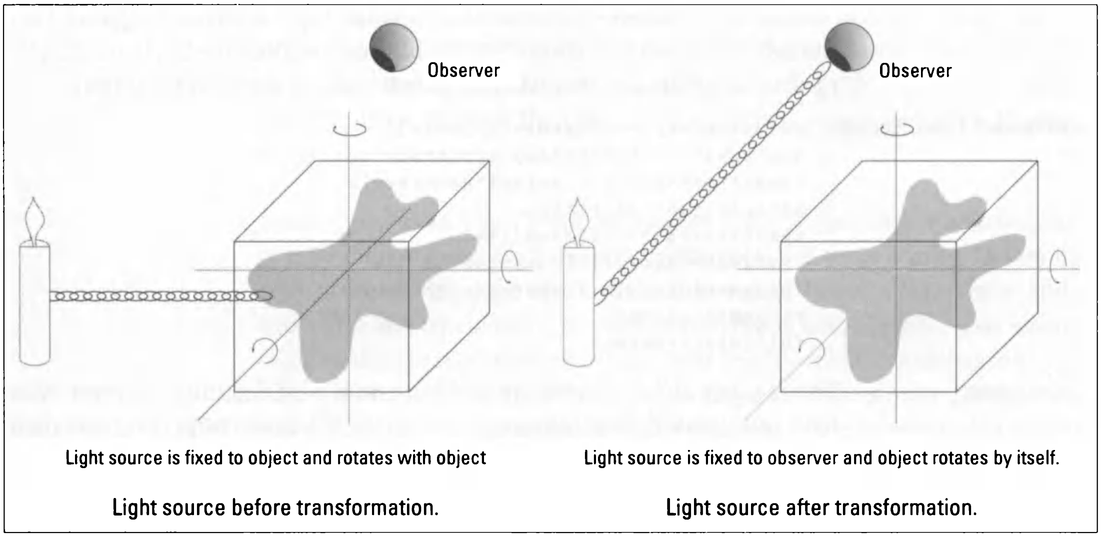
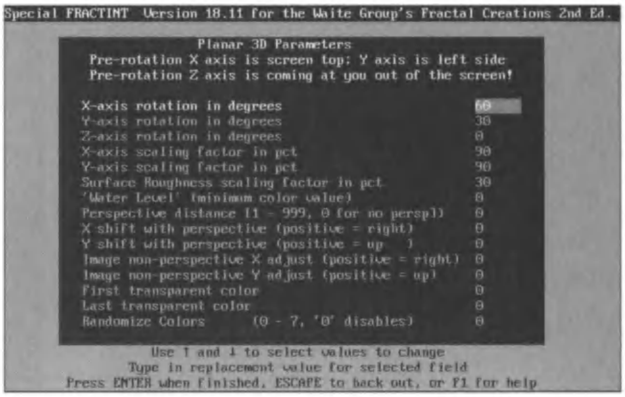

Observer
Observer
Light source is fixed to object and rotates with object
Light source is fixed to observer and object rotates by itself.
Light source before transformation.
Light source after transformation.

Figure 5-33 Two light source options



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.
                        Planar 3D Parameters
          Pre-rotation X axis is screen top; Y axis is left side
          Pre-rotation Z axis is coming at you out of the screen!

         X-axis rotation in degrees                              60
         Y-axis rotation in degrees                              30
         Z-axis rotation in degrees                              0
         X-axis scaling factor in pct                            90
         Y-axis scaling factor in pct                            90
         Surface Roughness scaling factor in pct                 30
         'Water Level' (minimum color value)                     0
         Perspective distance [1 - 999, 0 for no persp])         0
         X shift with perspective (positive = right)             0
         Y shift with perspective (positive = up   )             0
         Image non-perspective X adjust (positive = right)       0
         Image non-perspective Y adjust (positive = up)          0
         First transparent color                                 0
         Last transparent color                                  0
         Randomize Colors      (0 - 7, '0' disables)             0

                 Use ↑ and ↓ to select values to change
               Type in replacement value for selected field
         Press ENTER when finished, ESCAPE to back out, or F1 for help
```

Figure 5-34 Planar 3D Parameters screen

gray palette in ALTERN.MAP. Try color cycling with the (+) command and using the higher function keys (such as (F8) or (F9)) to get some interesting effects.

Select Planar 3-D Parameters (3)

Command Function: Choose various planar 3-D parameters (such as axis rotation, water level, etc.)

Menu Access: Automatically selected if you have not selected spherical projection as part of the basic 3-D options list.

Figure 5-34 shows the PLANAR 3D PARAMETERS input screen as it appears when the light fill option is in effect. If a light fill is not in effect, the last two items, the randomize colors and mono/color options, will not show on the screen.

Command-Line Access:

```
rotation=<xrot>[/<yrot>[/<zrot>]]
scalexyz=<scalex>[/<scaley>[/<scalez>]]
roughness=<scalez> waterline=<level>
perspective=<distance>
xyshift=<xshift>[/<yshift>]
xyadjust=<xadjust>[/<yadjust>]
transparent=<startcolor>/<stopcolor>
randomize=<nnn>
fullcolor=yes|no
```

Comments: The number of 3-D parameters in this menu is a bit daunting; however, most have reasonable default values, so you can usually press (ENTER) to accept them all. Therefore, you do not need to understand all of them to get 3-D working. You'll usually change only a few of these parameters, unless you want to explore.

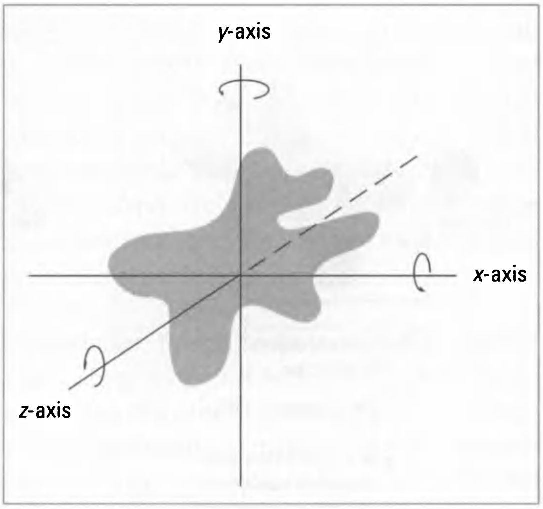


Figure 5-35 Rotating fractal objects

X-axis rotation in degrees, Y-axis rotation in degrees, Z-axis rotation in degrees The first entries are rotation values around the *x*-, *y*-, and *z*-axes. Think of your starting image as a flat map. The *x* value tilts the bottom of your monitor toward you by *x* degrees. The *y* value pulls the left side of the monitor toward you. The *z* value spins it counterclockwise. The final result of combining rotations depends on the order in which they are done. Fractint always rotates first along the *x*-axis, then along the *y*-axis, and finally along the *z*-axis. All rotations actually occur through the center of the original image. Figure 5-35 shows these three rotations.

X-axis scalingfactor in pct, Y-axis scalingfactor in pct, Surface roughness scalingfactor in pct Following the three rotation parameters are three scaling factors that control the resulting size of each axis of the image. Initially, leave the *x*- and *y*-axes alone and try changing the surface roughness factor (really *z*-axis scaling). High values of roughness assure that your fractal will be translated into steep Alpine mountains and improbably deep valleys; low values make gentle, rolling terrain. Negative roughness is legal. For example, if you're doing a Mandelbrot image and want the solid Mandelbrot lake to be below the ground, instead of eerily floating above, try a roughness of about –30%.

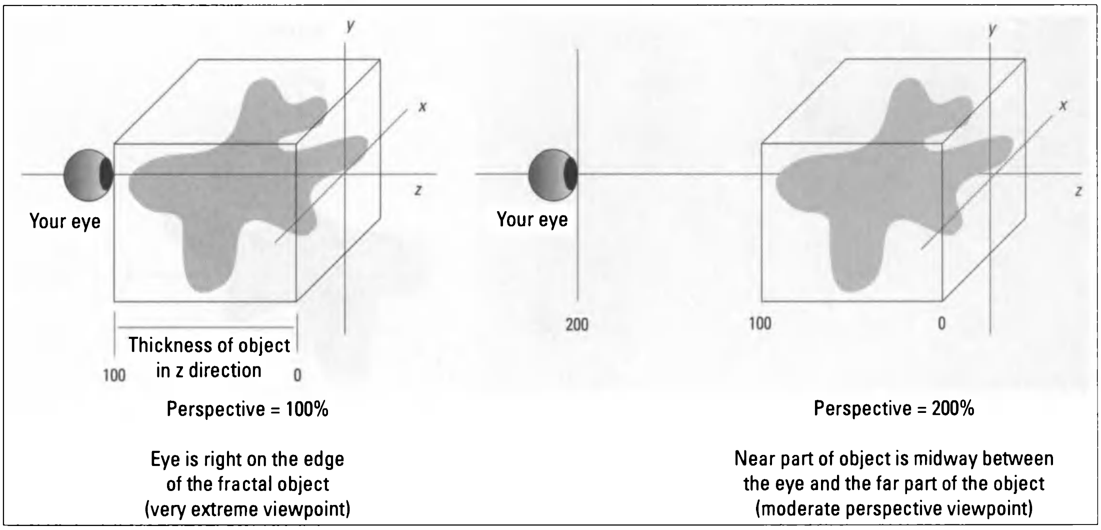

y
x
z
Your eye
Thickness of object in z direction
100
0
Perspective = 100%
Eye is right on the edge of the fractal object (very extreme viewpoint)
y
x
z
Your eye
200
100
0
Perspective = 200%
Near part of object is midway between the eye and the far part of the object (moderate perspective viewpoint)

Figure 5-36 The perspective parameter

Water Level (minimum color value) When a file is loaded into Fractint using the 3-D option, the colors are interpreted as elevations according to the number of the color. The water level option creates a minimum elevation in the resulting image. The result is exactly like flooding a valley. The higher the water level value, the more of the scene will be underwater. This works well with plasma landscapes.

Perspective distance [1–999, 0 for no persp] Perspective distance can be thought of as the distance from your eye to the image. A zero value (the default) means no perspective calculations, which makes the image appear flat, as though photographed through a telephoto lens. If you do set perspective to a nonzero value, nearer features to the observer will be larger than farther away features. To understand the effect of the perspective number, picture a box with the original *x-y* plane of your flat fractal on the bottom and your 3-D fractal inside. A perspective value of 100% places your eye right at the edge of the box and yields fairly severe distortion, like a close view through a wide-angle lens. A value of 200% puts your eye as far from the front of the box as the back is behind. A value of 300% puts your eye twice as far from the front of the box as the back is, and so on. Try about 150% for reasonable results. Much larger values put you far away for even less distortion, while values smaller than 100% put you "inside" the box. Try larger values first, and work your way in. Figure 5-36 shows how the perspective parameter relates to the distance from the viewer to the object.

X shift with perspective (positive = right), *Y shift with perspective (positive = up)*, *Image nonperspective x adjust (positive = right)*, *Image nonperspective y adjust (positive = up)* There are two types of *x* and *y* shifts that let you move the final image around if you'd like to recenter it. In the first set, *x* and *y* shift with perspective and move the image and change the viewing perspective. In the second set, *x* and *y* adjust without perspective, and simply move the image without changing the perspective viewpoint. They are used just for positioning the final image on the screen.

First transparent color, *Last transparent color* You may define a range of transparent colors. This option is most useful when using the Overlay command (see the following text) to place one image on top of another, so parts of the bottom image show through. Enter the color range (minimum and maximum value) for which you do not want to overwrite whatever may already be on the screen. The color ranges refer to the color numbers in the original image. The default is no transparency (overwrite everything).

Randomize Colors (0–7, '0' disables) The randomize option will smooth the transition between colors and reduce the banding that occurs with some maps. Select the value of randomize to between 0 (for no effect) and 7 (to randomize your colors almost beyond use). A setting of 3 is a good starting point.

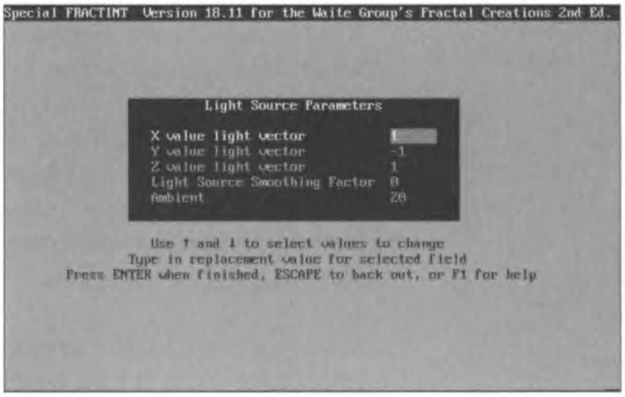
Here is an example of generating a Targa file based on the MTMAND.POT continuous-potential file. Type the following into a file called mtmand3d:

```
filename=mtmand.pot
3d=yes
filltype=6
randomize=3
fullcolor=yes
ambient=15
rotation=60/30/0
scalexyz=100/100
roughness=120
waterline=0
perspective=220
xyshift=10/-32
lightsource=1/-1/1
map=topo
```

After creating the file MTMAND3D, type

```
fractint @MTMAND3D savename=mtmand3D batch=yes
```

This will create a true-color Targa file called MTMAND3D.TGA.



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

                    Light Source Parameters

              X value light vector              1
              Y value light vector              -1
              Z value light vector              1
              Light Source Smoothing Factor     0
              Ambient                           20

              Use ↑ and ↓ to select values to change
            Type in replacement value for selected field
      Press ENTER when finished, ESCAPE to back out, or F1 for help
```

Figure 5-37 Light Source Parameters input screen

You can then use Piclab to convert this file to a GIF file with `dither=yes`. If you don't have Piclab, you can still do a monochrome image of MtMand in regular GIF format. In the previous example, remove the line `fullcolor=yes` and change the `map=yes` line to `map=altern`. The result will look very much like the cover of *The Beauty of Fractals*, by H.O. Peitgen and P.H. Richter, Springer-Verlag 1986.

Set Light Source Parameters ③

Command Function: Set light source parameters.

Menu Access: Selected automatically if you have selected a light source option in the 3-D restore FILL TYPE menu. Figure 5-37 shows the LIGHT SOURCE PARAMETERS input screen.

Command-Line Access:
```
lightsource=<x>[/<y>[/<z>]]
smoothing=<nnn>
ambient=<nnn>
haze=<nnn>
lightname=<filename>
```

Comments: The purpose of this screen is to control the details of an internal, simulated light that is shining on your fractal. You will need patience when using the light source option, because figuring out light directions can be confusing.

X *value light vector*, Y *value light vector*, Z *value light vector* First, if you have selected a light source fill, you must choose the direction of the light coming from the light source. This will be scaled in the *x*, *y*, and *z* directions the same as the image. For example, the values 1,1,3 position the light to come from the lower-right front

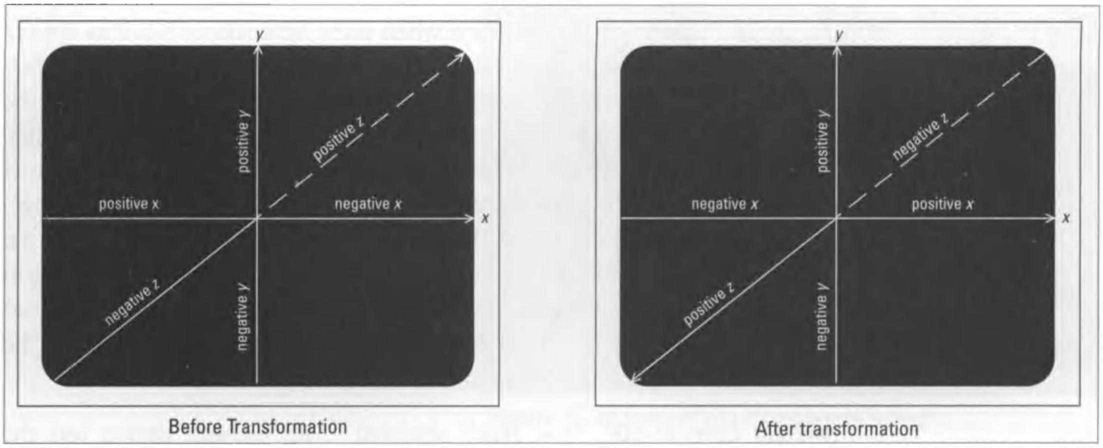


Figure 5-38 Two light coordinate systems

of the screen in relation to the untransformed image. It is important to remember that these coordinates are scaled the same as your image. Therefore, 1,1,1 positions the light to come from a direction of equal distances to the right, below, and in front of each pixel on the original image. However, if the *x,y,z* scale is set to 90,90,30, the result will be from equal distances to the right and below each pixel but from only 1/3 the distance in front of the screen (that is, it will be low in the sky, say, afternoon or morning).

Figure 5-38 shows the coordinate system used for defining the light vectors in the two light source modes. This coordinate system is not the same for the before transformation and after transformation light source options we explained earlier. For the light source before transformation option, the positive *x*-axis is on the left, the positive *y*-axis is up, and the positive *z*-axis is behind the screen. A good light vector to try would be *x*=1, *y*=1, and *z*=-3. With this light vector and rotations of 0,0,0, the light would appear to come from the upper right. For the light source after transformation option, the positive *x*-axis is on the right, the positive *y*-axis is up, and the positive *z*-axis is in front of the screen. To get the same effect as the above vector, the signs of the *x*- and *z*-coordinates of the light vector have to be reversed, yielding *x*=–1, *y*=1, and *z*=3. Confusion can be avoided by using one of the two light source options until you are familiar with the effects.

Light Source Smoothing Factor Next you are asked for a smoothing factor. Unless you used continuous potential (see the earlier description) when generating the starting 3-D image, the illumination when using light source fills may appear "sparkly," like a sandy beach in bright sun. This is because with only 256 colors in the original image, the *z*-coordinate has only 256 possible values, and the transformed image surface is broken into tiny facets. With continuous potential, there are 64,000 possible *z*-coordinate values, so a very smooth surface is possible. The smoothing factor averages colors in each line of the original image, smearing them together. A smoothing factor of 2 or 3 will allow you to see the large-scale shapes better. If you did use continuous potential and are loading in a "*.pot" file, you should turn off smoothing. If your fractal is not a plasma cloud and has features with sharply defined boundaries (e.g., Mandelbrot lake), the smoothing may cause the colors to run.

Ambient Light (0–100, '0' = 'Black' shadows) The ambient option sets the minimum light value a surface has if it has no direct lighting at all. All light values are scaled from this value to white. This effectively adjusts the depth of the shadows and sets the overall contrast of the image.

Haze Factor (0–100, '0' disables), Full Color Light File Name (if not LIGHT001.TGA) The last two input screen items appear only if you selected the full color option to make a Targa file. The haze factor makes distant objects more hazy. Close-up objects are little affected; distant objects will be obscured by haze. The value 0 disables the function, and 100 gives the maximum effect, with the farthest objects lost in the mist. Currently, this does not really use distance from the viewer; instead, Fractint cheats and uses the *y* value of the original image. So the effect really works only if the *y* rotation (set earlier) is between +/– 30.

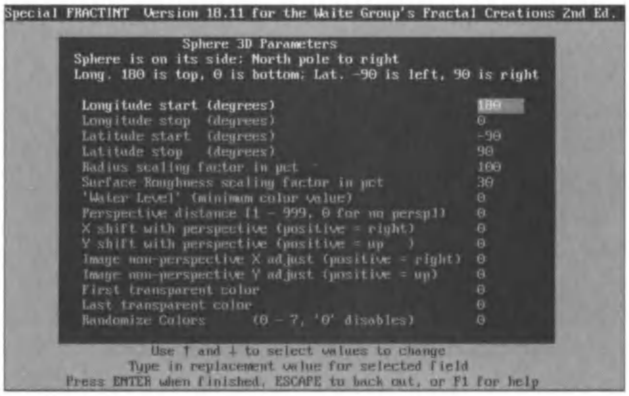
The last item allows you to choose the name for your light file. If you have a RAM disk handy, you might want to create the file on it for speed, so include its full path name in this option.

Select Spherical 3-D Parameters ③

Command Function: Select the various sphere 3-D parameters for wrapping an image around a globe

Menu Access: Selected automatically if you have selected a spherical transformation during the initial 3D Restore menu.

Figure 5-39 shows the SPHERE 3D PARAMETERS input screen as it appears when the light fill option is in effect. If a light fill is not in effect, the last two items, the randomize colors and mono/color options, will not show on the screen.



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.
                    Sphere 3D Parameters
         Sphere is on its side: North pole to right
         Long. 180 is top, 0 is bottom; Lat. -90 is left, 90 is right

          Longitude start (degrees)                          180
          Longitude stop  (degrees)                          0
          Latitude start  (degrees)                          -90
          Latitude stop   (degrees)                          90
          Radius scaling factor in pct                       100
          Surface Roughness scaling factor in pct            30
          'Water Level' (minimum color value)                0
          Perspective distance [1 - 999, 0 for no persp]]    0
          X shift with perspective (positive = right)        0
          Y shift with perspective (positive = up   )        0
          Image non-perspective X adjust (positive = right)  0
          Image non-perspective Y adjust (positive = up)     0
          First transparent color                            0
          Last transparent color                             0
          Randomize Colors      (0 - 7, '0' disables)        0
                 Use ↑ and ↓ to select values to change
               Type in replacement value for selected field
          Press ENTER when finished, ESCAPE to back out, or F1 for help
```

Figure 5-39 Sphere 3D Parameters screen

Command-Line Access:

```
longitude=<startdegree>/[<stopdegree>]
latitude=<startdegree>/[<stopdegree>]
radius=<scaleradius>
roughness=<scalez>
waterline=<level>
perspective=<distance>
xyshift=<xshift>[/<yshift>]
xyadjust=<xadjust>[/<yadjust>]
transparent=<startcolor>/<stopcolor>
randomize=<nnn>
fullcolor=yes|no
```

Comments: The sphere 3-D parameters function controls the wrapping of a fractal image around the surface of a sphere. In fact, you can project any GIF file image, whether from Fractint or not, onto the surface of a sphere.

Longitude start (degrees), *Longitude stop (degrees)*, *Latitude start (degrees)*, *Latitude stop (degrees)* Picture a globe lying on its side, north pole to the right. You will be mapping the *x*- and *y*-values of the starting image to latitude and longitude on the globe, so that what was a horizontal row of pixels becomes a line of longitude, while what was a vertical column of pixels becomes a line of latitude. The default values exactly cover the hemisphere facing you, from longitude 180 degrees (top) to 0 degrees (bottom) and latitude –90 (left) to latitude 90 (right). By changing these values you can map the image to a piece of the hemisphere or wrap it completely around the globe. Figure 5-40 shows how this works.

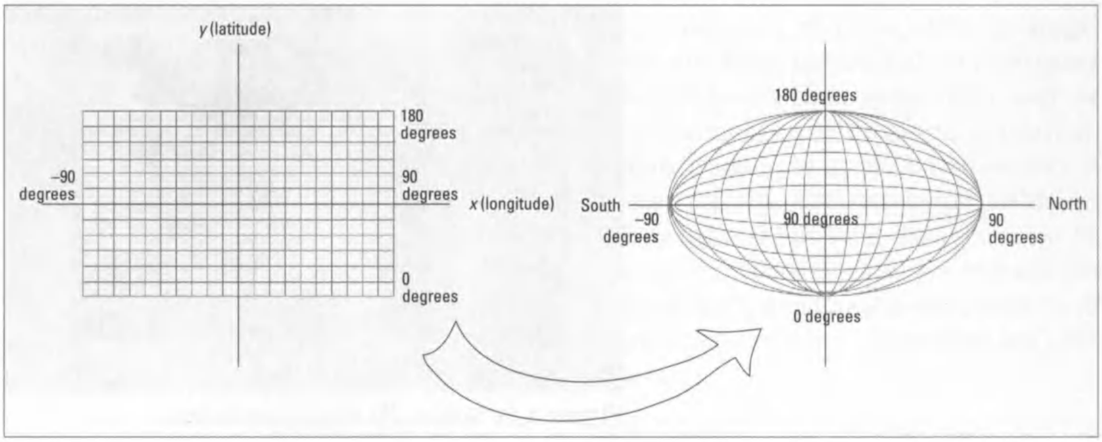


Figure 5-40 Mapping a fractal to a sphere

Radius scaling factor in pct The radius factor controls the overall size of the globe and is the sphere analog to the *x* and *y* scale factors. Use this parameter to enlarge or shrink your globe as you wish.

Surface Roughness scaling factor in pct The roughness factor in the sphere context controls the bumpiness of the surface of the sphere. A value of zero makes the sphere perfectly smooth.

The remaining screen items have the same meaning for a sphere transformation as they do for a plane transformation—see the PLANAR 3D PARAMETERS screen explanation.

When the wrap-around "construction" process begins at the edge of the sphere (the default) or behind it, it is plotting points that will be hidden by subsequent points as the process sweeps around the sphere toward you. Fractint's hidden-point algorithms "know" this, and the first few dozen lines may be invisible unless a high mountain happens to poke over the horizon. If you start a spherical projection and the screen stays black, wait awhile (a longer while for higher resolution or fill type 6) to see if points start to appear.

Select 3-D Overlay Parameters (#)

Command Function: Perform a 3-D transformation of a GIF file overlaid on the current image.

Menu Access: 3D OVERLAY FROM FILE <#> under the FILE section of the MAIN MENU.

Command-Line Access: none

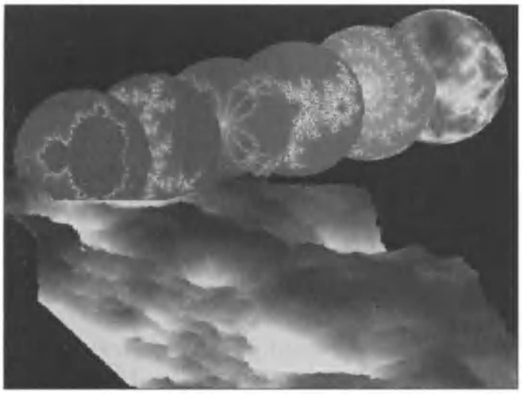
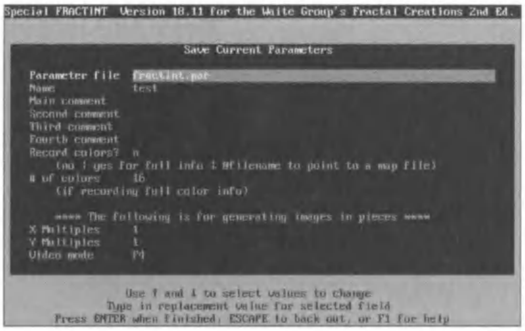


Figure 5-41 Moons over landscape



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

                        Save Current Parameters

  Parameter file   fractint.par
  Name             test
  Main comment
  Second comment
  Third comment
  Fourth comment
  Record colors?   n
      (no | yes for full info | @filename to point to a map file)
  # of colors      16
      (if recording full color info)

      **** The following is for generating images in pieces ****
  X Multiples      1
  Y Multiples      1
  Video mode       F4

                   Use ↑ and ↓ to select values to change
                  Type in replacement value for selected field
             Press ENTER when finished, ESCAPE to back out, or F1 for help
```

Figure 5-42 Save Current Parameters entry screen

Comments: This function is identical to the normal 3-D transformation accessed with the (3) command, with one important difference: the screen is not cleared prior to the drawing of the 3-D image. The new image is pasted on top of the old image. For example, if the first image is a plasma landscape, and you use the (#) command to make a sphere, the sphere image will be added to the plasma landscape picture. Figure 5-41 shows an example of the kind of images that are possible with this command.

Save Parameter File Entry (B)

Command Function: This command causes Fractint to save the parameters used to generate your current fractal image as an entry in a parameter file.

Menu Access: SAVE CURRENT PARAMETERS under the FILE section of the MAIN MENU.

Command-Line Access: none

Comments: Fractint has the ability to load and save sets of command-line entries describing images in the screen rather than the images themselves. These entries are stored in parameter files. Multiple entries can be saved in any one file: each parameter file entry contains all the information Fractint needs to re-create a single fractal image. Parameter files are a very powerful feature of Fractint, as they give you the ability to completely describe a fractal image with only a few lines of text.

When you press the (B) key, Fractint displays the SAVE CURRENT PARAMETERS entry screen shown in Figure 5-42.

Here are the options and what they do.

Parameter File This is the name of the parameter file to which you want to add your image description as a parameter file entry. The default name, FRACTINT.PAR, is not really a good choice, as that file is a standard one supplied with all versions of Fractint. You might want to enter another name (such as MYFILE) instead. If you don't enter your own filetype, .PAR is assumed. If the named parameter file doesn't exist, Fractint will silently create it for you.

Name This is the name of the entry you will be adding to the parameter file. If a parameter entry already exists with this name, you will be asked if you really want to overwrite it. If you indicate that you don't, you will be returned to the entry screen so that you can change it.

Main Comment, Second Comment, Third Comment, Fourth Comment You can enter up to four lines of comments as part of your parameter file entry. The first comment line is particularly important, as it is displayed along with the entry name when you use the (@) command to select and display entries in parameter files. The other three comment lines are currently stored in the parameter file as part of the entry, but are otherwise unused.

Record Colors? (No | Yes | @Filename), # of Colors These entries govern whether color information should be included in the entry. Usually the default value displayed by Fractint is what you want, as Fractint is pretty clever about remembering whether you've used a palette file or done any color cycling lately. The RECORD COLOR options are

No	Don't record colors. This is the default if the image is using Fractint's default colors.
Yes	Record the colors in the parameter entry in detail. This is the default when you've changed the display colors by color cycling.
@filename	Load the colors from the named color map file. This is the default if you have loaded your colors from a color map file and haven't modified them.

The # of Colors field only matters if RECORD COLORS is set to Yes. It specifies the number of color values to save as part of the parameter entry. Recording fewer colors takes less space, but loses nonvisible colors that might become visible later via color cycling. Usually, the default value displayed by Fractint is what you want.

X *Multiples*, Y *Multiples*, *Video Mode* These entries are only used if you want to use Fractint's DIVIDE-AND-CONQUER feature. The divide-and-conquer feature is only enabled if either the X MULTIPLES or Y MULTIPLES field is greater than 1.

The divide-and-conquer feature creates multiple PAR entries that break up a fractal image into pieces so that you can generate the image pieces one by one. There are two reasons for doing this. The first is in case the fractal is very slow, and you want to generate parts of the image at the same time on several computers. The second is that you might want to make an image greater than 2048 x 2048. The parameters for this feature are

X MULTIPLES	The number of images to create in the *x* direction (1 to 36).
Y MULTIPLES	The number of images to create in the *y* direction (1 to 36).
VIDEO MODE	The Fractint video mode for each piece (e.g., "F3").

The last item defaults to the currently selected video mode. If either *X* MULTIPLES or *Y* MULTIPLES are greater than 1, then multiple numbered PAR entries for the pieces are added to the PAR file, and a MAKEMIG.BAT file is created that uses Fractint to first build all of the component pieces and then stitch them together into a single multi-image GIF file, FRACTMIG.GIF. The current limitations of the DIVIDE-AND-CONQUER algorithm are 36 or fewer *X* and *Y* multiples (so you are limited to "only" 36x36=1296 component images), and a final resolution limit in both the *X* and *Y* directions of 65,535 (a limitation of the GIF format).

Multi-image GIF files are a perfectly legal GIF format—but that doesn't necessarily mean that all GIF decoders can handle them. Because of this, MAKEMIG.BAT includes one final line—commented out, but there in case you need it—that calls another program (SIMPLGIF, included on your companion disk and installed on your hard disk as part of the installation process described in Chapter 1, *Installation,*) that reads your multi-image GIF file and uses it to generate a simple, single-image GIF called SIMPLGIF.GIF.

One point should be emphasized here. Be aware that this process can be used to generate *huge* images, and huge images come with corresponding appetites for disk space. GIF files are highly compressed, but a 64K x 64K image represents over four billion pixels, and even the best compression techniques only go so far. Also, during the step that stitches the individual images together, both the individual images *and* the composite result must be available on your hard disk, so the storage requirement is doubled.

Finally, if the software with which you intend to use your monster images mandates that you process the images with SIMPLGIF, note that this program needs to create a temporary uncompressed version of your final image as an intermediate step. In this case, you really do need 4GB of free disk space for a 64K x 64K image—in addition to the space taken up by FRACTMIG.GIF and SIMPLGIF.GIF. As a practical matter, most users of the "divide and conquer" feature should limit their images to a "modest" 4K x 4K or 8K x 8K resolution (which are actually pretty high resolutions!).

Print Image (P)

Command Function: Print a fractal on the screen to a printer.

Menu Access: PRINT IMAGE under the FILE section of the MAIN MENU.

Command-Line Access: none

Comments: The (P) command prints the current fractal on the screen. There are a number of command-line options that govern printing. Printer-related command-line options can be specified interactively using the (G) (GIVE COMMAND STRING) command, but are usually stored in your SSTOOLS.INI file. These printer-related command-line options are described in detail in the final section of this chapter, "Command-Line Only Commands."

Fractint's current list of supported printers includes Epson/IBM-compatible dot-matrix printers (the default), HP-compatible laser printers, HP Paintjet-compatible printers, Postscript printers (both monochrome and color), and HP-GL-based plotters. Several of Fractint's Disk/RAM video options use resolutions specifically designed to work well with specific printers and Fractint's (P) command.

Shell to DOS (D)

Command Function: Use this command to leave Fractint in memory so you can exit to DOS and return to Fractint quickly

Menu Access: SHELL TO DOS under the FILE section of the MAIN MENU.

Command-Line Access: none

Comments: This option switches to DOS, leaving Fractint stored in memory, ready to resume when you type `exit` at a DOS prompt. Be careful not to do anything that changes the video mode, as the bulk of your graphics image is still squirreled away in a currently unused area of your video adapter's memory. If you do change video modes before returning to Fractint, your graphics image may not be intact when you return. You return to Fractint from a shell-to-DOS by typing `exit`.

Give Command String (G)

Command Function: Enter a command-line string interactively.

Menu Access: GIVE COMMAND STRING under the FILE section of the MAIN MENU.

Command-Line Access: none

Comments: This command gives you the option to enter a command-line (parameter) entry at any time while using Fractint. Suppose, for instance, that you have just created the perfect fractal image, are about to save it as a GIF file, and suddenly realize that the software you are creating this GIF file for only accepts the older GIF87a format. You can force Fractint to save its GIF files using the GIF87a format for the rest of your session by pressing the (G) key and entering the command-line string `gif87a=yes` at the command-string prompt.

Quit Fractint (ESC)

Command Function: Quit Fractint and return to DOS.

Menu Access: QUIT FRACTINT under the FILE section of the MAIN MENU.

Command-Line Access: none

Comments: You will be given the prompt EXIT FROM FRACTINT? (Y/N). Pressing either (Y) or (ENTER) exits, (N) returns to the Main Menu, other keys do nothing. Because (ESC) is also used to back through the menu system, this safety feature prevents exiting by inadvertently typing an extra (ESC).

Restart Fractint (INSERT)

Command Function: Restart Fractint.

Menu Access: RESTART FRACTINT under the FILE section of the MAIN MENU.

Command-Line Access: `reset`

Comments: Pressing (INSERT) on the numeric keypad from the MAIN MENU has the same effect as quitting and restarting Fractint. Use it when you have altered many Fractint settings and want to return to the startup defaults. Because Fractint is a rather large program and takes a few seconds to load, this command avoids an irritating delay that would be experienced by exiting and restarting.

The reset command-line parameter is used in parameter files to perform the same effect as the (INSERT) command.

Set Colors (C)

The Colors section of the Main Menu appears only if your graphics adapter supports palette manipulations. If you have CGA- or Hercules-compatible monochrome graphics, this menu will not appear. If you do have an EGA, VGA, or higher advanced graphics adapter, this menu is your avenue to some spectacular color effects, most notably color cycling.

Command Function: Enter the color-cycling mode with or without starting cycling.

Menu Access: Color Cycling Mode and Rotate Palette <+>, <-> under the Colors section of the Main Menu.

Command-Line Access: none

Comments: The purpose of the (C) command is to enter the color-cycling mode without actually starting color cycling. Because the (C) command itself does not start color cycling, a visual indicator of the mode is provided; the border area of the screen turns to white. You might want to do this to load a map file or do some of the other functions available under color cycling.

Use the (+) or (−) key to enter the color-cycling mode and start color cycling at the same time. Color cycling is one of the really exciting features in Fractint. An animation effect is achieved by rapidly changing how colors are mapped to the color numbers of the original image. Plasma images are particularly fascinating. The plasma colors flow into each other in an endless unfolding. Here is an explanation of what is happening: Your fractal image actually assigns numbers, not colors, to pixels. If your image is a 256-color image, the numbers range from 0 to 255. At any given time, your graphics adapter assigns each of these numbers to colors selected from a much larger set—262,144 different colors for a VGA. Color cycling plays musical chairs with these colors. Think of the 256 numbers as the chairs, and the colors as the kids circling with these chairs. If the kids go one chair at a time, they are doing exactly what color cycling does. All three of these commands cause the current mode to change from the display mode to the color-cycling mode.

Note that the palette colors available on an EGA adapter (16 colors at a time out of a palette of 64) are limited compared to those of VGA, super VGA, and MCGA (16 or 256 colors at a time out of a palette of 262,144). Color cycling in general looks *a lot* better in the VGA/MCGA modes. Also, because of the EGA palette restrictions, some color cycling commands are not available with EGA adapters.

A completely different set of keystroke commands applies while in color cycling mode. These are listed here.

ESC	Exits the color-cycling mode. For example, use this when you are ready to save a fractal with S.
C	Toggles cycling on and off.
F1	Brings up a Help screen with commands specific to color command mode.
+ or →	Cycles the palette forward. Each color moves to the higher color index. Colors at the last index move to the first index.
− or ←	Cycles the palette backward. Each color moves to the next-lower color index. Colors at the first color index move to the last. Alternate between − and + to see the colors throb!
<, >	Cycles the palette forward or backward a single step and then pauses the color cycling. Useful when you have *just* missed the perfect color combination and want to attempt to recover it.
↑ or ↓	Increases/decreases the cycling speed. The original purpose of this command was to eliminate flicker experienced on some displays when color cycling. But it is also useful just to slow down the color cycling on very fast machines to a more pleasing speed.
F2 – F10	Switches from simple color palette rotation to color selection using randomly generated color bands of short (F2) to long (F10) duration. Pressing any function key except the help key (F1) during color cycling causes Fractint to add new random colors. Pressing F2 causes a new random color to be created for each cycle. Higher function keys cause new colors to be generated at intervals and, in between, the colors smoothly merge from the last random color to the next. The higher the function key number, the more intermediate colors are calculated. To see a few colors cycled through many beautiful shades, use the higher keys.

1 – 9	Causes the screen to be updated every *n* color cycles (the default is 1), so smaller numbers give slower cycling, higher give faster cycling. Handy for slower computers.
ENTER	Randomly selects a function key (F2 through F10) and then updates *all* the screen colors prior to displaying them for instant, random colors. Press this over and over again to see your fractal with totally different colors.
SPACEBAR	Pauses cycling and turns the screen border white as a visual indication of the continuation of the color cycling mode.
SHIFT-F1 – F10	Pauses cycling and resets the palette to a preset two-color "straight" assignment, such as a spread from black to white. (Not for EGA.) These keys allow you to access some built-in palettes and see how they look with your fractal.
CONTROL-F1 – F10	Pauses cycling and uses a two-color cyclical assignment, for example, red_yellow_red (not for EGA). These are some more built-in palettes to try.
ALT-F1 – F10	Pauses cycling and uses a 3-color cyclical assignment, for example, green_white_blue (not for EGA). Still more built-in palettes!
D or A	Pauses cycling and loads an external color map from the files DEFAULT.MAP (D) (IBM default palette) or ALTERN.MAP (A) (continuous grayscale palette), supplied with the program.
L	Pauses cycling and prompts for the file name of an external color map. Several others are supplied with the program. (The .MAP extension is assumed.) Map files allow you to specify which color each color number represents. They are ordinary text files. Each line in the text file determines one color; the first line is color 0, the second line is color 1, and so forth. Each line of the map file has three numbers that determine the red, green, and blue content of that color number.

Read	Blue	Green
0	0	0
252	252	252
248	252	252
252	248	252 (this is color 3)
252	248	248
248	248	248

Table 5-3 The beginning of the ALTERN.MAP sequence

These numbers range from 0 to 255. The first few lines of ALTERN.MAP, a grayscale color map, are shown in Table 5-3.

For example, if you load in ALTERN.MAP, color 3 (the fourth row, because you count up from 0) would have a red component of 252, a green component of 248, and a blue component of 252. This is an almost-white shade with an almost invisible red-blue (magenta) tinge. Note that comments can be placed after the numbers. Color 0 is usually 0 0 0 because it is used for the normally black overscan border of your screen.

(S) Pauses cycling, prompts for a file name, and saves the current palette to the named file (.MAP assumed).

Enter Palette Editor (E)

Command Function: Enter palette editing mode for altering the color map in use.

Menu Access: PALETTE EDITING MODE under the COLORS section of the MAIN MENU.

Command-Line Access: none

Comments: The palette editing mode is a sophisticated mechanism for customizing and adjusting the Fractint color palette. It requires a graphics adapter supporting 256 colors, such as a VGA or super VGA.

Skilled fractal artists spend a lot of time manipulating the colors of their fractals; this is what separates the beginners from the true artists. When the palette editing mode is entered, an empty palette frame is displayed. Use the cursor keys to position the frame, use the (PAGE UP) and (PAGE DOWN) keys to size it, and then press (ENTER) to display the palette in a grid. Figure 5-43 shows the palette editor grid. Note that the palette frame shows R(ed) G(reen) and B(lue)

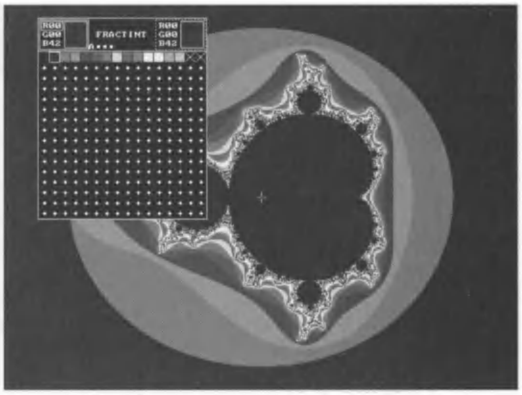


Figure 5-43 The palette editor grid

values for two color registers at the top. The active color register has a solid frame, the inactive register's frame is dotted. Within the active register, the active color component is framed. The mouse controls a cross hair.

Once the palette frame is displayed, the following commands are available:

ESC	Exit to the color-cycling mode.
H	Hide the palette frame to see the full image; the cross hair remains visible and all functions remain enabled; press H again to restore the palette display.
↑, ↓, ←, →	Move the cross-hair cursor around. In auto mode (the default), the center of the cross hair selects the active color register. Cursor-control keys move the cross hair faster. A mouse can also be used to move around.
R, G, B	Select the red, green, or blue component of the active color register for the subsequent INSERT OR DELETE and SELECT PREVIOUS OR NEXT COLOR COMPONENT IN ACTIVE REGISTER commands.
+, ', −	Increase or decrease the active color component by 1. Numeric keypad + and − keys do the same.
PAGE UP, PAGE DOWN	Increase or decrease the active color component by 5. Moving the mouse up or down with the left button held is the same.
0 – 6	Set active color component to 0, 10, 20,..., 60.

SPACEBAR	Select the other color register as the active one. (In auto mode this results in both registers set to the color under the cursor until you move it.)
,, .	Rotate the palette one step.
<,>	Rotate the palette continuously (until next keystroke).
C	Enter color-cycling mode.
=	Create a smoothly shaded range of colors between the two color registers.
D	Duplicate the inactive color register in active color.
T	Stripe-shade; create a smoothly shaded range of colors between the two color registers, setting only every *n*th register; after pressing T, press a number from 2 to 9 which is used as *n*.
SHIFT-F2 – F9	Store the current palette in a temporary save area associated with the function key; these save palettes are remembered only until you exit palette editing mode.
F2 – F9	Restore the palette from a temporary save area.
W	Converts the palette (or current exclude range) to grayscale.
N	Make a negative color palette. If in the X mode (see following text), only the current color is negated. If in the Y mode (see following text), only the current range is negated. Otherwise, the entire palette is negated.
\	Move or resize the palette frame. The frame outline is drawn; it can then be moved and sized with the cursor keys, PAGE UP and PAGE DOWN. Press ENTER when done moving/sizing.
I	Invert frame colors, useful with dark colors.
S	Prompt for a palette map file name (default file type is .MAP), and save the palette to that map file.

A	Toggle auto mode on or off. When on, the active color register follows the cursor; when off, ENTER must be pressed to set the register to the color under the cursor.
ENTER	Useful only when auto is off, as previously described; double-clicking the left mouse button is the same as ENTER.
X	Toggle exclude mode on or off—when toggled on, only the active color is displayed.
Y	Toggle exclude range on or off—when on, only colors in the range of the two color registers are shown.
!	(SHIFT-1) Swap the values in the red and green columns.
@	(SHIFT-2) Swap the values in the green and blue columns.
#	(SHIFT-3) Swap the values in the red and blue columns.
F	Toggle the "freestyle" palette editing mode (yes, a mode within a mode). When in the freestyle palette editing mode and with the cross-hair cursor inside the palette table, the mouse performs special editing functions. Freestyle mode changes a range of palette values (the upper and lower boundsof the palette spread are shown with checkered borders). While in freestyle mode, the CONTROL-INSERT and CONTROL-DELETE keys change the width of this palette spread. Pressing ENTER or double-clicking the left mouse button "fixes" the color values (exiting freestyle mode with the F toggle with an unfixed color band reverts that band to its unfixed values).

Make Starfield A

Command Function: Make a starfield from your favorite fractal image.

Menu Access: MAKE STARFIELD under the COLORS section of the MAIN MENU.

Command-Line Access: none

Comments: Once you have generated your favorite fractal image, you can convert it into a fractal starfield with the A transformation (for "astronomy"). The screen is filled with star-like distributions of individual pixels of different degrees of brightness. Stars are generated on a pixel-by-pixel basis—the odds that a particular pixel will coalesce into a star are based (partially) on the color index of that pixel.

If the screen is entirely black and the star density per pixel is set to 30, then a starfield transformation will create an evenly distributed starfield with an average of one star for every 30 pixels. Therefore, if you're on a 320 x 200 screen you have 64,000 pixels and would end up with about 2,100 stars. By introducing the variable of "clumpiness," we can create more stars in areas that have higher color values. At 100% clumpiness, a color value of 255 will change the average of finding a star at that location to 50:50. A lower clumpiness value will lower the amount of probability weighting. To create a spiral galaxy, draw your favorite spiral fractal (IFS, Julia, or Mandelbrot) and perform a starfield transformation. For general starfields, we recommend transforming a plasma fractal. For starfields based on fractals with lakes, such as the Mandelbrot fractal, be sure to set `inside=255` for the best effect.

Real starfields have many more dim stars than bright ones because very few stars are close enough to appear bright. To achieve this effect, the program will create a bell curve based on the value of ratio of dim stars to bright stars. After calculating the bell curve, the curve is folded in half and the peak is used to represent the number of dim stars.

Starfields can be shown in 256 colors only. Fractint will automatically try to load ALTERN.MAP and abort if the map file cannot be found.

FRACTINT'S AUTOKEY FEATURE

Fractint's *autokey* feature allows you to control Fractint using files containing sequences of simulated keystrokes. The autokey feature is different from parameter files in that parameter files contain sequences of command-line options used to control Fractint, while autokey files simulate sequences of keystrokes that Fractint treats as if your fingers were pressing the keyboard. You can set up Fractint sessions that use autokey sequences to teach Fractal exploring methods, generate special effects, attract people to a booth, etc.

A sample autokey file (DEMO.KEY) designed to show off some of the capabilities of Fractint and a batch file to run it (DEMO.BAT) are included on your companion disk. Type `demo` at the DOS prompt to run it.

Autokey mode and its various options are enabled using command-line parameters, enterable either on the command line or via the G (GIVE COMMAND STRING) command. The command-line parameters are

autokeyname=<filename> This command-line option sets the name of the file that the autokey option is going to use. If this option is not specified, the autokey option uses the default filename AUTO.KEY.

autokey=play This command-line option causes Fractint to begin reading keystrokes from the autokey file name rather than the keyboard. Fractint will continue to process its keystrokes from this file until it reaches the end of the file or you press the ESC key on the keyboard.

autokey=record This command-line option causes Fractint to begin recording and saving your keystrokes into the current autokeyname file. Once Fractint begins recording keystrokes, it continues to do so until you exit Fractint.

Autokey files are text-based, and can be created or edited using any ASCII text editor. Autokey files consist of quoted text, special keystroke symbols, and special autokey commands. Autokey files can include comments—anything from an unquoted semicolon to the end of a line is a comment. As with parameter files, several autokey sequences can either be bundled in a single line or placed on individual lines (when in record mode, Fractint stores every keystroke on a separate line). Because most Fractint sessions involve many keystrokes that aren't easy to put into text files (like F3 and PAGE UP), autokey files use a number of special symbols to represent them.

Table 5-4 lists the commands that may be placed in an autokey file.

Making Fractint demos can be tricky. Here are some useful hints:

- Start Fractint with `fractint autokeyname=mydemo.key autokey=record` or use the GIVE COMMAND STRING <G> sequence to issue the two options. (If you use the G command, remember to enter the autokeyname option first.)

- When in record mode, avoid using the cursor keys to select file names, fractal types, formula names, etc. Instead, try to type in names, and use the full name as often as possible. This will ensure that the exact item you want gets chosen during playback, even if the list you are selecting from is different.

- Beware of video mode assumptions. It is safest to build a separate demo for different resolution monitors. Not everyone can run his or her computer in 640 x 480 256-color mode.

- When you finish recording, clean up your autokey file. Insert a CALCWAIT after each keystroke that triggers something that takes a variable amount of time (calculating a fractal, restoring a file, saving a file). Watch for unwanted WAIT statements and fragmented text strings (while in record mode, Fractint dutifully recorded your every pause over 0.5 second and breaks up your text strings if you type too slowly). Convert multiple symbol entries to the *nn option.

Command	Meaning
"text"	Assume that the characters enclosed in quotes have been entered at the keyboard. For example, the sequence "t" "ifs" would issue the "t" (type) command and then enter the letters"i", "f", and "s" to start selecting the ifs type. Note that the autokey player treats the above two strings exactly the same as the single string "tifs"—the former sequence is just a little easier to understand.
WAIT <nnn.n>	Wait nnn.n seconds before continuing.
CALCWAIT	Pause until the current fractal calculation or file save or restore is finished. This command makes demo files more robust because calculation times depend on the speed of the machine running the demo—a "WAIT 10" command may allow enough time to complete a fractal on one machine, but not on another. The record mode does not generate this command—it should be added by hand to the autokey file whenever there is a process that should be allowed to run to completion.
SYMBOL [* nn]	Causes the named symbol to be entered (and repeated "nn" times, if the "* nn" notation is used). For example, PAGEUP * 10 causes Fractint to think you have pressed the PAGE UP key ten times. The "* nn" feature is useful for items like resizing and moving zoom boxes. Table 5-5 lists the symbol names currently recognized by the autokey command.
mmmm [* nn]	This notation is used whenever Fractint is in autokey record mode and needs to record the use of a special key that doesn't have an autokey symbol name. The number 'mmmm' is Fractint's internal representation for that keystroke. The DEMO.KEY file distributed on your companion disk has several such entries: the CONTROL-↑ keystroke combination, for example, is stored internally as keystroke number 1141.
MESSAGE nn <message>	Places a message on the top of the screen for "nn" seconds. The message string itself is not enclosed in quotes.

GOTO target	Locate the label "target:" in the autokey file and proceed from there. The label can be any word that does not duplicate a keyword. It must be present somewhere in the autokey file with a colon after it. For example:
	`MESSAGE 2 This is executed once`
	`start:`
	`MESSAGE 2 This is executed repeatedly`
	`GOTO start`
	GOTO is useful mainly for writing continuous loop demonstrations. It can also be useful for skipping sections of an autokey file when debugging it.

;	A semicolon indicates that the rest of the line containing it is a comment.

Table 5-4 Autokey commands

Symbol	Special Key
ENTER	ENTER
ESC	ESC
F1..F10	F1–F10
SF1..SF10	SHIFT-F1–SHIFT-F10
CF1..CF10	CONTROL-F1–CONTROL-F10
AF1..AF10	ALT-F1–ALT-F10
PAGEUP	PAGE UP
PAGEDOWN	PAGE DOWN
HOME	HOME
END	END
LEFT	←
RIGHT	→
UP	↑
DOWN	↓
INSERT	INSERT
DELETE	DELETE
TAB	TAB

Table 5-5 Autokey special-key symbols

- Add plenty of comments with the ";" feature, so you know what is going on when you look at your autokey file sometime in the future.
- It is a good idea to add an INSERT command before a GOTO that restarts the demo. The INSERT key resets Fractint just as if you exited the program and restarted it. It's far too easy to forget that you changed the iteration limit from 150 to 32,000 at some point in the middle of your demo loop.

WARNING: An autokey file built for any one version of Fractint will probably require some retouching before it works with future releases of Fractint. The authors have no intention of making sure that the same sequence of keystrokes will have exactly the same effect from one version of Fractint to the next. That would pretty much freeze Fractint development, and we just love to keep enhancing it!

COMMAND-LINE-ONLY COMMANDS

This section documents Fractint commands that exist only in command-line or batch form and do not have associated keystrokes or menu items. It is organized into video-related, printer-related, fractal-related, and miscellaneous commands, and is roughly in the order of importance/usefulness within those categories.

Video-Related Command-Line Options

```
adapter=cga|ega|egamono|mcga|vga|hgc|
      ati|everex|trident|ncr|video7|genoa|paradise|chipstech|tseng3000|
      tseng4000|aheada|aheadb|oacktech
```

Normally, Fractint automatically detects the type of video adapter on your system. The `adapter=` option is for those cases where Fractint's autodetection logic doesn't work (either "mis-detecting" your adapter type or, in the worst cases, messing up your screen during the process). It causes Fractint to skip the autodetect logic and assume the named adapter type is present. Use this only if the autodetect logic fails for your adapter. Entries on the second and third lines are for super VGA chipsets, and imply VGA compatibility. This will affect the default mode and any other Fractint features that depend on auto-detecting adapters. Note that this command does not actually change what your adapter can do—only what Fractint *thinks* your adapter can do. (The "|" means "or"—specify just one of these options.)

vesadetect=no Suppresses Fractint's VESA detection. Normally, Fractint checks for VESA support as its first Super VGA Chipset mode, and looks no further if it finds that your adapter is VESA compliant. This option bypasses that check, and is useful if your adapter has a faulty VESA driver.

textsafe=yes|no|bios|save When you press a key (such as X, Y, or TAB) that switches from a graphics image to text mode; Fractint remembers the image and displays it the next time you choose graphics mode. This is a fast method for saving the graphics screen, but it does not work perfectly for every graphics adapter in every video mode—especially high-resolution modes. If this method does not work on your computer, you can use the `textsafe=` command to specify methods that are slower but safer. Try various `textsafe=` options if you have the following display problems:

- A display image that is either garbled or overlaid with lines and dashes when you return to the graphics image after opening a menu, pressing (TAB), or pressing (F1) for help.
- A blank screen when you start running Fractint.

The following are the `textsafe=` options:

yes This option is the default. When you switch either to or from the graphics mode, Fractint saves only the part of video memory that EGA and VGA adapters are supposed to modify during the mode change.

no This option uses a monochrome, 640 x 200 x 2-mode to display text. It displays text quickly, but it uses characters that are chunky and, of course, colorless. If you use this option, specifying `textcolors=mono` might improve the text display.

bios This option saves memory just as `textsafe=yes` does, but it uses the adapter's BIOS routines to save and restore the graphics image. This option is fast, but it works perfectly on only a few adapters

save If all other options fail, try this one. It is slow, but it should work on all adapters and in all modes. It directs Fractint to save and restore the entire image. Expanded or extended memory is used if enough is available; otherwise, a temporary disk file is used.

askvideo=yes|no If "no," this eliminates the prompt asking you if the video mode specified in a file to be restored is OK for your current video hardware.

exitmode=nn Sets the bios-supported video mode to use upon exit (if not mode 3)—nn is the mode in hexadecimal. For people who like nonstandard text modes.

afi=yes Normally, Fractint accesses IBM 8514/A adapters and their clones by writing directly to their registers. This option forces Fractint to use the slower HDILOAD interface instead.

textcolors=mono Set text screen colors to simple black and white. Use this if the shades of color do not show up well on your monochrome screen.

textcolors=<aa>/<bb>/<cc>/... Set text screen colors. Each value is a hexadecimal number, with the first digit the background color from 0 to 7, and the second digit the foreground color from 0 to 15. Hex color values are as follows:

0	black	8	gray
1	blue	9	light blue
2	green	A	light green
3	cyan	B	light cyan
4	red	C	light red
5	magenta	D	light magenta
6	brown	E	yellow
7	white	F	bright white

A total of 31 different colors can be specified, with their use in Fractint as follows:

Heading:

1	Fractint version information
2	heading line development information (not used in released version)

Help:

3	subheading
4	main text
5	instructions at bottom of screen
6	hotlink field
7	highlighted (current) hotlink

Menu, selection boxes, parameter input boxes:

8	background around box and instructions at bottom
9	emphasized text outside box
10	low intensity information in box
11	medium intensity information in box
12	high intensity information in box (e.g., heading)
13	current key-in field
14	current key-in field when it is limited to one of *n* values
15	current choice in multiple choice list
16	speed key prompt in multiple choice list
17	speed key in multiple choice list

General (tab key display, IFS parameters, "thinking" display):

18	high intensity information
19	medium intensity information
20	low intensity information
21	current key-in field

Disk video:

22	background around box
23	high intensity information
24	low intensity information

Diagnostic messages:

25	error
26	information

Credits screen:

27	bottom lines
28	high intensity divider line
29	low intensity divider line
30	primary authors
31	contributing authors

The default is

```
textcolors=1F/1A/2E/70/28/71/31/78/70/17/
       1F/1E/2F/5F/07/0D/71/70/78/0F/
       70/0E/0F/4F/20/17/20/28/0F/07
```

(In a real command file, all values must be on one line.)

Printer-Related Command-Line Options

```
printer=<type>[/<resolution>[/<port#>]]
```

The command printer=<type>[/<resolution>[/<port#>]] defines your basic printer setup. The default printer type is the Epson-compatible, dot-matrix printer. Table 5-6 shows the possible values for <type>.

For dot-matrix and laser printers, the resolution is in dots per inch. Possible values are 60, 120, and 240 for the Epson/IBM; 75, 150, and 300 for the LaserJet; 90 and 180 for the PaintJet; and 10 through 600 for PostScript. Plotter resolution is in portions of a page, with acceptable values from 1 to 10 (3 means 1/3 of a page). The Printer port can be 1, 2, and 3 for LPT1-3 via the BIOS (21 or 22 for LPT1-2 using direct access); 11, 12, 13, and 14 for COM1-4 via the BIOS (31 or 32 for COM1-2 using direct access). Direct access methods are faster—when they work. With PostScript, a negative port number can be used to redirect printing to a file.

printfile=<filename> Causes printing to go to the file <filename> rather than directly to the printer. The file name is incremented after each print to file operation.

Type	Name
CO	Star Micronix/Epson Color
EP	Epson-compatible dot matrix
HP	Hewlett-Packard LaserJet
IB	IBM-compatible
PA	Hewlett-Packard PaintJet
PL	Plotter using HP-GL
PS	PostScript portrait
PSL	PostScript landscape

Table 5-6 Printer types

title=yes Enables or disables the printing of a Fractint-supplied title with the output (the default is no).

comport=port/baud/opts Performs serial printer port initialization. "Port" may be 1, 2, 3, or 4 for com1 through com4. "Baud" is the baud rate, which may be 115, 150, 300, 600, 1200, 2400, 4800, or 9600. "Options" includes bits, stop bits, and parity in any order. For example,

```
fractint comport=1/9600/n81
```

sets the printer for port com1, 9600 baud, no parity, 8 bits per character, and 1 stop bit.

linefeed=crlf|lf|cr Forces the use of control characters at the end of each line (crlf is the default).

colorps=yes|no This option is ignored for all but PostScript printers. It enables or disables Color PostScript extensions (the default is no).

rleps=yes|no This option is ignored for all but PostScript printers. It enables or disables PostScript RLE encoding (the default is no). Run-Length-Encoding results in smaller files—but they may take longer to print. The run length encoding code is based on pnmtops, which is copyright © 1989 by Jef Poskanzer, and carries the following notice: "Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appears in all copies and that both that copyright notice and this permission notice appear in supporting documentation. This software is provided 'as is' without express or implied warranty."

epsf=1|2|3 Forces print to a PostScript file (default file name FRACT001.EPS). The PostScript mode is turned on. Lower numbers force stricter adherence to the Encapsulated PostScript (EPS) format. 1 means by-the-book. 2 allows some EPS "no-nos" like settransfer and setscreen—but includes code that should make the code still work without affecting the rest of the non-EPS document. 3 is a free-for-all. The default value is 1.

translate=yes|<nnn> This option is ignored for all but PostScript printers. Translate=yes prints the negative image of the fractal. `Translate=nnn` reduces the image to that many colors. A negative value causes a color reduction as well as a negative image.

halftone=frequency/angle/style [/f/a/s/f/a/s/f/a/s] This option is ignored for all but PostScript printers. It is presented here for advanced PostScript owners and those who want to experiment. This option defines the halftone screen for PostScript, which affects how colors are rendered as dot patterns. The first value, frequency, defines the number of halftone lines per inch. The second chooses the angle (in degrees) at which the screen lies. The third option chooses the halftone "spot" style. Good default frequencies are between 60 and 80; good default angles are 45 and 0; the default style is 0. If the `halftone=` option is not specified, Fractint will print using the printer's default halftone screen, which should have been already set to do a fine job on the printer.

These are the only three options used when *colorps=no*. When color PostScript printing is being used, the other nine options specify the red, green, and blue screens. A negative number in any of these places will cause it to use the previous (or default) value for that parameter. NOTE: Especially when using color, the built-in screens in the printer's ROM may be the best choice for printing. The default values are as follows:

```
halftone=45/45/1/45/75/1/45/15/1/45/0/1
```

and these will be used if Fractint's halftone is chosen over the printer's built-in screen. The current halftone styles are

0	Dot
1	Dot (Smoother)
2	Dot (Inverted)
3	Ring (Black)
4	Ring (White)
5	Triangle (Right)
6	Triangle (Isosceles)
7	Grid

8	Diamond
9	Line
10	Microwaves
11	Ellipse
12	Rounded Box
13	Custom
14	Star
15	Random
16	Line (slightly different)

halftone=r/g/b This option is ignored for all but PaintJet printers. This halftone option is presented here for advanced PaintJet owners and those who want to experiment. It sets the gamma adjustment of the red, green, and blue components. Higher gamma values result in colors with more contrast being sent to the printer. Note that the PAINTJET.MAP color-map file uses colors specifically designed to match those on the paintjet.

plotstyle=0|1|2 This option is ignored for all but HP-GL-compatible plotters. It selects one of several plotting styles. The available styles are

0 3 parallel lines (red/green/blue) are drawn for each pixel, arranged like "///." Each bar is scaled according to the intensity of the corresponding color in the pixel. Using different pen colors (e.g., blue, green, violet) can come out nicely. The trick is to not tell anyone what color the bars are supposed to represent and they will accept these plotted colors because they look nice.

1 Same as 0, but the lines are also twisted. This removes some of the order of the image, which is a nice effect. It also leaves more whitespace making the image much lighter, but colors such as yellow are actually visible.

2 Color lines are at the same angle and overlap each other. This type has the most whitespace. Quality improves as you increase the number of pixels squeezed into the same size on the plotter.

Fractal-Related Command-Line Options

periodicity=no|show|nnn Allows control of periodicity checking; "no" turns it off, "show" lets you see which pixels were painted the inside color due to being caught by periodicity. Specifying a number causes a more conservative periodicity test (each increase of 1 divides the test tolerance by 2). Entering a negative number lets you turn on "show" with that number. Type lambdafn `function=exp` needs periodicity turned off to be accurate—there may be other cases.

symmetry=<symmetry> This option forces symmetry to one of `None`, `Xaxis`, `Yaxis`, `XYaxis`, `Origin`, or `PI` symmetry. Some fractals are symmetrical and have parts that are reflections of their other parts. For example, the top and bottom of the Mandelbrot fractal are reflections of each other. The Mandelbrot fractal has *x*-axis symmetry, because the top points are the reflections of the bottom points about the *x*-axis. *Y*-axis symmetry means the left and right sides of a fractal are reflections of each other. *XY*-axis symmetry is a combination of both of these. Origin symmetry reflects upper points to lower points on the opposite side. Finally, PI symmetry describes the symmetry of periodic fractals that repeat themselves every PI units.

This command forces symmetry whether or not the fractal really exhibits it. A portion of the fractal is calculated, and the symmetrical parts are reflections of the calculated part. The Fractint authors have attempted to automatically use symmetry when it exists, but they have not caught every case. For example, any of the fractal types with "fn" in their name (such as "fn+fn") exhibit different symmetry depending on which functions are used to replace "fn" in the formula. If you are experimenting with a fractal and can see that it has symmetry that Fractint doesn't know about, you can set the symmetry with this command and make the fractal run faster, because fewer points have to be calculated. You can also apply symmetry just to change any fractal and see how it looks. If you type

```
fractint symmetry=xyaxis
```

and plot the Mandelbrot fractal, you will see that it has changed. What is normally the upper-left corner of the Mandelbrot image is reflected to the other three corners; the right half of the fractal is no longer the same.

initorbit=pixel, initorbit=<nnn>/<nnn> Allows control over the value used to begin each Mandelbrot-type orbit. The command `initorbit=pixel` is the default for most types; this command initializes the orbit to the complex number corresponding to the screen pixel. The command `initorbit=nnn/nnn` uses the entered value as the initializer.

rseed=nn Forces Fractint to use value "nn" as its initial random number seed (otherwise, Fractint uses a value based on the current time). Useful when you need to reproduce an image that would otherwise have some degree of randomness involved, such as a plasma cloud.

showdot=nn Causes Fractint to color the pixel it is working on using color "nn" (the pixel color then gets reset when Fractint finishes its pixel calculation). Useful when an image takes a long time to calculate and you're not exactly sure how far along the image has been processed during that calculation.

formulafile=<formulafilename> Lets you specify the default formula file for `type=formula` fractals (the default is FRACTINT.FRM). Handy if you want to generate one of these fractal types in batch mode.

formulaname=<formulaname> Lets you specify the default formula name for `type=formula` fractals (the default is no formula at all). Required if you want to generate one of these fractal types in batch mode, as this is the only way to specify a formula name in that case.

ifsfile=<ifsfilename> Lets you specify the default lfile for `type=ifs` fractals (the default is fractint.ifs). Handy if you want to generate one of these fractal types in batch mode.

ifs=<ifssystemname> Lets you specify the default ifs name for `type=ifs` fractals (the default is the first type in the file). Required if you want to generate one of these fractal types in batch mode, as this is the only way to specify an ifs name in that case.

lfile=<lfilename> Lets you specify the default lfile for `type=lsystem` fractals (the default is FRACTINT.L). Handy if you want to generate one of these fractal types in batch mode.

lname=<lsystemname> Lets you specify the default lsystem name for `type=lsystem` fractals (the default is the first type in the lfile). Required if you want to generate one of these fractal types in batch mode, as this is the only way to specify an L-system name in that case.

function=<fn1>[/<fn2>[/<fn3>[/<fn4>]]] Allows setting variable functions found in some fractal type formulas. Possible values of the functions are `sin`, `cos`, `tan`, `cotan`, `sinh`, `cosh`, `tanh`, `cotanh`, `exp`, `log`, `sqr`, `recip` (1/z), `ident` (identity), and `cosxx` (an older cos function that contained a bug, left in for backward compatibility).

Miscellaneous Command-Line Options

autokey=play|record, autokeyname=filename These commands control Fractint's autokey feature, and are described in detail in this chapter's *autokey mode* section.

makemig=nn/nn This command causes Fractint to run in batch mode and build a multi-image GIF file from a number of component images. This command is not normally run manually, but is part of the MAKEMIG.BAT file created by the (B) (SAVE CURRENT PARAMETERS) command.

gif87a=yes Backward-compatibility switch to force creation of GIF files in the GIF87a format. Fractint now creates files in the new GIF89a format, which permits storage of fractal information within the format. This switch is needed only if you want to view Fractint images with a GIF decoder that cannot accept the newer format. The disadvantage of this option is that no fractal information will be stored with the file, and Fractint will not know how the file was created.

savetime=nn Forces Fractint to save the image it is working on every "nn" minutes. Added at the request of a frustrated user who had his computer set up to work on a fractal image all weekend and lost his power sometime on Saturday.

exitnoask=yes Suppresses Fractint's "are you sure?" safety message when you press ESC to exit Fractint.

colors=@filename|colorspec This option is generated automatically by the B (SAVE CURRENT PARAMETERS) command if the record colors prompt is set to Yes or @filename. The COLORSPEC option stores the color values using a compressed internal format.

ranges=nn[/nn[/nn...]] Causes Fractint's coloring scheme to use ranges of iteration values rather than a different color for each iteration value. Iteration counts up to and including the first value are mapped to color number 0, up to and including the second value to color number 1, and so on. The values must be in ascending order.

A negative value can be specified for "striping." The negative value specifies a stripe width, the value following it specifies the limit of the striped range. Two alternating colors are used within the striped range. Example:

```
RANGES=0/10/30/-5/65/79/32000\
```

This example maps iteration counts to colors as follows:

Color	Iterations
0	unused (formula always iterates at least once)
1	1 to 10
2	11 to 30
3	31 to 35, 41 to 45, 51 to 55, and 61 to 65
4	36 to 40, 46 to 50, and 56 to 60
5	66 to 79
6	80 and greater

hertz=nnn Sets the frequency of the sound produced by the `sound=x/y/z` option. Legal values are 200 through 10,000.

dither=yes Dither a color file into two colors for display on a black-and-white display. This gives a poor-quality display of gray levels. Note that if you have a 2-color display, you can create a 256-color GIF file with disk video and then read it back in dithered.

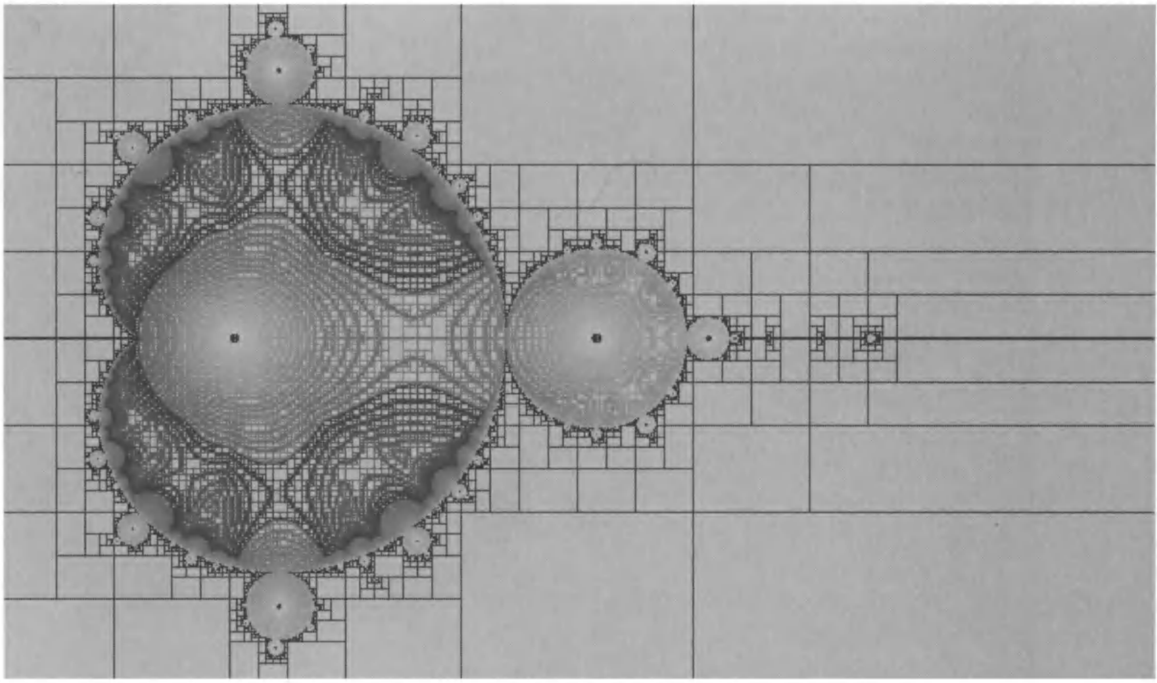
orbitdisplay=yes Causes the file ORBITS.RAW to be opened and the points generated by orbit fractals or IFS fractals to be saved in a raw format. This file can be read by the *Acrospin* program, which can rotate and scale the image rapidly in response to cursor-key commands. The file name ORBITS.RAW is fixed and will be overwritten each time a new fractal is generated with this option.

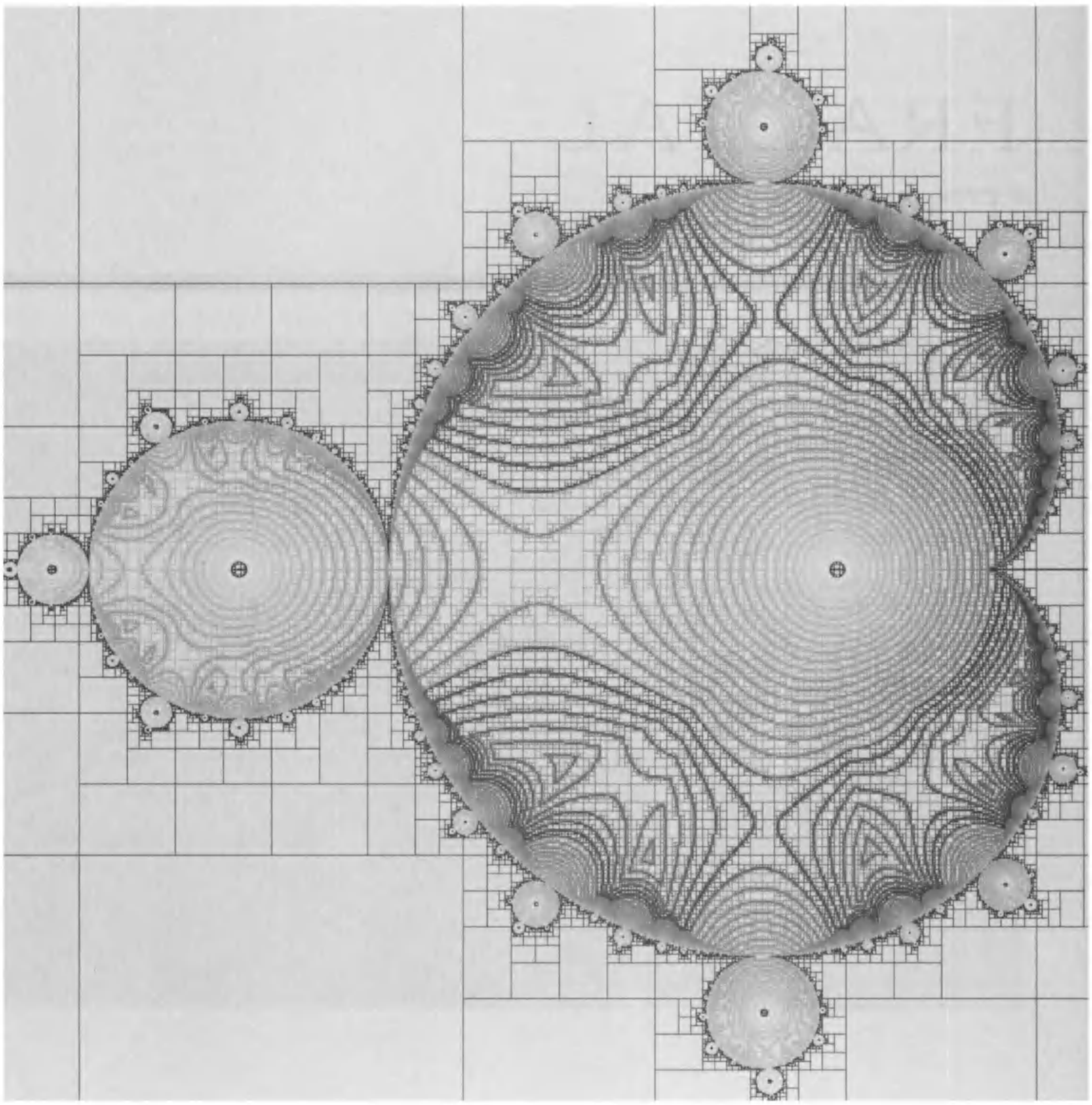
fpu=387|iit|noiit Normally, Fractint automatically detects the presence and type of floating-point unit (FPU) your PC has, and uses hand-tuned assembler routines that squeeze the most performance out of it. This option forces Fractint to assume the presence or absence of an advanced 80387 or IIT math coprocessor on those occasions when Fractint might not automatically detect one.

makedoc=filename When used on the command line, causes Fractint to build a special FRACTINT.DOC file from its internal help files and then exit.

CHAPTER 6

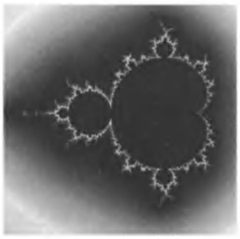
FRACTAL TYPES





CHAPTER 6

FRACTAL TYPES



This chapter is your explorer's atlas of the fractal universe. At last count there were 95 fractal types listed on the Fractint SELECT FRACTAL TYPE screen. Each of these types contains rich landscapes for you to explore. These worlds are huge beyond imagination, and the territory is largely uncharted. At the scale of your computer screen, each of these fractal landscapes stretches many millions of miles and could not be fully explored in a hundred lifetimes. Having 95 such worlds to explore might seem enough, but Fractint takes your much further. Some of the fractal types are not individual worlds at all. They are galaxies of worlds, representing whole fractal families that you select as you supply different parameters. Finally, among these 95 fractal types are three general purpose fractal-creating engines that let you invent your own fractal universe!

The first version of Fractint contained only two fractal types: the classic Mandelbrot set and the Julia set. Then other fractal types were added as the authors came across them in the fractal literature or invented new fractals by coding variations of existing fractals. Program users like yourself began sending the programmers additional fractal types. At first the programmers accepted and added to Fractint just about any new fractal type they received, but as time went by, the criteria for acceptance became more selective and more orderly. In an effort to reduce the number of fractal types, many of the algorithms were consolidated under single, more general types. The present list of 95 fractal types grew by this evolutionary process, driven by the enthusiasm of Fractint aficionados around the world who continue to propose new types and new program enhancements.

This chapter will help you find your way through this cornucopia of fractal possibilities, and provide you with useful information along the way. In these pages you will find:

- Ideas for exploring fractal types and generating great images.
- Details of how algorithms work (expecially helpful for the programmers among you).
- Source code reference—where to find the code that generates each type.
- All-new examples, complete with a range of options and outstanding colors.
- Interesting historical tidbits.

SELECTING FRACTAL TYPES AND ACCESSING HELP

You can access the SELECT A FRACTAL TYPE screen from the MAIN MENU or by pressing the (T) command. You will then see the names of Fractint's types on the screen in alphabetical order by rows. The number of types has grown to the point where Fractint can't even show them all in one screen, so you'll see (MORE) on either the top or bottom of the screen, showing you which end of the screen is hiding additional fractal types. You can use the (HOME) or (END) keys to jump to the beginning or end of the the types list and see the hidden type names.

To select a fractal type, you can navigate around the SELECT A FRACTAL TYPE screen using the arrow keys. You can also begin to type a fractal name, and Fractint's speedkey feature will jump the highlight to the first fractal name matching the letters you have typed so far.

Fractint has a flexible help system that you can use to learn more about each fractal type. While the SELECT A FRACTAL TYPE screen is showing, pressing (F1) will take you to the SUMMARY OF FRACTAL TYPES help screen showing the mathematical formula used by each fractal type. You can use the (↑) and (↓) keys to move the blue-green highlight from type to type. Pressing (ENTER) while a type name is highlighted will cause a hypertext jump to additional information for that kind of fractal. Pressing (BACKSPACE) backs up to the previous help screen. Pressing (ESC) exits the help system.

After you have selected a fractal on the SELECT A FRACTAL TYPE screen, pressing (ENTER) will take you to the PARAMETERS FOR FRACTAL TYPE <Fractal Name> screen. A fractal parameter is a number, function, or algorithm choice that affects how the fractal is calculated and, therefore, changes the resulting fractal image. At the bottom of the PARAMETERS screen is a box containing the mathematical formula for the fractal (the same formula you can also see in the SUMMARY OF FRACTAL TYPES help screen mentioned above). You can look at the formula to see how the parameters affect the fractal calculation. You can also just try different parameters and see what happens to the image without worrying about the mathematics. The default parameters are designed to give interesting images, so for initial explorations you don't need to change parameters. You can also access the fractal type help information from the PARAMETERS screen by pressing F1.

FRACTAL TYPE REFERENCE

To help you sort out the many fractal types available in Fractint, this chapter divides the types into groups, and discusses each group separately. Table 6-1 lists the fractal types alphabetically, and tells which section of this chapter covers each type. Use this chart to look up any particular fractal type you would like to know more about, and find out where it is discussed in the chapter. Don't forget to also consult the on-line help information discussed earlier in this chapter. The on-line help contains a wealth of information about fractal types.

In this chapter, Fractal types are divided into Escape-Time Fractals, 3-D Fractals, Bifurcation Fractals, Orbit Fractals, and Fractal Miscellania. This division is more practical than theoretical, and is designed to help you find what you are looking for. The 3-D Fractals section is where to look for all the types that can make stereo images you can view with the red/blue glasses that came with this book, even fractals that could be categorized differently.

In each of the sections below, you will find a brief discussion of the characteristics of all the fractals in that section, followed by particular information about each type. The mathematical formula used to generate the fractal is spelled out, along with a guide to finding the code in the Fractint source on your book disk.

Escape-Time Fractals

The most widely known kind of fractal images are generated by the escape-time method used to generate the historic first images of the famous Mandelbrot set. For each pixel on your computer screen, the escape-time method generates a series of orbit values by repeating, or iterating, a formula. For each iteration, the new orbit value is checked against an escape criterion. If the criterion is met, the computation is stopped and the screen pixel is colored according to the time it took for the orbiting number to escape. The escape time is taken to be the number of iterations required before the orbit value escaped.

Fractint Type	Chapter Section	Fractal Category
barnsleyj1	Escape-Time Fractals	Mandelbrot/Julia
barnsleyj2	Escape-Time Fractals	Mandelbrot/Julia
barnsleyj3	Escape-Time Fractals	Mandelbrot/Julia
barnsleym1	Escape-Time Fractals	Mandelbrot/Julia
barnsleym2	Escape-Time Fractals	Mandelbrot/Julia
barnsleym3	Escape-Time Fractals	Mandelbrot/Julia
bif+sinpi	Fractal Miscellania	Bifurcation
bif=sinpi	Fractal Miscellania	Bifurcation
biflambda	Fractal Miscellania	Bifurcation
bifmay	Fractal Miscellania	Bifurcation
bifstewart	Fractal Miscellania	Bifurcation
bifurcation	Fractal Miscellania	Bifurcation
cellular	Fractal Miscellania	Cellular Automaton
circle	Fractal Miscellania	Moire Pattern
cmplxmarksjul	Escape-Time Fractals	Mandelbrot/Julia
cmplxmarksmand	Escape-Time Fractals	Mandelbrot/Julia
complexbasin	Escape-Time Fractals	Escape Time to Finite Attractor
complexnewton	Escape-Time Fractals	Escape Time to Finite Attractor
diffusion	Fractal Miscellania	Random
dynamic	Orbit Fractals	Superimposed Orbits
fn(z)+fn(pix)	Escape-Time Fractals	Escape Time to Infinity Generalized
fn(*z***z*)	Escape-Time Fractals	Escape Time to Infinity Generalized
fn*fn	Escape-Time Fractals	Escape Time to Infinity Generalized
fn**z*+*z*	Escape-Time Fractals	Escape Time to Infinity Generalized
fn+fn	Escape-Time Fractals	Escape Time to Infinity Generalized
formula	Escape-Time Fractals	User-Defined Escape Time
frothybasin	Escape-Time Fractals	Escape Time to Finite Attractor
gingerbreadman	Orbit Fractals	2-D Orbit
halley	Escape-Time Fractals	Escape Time to Finite Attractor
henon	Orbit Fractals	2-D Orbit
hopalong	Orbit Fractals	2-D Orbit
hypercomplex	Escape-Time Fractals	4-D Escape Time

Table 6-1 Fractint's fractal types

Fractint Type	Chapter Section	Fractal Category
hypercomplexj	Escape-Time Fractals	4-D Escape Time
icons	Orbit Fractals	2-D Orbit
icons3d	3-D Fractals	3-D Orbit Fractal
ifs	3-D Fractals	Iterated Function Systems
julfn+exp	Escape-Time Fractals	Mandelbrot/Julia Generalized
julfn+zsqrd	Escape-Time Fractals	Mandelbrot/Julia Generalized
julia	Escape-Time Fractals	Mandelbrot/Julia
julia(fn||fn)	Escape-Time Fractals	Mandelbrot/Julia Generalized
julia_inverse	Orbit Fractals	2-D Orbit
julia4	Escape-Time Fractals	Mandelbrot/Julia
julibrot	3-D Fractals	3-D Solid
julzpower	Escape-Time Fractals	Mandelbrot/Julia
julzzpwr	Escape-Time Fractals	Mandelbrot/Julia
kamtorus	Orbit Fractals	Superimposed Orbits
kamtorus3d	3-D Fractals	3-D Orbit Fractal
lambda	Escape-Time Fractals	Mandelbrot/Julia
lambda(fn||fn)	Escape-Time Fractals	Mandelbrot/Julia Generalized
lambdafn	Escape-Time Fractals	Mandelbrot/Julia Generalized
lorenz	Orbit Fractals	2-D Orbit
lorenz3d	3-D Fractals	3-D Orbit Fractal
lorenz3d1	3-D Fractals	3-D Orbit Fractal
lorenz3d3	3-D Fractals	3-D Orbit Fractal
lorenz3d4	3-D Fractals	3-D Orbit Fractal
lsystem	Fractal Miscellania	L-systems
lyapunov	Fractal Miscellania	Bifurcation
magnet1j	Escape-Time Fractals	Mandelbrot/Julia
magnet1m	Escape-Time Fractals	Mandelbrot/Julia
magnet2j	Escape-Time Fractals	Mandelbrot/Julia
magnet2m	Escape-Time Fractals	Mandelbrot/Julia
mandel	Escape-Time Fractals	Mandelbrot/Julia
mandel(fn||fn)	Escape-Time Fractals	Mandelbrot/Julia Generalized
mandel4	Escape-Time Fractals	Mandelbrot/Julia

Table 6-1 Fractint's fractal types (*continued*)

Fractint Type	Chapter Section	Fractal Category
mandelcloud	Orbit Fractals	Superimposed Orbits
mandelfn	Escape-Time Fractals	Mandelbrot/Julia Generalized
mandellambda	Escape-Time Fractals	Mandelbrot/Julia
mandphoenix	Escape-Time Fractals	Mandelbrot/Julia
manfn+exp	Escape-Time Fractals	Mandelbrot/Julia Generalized
manfn+zsqrd	Escape-Time Fractals	Mandelbrot/Julia Generalized
manlam(fn||fn)	Escape-Time Fractals	Mandelbrot/Julia Generalized
manowar	Escape-Time Fractals	Mandelbrot/Julia
manowarj	Escape-Time Fractals	Mandelbrot/Julia
manzpower	Escape-Time Fractals	Mandelbrot/Julia
manzzpwr	Escape-Time Fractals	Mandelbrot/Julia
marksjulia	Escape-Time Fractals	Mandelbrot/Julia
marksmandel	Escape-Time Fractals	Mandelbrot/Julia
marksmandelpwr	Escape-Time Fractals	Escape Time to Infinity Generalized
martin	Orbit Fractals	2-D Orbit
newtbasin	Escape-Time Fractals	Escape Time to Finite Attractor
newton	Escape-Time Fractals	Escape Time to Finite Attractor
phoenix	Escape-Time Fractals	Mandelbrot/Julia
pickover	Orbit Fractals	2-D Orbit
plasma	Fractal Miscellania	Random
popcorn	Orbit Fractals	Superimposed Orbits
popcornjul	Escape-Time Fractals	Escape Time to Infinity
quat	Escape-Time Fractals	4-D Escape Time
quatjul	Escape-Time Fractals	4-D Escape Time
rossler3d	3-D Fractals	3-D Orbit Fractal
sierpinski	Escape-Time Fractals	Escape Time to Infinity
spider	Escape-Time Fractals	Escape Time to Infinity
sqr(1/fn)	Escape-Time Fractals	Escape Time to Infinity Generalized
sqr(fn)	Escape-Time Fractals	Escape Time to Infinity Generalized
test	Escape-Time Fractals	Escape Time to Infinity
tetrate	Escape-Time Fractals	Escape Time to Infinity
tim's_error	Escape-Time Fractals	Escape Time to Infinity
unity	Escape-Time Fractals	Escape Time to Finite Attractor

Table 6-1 Fractint's fractal types (*continued*)

The term "escape" was first applied to the escape-time algorithm because the escape criterion was a test for the orbit value exceeding a "threshold of no return." For example, the Mandelbrot set is generated by iterating the formula $z' = z^2 + c$, and testing whether $|z| > 2$. It is possible to prove that once $|z| > 2$, further iterations of the Mandelbrot formula will result in larger and larger orbit values diverging to infinity. The orbit values are like a rocket that has reached escape velocity of the solar system; the rocket will then head for the reaches of infinite space, never to return. When the escape criterion is a test for the magnitude of the orbit value exceeding a threshold value, we call the fractal an "escape-time-to-infinity" fractal.

The idea of escape time works just as well when the orbit "escapes" to some finite point rather than infinity. You might imagine our intrepid space explorers wandering too near a black hole and being swallowed up. The "escape" criterion would be the rocket's venturing inside the famous Schwarzschild radius. Once inside, there is no return and, indeed, the rocket is no longer even visible to the outside universe. Perhaps "capture time" is more apt than "escape time," especially if you don't like the idea of your space ship being swallowed by a black hole! Nevertheless, we'll call these fractals "escape-to-finite-attractor" fractals.

Escape-time fractals form the bulk of Fractint's types. We have divided them into Mandelbrot/Julia pairs, Mandelbrot/Julia Generalized, Escape Time to Infinity, Escape Time to Infinity Generalized, 4-D Escape Time, Escape Time to Finite Attractor, and User-Defined Escape Time.

Mandelbrot/Julia Pairs

In a special sense,the classic Mandelbrot set is a catalog of all the Julia sets. Both share the formula $z' = z^2 + c$, but they use the formula in different ways. In the Mandelbrot calculation, the variable c is mapped to the pixels on your computer screen. On the other hand, each Julia set is formed using a fixed c, and the initial values z that begin the iteration process are mapped to your screen. Therefore, a pixel in a Mandelbrot image represents a particular value of c that can in turn be used to generate a whole Julia image.

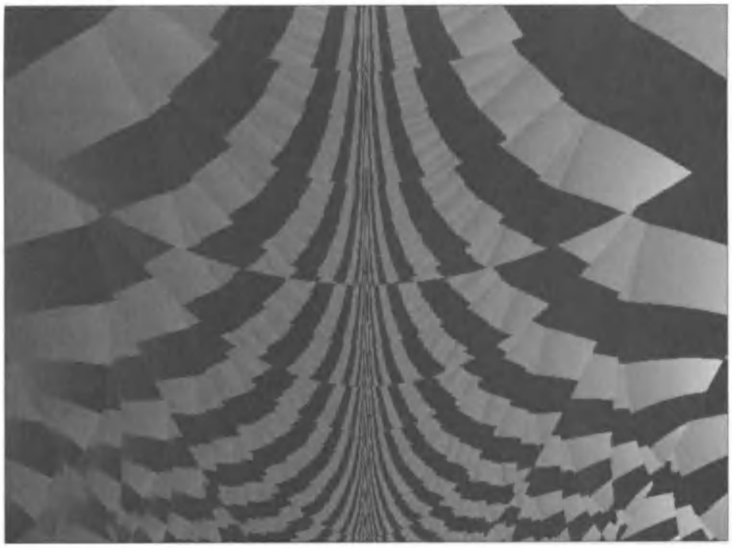
From the beginning of its development, Fractint has used the SPACEBAR key to let you explore this Mandelbrot/Julia relationship. Now in Fractint version 18, this capability has been greatly enhanced. Press T to get the SELECT A FRACTAL TYPE screen, and select type mandel. Generate an image by pressing a function key to select a video mode (such as SHIFT-F5 for the 640 x 480 x 256 SVGA video mode) or by pressing ENTER if you have already selected a video mode. When the image is complete, press SPACEBAR. A cross hair will appear on the screen, and a window will open in the lower right-hand side of your screen. This window contains the outline of the Julia set corresponding to the point on the Mandelbrot set where the cursor is pointing. You can move the cursor around the screen using the mouse or the arrow keys and instantly see how the Julia set changes. But there's more fun—when you have found an interesting Julia outline, press SPACEBAR again, and a complete full-screen Julia image will be generated. Pressing SPACEBAR again takes you back to the Mandelbrot image. (Fractint doesn't store the image but will regenerate it—fortunately, this calulation is very quick.) Finally, press SPACEBAR one more time and the cross-hair cursor will reappear at its previous location.

When the Julia is on the screen, pressing J takes you back and forth between the classic escape-time Julia set and the Julia_inverse type set which generates the Julia outline that you saw in the small window below the Mandelbrot display. (The Julia_inverse type is documented later in this chapter in the Orbits Fractals section.)

So far we have discussed the classic Mandelbrot and Julia sets, which are generated using Fractint's mandel and julia fractal types. Dr. Michael Barnsley, in his book *Fractals Everywhere* (Academic Press, 1988), points out that fractals created using other formulas have this same "Mandelbrot/Julia" relationship. Suppose a fractal is generated with the formula $z = f(z) + c$, where $f(z)$ is some function of z, say $z^2 + \sin(z)$. Then the "Mandelbrot" in the general sense would be created by assigning different values of c for each screen pixel, and the "Julia" set could be created by fixing c and initializing z for each screen pixel, just as with the classic Mandelbrot and Julia sets. For the rest of this chapter, we will adopt Dr. Barnsley's terminology, and use "Mandelbrot" and "Julia" to refer to these more general types, and not just types mandel and julia.

You can also explore these general Mandelbrot/Julia pairs using the SPACEBAR key. The only difference is that the window showing the Julia outline has not been implemented for any types except mandel and julia. To try this, press T and select any of the Mandelbrot types in this section. (For example, try Fractal type barnsleym1.) Press SPACEBAR, and you will see the cross-hair cursor, but no Julia window. (If you do not see a cross-hair cursor, then the type you selected is not a Mandelbrot type.) Pressing SPACEBAR again takes you to the Julia variant corresponding to the point on the Mandelbrot where the cursor was pointing.

Barnsleym1



Category: Mandelbrot/Julia Pair

This type is the Mandelbrot variant corresponding to fractal type barnsleyj1. The formula is taken from Dr. Michael Barnsley's book *Fractals Everywhere.*

Example:

```
Hot_Silk!          { ; Fractal Flag
                     ; (c)1992 Peter Moreland 100012,3213
  reset type=barnsleym1 passes=t
  corners=-1.907569/-1.35147/0.369773/-0.37169/-1.35147/-0.37169
  bailout=12123 decomp=256 periodicity=-256
  colors=000g0J<56>10y10y00z00z11y<77>vv4ww3xx2yy2yy1zz0<83>z10z00z00y01<22>h0\
  I
  }
```

Formula:

Initialize: $c = z = zpixel$

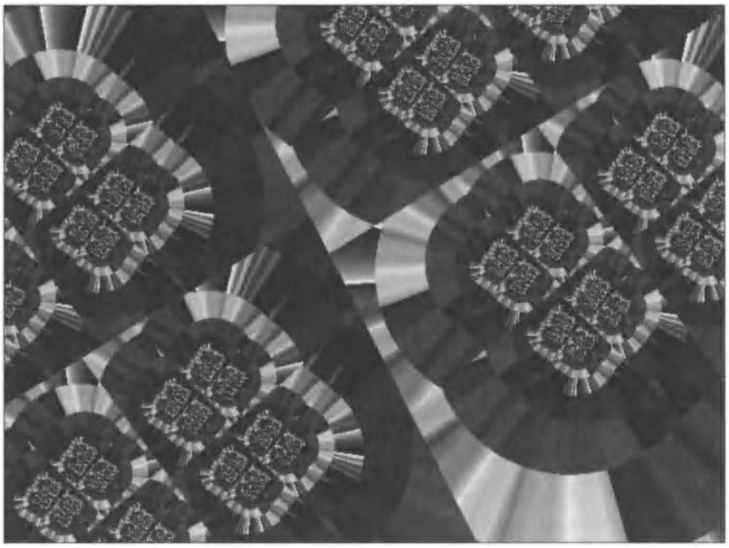
Iterate: $(z - 1)c$ if $x_z >= 0$
$(z + 1)c$ if $x_z <= 0$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_mandel_per_pixel,()	FRACTALS.C
Integer math orbit	Barnsley1Fractal()	FRACTALS.C

Code:	Routine Type	Routine Name	File
	Floating point initialization	othermandelfp_per_pixel()	FRACTALS.C
	Floating point orbit	Barnsley1FPFractal()	FRACTALS.C

Barnsleyj1



Category: Mandelbrot/Julia Pair

This type is the Julia variant corresponding to fractal type barnsleym1. The formula is taken from Dr. Michael Barnsley's book *Fractals Everywhere*.

Example:

```
Mirror, Mirror!    { ; (c) Peter Moreland 100012,3213
  reset type=barnsleyj1 corners=-1.3564493/-1.2725469/0.717811/0.7807173
  params=0.1414/1.79 float=y maxiter=32000 bailout=100 decomp=256
  colors=00000e0e00eee00e0eeL0eeeLLLLLzLzLLzzzLLzLzzzLzzz000555<3>HHHKKKOOOSSS\
  WWW___ccchhhmmmssszzz00z<3>z0z<3>z00<3>zz0<3>0z0<3>0zz<2>0GzVVz<3>zVz<3>zVV<\
  3>zzV<3>VzV<3>Vzz<2>Vbzhhz<3>zhz<3>zhh<3>zzh<3>hzh<3>hzz<2>hlz00S<3>S0S<3>S0\
  0<3>SS0<3>0S0<3>0SS<2>07SEES<3>SES<3>SEE<3>SSE<3>ESE<3>ESS<2>EHSKKS<2>QKSSKS\
  SKQSKOSKMSKK<2>SQKSSKQSKOSKMSKKSK<2>KSQKSSKQSKOSKMS00G<3>G0G<3>G00<3>GG0<3>0\
  G0<3>0GG<2>04G88G<2>E8GG8GG8EG8CG8AG88<2>GE8GG8EG8CG8AG88G8<2>8GE8GG8EG8CG8A\
  GBBG<2>FBGGBGGBFGBDGBCGBB<2>GFBGGBFGBDGBCGBBGB<2>BGFBGGBFGBDGBCG000<6>000
  }
```

Formula:

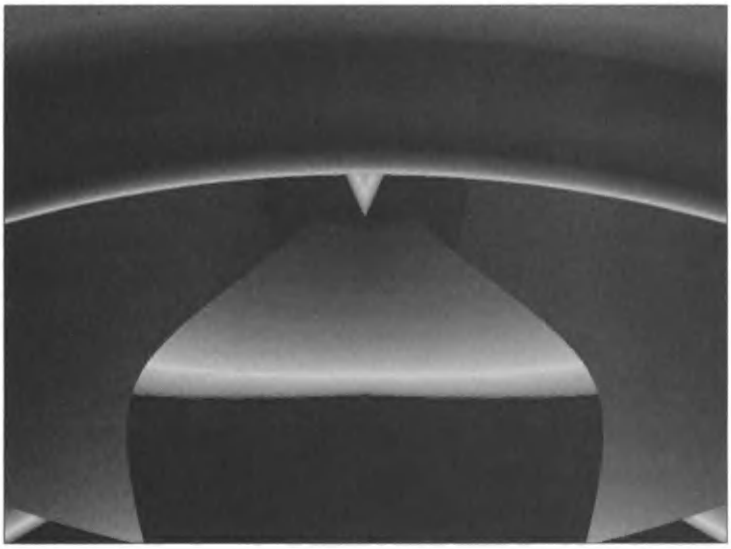
Initialize: $z = zpixel$

Iterate:
$$\begin{array}{ll} (z-1)c, & \text{if } x_z >= 0 \\ (z+1)/c, & \text{if } x_z < 0 \end{array}$$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_julia_per_pixel()	FRACTALS.C
Integer math orbit	Barnsley1Fractal()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	Barnsley1FPFractal()	FRACTALS.C

barnsleym2



Category: Mandelbrot/Julia Pair

This type is the Mandelbrot variant corresponding to fractal type barnsleyj2. The formula is taken from Dr. Michael Barnsley's book *Fractals Everywhere.*

Example:

```
rhs249            { ; (c) Richard Sherry Aug 8, 1992 76264,752
  reset type=barnsleym2 corners=-0.271995/0.272007/0.75088/1.158889
  params=1/-0.06 maxiter=500 inside=0 potential=255/1000/1
  colors=000"0<28>FF000F<62>00z00z00y<61>00FFF0<61>zz0zz0yy0<31>aa0
  }
```

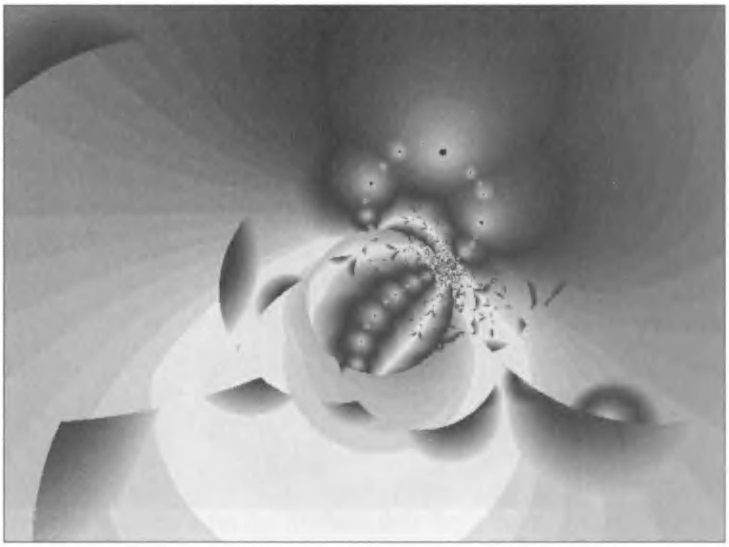
Formula:

Initialize: $c = z = zpixel$

Iterate: $(z-1)c \quad \text{if } x_z y_c + x_c y_z >= 0$
$(z+1)c \quad \text{if } x_z y_c + x_c y_z < 0$

Code:	Routine Type	Routine Name	File
	Fractal engine	StandardFractactal()	CALCFRACT.C
	Integer math initialization	long_mandel_per_pixel,()	FRACTALS.C
	Integer math orbit	Barnsley2Fractal()	FRACTALS.C
	Floating point initialization	othermandelfp_per_pixel()	FRACTALS.C
	Floating point orbit	Barnsley2FPFractal()	FRACTALS.C

barnsleyj2



Category: Mandelbrot/Julia Pair

This type is the Julia variant corresponding to fractal type barnsleym2. The formula is taken from Dr. Michael Barnsley's book *Fractals Everywhere.*

Example:

```
food_chain         { ; Jonah's View?
                     ; BG Dodson 1993 71636,1075
  reset type=barnsleyj2 corners=-31.99/20.4736/-20.174345/19.173355
  params=0.6/1.1 maxiter=20 bailout=22 inside=bof60 outside=summ
  invert=0.5/0.0533337/0
  colors=000hh0<14>330000003<14>00r<15>000<15>r00<15>000<15>0pp<14>044000222<1\
  4>hhh<14>333000000<26>007017037<13>4S4<13>2A2292171151131000<15>rAr<15>000<1\
  4>ee0
  }
```

Formula:

Initialize: $z = zpixel$

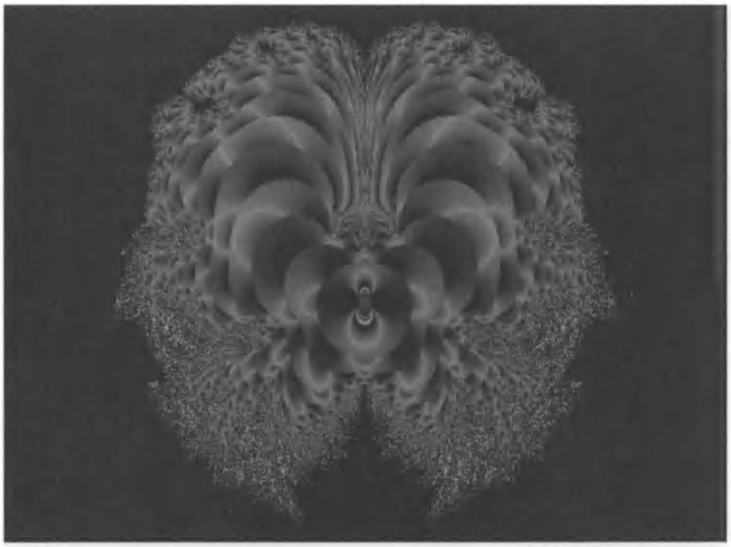
Iterate: $(z - 1)\,c$ $\quad$ $if\ x_z y_c + x_c y_z >= 0$

$(z + 1)\,c$ $\quad$ $if\ x_z y_c + x_c y_z < 0$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_julia_per_pixel()	FRACTALS.C
Integer math orbit	Barnsley2Fractal()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	Barnsley2FPFractal()	FRACTALS.C

barnsleym3



Category: Mandelbrot/Julia Pair

This type is the Mandelbrot variant corresponding to fractal type barnsleyj3. The formula is taken from Dr. Michael Barnsley's book *Fractals Everywhere*.

Example:

```
AeolisAndJanus      { ; 00:06:40.46
                      ; ... from JMS01:PigSnoutFace
  reset type=barnsleym3 corners=3.320626/-2.679374/-4.4/4.4/-2.679374/4.4
  maxiter=32767 bailout=16 outside=real logmap=yes invert=1/0/0
colors=000840<13>zX0<15>000<15>ut0<15>000GA4<12>2v10z00w0<14>000<15>00z<14>0\
00<16>z0X<15>000<15>p0w<15>000<15>zzz<15>000L00<13>z00<2>p00l00h00d00a00<9>0\
00420
}
```

Formula:

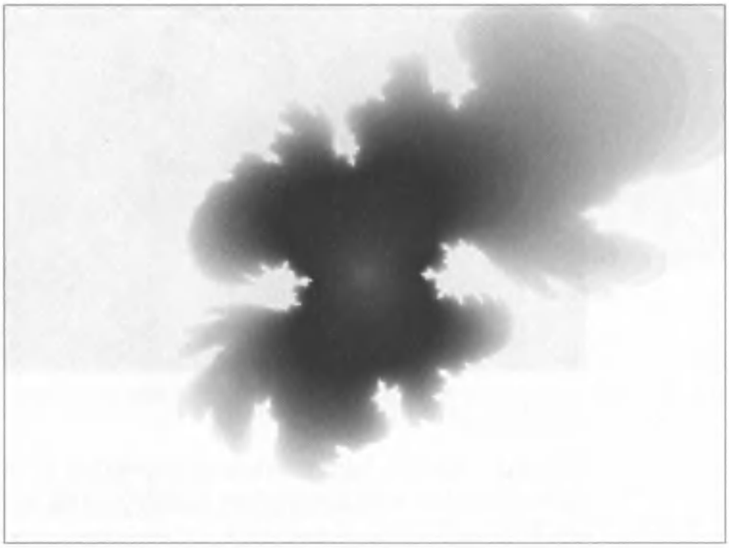
Initialize: $c = z = zpixel$

Iterate:
$$x_z^2 - y_z^2 - 1 + i2x_z y_z \quad \text{if } x_z > 0$$
$$x_z^2 - y_z^2 - 1 + x_c x_z + i(2x_z y_z + y_c x_z) \quad \text{if } x_z <= 0$$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_mandel_per_pixel,()	FRACTALS.C
Integer math orbit	Barnsley3Fractal()	FRACTALS.C
Floating point initialization	othermandelfp_per_pixel()	FRACTALS.C
Floating point orbit	Barnsley3FPFractal()	FRACTALS.C

barnsleyj3



Category: Mandelbrot/Julia Pair

This type is the Julia variant corresponding to fractal type barnsleym3. The formula is taken from Dr. Michael Barnsley's book *Fractals Everywhere*.

Example:

```
Fractal_Angel???    { ; Most bizzare.
                      ; BG Dodson 71636,1075
  reset type=barnsleyj3 corners=-31.99/31.989999/-23.992556/23.992536
  params=0.1/0.36 maxiter=32000 inside=bof61 potential=255/200/32000
  invert=7.99752/0/0
  colors=000zg3<44>v31v21u00<4>t00t00t00s00s00s00<34>g00g00f00e00<5>d00c00c00b\
  00b00<82>644644544444<57>yyyzzzzzzzzzzzzzzz
  }
```

Formula: Initialize: $z = zpixel$

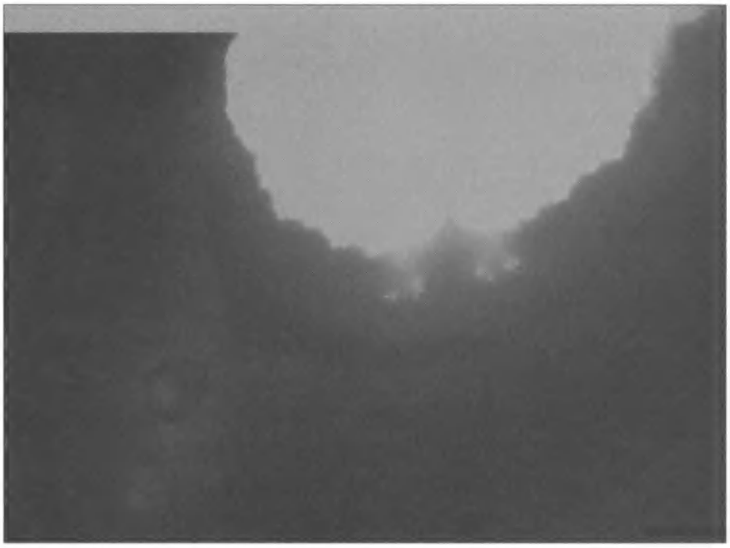
Iterate: $x_z^2 - y_z^2 - 1 + i2x_zy_z,$ if $x_z > 0$

$x_z^2 - y_z^2 - 1 + x_cx_z + i(2x_zy_z + y_cx_z),$ if $x_z <= 0$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_julia_per_pixel()	FRACTALS.C
Integer math orbit	Barnsley3Fractal()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	Barnsley3FPFractal()	FRACTALS.C

Cmplxmarksmand



Category: Mandelbrot/Julia Pair

This type is the Mandelbrot variant corresponding to fractal type cmplxmarksjul. The formula is from Mark Peterson. Note the exponent. Complex number exponents often introduce interesting discontinuities in fractals. If the exponent p is real, this type reduces to type marksmand.

Example:

```
YosemiteSunrise     { ; 00:01:05.63
                      ; 26 dec 92 .. caren park
  reset type=cmplxmarksmand passes=t
  corners=-0.761978/0.934022/-1.207056/0.064944 params=0.4/1/5/2 float=y
  maxiter=500 inside=maxiter potential=255/50/500 periodicity=-1
  colors=000zj0<6>zx0zz0zz1<29>zzxzzzzzz<61>zV1zU0zU0zT0<28>z10z00z00y00<30>c0\
  0b11a11'22_22<23>GEEFFFFFFFFF<29>x11z00z10<21>zh0
  }
```

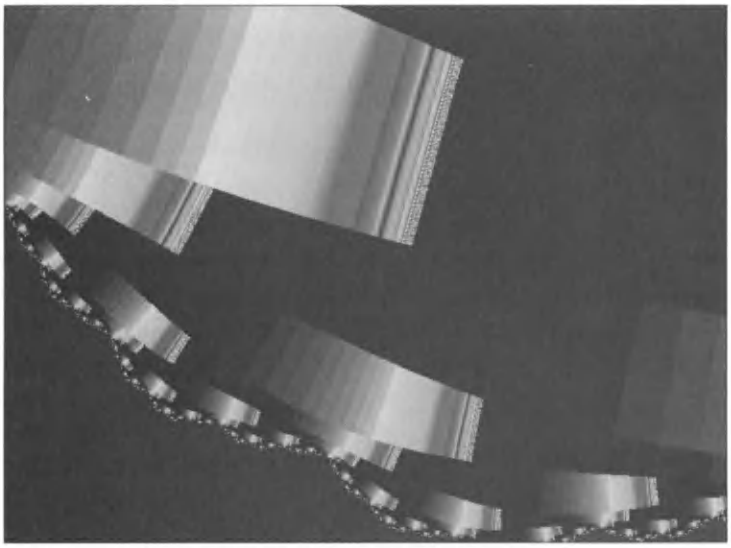
Formula: Initialize: $c = z = zpixel$

Iterate: $z^2 c^{(p-1)} + c$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Floating point initialization	MarksCplxMandperp	FRACTALS.C
Floating point orbit	MarksCplxMand()	FRACTALS.C

Cmplxmarksjul



Category: Mandelbrot/Julia Pair

This type is the Julia variant corresponding to fractal type cmplxmarksmand. If the exponent p is real, this type reduces to type marksmand.

Example:

```
Keyboard_madness   { ; An ergonomic nightmare.....  (c)1992 Peter Moreland
  reset type=cmplxmarksjul
  corners=9.592494/8.9412748/-1.074353/-0.3477897/9.0093047/-0.2967673
  params=1/3/1.1/2 float=y maxiter=30000 bailout=250 inside=256
  outside=mult
  colors=000554<8>'mF<2>diDfgChfBjdBkbA<3>rX6<4>Y6h<5>b5kb5lc5lc5md5ne4nf4oH6z\
  CDc7LH<8>FO'GPcHPeIQhJQjKRm<5>Udr<9>600<8>IYSKaTKaT<21>bcLbcLbbM<8>YXU<3>DHb\
  <3>vSj<5>bP9dFLf4X<5>RcuReuShv<30>UenUemUemUelUel<21>KfS_99<15>x11z00z10<29>\
  zx0zz0zz1<7>103
  }
```

Formula:

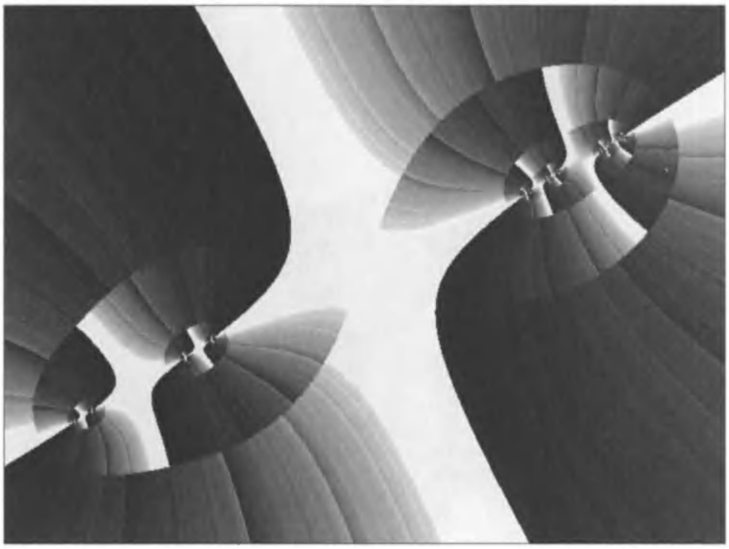
Initialize: $z = zpixel$

Iterate: $z^2c^{(p-1)} + c$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Floating point initialization	juliafp_per_pixel()	FRACTALS.C
Floating point orbit	MarksCplxMand()	FRACTALS.C

Mandel



Category: Mandelbrot/Julia Pair

This type is the Mandelbrot variant corresponding to fractal type julia. This is it—the one and only Mandelbrot set! Discovered by Benoit Mandelbrot.

Example:

```
The_Great_Divide   { ; Bridge that gap with....(C)1992 Peter Moreland1000123213
  reset type=mandel
  corners=-7.079179/-3.838073/-3.277214/4.310403/-9.554437/0.02313
  params=5/-0.555 float=y maxiter=30333 bailout=500 fillcolor=9
  inside=256 decomp=256 periodicity=-1
  colors=000311522343642334534356654931D31962D52964D74784A83D92FC2A95D95BC\
  5ED54799795ACA9ADA9AD9ED9AADDBDADEEDDG31H62L72H74L75P75HA3L93ID3LF3H95LA\
  5HD6LD6QC6HB8LB8ID9ME9HACHEDLEDPB8SB9PE9TE9QFCUFCFG6EHBJG3MG3IG6MH5NK6PI\
  3UH3RK3TL3PI4TH6QL5TM5U06IH9MHAIKAMLAIHEMIDILDMLEQHATHAPLAULAPIDTIDQLEUL\
  EMPESPC37J5EJCEI4ESAEQIEIPFJHE06GMEHKFPM6IR9KTCPUIHHLIHJLIMLHHILLILHKMML\
  LRLILPKTPKJLQQMQLPSRRRXE6XDAYL5YQ6dP6YKDdKCZPDeRCZMHdMIYPI'PIXTIaSJXQLaQ\
  LXTMaTLeRLYMRZTQfTQkTNlURUXJUXSYWEfXA_YMdXN'XTfYTecSmXUlfS4FW6IX8MX90ZJN\
  XRNXMQYSTZKRdRVcYUYfUYUX'UYe_Z_eYXiYXe'Xh'YeZ'iY'ea'ia'ac_gd'Z'ffafadhgf\
  gn'Zsa_nd'tcanbdsbdnffuffikinlitliZbkdbkafmfinaisnjlujkhlqimsllmplmnompo\
  mmmpqnpmoqppqunprtruupnptwrtqtxutuxtuxwutvyyvwuxzzzz<24>zzz
  }
```

Formula:

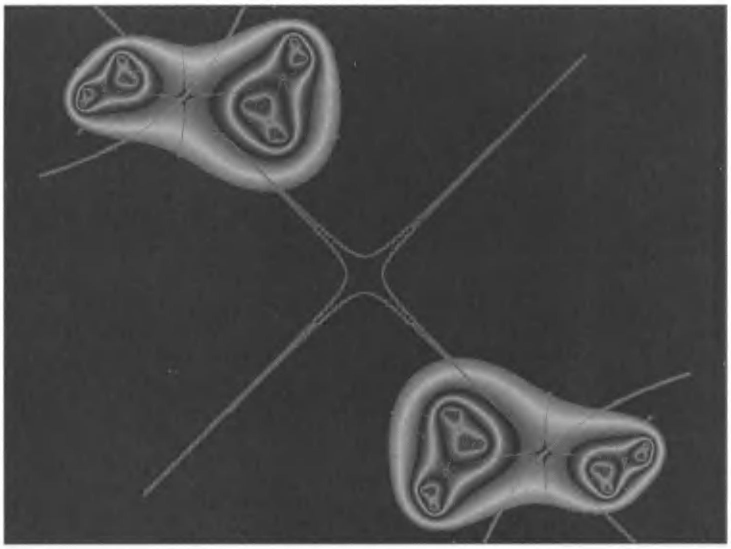
Initialize: $c = z = zpixel$

Iterate: $z' = z^2 + c$

Code:

Routine Type	Routine Name	File
C fractal engine	StandardFractactal()	CALCFRACT.C
ASM integer math fractal engine	calcmand()	CALCMAND.ASM
ASM floating point fractal engine	calcmandfp()	CALMANFP.ASM
Integer math initialization	mandel_per_pixel()	FRACTALS.C
Integer math orbit	MandelFractal()	FRACTALS.C
Floating point initialization	mandelfp_per_pixel()	FRACTALS.C
Floating point orbit	MandelfpFractal()	FRACTALS.C

Julia



Category: Mandelbrot/Julia Pair

This type is the julia variant corresponding to fractal type Mandelbrot.

Example:

```
sai003            { ; (c) 1993 Richard H. Sherry CIS:76264,752
  reset=1732 type=julia passes=b corners=-2.0/2.000012/-1.499989/1.5
  params=-7.15255737304688e-007/1.99 maxiter=256 fillcolor=241
  inside=-100 outside=summ logmap=yes potential=255/500/25
  periodicity=0
  colors=000eZM<16>xo'zpaypa<13>kcRibRibR<14>UPF111<29>ppp<31>000PFF<29>xe\
  DzfCyfC<30>PFF0A0<15>Zc0<14>0A0<15>Zc0<15>0A0SNC<11>dYM
  }
```

Formula: Initialize: $z = zpixel$

Iterate: $z^2 + c$

Code:

Routine Type	Routine Name	File
C fractal engine	StandardFractactal()	CALCFRACT.C
ASM integer math fractal engine	calcmand()	CALCMAND.ASM
ASM floating point fractal engine	calcmandfp()	CALMANFP.ASM
Integer math initialization	julia_per_pixel()	FRACTALS.C
Integer math orbit	JuliaFractal()	FRACTALS.C
Floating point initialization	juliafp_per_pixel()	FRACTALS.C
Floating point orbit	JuliafpFractal()	FRACTALS.C

Mandel4

Category: Mandelbrot/Julia Pair

This type is the Mandelbrot variant corresponding to fractal type julia4. It is a simple generalization of the Mandelbrot formula, using the fourth power instead of a square in the formula.

Example:

```
sac030             { ; (c) Richard H. Sherry 1993, CIS:76264,752
  reset=1731 type=mandel4
  corners=-1.259916/-1.109256/0.100439/-0.100434/-1.109256/-0.100434
maxiter=256 inside=0 logmap=yes potential=256/250/50 biomorph=0
periodicity=0
colors=000TTT<16>000PFF<52>uuE_NE<8>PFF0A0<15>Zc0<14>0A0<12>SY0000<17>00\
0SNC<3>WQF000<2>000aVJ<3>eZM000<2>000kcR<3>ogU000<2>000ulY<2>xo'000<3>00\
0ulZtkYsjX000<3>000meTldSkcRibR<5>dZN000<9>222333555<26>nnnpppooo<12>UUU
}
```

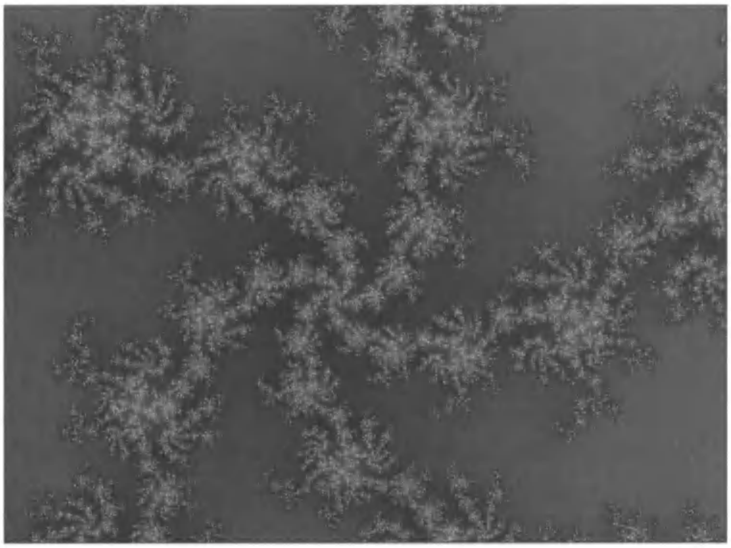
Formula: Initialize: $c = z = zpixel$

Iterate: $z' = z^4 + c$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	mandel_per_pixel()	FRACTALS.C
Floating point initialization	mandelfp_per_pixel()	FRACTALS.C
Integer math orbit	Mandel4Fractal()	FRACTALS.C
Floating point orbit	Mandel4fpFractal()	FRACTALS.C

Julia4



Category: Mandelbrot/Julia Pair

This type is the Julia variant corresponding to fractal type mandel4. It is a simple generalization of the Julia formula, using the fourth power instead of a square in the formula.

Example:

```
Julias Jewels      { ; These would be *very* nice to own!
  reset type=julia4 corners=0.43594/0.565764/-0.4706169/-0.3732543
  params=0.6/0.55
  colors=000exG<2>dyD<3>qd'<6>jdYmG8<3>pAA<16>3BNOCOOCO<36>MYsNZtNZtOZt<34>phq\
  qipphp<33>ABM
  }
```

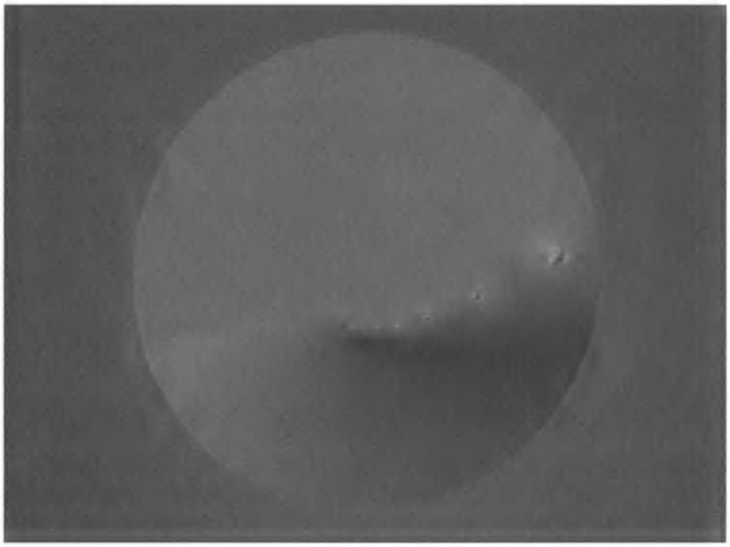
Formula: Initialize: $z = zpixel$

Iterate: $z' = z^4 + c$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	julia_per_pixel()	FRACTALS.C
Floating point initialization	juliafp_per_pixel()	FRACTALS.C
Integer math orbit	Mandel4Fractal()	FRACTALS.C
Floating point orbit	Mandel4fpFractal()	FRACTALS.C

Manzpower



Category: Mandelbrot/Julia Pair

This type is the Mandelbrot variant corresponding to fractal type julzpower. This type is a generalization of the Mandelbrot formula, using an exponent of z that can be a complex number. If $m = (2,0)$, this type reduces to the classic Mandelbrot.

Example:

```
TheyWentThat_A_Way { ; (c)1992 Peter Moreland 100012,3213
  reset type=manzpower
  corners=6.13366561/6.13069422/21.2647339/21.2103975/6.15867736/21.2313848
  params=11/6/4.555/12.666 float=y maxiter=32000 bailout=100 inside=255
  potential=255/128/0 decomp=256 biomorph=0
  colors=LSUcio<128>KQUJQUJQUJPUJPUIPU<11>FMUFLUFLUEKUEKUDJV<2>BLSAMR9MQ8MP6KM\
  <58>inujovinu<34>cio
  }
```

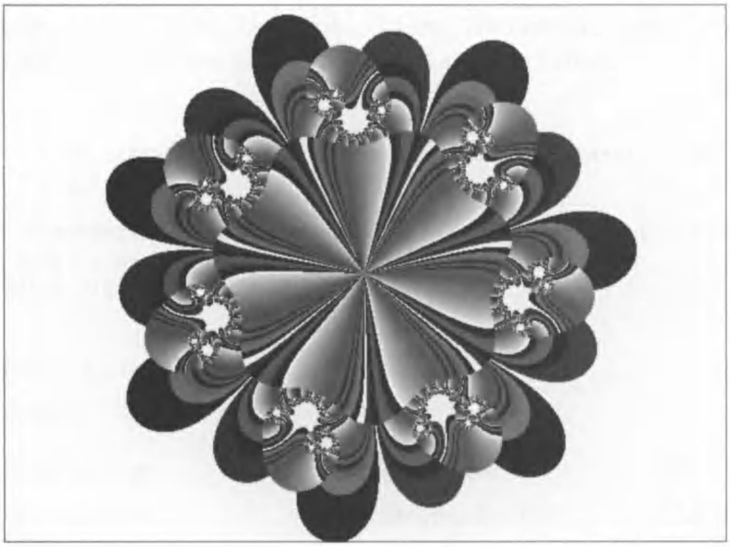
Formula: Initialize: $c = z = zpixel$

Iterate: $z' = z^m + c$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_mandel_per_pixel()	FRACTALS.C
Floating point initialization	othermandelfp_per_pixel()	FRACTALS.C
Integer math orbit	longZpowerFractal()	FRACTALS.C
Floating point orbit	floatZpowerFractal()	FRACTALS.C

Julzpower



Category: Mandelbrot/Julia Pair

This type is the Julia variant corresponding to fractal type manzpower. This type is a generalization of the Julia formula, using an exponent of z that can be a complex number. If $m = (2,0)$, this type reduces to the classic Julia.

Example:

```
decomp32          { ; An inverted decomposition julia using z=z^7+c
                            ; Paul Dickins
  reset type=julzpower corners=-1.834286/1.834286/-1.375714/1.375714
  params=-0.434234481334931/0.966090025277329/7 float=y maxiter=44
  invert=0.77/0/0 decomp=32
  colors=zzzzN00Xz00zNf000zNf0zX0000zzzD3z0000ee0e000e000zzz<9>zV_zRYz0VzKTzHQ\
  zDN<4>oBKmBJkBIiAHgAGe9G<2>Z8D
  }
```

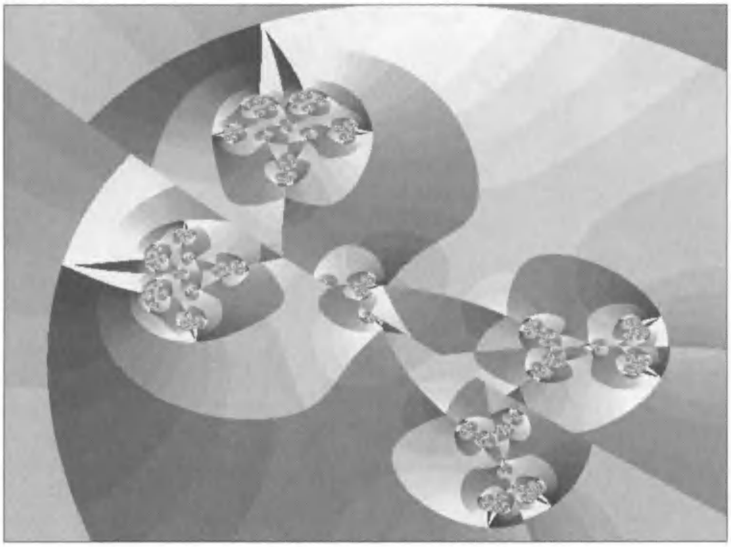
Formula: Initialize: $z = zpixel$

Iterate: $z' = z^m + c$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_julia_per_pixel()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Integer math orbit	longZpowerFractal()	FRACTALS.C
Floating point orbit	floatZpowerFractal()	FRACTALS.C

Manzzpwr



Category: Mandelbrot/Julia Pair

This type is the Mandelbrot variant corresponding to fractal type julzzpwr. This type was first explored by Clifford Pickover. The example is from the "face" series created by Dick Sherry, Peter Moreland, and Dan Farmer.

Example:

```
CutPaperI          { ; From "EVIL" By Dick Sherry, via SMOKIE by Peter Moreland
                     ; Dan Farmer
  reset type=manzzpwr
  corners=-1.1503476/-1.0993901/-0.002827848/-0.058661504/-1.0993901/-0.058550\
  457 params=0.1/0/2 float=y maxiter=32000 inside=0 decomp=255
  colors=000<126>00y00z00y<125>000
  }
```

Formula:

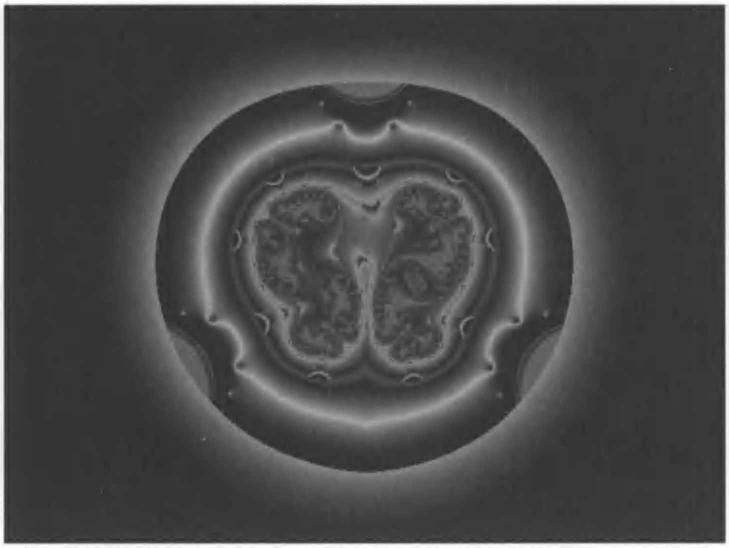
Initialize: $c = z = zpixel$

Iterate: $z' = z^z + z^m + c$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Floating point initialization	othermandelfp_per_pixel()	FRACTALS.C
Floating point orbit	floatZtozPluszpwrFractal()	FRACTALS.C

Julzzpwr



Category: Mandelbrot/Julia Pair

This type is the Julia variant corresponding to fractal type manzzpwr. This type was first explored by Clifford Pickover.

Example:

```
sac016              { ; (c) Richard H. Sherry 1993, CIS:76264,752
                      ; V17.32
  reset=1731 type=julzzpwr
  corners=2.55567/-2.76433/-3.546667/3.546667/-2.76433/3.546667
  params=0.010101/0.0707/3 float=y maxiter=256 inside=0 logmap=yes
  potential=256/200/75 periodicity=0
  colors=000Xb0<14>0A0SNC<6>ZTH000<54>000111333<28>ppp<31>000PFF<29>xeDzfC\
  yfC<30>PFF0A0<15>Zc0<14>0A0<15>Zc0
  }
```

Formula:

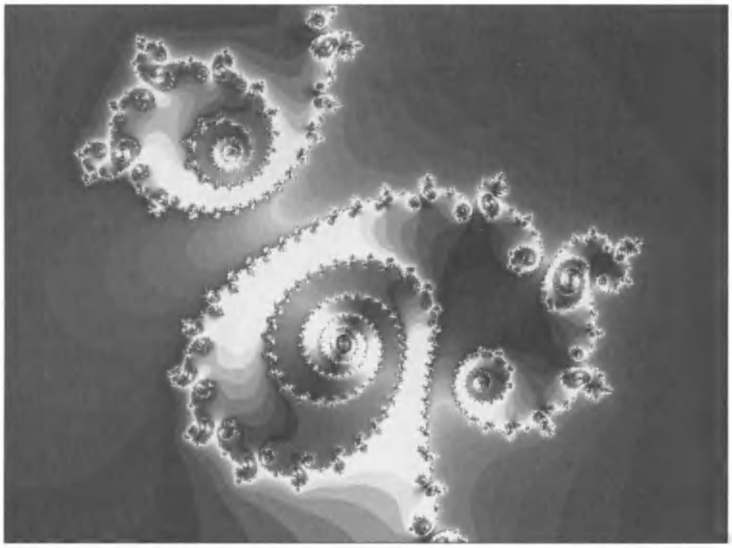
Initialize: $z = zpixel$

Iterate: $z' = z^z + z^m + c$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	floatZtozPluszpwrFractal()	FRACTALS.C

Mandellambda



Category: Mandelbrot/Julia Pair

This type is the Mandelbrot variant corresponding to fractal type lambda.

Example:

```
CoralAtolls          { ; 00:03:27.29
                       ;  26 nov 92 .. caren park
  reset type=mandellambda
  corners=-1.01561762/-1.00898082/0.10249492/0.10747252 params=0.1/0.2
  float=y maxiter=5000 bailout=8000 inside=maxiter periodicity=-1
  colors=000703<14>z0X<15>000<15>p0w<15>000<15>zzz<15>000L00<13>z00<2>p00l00h0\
  0d00a00<9>000<15>zX0<15>000<15>ut0<15>000GA4<12>2v10z00w0<14>000<15>00z<14>0\
  00301
  }
```

Formula:

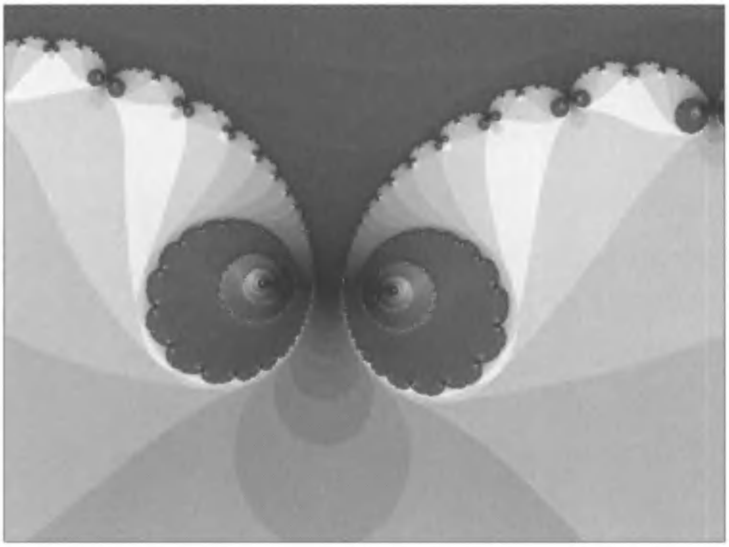
Initialize: $c = z = zpixel$

Iterate: $cz\,(1 - z)$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	mandel_per_pixel()	FRACTALS.C
Integer math orbit	LambdaFractal()	FRACTALS.C
Floating point initialization	mandelfp_per_pixel()	FRACTALS.C
Floating point orbit	LambdaFPFractal()	FRACTALS.C

Lambda



Category: Mandelbrot/Julia Pair

This type is the Julia variant corresponding to fractal type mandellambda.

Example:

```
Really?            { ; 00:01:27.27
                     ; 19 dec 92 .. caren park
  reset type=lambda corners=0.00437346/0.00645346/0.07547513/0.07701662
  params=1.00000000000005/0.2 float=y maxiter=500 inside=maxiter
  colors=000K0A<4>000<15>p0w<15>000<15>zzz<15>000L00<13>z00<2>p00l00h00d00a00<\
  9>000<15>zX0<15>000<15>ut0<15>000GA4<12>2v10z00w0<14>000<15>00z<14>000<16>z0\
  X<9>00C
  }
```

Formula:

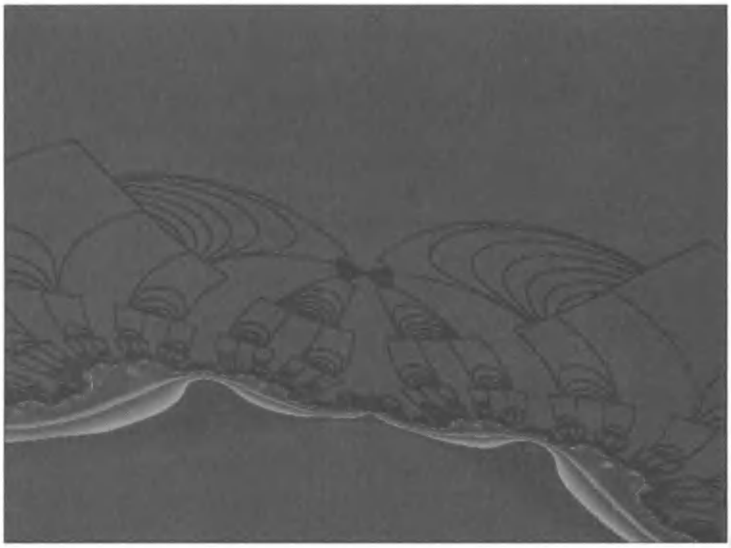
Initialize: $z = zpixel$

Iterate: $cz\,(1 - z)$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	julia_per_pixel()	FRACTALS.C
Integer math orbit	LambdaFractal()	FRACTALS.C
Floating point initialization	juliafp_per_pixel()	FRACTALS.C
Floating point orbit	LambdaFPFractal()	FRACTALS.C

Magnet1m



Category: Mandelbrot/Julia Pair

This type is the Mandelbrot variant corresponding to fractal type magnet1j. These fractals are based on formulas related to the study of magnetism.

Example:

```
The_Stargate       { ; The monolith should be here somewhere? (c)1992 Peter M.
  reset type=magnet1m passes=b
  corners=-15.976545/-12.5556/-0.799068/-3.116968/-11.998739/-3.812038
  params=1.76/4 maxiter=32000 bailout=500 fillcolor=9 inside=zmag
  potential=255/511/0 decomp=256
  colors=0009TU<35>9Im9ImAHnBHnCHnWGuDGo<19>WBu'ot<21>WCuv'I<21>XCtGSG<43>WBui\
  NG<16>WBsJyI<11>VEr4Yf<46>WBupuI<9>XFrJ5S<9>N7Z
  }
```

Formula:

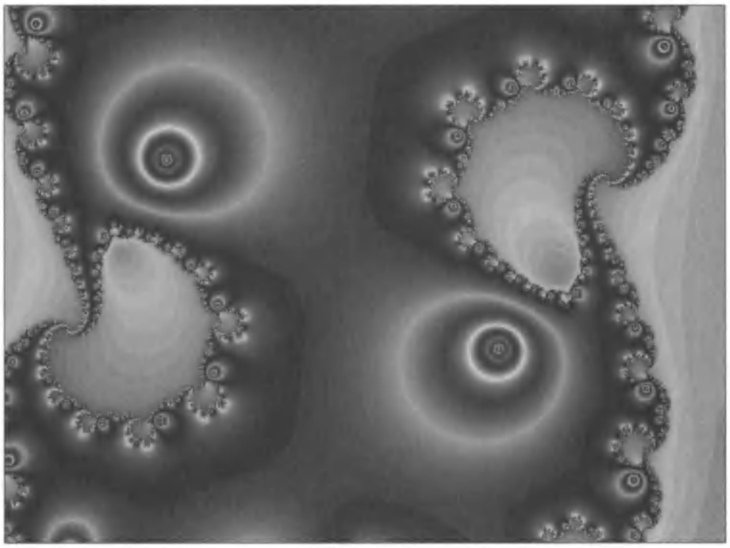
Initialize: $z = 0, c = zpixel$

Iterate: $z' = ((z^2 + (c - 1))/(2z + (c - 2)))^2$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Floating point initialization	mandelfp_per_pixel()	FRACTALS.C
Floating point orbit	Magnet1Fractal()	FRACTALS.C

Magnet1j



Category: Mandelbrot/Julia Pair

This type is the Julia variant corresponding to fractal type magnet1m.

Example:

```
OctopusBreast       { ; 00:30:23.96
                      ; 05 dec 92 .. caren park
  reset type=magnet1j corners=-4.218291/2.565709/-2.544/2.544
  params=-0.2/0.4 float=y maxiter=500 inside=maxiter periodicity=0
  colors=000dce<4>nrt<15>0I4<6>eYIg_KiaMkc0neQ<4>zp'<4>jdVgaUcZS'WRYUQ<6>AAJ<3\
  >ALPA0RARTBUVBXX<6>DqkDtmCqk<5>8Xb7T'9Q_<5>L4Q<8>VHH<5>rH4vH1zH0vH1<7>SDE<3>\
  N3IL0JL1JM2JN3J04I<10>tq2wu0zz0ww0<6>"8YY9XVA<6>KAIMAJ0AL<12>ziE<13>K8B4120\
  00<15>svbsvbsvbsvb<13>UMGSJEQII<5>EAa<6>7Nt5Pw5Nt<5>A5b<2>D6WheTG8R<2>KBK000\
  0EJ<7>b'b
  }
```

Formula:

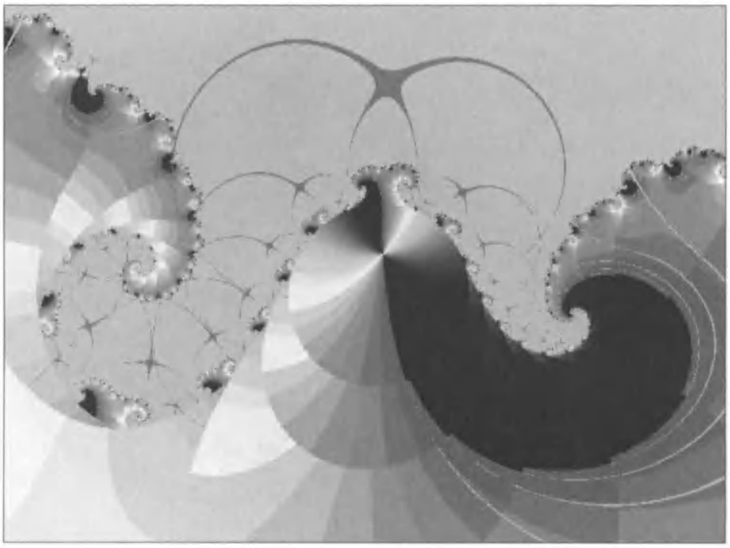
Initialize: $z = zpixel$

Iterate: $z' = ((z^2 + (c - 1))/(2z + (c - 2)))^2$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Floating point initialization	juliafp_per_pixel()	FRACTALS.C
Floating point orbit	Magnet1Fractal()	FRACTALS.C

Magnet2m



Category: Mandelbrot/Julia Pair

This type is the Mandelbrot variant corresponding to fractal type magnet2j. These fractals are based on formulas related to the study of magnetism.

Example:

```
BirdAndWaves        { ; 00:01:03.22
                      ; 14 dec 92 .. caren park
  reset type=magnet2m passes=t
  corners=-1.325065/-0.821314/-0.247782/-0.3197466/-1.109172/-0.5356402
  params=0.8/1 float=y maxiter=500 inside=-100 outside=real logmap=yes
  decomp=32 periodicity=-1
  colors=7Nu5Px<5>98fA5cB5'<3>G8SH9QIAQ<4>SKP<3>_VZ'Y'b'cdcf<4>nru<15>0I5<6>eY\
  Jg_LiaNkcPneR<4>zpa<4>jdWgaVcZT'WSYUR<6>AAK<3>ALQAOSARUBUWBXY<6>DqlDtnCql<5>\
  8Xc7Ta9Q'<5>L4R<8>VHI<7>zH0<8>SDF<3>N3JLOKL1KM2KN3K04J<12>zz0<7>"9YYAXVB<6>\
  KAJMAKOAM<12>ziF<13>K8C413000<15>svcsvcsvcsvc<4>jiWhfUfdTebS<4>YRKWPJUMHSJF<\
  6>EAb<5>8Lr cyclerange=0/255
  }
```

Formula:

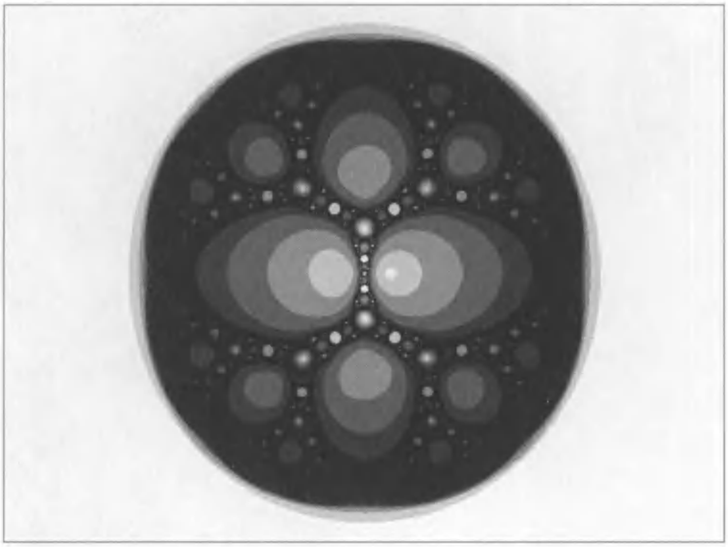
Initialize: $z = 0, c = zpixel$

Iterate: $z' = ((z^3 + 3(c-1)z + (c-1)(c-2))/(3z^2 + 3(c-2)z + (c-1)(c-2) + 1))^2$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Floating point initialization	mandelfp_per_pixel()	FRACTALS.C
Floating point orbit	Magnet2Fractal()	FRACTALS.C

Magnet2j



Category: Mandelbrot/Julia Pair

This type is the Julia variant corresponding to fractal type magnet2m.

Example:

```
Magnet2j         { ; Tim Wegner
reset=1822 type=magnet2j passes=1
corners=-14.01345/14.01345/-10.51009/10.51009
params=2.0131661626612556/0.0097989952144952319 float=y
periodicity=0
colors=000<5>eee<15>XXXWWWWWWWWW<6>SSSRRRRRRRRRQQQZZZ<4>YYYXXXXXXXWWWWWW<\
17>QQQQQQQQQQQQQQQQQQ<7>QQQeeeddddddbbbbbbbbb"'<15>QQQnnnmmmlllllllllkk\
kjjjjjj<4>gggffffffff<6>bbbQQQ<7>RRRRRRQQQRRRQQQRRR<14>QQQAAA<12>000LLL\
<7>NNNNNNNNN000000000<5>QQQBBB<2>DDDEEEEEEEEEE<3>GGGHHHHHHHHH<5>KKKLLLLLLL\
LLL<3>NNN000000000<3>QQQnnn<3>UUUIII<17>NNNNNNNNNNNN000000<5>QQQ
cyclerange=2/255
}
```

Formula:

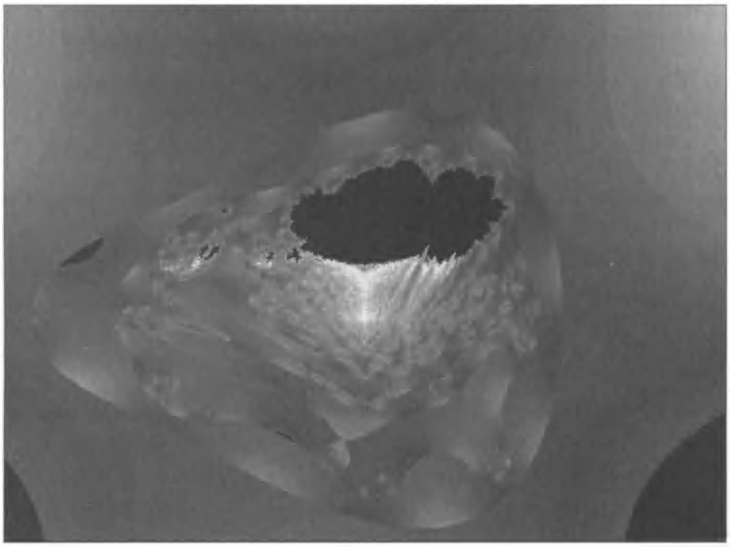
Initialize: $z = zpixel$

Iterate: $z' = ((z^3 + 3(c-1)z + (c-1)(c-2))/$
$(3z^2 + 3(c-2)z + (c-1)(c-2) + 1))^2$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Floating point initialization	juliafp_per_pixel()	FRACTALS.C
Floating point orbit	Magnet2Fractal()	FRACTALS.C

Mandphoenix



Category: Mandelbrot/Julia Pair

This type is the Mandelbrot variant corresponding to fractal type phoenix. The phoenix type defaults to the original phoenix curve discovered by Shigehiro Ushiki.

Example:

```
sac003            { ; (c) Richard H. Sherry 1993, CIS:76264,752
                    ; V17.32
  reset=1731 type=mandphoenix corners=-2.5/1.500012/-2.552616/0.447372
  params=0.5/1.3/-3 maxiter=256 inside=0 outside=imag logmap=yes
  periodicity=0
  colors=000CBCCABC8AD79<40>qe8qe8pd8<45>KAAKAALBA<41>sg8sg8rf8<40>K9B<15>\
  zz6<14>MFA96G<15>B_ZCa'Bb'Aca9da8da<21>CCD
  }
```

Formula:

Initialize: $c = zpixel$, $z = 0$, $w = 0$

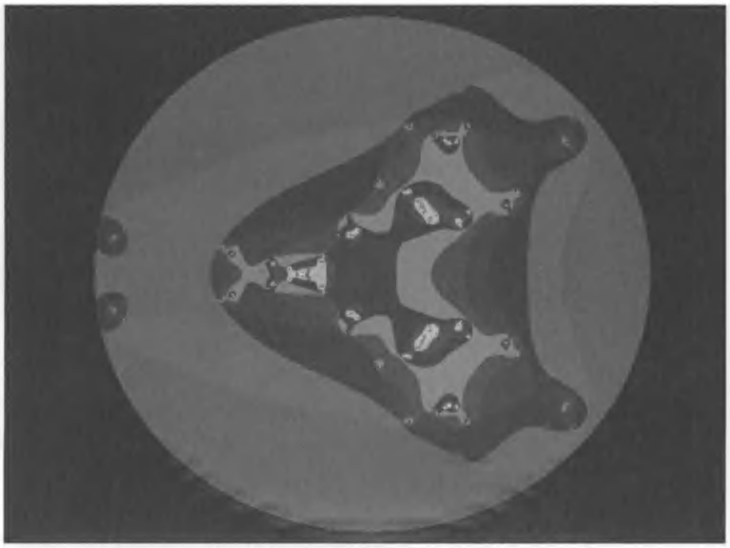
Iterate:
For degree of $z = 0$: $z' = z^2 + c_x + c_y w$, $w' = z$
For degree of $z >= 2$: $z' = z^{degree} + c_x z^{degree-1} + c_y w$, $w' = z$
For degree of $z <= -3$: $z' = z^{|degree|} + c_x z^{|degree|-2} + c_y w$, $w' = z$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_phoenix_per_pixel()	FRACTALS.C
Integer math orbit	LongPhoenixFractal()	FRACTALS.C

Code:	Routine Type	Routine Name	File
	Floating point initialization	phoenix_per_pixel()	FRACTALS.C
	Floating point orbit	PhoenixFractal()	FRACTALS.C
	Floating point orbit	Magnet2Fractal()	FRACTALS.C

Phoenix



Category: Mandelbrot/Julia Pair

This type is the Julia variant corresponding to fractal type mandphoenix. The phoenix type defaults to the original phoenix curve discovered by Shigehiro Ushiki.

Example:

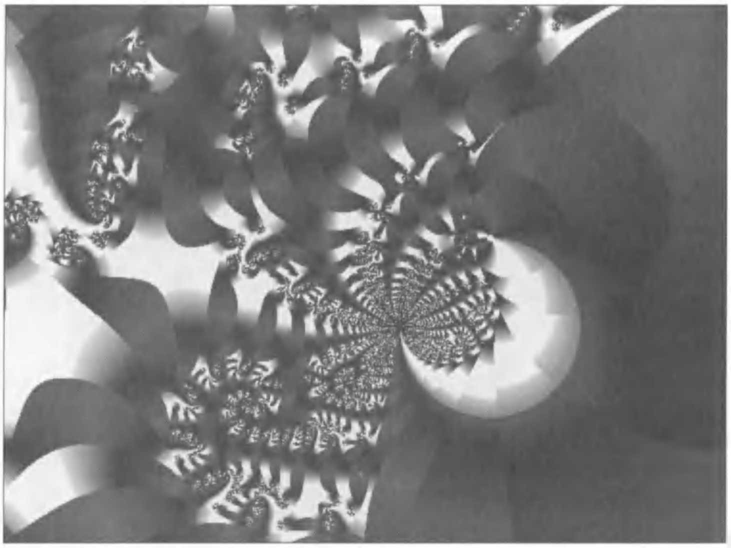
```
Wolf              { ; Jonathan Osuch
  reset type=phoenix corners=-1.557441/2.234555/-1.422003/1.421998
  params=-0.75078049362069/1.40200999374372/2 inside=0
  }
```

Formula: Initialize: $z = zpixel$, $w = 0$

Iterate:
For degree of $z = 0$: $z' = z^2 + c_x + c_y w$, $w' = z$
For degree of $z >= 2$: $z' = z^{degree} + c_x z^{degree-1} + c_y w$, $w' = z$
For degree of $z <= -3$: $z' = z^{|degree|} + c_x z^{|degree|-2} + c_y w$, $w' = z$

Code:	Routine Type	Routine Name	File
	Fractal engine	StandardFractactal()	CALCFRACT.C
	Integer math initialization	long_phoenix_per_pixel()	FRACTALS.C
	Integer math orbit	LongPhoenixFractal()	FRACTALS.C
	Floating point initialization	phoenix_per_pixel()	FRACTALS.C
	Floating point orbit	PhoenixFractal()	FRACTALS.C

Manowar



Category: Mandelbrot/Julia Pair

This type is the Mandelbrot variant corresponding to fractal type manowarj.

Example:

```
t16097            { ; Jon Horner
  reset type=manowar corners=-2.5/1.500012/-1.500014/1.5 inside=0
  invert=0.2/0.2/0.2 decomp=128
  colors=000J00<8>200000001<30>00z<30>002000100<30>wPF<30>211000101<29>_0_a0a'\
  0'<28>101000100<30>w00<20>L00
  }
```

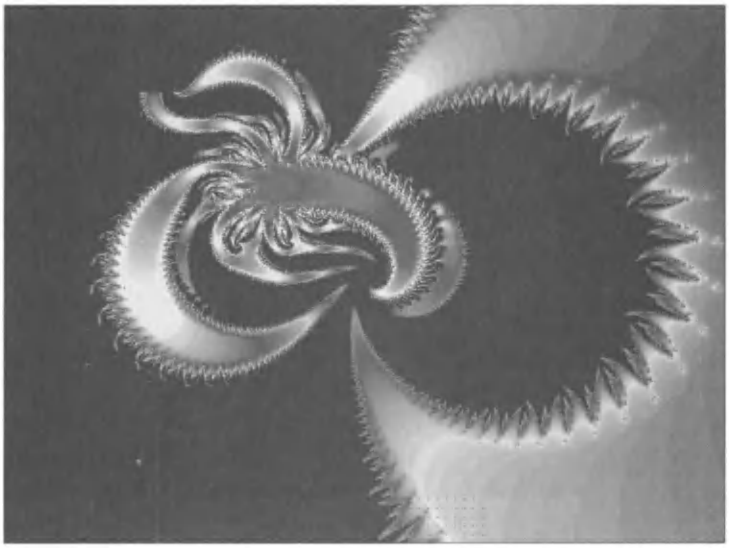
Formula: Initialize: $c = m = z = zpixel$

Iterate: $z' = z^2 + m + c$

$m' = z$

Code:	**Routine Type**	**Routine Name**	**File**
	Fractal engine	StandardFractactal()	CALCFRACT.C
	Integer math initialization	mandel_per_pixel()	FRACTALS.C
	Integer math orbit	ManOWarFractal()	FRACTALS.C
	Floating point initialization	mandelfp_per_pixel()	FRACTALS.C
	Floating point orbit	ManOWarfpFractal()	FRACTALS.C

Manowarj



Category: Mandelbrot/Julia Pair

This type is the Julia variant corresponding to fractal type manowar.

Example:

```
t16142              { ; Jon Horner
  reset type=manowarj corners=-5.803056/9.808064/-7.758382/3.949979
  params=0/0.01 inside=0 invert=0.3/0/0 periodicity=4
  colors=000KET<2>EAb<6>7Nu5Px5Nu<5>A5cB5'<6>KBLMCIOEK<3>WQUYSX_VZ'Y'<6>nru<4>\
  ggdfeadbYb'U<7>0I5<5>bVHeYJg_LiaN<6>zpa<4>jdWgaVcZT'WSYUR<6>AAK<4>AOSARUBUWB\
  XYB__Bba<5>Dtn<6>8Xc7Ta9Q'<5>L4R<8>VHI<7>zH0<7>WEDSDFRAG<2>N3JLOKL1KM2KN3K04\
  J<12>zz0<7>"9YYAXVB<6>KAJ<4>VBQXBS_CUbCV<6>zEe<15>000<15>svc<14>X0BVL9UKC<3\
  >MFP
  }
```

Formula: Initialize: $m = z = zpixel$

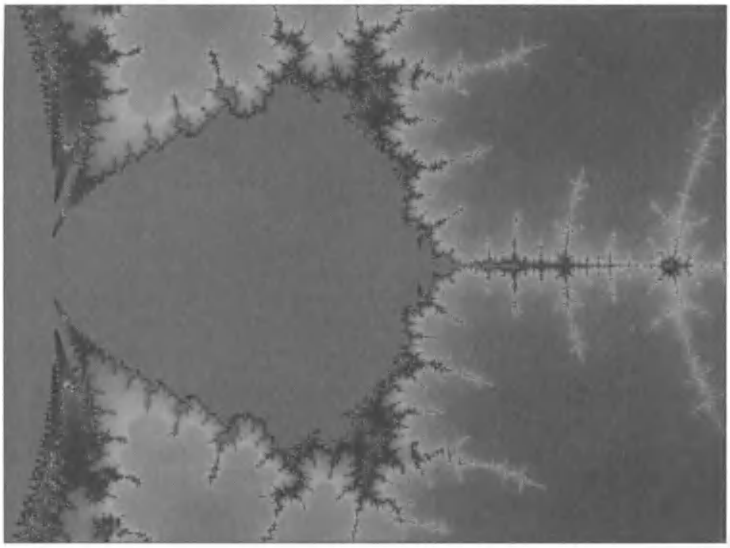
Iterate: $z' = z^2 + m + c;$

$m' = z;$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	julia_per_pixel()	FRACTALS.C
Integer math orbit	ManOWarFractal()	FRACTALS.C
Floating point initialization	juliafp_per_pixel()	FRACTALS.C
Floating point orbit	ManOWarfpFractal()	FRACTALS.C

Marksmandel



Category: Mandelbrot/Julia Pair

This type is the Mandelbrot variant corresponding to fractal type marksjulia. The formula was proposed by Mark Peterson.

Example:

```
Transmitter        { ; More fun, ready to zoom. Peter Moreland
  reset type=marksmandel corners=-1.236413/-1.466042/0.088695/-0.083512
  params=0.8/0/-1
  colors=0000aq<5>XQlYOkYKj<21>ejpekpfor<14>'Wd_UcX8BX8AW89Z54<26>5fjNOODYO1yq\
  3pn4gk5Zh5Sg<38>D7ZD6ZC4a<19>EHACG7EH9
  }
```

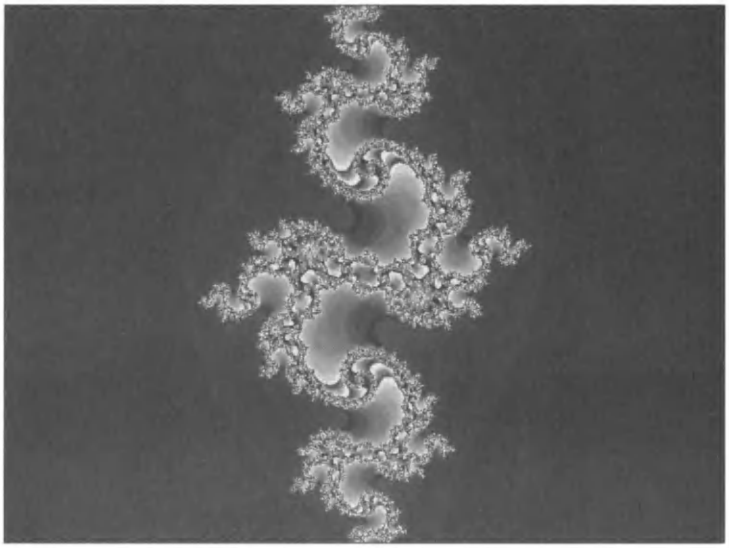
Formula: Initialize: $c = z = zpixel$

Iterate: $z^2c^{(p-1)} + c$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	mandel_per_pixel()	FRACTALS.C
Integer math orbit	MarksLambdaFractal()	FRACTALS.C

Marksjulia

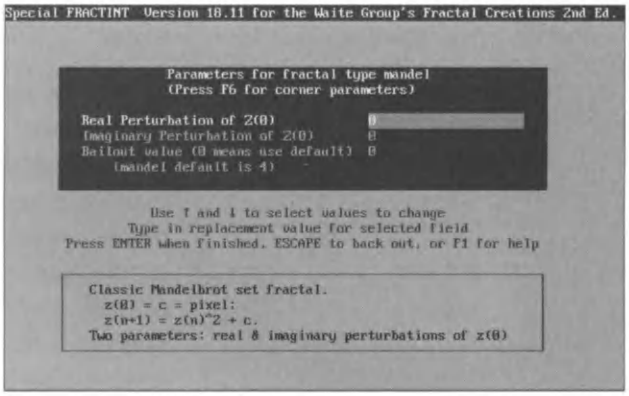


Category: Mandelbrot/Julia Pair

This type is the Julia variant corresponding to fractal type marksmandel. The formula was proposed by Mark Peterson.

Example:

```
SEASLUG          {
  reset=1611 type=marksjulia passes=1 corners=-2.0/2.0/-1.499983/1.5
  params=0.1/0.9 maxiter=1023 inside=0 symmetry=none
  colors=000F4E<11>7DdG66<8>1tHmsmIDB<3>MxbJ79<6>YobH25<18>DWLCYME6E<3>1Um\
  <18>fgzcWh_JQ<22>XpZWr_YDg_EkH26<21>GO_FPaH58<10>CvoI56<9>TrMMBA<4>kxaJ4\
  9<4>YKXG5B<4>7TeI35<16>lcGJ16<13>vARI26<20>r'fH25<20>QQ2H17G29G3B
  }
```



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

              Parameters for fractal type mandel
              (Press F6 for corner parameters)

      Real Perturbation of Z(0)                 0
      Imaginary Perturbation of Z(0)            0
      Bailout value (0 means use default)       0
          (mandel default is 4)

            Use ↑ and ↓ to select values to change
          Type in replacement value for selected field
   Press ENTER when finished, ESCAPE to back out, or F1 for help

    Classic Mandelbrot set fractal.
      z(0) = c = pixel:
      z(n+1) = z(n)^2 + c.
    Two parameters: real & imaginary perturbations of z(0)
```

Figure 6-1 A fractal type parameters screen

Formula:

Initialize: $z = zpixel$

Iterate: $z^2c^{(p-1)} + c$

Code:

Routine Type	**Routine Name**	**File**
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	julia_per_pixel()	FRACTALS.C
Integer math orbit	MarksLambdaFractal()	FRACTALS.C

Mandelbrot/Julia Generalized

Originally Fractint had only two fractal types: mandel and julia. Every time your authors saw an interesting fractal formula in a book or magazine, they coded it into Fractint, and the types proliferated. Then with the advent of Fractint's user-defined formula type (described later in this chapter), Fractint's clever users began inventing new fractal types in massive numbers. Formulas were proposed using permutations and combinations of functions, such as sin + sin + c, sin + cos + c, sin + log + c, and so forth. You get the idea! To defend Fractint from a 100-page-long Fractal Types screen, the programmers designed a way to build variable functions into types. This way hundreds of proposed fractal types can be combined to a single type.

To see how this works, consider type lambdafn. The iterated formula is $z' = c\text{fn}(z)$. The trick is that after selecting the type, you can use the PARAMETERS FOR FRACTAL TYPE screen to assign a particular function to fn. Figure 6-1 shows a typical parameter screen. You can use the arrow keys to move the highlight to the line labelled FIRST FUNCTION. Then use the ← or → key to cycle through the

Function Variable Value	Name	Calculation
conj	complex conjugate	$f(x + iy) = x - iy$
cos	complex cosine	$f(z) = \cos(z)$
cosh	complex hyperbolic cosine	$f(z) = \cosh(z)$
cosxx	conjugate of cosine	$f(z) = \text{conj}(\cos(z))$
cotan	complex cotangent	$f(z) = \text{cotan}(z)$
cotanh	complex hyperbolic cotangent	$f(z) = \text{cotanh}(z)$
exp	complex exponential	$f(z) = e^z$
flip	swap real and imaginary parts	$f(x + iy) = y + ix$
ident	identity function	$f(z) = z$
log	natural logarithm	$f(z) = \log_e(z)$
recip	reciprocal	$f(z) = 1/z$
sin	complex sine	$f(z) = \sin(z)$
sinh	complex hyperbolic sine	$f(z) = \sinh(z)$
sqr	square	$f(z) = z^2$
tan	complex tangent	$f(z) = \tan(z)$
tanh	complex hyperbolic tangent	$f(z) = \tanh(z)$
zero	contsant zero	$f(z) = 0$

Table 6-2 Function variable values

possible functions. There is also a speed key feature: if you repeatedly press a letter key, the function variable field will cycle through the functions beginning with that letter. For example, repeatedly pressing $\boxed{S}$ causes the FIRST FUNCTION field to cycle through sin, sinh, and sqr.

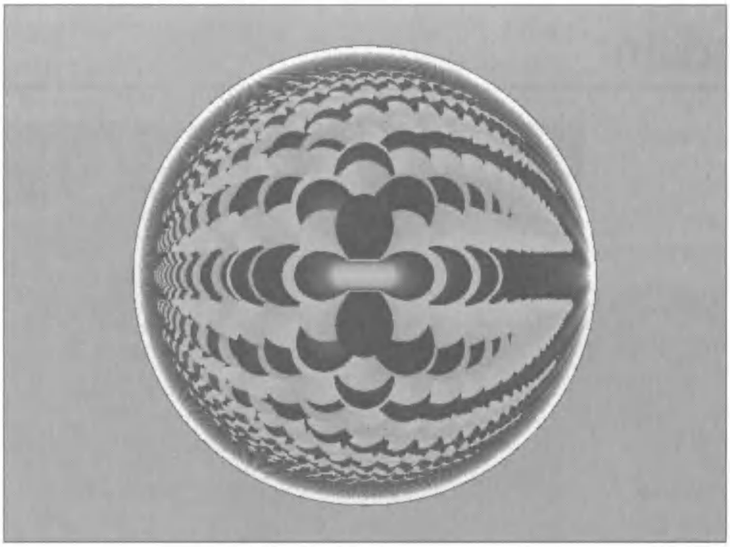
The possibilities for fn(z) include all of these functions: conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, and zero. Depending on which of these functions you select, the actual iterated formula will be $z' = c\text{conj}(z)$, $z' = c\cos(z)$, $z' = c\cosh(z)$, and so forth, with "fn" replaced by one of the functions to create a new formula. One fractal type does the work of 17 formulas. Actually, one formula can do the work of many *more* than 17 formulas, because your humble authors added the capability for four different function variables to be used in one fractal type. Therefore, the possible number of combinations is 17^4 or 83,521! However, as of yet no actual Fractint fractal type (other than the formula type) uses more than two of these function variables, so as a practical matter one fractal type can incorporate 289 different formulas.

Most of Fractint's variable functions are familiar to students of mathematics. The standard transcendental functions include cos, cosh, cotan, cotanh, exp, log, sin, sinh, tan, and tanh. (The log function is the natural logarithm, not the base ten logarithm.) Table 6-2 gives the definition of all of these functions.

Remember that you don't have to understand these functions to use them to make images! Just try them and see what happens.

The fractal types in this section all have escape-to-infinity algorithms, and have the same Mandelbrot/Julia relationship discussed in the previous section.

Mandelfn



Category: Mandelbrot/Julia Generalized

This type is the Mandelbrot variant corresponding to fractal type lambdafn, and is the generalization of type manlambda.

Example:

```
OrnateCBCLogo           { ; Caren Park 00:02:04.73
                              ; ... from JMS:PseudoSphere
    reset  type=mandelfn  function=ident
     corners=-1.552978/1.551022/-1.155134/1.172866   float=y   maxiter=32767
    outside=real  logmap=yes  invert=1/0/0  finattract=y
     colors=000002<122>00z000<125>z00000<2>001
  }
```

Formula:

Initialize: $c = z = zpixel$

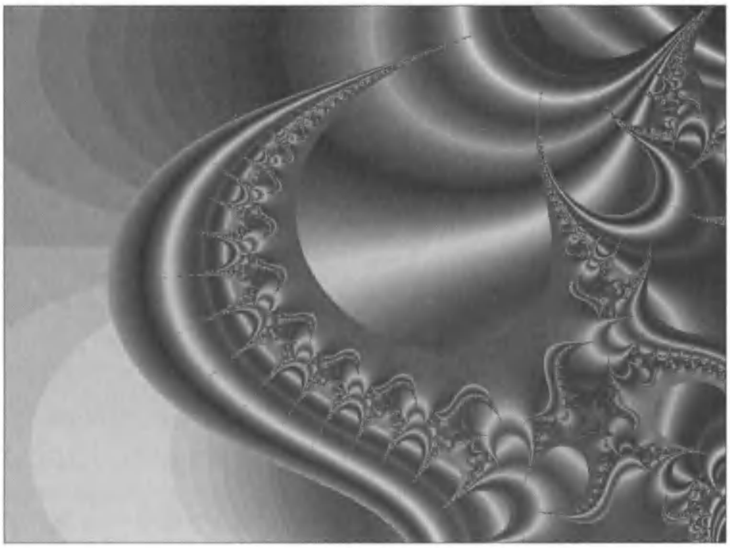
Iterate: $c\text{fn}(z)$,

where fn(z) is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	**Routine Name**	**File**
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_mandel_per_pixel()	FRACTALS.C
Integer math orbit	LambdaTrigFractal()	FRACTALS.C
Floating point initialization	othermandelfp_per_pixel()	FRACTALS.C
Floating point orbit	LambdafpFractal()	FRACTALS.C

Lambdafn



Category: Mandelbrot/Julia Generalized

This type is the Julia variant corresponding to fractal type mandelfn, and is the generalization of type lambda.

Example:

```
cet16109           { ; Jon Horner
  reset type=lambdafn function=exp
  corners=-5.711838/-0.014282/0.165024/4.436798 params=2/4 inside=0
  potential=255/511/0
colors=000332SDF<3>N3JLOKL1KM2KN3K04J<12>zz0<7>"9YYAXVB<6>KAJMAKOAM<12>ziF<\
13>K8C413000775<13>svcsvcsvcsvc<13>UMHSJFQIJ<5>EAb<6>7Nu5Px5Nu<5>A5cB5'<6>KB\
LMCIOEK<3>WQUYSX_VZ'Y'<6>nru<15>0I5<5>bVHeYJg_LiaN<6>zpa<4>jdWgaVcZT'WSYUR<6\
>AAK<4>AOSARUBUWBXYB__Bba<5>Dtn<6>8Xc7Ta9Q'<5>L4R<8>VHI<7>zHO<7>WED
cyclerange=2/255
}
```

Formula: Initialize: $z = zpixel$

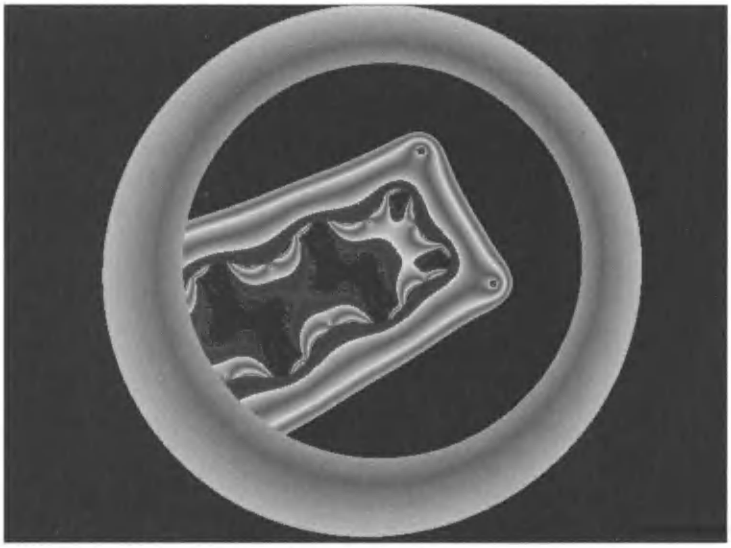
Iterate: $c\text{fn}(z)$,

where fn(z) is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	**Routine Name**	**File**
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_julia_per_pixel()	FRACTALS.C
Integer math orbit	LambdaTrigFractal()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	LambdafpFractal()	FRACTALS.C

Manfn+exp



Category: Mandelbrot/Julia Generalized

This type is the Mandelbrot variant corresponding to fractal type julfn+exp.

Example:

```
rhs131          { ; (c) June 1992 Dick Sherry 76264,752
                  ; no commercial use w/o permission
  reset type=manfn+exp function=cos float=yes
  corners=4.11792/4.546936/4.822723/5.145401 maxiter=500 inside=0
  potential=255/300/150 decomp=265 biomorph=0
  colors=000000000usp<7>d_6<8>zzz<18>'ZI_XG'YI<21>zzzKU5<10>7L15K06K0<11>KU400\
  0KU0000KU0000KU0000KU0000KU0000KU0000<154>000
  }
```

Formula:

Initialize: $c = z = zpixel$

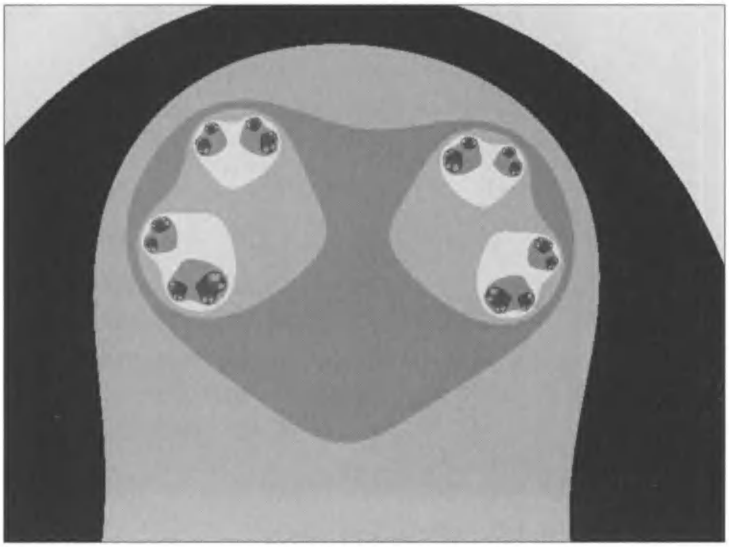
Iterate: $z' = \text{fn}(z) + e^z + c$

where fn(z) is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_mandel_per_pixel()	FRACTALS.C
Floating point initialization	othermandelfp_per_pixel()	FRACTALS.C
Integer math orbit	LongTrigPlusExponentFractal()	FRACTALS.C
Floating point orbit	FloatTrigPlusExponentFracta()	FRACTALS.C

Julfn+exp



Category: Mandelbrot/Julia Generalized

This type is the Julia variant corresponding to fractal type manfn+exp.

Example:

```
CrazyEyes          {
  reset=1733 type=julfn+exp function=sqr
  corners=-1.20961/0.6157227/-0.1073608/1.2621
  params=-0.57696533203125/-0.0302734375
  }
```

Formula:

Initialize: $z = zpixel$

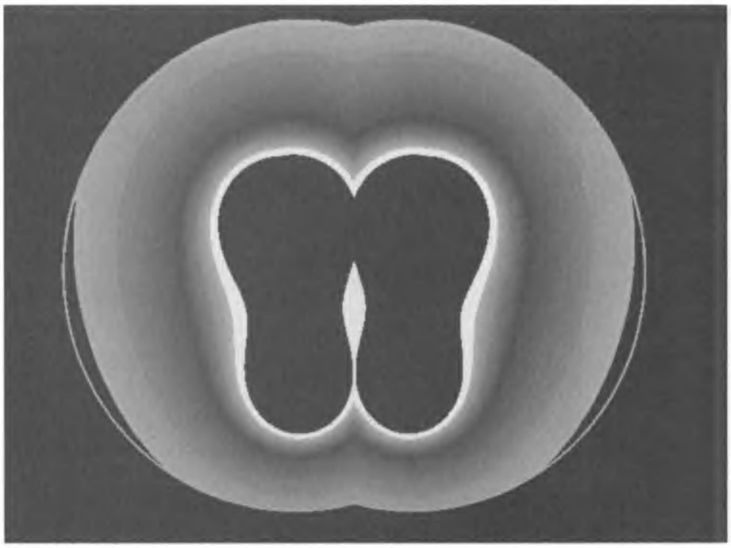
Iterate: $z' = \mathrm{fn}(z) + e^z + c$

where fn(z) is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_julia_per_pixel()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Integer math orbit	LongTrigPlusExponentFractal()	FRACTALS.C
Floating point orbit	FloatTrigPlusExponentFracta()	FRACTALS.C

Manfn+zsqrd



Category: Mandelbrot/Julia Generalized

This type is the Mandelbrot variant corresponding to fractal type julfn+zsqrd.

Example:

```
rhs137            { ; (c) June 1992 Dick Sherry 76264,752
                    ; no commercial use w/o permission
  reset type=manfn+zsqrd function=log float=yes
  corners=-23.043366/-26.245789/-2.199417/2.071243/-26.245789/2.071243
  params=20 maxiter=32000 inside=0 potential=255/300/150 decomp=265
  biomorph=0
  colors=000<14>0000K0<23>070060070<10>0L0<3>mM1020<192>011011000f_8eZ8
  }
```

Formula:

Initialize: $c = z = zpixel$

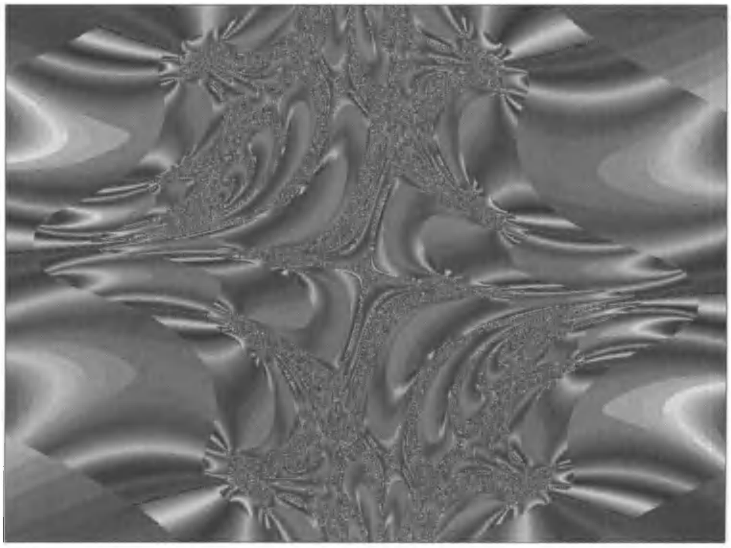
Iterate: $z' = \text{fn}(z) + z^2 + c$

where fn(z) is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	mandel_per_pixel()	FRACTALS.C
Floating point initialization	mandelfp_per_pixel()	FRACTALS.C
Floating point orbit	TrigPlusZsquaredfpFracta()	FRACTALS.C
Integer math orbit	TrigPlusZsquaredFracta()	FRACTALS.C

Julfn+zsqrd



Category: Mandelbrot/Julia Generalized

This type is the Julia variant corresponding to fractal type manfn+zsqrd.

Example:

```
ChevyChrome        { ; 00:02:50.00
                     ; 08 Dec 92 .. caren park
  reset type=julfn+zsqrd function=cosxx
  corners=-2.00383/2.007874/-1.507797/1.5 params=-0.5/0.5 maxiter=2250
  bailout=30 decomp=256 biomorph=0
  colors=000WPJUMHSJF<6>EAb<6>7Nu5Px5Nu<5>A5cB5'<2>E7UG8SH9QJANKBL0000EK<5>_VZ\
  'Y'b'cdcf<4>nru<15>0I5<6>eYJg_LiaNkcPneR<4>zpa<4>jdWgaVcZT'WSYUR<6>AAK<3>ALQ\
  A0SARUBUWBXY<6>DqlDtnCql<5>8Xc7Ta9Q'<5>L4R<8>VHI<7>zH0<8>SDF<3>N3JL0KL1KM2KN\
  3K04J<12>zz0<7>"9YYAXVB<6>KAJMAK0AM<12>ziF<13>K8C413000<15>svcsvcsvcsvc<11>\
  YRK
  }
```

Formula:

Initialize: $z = zpixel$

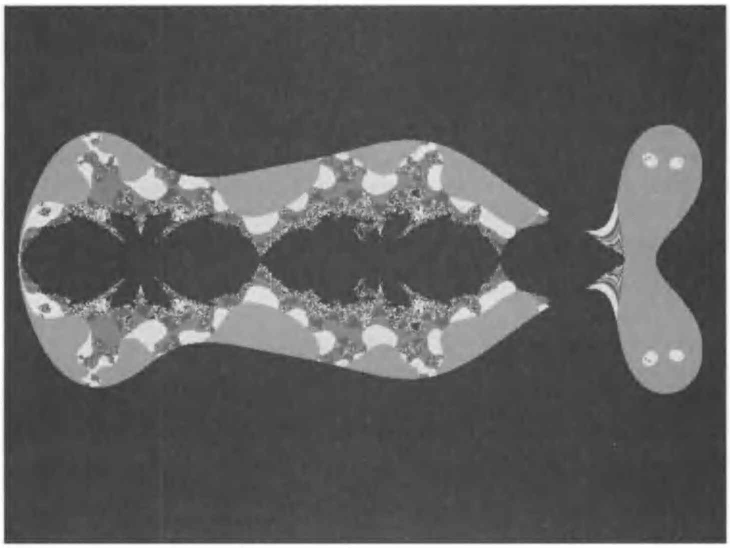
Iterate: $z' = \text{fn}(z) + z^2 + c$

where fn(z) is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	**Routine Name**	**File**
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	julial_per_pixel()	FRACTALS.C
Floating point initialization	juliafp_per_pixel()	FRACTALS.C
Floating point orbit	TrigPlusZsquaredfpFracta()	FRACTALS.C
Integer math orbit	TrigPlusZsquaredFracta()	FRACTALS.C

Mandel(fn||fn)



Category: Mandelbrot/Julia Generalized

This type is the Mandelbrot variant corresponding to fractal type julia(fn||fn). This interesting variation on the theme of generalizing the Mandelbrot set was proposed by Jonathan Osuch. The magnitude of the current orbit value is used to switch between two different variable functions. The "||" in the type name is the C programmer's logical "or." If the two function variables are sqr (the default), then this reduces to the usual Mandelbrot. Try making them different.

Example:

```
Flatworm          {
  reset=1733 type=mandel(fn||fn) function=sqr/cos
  corners=-8.279133/10.66292/-7.483134/6.719686 params=0/0/0.5 float=y
  }
```

Formula:

Initialize: $z = p1$, $c = zpixel$

Iterate: if $|z|^2 <$ shift value, then

$$z' = \mathrm{fn1}(z) + c,$$

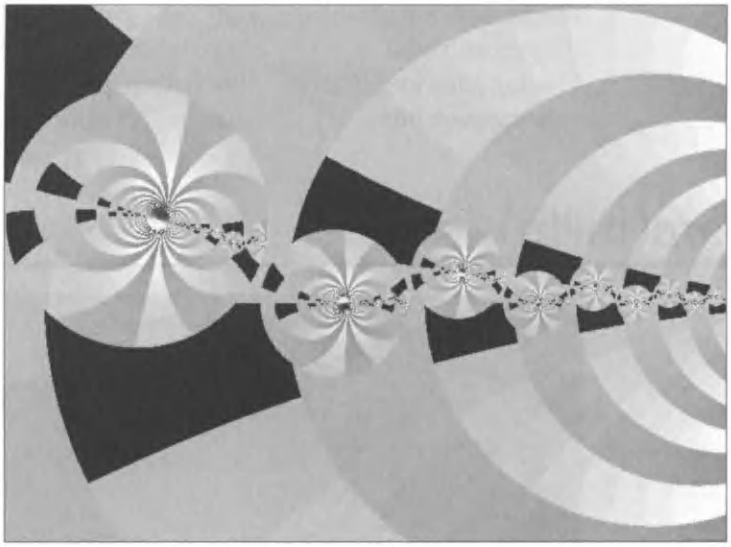
else

$$z' = \mathrm{fn2}(z) + c.$$

where fn1(z) and fn2(z) are each one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:	Routine Type	Routine Name	File
	C fractal engine	StandardFractactal()	CALCFRACT.C
	Integer math initialization	long_mandel_per_pixel()	FRACTALS.C
	Integer math orbit	JuliaTrigOrTrigFractal()	FRACTALS.C
	Floating point initialization	othermandelfp_per_pixel()	FRACTALS.C
	Floating point orbit	JuliaTrigOrTrigfpFractal,()	FRACTALS.C

Julia(fn||fn)



Category: Mandelbrot/Julia Generalized

This type is the Julia variant corresponding to fractal type mandel(fn||fn). This type was proposed by Jonathan Osuch. The algorithm switches between two different functions depending on the magnitude of the current orbit value.

Example:

```
Vortex......        { ; "They are moving right into the vortex!" (C)1993 PGM
                      ; (c)1993  Peter Moreland 100012,3213
  reset type=julia(fn||fn) function=recip/conj
  corners=-7.395582/-7.086847/-0.432864/-0.201313 params=1/-0.33/0.25
  float=y maxiter=20000 bailout=500 decomp=256
  colors=000QEF<11>G2JFOKFOK<11>IONIONIONIOOIOOIOP<33>QOXROYROYROZROZ<6>TO\
  bTObU1bU2c<62>zzzzzzzzy<60>zz1zz0zy0yx0<45>QFF
  }
```

Formula:

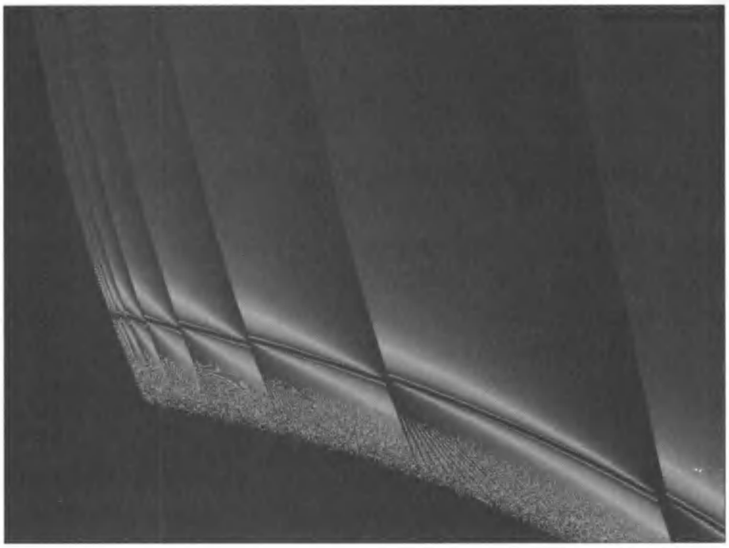
```
Initialize:  $z = zpixel$
Iterate:     if $|z|^2 <$ shift value, then
                 $z' = \text{fn}1(z) + c$,
             else
                 $z' = \text{fn}2(z) + c$.
```

where fn1(z) and fn2(z) are each one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	**Routine Name**	**File**
C fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_julia_per_pixel()	FRACTALS.C
Integer math orbit	JuliaTrigOrTrigFractal()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	JuliaTrigOrTrigfpFractal,()	FRACTALS.C

Manlam(fn||fn)



Category: Mandelbrot/Julia Generalized

This type is the Mandelbrot variant corresponding to fractal type lambda(fn||fn), and is another interesting generalization of mandelfn proposed by Jonathan

Osuch. The algorithm switches between two different functions depending on the magnitude of the current orbit value.

Example:

```
The_Final_Curtain  { ; Regrets, I have a few, but then again, Sid Lives! <g>.
                     ; (c)1993 Peter Moreland,  CIS Address 100012,3213
                     ;
  reset type=manlam(fn||fn) function=sqr/conj
  corners=-1.078314/-0.623828/0.775666/0.47905/-0.772323/0.367679
  params=1.76/0/10 float=y maxiter=500 bailout=90
  colors=LSU6KM<58>inujovinu<164>JRTIRTIRTHRTHRT<23>8MP
  }
```

Formula:

Initialize: z = parameters

Iterate: if $|z|^2 <$ shift value, then

$z' = c\ \text{fn1}(z)$,

else

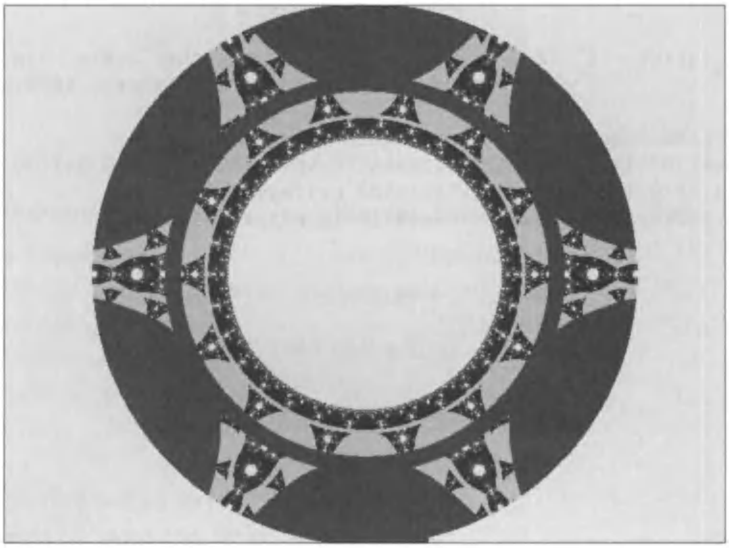
$z' = c\ \text{fn2}(z)$

where fn1(z) and fn2(z) are each one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	Routine Name	File
C fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_mandel_per_pixel()	FRACTALS.C
Integer math orbit	LambdaTrigOrTrigFractal()	FRACTALS.C
Floating point initialization	othermandelfp_per_pixel()	FRACTALS.C
Floating point orbit	LambdaTrigOrTrigfpFractall,()	FRACTALS.C

lambda(fn||fn)



Category: Mandelbrot/Julia Generalized

This type is the Julia variant corresponding to fractal type manlam(fn||fn) proposed by Jonathan Osuch. The algorithm switches between two different functions depending on the magnitude of the current orbit value.

Example:

```
Plate              {
  reset=1733 type=manlam(fn||fn) function=sqr/cos
  corners=-2.642944/2.642944/-1.982208/1.982208 params=0/0/10 float=y
  }
```

Formula:

Initialize: $z = zpixel$

Iterate: if $|z|^2 <$ shift value, then

$z' = c\ \mathrm{fn1}(z)$,

else

$z' = c\ \mathrm{fn2}(z)$

where fn1(z) and fn2(z) are each one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

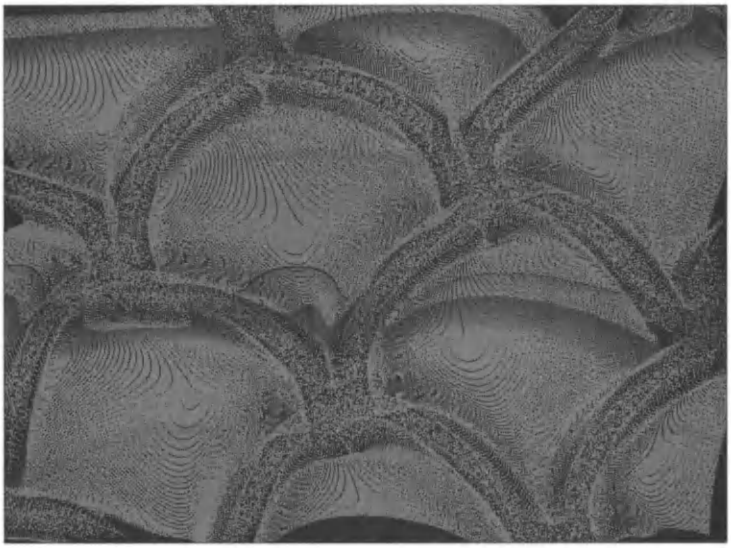
Code:

Routine Type	Routine Name	File
C fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_julia_per_pixel()	FRACTALS.C
Integer math orbit	LambdaTrigOrTrigFractal()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	LambdaTrigOrTrigfpFractall,()	FRACTALS.C

Escape Time to Infinity

The fractal types in this section are additional escape-to-infinity types that do not come in Mandelbrot/Julia pairs.

Popcornjul



Category: Escape Time to Infinity

This formula came from Clifford Pickover's Popcorn fractal. Pickover used it in a completely different way (see type popcorn later in this chapter). It seemed reasonable to use the formula in a Julia-style escape-time fractal, so here you are.

Example:

```
Cellular          { ; (c)1992 Peter Moreland 100012,3213
  reset type=popcorn
  corners=5.335739/5.043655/3.106827/6.645966/3.498291/5.510727
  params=0.078
  colors=000gVP<7>nYKt'GnYJ<5>s'Gt'FsRh<5>t_JaAb<4>eFZeGZfGYfHXgIW<23>t'Foac<7\
  4>t'Ft'Ft'Ft_Ft_FtZFtZFsYF<7>sXF
  }
```

Formula:

Initialize: $x = zpixel_x, y = zpixel_y,$

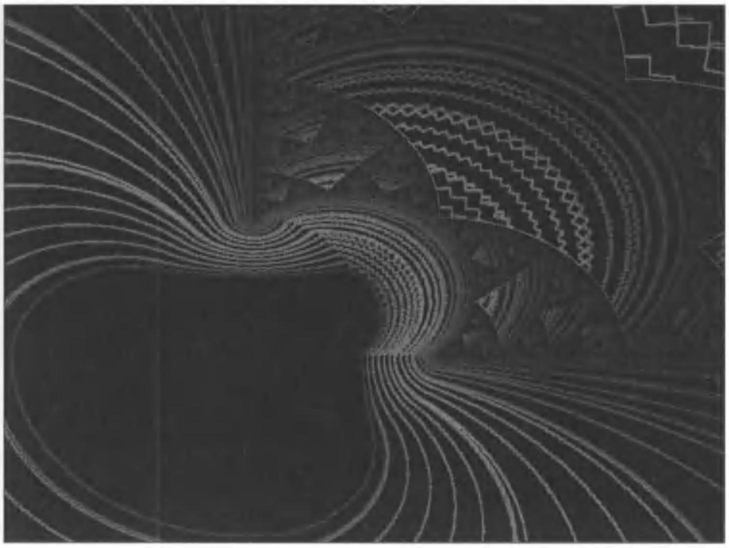
Iterate: $x' = x - 0.05 \sin(y) + \tan(3y)$

$y' = y - 0.05 \sin(x) + \tan(3x)$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_julia_per_pixel()	FRACTALS.C
Integer math orbit	LPopcornFractal()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	PopcornFractal()	FRACTALS.C

Sierpinski



Category: Escape Time to Infinity

This Sierpinski fractal came from Michael Barnsley's book *Fractals Everywhere*. The Sierpinski gasket is one of those ubiquitous fractals—Fractint can generate it using the escape-time method (as is done here with the sierpinski type) as an L-systems fractal, and as an Iterated Function System (IFS) fractal. A Sierpinski Gasket is a nested set of triangles formed by cutting the center out of a triangle, then repeating recursively for the remaining three triangles.

Comparison with the IFS version is enlightening. The IFS Sierpinski gasket is generated with three affine transformations (see Chapter 2, *Fractals: A Primer*). Take the inverse of those transformations, divide the formula into three cases, and Voila! You have a Sierpinski Julia set. Notice that the original IFS transformation divided by 2, and the cases in the formula below multiply by 2. Also note that three IFS affine functions became three cases in the Julia formula.

Example:

```
sba051             { ; (c) 1993 Richard H. Sherry, CIS:76264,752
  reset type=sierpinski passes=b corners=-0.9/1.699996/-0.8999948/1.7
  maxiter=256 bailout=155 fillcolor=253 inside=zmag outside=mult
  potential=255/200/0 invert=0.67/0/0 periodicity=0
  colors=0008H0<11>Zc0<2>TY0QW00U0LS0JQ0<7>0A0<15>Zc0<15>0A0SNC<29>xo'zpay\
  pa<13>kcRibRibR<14>UPF111<29>ppp<31>000PFF<29>xeDzfCyfC<30>PFF0A0<2>6F0
  }
```

Formula:

Initialize: $x = zpixel_x, y = zpixel_y,$

Iterate:

$x' = 2x - 1 \quad \text{if } x > .5$

$x' = 2x \quad \text{if } x <= .5$

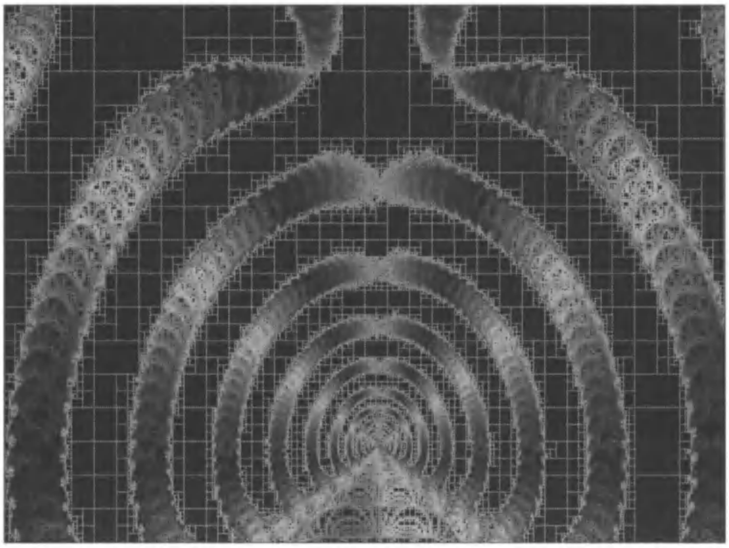
$y' = 2y - 1 \quad \text{if } y > .5$

$y' = 2y \quad \text{if } y <= .5$

Code:

Routine Type	**Routine Name**	**File**
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_julia_per_pixel()	FRACTALS.C
Integer math orbit	SierpinskiFractal()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	SierpinskiFPFractal()	FRACTALS.C

Spider



Category: Escape Time to Infinity

Yet another variation on the basic Mandelbrot formula. In this formula, the variable *c*, which remains fixed for each pixel calculation, is repeatedly divided by 2 and added to the new orbit to get the new *c*.

Example:

```
SpiderLeggings      { ; 00:11:23.54
                      ; 11 dec 92 .. caren park
  reset type=spider passes=t
  corners=-1.3204703/-1.3828737/-0.052686/0.056887/-1.3828737/0.056887
  maxiter=500 fillcolor=7 inside=zmag outside=imag logmap=yes
  periodicity=4
  colors=000ZKD<2>RCCEEEHHGOOOLLL<12>ppn<15>FOAUPF<2>0I5<6>eYJg_LiaNkcPneR<4>z\
  pa<12>UPFHGMEDLAAK<3>ALQAOSARUBUWBXY<6>DqlDtnCql<5>8Xc7Ta9Q'<5>L4R<8>VHI<7>z\
  H0<8>SDF<4>LOKOG5<21>Np6Pr70p7<16>0G5ZJE<6>wgF<8>XKDUHDSGC<6>AB7<12>svcsvcsv\
  csvc<9>aX0<6>kbL000ocJ<6>zjF<9>aMD
  }
```

Formula:

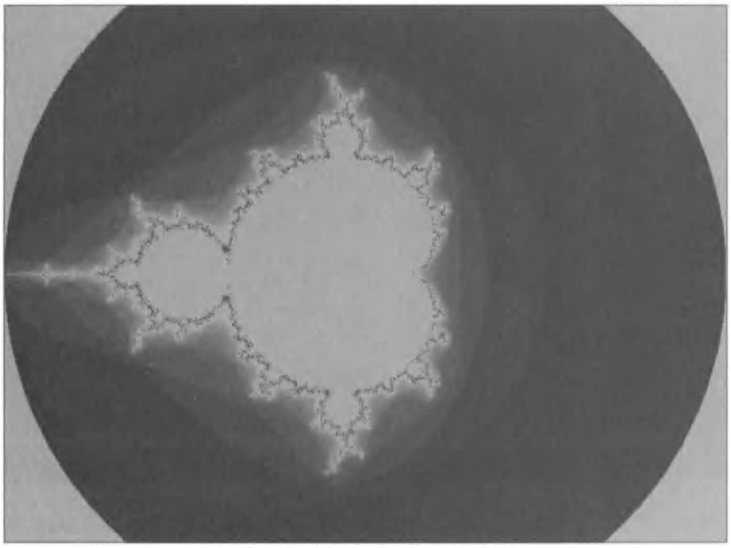
Initialize: $z = c = zpixel$

Iterate: $z' = z^2 + c$

$c' = c/2 + z'$

Code:	Routine Type	Routine Name	File
	Fractal engine	StandardFractactal()	CALCFRACT.C
	Integer math initialization	mandel_per_pixel()	FRACTALS.C
	Integer math orbit	SpiderFractal()	FRACTALS.C
	Floating point initialization	mandelfp_per_pixel()	FRACTALS.C
	Floating point orbit	SpiderfpFractal()	FRACTALS.C

Test



Category: Escape Time to Infinity

The test fractal type exists for the benefit of intrepid C programmers who want to try "rolling their own" fractals without delving too deeply into Fractint's somewhat challenging code. Programmer's who want to try this and have a C compiler will find the routines in the file TESTPT.C.

Example:

```
Test              { ; Shows missing stripe
  reset=1733 type=test corners=-3.140845/3.140845/-2.355634/2.355634
  float=y
  }
```

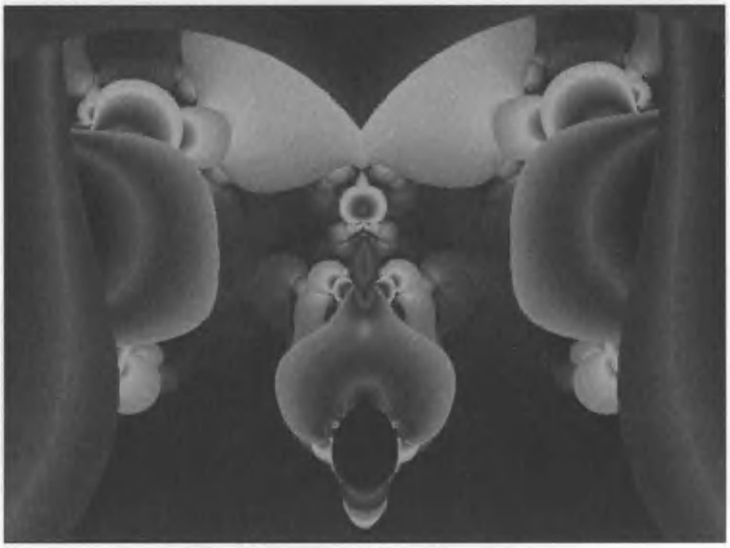
Formula: Initialize: $c = z = zpixel$

Iterate: $z' = z^2 + c$

Code:

Routine Type	Routine Name	File
C fractal engine	test()	TESTPT.C

Tetrate



Category: Escape Time to Infinity

Iterating a simple exponential function. The default fractal has large chunky splotches that look very different from the usual stylish escape-time stripes.

Example:

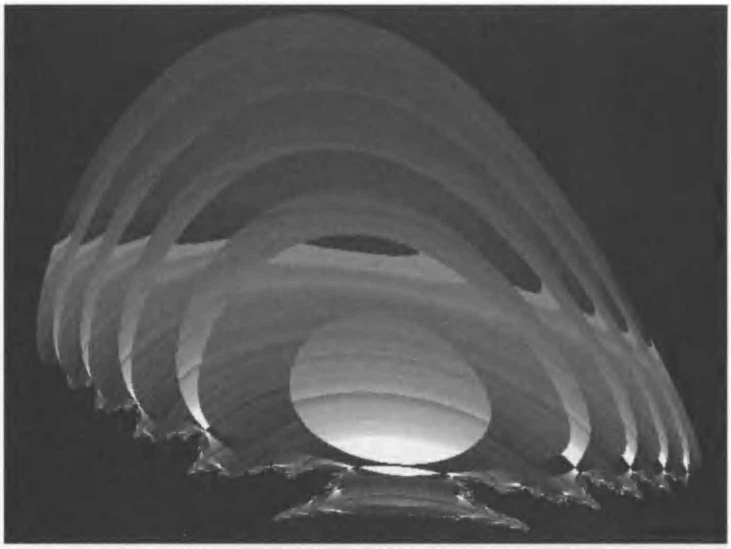
```
t17058            {
  reset type=tetrate
  corners=-0.099385/0.048997/0.109008/-0.088834/0.048997/-0.088834
  params=-1 inside=zmag outside=imag potential=255/511/0
  invert=0.05/0.01/0.01
  colors=000ME8LD7<53>kSIlSImTJnTJnTJ<2>qVKqVKqUJ<28>r53s42s53<60>spyspyspx<19\
  >slnslmsllsklskkskj<21>tdWucVtcV<7>nZQmYPlXPlXP<35>NF8
  }
```

Formula: Initialize: $z = c = zpixel$

Iterate: $z' = c^z$

Code:	Routine Type	Routine Name	File
	Fractal engine	StandardFractactal()	CALCFRACT.C
	Floating point initialization	othermandelfp_per_pixel()	FRACTALS.C
	Floating point orbit	TetratefpFractal()	FRACTALS.C

Tim's_error



Category: Escape Time to Infinity

A coding error by one of your intrepid authors produced this fractal formula. The corollary to "everyone is famous for 15 minutes" is "everyone gets one fractal named after himself or herself," so this is it. Try setting `function=sqr` to find a prehistoric pterodactyl.

Example:

```
Eggzactly!          { ; You must be yoking... (C)1993 Peter Moreland 100012,3213
  reset type=tim's_error function=sqr
  corners=-1.207252452/-1.087561701/-0.0635867674/-0.2180058723/-1.0877397\
  04/-0.218058332 float=y maxiter=25 bailout=10 outside=real
  decomp=256 biomorph=256
  colors=000310521342641333533355653930D30961D51963D73783A82D91FC1A94D94BC\
  4ED44789785ABA99DA8AD8ED8AACDBCADDEDCG30H61L71H73L74P74HA2L92ID2LF2H94LA\
  4HD5LD5QC5HB7LB7ID8ME8HABHECLECPB7SB8PE8TE8QFBUFBFG5EHAJG2MG2IG5MH4NK5PI\
  2UH2RK2TL2PI3TH5QL4TM4U05IH8MH9IK9ML9IHDMICILCMLDQH9TH9PL9UL9PICTICQLDUL\
DMPDSPB37I5EICEH4ERAEPIEHPFIHEN6GLEHJFPL6IQ9KSCPTIHGLIGJLHMLGHIKLIKHKLML\
KRLHLPJTPJJLPQMPLPRRRQXE5XD9YL4YQ5dP5YKCdKBZPCeRBZMGdMHYPH'PHXTHaSIXQKaQ\
KXTLaTKeRKYMQZTPfTPkTMlUQUXIUXRYWDfX9_YLdXM'XSfYSecRmXTlfR4FV6IW8MW9OYJN\
WRNWMQXSTYKRcRVbYUXfUXUX_UYd_ZZeYWiYWe'Wh'XeZ_iY_ea_ia_acZgd_Z'efaeadggf\
fn'YsaZnd_tc'nbcsbcnfeufeikhnlhtlhZbjdbjaflfimairnjkujjhlpimrlllpllnolpo\
lmmoqnomoppppunortquuonpswrsqtwuttxttxwttvxyvvuxyzzy<24>zzy
}
```

Formula: Initialize: $z = zpixel, c = z^{z-1}$

Iterate: $tmp = \mathrm{fn}(z)$

$tmp_x = tmp_x\, c_x - tmp_y\, c_y;$

$tmp_y = tmpx\, c_y - tmp_y\, c_x;$

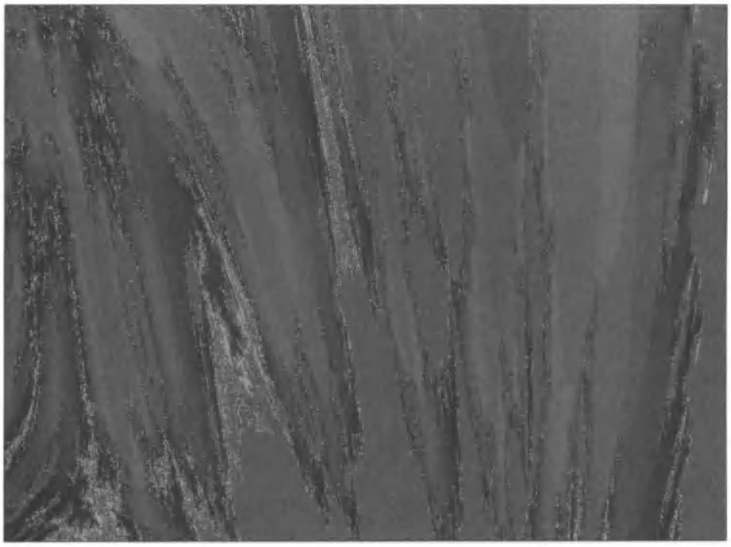
$z' = tmp + pixel;$

where fn(z) is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

ESCAPE TIME TO INFINITY GENERALIZED

In this section, we look at more escape-time-to-infinity fractals. These fractals have no Mandelbrot/Julia variations and they use function variables, so this small number of types account for quite a few fractal possibilities.

fn(z)+fn(pix)



Category: Escape Time to Infinity Generalized

If $p_1 = p_2 = 1$, fn1(z) = sqr and fn2(z) = ident, then this type is the classic Mandelbrot.

Example:

```
Downpoor          { ; Pastel watercolour thingy - (c) 1992 Pete M. 100012,3213
                    ; Blantyre House, Cossack SQ, Nailsworth, Gloucs, UK.
  reset type=fn(z)+fn(pix) function=cosxx/cosh
  corners=-1.502386/-1.371228/2.3414666/2.4398491 params=1.5/0.06/1
  colors=000eQUhFLk4C<7>duQ'aaYIl<2>ZViZZhXWe<6>K8H<2>5G'<7>Y9WXXSWtP<7>rgvkVu\
  dIu<6>nVC<2>ifQgjVgbQ<2>eDAeV_ely<3>TDd<6>QuJIhYBXl<2>0h_<5>mjr<5>ohrohrqbi<\
  3>uGBqWQnjc<2>1Vw<7>kDj<5>MEM<5>cX5MEv<6>gctjgtkhr<4>nkf<7>73g0FhdRieIh
  }
```

Formula:

Initialize: $c = z = zpixel$

Iterate: $z' = p_1\mathrm{fn}(z) + p_2\mathrm{fn}(c)$

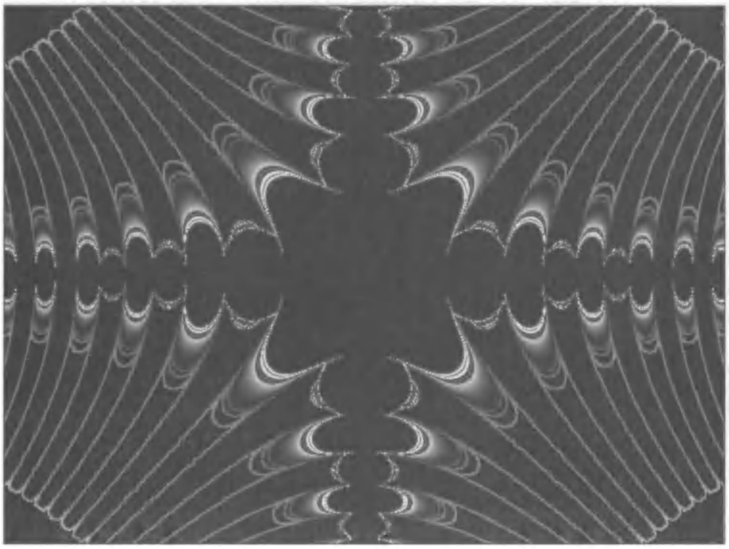
where fn(z) is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_richard8_per_pixel()	FRACTALS.C

Integer math orbit	Richard8Fractal()	FRACTALS.C
Floating point initialization	otherrichard8fp_per_pixel()	FRACTALS.C
Floating point orbit	Richard8fpFractal()	FRACTALS.C

fn(z*z)



Category: Escape Time to Infinity Generalized

This fractal is a generalization of the clasic Julia with no constant term.

Example:

```
Uh...yeah...II     { ; Interesting
                     ; BG Dodson 1992 71636,1075
  reset type=fn(z*z) function=cotan passes=b corners=-4.0/4.0/-3.0/3.0
  float=y maxiter=32000 fillcolor=200 inside=bof61 logmap=4
  potential=253/3000/1 decomp=128
  colors=000151131000<15>rAr<15>000<15>hh0<14>330000003<14>00r<15>000<15>r00<1\
  5>000<15>0pp<14>044000222<14>hhh<14>333000000<26>007017037<13>4S4<15>171
  }
```

Formula:

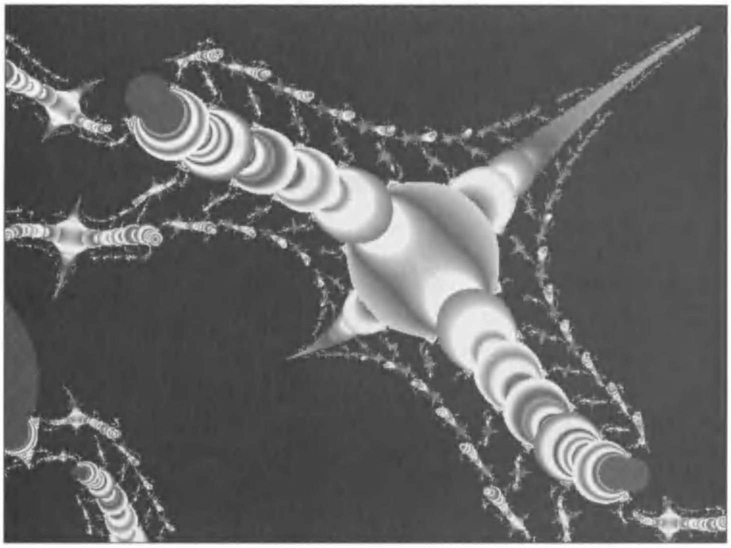
Initialize: $z = zpixel$

Iterate: $z' = \text{fn}(z^2)$

where fn(z) is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:	Routine Type	Routine Name	File
	Fractal engine	StandardFractactal()	CALCFRACT.C
	Integer math initialization	long_julia_per_pixel()	FRACTALS.C
	Integer math orbit	TrigZsqrdFractal()	FRACTALS.C
	Floating point initialization	juliafp_per_pixel()	FRACTALS.C
	Floating point orbit	TrigZsqrdfpFractal()	FRACTALS.C

fn*fn



Category: Escape Time to Infinity Generalized

This generalized function covers over 200 cases, all formed by taking the product of two functions with no constant term.

Example:

```
cet16104          {
  reset type=fn*fn function=exp/tanh
  corners=0.144507/1.626775/1.426092/2.537793 float=y inside=0
  potential=255/511/0
  colors=000AB7vH3zH0<7>WEDSDFRAG<2>N3JLOKL1KM2KN3K04J<12>zz0<7>"9YYAXVB<6>KA\
  JMAKOAM<12>ziF<13>K8C413000332775EEA<11>svcsvcsvcsvc<13>UMHSJFQIJ<5>EAb<6>7N\
  u5Px5Nu<5>A5cB5'<6>KBLMCIOEK<3>WQUYSX_VZ'Y'<6>nru<15>0I5<5>bVHeYJg_LiaN<6>zp\
  a<4>jdWgaVcZT'WSYUR<6>AAK<4>AOSARUBUWBXYB__Bba<5>Dtn<6>8Xc7Ta9Q'<5>L4R<8>VHI\
  <5>rH5 cyclerange=2/255
  }
```

Formula: Initialize: $z = zpixel$

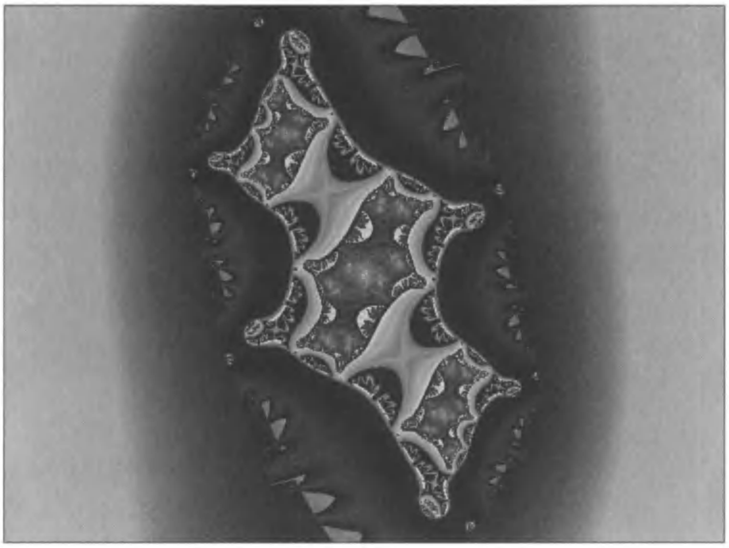
Iterate: $z' = \mathrm{fn}(z)^2$

where fn(z) is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_julia_per_pixel()	FRACTALS.C
Integer math orbit	TrigXTrigfpFractal()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	TrigXTrigfpFractal()	FRACTALS.C

fn*z+z



Category: Escape Time to Infinity Generalized

Another generalized fractal formula, this time with the variable z added.

Example:

```
Cut_me              { ; Cut me and I bleed Fractals!
                      ; Ronald C. Lewen, 76376,2567
  reset type=fn*z+z function=cosh corners=-4.0/3.999993/-2.999991/3.0
  params=1.414/0/0/1.414 inside=255 potential=255/128/0
colors=00000j0j0jj0j00j0j0jjkkkkrkdmwCCC_C_<9>sCs<20>ECsCCsCEs<19>CqsCssCsq<\
19>CsECsCEsE<19>qsqssssq<19>ssEssCsqC<19>sECsCCqCC<18>GCCECCACCECE<9>YCYbjb\
<2>zjbFrb<5>zrbFzb<5>zzbFFj<5>zFjFNj<5>zNjFVj<5>zVjFbj<5>zbjFjj<5>zjjFrj<4>r\
rjzywccdWWWz000z0zz000zz0z0zzzzz
}
```

Formula: Initialize: $z = zpixel$

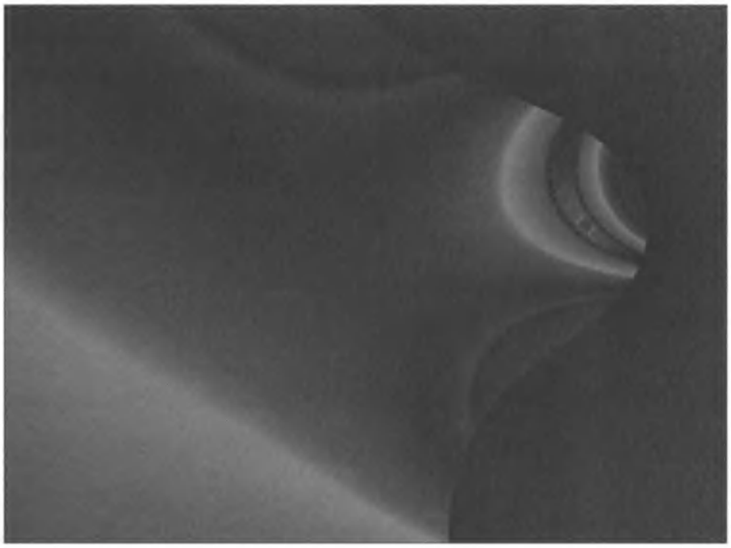
Iterate: $z' = p_1\text{fn}(z)z + p_2z$

where fn(z) is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	**Routine Name**	**File**
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	julia_per_pixel()	FRACTALS.C
Integer math orbit	ZXTrigPlusZFractal()	FRACTALS.C
Floating point initialization	juliafp_per_pixel()	FRACTALS.C
Floating point orbit	ZXTrigPlusZfpFractal()	FRACTALS.C

fn+fn



Category: Escape Time to Infinity Generalized

This type encompasses over 200 variations formed by adding two functions with no constant term.

Example:

```
BRACLET            {; Lee Skinner
  reset=1720 type=fn+fn function=cosh/sqr passes=1
  corners=1.77888557/1.78734031/1.16095279/1.1673213 params=1/0/-1
  float=y maxiter=1023 bailout=8192 inside=maxiter logmap=yes
  potential=255/511/0
  colors=000OSV<65>BCWEC0OCG<3>8Q7<6>LE5NC4NC4<5>NB2NB2OB2PB2<24>gN1hN1hP1\
  <6>nb1oe1oe1<26>mS6mS6lR7kR7<8>hPAhPAhPBhOBhNAhMA<2>gI8gH8eG8<9>M32<9>z3\
  3Y3g<55>zOVOTWOSW
  }
```

Formula: Initialize: $z = zpixel$

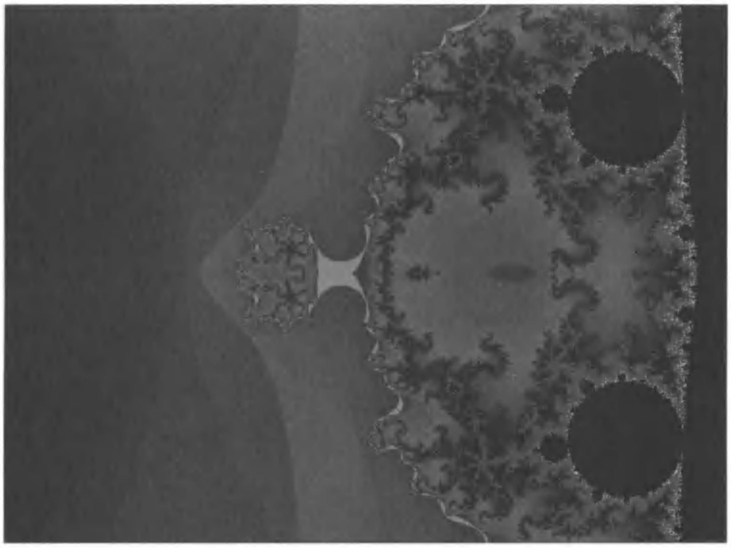
Iterate: $z' = p_1\mathrm{fn}(z) + p_2\mathrm{fn}(z)$

where fn(z) is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	**Routine Name**	**File**
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_julia_per_pixel()	FRACTALS.C
Integer math orbit	TrigPlusTrigFractal()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	TrigPlusTrigfpFractal()	FRACTALS.C

Marksmandelpwr



Category: Escape Time to Infinity Generalized

This formula was first proposed by Mark Peterson. This formula is set apart from others by the function coefficient *c* that is initialized for each pixel to z^{z-1}.

Example:

```
SNAKDEN.GIF          { ; Lee Skinner
  reset=1720 type=marksmandelpwr function=sqr passes=1
  corners=-1.10652/-1.0778615/-0.0107626/0.01077 float=y maxiter=2047
  inside=0 logmap=yes
  colors=000sgJ<4>hXabVeMDU<2>VKSdRS<5>F07vdOPGWcMW<6>X_W<2>UXZTWZSVY<20>2\
  2L00K00K<10>A0KB0KD0J<41>u0Gw1Fw1F<117>z00zt5<12>zt2zt1zt1zs0ys1<2>vs6zm\
  8xkCviG
  }
```

Formula:

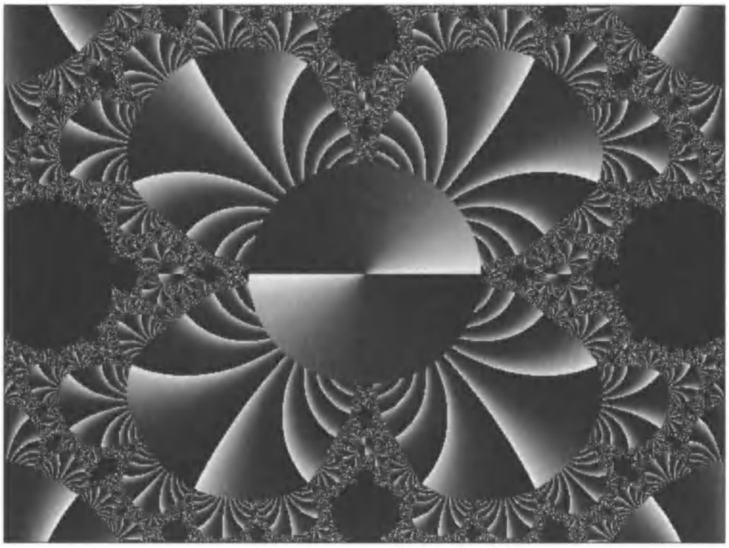
Initialize: $z = zpixel$, $c = z^{z-1}$

Iterate: $z' = c\text{fn}(z) + zpixel$;

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	marks_mandelpwr_per_pixel()	FRACTALS.C
Integer math orbit	MarksMandelPwrFractal()	FRACTALS.C
Floating point intialization	marks_mandelpwrfp_per_pixel	FRACTALS.C
Floating point orbit	MarksMandelPwrfpFractal()	FRACTALS.C

sqr(1/fn)



Category: Escape Time to Infinity Generalized

This type is one of the few Fractint formulas that divides by a function.

Example:

```
Purple_KaleidoRing { ; (c) 1992 Bill Potter/Rings of foil kaleidoscope
                     ; have fun use & abuse but no commercial use wo permission
  reset type=sqr(1/fn) function=sin passes=g float=y
  corners=-1.12/1.12/-0.84/0.84
  maxiter=5000 decomp=255
  colors=000F0K<57>T0aT0bT0bU0cU0c<62>zzzzzzzzy<60>zz1zz0zy0yx0<56>I4IH3JG2JF0\
  KF0KF0KF0K
  }
```

Formula:

Initialize: $z = zpixel$

Iterate: $z' = 1/\mathrm{fn}(z)^2$

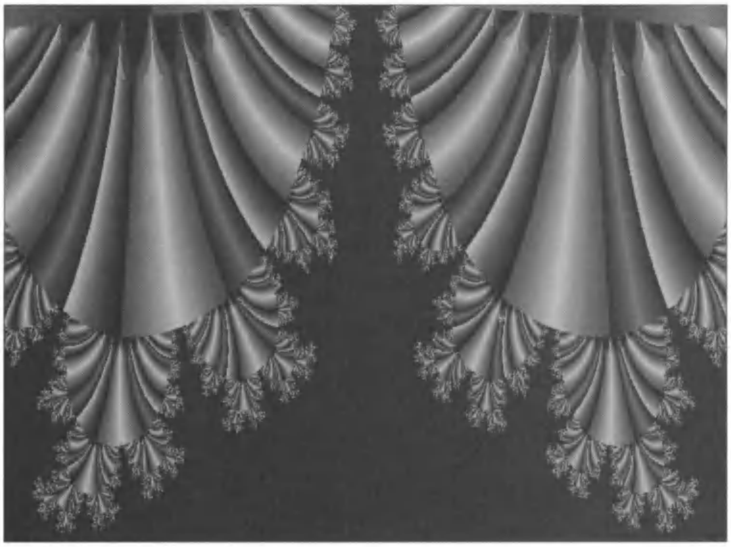
where fn(z) is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_julia_per_pixel()	FRACTALS.C
Integer math orbit	Sqr1overTrigFractal()	FRACTALS.C

Floating point initialization	otherjuliafp_per_pixel	FRACTALS.C
Floating point orbit	Sqr1overTrigfpFractal()	FRACTALS.C

sqr(fn)



Category: Escape Time to Infinity Generalized

The square of the function—that tells it all. The formula has no constant term added.

Example:

```
sae008              { ; (c) 1993 Richard H. Sherry CIS:76264,752
  reset=1732 type=sqr(fn) function=sin
  corners=-1.26561/1.259735/0.506409/1.76059 maxiter=256 fillcolor=0
  inside=-102 outside=real logmap=yes decomp=255 periodicity=0
  colors=0002B0<14>Zc0<15>0A0SNC<29>xo'zpaypa<13>kcRibRibR<14>UPF111<29>pp\
  p<31>000PFF<29>xeDzfCyfC<30>PFF0A0<15>Zc0<13>3C00F0
  }
```

Formula: Initialize: $z = zpixel$

Iterate: $z' = \mathrm{fn}(z)^2$

where fn(z) is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_julia_per_pixel()	FRACTALS.C
Integer math orbit	SqrTrigFractal()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel	FRACTALS.C
Floating point orbit	SqrTrigfpFractal()	FRACTALS.C

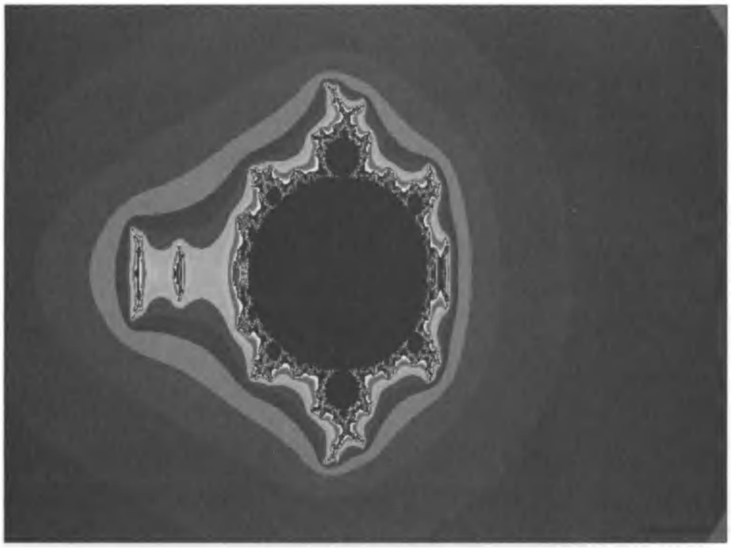
4-D Escape Time

The complex numbers that Fractint uses for most of its fractal calculations are two-dimensional extensions of familar real numbers. What about doing arithmetic with still higher dimensional numbers? At the turn of the century, mathematicians proved that there is no perfect solution to this problem. All four-dimensional generalizations of real and complex numbers will fail some of the common rules of real arithmetic. These nineteenth century mathematicians adopted quaternions as their preferred kind of four-dimensional numbers. Quaternions satisfy all the algebraic rules governing real and complex numbers except the communtative law of multiplication, which states that multiplication of two numbers in either order gives the same result. They rejected an alternative kind of numbers called hypercomplex numbers. These numbers satisfy the commutative law of multiplication, but division by nonzero numbers does not always work. The early mathematicians thought that the ability to divide was more important than the order of multiplication. For fractal purposes it turns out that the long-forgotten hypercomplex numbers are better—Fractint's variable functions work great with hypercomplex numbers, but don't work with quaternions. (See Appendix C, *Complex and Hypercomplex Numbers,* for more details on how four-dimensional arithmetic works.)

The algebraic limitations of quaternions and hypercomplex numbers do not need to stop us from using them to generate fractals. The classic Mandelbrot and Julia sets generalize easily using either kind of number. The Mandelbrot and Julia sets have orbits that trace paths in two dimensions, and result in sets of numbers in two dimensions. Quaternion and hypercomplex Mandelbrot and Julia sets have four-dimensional orbit paths and result in four-dimensional sets. Our problem is how to view these higher dimensional fractals. We present a 2-D approach here, and a 3-D approach later in this chapter in connection with julibrots.

The simplest way to plot higher dimensional fractals is to slice them with a plane. Just as an architect can draw a two-dimensional cross section of a three-dimensional building, Fractint can slice a four-dimensional fractal and get a two-dimensional result. (It is also possible to slice a four-dimensional fractal and get a three-dimensional result—see fractal type julibrot in the 3-D Fractals section later in this chapter.)

Quat



Category: 4-D Escape Time

This fractal type is the Mandelbrot variant corresponding to fractal type quatjul. The quat fractal is the quaternion Mandelbrot set. In fact, the classic Mandelbrot set is a special case of type quat.

As with the complex Mandelbrot set, the orbit variable is initialized with a value corresponding to a screen pixel. Because your screen is two-dimensional, only two of the four dimensions of the variable *c* can be initialized. The other two variables can be enterd as parameters; they determine where the "knife" slices the 4-D Mandelbrot. These parameters are labeled c_j and c_k' in the following fomula. When c_j, and c_k are both zero, the resulting fractal is the familiar Mandelbrot set. The quaternion Mandelbrot set is a four-dimensional object. Sliced directly through the middle, the 2-D cross section is the classic Mandelbrot.

Example:

```
quat              {
  reset=1733 type=quat corners=-2/2/-1.5/1.5
  params=0/0/0.3/-0.2 float=y
  periodicity=0
  }
```

Formula:

Initialize: $q = (0,0,0,0)$, $c = (zpixel_x, zpixel_y, c_j, c_k)$

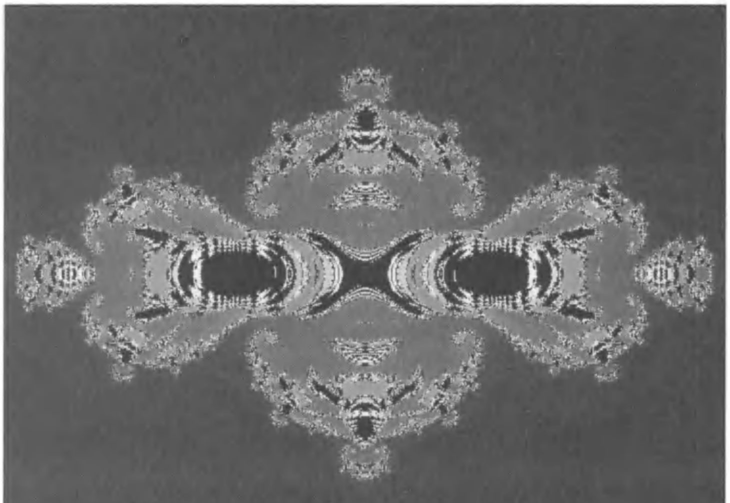
Iterate: $q' = q^2 + c$

where $q = (q_1, q_i, q_j, q_k)$ and $c = (c_1, c_i, c_j, c_k)$ are four-dimensional quaternion numbers.

Code:

Routine Type	**Routine Name**	**File**
C fractal engine	StandardFractactal()	CALCFRACT.C
Floating point initialization	quaternionfp_per_pixel()	FRACTALS.C
Floating point orbit	QuaternionFPFractal()	FRACTALS.C

Quatjul



Category: 4-D Escape Time

This fractal type is the Julia variant corresponding to fractal type quat. The quatjul fractals are the quaternion Julia sets. In fact, the classic Julia sets are special cases of type quatjul.

The quatjul fractal type requires six parameters. Four parameters are required for the variable *c* that is fixed for each Julia set because quaternions are four dimensional. Another two parameters are required for the orbit initializer to determine where the 2-D slice of the 4-D fractal is made.

Example:

```
QuatJ1              { ; Julia Quaternion in blue-greens
                      ; By Dan Farmer
                      ; Quaternion example
  reset type=quatjul corners=-1.388288/1.388288/-1.041216/1.041216
  params=-0.745/0/0.113/0.05 float=y inside=0 periodicity=0
  colors=000143<146>Ioh000000
  }
```

Formula:

Initialize: $q = (zpixel_x, zpixel_y, z_j, z_k)$

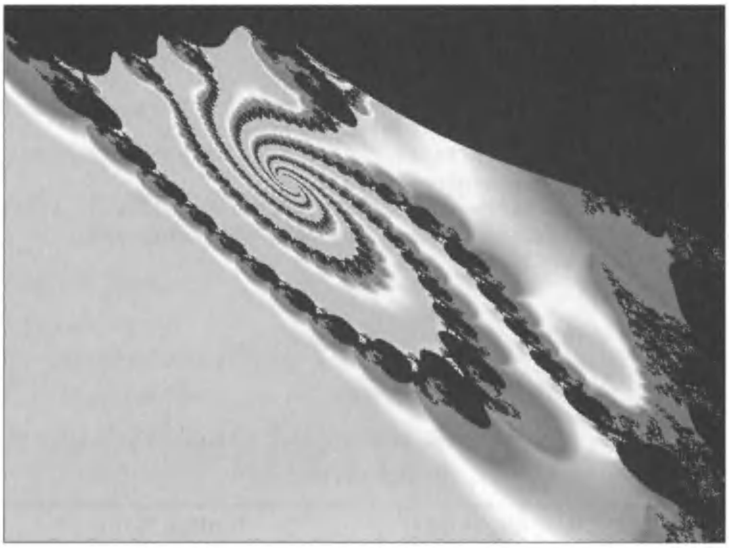
Iterate: $q' = q^2 + c$

where $q = (q_1, q_i, q_j, q_k)$ and $c = (c_1, c_i, c_j, c_k)$ are four-dimensional quaternion numbers.

Code:

Routine Type	Routine Name	File
C fractal engine	StandardFractactal()	CALCFRACT.C
Floating point initialization	quaternionjulfp_per_pixel()	FRACTALS.C
Floating point orbit	QuaternionFPFractal()	FRACTALS.C

Hypercomplex



Category: 4-D Escape Time

This fractal type is the Mandelbrot variant corresponding to fractal type hypercomplexj. The hypercomplex fractal is the hypercomplex Mandelbrot set. In fact, the classic Mandelbrot set is a special case of type hypercomplex.

As with the complex Mandelbrot set, the orbit variable is initialized with a value corresponding to a screen pixel. Since your screen is two dimensional, only two of the four dimensions of the variable c can be initialized. The other two variables can be entered as parameters; they determine where the "knife" slices the 4-D Mandelbrot. These parameters are labelled c_j and c_k in the folowing fomula. When c_j, and c_k are both zero, the resulting fractal is the familiar Mandelbrot set. The hypercomplex Mandelbrot set is a four-dimensional object. Sliced directly through the middle, the 2-D cross section is the classic Mandelbrot.

Hypercomplex numbers were brought to the attention of the authors by Clyde Davenport, the author of *A Hypercomplex Calculus with Applications to Special Relativity*. The hypercomplex and hypercomplexj fractal types were proposed and implemented in Fractint by Tim Wegner. To the best of our knowledge, the use of hypercomplex numbers to create fractals is published here for the first time. A somewhat similar scheme was implemented independently by Jason McGinnis of the United Kingdom in his version of Fractint. The authors hope to merge Jason's ideas with the current approach in a future version.

Example:

```
Hypercomplex{ ; (C)1993 Peter Moreland
                    ; 100012,3213
  reset type=hypercomplex function=exp passes=1
  corners=-1.38284756/-1.38985234/-5.10519207/-5.08976125/-1.40480462/-5.0\
  8092665 params=1.3/1/1.5/1.11
  float=y maxiter=250 bailout=100 inside=bof60 decomp=256 biomorph=0
  periodicity=0 viewwindows=4.2/0.75/yes/0/0
  colors=000Bzz<6>2zz0zz0yz<37>0Ez0Cz0Bz09z08z<3>02z00z00y<59>002000000000\
  <29>00k00m01m<29>0ky0mz1mz<30>zzz<38>Czz
  }
```

Formula:

Initialize: $h = (0,0,0,0)$, $c = (zpixel_x, zpixel_y, c_j, c_k)$

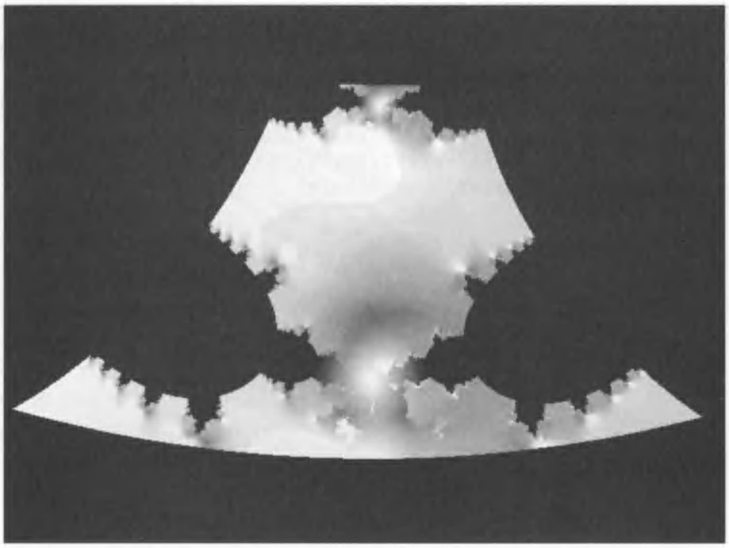
Iterate: $h' = \text{fn}(h) + c$

where $h = (h_1,h_i,h_j,h_k)$ and $c = (c_1,c_i,c_j,c_k)$ are four-dimensional hypercomplex numbers, fn(h) is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero, generalized to work with hypercomplex numbers.

Code:

Routine Type	**Routine Name**	**File**
C fractal engine	StandardFractactal()	CALCFRACT.C
Floating point initialization	quaternionfp_per_pixel()	FRACTALS.C
Floating point orbit	HyperComplexFPFractal()	FRACTALS.C

Hypercomplexj



Category: 4-D Escape Time

This fractal type is the Julia variant corresponding to fractal type hypercomplex. The hypercomplexj fractals are the hypercomplex Julia sets. In fact, the classic Julia sets are special cases of type hypercomplexj.

The hypercomplexj fractal type requires six parameters. Four parameters are required for the variable c that is fixed for each Julia set because hypercomplex numbers are four dimensional. Another two parameters are required for the orbit initializer to determine where the 2-D slice of the 4-D fractal is made.

Example:

```
Flaked_Gold        { ; Museum exhibit from a dig, dig it!
                     ; (C)1993 Peter Moreland 100012,3213
  reset type=hypercomplexj function=exp
  corners=0.7495447/0.6251076/-0.1647639/0.0808346/0.58686/0.052149
  params=-0.745/0/0.113/0.05 float=y maxiter=250 bailout=100
  inside=bof60 decomp=256 distest=1/71 biomorph=0 periodicity=0
  colors=000kf6<38>I4IH3JG2JFOKFOK<9>HOMHOMHONIONIONIOO<45>TOaTObTObUOcUOc\
  <62>zzzzzzzzy<60>zz1zz0zy0yx0<16>lg6
  }
```

Formula: Initialize: $h = (zpixel_x, zpixel_y, z_j, z_k)$

Iterate: $h' = \text{fn}(h) + c$

where $h = (h_1,h_i,h_j,h_k)$ and $c = (c_1,c_i,c_j,c_k)$ are four-dimensional hypercomplex numbers, fn(h) is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero, generalized to work with hypercomplex numbers.

Code:

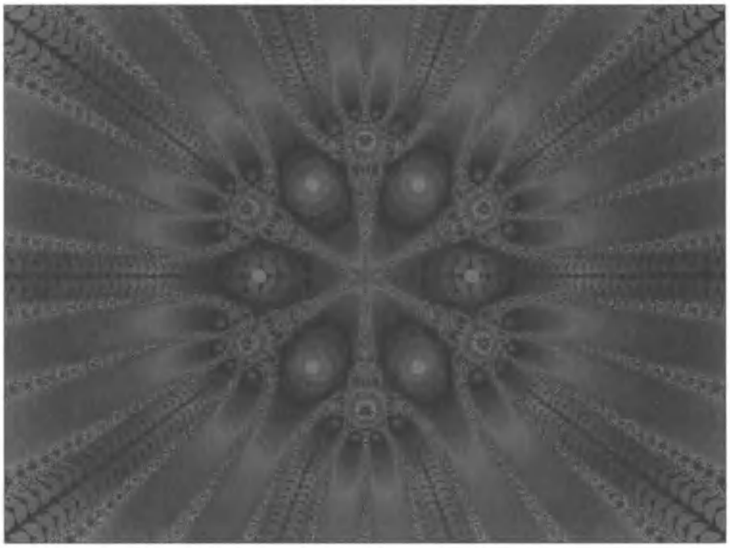
Routine Type	**Routine Name**	**File**
C fractal engine	StandardFractactal()	CALCFRACT.C
Floating point initialization	quaternionjulfp_per_pixel()	FRACTALS.C
Floating point orbit	HyperComplexFPFractal()	FRACTALS.C

ESCAPE TIME TO FINITE ATTRACTOR

Fractal images can be created by coloring pixels according to the number of iterations it takes for an orbit to escape a bailout radius. An entirely different approach is to color pixels according to the number of iterations it takes for the orbit to be captured by a finite attractor. A finite attractor is a point that orbits gravitate toward. Fractals are created by the titanic struggle of multiple attractors striving to capture orbits. Orbits that begin near an attractor dive straight toward that attractor, but orbits that begin between attractors are caught in a tug-of-war. Sometimes, one attractor wins, sometimes another. The most famous example of this behavior is the Newton fractal.

Two different coloring schemes are used with these fractals. Newton, complexnewton, and halley use escape-time coloring; pixels are colored according to the number of iterations before an orbit is within a fixed threshold distance to an attractor. Types newbasin, complexbasin, and frothybasin color pixels according to *which* attractor wins. The fractal image is broken into solid areas where all the orbits originating in these areas are captured by the same attractor.

Newton



Category: Escape Time to Finite Attractor

Newton's method is a famous algorithm for finding the roots of polynomials. Fractal type Newton uses a special case of this algorithm based on the polynomial $z^n - 1$. A root of this polynomial is a number z such that $z^n - 1 = 0$. If z is a real number, the solution to this equation is easy: $1^n - 1 = 0$ so $z = 1$. However, because z can be a complex number, other answers are possible. It turns out that the equation $z^n - 1 = 0$ has n roots in the complex plane. These roots are evenly spaced on a circle of radius 1 centered at the origin. Each of these roots is a finite attractor for the following Newton formula. The idea of the Newton formula is to guess the value of a root, plug this guess into the formula, and get a better guess out of the formula. By repeating this process, the root can be rapidly calculated with great accuracy. The fun starts if the initial guess is between two of the roots; then Newton's method suffers from indecision and the resulting chaos creates a great fractal.

Example:

```
t17064             { ; Jon Horner
  reset type=newton corners=-0.344/0.344/-0.25567/0.26033 params=6 float=y
  inside=-100 invert=0.1/0/0
  colors=000XZTPQP<2>5A2LAG<6>xC5<11>50P<8>9da<13>9Kv<4>000<7>IATIBTHCUHEUGFV<\
  6>CQ_CR_DTZMQ_<2>WMZ_LZbIb<5>z0z<14>'8TZ9QY9P<8>K9GI8FI8LJ8RK8X<6>dLk<14>i23\
j00i02<13>I9dGAgG9e<7>M6RN5P060<5>_EHaGFcHEeJCgKB<6>uT0qT2lS5<3>SLG<7>HDLFCL\
CCN<2>BGNBHNAHNCHM<12>l00<7>'NCZNDXMFVLH<5>2AR<7>ifU<9>svckn_dfX
}
```

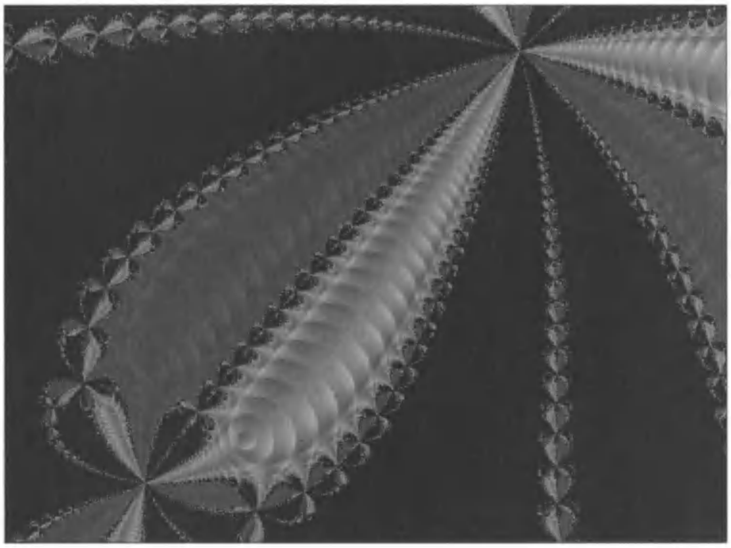
Formula: Initialize: $z = zpixel$

Iterate $z' = ((n-1)z^n + 1)/(nz^{n-1})$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
MPC math initialization	MPCjulia_per_pixel()	FRACTALS.C
MPC math orbit	MPCNewtonFractal()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	NewtonFractal2()	NEWTON.ASM

Newtbasin



Category: Escape Time to Finite Attractor

Fractal type newtbasin is identical to Newton except that the basin coloring scheme is used. A pixel is colored according to which attractor captures the orbit launched from that pixel's coordinates.

Example:

```
Genesis_Wave       { ; Darn! They have used Proto-Matter in the matrix.......
                     ; (c) 1992 Peter Moreland 100012,3213
  reset type=newtbasin
  corners=12.1762344559/12.176355783/12.1888035646/12.1890084717/12.1761797786\
  /12.1888764682 params=4 float=y maxiter=32000 decomp=256
  colors=000<56>000ix8<4>lgBlcCl'DlYDlVE<2>nLGoIHoEHoBI<15>cJQfZg<20>k_hk_hix8\
  <13>oEHoBInCJ<14>cJQ000<109>000
  }
```

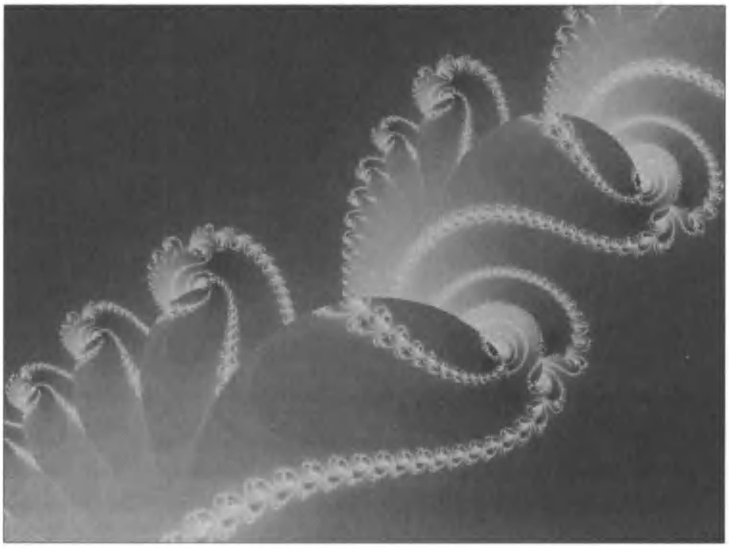
Formula: Initialize: $z = zpixel$

Iterate $z' = ((n-1)z^n + 1)/(nz^{n-1})$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
MPC math initialization	MPCjulia_per_pixel()	FRACTALS.C
MPC math orbit	MPCNewtonFractal()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	NewtonFractal2()	NEWTON.ASM

Complexnewton



Category: Escape Time to Finite Attractor

Type complexnewton generalizes type Newton in two ways. Instead of finding the roots of the polynomial $z^n - r = 0$ where n is an integer and r is 1, both n and r are allowed to be complex numbers. Complex functions involving noninteger exponents are fundamentally multiple valued, and the only way to make them single valued is to tear the function's graph somewhere. A multiple-valued function is like a spiral staircase—there is more than one stair above any point under the staircase. If you cut away all but one 360° turn of the staircase to make it single valued (only one stair above any point under the staircase), the ends of the staircase where you cut are discontinuous. The fractal generated by this formula has interesting discontinuities as a result. In true fractal fashion, these discontinuities are propagated throughout the image at all scales.

Example:

```
CMPNWT1.GIF        { ; Lee Skinner
  reset=1611 type=complexnewton passes=1
  corners=-7.425266/3.0421/3.319312/11.169836
  params=3/2.71828182845905/3.14159265358979/9 float=y inside=0
  logmap=yes periodicity=0 viewwindows=1.25/0.75/yes/0/0
  colors=0000A0<8>IDCLEEMDD<24>x11z00z10<29>zx0zz0zz1<29>zzxzzzzzz<61>zV1z\
  U0zU0zT0<28>z10z00z00y00<30>c00b11a11'22_22<19>KCC
  }
```

Formula:

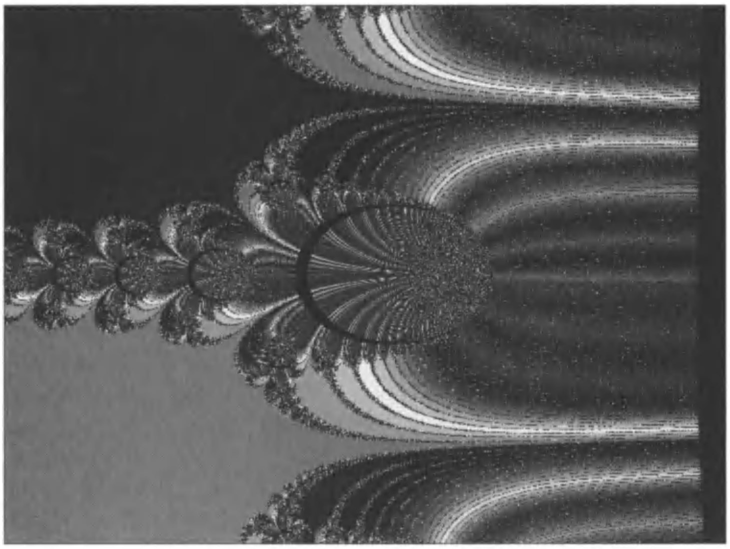
Initialize: $z = zpixel$

Iterate $z' = ((n-1)z^n + r)/(nz^{(n-1)})$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	ComplexNewton()	MPMATH_C.C

Complexbasin



Category: Escape Time to Finite Attractor

Fractal type newtbasin is identical to Newton except that the basin coloring scheme is used. A pixel is colored according to which attractor captures the orbit launched from that pixel's coordinates.

Example:

```
Jukebox            { ; 01:39:09.43
                     ; 03 jan 93 .. caren park
  reset type=complexbasin corners=-1.074297/-0.955225/-0.044652/0.044652
  params=111/0/111 float=y maxiter=500 inside=maxiter periodicity=0
  colors=0000i0<5>0A0<6>eXU<7>0A0AKK<6>mmK<7>AKMFF0<6>zz0<7>FF0F00<6>z0A<7>F00\
  FFF<6>zzz<7>FFF0FK<6>0Um<7>0FKPF0<6>mK0<7>PF0FKA<5>NSIPUK0TJ<6>FKA00K<4>57j7\
  9p78l<6>00K0D1<6>1aA<7>1F0KF1<6>hZ0<6>0I0KF0F0F<6>Z0U<7>F0FKF0<6>hZ0<7>KF000\
  A<6>00k<7>00AA00<6>z00<7>A00<6>2r00z00u00o0
  }
```

Formula:

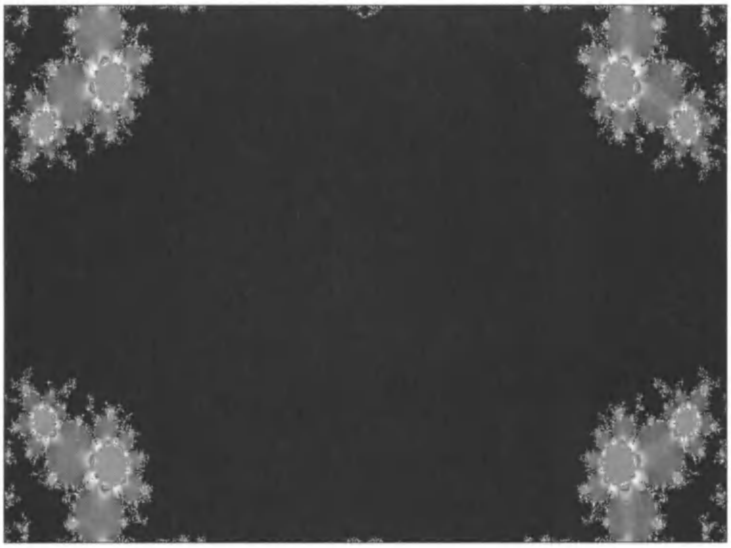
Initialize: $z = zpixel$

Iterate $z' = ((n-1)z^n + r)/(nz^{(n-1)})$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	ComplexBasin()	MPMATH_C.C

Halley



Category: Escape Time to Finite Attractor

Fractal type halley is very similar to type Newton, but uses a somewhat more complicated formula to converge to the roots of a polynomial.

Example:

```
Jewelry_1           {  ; Delicate Halley
                       ; By Dan Farmer
                       ; Halley example
  reset type=halley passes=b corners=-1.352/1.352/-0.773344/0.773344
  params=6/2/0.0001 float=y maxiter=256 inside=maxiter outside=real
  periodicity=0
  colors=00033322200OPFF<27>vcDwdDxeDzfCyfCxeC<29>PFF0A0<15>Zc0<14>0A0<15>\
  Zc0<15>0A0SNC<29>xo'zpaypa<29>UPF000<4>888000BBB<23>ppp<28>555
  }
```

Formula:

Initialize: $z = zpixel$

Iterate: $z' = z - R\,F / [F' - (F''F / (2F'))]$

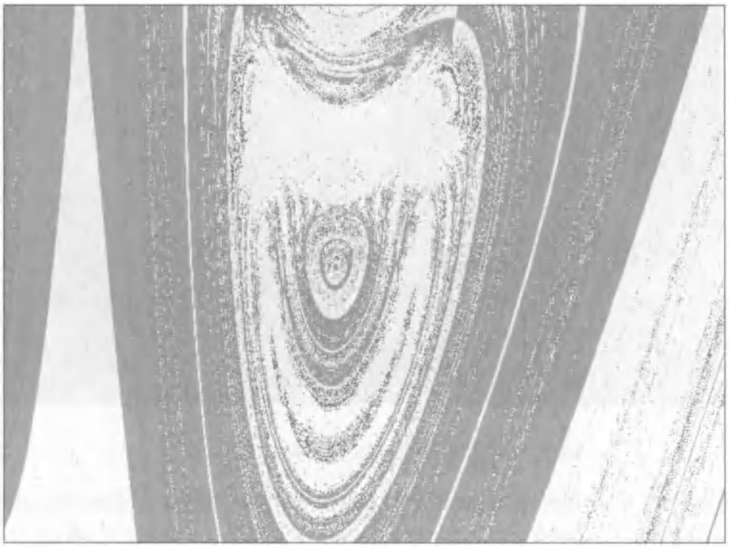
where $F = z(z^a - 1)$, F' and F'' are first and second derivatives of F, $F' = (a + 1)(z^a - 1)$, and $F'' = (a + 1)az^{a-1}$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
MPC math initialization	MPCHalley_per_pixel()	FRACTALS.C

MPC math orbit	MPCHalleyFractal()	FRACTALS.C
Floating point initialization	Halley_per_pixel()	FRACTALS.C
Floating point orbit	HalleyFractal()	FRACTALS.C

Frothybasin



Category: Escape Time to Finite Attractor

Frothy Basins, or Riddled Basins, were discovered by James C. Alexander of the University of Maryland. This type is interesting because the attractors are strange attractors that intersect. (Roughly speaking, a strange attractor is an attractor that is a fractal.)

Example:

```
EvilFrog           { ; Kermit with rabies                    Wesley Loewer
  reset type=frothybasin passes=1
  corners=0.1987794025371462/0.1987794025306378/-1.287290394040766/-1.2872\
  90394029171/0.1987794025292465/-1.287290394030193 params=6/1 float=y
  maxiter=100
  colors=000z00<82>M00z0z<40>M0M0z0<40>0M000z<40>00Mzz0<40>MM000000000
  }
```

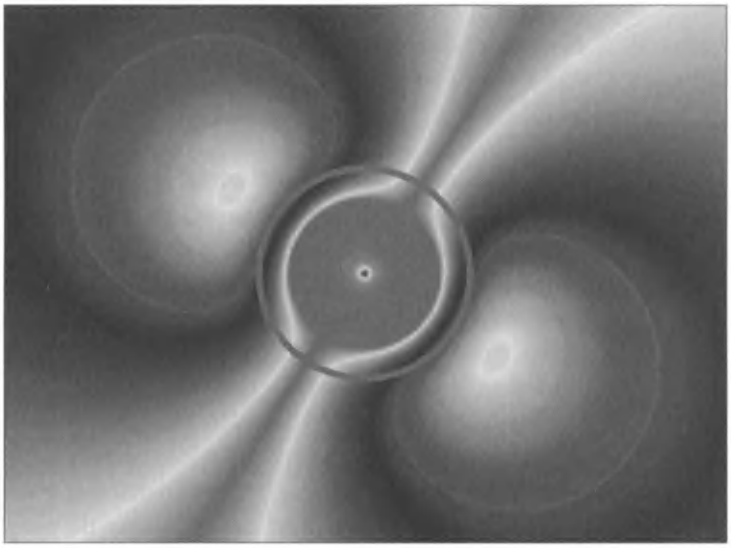
Formula: Initialize: $z = zpixel$

Iterate: $z' = z^2 - c\,\mathrm{conj}(z)$

where $c = 1 + ai$, and $a = 1.02871376822\ldots$

Code:	**Routine Type**	**Routine Name**	**File**
	Fractal engine	calcfroth()	MISCFRACT.C
	Initialization	froth_setup()	MISCFRACT.C

Unity



Category: Escape Time to Finite Attractor

This intriguing fractal is based on a formula trying desperately to converge to the number one. This formula was proposed by Mark Peterson.

Example:

```
Lo_Iteration_Unity { ; (C)1993 Peter Moreland 100012,3213
                     ; Thanks to Lee Skinner for the palette
  reset=1732 type=unity passes=t
  corners=-31.914182/31.914182/-2.438782/2.438782/-11.277216/-29.954578
  float=y maxiter=2 inside=bof60 invert=7.47995/0/0 distest=3/5
  periodicity=0
  colors=MgCMgC<19>zz9<60>2I10H00I1<8>4WH<4>MU'YUzUTh<6>c0heNge0g<25>soltp\
  luqlvrlwslxumyvmzwn<47>C2W<8>V6aY7bZ9'<14>zg0<13>nGDmDEmCDmBDmACl8B<2>r6\
  C<15>IfCKfC
  }
```

Formula: Initialize: $x = zpixel_x, y = zpixel_y,$

Iterate: $One = x^2 + y^2$

$y' = (2 - One)\,x;$

$x' = (2 - One)y';$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_julia_per_pixel()	FRACTALS.C
Integer math orbit	UnityFractal()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	UnityfpFractal()	FRACTALS.C

3-D Solid Fractals

Since Fractint version 12, the fractal type julibrot has implemented what is known in fractal literature as "stacked Julias." The idea is to layer Julia sets one on top of the other, changing the c parameter in the Julia $z^2 + c$ formula continuously in the third dimension. The result is a hauntingly beautiful crystal-like solid. This solid is rendered by coloring each pixel according to the distance from a virtual viewer, with the foreground surface lighter, and receding surfaces shaded an increasingly darker gray. Both grayscale and red/blue stereo images can be made of julibrot crystals. Note that a 256-color video mode must be used.

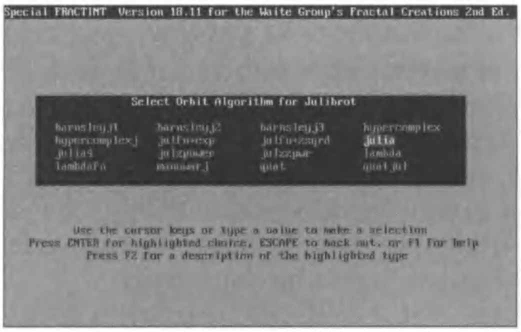
Starting with Fractint version 18 (the version included with this book), the julibrot type has been generalized in two different ways. First, stacked Julias can now be made using orbit routines other than the classic Julia. You can use the julibrot type to stack 2-D fractals from types barnsleyj1, barnsleyj2, barnsleyj3, julfn+exp, julfn+zsqrd, julia, julia4, julzpower, julzzpwr, lambda, lambdafn, and manowarj. Second, the julibrot rendering mechanism can now be used to make 3-D "cross sections" of the 4-D fractal types quat, quatj, hypercomplex, and hypercomplexj.

Julibrot Parameters

To access the fractal type julibrot, press (T), select JULIBROT from the SELECT FRACTAL TYPE screen, and press (ENTER). You will then see the SELECT ORBIT ALGORITHM FOR JULIBROT screen, as shown in Figure 6-2.

Orbit Algorithms

The default orbit algorithm is Julia, which results in the original "Julibrot" image of past Fractint versions. This screen works exactly like the SELECT FRACTAL TYPE screen. Select an orbit algorithm and press (ENTER).



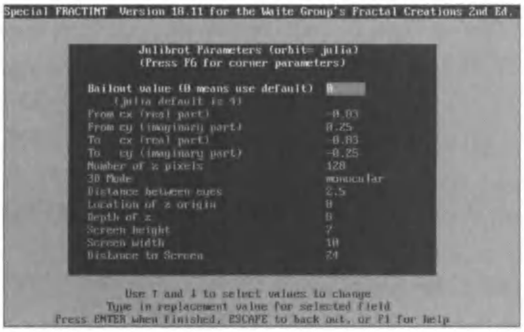
```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

                 Select Orbit Algorithm for Julibrot

    barnsleyj1      barnsleyj2      barnsleyj3      hypercomplex
    hypercomplexj   julfn+exp       julfn+zsqrd     julia
    julia4          julzpower       julzzpwr        lambda
    lambdafn        manowarj        quat            quatjul

            Use the cursor keys or type a value to make a selection
    Press ENTER for highlighted choice, ESCAPE to back out, or F1 for help
              Press F2 for a description of the highlighted type
```

Figure 6-2 Select Orbit Algorithm for Julibrot screen



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

               Julibrot Parameters (orbit= julia)
               (Press F6 for corner parameters)

       Bailout value (0 means use default)  0
           (julia default is 4)
       From cx (real part)                  -0.83
       From cy (imaginary part)             0.25
       To   cx (real part)                  -0.83
       To   cy (imaginary part)             -0.25
       Number of z pixels                   128
       3D Mode                              monocular
       Distance between eyes                2.5
       Location of z origin                 0
       Depth of z                           8
       Screen height                        7
       Screen width                         10
       Distance to Screen                   24

              Use ↑ and ↓ to select values to change
             Type in replacement value for selected field
       Press ENTER when finished, ESCAPE to back out, or F1 for help
```

Figure 6-3 Julibrot Parameters screen

Orbit Fractal Type Parameters

The Julibrot Parameters screen is shown in Figure 6-3. Some of the details of this screen vary depending on which orbit algorithm you selected, so we'll discuss various cases individually. In all cases, the formulas for the orbit algorithms are in the form $z' = f(z) + c$ for some function $g(z)$. When we mention the variable c, we are referring to the c in this formula.

2-D Julia Case Let's assume that you selected one of the 2-D Julia orbits (barnsleyj1, barnsleyj2, barnsleyj3, julfn+exp, julfn+zsqrd, julia, julia4, julzpower, julzzpwr, lambda, lambdafn, or manowarj). The complex coordinates corresponding to each pixel of your computer screen are used to initialize z to start each orbit calculation.

The first few items of the Julibrot Parameters are parameters for the orbit algorithm you selected. All the parameters that you would normally see for the orbit algorithm's fractral type will be shown on the screen, except parameters that the Julibrot renderer uses to calculate the third dimension. For the 2-D Julia orbits, the parameters not shown are the real and imaginary parts of c. Other parameters, such as the bailout value and the function variable parameters of type julfn+zsqrd, are shown and may be entered.

Following the orbit algorithm parameters are four entries labeled "from" and "to." The first two of the three dimensions of a Julibrot image are mapped to the screen pixels in the same way as is done for 2-D Julia sets. Imagine a stack of Julia sets parallel to your computer screen, some closer to you, some farther away. The third dimension (perpendicular to your screen) is controlled by changing the variable c continuously from the From cx/From cy value (near the viewer) to the To cx/To cy value (farther from the viewer). Each value of (cx,cy) determines a 2-D Julia set that is a cross section of the Julibrot image parallel to your computer screen.

4-D Julia case Now let's consider the 4-D Julia orbit algorithms quatjul and hypercomplexj. The variables *z* and *c* in the orbit formula are four dimensional. As with 2-D Julias, the coordinates corresponding to each pixel of your computer screen are used to initialize *z* to start each orbit calculation, but because *z* is four dimensional, there are two dimensions of *z* undetermined. These are determined by the From/To settings of the JULIBROT PARAMETERS screen. Because the four components of *c* are not required to form the third dimension as in the 2-D case, these parameters appear at the top of the JULIBROT PARAMETERS screen.

The main difference between using 2-D and 4-D orbits to generate a Julibrot image is that in the 2-D case many different Julia sets are stacked together to form a 3-D solid, whereas in the 4-D case the solid is formed from a single 4-D Julia set.

4-D Mandelbrot Case Finally, we'll look at the 4-D Julia set orbit algorithms quat and hypercomplex. The coordinates corresponding to each pixel of your computer screen are used to set *c* rather than initialize *z* as was the case in the Julia example. Because *c* is four dimensional, there are two dimensions of *c* undetermined. These are set by the From/To settings of the JULIBROT PARAMETERS screen.

3-D Control Parameters

The remaining parameters on the JULIBROT PARAMETERS screen allow you to control the way the 3-D transformations are set up. The julibrot type uses its own 3-D mechanisms—they do not interact with Fractint's other 3-D settings. Most of these settings never need to be changed.

Number of z pixels This parameter sets how many slices are used to build up the solid image. The default value of 128 is good for most images. A higher number gives finer results, but you will discover that much higher values than 128 have dimishing returns of better quality and exact a high price of increased calculation time. For your final show-quality penultimate image, you might want to increase this to 256 or more. Conversely, because the julibrot takes a long time to generate, you can use lower values to explore. The calculation time is proportional to this setting; a value of 256 will take twice the time as a setting of 128.

3D mode The possible 3-D modes are monocular, left eye, right eye, and red-blue. The first makes a grayscale image. The red-blue mode makes a stereo anaglyph that you can view with the red/blue glasses that came with this book. The left eye and right eye modes produce images that look similar to images made in the monocular mode, but the perspective shifts to the left and right, allowing you to make stereo slides or use other methods of combining the images into a stereo form.

Distance between eyes This number controls the stereo effect. A slightly higher number enhances the stereo separation, but may make the images more difficult to fuse. This parameter is ignored in monocular mode.

Reference Frame Parameters The remainder of the parameters are needed to construct the 3-D picture so the fractal appears with the desired depth and proper *z* location. With the origin set to 8 inches beyond the screen plane and the depth of the fractal at 8 inches, the default fractal will appear to start at 4 inches beyond the screen and extend to 12 inches if your eyes are 2.5 inches apart and located at a distance of 24 inches from the screen. The screen dimensions provide the reference frame.

Tricks for Exploring Julibrots

You can use the 2-D fractal types corresponding to the orbit algorithm to help you set up julibrot images. Keep in mind that the interior "lake" of the 2-D cross sections is used to build the julibrot solids. Use the X BASIC OPTIONS screen and set inside color to 8 and outset color to 0. These settings show off the "lake" area without the distracting escape-time bands. Find a range of parameters that start with a very small lake area, move through a larger lake, and change back to a small lake. Use the end points of this range as your two From/To pairs.

Now get your thinking caps on. It may take a while for this to sink in; but when it does, you'll experience the pleasure of real mathematical insight and enjoy how all these topics fit together. Recall that the Mandelbrot set is an index of of Julia sets. The easiest way to find a series of Julia sets that stack together to make a julibrot is to use a Mandelbrot set. Generate an image of the Mandelbrot variant of the orbit algorithm you want to use. For example, if you want to make a julibrot using type barnsleyj1, generate a barnsleym1 image. (Press T, select barnsleym1, and press ENTER.) A low resolution video mode is fine; try F3. When the image is complete, press O to turn on the cross-hair cursor. Then press N (for "numbers") and the coordinates of the cursor will appear on the screen. Move the cursor just "on shore" and write down the coordinates (you don't need to write down all the digits—the precision is not critical). Then move the cursor across to the other side of the lake to a position on the opposite shore and write down the coordinates of that point. The two sets of numbers you wrote down are the From/To julibrot settings. Then press T, select type julibrot, select the barnsleyj1 orbit algorithm. Enter the recorded coordinates—the first pair as the FROM CX and FROM CY values, and the second as the TO CX and TO CY values. Go ahead and generate the julibrot image. (Because julibrot is slow, you can use V

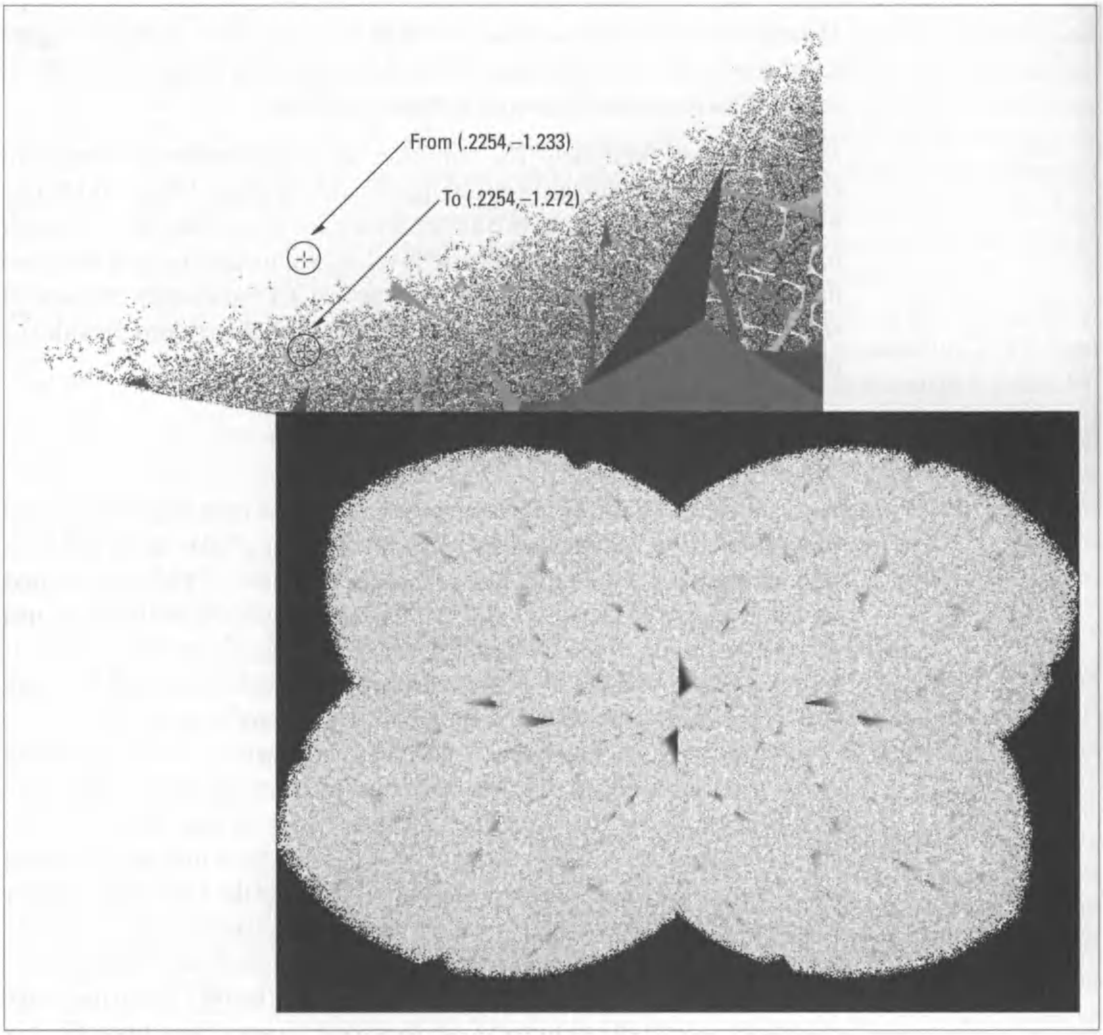
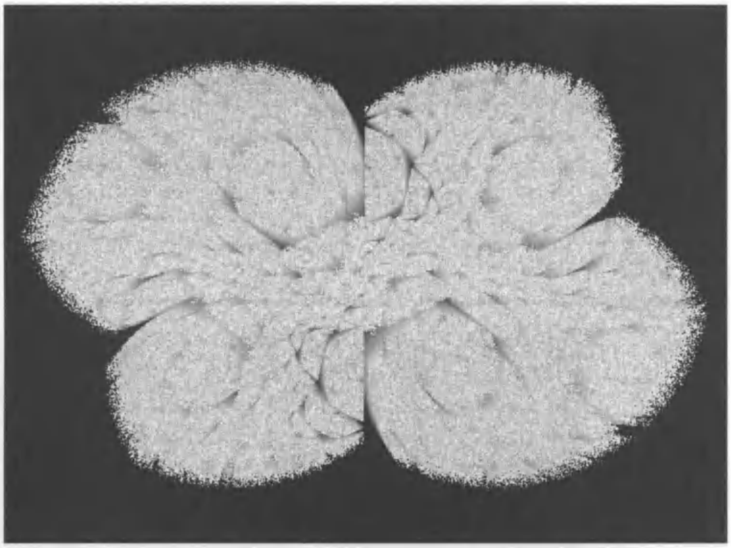


Figure 6-4 Exploring Julibrots using 2-D cross sections

to turn on the view windows feature and make a small image to speed things up.) Figure 6-4 shows a barnsleym1 image with the From and To values marked, a series of Julia cross sections, and the resulting Julibrot image. For the image in the figure, the "From" is just a bit off shore, and the To point is a bit onshore. The reason for this was to magnify the texture of the surface. If the From/To points had spanned the whole lake, the surface texture would have been such a small percentage of the total depth as to be invisible.

Julibrot



Category: 3-D Solid Fractals

Fractint's "poor person's ray tracer"—a solid fractal renderer that lets you see stacked Julia sets as well as true 3-D fractals. Note that a 256-color video mode is needed. When exploring, use the View Windows mode (V) to speed up what can be a very slow fractal type.

Example:

```
Barnsley_Sponge    { ; Tim Wegner
  reset=1733 type=julibrot julibrotfromto=0.45/0.55/1.1/1.1
  julibroteyes=2.5 orbitname=barnsleyj1 3Dmode=red-blue
  corners=-2.336449/2.336452/-1.752338/1.752337 float=y
  colors=@glasses1.map
  }
```

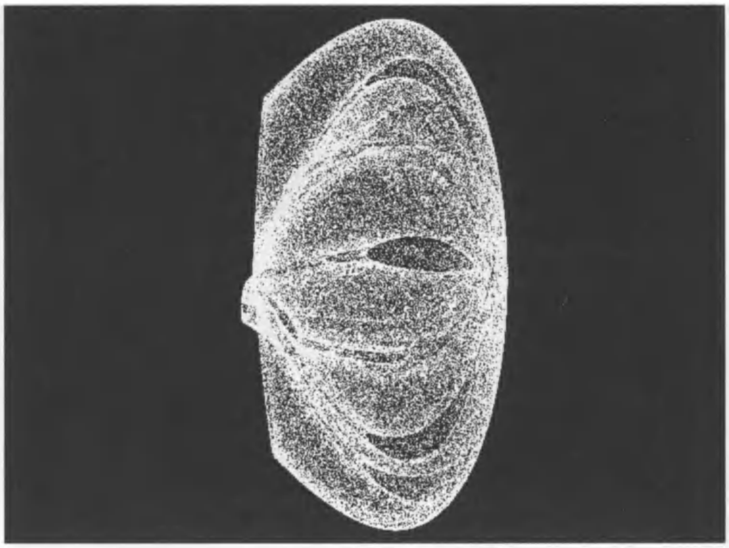
Code:

Routine Type	Routine Name	File
Fractal engine	Std4dFractal()	JULIBROT.C
Integer math initialization	jb_per_pixel()	JULIBROT.C
Floating point initialization	jbfp_per_pixel()	JULIBROT.C

3-D Orbit Fractals

These fractals trace out orbit paths in three dimensions. You can alter the point of view, perspective, and scale using the (I) command. Try viewing them using the stereo method with the superimpose method. (Enter 2 at the number 2—the STEREO (R/B 3D)? (0=NO,1=ALTERNATE,2=SUPERIMPOSE,3=PHOTO) prompt.) Orbit fractals generally benefit from higher resolution video modes; if your graphics hardware supports it, try a 1024 x 768 mode.

Icons3d



Category: 3-D Orbit Fractal

The icons3D type, which was inspired by the book *Symmetry in Chaos* by Michael Field and Martin Golubitsky, produces some of the most pleasing red/blue stereo images you can make with Fractint. The images have the appearance of a delicate but highly structured web. The formula implemented here maps the classic population logistic map onto the complex plane and is, therefore, the distant cousin of the bifurcation fractals mentioned later is this chapter, but with an entirely different result.

Example:

```
Sand_Dollar_3d_II  { ; Rotated view
  reset type=icons3d corners=-2.134489/2.036423/-1.519231/1.608953
  params=-2.34/2/0.2/0.1/0.0/5 maxiter=32767 inside=0
  rotation=90/45/270 perspective=180 xyshift=0/0 stereo=2 interocular=3
  converge=-3 crop=4/0/0/4 bright=100/100 colors=@glasses2.map
  }
```

Formula:

Intitialize: $x = 0.01$
$y = 0.003$

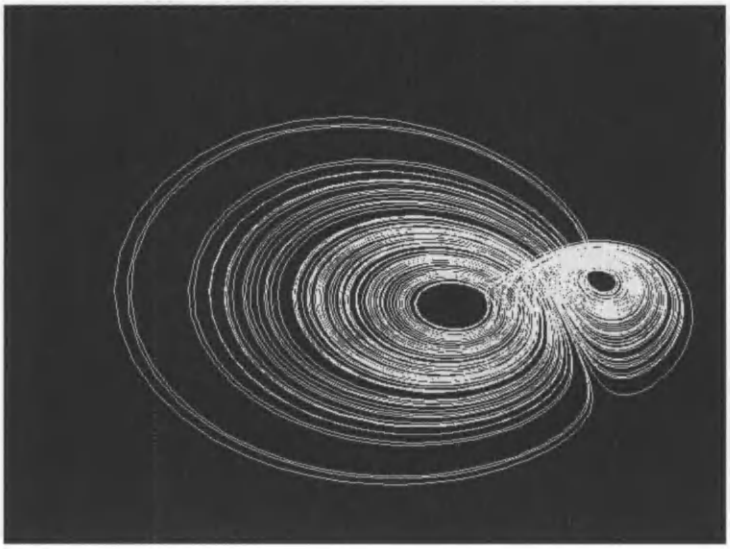
Iterate: $p = lambda + alpha(x^2 + y^2) + beta\,(x\,z_{real} - y\,z_{imag})$
$x' = p\,x + gamma\,z_{real} - omega\,y$
$y' = p\,y - gamma\,z_{imag} + omega\,x$
$z' = x^2 + y^2$

Parameters: Lambda, Alpha, Beta, Gamma, Omega, and Degree

Code:

Routine Type	Routine Name	File
2D floating point fractal engine	orbit2dfloat()	LORENZ.C
3D floating point fractal engine	orbit3dfloat()	LORENZ.C
Floating point orbit	iconfloatorbit()	LORENZ.C
Floating point orbit	lorenz3d1floatorbit()	LORENZ.C
Floating point orbit	lorenz3d3floatorbit()	LORENZ.C
Floating point orbit	lorenz3d4floatorbit()	LORENZ.C
Integer math orbit	lorenz3dlongorbit()	LORENZ.C

Lorenz3d



Category: 3-D Orbit Fractal

The Lorenz Attractor fractal is based on a simple set of three deterministic equations developed by Edward Lorenz while studying the nonrepeatability of weather patterns. The orbit forms two saucer-like disks at an angle to each other in three dimensions. The orbit filaments come arbitrarily close to each other but never touch. The orbit is torn between the two disks, first spinning about one, then the other.

Example:

```
Lorenz_two_lobe    {
  reset=1733 type=lorenz3d
  corners=-100.4318/26.56676/-63.80547/63.18771
  params=0.02/5/15/1/0/3 maxiter=300
  rotation=34/72/30 perspective=150 xyshift=0/0 stereo=2 interocular=2
  converge=-3 crop=4/0/0/4 bright=80/100 colors=@glasses2.map
  }
```

Formula:

Intitialize: $x' = x - (ax + ay)dt$

$z = y = z = 1$

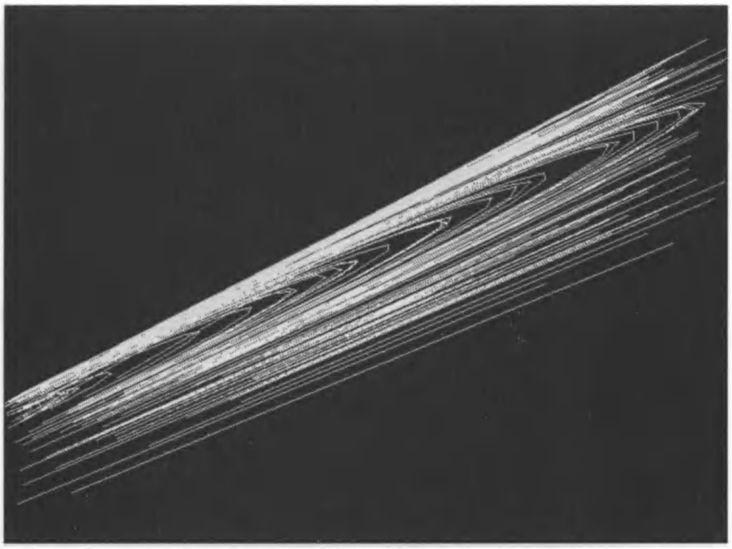
Iterate: $y' = y + (bx - y - zx)dt$

$z' = z - (cz + xy)dt$

Code:

Routine Type	Routine Name	File
3D floating point fractal engine	orbit3dfloat()	LORENZ.C
3D integer math fractal engine	orbit3dlong()	LORENZ.C
Floating point orbit	lorenz3dfloatorbit()	LORENZ.C
Floating point orbit	lorenz3dlongorbit()	LORENZ.C

Lorenz3d1



Category: 3-D Orbit Fractal

This type is a one-lobe variation of lorenz3d.

Example:

```
Lorenz_one_lobe    {
  reset=1733 type=lorenz3d1
  corners=2.181638/7.261645/-24.08926/-19.00947
  params=0.02/5/15/1/0/3 float=y maxiter=300
  rotation=34/72/30 perspective=150 xyshift=0/0 stereo=2 interocular=2
  converge=-3 crop=4/0/0/4 bright=80/100 colors=@glasses2.map
  }
```

Formula: Initialize: $n = (x^2 + y^2)$

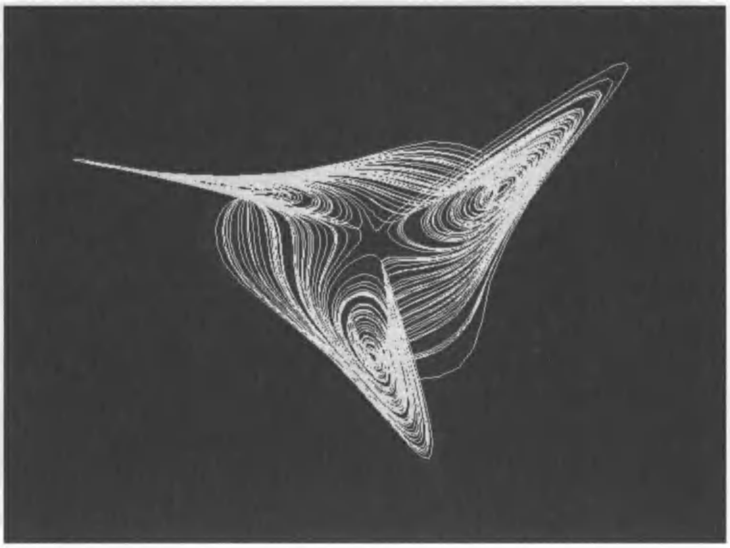
$z = y = z = 1$

Iterate: $x' = x + (-ax - x + ay - by + n - an + yz)dt$
$y' = y + (bx - ax - ay - y + bn + an - xz - nz)dt$
$z' = z + (y/2 - cz)dt$

Code:

Routine Type	Routine Name	File
3D floating point fractal engine	orbit3dfloat()	LORENZ.C
Floating point orbit	lorenz3d1floatorbit()	LORENZ.C

Lorenz3d3



Category: 3-D Orbit Fractal

This type is a three-lobe variation of lorenz3d.

Example:

```
lorenz_three_lobe  {
  reset type=lorenz3d3 passes=t corners=-20.28/20.28/-20.28/20.28
  params=0.02000000000000000416/10/28/2.660000000000001421 float=y
  maxiter=200 fillcolor=1 inside=bof60 outside=0 rotation=0/0/0
  perspective=150 xyshift=0/0 stereo=2 interocular=3 converge=-4
  crop=4/0/0/4 bright=80/100 colors=@glasses2.map
  }
```

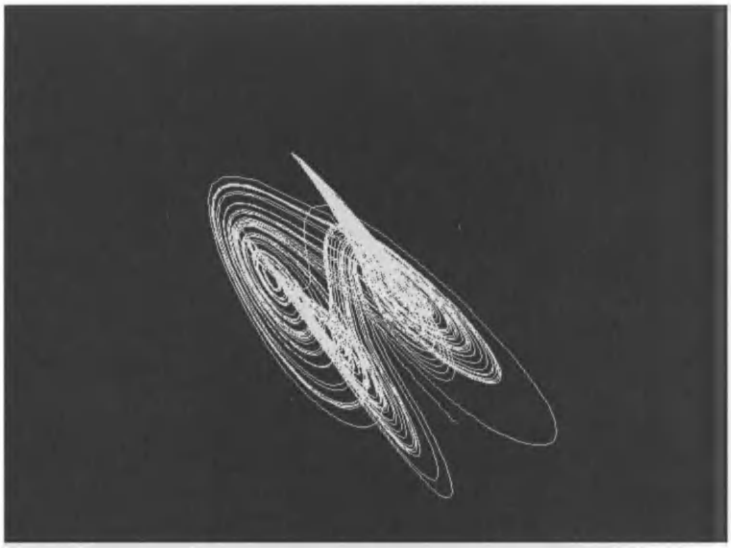
Formula: Initialize: $n = (x^2 + y^2)$
$z = y = z = 1$

Iterate: $x' = x + ((-ax - x + ay - by + yz)/3 + n^2 - an^2(2xyn/3)(b + a - z))dt$
$y' = y + ((bx - ax- zx - ay - y)/3+ 2axy - 2xy + (b + a - z)(x^2 - y^2)/3n)dt$
$z' = z +(3x^2y - y^3)/2 - cz)dt$

Code:

Routine Type	**Routine Name**	**File**
3D floating point fractal engine	orbit3dfloat()	LORENZ.C
Floating point orbit	lorenz3d3floatorbit()	LORENZ.C

Lorenz3d4



Category: 3-D Orbit Fractal

This type is a one-lobe variation of lorenz3d.

Example:

```
Lorenz_four_lobe    {
  reset=1733 type=lorenz3d4 corners=-28.75506/31.99/-31.99/28.75506
  params=0.02/10/28/2.66/0/3
  float=y maxiter=300 rotation=10/20/30 perspective=150 xyshift=0/0
  stereo=2 interocular=2 converge=-3 crop=4/0/0/4 bright=80/100
  colors=@glasses2.map
  }
```

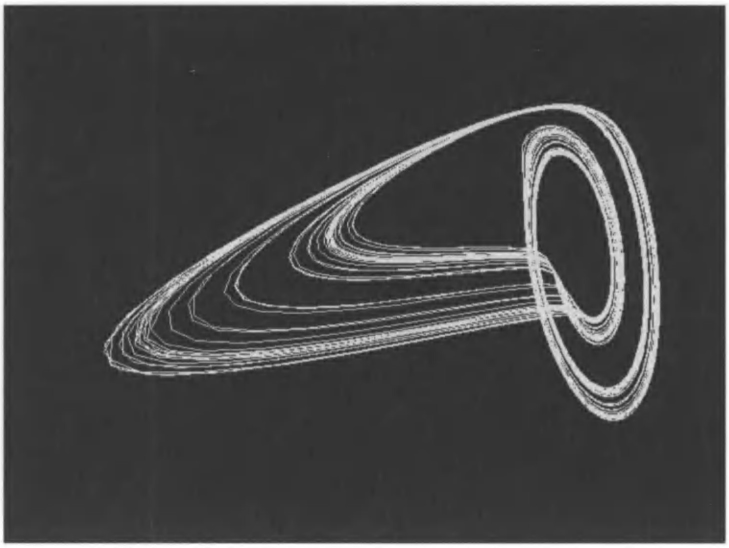
Formula: Initialize: $n = (x^2 + y^2)$
$z = y = z = 1$

Iterate: $x' = x + ((-ax^3 + (2a + b - z)x^2y + axy^2 - 2xy^2 + (zy^3 - by^3)/(2x^2 + 2y^2))dt$
$y' = y + ((bx^3 - zx^3 + ax^2y - 2x^2y + (-2a - b + z)xy^2 - ay^3/(2x^2 + 2y^2))dt$
$z' = z + (2x^3y - 2xy^3 - cz)dt$

Code:

Routine Type	**Routine Name**	**File**
3D floating point fractal engine	orbit3dfloat()	LORENZ.C
Floating point orbit	lorenz3d4floatorbit()	LORENZ.C

Rossler3d



Category: 3-D Orbit Fractal

This strange attractor is named after the German Otto Rossler. His fractal namesake looks like a band of ribbon with a fold in it.

Example:

```
Rossler3d          {
  reset=1733 type=rossler3d corners=-30/30.00375/-19.99925/40
  params=0.04/0.2/0.2
  11/5.7 rotation=60/30/30 perspective=150
  xyshift=0/0 stereo=2 interocular=1 converge=-6 crop=4/0/0/4
  bright=80/100 colors=@glasses2.map
  }
```

Formula:

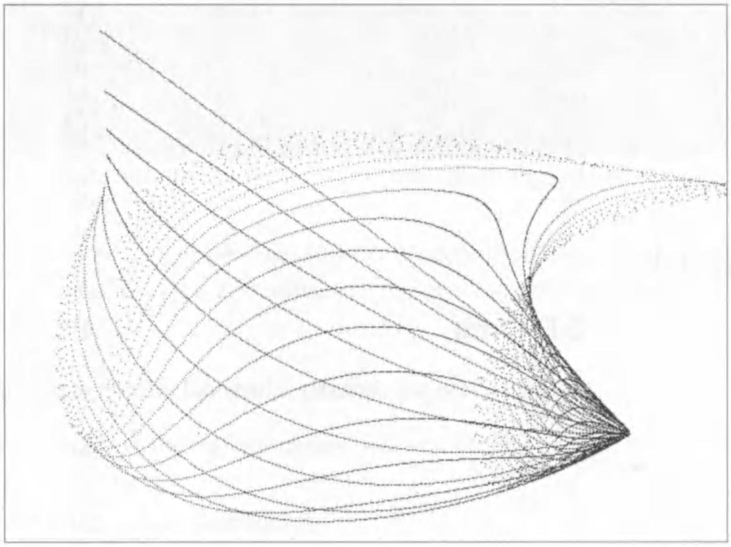
Initialize: $x = y = z = 1$

Iterate:
$$x' = x - (y - x)dt$$
$$y' = y + (x + ay)dt$$
$$z' = z + (b + xz - cz)dt$$

Code:

Routine Type	Routine Name	File
Floating point fractal engine	orbit3dfloat()	LORENZ.C
Floating point orbit	rosslerlongorbit()	LORENZ.C
Integer math fractal engine	orbit3dlong()	LORENZ.C
Integer math orbit	rosslerfloatorbit()	LORENZ.C

Kamtorus3d



Category: 3-D Orbit Fractal

This fractal is a variation on the orbits theme. The fractal is not a single orbit but the superimposition of many with different initializers.

Example:

```
pod034            { ; Par of the day 11-15-92, Dick Sherry, 76264,752
                    ; (c) 1992
  reset type=kamtorus3d
  corners=0.103714/-0.102463/-0.372742/2.21312/-0.979996/1.762589
params=0.33/0.005/200/50 maxiter=500 inside=0 logmap=yes periodicity=0
rotation=60/30/0 perspective=0 xyshift=0/0
colors=000IVUNWdR9ejk0<2>W_bs6IeJZaYYXXfibQc_ZYYgvHUfPdK'4<2>QXcdGSeC6'JMWQ'\
FL7MRUlpPaebc7V<2>VQj3cC<2>MYeEsGLgY1eZ71GIHYxltmfrb'pKEEowhgmk_dmG8TMKd9RRG\
T_MVgEomHRJNU_9iURJVIHUMMaPRhDeLIaWNZeD7cIGgN0k0m'<2>L_lq31<2>YPcastaT_YUeVV\
jvsPkkYacfgVb_WitrPjjY'bfYjgWejU'mP6BQFPR0bKcR0kzQctfVWZWe9G7J0UQ4rRIpb40XIQ\
XpvUersRiiTk'Vm9p1JeRjRG<2>WVfmEnYnC<2>T_e'KztVPkP7aTUmYmbXn53KHI_'rN<2>U'hf\
DV<2>VSjKbD<2>QXfchK_cVW_e2xj7_YGlz<2>P_qEqRJi_0bgbm4ZgKVa_MDB<2>RSeXNnVQoTT\
oHte<2>Qam8yJjd4<2>WYcCDvIKsNQquwukmsadqIW8NWU54xDEuLNrvT'nUdfVhZWlTf0SbYSZf\
0hN<2>LZh374HXH<2>QWgkDvdKsYQqr1KdH_UpT<2>S'j6tFHgYVzPlZL'aTPd_fljwtu'BAcXQe\
qdnDXlKZkR_<2>FBU89H2744FG6NR7UaBgkFtt'rXuq9mlD<2>lLJlCLi7dg3w<3>fcivHddE2<2\
>VScMkG<2>R_f8U7DVJ
}
```

Formula:

Initialize: $x = y = 0$

Iterate: $x' = x\cos(a) + (x^2 - y)\sin(a)$
$y' = x\sin(a) - (x^2 - y)\cos(a)$

Code:

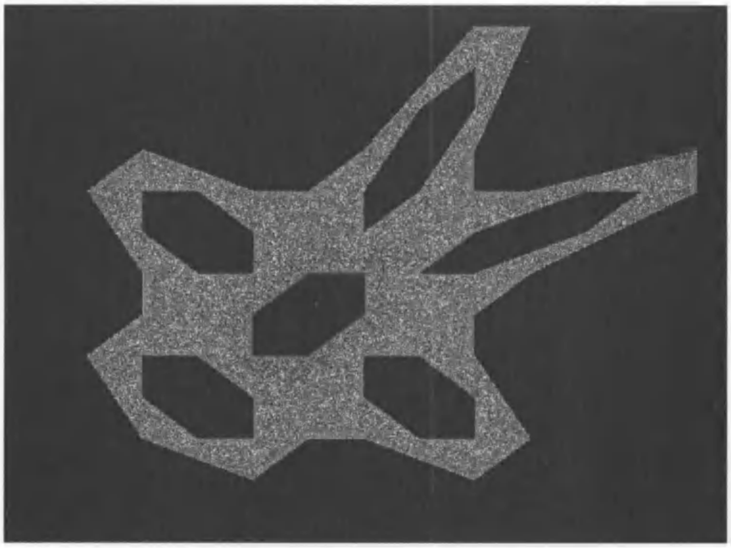
Routine Type	Routine Name	File
3D floating point fractal engine	orbit3dfloat()	LORENZ.C
3D integer math fractal engine	orbit3dlong()	LORENZ.C
Floating point orbit	kamtorusfloatorbit()	LORENZ.C
Integer math orbit	kamtoruslongorbit()	LORENZ.C

Orbit Fractals

2-D Orbit

In this section we consider additional orbit Fractals that create 2-D images.

Gingerbreadman



Category: 2-D Orbit

This intriguing orbit fractal looks like a gingerbread man cookie. While the fractal is generating, new structures will suddenly appear after periods of apparent stability.

Example:

```
gingerbreadman {
  reset colors=@default.map type=gingerbreadman maxiter=20000
  }
```

Formula:

Initialize: x,y = parameters

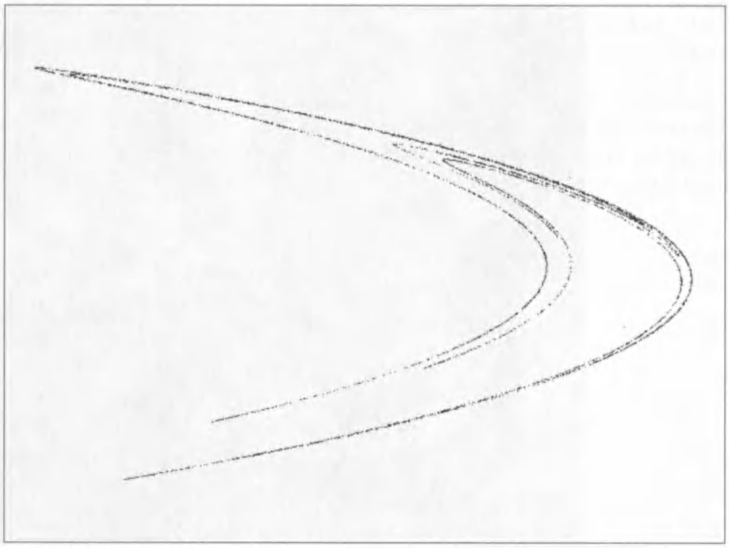
Iterate: $x' = 1 - y + |x|$

$y' = x$

Code:

Routine Type	Routine Name	File
Fractal engine	orbit2dfloat()	LORENZ.C
Floating point orbit	gingerbreadfloatorbit()	LORENZ.C

Henon



Category: 2-D Orbit

This fractal type was discovered by Michel Henon, an astronomer at Nice. The shape is made up of thicker and thinner parts, and is more complex than it first appears. This pattern continues even at the most extreme magnifications.

Example:

```
henon            {
  reset=1733 type=henon corners=-1.399994/1.398849/-0.49900818/0.5
  params=1.4/0.3
  }
```

Formula:

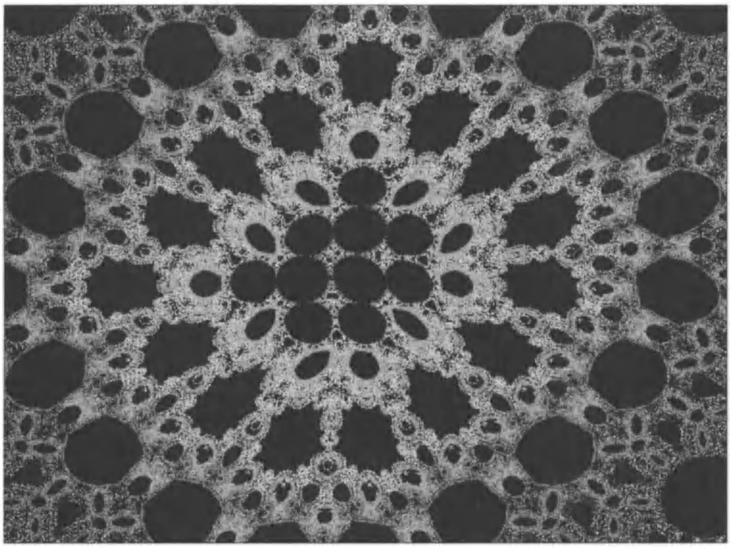
Initialize: $x = y = 1$

Iterate: $x' = 1 + y - ax^2$
$y' = bx$

Code:

Routine Type	Routine Name	File
Floating point fractal engine	orbit2dfloat()	LORENZ.C
Floating point orbit	henonlongorbit()	LORENZ.C
Integer math fractal engine	orbit2dlong()	LORENZ.C
Integer math orbit	henonfloatorbit()	LORENZ.C

Hopalong



Category: 2-D Orbit

This fractal is due to Barry Martin. It looks like a ribbon tied in a bow, and it develops in sudden leaps after periods of quiescence.

Example:

```
NETWORK          { ; A hopalong lacey little number Peter Moreland 100012,3213
  reset type=hopalong corners=-24.694914/29.486391/-22.866557/23.187553
  params=1.666/0.556/4.6 float=y maxiter=32767 bailout=300 inside=0
  logmap=yes
  colors=00000e0e00eee00e0eeL0eeeLLLLLzLzLLzzzLLzLzzzLzzz000555<3>HHHKKKOOOSSS\
  WWW___ccchhhmmmssszzz00z<3>z0z<3>z00<3>zz0<3>0z0<3>0zz<2>0GzVVz<3>zVz<3>zVV<\
  3>zzV<3>VzV<3>Vzz<2>Vbzhhz<3>zhz<3>zhh<3>zzh<3>hzh<3>hzz<2>hlz00S<3>S0S<3>S0\
  0<3>SS0<3>0S0<3>0SS<2>07SEES<3>SES<3>SEE<3>SSE<3>ESE<3>ESS<2>EHSKKS<2>QKSSKS\
  SKQSKOSKMSKK<2>SQKSSKQSKOSKMSKKSK<2>KSQKSSKQSKOSKMS00G<3>G0G<3>G00<3>GG0<3>0\
  G0<3>0GG<2>04G88G<2>E8GG8GG8EG8CG8AG88<2>GE8GG8EG8CG8AG88G8<2>8GE8GG8EG8CG8A\
  GBBG<2>FBGGBGGBFGBDGBCGBB<2>GFBGGBFGBDGBCGBBGB<2>BGFBGGBFGBDGBCGkkk<5>343642
  }
```

Formula:

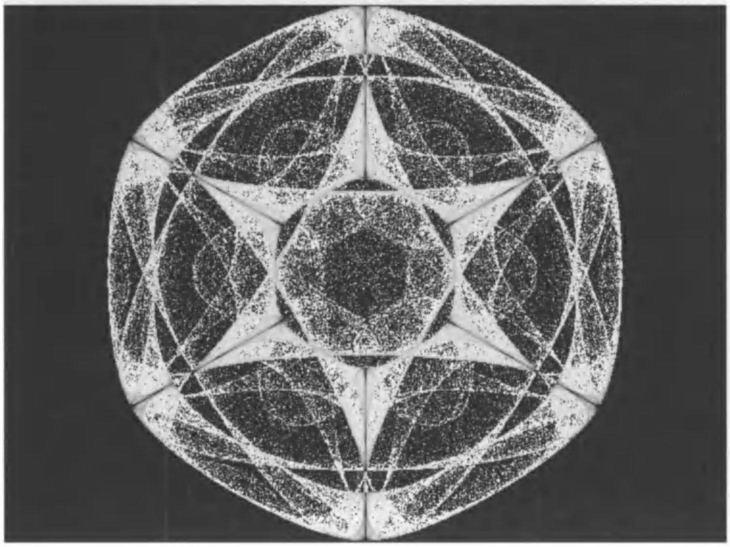
Initialize: $x = y = 0$:

Iterate: $x' = y - \text{sign}(x)\,(\text{abs}(\,bx - c))$
$y' = a - x$

(The function "sign()" returns 1 if the argument is positive, –1 if argument is negative.)

Code:	**Routine Type**	**Routine Name**	**File**
	Floating point fractal engine	orbit2dfloat()	LORENZ.C
	Floating point orbit	hopalong2dfloatorbit()	LORENZ.C

Icons



Category: 2-D Orbit

The 2-D form of ICONS3D.

Example:

```
Halloween          {
  reset type=icons corners=-1.0496/1.0496/-0.7872/0.7872
  params=-2.7/5/1.5/1/0.0/6 float=y inside=0
  colors=000TgsSfsSestYS<3>zvErMY<39>fZEqMY<32>FUyrMY<62>0V4rMYqNXqNW
  }
```

Formula:

Intitialize: $x = 0.01$

$y = 0.003$

Iterate: $p = lambda + alpha(x^2 + y^2) + beta\,(x\, z_{real} - y\, z_{imag})$

$x' = p\,x + gamma\; z_{real} - omega\; y$

$y' = p\,y - gamma\; z_{imag} + omega\; x$

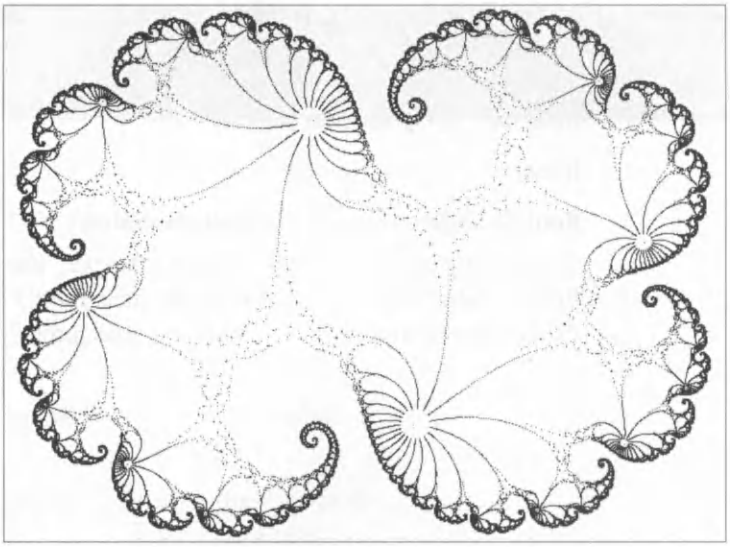
$z' = x^2 + y^2$

Parameters: *Lambda*, *Alpha*, *Beta*, *Gamma*, *Omega*, and *Degree*

Code:

Routine Type	Routine Name	File
2D floating point fractal engine	orbit2dfloat()	LORENZ.C
3D floating point fractal engine	orbit3dfloat()	LORENZ.C
Floating point orbit	iconfloatorbit()	LORENZ.C
Floating point orbit	lorenz3d1floatorbit()	LORENZ.C
Floating point orbit	lorenz3d3floatorbit()	LORENZ.C
Floating point orbit	lorenz3d4floatorbit()	LORENZ.C
Integer math orbit	lorenz3dlongorbit()	LORENZ.C

Julia_inverse



Category: 2-D Orbit

The classic Julia formula ($z' = z^2 + c$)is attracted to infinity if it is initialized by points outside the Julia set, and it is attracted to points within the Julia set otherwise. The inverse formula ($z' = \pm (z–c)$) has the opposite characteristic, and it is repelled by those same attractors. This means that the orbit of this inverse function tends to trace out the boundary of the Julia set. This method for drawing Julia sets is called the Inverse Iteration Method (IIM).

Not every border point is visited equally often by the orbit of the inverse function. Fractint keeps track of the more often visited points and avoids them. This is called the Modified Inverse Iteration Method, or MIIM, and is much faster than the IIM. Different options are provided for traversing the decision tree to decide which of the two square roots to take in the formula, including Breadth first, Depth first (left or negative first), Depth first (right or positive first), and completely at random. The results vary significantly depending on the traversal method chosen. As far as we know, this fact is an original discovery by Michael Snyder, the implementor of Fractint's julia_inverse type, and is published here for the first time.

Example:

```
SeaShell            { ; MIIM Julia                            Michael Snyder
                      ; Like a Chambered Nautilus cut in half; my favorite.
                      ; Give it your highest 16-color resolution.
  reset=1732 type=julia_inverse miim=depth/right
  corners=0.839251/-0.839269/-1.119046/1.119006/-0.839269/1.119006
  params=0.27334/0.00742/5/1024
  }
```

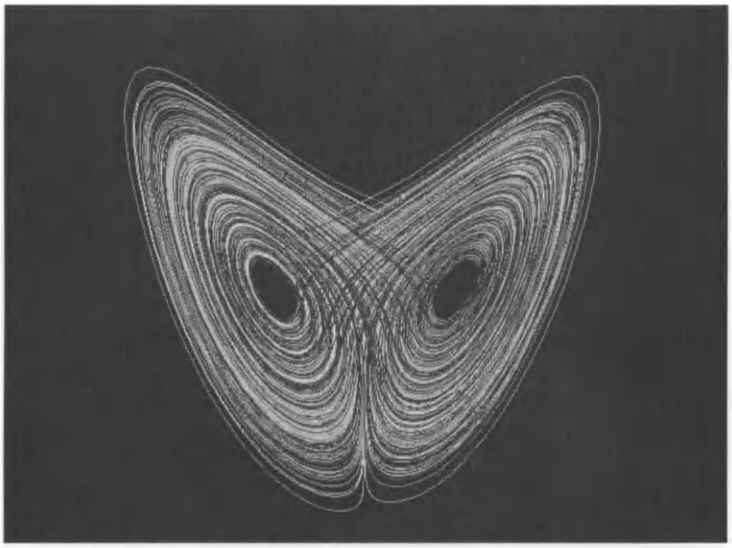
Formula: Initialize: z = point near Julia set

Iterate: $z' = + - (z - c,)$

Code:

Routine Type	Routine Name	File
C fractal engine	inverse_julia_per_image()	CALCFRACT.C
Integer math orbit	Linverse_julia_orbit()	FRACTALS.C
Floating point orbit	Minverse_julia_orbit()	FRACTALS.C

Lorenz



Category: 2-D Orbit

This type is the 2-D projection of type lorenz3d.

Example:

```
lorenz {
  reset colors=@default.map type=lorenz
  }
```

Formula:

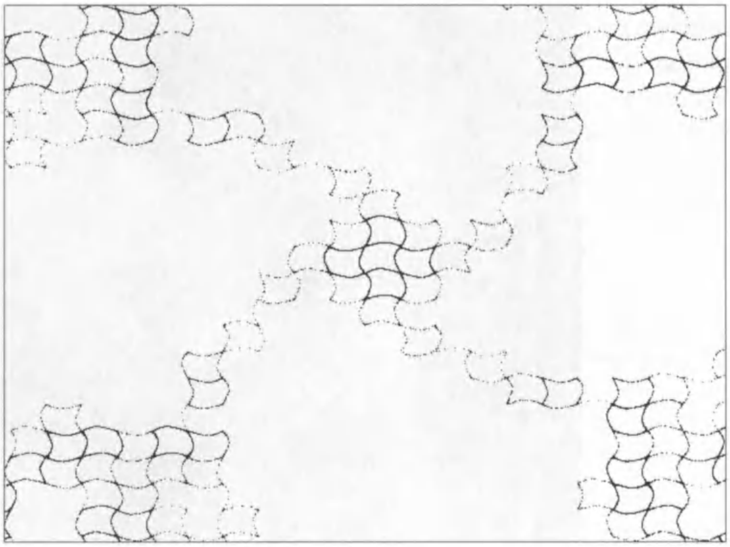
Intitialize: $x' = x - (ax + ay)dt$
$z = y = z = 1$

Iterate: $y' = y + (bx - y - zx)dt$
$z' = z - (cz + xy)dt$

Code:

Routine Type	Routine Name	File
2D floating point fractal engine	orbit2dfloat()	LORENZ.C
2D integer math fractal engine	orbit2dlong()	LORENZ.C
Floating point orbit	lorenz3dfloatorbit()	LORENZ.C
Floating point orbit	lorenz3dlongorbit()	LORENZ.C

Martin



Category: 2-D Orbit

The martin type is attributed to Barry Martin of Aston University in Birmingham, England. The orbit traces out a quilt-like pattern, and keeps changing in sudden spurts.

Example:

```
martin {
  reset colors=@default.map type=martin params=3.14159
  }
```

Formula:

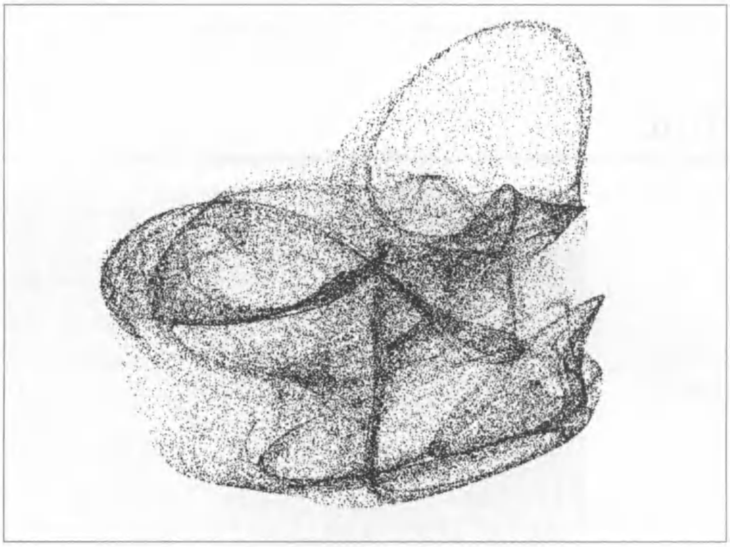
Initialize: $x = y = 0$:

Iterate: $x' = y - \sin(x)$
$y' = a - x$

Code:

Routine Type	Routine Name	File
2D floating point fractal engine	orbit2dfloat()	LORENZ.C
Floating point orbit	martin2dfloatorbit()	LORENZ.C

Pickover



Category: 2-D Orbit

This type is due to Clifford A. Pickover of the IBM Thomas J. Watson Research Center. Be sure to try this one using 3-D. (Press I and select 2 at the STEREO (R/B 3D)? prompt. The fractal looks like a 3-D web with a wispy structure.)

Example:

```
pickover            { ; 3-D image - use red/blue glasses
  reset type=pickover corners=-3.696451/3.79018/-2.249562/3.098032
  params=2.24/0.43/-0.65/-2.43
  float=y maxiter=1000 inside=0 rotation=60/30/0
  perspective=150 xyshift=0/0 stereo=2 interocular=3 converge=-4
  crop=4/0/0/4 bright=80/100
  colors=@glasses2.map
  }
```

Formula:

Initialize: $x = y = z = 1$

Iterate: $x' = \sin(ax) - z\cos(bx)$
$y' = z\sin(cx) - \cos(dy)$
$z' = \sin(x)$

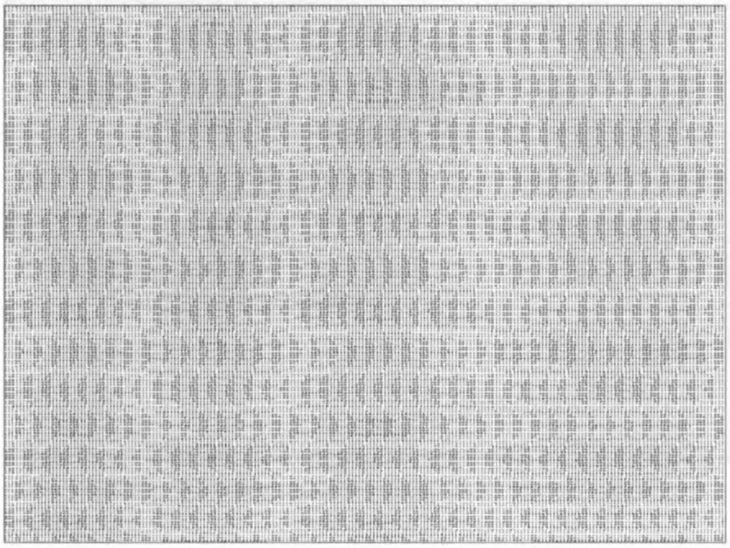
Code:

Routine Type	Routine Name	File
Floating point fractal engine	orbit3dfloat()	LORENZ.C
Floating point orbit	pickoverfloatorbit()	LORENZ.C

Superimposed Orbits

These fractals are not the graphs of a single orbit, but the superimposition of many orbits, one on top of the other.

Dynamic



Category: Superimposed Orbits

This type, taken from Clifford Pickover's book *Computers, Pattern, Chaos, and Beauty* is much like Pickover's popcorn fractal. There are two options controllable from the BASIC OPTIONS <x> screen that have unusual effects on these fractals. The ORBIT DELAY value controls how many initial points are computed before the orbits are displayed on the screen. This allows the orbit to settle down. Setting the outside option to summ causes each pixel to increment color every time an orbit touches it; the resulting display is a 2-D histogram. (Note: this is unrelated to the normal effect of the `outside=summ` option when used with escape-time fractals.)

Example:

```
Sin_Waves          { ; Dynamical System Fractal
                     ; By Dan Farmer
                     ; "Dynamic"  Example
  reset type=dynamic function=log passes=1
corners=-31.99/-6.208486/5.823209/31.604723 params=200/0.001/10/5
float=y maxiter=25 inside=0 outside=summ
colors=000rT3LPb<10>ksMJNc<40>JAYI9XJNb<25>Gk3JNc<54>kA3J0b<25>UmBJNc<11\
>MPkJNc<40>J9'I8_JNb<19>XI5L0cNQbMN'PSa<7>fgV
}
```

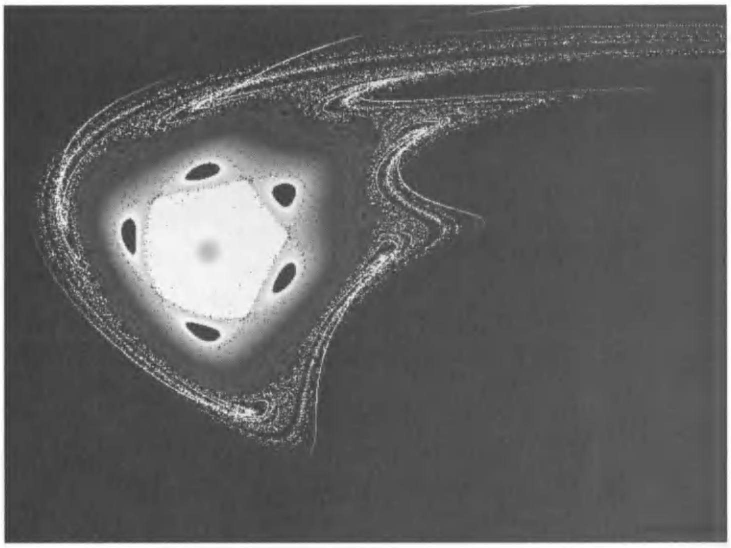
Formula: $y' = y + \mathrm{f}(x)$
$x' = x - \mathrm{f}(y)$

where $\mathrm{f}(k) = \sin(k + a\,\mathrm{fn}(bk))$, and $\mathrm{fn}(z)$ is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	Routine Name	File
Fractal engine	dynam2dfloat()	LORENZ.C
Floating point initialization	dynam2dfloatsetup()	LORENZ.C
Floating point orbit	dynamfloat()	LORENZ.C

Kamtorus



Category: Superimposed Orbits

This type is the 2-D version of kamtorus3D.

Example:

```
NYUF013             { ; Kamtorus variation #2 - Dan Farmer
  reset type=kamtorus corners=-1.103439/2.785873/-1.57457/1.341705
  params=1.3/0.005/300/500 maxiter=3200 inside=0
  colors=000dMk<40>zzzzzzzzy<60>zz1zz0zy0yx0<58>G2JF0KF0K<4>F0LG0LG0LG0MG0MH0M\
  <52>T0bU0cU0cV1cV2d<18>dLjs
  }
```

Formula:

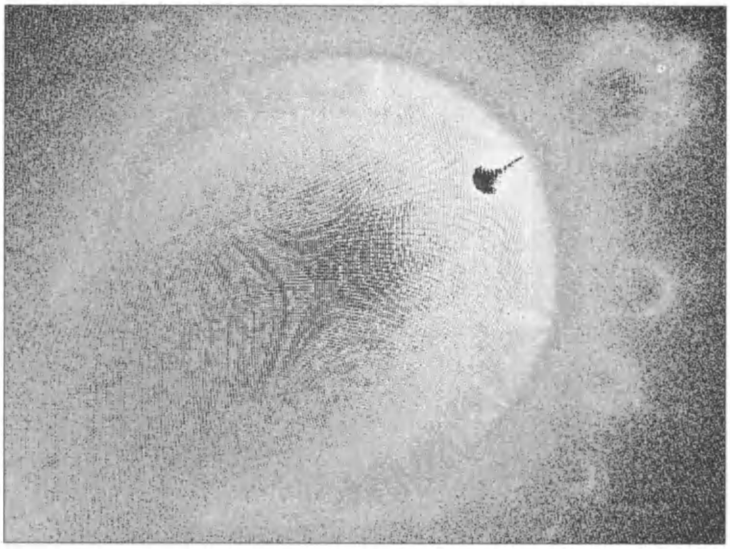
Initialize: $x = y = 0$

Iterate: $x' = x\cos(a) + (x^2 - y)\sin(a)$
$y' = x\sin(a) - (x^2 - y)\cos(a)$

Code:

Routine Type	Routine Name	File
2D floating point fractal engine	orbit2dfloat()	LORENZ.C
2D integer math fractal engine	orbit2dlong()	LORENZ.C
Floating point orbit	kamtorusfloatorbit()	LORENZ.C
Integer math orbit	kamtoruslongorbit()	LORENZ.C

Mandelcloud



Category: Superimposed Orbits

This type demonstrates yet another method of using the classic Mandelbrot/Julia formula to make a fractal image. The Mandelbrot orbits are superimposed.

Example:

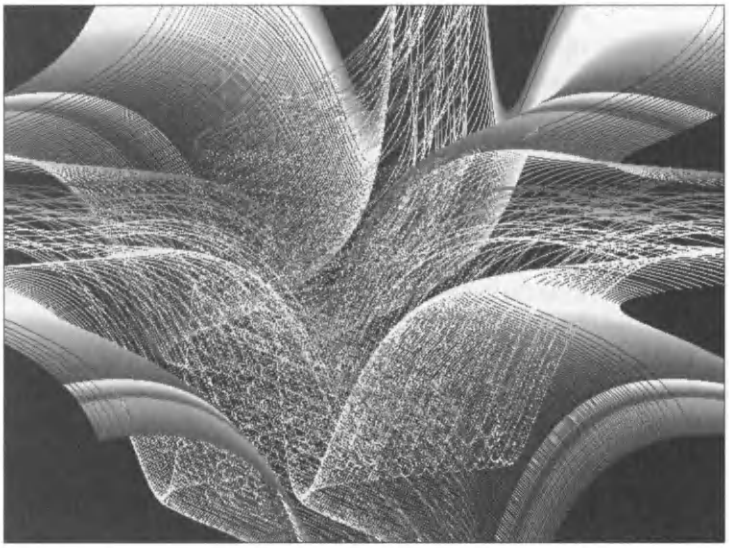
```
In_the_Abyss       { ; Extra-terrestrial Jelly-fish! (C)1993 Peter Moreland
                     ; Data 486 | 33 | Y | SF7 | 17.31(5) | 0:2:22.91
                     ; Produced 31.1.93
                     ; Map = Joe
  reset type=mandelcloud
  corners=-0.833158/-1.151253/0.437465/-0.407465/-0.542105/0.049396
  params=350 float=y maxiter=55 outside=summ
  colors=0004FW6IX8MX90ZJNXRNXMQYSTZKRdRVcYUYfUYUX'UYe_Z_eYXiYXe'Xh'YeZ'iY\
  'ea'ia'hb'gd'Z'ffafadhgfgn'Zsa_nd'tcanbdsbdnffuff<2>tliZbkdbkafmfinaisnj\
  lujkhlqimsllmplmnompommmpqnpmoqppqunprtruupnptwrtqtxutuxtuxwutvyyvwuxzzz\
  z<24>zzz3115223436423345343566549 31D31962D52964D74784A83D92FC2A95D95BC5E\
  D54799795ACA9ADA9AD9ED9AADDBDADEEDDG31H62L72H74L75P75HA3L93ID3LF3H95LA5H\
  D6LD6QC6HB8LB8ID9ME9HACHEDLEDPB8SB9PE9TE9QFCUFCFG6EHBJG3MG3IG6MH5NK6PI3U\
  H3RK3TL3PI4TH6QL5TM5U06IH9MHAIKAMLAIHEMIDILDMLEQHATHAPLAULAPIDTIDQLEULEM\
  PESPC37J5EJCEI4ESAEQIEIPFJHE06GMEHKFPM6IR9KTCPUIHHLIHJLIMLHHILLILHKMMLLR\
  LILPKTPKJLQQMQLPSRRRXE6XDAYL5YQ6dP6YKDdKCZPDeRCZMHdMIYPI'PIXTIaSJXQLaQLX\
  TMaTLeRLYMRZTQfTQkTNlURUXJUXSYWEfXA_YMdXN'XTfYTecSmXUlfS
  }
```

Formula: Iterate: $z' = z^2 + c$

Code:

Routine Type	Routine Name	File
Fractal engine	dynam2dfloat()	CALCFRACT.C
Floating point orbit	mandelcloudfloat()	FRACTALS.C

Popcorn



Category: Superimposed Orbits

Here is another Clifford Pickover fractal. The strange appearance of this fractal is due to the superimposition of many orbits together.

Example:

```
sba016           { ; (c) 1993 Richard H. Sherry, CIS:76264,752
  reset=1701 type=popcorn
  corners=-2.047073/-1.013535/4.3996124/5.1545563
  params=0.14999999999999999445 inside=0 float=no
  colors=000RD6RC5RC4<66>umZumZtlYtlYskYCB3skX<97>CA2lWs<13>EB5BITCEFC_C<3\
  2>CA2JwE<28>CB2
  }
```

Formula:

Initialize: $x = zpixel_x, y = zpixel_y,$

Iterate: $x' = x - 0.05\sin(y) + \tan(3y)$

$y' = y - 0.05\sin(x) + \tan(3x)$

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Integer math initialization	long_julia_per_pixel()	FRACTALS.C
Integer math orbit	LPopcornFractal()	FRACTALS.C
Floating point initialization	otherjuliafp_per_pixel()	FRACTALS.C
Floating point orbit	PopcornFractal()	FRACTALS.C

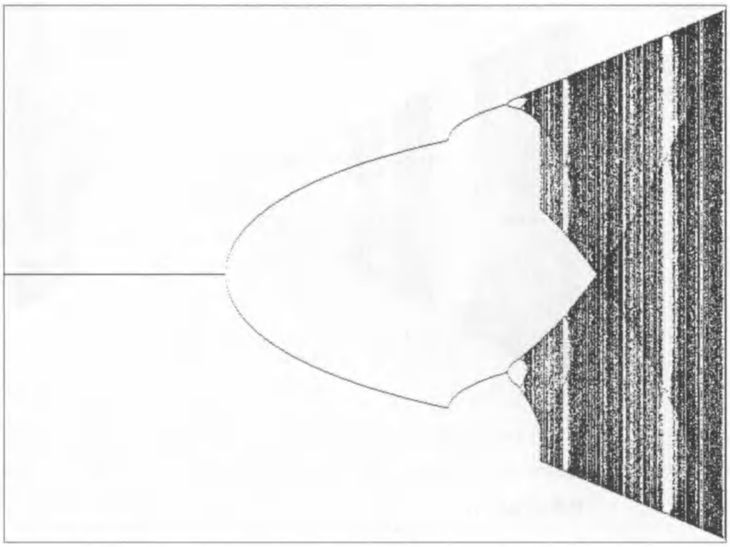
Fractal Miscellanea

This section includes Fractal types that we just couldn't fit into our grand scheme. Included are the Bifurcation Fractals, several random fractal types, and the circle and cellular types.

Bifurcation

Bifurcation fractals are based on an intriguing kind of dynamic system that behaves normally up to a certain level of some controlling parameter, then goes through a transition in which there are two possible solutions, then four, and finally a chaotic array of possibilities. Examples of this emerged many years ago in biological models of population growth. The following bifurcation fractal types demonstrate that this behavior exists for a wide variety of iterated formulas.

Bif+sinpi



Category: Bifurcation

This type originally used the sine function. The formula involves the sum of a linear term and a function. The formula has been generalized starting in Fractint version 18.

Example:

```
bifurcation_plus_sinepi {
  reset maxiter=20000 colors=@default.map type=bif+sinpi
  }
```

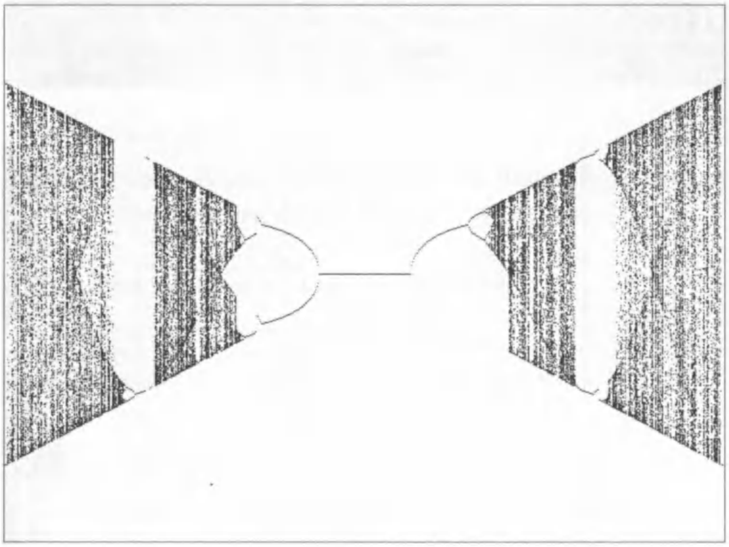
Formula: $x' = x + r\text{fn}((x))$

where "fn" is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	Routine Name	File
Fractal engine	Bifurcation()	CALCFRACT.C
Integer math orbit	LongBifurcAddSinPi()	CALCFRACT.C
Floating point orbit	BifurcAddSinPi()	CALCFRACT.C

bif=sinpi



Category: Bifurcation

The formula is the same as bif+sinpi with the linear term cut. The formula has been generalized starting in Fractint version 18.

Example:

```
bifurcation_equal_sinepi {
  reset maxiter=20000 colors=@default.map type=bif=sinpi
  }
```

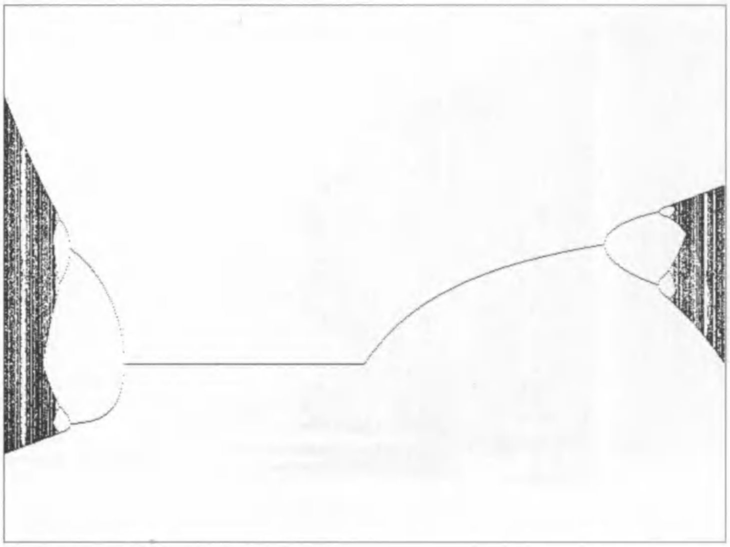
Formula: $x' = r\text{fn}((x))$

where "fn" is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	Routine Name	File
Fractal engine	Bifurcation()	CALCFRACT.C
Integer math orbit	LongBifurcSetSinPi()	CALCFRACT.C
Floating point orbit	BifurcSetSinPi()	CALCFRACT.C

Biflambda



Category: Bifurcation

When fn is ident, the formula for this bifurcation fractal is the same as that for the lambda escape-time fractal. The formula has been generalized starting in Fractint version 18.

Example:

```
biflambda {
  reset maxiter=20000 colors=@default.map type=biflambda
  }
```

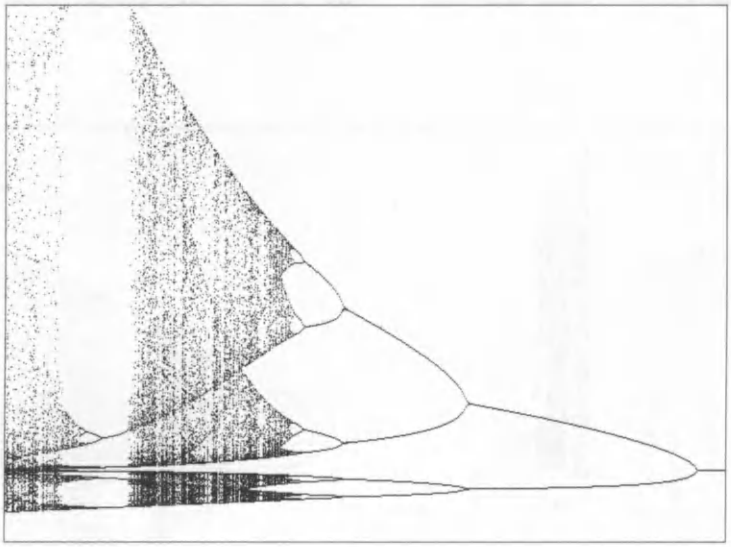
Formula: $x' = r\text{fn}(x)(1 - \text{fn}(x))$

where “fn” is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:

Routine Type	Routine Name	File
Fractal engine	Bifurcation()	CALCFRACT.C
Integer math orbit	LongBifurcLambda()	CALCFRACT.C
Floating point orbit	BifurcLambda()	CALCFRACT.C

Bifmay



Category: Bifurcation

This bifurcation fractal has quite a different iterated formula that involves an exponent.

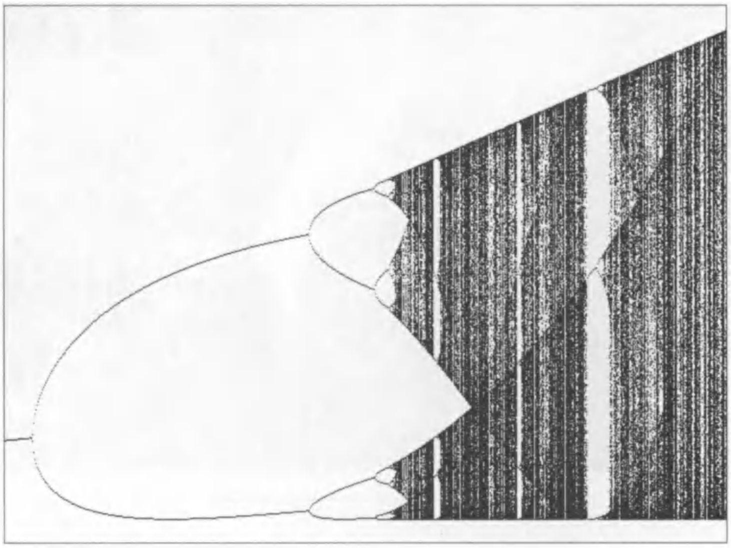
Example:

```
bifmay {
  reset maxiter=20000 colors=@default.map type=bifmay
  }
```

Formula: $x' = rx/((1+x)^{beta})$

Code:	Routine Type	Routine Name	File
	Fractal engine	Bifurcation()	CALCFRACT.C
	Integer math setup	LongBifurcMaySetup()	CALCFRACT.C
	Floating point setuo	BifurcMaySetup()	CALCFRACT.C
	Integer math orbit	LongBifurcMay()	CALCFRACT.C
	Floating point orbit	BifurcMay()	CALCFRACT.C

Bifstewart



Category: Bifurcation

This fractal is a variation on bifurcation without an *x* term in the formula. The formula has been generalized starting in Fractint version 18.

Example:

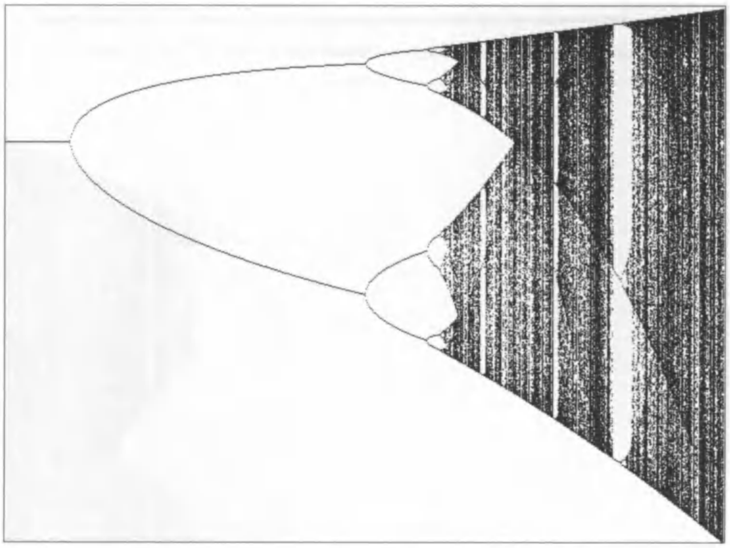
```
bifurcation_Stewart{
  reset maxiter=20000 colors=@default.map type=bifstewart
  }
```

Formula: $x' = r\text{fn}(x)^2 - 1$

where "fn" is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:	Routine Type	Routine Name	File
	Fractal engine	Bifurcation()	CALCFRACT.C
	Integer math orbit	LongBifurcStewart()	CALCFRACT.C
	Floating point orbit	BifurcStewart()	CALCFRACT.C

Bifurcation



Category: Bifurcation

This is the original bifurcation fractal based on the population model. The formula has been generalized starting in Fractint version 18.

Example:

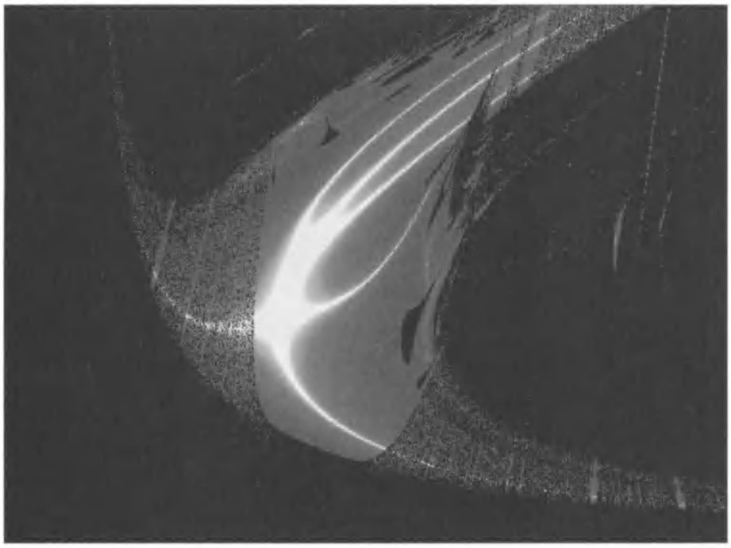
```
bifurcation {
  reset maxiter=20000 colors=@default.map type=bifurcation
  }
```

Formula: $x' = x + r\text{fn}(x)(1 - \text{fn}(x))$

where "fn" is one of conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, or zero.

Code:	Routine Type	Routine Name	File
	Fractal engine	Bifurcation()	CALCFRACT.C
	Integer math orbit	LongBifurcVerhulst()	CALCFRACT.C
	Floating point orbit	BifurcVerhulst()	CALCFRACT.C

Lyapunov



Category: Bifurcation

This variation on a fractal made from the population model appeared in A.K. Dewdney's "Mathematical Recreations" column of *Scientific American*, September, 1991.

The bifurcation fractal illustrates what happens in a simple population model as the growth rate increases. The Lyapunov fractal expands that model into two dimensions by letting the growth rate vary in a periodic fashion between two values. Each pair of growth rates is run through a logistic population model and a value called the Lyapunov Exponent is calculated for each pair and is plotted. The Lyapunov Exponent is calculated by adding up log $|r-2rx|$ over many cycles of the population model and dividing by the number of cycles. Negative Lyapunov exponents indicate a stable, periodic behavior and are plotted in color.

Positive Lyapunov exponents indicate chaos (or a diverging model) and are colored black.

Example:

```
Lyapunov_one       { ; (c) 1992 Peter Moreland, The Saint
                    ; come in Brett Sinclair!
  reset type=lyapunov corners=2.93748764/2.94703484/3.67514065/3.68217105
  params=4/0.75 maxiter=50 colors=000000<29>00k00m01m<16>0St
  }
```

Formula: $x' = rx(1 - x)$

Code:

Routine Type	Routine Name	File
Fractal engine	lyapunov()	CALCFRACT.C
asm routines		LYAPUNOV.ASM
Floating point orbit	BifurcLambda()	CALCFRACT.C

Random Fractals

Most of the fractals generated by Fractint are *deterministic*. The mathematical methods used to generate these fractals are repeatable and give the same result each time. The plasma and diffusion types are different; they use a *random* element in their generation. Each plasma and diffusion image is unique. If you generate 30 images using these two types, you will get 30 results.

Plasma

Category: Random

The plasma fractal creates smoothly varying cloud-like colors. Be sure to try rotating the colors using (+). When used with the (3) command, makes mountains by converting colors to a third dimension.. At the request of ray tracing enthusiasts, there is now an option to output 16-bit .POT files, making 65,536 possible "mountain elevations" instead of just 256. The program POV-Ray that comes with both the *Ray Tracing Creations* (by Drew Wells and Chris Young, © 1993, Waite Group Press) and *Image Lab* (by Tim Wegner, © 1992, Waite Group Press) books can read these .POT files.

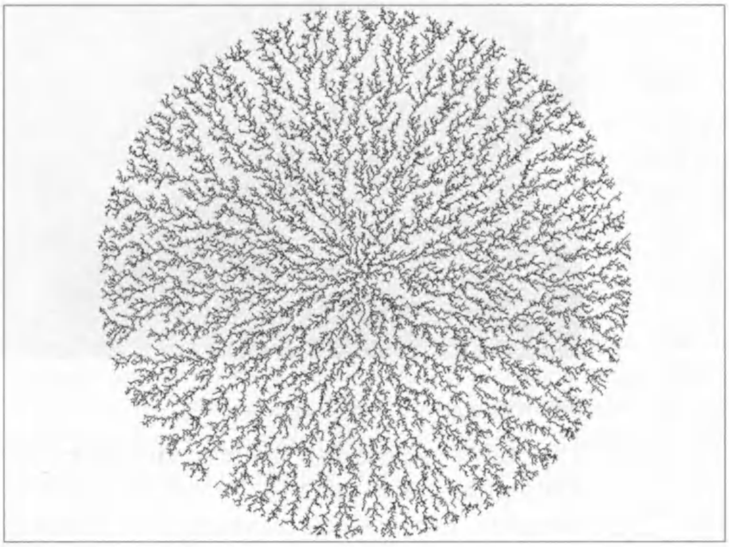
The (TAB) status display shows the random seed (rseed) value used with the current plasma image. Using this value with the `rseed=` command line option allows the exact duplication of the plasma fractal, so that specific images can be included in .PAR files. Otherwise, each image is unique.

Example:

```
plasma          {
  reset=1733 type=plasma corners=-2/2/-1.5/1.5 params=2
  }
```

Formula:	variant of the midpoint displacement algorithm		
Code:	**Routine Type**	**Routine Name**	**File**
	Fractal engine	plasma()	CALCFRACT.C

Diffusion



Category: Random

The diffusion fractal works by randomly choosing pixels inside a box. When one touches an existing, colored pixel, then the new pixel is colored. The result is a growth pattern like a cystal in a supersaturated solution.

Example:

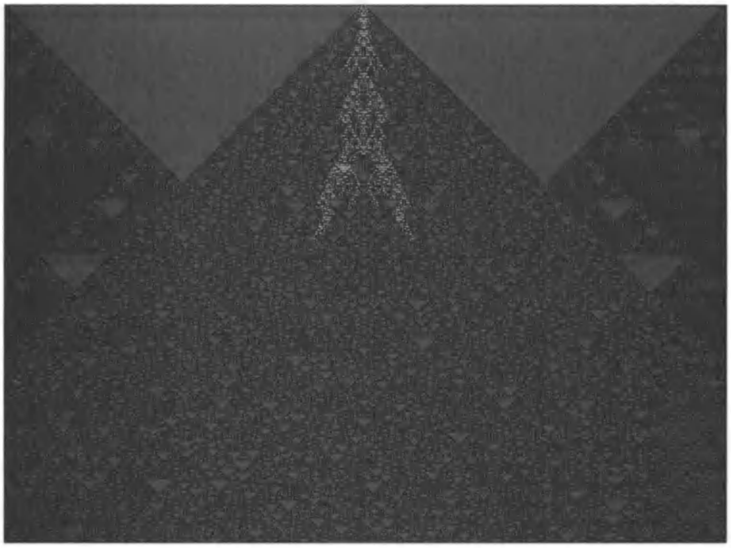
```
diffusion {
  reset colors=@default.map type=diffusion params=1
  }
```

Code:	**Routine Type**	**Routine Name**	**File**
	Fractal engine	diffusion()	CALCFRACT.C

Cellular Automaton

The Cellular Automation is a bit of a misfit in Fractint. It is an idealistic mathematical machine that generates images in a very different way from the methods used with other fractal types. We are content to leave this lonely fractal type here in its own category with no apologies as a reminder that the study of fractals is not a very tidy subject!

Cellular



Category: Cellular Automaton

A cellular automaton colors cells using simple rules that determine the contents of a cell from the colors of neighboring cells. In the spirit of fractals, extremely complex patterns arise from simple rules.

Example:

```
Type_61            { ; Jonathan Osuch
  reset type=cellular corners=-1.0/1.0/-1.0/1.0
  params=11.0/2111000355004045.0/61.0 inside=0 colors=000e00AZADCTwuIeL0
  }
```

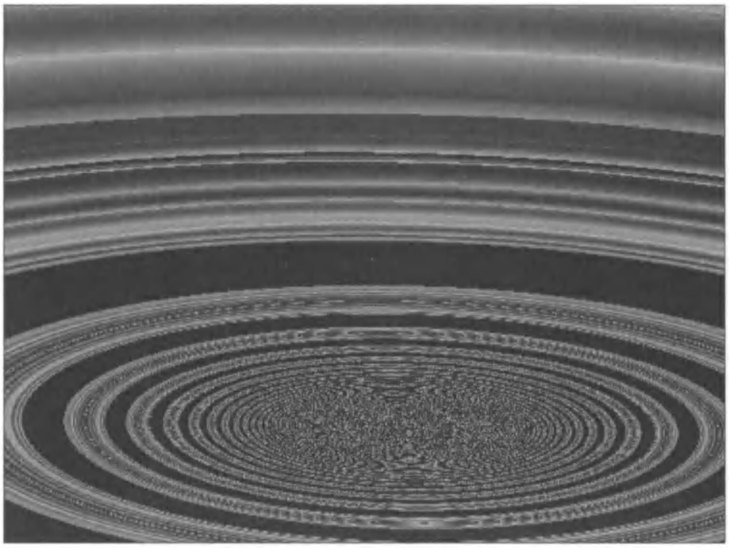
Code:

	Routine Type	Routine Name	File
	Fractal engine	Cellular()	MISCFRAC.C
	Integer math setup	CellularSetup()	MISCFRAC.C

Moiré Pattern

Moiré patterns are most commonly created by the interference of two patterns. You might see such a Moiré pattern looking through two window screens at an oblique angle, or in the colors on the surface of a soap bubble. Once again we prove that the discipline of fractals is the study of "loose ends," as we classify the circle fractal type in a category by itself.

Circle



Category: Moiré Pattern

Moiré interference patterns are created by truncating the fractional part of the distance from the center of each pixel. Is this a fractal? If you zoom in, detail disappears; but if you zoom out, it increases. No problem, apply inversion (under the Ⓨ EXTENDED OPTIONS screen) and you'll find it is indeed a fractal! But even noninverted images can be quite interesting.

Example:

```
whirlpool          { ; Tim Wegner
                     ; Inverted circle, based on Lee Skinner's SKIP.GIF
  reset=1720 type=circle passes=1
  corners=-2.44259235/0.689915143/0.45481222/-0.32858413/0.597129484/-0.77\
  192783 params=10000 float=y bailout=6 inside=0 logmap=yes
  invert=0.130566/0/0 viewwindows=1.25/0.75/yes/0/0
  colors=000Fmi<2>Fnb<5>thM<7>rJu<7>lvv<4>o3J<4>vnXwedxXmzNv<4>s59m89fC9_G\
9VXB<3>'n7PFJ<6>P5qv46OHGNIIWMK<3>ykcDDFOAFE_NEaS3xdPFE<4>065NMHKSKZJVHY\
N<3>6vZ<5>IEZL6_MEX<5>SyClWq<3>NPe<7>NvL<7>MrCVvGdzK<6>ZLD<6>VxL<2>GGjFN\
qDVyRerdpk<7>DOTRQuRVmR_d<7>wzw<7>JFu<2>zur<5>GxS<2>EdWDYYCTW<2>8D067L68\
J<5>8G2<5>aG4fH5iJGlMRpPa<7>Ylx<7>CVB<5>qWq<4>rPb<7>Ifi<3>lpX<6>FmoFmmFm\
k
}
```

Formula: c = integer part of $a(x^2 + y^2)$
$color = c$ modulo(number of colors)

Code:

Routine Type	**Routine Name**	**File**
Fractal engine	StandardFractal()	CALCFRACT.C
Floating point orbit	CirclefpFractal()	FRACTALS.C

USER-DEFINED FRACTALS

The last three fractal types in this chapter are not "types" at all, they are fractal universes in and of themselves. Each of these types allows you to define new fractal types by entering the definitions in a Fractint-readable file using a text editor like the DOS EDIT.EXE program.

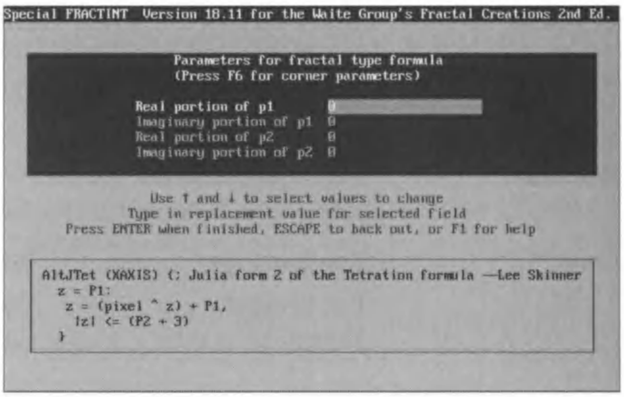
The Fractint Formula Parser

The Fractint formula parser is one of the all-time popular features of Fractint. Using it, you can create new fractal formulas, and try them out on the spot, without having to be a programmer! This facility has unleashed the creativity of fractal enthusiasts from every continent, who have created fractals from all kinds of different formulas. A small sample of the product of their labors may be found in the FRACTINT.FRM file on your book disk.

The formula parser has been greatly speeded up when run on machines with a math coprocessor. All time-critical parser code has been rewritten in assembler for Fractint version 18. If you have a computer with fast floating-point performance, such as an 80486-based machine, you will discover there is little discernable difference in speed between a formula type and a built-in fractal. If you don't have a math coprocessor, the formula parser can use Fractint's fast integer math, but will be somewhat slower than the equivalent built-in fractal type.

Accessing a Formula File

To generate a fractal using the formula parser, press T to get the SELECT A FRACTAL TYPE screen, select FORMULA, and press ENTER. If Fractint can find the file FRACTINT.FRM, you will be presented with a list of formulas defined in that file



```
Special FRACTINT  Version 18.11 for the Waite Group's Fractal Creations 2nd Ed.

              Parameters for fractal type formula
              (Press F6 for corner parameters)

          Real portion of p1        0
          Imaginary portion of p1   0
          Real portion of p2        0
          Imaginary portion of p2   0

            Use ↑ and ↓ to select values to change
          Type in replacement value for selected field
     Press ENTER when finished, ESCAPE to back out, or F1 for help

 AltJTet (XAXIS) {; Julia form 2 of the Tetration formula —Lee Skinner
   z = P1:
    z = (pixel ^ z) + P1,
     |z| <= (P2 + 3)
   }
```

Figure 6-5 Parameters for Fractal Type Formula screen

in a screen labeled: FORMULA SELECTION FILE: FRACTINT.FRM. If Fractint cannot find the file in the current directory, you will see instead a standard Fractint file selection and directory navigation screen. You can directly enter the directory where the file FRACTINT.FRM resides (usually \FRACTINT) or navigate your directory tree by selecting ".." to go up a directory or a subdirectory name to go down. Once you have located FRACTINT.FRM (or another .FRM file), select it with the arrow keys and press ENTER. Once in the FORMULA SELECTION screen, if you want to open a different .FRM file, press F6. You don't need to remember this command because there is a reminder at the bottom of the screen. Note also that pressing F2 from the FORMULA SELECTION screen shows you the details of the highlighted formula.

Once in the FORMULA SELECTION screen, select a formula name and press ENTER. This takes you to the PARAMETERS FOR FRACTAL TYPE FORMULA screen, as shown in Figure 6-5. At the bottom of the screen in a box you can see the formula parser definition. After entering any desired parameters, press ENTER to generate the fractal. If you have previously selected a video mode, the fractal will begin to generate. Otherwise, press a function key to select a video mode, or press DELETE to select a mode from the video mode list.

The Structure of a Formula File Entry

Figure 6-6 is an annotation of the Richard8 formula that shows what the different parts do. Let's walk through this example.

A formula begins with a name, in this case Richard8. This is the name that appears in the list of entries when you open the .FRM file. Following the name,

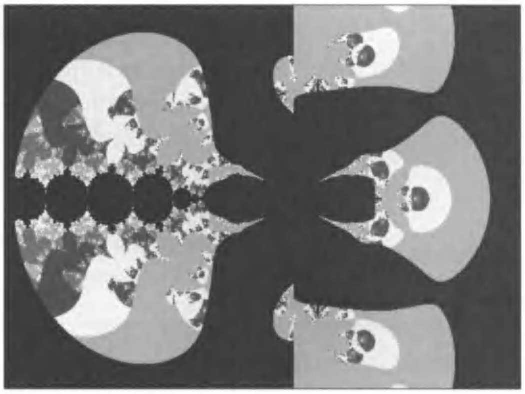


Figure 6-6 The world-famous "Authors" fractal

enclosed in parentheses, is the symmetry. You do not have to specify symmetry; the purpose is to enable the fractal to calculate faster by exploiting symmetry if indeed the fractal is symmetrical. The Richard8 fractal has XYAXIS symmetry, which means that the fractal is symmetrical about both the *x*-axis and *y*-axis. The possible values for symmetry are XAXIS, YAXIS, XYAXIS, ORIGIN, and PI.

The body of the formula definition is enclosed in curly brackets ({ and }). The semicolon character (;) means that the rest of the line is ignored, so you can enter comments in your formula. The first line of the Richard8 formula uses a comment to remind us of the name of the good professor Jm Richard-Collard, who designed the Richard8 formula.

The body of a formula is divided into three parts, a per-pixel initializer, the iterated formula, and a bailout criterion. In the Richard8 formula, the initializer is the line

```
z=pixel, sinp = sin(pixel):
```

This single line contains two statements separated by a comma. (The two statements do not have to be on the same line, although for clarity it is a good idea.) The first statement, *z=pixel*, creates a complex variable *z* and assigns to it the value *pixel*. (The parser operates using the complex number system.) The variable *pixel* is a predefined variable that always contains the complex coordinates corresponding to the current pixel. Generally, the initialization section makes some use of the *pixel* variable. The second statement in the initialization line is `sinp = sin(pixel)`. This line creates a variable sinp, and assigns to it the value of the sine function applied to *pixel*. The colon character (:) signals the end of the initialization section. Subsequent statements will be executed every iteration.

The next line of the formula definition is the iterated formula. In this case, the formula is made up of the single statement $z = \sin(z) + sinp$. This statement calculates the sine function of the variable z, adds the result to the complex number stored in the variable *sinp*, and replaces the old value stored in z with the result.

The last line gives the bailout criterion. The iteration process will continue as long as this condition is true. The Richard8 criterion is |z| <= 4. This statement is not quite what it appears to be: |z| is what we call the "programmer's

Operator	Meaning	Operator	Meaning
(2.33, -2.23)	A complex constant	sinh()	Hyperbolic sine
3.245	A complex constant—same as (3.245,0)	sqr()	Square
xyz	Creation of variable xyz	srand()	Seed random number generator
|...|	Modulus squared operator $\lvert(x,y)\rvert = (x^2 + y^2,0)$	tan()	Tangent
abs()	Converts both real and imaginary components to positive numbers	tanh()	Hyperbolic tangent
conj()	Complex conjugate	-	Negation
cos()	Cosine	^	Power
cosh()	Hyperbolic cosine	*	Multiplication
cosxx()	Complex conjugate of cosine	/	Division
cotan()	Cotangent	+	Addition
cotanh()	Hyperbolic cotangent	-	Subtraction
exp()	Exponential	=	Assignment
flip()	Swap real and imaginary parts	<	Comparison of real components (less than)
fn1()	First function variable	<=	Comparison of real components (less than or equal to)
fn2()	Second function variable	>	Comparison of real components (greater than)
fn3()	Third function variable	>=	Comparison of real components (greater than or equal to)
fn4()	Fourth function variable	==	Equal to
imag()	Real part of argument	!=	Not equal to
log()	Natural logarithm	&&	Logical AND
real()	Imaginary part of argument	||	Logical OR
sin()	Sine		

Table 6-3 The formula parser operator definitions

Variable	Predefined Meaning
z	Used for periodicity checking
p1	Parameters 1 and 2
p2	Parameters 3 and 4
pixel	Screen coordinates
LastSqr	Modulus from the last sqr() function
rand	Complex random number

Table 6-4 The formula parser predefined variables

absolute value," not the more universally understood "mathematician's absolute value." If $z = x + iy$, then in the parser language $|z| = x^2 + y^2$ rather than the value ($x^2 + y^2$) that a mathematician would expect. The reason for this is that no self-respecting programmer would compare ($x^2 + y^2$) with 2 when comparing $x^2 + y^2$ with 4 has the same result, because square roots require considerable computation resources. If you are aware of this idiosyncracy of Fractint, no problem will result.

Formula Parser Operators

Table 6-3 defines all the functions and operators, and Table 6-4 defines all the predefined variables available to you for use in the parser. They are given in decreasing order of precedence (operators earlier in the list will be performed before operators later on the list if there are no parentheses to specify the order explicitly). Most functions are complex valued functions of a complex variable. The exceptions are the comparison operators which return TRUE or FALSE, and the logical operators which take truth values as arguments. Note that $|z|$ and abs(z) have definitions somewhat different from common mathematical useage.

You may be a bit puzzled by the function cosxx. When the cosine function was first implemented in Fractint, there was a sign error in the code. This error was corrected in later versions of Fractint, but by this time many users had made fractals using cosxx. So the Fractint authors left the incorrect function in Fractint with a different name.

Predefined Variables

Certain variables have been given predefined meanings by the parser. You should use the variable *z* as the main variable whose value is changed each iteration.

More Examples

Consider the following formula entry, taken from FRACTINT.FRM:

```
Cubic (XYAXIS) {; Lee Skinner
  p = pixel, test = p1 + 3,
  t3 = 3*p, t2 = p*p,
  a = (t2 + 1)/t3, b = 2*a*a*a + (t2 - 2)/t3,
  aa3 = a*a*3, z = 0 - a :
   z = z*z*z - aa3*z + b,
     |z| < test
}
```

The first few lines of this formula are actually part of the intialization section, which is not finished until a ":" is encountered. Therefore, the iterated formula consists only of the line

```
z = z*z*z - aa3*z + b,
```

Note also that the variable *test* is used as a bailout threshold. Because *test* = *p1* + 3, the user can control the bailout from the parameter screen using the predefined variable *p1*. For example, if you set Real Portion Of P1 to 4 in the Parameters For Fractal Type Formula screen, then the variable test would be 4 + 3 and the bailout check would be the same as

```
|z| < 7
```

Remember that the comparison operators (such as "<") operate on the real part of variables only. In this particular formula, it, therefore, makes no difference how the Imaginary Portion Of P1 is set.

Let's look another formula.

```
Halley (XYAXIS) {; Chris Green. Halley's formula applied to x^7-x=0.
  ; P1 real usually 1 to 1.5, P1 imag usually zero. Use floating point.
  ; Setting P1 to 1 creates the picture on page 277 of Pickover's book
  z=pixel:
   z5=z*z*z*z*z;
   z6=z*z5;
   z7=z*z6;
   z=z-p1*((z7-z)/ ((7.0*z6-1)-(42.0*z5)*(z7-z)/(14.0*z6-2))),
    0.0001 <= |z7-z|
  }
```

This formula has a one-line initialization section, but a four-line iterated section. The bailout criterion is a bit different—the test that 0.0001 <= |z7-z| means that the z is close to a root of the polynomial $z^7 - z$. This is actually an escape-to-finite-attractor fractal. Try it, and you'll see that it looks a lot like the built-in types Halley and Newton. Remember to set the first parameter to a nonzero number when you try this fractal, otherwise you will get a blank screen. To understand why, look at the last long line of the iterated formula. If p1 is 0, then $z = z$ and the formula does nothing!

One final example. When the formula parser was first added to Fractint, Professor Jm Richard Collard was so prolific at proposing formulas, that the Stone Soup team added Fractint's function variable facility to the parser in an attempt to keep the FRACTINT.FRM file to a reasonable size. In the following formula, *fn1*, *fn2*, and *fn3* are function variables. Any of 17 functions can be assigned to these variables using the PARAMETERS FOR FRACTAL TYPE FORMULA screen. The possible functions that can replace the function variables are conj, cos, cosh, cosxx, cotan, cotanh, exp, flip, ident, log, recip, sin, sinh, sqr, tan, tanh, and zero. Because Professor Richard Collard's formula has three function variables, the number of possible variations on his formula with these 17 functions is 17^3 or 4,913 different formulas! Because Jm has contributed 27 such generalized formulas, you can see that his contributions add up to a lot of fractal possibilities!

```
Jm_14 {; generalized Jm Richard-Collard type
  z=pixel,t=p1+4:
   z=fn1(fn2(fn3(z)*pixel))+pixel,
    |z|<=t
  }
```

You will find a PAR example using Jm_14 at the end of this section.

Fame and Fractals

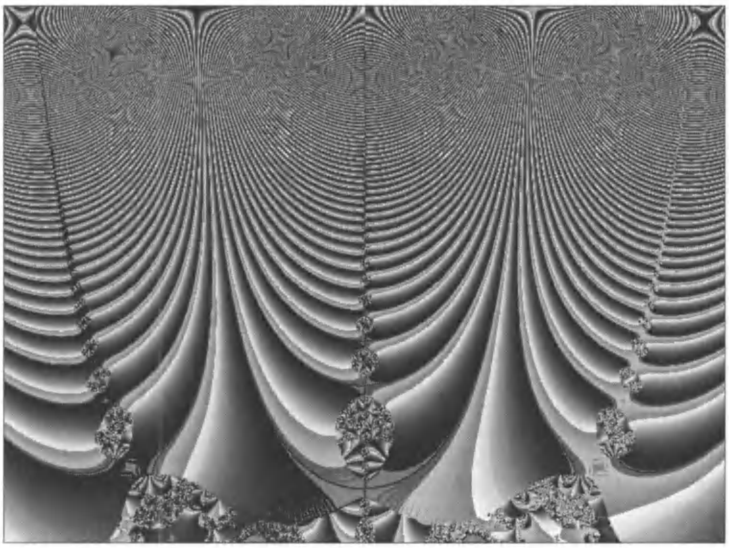
If Dr. Mandelbrot can have a fractal named after him, why not have a fractal named after yourself? To see how much fun this is, we'll invent one right now. If you have never heard of a hyperbolic sine function, you may think you are at a decided disadvantage in inventing fractal formulas. Just remember that even the most learned mathematicians (except the ones reading this book) have never heard of "flip" or "cosxx" either, so you're even!

The best strategy for inventing fractal formulas is to start with a formula that works, and change it a little at a time. However, once you get the hang of it, be bold! Here goes:

```
Authors {
  z = pixel:
  z = sinh(log(Sqr(z))/Sqr(z)) + z
    |z| < p1 + 4
 }
```

We're calling this the "Authors" fractal in memory of ourselves. The result of generating this fractal is shown in Figure 6-6. On a distant planet in another galaxy, monster mutant insects are lined up about to do battle, giant pincers poised.... Believe it or not, this fractal resulted from our first attempt. Now it is your turn!

Formula



Category: User Defined

This type invokes the famous Fractint formula parser. You can enter your own fractal formulas using the parser language.

Example:

```
Stars_Like_Jewels  { ; (Well...WHAT would you call it????)
                     ; BG Dodson 1992 71636,1075
  reset type=formula formulafile=fractint.frm formulaname=Jm_14
  function=sin/sqr/sinh passes=b
  corners=2.828701/0.84579/-1.28/1.28/0.84579/1.28 float=y fillcolor=200
  logmap=4 decomp=128
  colors=000<78>ppnuud<7>yyazzMzze<4>zzzGez<23>PzzQ'0<13>'Q0jQ0<13>eN0
  }
```

Formula:

Initialize:	user defined
Iterate:	user defined

Code:

Routine Type	Routine Name	File
Fractal engine	StandardFractactal()	CALCFRACT.C
Initialization and orbits	Formula()	PARSER.C
Assembler speedups		PARSERA.ASM

L-System Fractals

Lindenmayer systems, or L-systems for short, were conceived as a mathematical theory of plant development by Aristid Lindenmeyer. They provide an interesting connection between abstract rules and natural-looking graphics images. Fractint faithfully implements a subset of the L-system language used in books such as *The Algorithmic Beauty of Plants* by Prezemyslaw Prusinkiewicz and Aristid Lindenmayer (Springer-Verlag, © 1990). You can type in examples from the earlier chapters of this book and run them with Fractint.

Although it does not follow that just because a theory successfully models something like plant growth that plants actually use mechanisms analogous to the theory; after a little experience with L-systems your perception of plants will be different. You will find yourself noticing the characteristic branching patterns of your favorite plants. It is amazing how a few simple rules can create complex bushes, leaves, and flowers!.

L-systems can do a lot more than make plant-like images. You will find examples of Koch snowflakes, Penrose tilings, Sierpinski gaskets along with all manner of bushes and shrubs in the FRACTINT.L file.

Accessing an L-Systems File

To generate an L-systems image, press (T) to get the SELECT A FRACTAL TYPE screen, select LSYSTEM, and press (ENTER). If Fractint can find the file FRACTINT.L, you will be presented with a list of L-systems definitions in a screen labeled: LSYSTEM SELECTION FILE: FRACTINT.L. If Fractint cannot find the file in the current directory, you will see instead a standard Fractint file selection and directory navigation screen. You can directly enter the directory where the file FRACTINT.L resides (usually \FRACTINT) or navigate your directory tree by selecting ".." to go up a directory or a subdirectory name to go down. Once you have located FRACTINT.L (or another .L file), select it with the arrow keys and press ENTER. Once in the LSYSTEMS SELECTION screen, if you want to open a different .L file, press F6. Pressing F2 from the LSYSTEMS SELECTION screen shows you the details of the highlighted L-systems.

Once in the LSYSTEMS SELECTION screen, select an L-systems name and press ENTER (try BUSH). This takes you to the PARAMETERS FOR FRACTAL TYPE LSYSTEMS screen. At the bottom of the screen in a box you can see the L-systems definition. L-systems have one parameter, the order. The higher the order, the more times the recursive L-systems process is accomplished. Be careful with this parameter—calculation time increases exponentially with the order. (With this one type, a higher resolution has little to do with the time.) Start with smaller numbers like 2 or 3 and slowly increase, or you may find yourself waiting for hours! (If you are trying the L-systems definition BUSH, an order of 4 works well.) After entering the order, press ENTER to generate the L-systems image. If you have previously selected a video mode, the fractal will begin to generate. Otherwise press a function key to select a video mode, or press DELETE to select a mode from the video mode list. Use as high a video mode as possible; this will not increase the time but will let you see more detail.

The Structure of an L-Systems File Entry

L-systems fractals are constructed from line segments using rules specified in drawing commands. An initial string of drawing commands, called an axiom, is specified, along with an angle that is used with a rotate command. Then transformation rules are defined that give rules for changing strings of drawing commands to other strings. The rules are repeatedly applied to the initial string, generating a sequence of strings of drawing commands. This process is repeated until the user-entered order is reached. Then the drawing commands of the last string are interpreted, and the image is drawn on your screen.

Each L-system entry in the file contains a specification of the angle, the axiom, and the transformation rules. Each item must appear on its own line and each line must be less than 160 characters long. The statement "angle n" sets the angle to 360/n degrees; *n* must be an integer greater than 2 and less than

50. The axiom begins with the keyword `axiom` followed by a series of drawing commands.

Transformation rules are specified as a=string and convert the single character "a" into "string." If more than one rule is specified for a single character, all of the strings will be added together. This allows specifying transformations longer than the 160 character limit. Transformation rules may operate on any characters except space, tab, or "}". The ";" (semicolon) character on a line causes the rest of the line to be treated as a comment.

Drawing Commands

Strings are made up of the turtle graphics drawing commands shown in Table 6-5. This kind of graphics language is called *turtle graphics* because you imagine giving commands to a turtle moving around the screen drawing a picture.

Command	Meaning
F	Draw forward
G	Move forward (without drawing)
+	Turn counterclockwise amount *angle*
-	Turn clockwise amount *angle*
|	Turn 180 degrees.
D	Draw forward
M	Move forward
\nn	Turn counterclockwise nn degrees
/nn	Turn counterclockwise nn degrees
Cnn	Select color nn
<nn	Increment color by nn
–nn	Decrement color by nn
!	Reverse directions (Switch meanings of +, – and \, /)
@nnn	Multiply line segment size by nnn (nnn may be a plain number, or may be preceded by I for inverse, or Q for square root)
[	Push. Stores current angle and position on a stack
]	Pop. Return to location of last push

Table 6-5 L-systems drawing commands

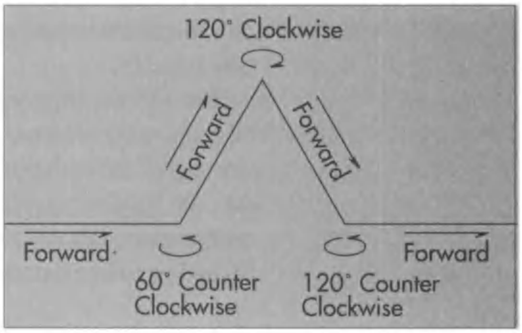


Figure 6-7 Forming the Koch Curve fractal

Other characters are perfectly legal in command strings. They are ignored for drawing purposes, but can be used to achieve complex translations.

How an L-Systems Definition Works

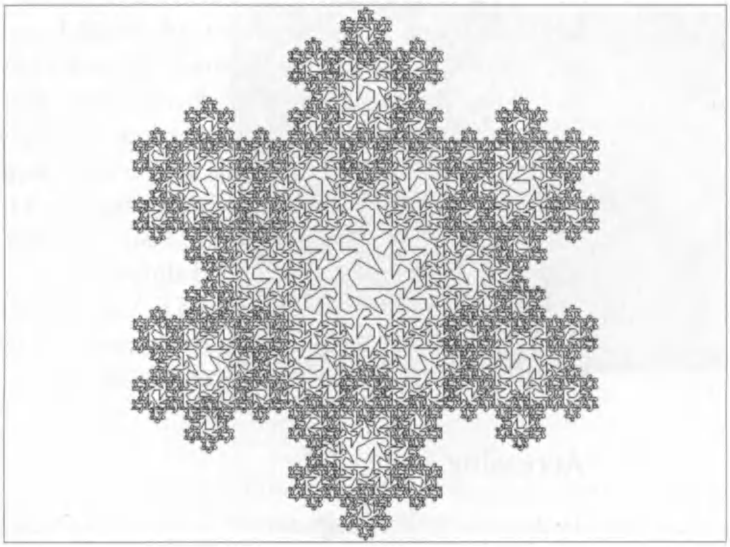
Let's look at the Koch Curve example shown in Figure 6-7. The angle is set to 6, which means 360°/6 or 60°. The axiom is simply the single character F. Therefore if this L-systems was plotted without applying any transformations, the plot would be a horizontal line.

The transformation rule is F=F+F--F+F. This means to replace each F in the current string with F+F--F++F. Understood graphically, it means to replace the segment drawn by F with the more complicated series of segments drawn by F+F--F++F. This string may be interpreted:

F	Go forward one unit
+F	Turn 60° counterclockwise and go forward one unit
--F	Turn 120° clockwise and go forward one unit
++F	Turn 120° counterclockwise and go forward one unit

Figure 6-7 shows how this works. A single segment has a triangular peak bent out of it. It is easy to imagine repeating this "bending" process recursively to get the final Koch Curve shape.

Lsystem



Category: L-Systems

This type allows you to access Fractint's Lindenmayer Systems generator. You can run predefined L-systems images in FRACTINT.L or enter your own with a text editor.

Example:

```
Snowflake          { ; Renders well at 1024x768
                     ; Use a lower order at lower resolutions
  reset=1733 type=lsystem lfile=fractint.l lname=SnowFlake1
  corners=-1/1/-1/1 params=5 float=y
  }
```

Formula: User defined

Code:

Routine Type	Routine Name	File
Fractal engine	lsystem()	LSYS.C
Lsystem asm routines		LSYSA.ASM

Iterated Function Systems

Iterated Function Systems (IFS) is a method of creating fractals developed by Dr. Michael Barnsley, the author of the book *Fractals Everywhere* and founder of the fractal compression company he named, not surprisingly, "Iterated Systems." The IFS method of generating fractals is directly based on the notion of self-similarity, in which a part of an object is a smaller copy of the whole object. The formulas making up the definition of an IFS fractal are mathematical specifications of the self-similarities of the fractal. These formulas are called contractive affine transformations. They are of the form $X' = AX + B$, where A is a matrix and B is a vector. Fractint can generate both two-dimensional and three-dimensional IFS fractals. For 3-D fractals, the matrix A is a 3 x 3 matrix and X', X, and B are three-dimensional vectors. For 2-D fractals, the matrix A is a 2 x 2 matrix and X', X, and B are two-dimensional vectors. We'll discuss these transformations shortly.

Accessing an IFS File

To generate an IFS image, press T to get to the SELECT A FRACTAL TYPE screen, select IFS, and press ENTER. If Fractint can find the file FRACTINT.IFS, you will be presented with a list of IFS definitions defined in that file in a screen labeled: IFS SELECTION FILE: FRACTINT.IFS. If Fractint cannot find the file in the current directory, you will see instead a standard Fractint file selection and directory navigation screen. You can directly enter the directory where the file FRACTINT.IFS resides (usually \FRACTINT) or navigate your directory tree by selecting ".." to go up a directory or a subdirectory name to go down. Once you have located FRACTINT.IFS (or another .IFS file), select it with the arrow keys and press ENTER. Once in the IFS SELECTION screen, if you want to open a different .IFS file, press F6. Pressing F2 from the IFS SELECTION screen shows you the details of the highlighted L-systems.

A 2-D IFS Fractal

Once in the IFS SELECTION screen, select the IFS name FERN and press ENTER. This takes you to the PARAMETERS FOR FRACTAL TYPE IFS screen. There is one choice to make: the coloring method. The default value 0 colors according to how many times a particular screen pixel is written. If you enter 1, the parts of the fractal are colored according to which transformation was used to generate the points. This method lets you clearly see the self-similarities of the fractal.

Use a high resolution mode to view IFS fractals. The images are built up pixel by pixel, and using a higher resolution will not slow the fractal generation, but

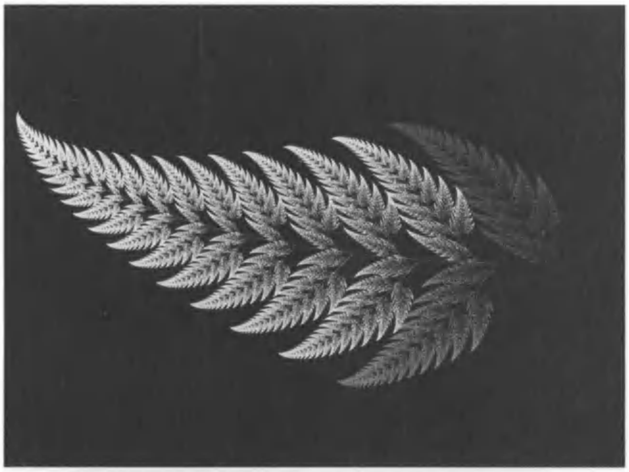


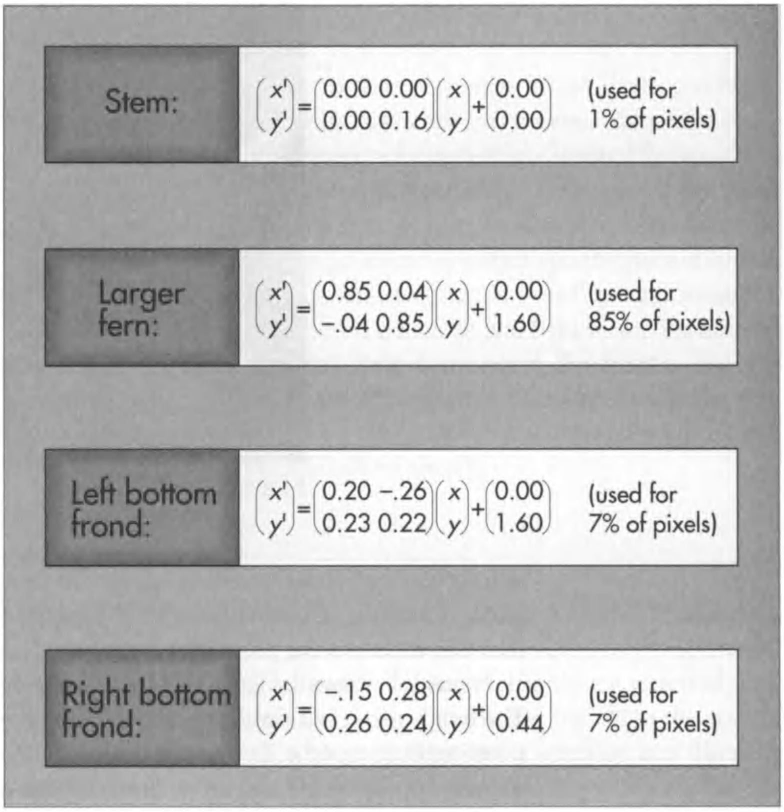
Figure 6-8 IFS Fern

will prevent the pixels from merging together so soon and let you see more detail. Note that the zoom box works with IFS fractals so you can zoom in on details or rotate and skew the zoom box.

Figure 6-8 shows the Barnsley's famous fern fractal using coloring method 1. The fern has four self-similarities, corresponding to the four affine transformations making up the fern entry FRACTINT.IFS. The two most obvious self-similarities are the two lower fronds. These are clearly copies of the whole fern. If you cut away these two fronds and the bottom stem, you will see another self-similarity The remaining fern is a slightly smaller and slightly rotated copy of the original. There is one additional "self-similarity" that you might not expect. This is the stem, which doesn't look like the fern at all, but in a mathematical sense is "similar." The affine transformation that maps the whole fern to the stem compresses it completely into a line segment.

3-D IFS Fractals

The IFS entries in FRACTINT.IFS that begin with "3" are 3-D examples. When you generate them, you will see a 2-D projection onto the screen of the 3-D image, unless you set Fractint to red/blue stereo using the (I) command. See the 3-D Fern example. A 2-D IFS entry has seven numbers—four numbers for the matrix, two for the vector, and one for the probability. A 3-D entry has nine for the matrix, three for the vector, and one for the probability, totaling 13 values. Fractint looks for the "3" in the first letter of the name to help it decide that an IFS entry is 3-D, so if you add new entries, you should follow that convention.



Stem: $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0.00 & 0.00 \\ 0.00 & 0.16 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0.00 \\ 0.00 \end{pmatrix}$ (used for 1% of pixels)

Larger fern: $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0.85 & 0.04 \\ -.04 & 0.85 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0.00 \\ 1.60 \end{pmatrix}$ (used for 85% of pixels)

Left bottom frond: $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0.20 & -.26 \\ 0.23 & 0.22 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0.00 \\ 1.60 \end{pmatrix}$ (used for 7% of pixels)

Right bottom frond: $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} -.15 & 0.28 \\ 0.26 & 0.24 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0.00 \\ 0.44 \end{pmatrix}$ (used for 7% of pixels)

Figure 6-9 Matrix form of Fern IFS equations

The Structure of an IFS File Entry

The FRACTINT.IFS entry for the fern looks like this:

```
fern {
        0.00    0.00    0.00    .16    0.00    0.00    .01
        0.85    0.04   -0.04    .85    0.00    1.60    .85
        0.20   -0.26    0.23    .22    0.00    1.60    .07
       -0.15    0.28    0.26    .24    0.00    0.44    .07
}
```

The first four numbers on each line form the matrix A in the affine formula $AX + B$. The next two numbers form the vector B. The last number is a probability that is used to determine how often an equation is used. In matrix format, these same transformations look as shown in Figure 6-9.

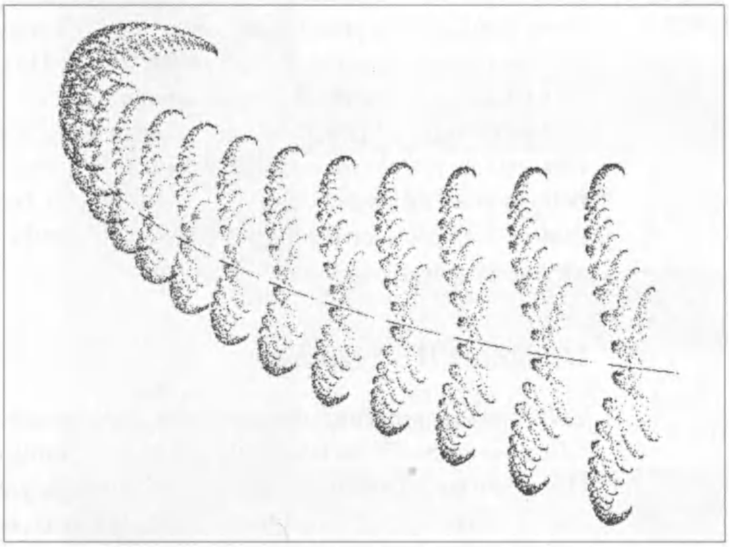
IFS Drawing Algorithm

Here is how the IFS drawing engine works. A starting point is chosen. It really doesn't make any difference what point is used. Then one of the transformations is chosen at random, using the probability assigned to that transformation. The transformation is applied to the initial point to get a new point. This process is repeated, with a different transformation chosen each time. The first few iterations are not plotted on the computer screen, in order to allow the moving point to gravitate toward the fractal. From then on, because all of the transformations map points on the fractal to other points on the fractal, the point dances around the screen and draws the fractal.

Designing IFS Fractals

Unless you are very familiar with matrices and transformations, you will find it difficult to create IFS fractals by directly entering numbers in the FRACTINT.IFS file. (There is no harm in trying, though—you could just type in some numbers and see what happens!) Fortunately, there is a much better way. The program Fdesign will let you interactively design IFS fractals in a visual way, and save the results in an .IFS file. Fdesign is described in Chapter 7, *Making IFS Fractals with Fdesign*.

IFS



Category: Iterated Function Systems

IFS fractals are generated using a special orbit mechanism that is based on user-defined affine transformations. Both 2-D and 3-D fractals can be generated using the method pioneered by Dr. Michael Barnsley.

Example:

```
3D_Fern            {
  reset=1733 type=ifs3d ifsfile=fractint.ifs ifs=3dfern
  corners=-4.999313/7.013138/-4.505173/7.503448 params=1
  rotation=30/10/0 perspective=150 xyshift=0/0 stereo=2 interocular=3
  converge=-5 crop=4/0/0/4 bright=80/100 colors=@glasses2.map
  }
```

Formula: Iterate: $X' = AX + B$

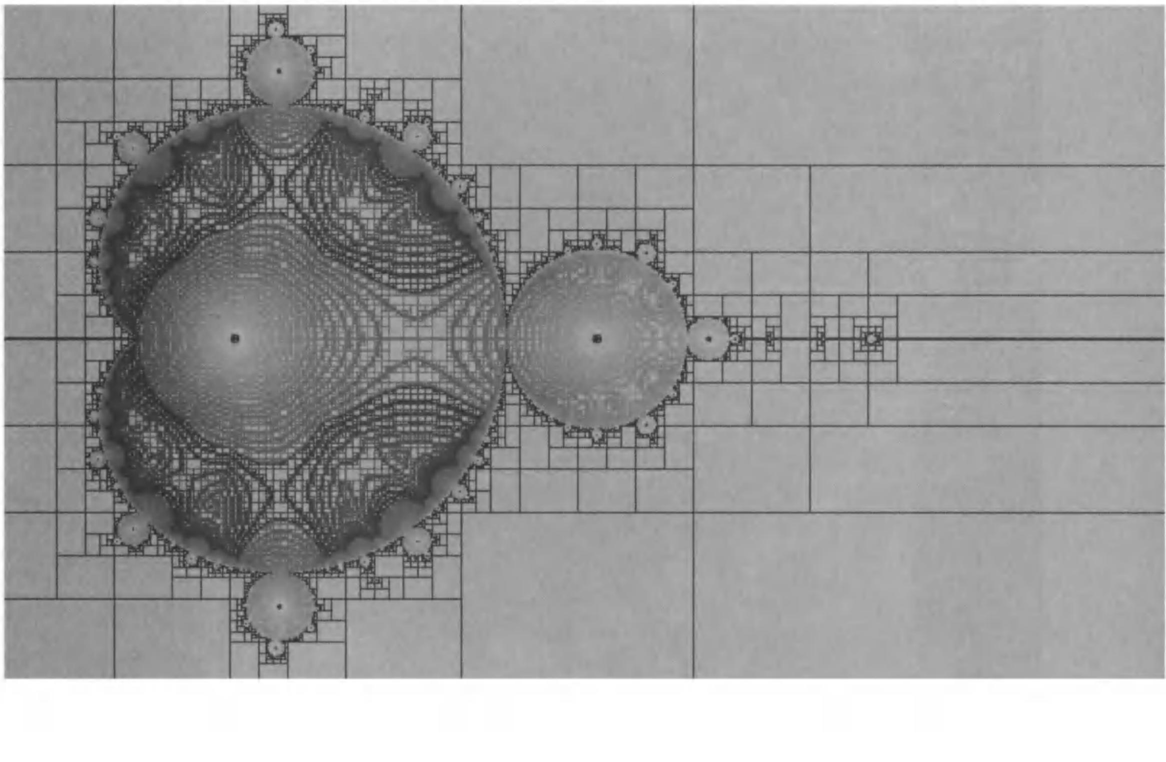
for various randomly selected affine transformations made up of matrices A and vectors B.

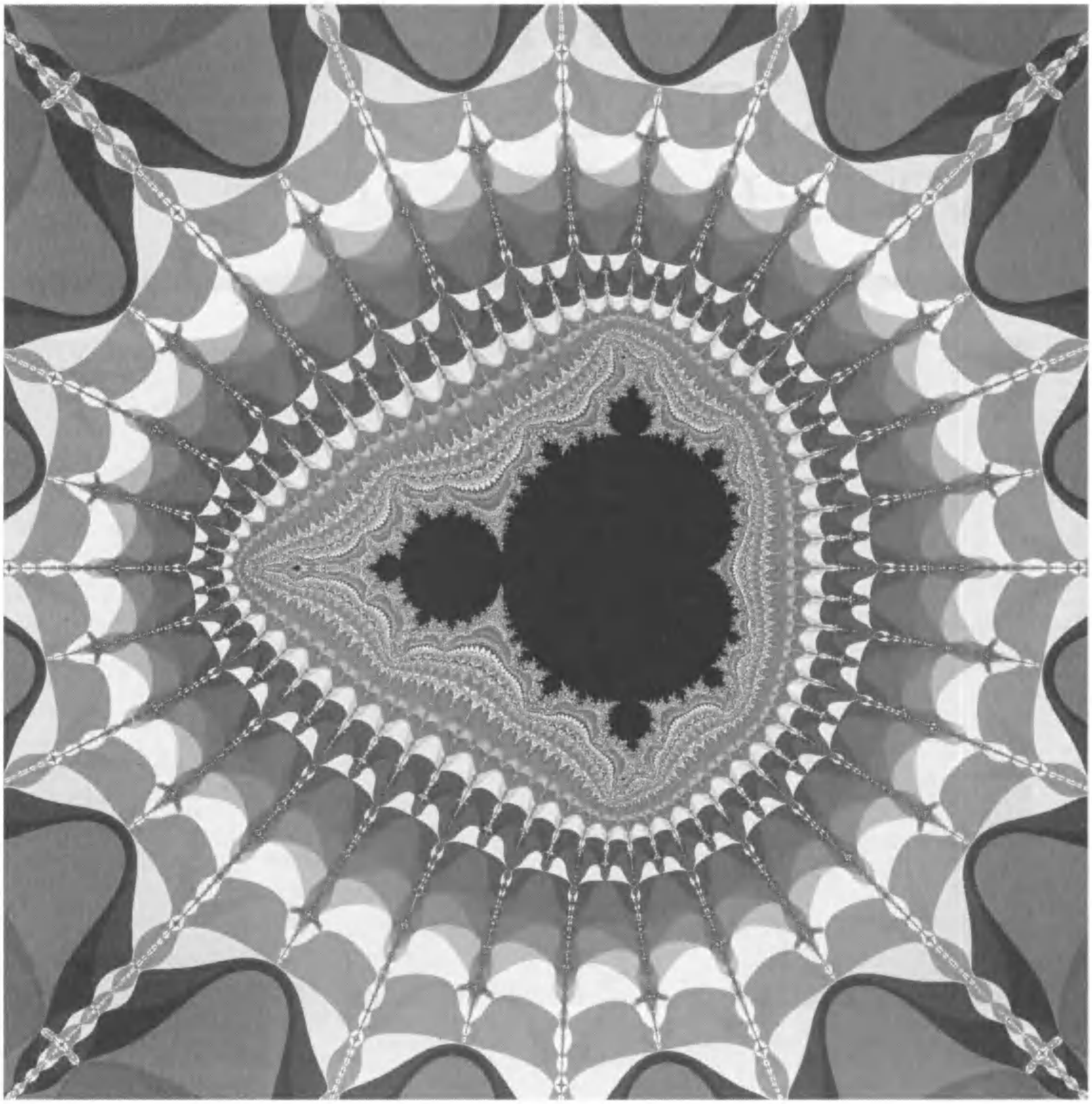
Code:

Routine Type	Routine Name	File
Fractal engine	ifs()	LORENZ.C

CHAPTER 7

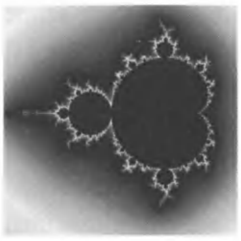
MAKING IFS FRACTALS WITH FDESIGN





CHAPTER 7

MAKING IFS FRACTALS WITH FDESIGN



Fdesign is a public domain program that lets you visually create and edit Iterated Fractal Systems (IFS) fractals. Recall that IFS fractals are especially useful for making images that have a natural appearance. You can make images of trees, shrubs, tiled patterns, nested triangles, and exotic gaskets. You load and save IFS images in either Fractint's FRACTINT.IFS file format or Fdesign's own native format. Best of all, you can see simple representations of the transformations used to generate IFS fractals, and modify them by simply pointing and clicking with the mouse. You can instantly see the effects of your changes on your computer screen.

IFS fractals take the idea of self-similarity to the limit. A mathematical object is self-similar if small parts of itself are similar to the whole. This definition hinges on what is meant by "similarity." For most fractals, the part and the whole might be similar only in the sense that they both have the same degree of roughness and structure (fractal dimension). IFS fractals are self-similar in a much stronger sense; the whole contains true copies of itself. These copies may be rotated, shrunk, compressed, and moved compared to the whole, but they are otherwise identical. The mathematical term for this strong kind of similarity is that the part is the image of the whole under a contractive affine transformation. (A contractive affine transformation is one that rotates, shrinks, compresses, or moves an object, but does not otherwise distort it.)

Consider Figure 7-1. You can see the word "FRACTINT" spelled out in capital letters made up of straight segments. But if you look closely, you will see that each of these letter segments is a miniature copy of the whole word "FRACTINT."

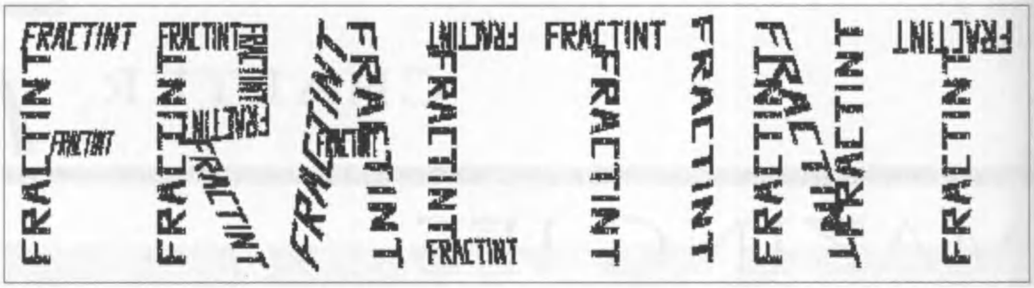


Figure 7-1 The self-similar word "FRACTINT"

These miniature FRACTINTs are not absolutely identical to the whole, but look very much the same. The vertical stem of the "F" is much larger than the horizontal middle stroke. The diagonal, lower right segment of the "R" is slanted. The top of the "C" is upside down. But within the limits of these affine relationships, the small segments are copies of the whole word "FRACTINT." Furthermore, because these small FRACTINTs are copies of the whole, their letters are in turn made up of the word FRACTINT and so on forever.

Fdesign is a program designed to help you explore IFS fractals. This chapter will tell you how to install and run Fdesign, how to read and write IFS files using Fractint's format, how to easily change the transformations to create your own images. You will learn how to load and manipulate 128 different built-in examples from Fdesign as well as the examples in Fractint's own FRACTINT.FRM file. For more background on the theory of IFS fractals, see Chapter 2, *Fractals: A Primer*.

PROGRAM DISTRIBUTION

Fdesign was programmed in Turbo C++ by Doug Nelson. He has released the program to the public domain, so you are free to copy, use, and redistribute it without any fee. The program is publicly available on CompuServe and many BBSs, and is provided on your distribution disk for your convenience.

Fdesign is a wonderful example of high quality, freely distributed software. The program has been written and distributed with the same "Stone Soup" spirit as Fractint, and the Fractint authors are pleased that Doug made Fdesign compatible with Fractint's IFS file format so that both programs can be used together.

UP AND RUNNING

The details of the installation of Fdesign are given in Chapter 1, *Installation*. Because you cannot change directories from inside Fdesign, all the .TRN example files, as well as Fractint's FRACTINT.IFS file, should be in the same directory. Your Fractint directory would be a good place. The FDESIGN.EXE program itself can reside in any directory that is included in your DOS "path."

Hardware Requirements

Fdesign requires the following hardware:

- A Microsoft-compatible mouse for operation. The author reports that some clone mice have not worked.
- An Intel 80x87-compatible math coprocessor greatly speeds up the generation of images, but is not an absolute requirement. To give you an idea of the coprocessor's speed, an Fdesign will run faster on an 8 MHz XT with a coprocessor than on a 20 MHz 386 without a coprocessor.
- EGA or VGA graphics are required, but VGA is much better because the pixel size is square (same number of dots per inch horizontally as vertically).
- You'll also need 350K of memory free before starting Fdesign (less if you don't use the Virtual Screen plot feature).
- Fdesign supports the following printers: HP Laser Jet II or compatible, Epson 24 pin compatible, and Epson 9 pin compatible.

If you have an 80386 or lower class of PC, you should really consider getting a math coprocessor. Prices have dropped to moderate levels (under $100), and for floating-point intensive software such as Fdesign, a math processor provides the single most dramatic upgrade you can buy for your computer. Be aware that an 486SX processor has no floating-point unit, but an 486DX does. Fdesign really flies on 486DX or DX2.

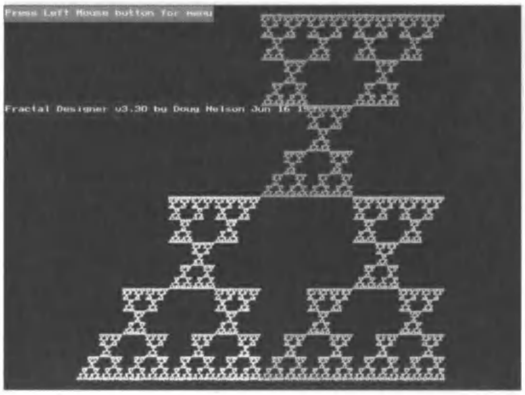

Press Left Mouse button for menu
Fractal Designer v3.30 by Doug Nelson Jun 16 1

Figure 7-2 The Fdesign opening screen

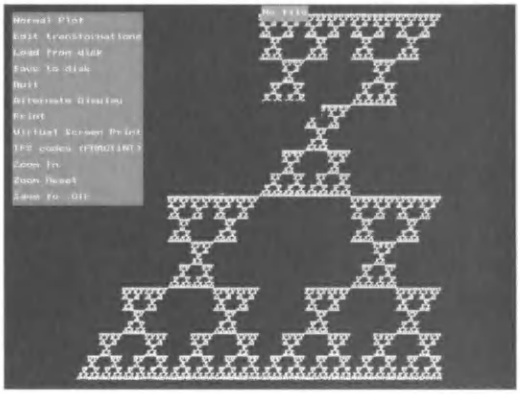


Figure 7-3 The Fdesign Main menu

Fdesign has only modest video requirements, and will run on any EGA or VGA graphics adapter. A virtual screen capability lets you create higher resolution images for printing. You can also save your results in Fractint's format and then regenerate the images using any of Fractint's jillions of drivers at whatever super VGA resolutions your hardware supports.

Software Requirements

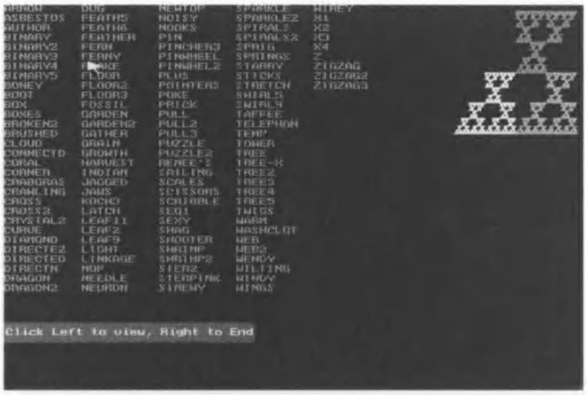
Fdesign works under DOS version 3.0 or later. It runs fine in a full-screen DOS window under Windows 3.1 or OS/2. You need to make sure that a mouse driver is loaded in either your AUTOEXEC.BAT (with a name like "mouse") or in your CONFIG.SYS. Note that Windows 3.1 uses its own mouse driver, so being able to use your mouse with Windows does not mean that the DOS mouse driver is loaded.

Starting Fdesign

Change directories on your hard disk to the directory where the Fdesign .TRN files are located. Make sure the FRACTINT.IFS file is also in the directory. Assuming you have installed Fdesign in your \FRACTINT directory, type

```
cd \FRACTINT(ENTER)
FDESIGN(ENTER)
```

You should then see the opening screen, which looks like Figure 7-2. To get the MAIN MENU shown in Figure 7-3, press any key or click either mouse button.



```
ARROW     DUG       NEWTOP    SPARKLE   WIREY
ASBESTOS  FEATH5    NOISY     SPARKLE2  X1
AUTHOR    FEATH6    NOOKS     SPIRAL    X2
BINARY    FEATHER   PIN       SPIRALS2  X3
BINARY2   FERN      PINCHER3  SPRIG     X4
BINARY3   FERNY     PINWHEEL  SPRINGS   Z
BINARY4   FLAKE     PINWHEL2  STARRY    ZIGZAG
BINARY5   FLOOR     PLUS      STICKS    ZIGZAG2
BONEY     FLOOR2    POINTERS  STRETCH   ZIGZAG3
BOOT      FLOOR3    POKE      SWIRL3
BOX       FOSSIL    PRICK     SWIRL4
BOXES     GARDEN    PULL      TAFFEE
BROKEN2   GARDEN2   PULL2     TELEPHON
BRUSHED   GATHER    PULL3     TEMP
CLOUD     GRAIN     PUZZLE    TOWER
CONNECTD  GROWTH    PUZZLE2   TREE
CORAL     HARVEST   RENEE'S   TREE-X
CORNER    INDIAN    SAILING   TREE2
CRABGRAS  JAGGED    SCALES    TREE3
CRAWLING  JAWS      SCISSORS  TREE4
CROSS     KOCH3     SCRIBBLE  TREE5
CROSS2    LATCH     SEQ1      TWIGS
CRYSTAL2  LEAF11    SEXY      WARM
CURVE     LEAF2     SHAG      WASHCLOT
DIAMOND   LEAF9     SHOOTER   WEB
DIRECTE2  LIGHT     SHRIMP    WEB2
DIRECTED  LINKAGE   SHRIMP2   WENDY
DIRECTN   MOP       SIER2     WILTING
DRAGON    NEEDLE    SIERPINK  WINDY
DRAGON2   NEURON    SINEWY    WINGS

Click Left to view, Right to End
```

Figure 7-4 The Load File screen

Fdesign is a straightforward program to run, and you can probably figure out how to run it just by clicking on the MAIN MENU items and seeing what happens. In the rest of this chapter, we'll take you go through a step-by-step tutorial in Fdesign's operation and intersperse a bit more explanation of what IFS fractals are all about.

For Mouseless Readers... So you don't have a mouse? All is not lost. You won't be able to edit and manipulate IFS fractals but you can still generate the .TRN examples. (The extension "TRN" stands for "Transformations.") For example, to view WEB.TRN, make sure the WEB.TRN is in the current directory, and type `Fdesign Web`(ENTER). You will see the WEB.TRN fractal generating on your screen. You can exit by pressing (ESC). While viewing these IFS files is fun, editing and changing the IFS fractals is even more fun, but for that you need a mouse!

USING FDESIGN

Start Fdesign and click either mouse button to see the MAIN MENU. Fdesign starts with an IFS fractal automatically generating. The colors can be rendered in two different ways. If you click on NORMAL PLOT on the MAIN MENU, the menu will vanish and the fractal will be regenerated using the whole screen. The parts of the fractal are colored according to which IFS affine transformation maps the whole image to that colored subpart. (We'll discuss the affine transformations in a moment.) Click either mouse button to get back to the MAIN MENU. Now click

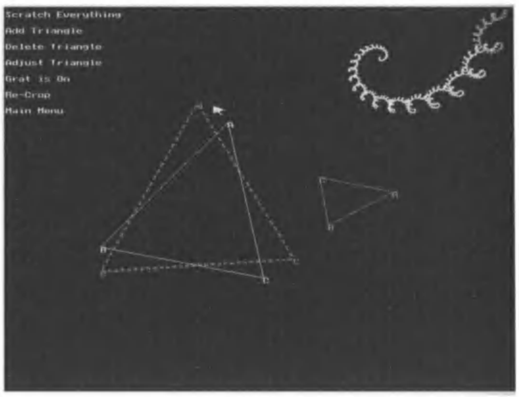


Figure 7-5 The Edit Transformations screen

on ALTERNATE DISPLAY, which is the sixth menu item. This time the pixels are colored according to how often they are written in the process of displaying an image. Fdesign uses the random IFS method of generating fractals, and runs continuously. The image is built up from individual pixels written all over the screen. When a pixel is written that is already colored, the color number is bumped up one.

The Edit Screen

Click either mouse button to get back to the MAIN MENU. Now click on LOAD FROM DISK. You will see a screen that looks like Figure 7-4. The list of file names of the Fdesign .TRN files are shown on the screen. As you click on the names with the mouse, you will see what the fractal stored in that file looks like. If you have a fast 486DX machine or a fast 386 with a math coprocessor, the small fractal will appear to be generated almost instantaneously. Now click on the entry TELEPHON. To accept your selection, click the right mouse button. This will take you to a screen that we'll call the EDIT TRANSFORMATIONS screen, shown in Figure 7-5. The TELEPHON.TRN fractal will still appear in the upper right corner, but you will also see a new menu on the left and some triangles in the center of the screen, superimposed on top of a grid of dots. This screen is the key to understanding IFS fractals and using Fdesign.

The Affine Transformations of the Telephone Fractal

Iterated Function Systems use contractive affine (pronounced uh fine) transformations. An affine transformation changes shapes to other shapes. It can shrink, rotate, move, or compress a shape. These transformations are called "contractive" because they always map images to smaller images. An affine transformation can be completely specified by what it does to a triangle. Fdesign allows you to graphically visualize affine transformations using "before" and "after" triangles. The "before" triangle is drawn with thick dotted lines and colored red. Doug Nelson, the creator of Fdesign, calls this the reference triangle. The corners are labeled A, B, and C.

The TELEPHON.TRN fractal (called the telephone fractal because it looks a lot like a coiled telephone cord) is generated by two contractive affine transformations. These two transformations are shown graphically by two additional triangles, also with the corners labeled A, B, and C, as shown in Figure 7-5. Each of the two solid lined triangles is the result (or image) of applying a different affine transformation to the large dash-lined reference triangle.

First Transformation

Consider first the larger of the two solid-lined triangles shown in Figure 7-5, which we'll call the first triangle. (This triangle appears yellow on your computer screen if you are following along.) Now imagine what it would take to transform the dashed reference triangle into the first triangle, so that the respective corners labeled A, B, and C map to each other. The first triangle is only slightly different from the reference triangle. It is a little smaller, and rotated a few degrees clockwise. So the transformation that maps the reference triangle to the first triangle must shrink a little and rotate clockwise a few degrees.

Now look again at Figure 7-5. The telephone fractal has a little loop in the upper right corner. This loop is purple on your computer screen and shows as a darker gray in the figure. If you were to snip this small loop off of the telephone fractal, the remaining piece would be a little smaller than the original and rotated slightly clockwise. The relationship between the whole telephone fractal and the piece remaining after snipping the small loop is exactly the same as the relationship of the reference triangle to the first triangle—slightly smaller and rotated clockwise. We have identified a self-similarity in the telephone fractal.

Second Transformation

The second small triangle shown in Figure 7-5 (purple on your computer screen, a darker gray in the figure) determines another affine transformation, which in turn is related to another self-similarity of the telephone fractal. This small triangle is much smaller than the reference triangle, and the corners A and C have been swapped. One way to swap these two corners would be to flip the triangle over like a pancake. Now consider what would happen if you shrunk the telephone fractal to a quarter of its size and flipped it upside down. It would look just like the small tail of the telephone fractal.

On your color computer screen it is easy to keep track of which self-similarities are connected to which triangle because they are color coded. In the case of the telephone fractal, the large yellow triangle (lighter in Figure 7-5) is related to the large yellow piece of the fractal. Similarly, the small purple triangle (darker in the figure) is related to the small purple tail of the fractal.

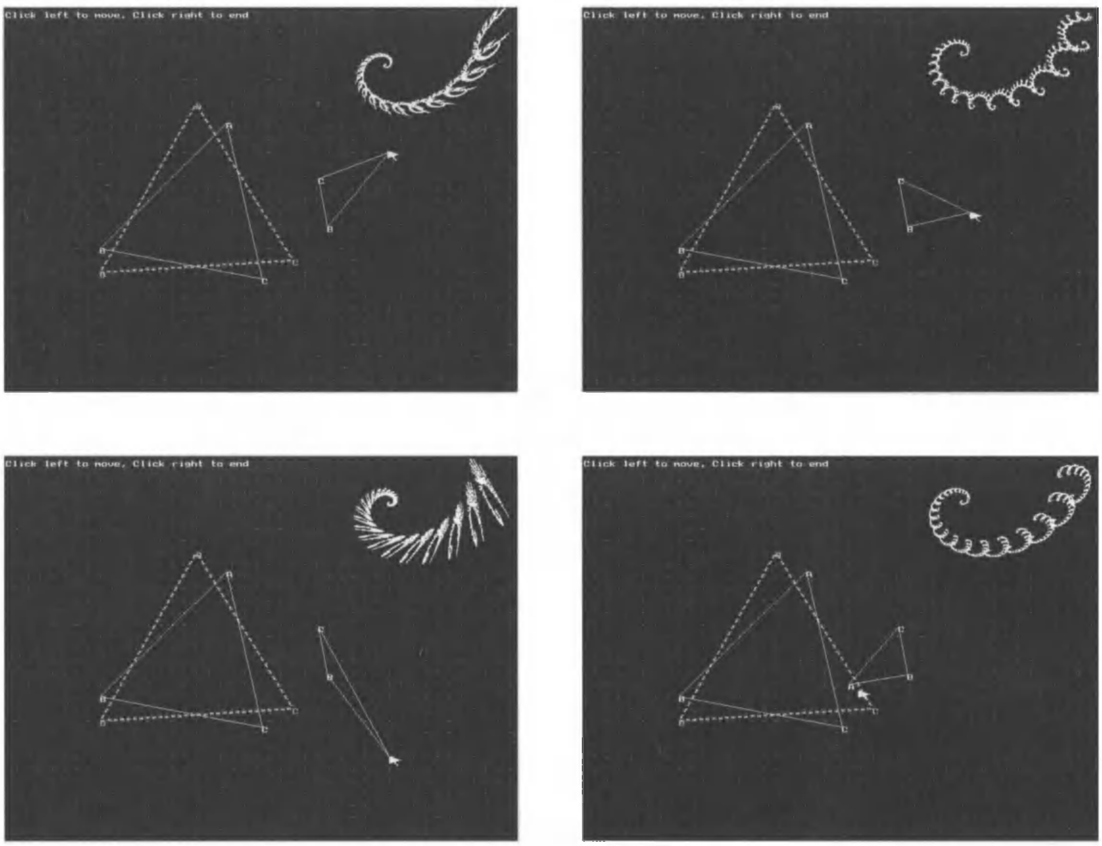
How Fdesign Works

An IFS fractal is generated by affine transformations that rotate, shrink, compress, or move objects. Each transformation maps the whole fractal to a self-similar part. There will be as many self-similar relationships in a fractal as there are transformations used to generate the fractal. An affine transformation is determined by two triangles: a reference triangle, and the triangle that results by applying that transformation to the reference triangle. Fdesign lets you edit affine transformations that generate fractals by manipulating triangles that determine those transformations.

Adjusting Affine Transformations

Now for the fun. Suppose you change these transformations, then how would the image be changed? To find out, click on the ADJUST TRIANGLE menu item. You will then see the prompt SELECT TRIANGLE TO MOVE. Point to the small purple triangle, and click the left mouse button. Then Fdesign will ask, SELECT CORNER TO CHANGE. Point the mouse cursor at corner A of the purple triangle, and click the left mouse button. Now point the mouse a small distance from corner A of the purple triangle, and click the left mouse button again. Like magic, corner A will jump to where the mouse cursor was pointing, and the fractal will change before your eyes! Figure 7-6 shows the results of several such experiments. Just keep pointing the mouse to different places and clicking the left mouse button. If you make the triangle too big, Fdesign will give the error message "not affine." Affine transformations used to generate IFS fractals must be contractive; that is, they must make the triangles at least slightly smaller. If you see this error, just click the left mouse button and continue, trying points that make the triangle smaller.

When you are satisfied with the current fractal, press the right mouse button to return to the edit menu. Click on MAIN MENU, and you will see your IFS fractal filling the whole screen. Clicking either mouse button one more time will restore the MAIN MENU.


Click left to move, Click right to end

Click left to move, Click right to end

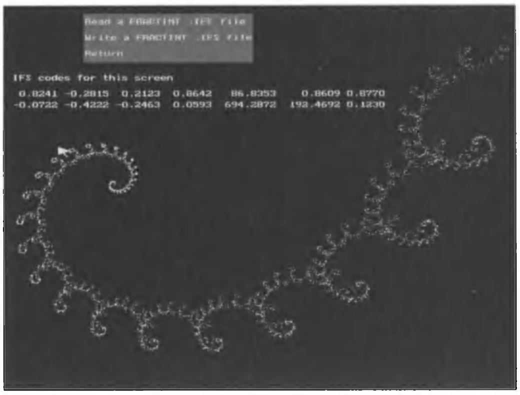
Click left to move, Click right to end

Click left to move, Click right to end

Figure 7-6 Transformation adjustment experiments

Saving Your Fractal

You can save your work three different ways in Fdesign. Clicking on SAVE TO DISK at the MAIN MENU saves the IFS parameters in Fdesign's native .TRN format. You can also save the image as a GIF by clicking on SAVE TO .GIF. (Unlike Fractint, Fdesign does not save the fractal parameters with the GIF file, but only saves the image.) The third way to save the file is as a Fractint-format .IFS file entry. To do this, click on IFS CODES (FRACTINT) on the MAIN MENU. You will then see the IFS codes on the screen, along with a menu with options to read from or write to an .IFS file, as shown in Figure 7-7. Click on WRITE A



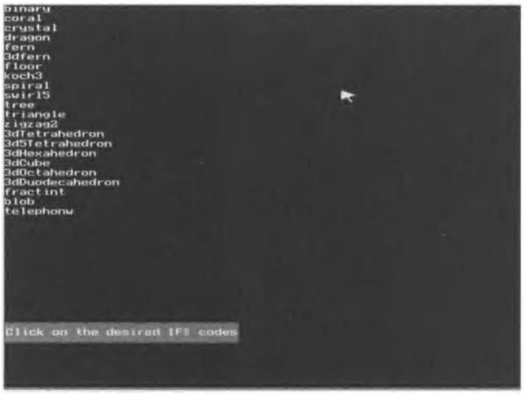
```
IFS codes for this screen
 0.8241 -0.2815  0.2123  0.8642   86.8353    0.8609 0.8770
-0.0722 -0.4222 -0.2463  0.0593  694.2872  192.4692 0.1230
```

Figure 7-7 The IFS Codes (Fractint) screen

FRACTINT .IFS FILE. Fdesign will then ask you to fill in an .IFS file name. Type in the file name **MYIFS** and press ENTER. (There is only room to for you to write in the file name without the extension—Fdesign supplies the .IFS extension.) This will add to the MYIFS.IFS file, or create this file in the current directory if it doesn't exist. Then you will be prompted for the NAME OF IFS CODES. Type in **TELEPHONE**. Click on Return to restore the MAIN MENU to the screen, and then click on QUIT to return to the DOS prompt.

Loading Your Fractal into Fractint

Now that we've made and saved an IFS fractal using Fdesign, let's load it into Fractint so we can display it using different resolutions. Start Fractint in the same directory where the MYIFS.IFS file was saved. Because Fdesign only works with files in the current directory, this will be the directory that was current when you exited Fdesign. Press T to get the SELECT A FRACTAL TYPE screen, and type in **IFS** (or select with the cursor keys) and press ENTER. If the file FRACTINT.IFS is in the current directory, you will be presented with a list of named IFS parameter sets in the FRACTINT.IFS file. Press F6 to see the list of .IFS files in the current directory. (If Fractint could not find the FRACTINT.IFS file in the current directory, you will be taken directly to a list of .IFS files without having to press F6.) Select MYIFS (the file where we saved the TELEPHONE fractal) from the list of files using the arrow keys, and press ENTER. You will then see a list of fractals stored in MYIFS.IFS. (You can put more than one fractal into the MYIFS.IFS file, but so far we have only added TELEPHONE, so unless you saved other fractals, you will only see this one entry.) Select TELEPHONE with the cursor keys and



```
binary
coral
crystal
dragon
fern
3dfern
floor
koch3
spiral
swirl5
tree
triangle
zigzag2
3dTetrahedron
3d5Tetrahedron
3dHexahedron
3dCube
3dOctahedron
3dDuodecahedron
fractint
blob
telephonw

Click on the desired IFS codes
```

Figure 7-8 The IFS entries in FRACTINT.IFS

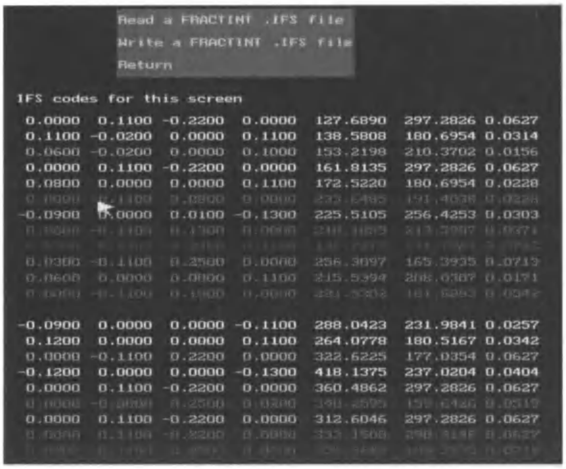
press (ENTER). At the Parameters For Fractal Type IFS screen, you can select one of two different coloring schemes. The default coloring scheme (Coloring Method set to 0) is the same as Fdesign's "Alternate Display." If you enter 1 for the Coloring Method, Fractint will use a coloring scheme similar to Fdesign's "Normal Plot" that colors pixels according to which transformation was selected to draw the pixel. Change Coloring Method to 1 and press (ENTER).

Select the highest video mode your graphics equipment will support. Many newer super VGA adapters have a 1024 x 768 pixel 16-color (SF3) mode. If you know your graphics adapter/monitor combination supports that mode, press (SHIFT)-(F3), or else press (DELETE) to see the mode list, select a mode with the arrow keys, and press (ENTER). A 16-color mode is fine for IFS fractals.

You can see that IFS fractals look better in high resolution modes. You can use a text editor (such as the DOS program EDIT) to edit the numbers in the FRACTINT.IFS file. However, this is a difficult way to experiment with IFS fractals because editing the numbers is less intuitive than using the mouse to edit IFS transforms in Fdesign. But Fractint has much more powerful video support than Fdesign. You can have the best of both worlds by using Fdesign to create your IFS fractals, and then move IFS fractals from Fdesign to Fractint to display it in a high resolution super, VGA video mode.

Loading and Editing a FRACTINT.IFS Fractal in Fdesign

Now let's try opening some of Fractint's IFS fractals in Fdesign. If you are still running Fractint, exist by pressing (ESC) several times and answering the Exit From Fractint (Y/N)? prompt by typing y.



```
IFS codes for this screen
 0.0000  0.1100 -0.2200  0.0000  127.6890  297.2826 0.0627
 0.1100 -0.0200  0.0000  0.1100  138.5808  180.6954 0.0314
 0.0600 -0.0200  0.0000  0.1000  153.2198  210.3702 0.0156
 0.0000  0.1100 -0.2200  0.0000  161.8135  297.2826 0.0627
 0.0800  0.0000  0.0000  0.1100  172.5220  180.6954 0.0228
 0.0000  0.1100  0.0800  0.0000  233.6485  191.4038 0.0228
-0.0900  0.0000  0.0100 -0.1300  225.5105  256.4253 0.0303
 0.0000 -0.1100  0.1300  0.0000  [illegible]  [illegible] [illegible]
 [illegible]
 0.0300 -0.1100  0.2500  0.0000  256.3097  165.3935 0.0713
 0.0600  0.0000  0.0000  0.1100  215.5394  208.0387 0.0171
 0.0000 -0.1100  0.1900  0.0000  [illegible]  [illegible] [illegible]

-0.0900  0.0000  0.0000 -0.1100  288.0423  231.9841 0.0257
 0.1200  0.0000  0.0000  0.1100  264.0778  180.5167 0.0342
 0.0000 -0.1100  0.2200  0.0000  322.6225  177.0354 0.0627
-0.1200  0.0000  0.0000 -0.1300  418.1375  237.0204 0.0404
 0.0000  0.1100 -0.2200  0.0000  360.4862  297.2826 0.0627
 0.0000 -0.0800  0.2500  0.0200  [illegible]  [illegible] 0.0519
 0.0000  0.1100 -0.2200  0.0000  312.6046  297.2826 0.0627
 0.0000  0.1100 -0.2200  0.0000  333.1500  [illegible] 0.0627
 [illegible]
```

Figure 7-9 The 22 affine transformations that generate the word "Fractint" as numbers

If the file FRACTINT.IFS is not in the current directory with the Fdesign .TRN files, copy it with the command `copy \FRACTINT\FRACTINT.IFS` ENTER, where "\FRACTINT" is your Fractint directory. Start Fdesign by typing `fdesign`ENTER and click either mouse button to see the MAIN MENU. Click on the menu item IFS CODES (FRACTINT). You should then see once again the IFS codes menu shown in Figure 7-7. This time click on READ A FRACTINT .IFS FILE. At the READ FROM WHAT .IFS FILE prompt, fill in the name FRACTINT and press ENTER. You will then see a list of all the IFS entries in FRACTINT.IFS shown in Figure 7-8. Note that some of these entries begin with "3D." These entries don't work in Fdesign. (How to manipulate a 3-D triangles with a mouse is a tough problem for a programmer!) If you select a 3-D IFS fractal by mistake, you will briefly see the message "Number of triangles was zero" and will be taken back to the READ/WRITE FRACTINT IFS FILES menu.

Select the IFS fractal FRACTINT from the list by pointing at the mouse and clicking. You will briefly see the message "22 IFS codes read in" and will be taken back to the READ/WRITE FRACTINT IFS FILES menu.

Click on RETURN to get to the MAIN MENU. You will see the message "No File" at the top of the screen. You haven't made a mistake. "No File" simply means that the current IFS fractal did not come from an Fdesign .TRN file. To see the fractal, click on NORMAL PLOT. Eureka! The word "Fractint" will appear on the screen! It should look very much like Figure 7-1 at the beginning of this chapter. Each letter is made up of small versions of the word "Fractint."

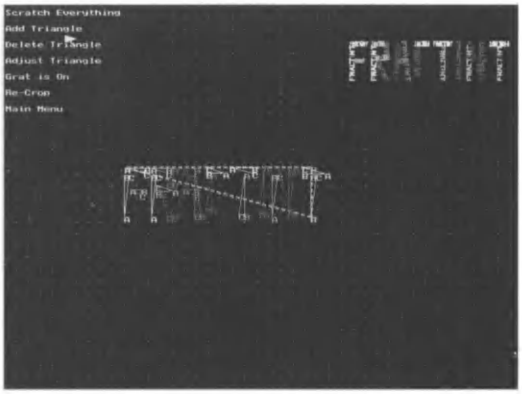


Figure 7-10 The Edit Triangles screen

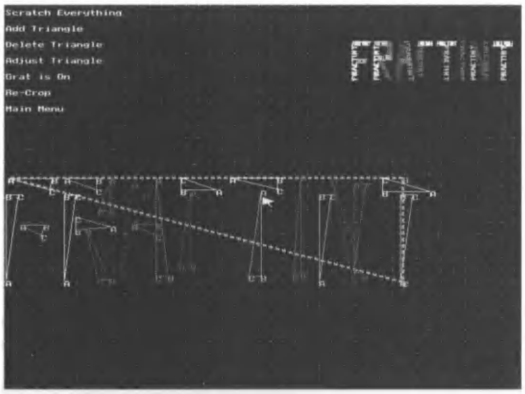


Figure 7-11 Recropped edit triangles

This IFS fractal uses 22 different transformations, one for each "pen stroke" needed to spell the word. Each transformation maps the whole word "Fractint" to one segment of a letter. We'll show you two ways to see those transformations. Click a mouse button to get the MAIN MENU, and then click once again on IFS CODES (FRACTINT).

Figure 7-9 shows the codes making up the 22 affine transformations spelling "Fractint." We'll let you in on a little secret. Even Dr. Michael Barnsley, the inventor of IFS fractals and a method of using them to compress images, couldn't tell by looking at those numbers that the resulting fractal spells the word "Fractint"! So don't worry that the numbers don't mean a lot to you. Just be thankful that the Fdesign program makes it easy to edit the numbers in a way that is easy to understand. Click on RETURN to get to the MAIN MENU.

To see these transformations in a more understandable form, click on EDIT TRANSFORMATION. You will see the screen shown in Figure 7-10. The triangles spelling the word "Fractint" are a bit small to see easily, so let's make them larger. Click on RECROP. This creates a zoom box that represents the size of the edit triangles after recropping. Usually, fractal program zoom boxes are for the purpose of letting you zoom in on fractal details, but in this case the zoom box is a convenience to make it easier for you to edit the IFS triangles. Make the zoom box as big as possible by holding down the left mouse button and pulling the mouse toward you. Then position the large zoom box in the lower right corner by moving the mouse without holding down any mouse buttons. Try adjusting the zoom box so it won't interfere with the fractal image in the upper right corner by holding down the left mouse button and pushing the mouse away from you. Click both mouse buttons together, and you should see the larger set of edit triangles shown in Figure 7-11.

Figure 7-12 Author's IFS creation

Creating a New IFS Fractal

Ready for some real fun? Now let's invent a brand new fractal. From the EDIT menu, click on SCRATCH EVERYTHING. You will then be prompted to create a reference triangle. This triangle should be as large as possible, but leave the upper right corner free for the fractal image. Move the mouse cursor to the upper left corner, and click the left mouse button. The letter A will appear. Then move about 2/3 of a screen to the right and click again. The letter B will appear with a dashed line connecting it with point A. Finally, move the cursor somewhere near the bottom of the screen and click again. The reference triangle is now complete. In order to create an IFS fractal, you will need to create two additional triangles. They should be smaller than the reference triangle. Continue to point and click at different positions on the screen. As soon as you have completed six additional points, the fractal will appear in the upper-right corner. But don't stop, keep clicking and adding more triangles. Like magic, as fast as you complete a triangle, the modified fractal will appear on the screen! When you are satisfied, click the right mouse button to return to the edit screen.

The results of the unbridled creativity of one of your authors is shown in Figure 7-12. We're sure you can do better!

Printing with Fdesign

Fdesign can display IFS fractals at only 640 x 480 pixels, but it can print them at the much higher resolution of 1504 x 1200 using the virtual screen capability. You can also print images displayed on the screen, along with the transformation triangles and codes.

After you have generated a fractal image that you want to print, click either mouse button to return to the MAIN MENU. Then click on PRINT. You will be prompted to select a printer. Three printers are supported, Epson 9 pin, Epson 24 pin, and Hewlett-Packard Laserjet II. Almost every printer can emulate one of these, so if you have a different printer, check your printer documentation. Click on the desired printer name, then click on CONTINUE. Fdesign will print your fractal, the triangles, and the IFS codes on one page.

For a higher quality printout, click on VIRTUAL SCREEN PRINT from the MAIN MENU. You must first generate the fractal using an invisible virtual screen, and then print it. Click on PLOT 1000K POINTS. (If the result turns out to have too few points to look good, you can click on PLOT N POINTS instead, and fill in a number larger than 1,000.) A virtual plot status display will come on your screen, allowing you to monitor progress. When the PLOT POINTS menu returns, click on PRINT, and once again select a printer as before. Your fractal will fill a whole page at high resolution when printed.

AN FDESIGN SAMPLER

A really good way to use Fdesign is to start with the great collection of images that the Fdesign programmer provides with the program. You can load and view these images, and alter the IFS transformations to make your own fractals.

Natural Fractals

IFS images can look very lifelike because self-similarity is a characteristic of some natural phenomena, particularly plants. If the IFS transformations mimic the actual self-similarity of a plant, the result will look very much like the plant.

Tree

From the MAIN MENU of Fdesign, click on LOAD FROM DISK. Point the mouse cursor at TREE and click the right mouse button. This will take you to the edit screen for the TREE fractal. The transformation triangles for this fractal are shown in Figure 7-13. There are three IFS transforms, one that makes the tree branch to the left and one to the right. The remaining transformation generates the stem. The "triangle" that makes the stem is interesting because it is not a true triangle at all. If you make two of the corners of a transformation triangle coincide, the "triangle" becomes a line segment, and the part of the fractal generated by this transformation will be a segment.

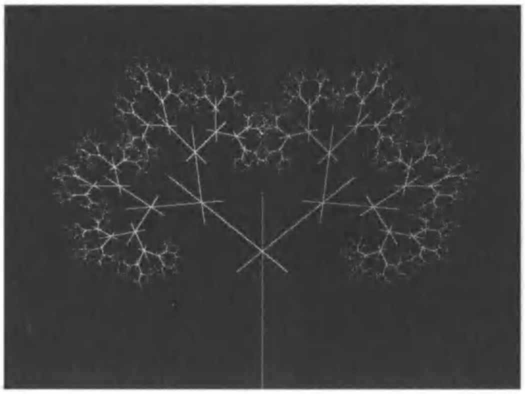


Figure 7-13 TREE.TRN

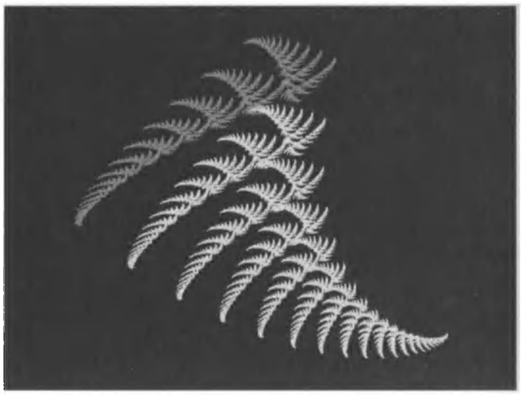


Figure 7-14 FEATHER.TRN

The resulting tree is interesting because the foliage on the tree is made entirely of stems! The stems continually branch in two directions, until you are no longer aware of individual stems, but only the bushy tree shape. To see a full screen view of the tree, click on MAIN MENU.

Feather

Click on LOAD FROM DISK again, and this time load the image FEATHER by pointing and clicking with the right mouse button. This fractal is shown in Figure 7-14. The two transformations are similar to those for the TELEPHON.TRN fractal discussed earlier. One of the transformations generates a slight shrinkage and rotation of the image, creating a gentle curve. The other transformation creates miniatures of the whole fractal. The result looks a lot like a feather.

Grain

Now load GRAIN.TRN. This one has a symmetry very much like a grain of wheat. Where the feather fractal had a transformation with a slight rotation, this one turns nearly (but not exactly) 90°. Because the A and B corners were swapped in the larger triangle, the components of the grain flop back and forth, resulting in a nested series of grain fronds at nearly right angles to each other.

Mathematical Monsters

The key to many natural looking fractals are gentle spirals and almost-exact turns. You will get completely different and very unnatural results from

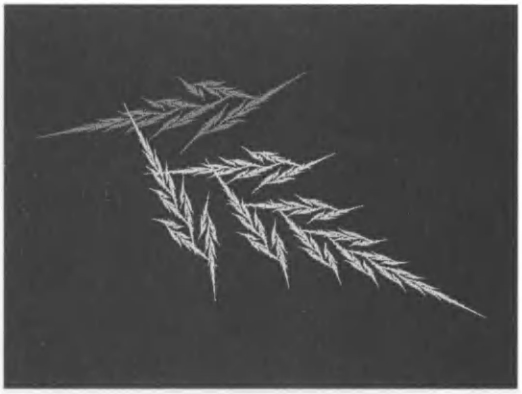


Figure 7-15 GRAIN.TRN

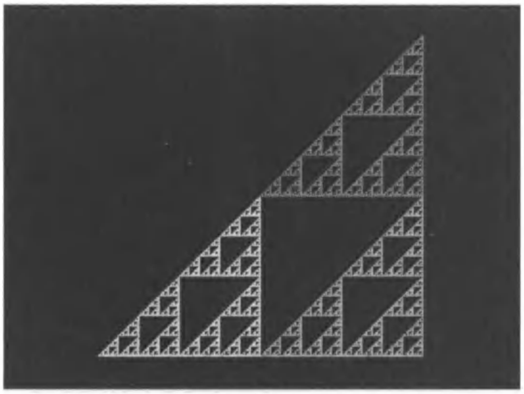


Figure 7-16 Sierpinski gasket

transformations that accomplish even turns and contractions by integral factors. Let's look at some examples.

Binary

Load the file BINARY.TRN. The triangles have a very orderly look about them, and the result shows in the resulting fractal. This fractal looks like a mathematical gasket. (A gasket is a flat object that has pieces cut out of it.) Click on ADJUST TRIANGLE and select the top triangle by pointing and clicking, then at the prompt SELECT CORNER TO MOVE, point at corner C. Move this corner directly down so it coincides with corner C of the lower left triangle. The result should look like Figure 7-16. Voila! A Sierpinski gasket!

Cross

Load the file CROSS.TRN and look at the transformation triangles. By now you probably have memorized the commands. From the MAIN MENU of Fdesign, click on LOAD FROM DISK. Point the mouse cursor at CROSS and click the right mouse button. This will take you to the edit screen for the CROSS fractal, shown in Figure 7-17. The triangles are the picture of symmetry, one in the center, and four others rotating around the corners of an invisible square that would just contain the center triangle. The fractal reflects the highly symmetrical transformations. This fractal has five-fold self-similarity. (Of course, you might already have guessed that from the fact that there are five IFS transformations!)

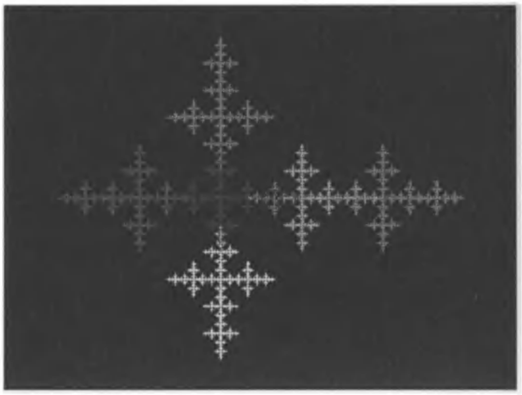


Figure 7-17 CROSS.TRN

MOVING ON

Table 7-1 summarizes all the commands of Fdesign. You can use this as a handy reference to the program. As you have seen, Fdesign is easy to use and meshes well with Fractint. With Fdesign, you have an interactive fractal design tool that makes it easy to visualize the transformations that generate IFS images, modify them, and save them for later display by Fractint.

Main Menu

Normal Plot	Plots (to the screen) the current fractal in EGA or VGA 16 colors.
Edit Transformations	Allows you to change the triangle transformations defining the fractal.
Load from Disk	Loads a transformation (.TRN) file. Click the left mouse on a file name to preview the fractal in that file. When you see the one you want, click the right mouse button. You will move to the Edit Transformations screen. If you want a big screen image, click on MAIN MENU immediately.
Save to Disk	Save the current transformations to a file. Type in the file name (without the extension) that you want to save the transformations to. If the file name already exists, you will be prompted for an OK. Saving to another subdirectory or disk is not supported.
Quit	Returns you to DOS.
Alternate Display	Plots in 16 colors, but starts with color 1 and increments the color every time a pixel is replotted.
Print	Prints the file name, the current screen, the edit transformations, and IFS codes. This is a good way to document your IFS fractals.
Virtual Screen Print	Creates high-resolution pictures on a printer.
IFS Codes (Fractint)	Allows transferring files to/from Fractint.
Zoom In	Brings up a zoom box. The box may be moved by moving the mouse. The size of the box may be changed by holding the left mouse button and moving the mouse up or down. To accept the zoom box, press both mouse buttons, or type any key. Multiple zooms are allowed, but the computation time required increases with every zoom.
Zoom Reset	Reset to the original full-sized image.
Save to .GIF	Copies the screen to a .GIF format graphics file.

Edit Transformations Menu

Scratch Everything	Cleans the work space, deleting the current fractal from memory. You may then begin again to define a new fractal, specifying a reference triangle and transformation triangles until your right mouse button ends the input.
Add a Triangle	You enter three points defining an additional triangle.
Delete Triangle	You select a triangle to delete.

Table 7-1 Fdesign menu summary

Edit Transformations Menu (continued)

Adjust Triangle	Allows you to move a single vertex of a triangle. First select a triangle by finding an unambiguous edge of the triangle and click the left mouse button. Then click on the corner you want to move. Now you may move the cursor to the new position and click the left mouse button. The fractal image in the upper right updates with your edits. You may continue to click the left mouse button and watch the fractal change in the upper right corner. When you're satisfied, click the right mouse button.
Grat is On/Off	Clicking on this will toggle the "jump to grid" on or off. If grat is off, you may place triangle vertices between dots. If grat is on, the closest graticule will be used to place the vertex.
Recrop	Alters the magnification of the editing triangles
Main Menu	Return to plot the fractal on the big screen, then return to MAIN MENU when done.

Virtual Screen Print Menu

Plot 1000000 Points	Plots a million points to a virtual screen (1504 x 1200). A million points is adequate for most images. This number should be increased if you have already zoomed on the image.
Plot N Points	Plots to the virtual screen with as many points as you specify. Occasionally, you may need to plot 20 million points to get a full picture. You specify the number of points in thousands (5,000 means 5 million).
Print	Prints the virtual screen to your printer on LPT1.
Return	Gets you back to the top menu.

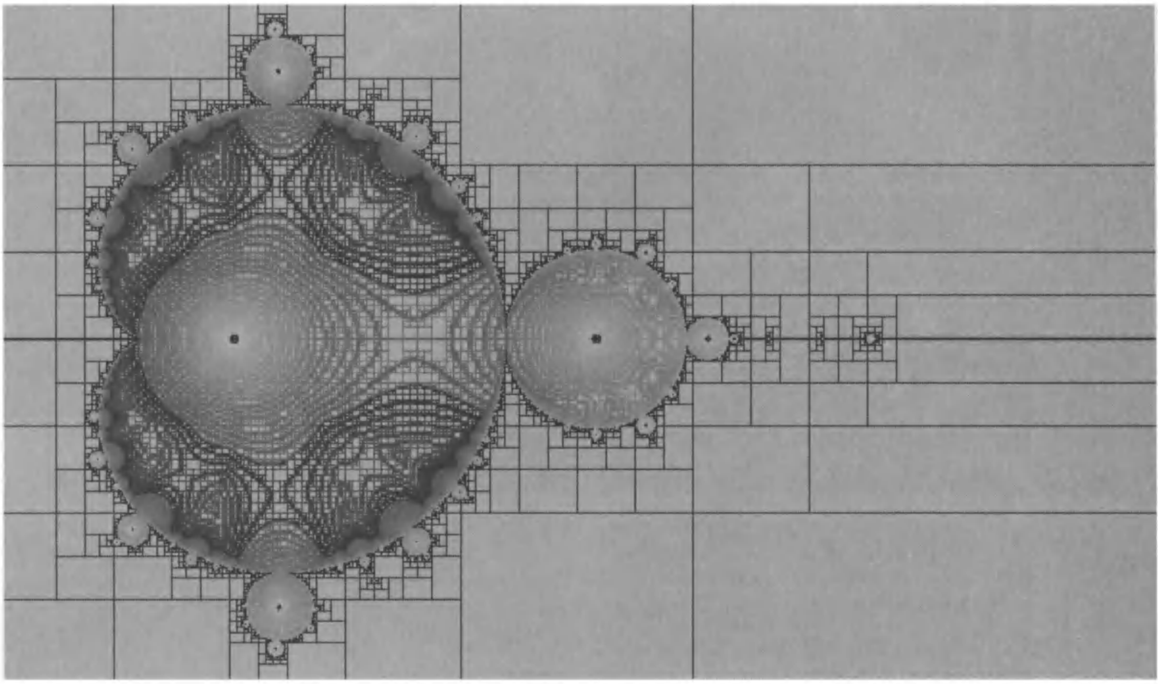
IFS Codes (Fractint) Menu

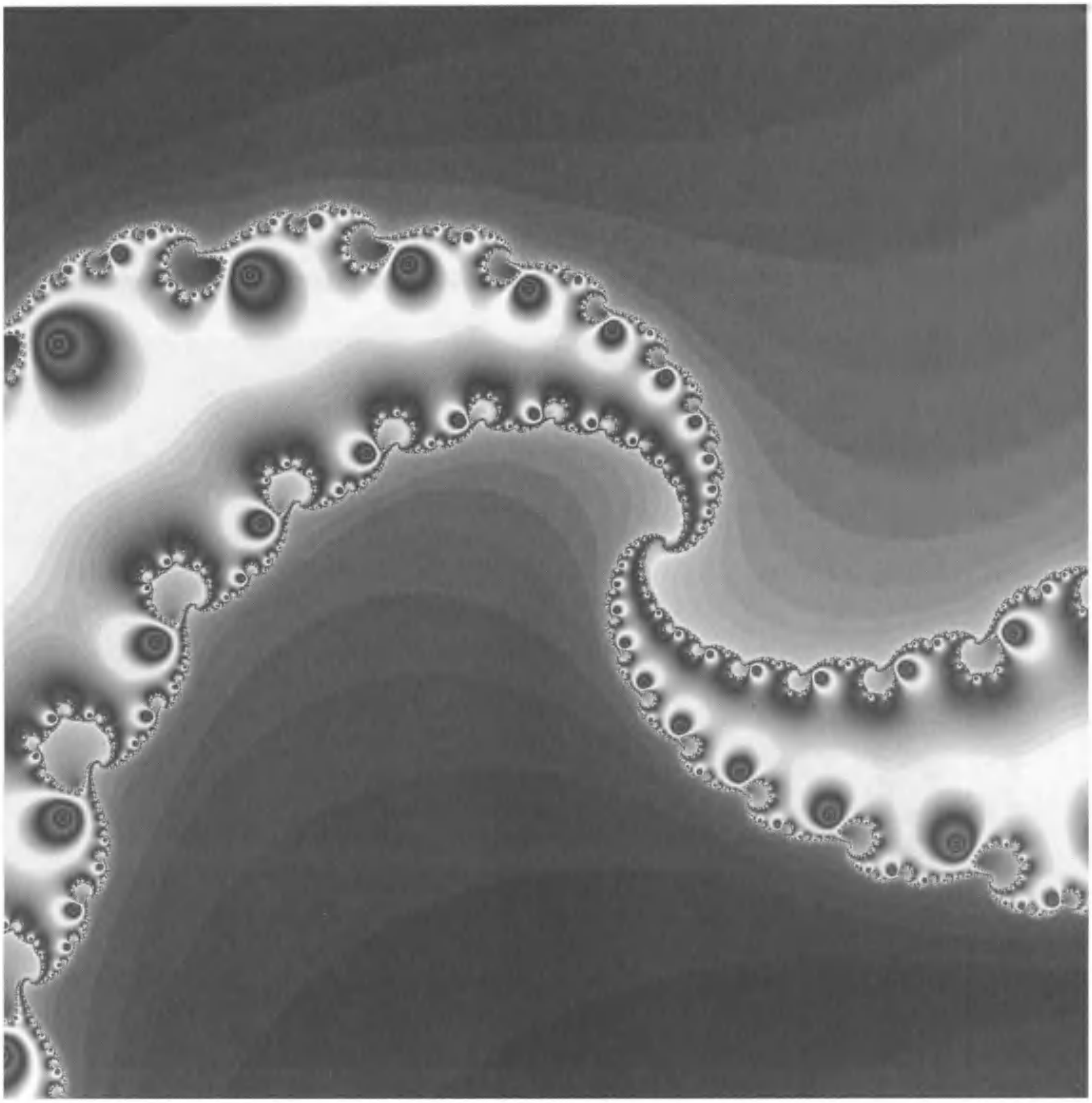
Write a Fractint File	Computes IFS codes for Fractint's screen size and writes text file with an extension of .IFS.
Read a Fractint File	Computes triangle transformations from the IFS codes. You may then edit the transformations as with any Fdesign file.

Table 7-1 Fdesign menu summary (*continued*)

CHAPTER 8

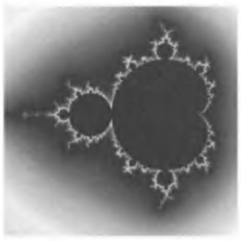
FRACTINT'S SOURCE CODE





CHAPTER 8

FRACTINT'S SOURCE CODE



If you are a programmer, you are probably curious about some of the coding magic buried in the innards of Fractint. Perhaps you want to experiment with a new fractal type that Fractint doesn't handle yet. Perhaps you want to see why Fractint's GIF encoder is faster (or slower) than yours. Maybe you just want to see how Fractint manages to access the IBM 8514/A video adapter without the need for IBM's HDILOAD API. Or maybe you want to borrow some portion of our program and use it in your noncommercial application.

To help you explore such topics, we've included the complete source code for Fractint on your *Fractal Creations* companion CD-ROM. This chapter will show you how to extract it from the distribution CD-ROM disk, rebuild the executable file from the source code (assuming you have an appropriate compiler), and modify the source files to add your own features and/or fractal types. Note that we're making the assumption that you're familiar with MS-DOS commands, the "C" programming language, and your favorite "C" compiler.

In this chapter, we'll cover four topics. First, we'll describe the basic steps needed to extract the source code from the companion disk and rebuild Fractint using the popular Microsoft and Borland compilers. Next, we'll briefly describe the various source code files in an effort to help you find specific code segments. You'll find, for example, that Fractint's GIF encoder logic is in the ENCODER.C module and that its 8514/A routines are in FR8514A.ASM. Third, we'll describe the C language structure that contains the core information about all the Fractint fractal types. Finally, we'll walk you, the developer, through adding several new fractal types to Fractint's source code.

EXTRACTING THE SOURCE FILES

The source code to Fractint is stored on your companion CD-ROM disk inside a file called FRASRC.EXE. FRASRC.EXE is a self-extracting archive file, a special type of MS-DOS program that contains other programs stored in compressed form. To obtain Fractint's source code in a usable form, first set up an appropriate directory on your hard disk. This directory is referred to here as C:\FRASRC, although you can name it anything you like. Then make that new directory your current one, insert your distribution CD-ROM disk into your CD-ROM disk drive, and run the FRASRC program from the DOS prompt. If you don't have a CD-ROM drive but have access to another PC that does have one, you can copy the FRASRC.EXE file to a floppy disk and use that copy. FRASRC will extract all of the appropriate source files from its innards and deposit them onto your current directory. Here's an example of what you'd type

```
C:> MD \FRASRC(ENTER)
C:> CD \FRASRC(ENTER)
C:> D:\FRASRC(ENTER)
```

assuming that your CD drive is D.

REBUILDING FRACTINT.EXE FROM THE SOURCE CODE

You will need to have access to an appropriate C compiler in order to rebuild a Fractint EXE file from its source files. Instructions for using several such compilers are in the next few paragraphs. Other environments may work as well, but we can't vouch for them. Although several of the source modules to Fractint are written in assembler language, you don't need an assembler program unless you intend to modify those modules—current object files for all of the assembler modules have been included with the source files.

Rebuild with Microsoft C

The source files include a MAKEFRAC.BAT file that you invoke to rebuild Fractint. The MAKEFRAC.BAT file is set up to use Microsoft C version 7.0, 6.0, or 5.1 or QuickC version 2.5. We will caution you that Fractint's main authors use version 7.0 these days and, in fact, the final version of the source code was probably tested using only version 7.0, so there is always a good possibility that we have created some sort of minor problems for the other compilers at the last minute.

MAKEFRAC.BAT, as distributed, assumes that you are using Microsoft C 7.0. If you're using another version of Microsoft C, edit MAKEFRAC.BAT to select the appropriate compiler by "un-commenting" the appropriate GOTO as follows:

```
rem remove the 'rem' preceding the goto that applies to you...
rem goto msc7debug
    goto msc7
rem goto msc6
rem goto msc5
rem goto quickc
```

The build process for Microsoft C includes several other files that are invoked directly or indirectly by MAKEFRAC.BAT: several MAKEfiles (FRACTINT.MAK, FRACHELP.MAK), an indirect command file used during the link process (FRACTINT.LN7 for Microsoft C 7.0, FRACTINT.LNK for the other compilers), and FRACTINT.DEF, used only by the linker under MSC 7. The alternate link command file and the DEF file take advantage of the improved overlay capabilities of the linker supplied with MSC 7.

Rebuild with Borland C++ and Turbo C

The source files include BCFRACT.PRJ and BCHELP.PRJ files for the Borland C++ compilers, and TCFRACT.PRJ, TCHELP.PRJ, and FRACTINT.DSK files for the Turbo C compilers. When you start up your copy of Borland's C++ or Turbo C using the source directory, it will find the project files for you automatically.

Warning! Fractint's main authors use Microsoft C 7.0 these days, and in fact the final version of the source code was probably only tested against that Microsoft compiler, so there is always the possibility that we have created some sort of minor problems for the other compilers at the last minute.

THE SOURCE FILES

The Fractint source code can be logically divided into two sections. These sections, and the routines that compose them, are

- The user interface code. This code handles all of the user interaction and displays, handing off any actual fractal generating functions to the core fractal routines.

- The fractal-generating engine code. This code is concerned only with generating fractal images, and calls user-interface routines to handle any keyboard/mouse activity and video handling.

The User Interface Code

The user interface routines consist of the following modules.

FRACTINT.C

This is where you'll find the main() routine. FRACTINT.C can be thought of as a traffic cop for the rest of the program. On startup, this module first calls the routines that detect and initialize the environment under which Fractint is running. It then displays the initial scrolling credits screen and waits for a keypress. Once you've selected an initial video mode, the module drops into its main message loop, detecting keystrokes and mouse movements via calls to the keypressed() routine and invoking appropriate user interface or fractal engine routines to process them. Listing 8-1, for instance, contains the code in the main message loop that invokes Fractint's image-flipping logic when you press CONTROL-X, CONTROL-Y, or CONTROL-Z while a fractal image is on the screen.

Listing 8-1 The image-flipping logic in FRACTINT.C

```
case 24:               /* Ctl-X, Ctl-Y, CTL-Z do flipping */
case 25:
case 26:
   /* Note: 'kbdchar' contains the keyboard character that
      'keypressed()' has reported is in the input queue.
      'flip_image()' (in module MISCOVL.C) is the routine
      that actually performs the image-flipping and removes
      the keypress from the input queue.
   */
   flip_image(kbdchar);
   break;
```

PROTOTYP.H

This is a file of *function prototypes*. Any routine that is defined in one module and called from another has a function prototype defined here. The Fractint authors have found that using a standard include file like this significantly reduces the problems one runs into when routine "a" hands an integer value to routine "b"—which expects to see a floating-point value.

PORT.H

This is an include file of portability equates, which serves to hide some of the differences between different operating environments from the rest of Fractint's source code. Fractint's original MS-DOS implementation continues to be its most popular, but there are currently Windows 3.x, OS/2 (1.x and 2.x), and X-Windows ports of Fractint in existence using many of the same source modules. It is in here, for instance, that PRINTER is defined to be "PRT:," "/dev/prn," or "/dev/lp" based on the type of computer system in use.

CMDFILES.C

The routines in this module handle all of Fractint's startup parameters, as specified either on the startup command-line, the fractint section of your SSTOOLS.INI file, or any parameter files being processed. If you wanted to modify Fractint to perform backflips, for example, you could make Fractint recognize a new `backflip=[yes|no]` command-line option by adding it to the list of `if(strncmp(...` clauses in the cmdarg() routine in this module (see Listing 8-2).

Listing 8-2. Adding "backflip=[yes|no]" command-line sensitivity to Fractint

```
/* the following code, when added to the 'cmdarg()' routine in
   CMDFILES.C, acts on the presence of a "backflip=[yes|no]"
   command-line option. 'cmdarg()' has already set up the following:
        variable    is a string containing the argument name
        yesnoval    contains   1  if the argument value was "yes"
                               0  if the argument value was "no"
                              -1  if it was something else
*/

if (strcmp(variable,"backflip") == 0 ) {
    if (yesnoval < 0) goto badarg;
    if (yesnoval == 1) {
        /* The logic for actually getting Fractint to perform backflips
           would go here, and is left as an exercise for the reader   */
        }
    /* a return value of 0 indicates that all is well. */
    return 0;
    }
```

PROMPTS1.C, PROMPTS2.C

The routines in these modules handle Fractint's full-screen prompting logic, which handles all of Fractint's data entry functions. These routines were originally in a single module, and were split up only because the original module grew too large for some compilers to handle. The fullscreenprompt() routine, which accepts lists of prompt strings, field types, and default entry values, presents the user with a full-screen display, and returns with his selections, is in PROMPTS1.C.

HELP.C, HELPDEFS.H

This module and its include file handles Fractint's on-line help engine. They are invoked whenever you press F1 for help. The help file itself is built using a stand-alone help compiler developed especially for Fractint. Fractint can access this help file either as a separate file (FRACTINT.HLP), or as text attached to the end of FRACTINT.EXE. The latter option reduces the number of files required by Fractint, but isn't compatible with all compilers and isn't always portable across operating systems.

HC.C, HELPCOM.H, HELP*.SRC

HC.C is the source code for a stand-alone program that builds Fractint's FRACTINT.HLP and HELPDEFS.H files from the various HELP*.SRC source files. It is compiled separately from the rest of Fractint (but handled automatically by the MAKEFRAC.BAT file used by the Microsoft compilers).

INTRO.C

The routines in this module handle Fractint's introductory scrolling-credits screen. This is a very small module, but it is kept separate from the rest of the Fractint code so that it can be easily used as an overlay module.

SLIDESHW.C

The routines in this module handle Fractint's demo mode `autokey=` (slideshow) logic. This lets you build files of "keystrokes" controlling Fractint in stand-alone demos. Run the DEMO.BAT file included on your distribution disk to see how this works.

PLOT3D.C, 3D.C, LINE3D.C

The routines in these modules handle Fractint's 3-D capabilities as invoked by the 3-D and 3-D Overlay functions. The draw_line( ) routine in PLOT3D.C, for instance, is a speedy little routine that draws a straight line from point A to point B using the Bresenham algorithm.

ROTATE.C

The routines in this module handle Fractint's color-cycling logic. The low-level routines that handle the actual color cycling are assembler-based and in the VIDEO.ASM module.

EDITPAL.C

The routines in this module handle Fractint's palette-editor logic. The low-level routines that handle the actual palette manipulation are assembler-based and in the VIDEO.ASM module.

PRINTER.C, PRINTERA.ASM

The routines in these modules handle all of the printer routines (the ones that get control when you press the Ⓟ key to print an image). The assembler module exists only so that some of the larger printer tables can be stuffed into an overlaid code segment, keeping Fractint's total memory requirements as low as possible.

GENERAL.ASM

The routines in this module handle Fractint's "general purpose" assembler modules. There is a lot of excellent code buried in here, most of which has nothing to do with fractals. GENERAL.ASM holds Fractint's CPU/FPU detectors, extended/expanded memory access routines, keyboard/mouse routines, sound routines, and the 32-bit scaled integer multiply() and divide() routines used as the core of Fractint's integer math modules. Comments at the beginning of GENERAL.ASM give a complete list of its routines by name and function.

VIDEO.ASM

The routines in this module handle the bulk of Fractint's low-level video access (a few specialty routines, described later, handle some of the more exotic video adapters). As with GENERAL.ASM, there is a lot of excellent code buried in here, none of which has anything to do with fractals. For example, the adapter_detect() routine, called when Fractint first starts up, automatically detects the presence of a CGA, EGA, VGA, XGA, and almost every super VGA adapter known to humankind. Comments at the beginning of GENERAL.ASM give a complete list of the routines that the rest of Fractint code knows about by name and function.

FR8514A.ASM, HGCFRA.ASM, TARGA.C, TARGA.H, TARGA_LC.H, TPLUS.C, TPLUS_A.ASM, TPLUS.DAT, TP3D.C, LOADMAP.C

The routines in these modules handle Fractint's 8514/A and 8514/A clones (FR8514A), Hercules (HGCFRA), and Targa (all of the others) video routines.

DISKVID.C

The routines in this module handle Fractint's Disk/RAM Video routines, which let you use your real/extended/expanded memory or even your disk drive as a virtual video adapter. To the rest of the Fractint code, this "virtual video" is no different from any other "real" video mode. With these routines, even the user with a lowly CGA video adapter can generate 256-color images at resolutions up to 2048 x 2048 pixels.

YOURVID.C

The routines in this module give the programmer a simple way to add support for video adapters not currently supported by Fractint. All the programmer has to do is modify the existing routines that set up the video mode, end it, and read and write pixels to the screen while in that video mode. The module as shipped on the companion CD invokes IBM's MCGA/VGA 320 x 200 x 256 mode.

ZOOM.C

The routines in this module, working with the low-level routines in VIDEO.ASM, support Fractint's zoom-box functions.

ENCODER.C, DECODER.C, LOADFILE.C, GIFVIEW.C, TGAVIEW.C, F16.C

The routines in these modules handle the GIF and TARGA file loading and saving capabilities of Fractint. The main GIF encoding and decoding routines are located in ENCODER.C and DECODER.C.

REALDOS.C, LOADFDOS.C

The routines in these modules handle DOS-specific functions (mostly video-related) that are handled differently in other environments, such as Windows and OS/2. The stopmsg() routine in REALDOS.C, for example, switches your video to text mode, displays the message given it, waits for a keypress, and reports the results to whatever routine called it. The Windows port of Fractint uses an alternate module (WINDOS.C, which is not included as part of the Fractint source code) and the Windows messagebox() routine to perform the same function.

MISCRES.C, MISCOVL.C

The routines in these modules are the routines that literally didn't fit anywhere else. MISCOVL.C includes the routines that can be stuffed into an overlay section, and MISCRES.C includes those that (at least under versions of Microsoft C prior to version 7.0) can't.

The Fractal-Generating Code

The core fractal-generating routines concern themselves only with generating fractal images, and hand off all user input and screen display functions to the user interface routines. Although the original reason for this splitting of functionality was simply ease-of-programming, it has resulted in a very handy dividend: these fractal-generating routines are now generally common to both Fractint and its various ports to other environments (WINFRACT for Microsoft Windows, PMFRACT for OS/2, XFRACT for the X-Windows environment, etc.). When a new version of Fractint is released, its new fractal-generating code can be quickly added to its sister programs—sometimes in a matter of hours. These fractal-generating routines consist of the following modules.

FRACTINT.H

This is an include file of common definitions that virtually every other routine must know about.

FRACTALP.C, FRACTYPE.H

FRACTALP.C contains a single fractalspecific structure containing information about each of the fractal types included in Fractint—in fact, the only items that most of the other modules know about the fractal types in Fractint are contained in this structure. Because of its importance to the rest of the program, this 'fractalspecific' structure is described in more detail in the next section. FRACTYPE.H contains #defines for the entry locations of each fractal type in the structure.

CALCFRAC.C

CALCFRAC.C contains the code for the basic fractal engine and its general algorithms. Calcfract() is the fractal engine entry point called by the main fractal loop in FRACTINT.C, and oversees the remainder of the fractal engine code. Calcfract() scans the fractalspecific structure (described in the next section) for any fractal-specific information, initializes its pointers accordingly, and invokes whichever fractal routine that structure states should generate that image.

CALCFRAC.C also contains the routines handling many of the general fractal-related algorithms. StandardFractal(), for example, is the general escape-time fractal image generator and handles most of the options specific to escape-time fractals, such as the inside and outside options. The solidguess(), boundary_trace(), and tesseral() routines are also found here.

FRACSUBR.C

FRACSUBR.C contains service routines invoked by the startup fractal code in FRACTINT.C and the core fractal code in CALCFRAC.C. It's actually an overflow module that was created when CALCFRAC.C simply grew too large for many of the popular MS-DOS-based compilers to handle.

The most important routine by far in this module is calcfracinit() which is called by FRACTINT.C prior to starting any fractal image. Calcfracinit() determines whether or not an image has been zoomed too deep for its integer algorithm (and if so, switches to its floating-point equivalent algorithm), determines the actual image corners, fills up coordinate arrays determining the location of every pixel, and initializes a few coordinate-related variables.

FRACTALS.C

FRACTALS.C contains most of the fractalspecific code for the escape-time fractals (escape-time fractals are those fractal types which are calculated by iterating formulas until an intermediate value exceeds a predetermined value). The fractal-specific functions for the Lambda fractal, for example (a standard escape-time fractal using the formula $z(n+1) = Lambda \times z(n) \times (1 - z(n)$^2) are located in FRACTALS.C.

FRACSUBA.ASM

FRACSUBA.ASM contains several general service routines that have been moved into assembler for speed. The longbailout() routine, for example, quickly checks to see if the long integer values X and Y meet the condition $(((X \times X) + (Y \times Y)<Z)$ while worrying about possible integer overflow conditions. Every routine in this module was at one time a small C routine in FRACTALS.C (and for portability purposes still has a C-based equivalent routine there).

CALCMAND.ASM, CALMANFP.ASM

These are the Mandelbrot/Julia Set routines, hand-tuned in assembler for the last ounce of speed. When folks compare the speed of various fractal programs, they're almost invariably discussing how fast those programs generate a Mandelbrot fractal. There are far more readable (but slower) C versions of these same fractal types in the core fractal engine. Those C routines are used when an option (such as biomorphs) is in effect that the superfast assembler routines don't check for.

NEWTON.ASM, LORENZ.C, JB.C, PARSER.C, PARSERA.ASM, TESTPT.C, LSYS.C, LSYSA.ASM, LYAPUNOV.ASM, HCMPLX.C, JIIM.C

These routines handle the Newton, Lorenz, JuliBrot, Formula, Test, L-systems, Lyapunov, HyperComplex, and real-time Julia set fractal types respectively. These fractal types either didn't fit into the core fractal engine structure or, as with the previous Mandelbrot/Julia routines, were hand-coded for speed.

MISCFRAC.C

MISCFRAC.C is a catchall routine that contains several miscellaneous routines that handle various fractal types. Stand-alone fractal types with relatively short generating routines tend to end up here. The plasma() routine handles plasma cloud images. The diffusion() routine handles diffusion fractals, while the bifurcation() routine handles the various bifurcation fractals. The popcorn(), cellular(), and frothybasin routines are in here as well.

MPMATH_C.C, MPMATH_A.ASM, MPMATH.H

These MP (Mark Peterson's) math routines handle floating-point mathematics using integer exponent/mantissa pairs. They are an alternate to Fractint's scaled long-integer math routines for those situations where scaled-integer math doesn't work very well.

FPU087.ASM, FPU387.ASM

These are customized floating-point math operations, written in assembler for speed.

THE FRACTALSPECIFIC STRUCTURE

The fractalspecific structure in FRACTALP.C contains all of the information most of the modules in Fractint need to know about fractals. Even the main fractal engine code uses that structure to access any routines that are specific to any fractal type.

The fractalspecific structure and the definitions of the flag bits included in it are located in the FRACTINT.H include file. Fractal types are added to Fractint simply by adding any new routines required to generate the fractal types to the basic fractal engine and then adding new entries to this structure describing the fractal types and pointing to the routines that generate them. In the next section, we'll go through this process, adding several new fractal types to Fractint.

Listing 8-3 shows the layout of the fractalspecific structure. Listing 8-4 shows the fractalspecific entries and related variables for the Mandelbrot/Julia family of fractals. Listings 8-5 and 8-6 appear later in this section with detailed descriptions of their related flag values.

Listing 8-3 Fractint's fractalspecific structure

```
struct fractalspecificstuff
{
   char  *name;               /* name of the fractal */
   char  *param[4];           /* name of the parameters */
   float paramvalue[4];       /* default parameter values */
   int   helptext;            /* helpdefs.h HT_xxxx, -1 for none */
   int   helpformula;         /* helpdefs.h HF_xxxx, -1 for none */
   int   flags;               /* constraints */
   float xmin;                /* default XMIN corner */
   float xmax;                /* default XMAX corner */
   float ymin;                /* default YMIN corner */
   float ymax;                /* default YMAX corner */
   int   isinteger;           /* 1 if integerfractal, 0 otherwise */
   int   tojulia;             /* mandel-to-julia switch */
   int   tomandel;            /* julia-to-mandel switch */
   int   tofloat;             /* integer-to-floating switch */
   int   symmetry;            /* applicable symmetry logic */
   int (*orbitcalc)();        /* function that calculates one orbit */
   int (*per_pixel)();        /* once-per-pixel init */
   int (*per_image)();        /* once-per-image setup */
   int (*calctype)();         /* name of main fractal function */
   int orbit_bailout;         /* usual bailout value for orbit calc */
} ;
```

Listing 8-4 The Mandelbrot/Julia fractalspecific entries

```
#define NOFRACTAL                -1
#define MANDEL                    0
#define JULIA                     1
#define MANDELFP                  4
#define JULIAFP                   6

static char realz0[] = "Real Perturbation of Z(0)";
static char imagz0[] = "Imaginary Perturbation of Z(0)";
static char realparm[] = "Real Part of Parameter";
static char imagparm[] = "Imaginary Part of Parameter";

struct fractalspecificstuff far fractalspecific[] ={
 /*  fractal name, parameter text strings, parameter values,
     helptext, helpformula, flags,
     xmin  xmax  ymin  ymax int tojulia   tomandel tofloat  symmetry
       orbit fnct     per_pixel fnct  per_image fnct  calctype fcnt    bailout    */

   "mandel",      realz0, imagz0,"","",0,0,0,0,
   HT_MANDEL, HF_MANDEL, WINFRAC,
   -2.5,  1.5, -1.5,  1.5, 1, JULIA,     NOFRACTAL, MANDELFP, XAXIS_NOPARM,
   JuliaFractal,  mandel_per_pixel,MandelSetup,StandardFractal, STDBAILOUT,
```

continued on next page

continued from previous page

```
   "julia",       realparm, imagparm,"","",0.3,0.6,0,0,
   HT_JULIA, HF_JULIA, WINFRAC,
   -2.0,  2.0, -1.5,  1.5, 1, NOFRACTAL, MANDEL, JULIAFP,  ORIGIN,
   JuliaFractal,   julia_per_pixel, JuliaSetup,StandardFractal, STDBAILOUT,
...
   "*mandel",    realz0, imagz0,"","",0,0,0,0,
   HT_MANDEL, HF_MANDEL, WINFRAC,
   -2.5,  1.5, -1.5,  1.5, 0, JULIAFP,   NOFRACTAL, MANDEL,  XAXIS_NOPARM,
   JuliafpFractal,mandelfp_per_pixel, MandelfpSetup,StandardFractal, STDBAILOUT,
...
   "*julia",     realparm, imagparm,"","",0.3,0.6,0,0,
   HT_JULIA, HF_JULIA, WINFRAC,
   -2.0,  2.0, -1.5,  1.5, 0, NOFRACTAL, MANDELFP, JULIA,  ORIGIN,
   JuliafpFractal, juliafp_per_pixel,  JuliafpSetup,StandardFractal,STDBAILOUT
```

The Fractalspecific Structure—User Interface Entries

The Fractint modules that handle the user interface are interested in the following fractalspecific items (the items are not necessarily listed in the order in which they appear in the structure).

name

This is the name of the fractal type. Fractal names can be up to 16 characters long, and cannot include spaces. If a particular fractal type has both integer and floating-point algorithms, it has two structure entries. In such cases, one of the entries has a leading "*" in the name field (note that there's a "mandel'"and a "*mandel" entry in Listing 8-4). The display routines use this "*" as a flag indicating that this entry is a duplicate that should not be displayed for user-selection purposes. In these cases, each entry contains flags indicating which type it is (isinteger) and pointing towards its alternative (tofloat), so Fractint can select the appropriate entry automatically.

param[4], paramvalue[4]

These array entries contain the string descriptions and default values of up to four optional user-selectable parameters. Fractal types with fewer than four such parameters have null entries in the name fields that aren't relevant. Parameter names should be kept under 40 characters long just so they'll fit in Fractint's dialog boxes. All members of the Mandelbrot/Julia family of fractals accept two parameters.

xmin, xmax, ymin, ymax

These are the corner values of the default image that is displayed when this fractal type is first selected.

flags

This field contains a bitmask of flags that the Fractint modules query to determine the capabilities and limitations of this fractal type. These flags and their values are detailed in Listing 8-5. Although many of the entries in this field are of interest only to the routines that actually generate the fractal, some of them are of interest to the other modules as well. In particular, the presence of the WINFRAC flag indicates that Fractint's Windows port, Winfract, has the capability to generate this fractal type. Usually, every fractal type in Fractint can also be generated using Winfract and sports this flag. All members of the Mandelbrot/Julia family of fractals use only the WINFRAC flag.

Listing 8-5 Fractint's fractal-specific bitmasked flags

```
/* bitmask defines for fractalspecific flags */
#define  NOZOOM          1    /* zoombox not allowed at all    */
#define  NOGUESS         2    /* solid guessing not allowed    */
#define  NOTRACE         4    /* boundary tracing not allowed   */
#define  NOROTATE        8    /* zoombox rotate/stretch not allowed */
#define  NORESUME       16    /* can't interrupt and resume    */
#define  INFCALC        32    /* this type calculates forever   */
#define  TRIG1          64    /* 1 trig function  in formula   */
#define  TRIG2         128    /* 2 trig functions in formula   */
#define  TRIG3         192    /* 3 trig functions in formula   */
#define  TRIG4         256    /* 4 trig functions in formula   */
#define  WINFRAC       512    /* supported in WinFrac */
#define  PARMS3D      1024    /* uses 3d parameters */
```

helptext, helpformula

These entries contain pointers (defined in HELPDEFS.H) to locations in Fractint's help file describing this particular fractal type. A -1 indicates that no Fractint-style HELP reference is available. When you add fractal types to Fractint and haven't expanded Fractint's internal help files to accomodate them, use a value of −1 for these fields.

tojulia, tomandel

These fields contain an entry number giving the location in the fractalspecific array of the Julia or Mandelbrot fractals matching this fractal type, if any such fractal types exist. The main message loop in FRACTINT.C uses these flags when the user requests a Mandelbrot/Julia switch. These entry numbers are coded as #defines in FRACTYPE.H. Note in Listing 8-4 how the Mandelbrot and Julia set entries point to each other.

The Fractalspecific Structure—Fractal-Generation Entries

The Fractint modules that actually generate the fractal image are interested in the following fractalspecific items (the items are not necessarily listed in the order in which they appear in the structure).

isinteger

This field is nonzero if this fractal type uses an integer algorithm, and zero if this fractal type uses floating-point math. Many of the fractal types, including the Mandelbrot/Julia fractals in Listing 8-4, have both an integer and a floating-point version, with an entry for each.

tofloat

This field contains the entry number of the fractal type which is the integer or floating-point equivalent to this one (the name is misleading, as this may be a floating-point fractal type pointing to its integer equivalent). A zero in this entry indicates that there is no such equivalent. Note in Listing 8-4 how the Mandelbrot and Julia set entries point to their sister entries. The calcfracinit() routine in FRACSUBR.C uses the *tofloat* and *isinteger* flags to automatically (and silently) select the fractal type appropriate to the users preference and the current zooming depth (floating-point fractal types can zoom in farther than their integer equivalents). When using fractal types that have only one entry (plasma clouds, for example, have no floating-point equivalent). Fractint uses that one algorithm regardless of any preference the user has indicated.

symmetry

Many fractal types are symmetrical in some respect, and Fractint's fractal engine can recognize and handle many different kinds of symmetry. The Mandelbrot fractal, for example, is symmetrical along the *x*-axis as long as none of its optional parameters has been set to a nonzero value. The Julia fractal is symmetrical about the origin. Whenever it can, Fractint takes advantage of symmetry to reduce the amount of time it takes to calculate and display an image—note how the bottom half of the initial Mandelbrot image is generated and displayed at the same time as the top half.

If a particular fractal type has symmetrical properties, that symmetry is indicated using this flag. Listing 8-6 includes an annotated listing of all the symmetry types currently recognized by Fractint.

Listing 8-6 Fractint's fractalspecific symmetry flags

```
/* defines for symmetry */
#define  NOSYM     0          /* no symmetry at all */
#define  XAXIS_NOPARM  -1     /* X-axis symmetry if no parameters are given */
#define  XAXIS     1          /* X-axis symmetry */
#define  YAXIS_NOPARM  -2     /* Y-axis symmetry if no parameters are given */
#define  YAXIS     2          /* Y-axis symmetry */
#define  XYAXIS_NOPARM -3     /* XY-axis symmetry if no parameters are given */
#define  XYAXIS    3          /* XY-axis symmetry */
#define  ORIGIN_NOPARM -4     /* Origin symmetry if no parameters are given */
#define  ORIGIN    4          /* Origin symmetry */
#define  PI_SYM_NOPARM -5     /* PI symmetry if no parameters are given */
#define  PI_SYM    5          /* PI symmetry */
#define  XAXIS_NOIMAG  -6     /* X-axis symmetry if param has no imag component */
#define  XAXIS_NOREAL   6     /* X-axis symmetry if param has no real component *
#define  SETUP_SYM  100       /* for formula  fractals - symmetry declared in the file */
```

flags

This field of bitmasked flags, already mentioned as a user interface field, also contains a number of flags of interest to the code that actually generates fractals. Some fractal types, for example, don't work correctly with the solid-guessing and/or boundary-tracing options, so the fractal engine must ignore the user's preferences for those options when generating those fractal types. Listing 8-5 includes an annotated listing of the bitmasked flags contained in this field.

orbit_bailout

For escape-time fractal types, this is the comparison value that is used by default to detect that an iteration has escaped from the fractal set. For the Mandelbrot fractals, this value is 4, as the calculation of the fractal ends when the value of $|z(n)|^2$ is greater than 4. For non-escape-time fractals, this field is meaningless, and is usually set to NOBAILOUT (#defined to be 0.0) as an indicator of that fact.

The four following entries are pointers to functions, and are used by the fractal engine to connect the fractal type to any fractalspecific code it requires. None of these routines is called with any parameters (they all use global variables instead for speed). All of the functions return values, although in some cases those values are ignored.

per_image()

This is a pointer to a function specific to this fractal type that handles any initialization that must be taken care of on a "per-image" basis. The per_image() function returns a value of 0 if it has completed the fractal image on its own, or returns a nonzero value if the main fractal engine should continue by calling the calctype() function (described in the next section).

Fractal types such as the plasma, IFS, and L-system fractals (which use completely unique calculation logic) traditionally point to the standard StandaloneSetup() routine here and then point to their custom routine in the calctype() pointer, described next. StandaloneSetup() calls a routine which starts a timer and then calls the calctype() routine for this fractal type. StandaloneSetup() always returns a 0 to indicate that the calculation process is completed.

The more common escape-time fractal types usually point to one of several standard initialization routines. These routines initialize a few variables and handle special cases that affect items like symmetry. The Mandelbrot and Julia fractals, for instance, determine at this point whether they can use their fast assembler algorithms or must use slower C-based algorithms because some option (such as biomorphs or decomposition) is in effect, which the fast assembler algorithms cannot handle. The four most commonly used initialization routines for escape-time fractals are MandellongSetup(), MandelfpSetup(), JulialongSetup(), and JuliafpSetup(), which handle generic Mandelbrot- and Julia-style fractals for the integer and floating-point algorithms, respectively.

Listing 8-15, shown later in this chapter in the section on adding escape-time fractals, lists the MandellongSetup() routine used by many of the integer Mandelbrot-like escape-time fractals (and Listing 8-16 shows what this routine would look like without all of its special-case logic).

calctype()

This is a pointer to a function that handles the overall fractal calculation. For some fractal types (plasma clouds, IFS, and L-system fractals, for example), this function is specific to the fractal type and handles the entire image with a single call.

For all of the escape-time fractals this item points to the StandardFractal() routine. The core fractal logic in CALCFRAC.C recognizes the StandardFractal() entry as being the escape-time fractal routine. When it sees that StandardFractal() is specified as the calctype() entry, the core fractal logic switches to its escape-time logic, enables its escape-time options such as solid-guessing and boundary-tracing, and calls StandardFractal() once for each pixel. The StandardFractal() routine handles the per-pixel options like biomorphs, decomposition, inside and outside colors, and calls the per_pixel() and orbitcalc() functions described in the next two sections to do the actual work of performing any fractalspecific calculations.

The calctype() routine returns a zero to indicate that it completed the image, or a nonzero value to indicate that it didn't (maybe you pressed the ESC key to abort the calculation). This returned value is used by the core fractal engine logic to tag the image as being complete or incomplete. StandardFractal() is a special case, and returns the color of the pixel it has just calculated to the escape-time fractal routines that called it (information that the solid-guessing, boundary-tracing, and tesseral algorithms need).

Of all the routines specified in the fractalspecific array, the routine pointed to by calctype() is by far the most complex. For non-escape-time fractals, it's often the only routine specific to the fractal type. If you are adding an escape-time fractal algorithm to Fractint, you must use StandardFractal(). On the other hand, if you're adding a fractal type that is completely unlike anything Fractint currently features, the combination of the calctype() entry point and the per_image() one gives you the ability to do so.

Listings 8-8 through 8-11, shown later in this chapter in the section on adding stand-alone fractals, contain a complete stand-alone calctype() function implementing a drunkard's walk fractal type.

per_pixel()

This is a pointer to a function called by StandardFractal() that handles any fractal-specific initialization that must be taken care of on a per-pixel basis (usually precalculating $z(0)$). Many escape-time fractal types share common per_pixel() routines, although writing your own is easily done. Listing 8-18, shown later in this chapter in the section on adding escape-time fractals, lists simple per_pixel() routines for integer and floating-point Mandelbrot and Julia fractals. StandardFractal() ignores any value returned by this routine.

Non-escape-time fractal types are free to use this function pointer for their own purposes. Usually, their fractalspecific entries for this function pointer are just set to NULL.

orbitcalc()

This is a pointer to a function called by StandardFractal() that handles a single orbit (iteration) calculation. This function calculates $z(n+1)$ given the value of $z(n)$, and is often the only routine that is specific to a particular family of escape-time fractal types. This routine returns 0 if the new value has not reached its bailout limit, or 1 if it has. Listing 8-19, shown later in this chapter in the section on adding escape-time fractals, lists the orbitcalc() routines used for integer and floating-point Mandelbrot and Julia fractals.

Non-escape-time fractal types are free to use this function pointer for their own purposes. Usually, their fractalspecific entries for this function pointer are just set to NULL.

ADDING NEW FRACTAL TYPES

Let's put all the claims about adding new fractals to Fractint to a test by actually doing it. We're going to add two different kinds of fractals to Fractint.

First, we'll add a new stand-alone fractal type to Fractint—a simple *drunkard's walk.* (A drunkard's walk involves an object taking random-length steps in random directions.) For demonstration purposes, we'll also give this fractal type the ability to generate Mandelbrot images, even though you'd normally generate Mandelbrot fractals using the escape-time fractal engine.

Then, we'll add a family of four escape-time fractal types—integer and floating-point versions of two fractal types that form a Mandelbrot/Julia pair. In fact, in this case our "new" fractals are going to be the familiar Mandelbrot and Julia sets.

By the time we have added these fractals, we'll have covered all the basic concepts of adding fractal types to Fractint.

Let's admit to a little cheating here—the entries you are going to add are already in their relevant modules. They've just been commented out so as not to show up in the distributed Fractint executable. When the text refers to your adding these entries to the relevant modules, all you *really* have to do is remove the start (/*) and end (*/) of the comments. Nothing eliminates typing errors quite as efficiently as eliminating the typing itself.

Adding Stand-Alone Fractals

Now let's add our stand-alone drunkard's walk fractal (with its Mandelbrot option) to Fractint.

Adding Entries to the FRACTYPE.H File

The first thing you must do when adding new fractal types to Fractint is to add their definitions to the FRACTYPE.H file. These definitions point to the fractal's locations in the fractalspecific structure (recall that the rest of Fractint's code uses that fractalspecific structure to access any information it needs about a particular fractal type). For many fractal types, these definitions are only used in the fractalspecific structure and, in the case of our drunkard's walk fractal, this definition is never actually going to be used anywhere at all. Nevertheless, it's a good practice to always add the definition to FRACTYPE.H so that all of the fractal types are defined in one place. For one thing, it's quite confusing when you're adding fractal types if the last entry in this list isn't also the last entry in the fractalspecific array. The value of this entry will be the location of the entry you will be adding to the end of the fractal-specific structure. You'll need a single entry for the drunkard's walk fractal, indicating that it is located in the 161st entry in the fractal-specific structure:

```
#define DEMOWALK            161
```

Adding Entries to the Fractalspecific Structure

Having added the entry for this new fractal type to our list in FRACTYPE.H, add its entry to the end of the fractalspecific array in FRACTALP.C, just before the NULL entry indicating the end of the list (in this case, just before the four new escape-time fractal entries that are also commented out just prior to the NULL entry—we'll discuss them in the next section). Listing 8-7 shows the new entry you need to add.

Listing 8-7 The fractalspecific entry for drunkard's walk

```
"demowalk", "Average Stepsize (% of image)",
"Color (0 means rotate colors)","","",5,0.0,0,0,-1, -1,NORESUME+WINFRAC,
-2.5, 1.5,  -1.5,   1.5, 0, NOFRACTAL, NOFRACTAL, NOFRACTAL, NOSYM,
NULL,  NULL,  StandaloneSetup,  demowalk,  NOBAILOUT,
```

Refer to Listings 8-3 through 8-6 as we examine this fractalspecific entry, as they include the definitions of all of the flags it uses. The fractal type's name, used whenever Fractint needs to identify this fractal type for the user, is "demowalk." This fractal type accepts two optional parameters: an average stepsize expressed as a percentage of the image size, and the color used to display the walk. The first parameter (the stepsize) defaults to 5%, and the second (the color) defaults to zero (successive lines generated as the fractal iterates will be generated using different colors as it walks).

Helptext and helpformula are –1, as we have not developed any help text for this fractal type. This fractal uses two fractal-specific flags. NORESUME indicates that this fractal type is not resumable. If we save one of its images in the middle of a calculation and reload it later, this fractal type doesn't contain any special logic letting it resume its calculations where they left off (the escape-time fractal types rely on logic inside the standard escape-time fractal engine to perform this task). The WINFRAC flag signals that Fractint's Windows port, Winfract, can handle this fractal type.

The initial screen corners are –2.5,1.5,–1.5,1.5—the standard, slightly off-center corner values that display the initial Mandelbrot image well. Isinteger is zero, indicating that this fractal type is a floating-point fractal and the core fractal engine should set up floating-point pixel coordinates when calling it. This fractal has no Julia equivalent, Mandelbrot equivalent, or integer/floating point equivalent, so those entries are all NOFRACTAL. This fractal has no symmetry (NOSYM).

The demowalk fractal follows the convention of the stand-alone fractal types: the per_image() function points to StandaloneSetup(), the calctype() function points to demowalk's own fractal-specific routine, and the per_pixel() and orbitcalc() functions are unused and set to NULL as an indicator of that fact. Its bailout value has been set to NOBAILOUT because the user can't change the bailout value of its simple Mandelbrot function.

Once you've added (or removed the comments from) these entries in FRACTYPE.H and FRACTALP.C, rebuild Fractint and run it again. You now have a new fractal type, demowalk.

The calctype() Routine

As is the convention with the stand-alone fractal types, the calctype() function does all the work for generating this particular fractal type. Listings 8-8 through 8-11 contain the complete code for the demowalk() function, which you will find that we've already added to FRACTALS.C.

Let's walk through this routine line-by-line to see what functions a stand-alone fractal type must perform. Stand-alone routines, because they have no general fractal engine overseeing their function as the escape-time fractals do, have to handle more functions on their own (like updating the actual image and worrying about whether the user is frantically banging on the keyboard attempting to bring up a spreadsheet before the boss arrives).

Defining and Initializing the Variables

Listing 8-8 shows the portion of the demowalk() routine that defines and initializes the variables we need.

The first seven lines reference global variables that demowalk() needs to access to perform its function. Param[] is an array holding up to four optional parameter values, either the default values from the fractalspecific entry or their revised values as modified by the user. Maxit is the maximum number of iterations. Rflag and rseed are values for the random number generator—using these global values instead of its own lets demowalk() take advantage of the `rseed=` command-line parameter in case anyone ever needs to generate replicatable random walks (hey, you never know). Xdots and ydots give the resolution of the current video mode, and colors gives the number of colors in this mode. Dx0[], dy0[], dx1[] and dy1[] are arrays used to determine the floating-point values of each pixel coordinate on the image. The drunkard's walk section of demowalk() doesn't need them, but its Mandelbrot option does (and your fractal types probably will).

The next six lines describe local variables that this fractal type needs for its calculations.

Listing 8-8 The demowalk() fractal generation routine—initialization code

```
demowalk()
{
    extern double param[];                  /* optional user parameters */
    extern int maxit;                       /* maximum iterations (steps) */
    extern int rflag, rseed;                /* random number seed */
    extern int xdots, ydots;                /* image coordinates */
    extern int colors;                      /* maximum colors available */
    extern double far *dx0, far *dy0;       /* arrays of pixel coordinates */
    extern double far *dx1, far *dy1;       /* (... for skewed zoom-boxes) */

    float stepsize;                         /* average stepsize */
    int xwalk, ywalk;                       /* current position */
    int xstep, ystep;                       /* current step */
    int steps;                              /* number of steps */
    int color;                              /* color to draw this step */
    float temp, tempadjust;                 /* temporary variables */
```

Accessing the Parameters

Listing 8-9 shows the portion of the demowalk() routine that examines the user-settable parameters and sets up our walk accordingly.

The first line of executable code just lets us branch around the drunkard's walk routine if the user has set the first optional parameter to 999—our way of getting to the Mandelbrot option. Let's ignore the Mandelbrot option for the moment (we'll get to it later) and concentrate on the drunkard's walk algorithm.

Next, the executable code seeds the random number generator (and bumps up the seed value so the next random walk will be different from this random walk). Then the routine sets up its initial xwalk and ywalk values to start its walk in the center of the image.

The next few lines of code reference the user-modifiable parameters, param[0] and param[1]. Param[0] is defined in the demowalk fractalspecific structure as the average stepsize in terms of a percentage of the image. Param[1] is defined as the color to use for the walk.

Listing 8-9 The demowalk() fractal generation routine—parameter access code

```
if (param[0] != 999) {                      /* if 999, do a Mandelbrot instead */

    srand(rseed);                           /* seed the random number generator */
    if (!rflag) ++rseed;
    tempadjust = RAND_MAX >> 2;             /* adjustment factor */

    xwalk = xdots / 2;                      /* start in the center of the image */
    ywalk = ydots / 2;

    stepsize = min(xdots, ydots)            /* calculate average stepsize */
             * (param[0]/100.0);            /* as a percentage of the image */

    color = max(0, min(colors, param[1]));  /* set the initial color */
```

Let's Walk!

Listing 8-10 shows the portion of the demowalk( ) routine that actually performs the drunkard's walk.

First, note the call to keypressed( ) at the beginning of this loop, with a return with a zero value if keypressed() returns a value. This logic performs a very important function: It gives Fractint the ability to force demowalk( ) to exit early. Maybe the user fired up the demowalk fractal using a maxit of 20,000 and has decided about 10,000 iterations into the image that he wants to see another fractal type instead. Maybe his boss is walking in the door and our inspired user is *supposed* to be working on next year's budget.

The next eight lines perform a fairly standard drunkard's walk routine, randomly choosing *x* and *y* directions and handling the case where the poor drunkard smashes into the walls at the edges of the image (in our case, sticking to the wall until the end of that walk).

The next three lines handle the optional color rotation in the case where param[1] is zero, being careful to avoid using the background color 0 or exceeding the number of colors in the image.

Finally, demowalk( ) has to update the image. Rather than build our own line drawing routine, we borrowed one from the 3-D logic in PLOT3D.C. Why invent code when you can steal ...uhh... borrow it?

Note that there is alternative, commented out code that uses the putcolor() routine to display just the endpoints instead of the entire line. This code was added just to demonstrate how to update images a pixel at a time.

The last two lines in the loop replace our drunkard's old location with his new one in preparation for taking the next step.

Finally, if the routine falls out of the bottom of the generation loop it returns with a value of one indicating to the calling routines that it completed its image normally.

Listing 8-10 The demowalk( ) fractal generation routine—performing the walk

```
for (steps = 0; steps < maxit; steps++) { /* take maxit steps */
    if (keypressed())                       /* abort if told to do so */
        return(0);
    temp = rand();                          /* calculate the next xstep */
    xstep = ((temp/tempadjust) - 2.0) * stepsize;
    xstep = min(xwalk + xstep, xdots - 1);
    xstep = max(0, xstep);
    temp = rand();                          /* calculate the next ystep */
    ystep = ((temp/tempadjust) - 2.0) * stepsize;
    ystep = min(ywalk + ystep, ydots - 1);
    ystep = max(0, ystep);
    if (param[1] == 0.0)                    /* rotate the colors? */
        if (++color >= colors)              /* rotate the colors, avoiding */
            color = 1;                      /* the background color 0 */
    /* the draw_line function is borrowed from the 3D routines */
    draw_line(xwalk, ywalk,xstep,ystep,color);
    /* or, we could be on a pogo stick and just displaying
       where we landed...
    putcolor(xstep, ystep, color);
    */

    xwalk = xstep;                          /* remember where we were */
    ywalk = ystep;
    }
return(1);                                  /* we're done */
```

Generating a Mandelbrot Image

Listing 8-11 shows the portion of the demowalk() routine that performs the optional Mandelbrot function invoked when the user sets the average stepsize parameter to 999.

This Mandelbrot routine was inserted only to show how a stand-alone fractal type accesses the *x*- and *y*- coordinates of each pixel in its image. However, it also serves to demonstrate the disadvantages of implementing an escape-time fractal type as a stand-alone fractal. Because it's not using the escape-time fractal engine, this fractal type does not have automatic access to any of the standard escape-time options like solid-guessing, boundary-tracing, biomorphs, decomposition, any of the inside or outside options, etc. Also, it's incredibly slow.

Note that the floating-point arrays used by the Mandelbrot option are only filled in if a fractal type has been declared to use a floating-point type algorithm. Integer fractals use long far array equivalents called lx0[], ly0[], lx1[], and ly1[].

Listing 8-11 The demowalk() fractal generation routine—the Mandelbrot option

```
} else {                        /* a simple Mandelbrot routine */

    /* the following routine determines the X and Y values of
       each pixel coordinate and calculates a simple mandelbrot
       fractal with them - slowly, but surely */
    int ix, iy;
    for (iy = 0; iy < ydots; iy++) {
        for (ix = 0; ix < xdots; ix++) {
            int iter;
            double x, y, newx, newy, tempxx, tempxy, tempyy;
            /* first, obtain the X-and Y-coordinate values of this pixel */
            x = dx0[ix]+dx1[iy];
            y = dy0[iy]+dy1[ix];
            /* now initialize the temporary values */
            tempxx = tempyy = tempxy = 0.0;
            if (keypressed())       /* abort if told to do so */
                return(0);
            /* the inner iteration loop */
            for (iter = 1; iter < maxit; iter++) {
                /* calculate the x and y values of Z(iter) */
                newx = tempxx - tempyy + x;
                newy = tempxy + tempxy + y;
                /* calculate the temporary values */
                tempxx = newx * newx;
                tempyy = newy * newy;
                tempxy = newx * newy;
                /* are we done yet? */
                if (tempxx + tempyy > 4.0) break;
                }
            /* color in the pixel */
            putcolor(ix, iy, iter & (colors - 1));
            }
        }
    return(1);                      /* we're done */
    }

}
```

Adding Escape-Time Fractals

Now let's add a family of four escape-time fractal types to Fractint—integer and floating-point versions of the familiar Mandelbrot and Julia sets.

Adding Entries to the FRACTYPE.H File

The first thing you must do when adding new fractal types to Fractint is to add definitions in the FRACTYPE.H file. These definitions point to the fractal's locations in the fractalspecific file. For many fractal types, including the ones you're about to add, these definitions are used only inside the fractalspecific structure (and in fact for some fractal types, they aren't even used there), but it's a good practice to always add these definitions to FRACTYPE.H, so they're all in one place. You'll need four entries for the integer and floating-point variations of both the Mandelbrot and Julia types. The values of these entries will be the location of the entries you will be adding to the end of the fractalspecific structure. Listing 8-12 shows the four entries you need to add.

Listing 8-12 Adding the demo fractal types to FRACTYPE.H

```
#define DEMOMANDEL           162
#define DEMOJULIA            163
#define DEMOMANDELFP         164
#define DEMOJULIAFP          165
```

Adding Entries to the Fractalspecific Structure

Having added the entries for these four new fractal types, let's add their entries to the end of the fractalspecific array in FRACTALP.C, just before the NULL entry indicating the end of the list. Listing 8-13 shows the four new entries we need to add.

Listing 8-13 Adding the demo fractal types to FRACTALP.C

```
"demomandel", realz0, imagz0,"","",0,0,0,0,
-1, -1, WINFRAC,
-2.5, 1.5, -1.5, 1.5, 1, DEMOJULIA, NOFRACTAL, DEMOMANDELFP, XAXIS_NOPARM,
JuliaFractal, mandel_per_pixel, MandellongSetup, StandardFractal, STDBAILOUT,

"demojulia", realparm, imagparm,"","",0.6,0.55,0,0,
-1, -1, WINFRAC,
-2.0, 2.0, -1.5, 1.5, 1, NOFRACTAL, DEMOMANDEL, DEMOJULIAFP, ORIGIN,
JuliaFractal, julia_per_pixel, JulialongSetup, StandardFractal, STDBAILOUT,

"*demomandel", realz0, imagz0,"","",0,0,0,0,
```

continued on next page

continued from previous page

```
-1, -1, WINFRAC,
-2.5, 1.5, -1.5, 1.5, 0, DEMOJULIAFP, NOFRACTAL, DEMOJULIA, XAXIS_NOPARM,
JuliafpFractal, mandelfp_per_pixel, MandelfpSetup, StandardFractal, STDBAILOUT,

"*demojulia", realparm, imagparm,"","",0.6,0.55,0,0,
-1, -1, WINFRAC,
-2.0, 2.0, -1.5, 1.5, 0, NOFRACTAL, DEMOMANDELFP, DEMOMANDEL, ORIGIN,
JuliafpFractal, juliafp_per_pixel, JuliafpSetup, StandardFractal, STDBAILOUT,
```

Let's go over the demomandel entry. Refer to Listings 8-3 through 8-6 as we do so, as they include the definitions of all flags and parameter strings used by this entry. The fractal type's name, used whenever Fractint needs to identify this fractal type, is "demomandel." It accepts two optional parameters (realparm and imagparm), both of which default to 0. Helptext and helpformula are –1, as we have not developed any help text for this fractal type. Its only flag is WINFRAC, which signals that Fractint's Windows port, Winfract, can handle it. Its screen corners are –2.5.1.5,–1.5,1.5—the standard, slightly off-center corner values that display the initial Mandelbrot image well. Isinteger is 1, indicating that this is an integer fractal. Its Julia equivalent, reached by pressing the right mouse button, is DEMOJULIA. (Being a Mandelbrot fractal, it has no Mandelbrot equivalent so that entry is NOFRACTAL.) Its alternative floating-point algorithm is the one in the DEMOMANDELFP entry. Its symmetry (XAXIS_NOPARM) is about the *x*-axis, but only if its optional parameters are all zero. We'll go into the four function pointers in detail in the following paragraphs. Finally, its bailout value is the standard bailout value, 4.0.

Once you've added (or removed the comments from) these entries in FRACTYPE.H and FRACTALP.C, rebuild Fractint and run it again. You now have two new fractal types, demomandel and demojulia. These new fractal types have both floating-point and integer versions, and are connected via the Mandelbrot/Julia toggle activated by the right mouse button. Not only do the basic algorithms work, but they also work with all of Fractint's myriad options and doodads.

Now take a look at the entry for the mandel4 fractal type in FRACTALP.C, shown in Listing 8-14. Note that the main difference between this entry and the one for demomandel is the pointer to a different orbitcalc() routine, Mandel4Fractal() (found in FRACTALS.C). All of the other function pointers are identical.

Listing 8-14 The Mandel4 entry in FRACTALP.C

```
"mandel4", realz0, imagz0,"","",0,0,0,0,
HT_MANDJUL4, HF_MANDEL4, WINFRAC,
-2.0,  2.0, -1.5,  1.5, 1, JULIA4,     NOFRACTAL, NOFRACTAL,  XAXIS_NOPARM,
Mandel4Fractal,  mandel_per_pixel, MandellongSetup, StandardFractal,  STDBAILOUT,
```

The Four Function Pointers

The function pointers are the heart of the actual fractal calculation process, so it's worthwhile to go over the four function pointers in each of these new entries and see what the functions they point to actually do. If you're perusing through the source code while you're reading this chapter, all of the functions referred to in this section reside in the FRACTALS.C module unless noted otherwise.

The per_image() Routines

The new fractal types point to one of four per_image() routines. The routine depends on whether the fractal is a Mandelbrot or a Julia and on whether it uses an integer or floating-point algorithm. In all four cases, we're invoking standardized routines used by a number of escape-time fractal types. The routines are actually quite similar, and they're really simpler than they first appear. In fact, the bulk of the code in these routines covers special cases and variables that don't apply to any of the fractal types in our examples (the usual penalty for writing general-purpose subroutines). Listing 8-15 shows the source code for the MandellongSetup() routine as it appears in the FRACTALS.C module.

Listing 8-16 shows the same routine and its three companions stripped down to the code that affects our new routines. All they do is initialize a pointer to a parameter structure (longparm for the integer types or floatparm for the floating-point types). In the case of the Mandelbrot fractals, this parameter points to a structure containing the current pixel coordinates. In the case of the Julia fractals, it points to a structure containing a parameter value (the parameter value is provided either manually by the user or automatically via the Mandelbrot/Julia toggle function). The Julia fractals also call the get_julia_attractor() routine (located in the FRACSUBR.C module) that checks to see if the finite attractor option has been enabled. If the finite attractor option has been enabled, get_julia_attractor() performs some special initialization. Normally, get_julia_attractor() just returns without doing anything.

Listing 8-15 MandellongSetup() source

```
MandelLongSetup()
{
   FgHalf = fudge >> 1;
   c_exp = param[2];
   if(fractype==MARKSMANDEL && c_exp < 1)
      c_exp = 1;
   if(fractype==LMANDELZPOWER && c_exp < 1)
      c_exp = 1;
```

continued on next page

continued from previous page

```
   if((fractype==MARKSMANDEL   && !(c_exp & 1)) ||
       (fractype==LMANDELZPOWER && c_exp & 1))
      symmetry = XYAXIS_NOPARM;    /* odd exponents */
   if((fractype==MARKSMANDEL && (c_exp & 1)) || fractype==LMANDELEXP)
      symmetry = XAXIS_NOPARM;
   if(fractype==SPIDER && periodicitycheck==1)
      periodicitycheck=4;
   longparm = &linit;
   if(fractype==LMANDELZPOWER)
   {
      if(param[4] == 0.0 && debugflag != 6000  && (double)c_exp == param[2])
          fractalspecific[fractype].orbitcalc = longZpowerFractal;
      else
          fractalspecific[fractype].orbitcalc = longCmplxZpowerFractal;
      if(param[3] != 0 || (double)c_exp != param[2] )
         symmetry = NOSYM;
    }

   return(1);
}
```

Listing 8-16 ...Setup() source simplified

```
MandellongSetup()
{
    /* initialize the parm pointer to point to the current pixel coordinate values */
     longparm = &linit;
     /* return, indicating that the fractal has yet to be generated */
     return(1);
}

JulialongSetup()
{
    /* initialize the parm pointer to point to the parameter entry  */
    longparm = &lparm;
    /* invoke the julia attractor option, if its been set */
    get_julia_attractor (0.0, 0.0);
    /* return, indicating that the fractal has yet to be generated */
   return(1);
}
MandelfpSetup()
{
    /* initialize the parm pointer to point to the current pixel coordinate values */
   floatparm = &init;
    /* return, indicating that the fractal has yet to be generated */
   return(1);
}
JuliafpSetup()
{
     /* initialize the parm pointer to point to the parameter entry */
    floatparm = &parm;
     /* invoke the julia attractor option, if its been set */
     get_julia_attractor (0.0, 0.0);
     /* return, indicating that the fractal has yet to be generated */
    return(1);
}
```

The calctype() Routines

All of the previous examples use the same calctype() function. This is the standard, escape-time fractal routine, StandardFractal(), located in the CALCFRAC.C module. The core fractal logic in CALCFRAC.C recognizes the StandardFractal() entry as being the escape-time fractal routine. When it sees that StandardFractal() is specified as the calctype() entry, the core fractal logic switches to its escape-time routines, enables its escape-time options such as solid-guessing and boundary-tracing, and calls StandardFractal() once for each pixel. StandardFractal() handles all sorts of options like inside coloring, outside coloring, biomorphs, decomposition, logarithmic palettes, and such automatically, making them a feature of every escape-time fractal type.

The per_pixel() Routines

The examples use one of several per_pixel() routines, depending on whether they are Mandelbrot or Julia fractals and on whether they use the integer or floating-point algorithms. As in the case of the per_image() routines, these are all standardized routines used by a number of escape-time fractal types. Listing 8-17 shows the mandel_per_pixel() routine called by many of the Mandelbrot-style functions.

This routine has special logic to handle the inversion option (with inversion, the initial pixel value is really from a different (everted) location) and several of the esoteric inside options. Listing 8-18 shows the same routine and its sister routines stripped of those options and down to fighting trim for comparison purposes.

Aside from those option checks, all the routines do is initialize a few variables so that the first call to the orbitcalc() routines can calculate $z(1)$. The old/lold structures hold the results of the previous iteration—$z(0)$ in this case, and the value of the current pixel coordinate position. For the Mandelbrot fractals, $z(0)$ may be modified by an optional parameter. The *tempsqrx/tempsqry* variables contain the square of the real and imaginary components of the previous calculation. Normally, they are leftover values from the previous iteration, so they have to be explicitly calculated here because on the first call to the per_orbit() routines there is no previous iteration.

Listing 8-17 The mandel_per_pixel() routine

```
int mandel_per_pixel()
{
   /* mandel */

   if(invert)
   {
      invertz2(&init);

      /* watch out for overflow */
      if(bitshift <= 24)
         if (sqr(init.x)+sqr(init.y) >= 127)
         {
          init.x = 8;  /* value to bail out in one iteration */
          init.y = 8;
         }
      if(bitshift >  24)
      if (sqr(init.x)+sqr(init.y) >= 4)
         {
          init.x = 2;  /* value to bail out in one iteration */
          init.y = 2;
         }

      /* convert to fudged longs */
      linit.x = init.x*fudge;
      linit.y = init.y*fudge;
   }
   else
      linit.x = lx0[col]+lx1[row];
   switch (fractype)
     {
      case MANDELLAMBDA:              /* Critical Value 0.5 + 0.0i   */
        lold.x = FgHalf;
        lold.y = 0;
        break;
      default:
        lold = linit;
        break;
      }

   /* alter init value */
   if(useinitorbit == 1)
      lold = linitorbit;
   else if(useinitorbit == 2)
      lold = linit;
   if(inside == -60 || inside == -61)
   {
      /* kludge to match "Beauty of Fractals" picture since we start
       Mandelbrot iteration with init rather than 0 */
      lold.x = lparm.x; /* initial pertubation of parameters set */
      lold.y = lparm.y;
      color = -1;
   }
   else
   {
      lold.x += lparm.x; /* initial pertubation of parameters set */
      lold.y += lparm.y;
   }
   ltmp = linit; /* for spider */
   ltempsqrx = multiply(lold.x, lold.x, bitshift);
   ltempsqry = multiply(lold.y, lold.y, bitshift);
   return(1); /* 1st iteration has been done */
}
```

Listing 8-18 The ...per_pixel functions simplified

```
int mandel_per_pixel()
{
   /* Z(0) is the value of the pixel coordinates */
   lold = linit;
   /* add in the optional parameter, if any */
   lold.x += lparm.x;
   lold.y += lparm.y;
   /* precalculate temporary values for the next iteration */
   ltempsqrx = multiply(lold.x, lold.x, bitshift);
   ltempsqry = multiply(lold.y, lold.y, bitshift);
   /* return - we're done */
   return(0);
}
int julia_per_pixel()
{
   /* Z(0) is the value of the pixel coordinates */
   lold.x = lx0[col]+lx1[row];
   lold.y = ly0[row]+ly1[col];
   /* precalculate temporary values for the next iteration */
   ltempsqrx = multiply(lold.x, lold.x, bitshift);
   ltempsqry = multiply(lold.y, lold.y, bitshift);
   /* return - we're done */
   return(0);
}

int mandelfp_per_pixel()
{
```

continued on next page

continued from previous page

```
   /* Z(0) is the value of the pixel coordinates */
   old = init;
   /* add in the optional parameter, if any */
   old.x += parm.x;
   old.y += parm.y;
   /* precalculate temporary values for the next iteration */
   tempsqrx = sqr(old.x);
   tempsqry = sqr(old.y);
   /* return - we're done */
   return(0);
}

int juliafp_per_pixel()
{
   /* Z(0) is the value of the pixel coordinates */
   old.x = dx0[col]+dx1[row];
   old.y = dy0[row]+dy1[col];
   /* precalculate temporary values for the next iteration */
   tempsqrx = sqr(old.x);
   tempsqry = sqr(old.y);
   /* return - we're done */
   return(0);
}
```

The orbitcalc() Routines

The Mandelbrot and Julia examples use the same orbitcalc() logic. Given that the longparm/floatparm pointer has already been redirected by the per_image() routine, once you get those fractal types initialized with $z(0)$ they all use the same formula to calculate $z(n+1)$ from $z(n)$. There are floating-point and integer variants of this routine, JuliaFractal() and JuliafpFractal(). Listing 8-19 shows both of them and the bailout routines they call. This is one case where the floating-point algorithms are easier to follow, as the integer algorithms have to include some fairly contorted logic to ensure that they detect all the overflow possibilities.

Listing 8-19 The JuliaFractal and JuliafpFractal routines and their bailout functions

```
JuliaFractal()
{
   lnew.x  = ltempsqrx - ltempsqry + longparm->x;
   lnew.y = multiply(lold.x, lold.y, bitshiftless1) + longparm->y;
   return(longbailout());
}
JuliafpFractal()
{
   new.x = tempsqrx - tempsqry + floatparm->x;
   new.y = 2.0 * old.x * old.y + floatparm->y;
   return(floatbailout());
}
```

```
static int near floatbailout()
{
   if ( ( magnitude = ( tempsqrx=sqr(new.x) )
                    + ( tempsqry=sqr(new.y) ) ) >= rqlim ) return(1);
   old = new;
   return(0);
}
int longbailout()
{
  /* the real routine is in assembler for speed: this is the C equivalent */
   ltempsqrx = lsqr(lnew.x);
   ltempsqry = lsqr(lnew.y);
   lmagnitud = ltempsqrx + ltempsqry;
   if (lmagnitud >= llimit || lmagnitud < 0 || labs(lnew.x) > llimit2
         || labs(lnew.y) > llimit2 || overflow) {
               overflow=0;
               return(1);
               }
   lold = lnew;
}
```

Algorithmically, all the routines do is perform the function listed here. The actual code is contorted to save calculation time: this iteration's new.x is also next iteration's old.x, so there's no reason to recalculate the squares of the old values given that you've already calculated them (as the squares of the new values) in the previous iteration.

```
/*  algorithm notes:
    old.x and old.y are the values of the x and y results of
the prior iteration
    new.x and new.y are the values of the x and y results of
this iteration
    tempsqrx and tempsqry are temporary values for x squared
and
    y squared. The real routines, being clever, re-use these
values
    when they calculate new.x in the next iteration.
    bailoutlimit is the bailout value (usually 4) defined by
the orbit_bailout
    entry in the fractal-specific structure, but perhaps
changed to a
    different value as a user option.
*/
new.x = (old.x * old.x) - (old.y * old.y) + param.x;
new.y = 2 * old.x * old.y + param.y;
tempsqrx = new.x * new.x;
tempsqry = new.y * new.y;
if  ( (tempsqrx + tempsqry ) >= bailoutlimit)
    return(1);
else
    return(0);
```

APPENDIX A

FRACTINT AND VIDEO ADAPTERS

FRACTINT AND VIDEO ADAPTER DETECTION

When Fractint starts up, one of the first things it does is examine your hardware to determine what kind of video adapter is available. First, it runs through a series of basic checks to determine if your video adapter is CGA, Hercules, EGA, or VGA-compatible. Then, if your video adapter is VGA-compatible, Fractint runs through a second series of checks to determine whether your video adapter is based on one of several popular super VGA chipsets that can handle higher resolution modes like 640 x 480 x 256 or 1024 x 768 x 16. This video autodetection logic is what gives you the ability to tell Fractint to generate a 640 x 480 256-color image without telling it (or even knowing) what kind of video adapter is on your system. This appendix covers the limitations of Fractint's autodetection logic and describes how to modify or disable it if it runs into problems on your system.

VIDEO ADAPTER MEMORY

One limitation of Fractint's video autodetection logic is that it assumes that every video adapter has the maximum amount of video memory available to it. This means that Fractint sometimes thinks that your PC can display video modes it really can't, because your video adapter just doesn't have the video memory to work with. For instance, Fractint assumes that every EGA adapter has 256K of adapter memory and can handle 640 x 350 16-color mode—cheerfully ignoring the fact that some early IBM EGA adapters came supplied with only 64K of video memory and topped out at 640 x 350 2-color mode. Many third-party super VGA chipsets can handle video modes that require a full megabyte of memory or more—*if* the video adapter on which they reside contains that much memory.

When you attempt to use a video mode you don't really have the capability to handle—say, using a 640 x 480 256-color mode which requires that a video adapter have 512K of video memory on an adapter that only has 256K—Fractint goes right ahead and tells your video BIOS to go into that mode.

When that happens, one of several results can occur. Your video may stay in text mode (even though Fractint thinks it is in a graphics mode), or it may be in graphics mode, but with only a portion of the display showing a good image. In either case, the solution is the same—press ESC to get back into text mode and then select another video mode.

VIDEO MONITOR LIMITATIONS

All VGA monitors can handle 640 x 480 resolution—but not every monitor can handle 800 x 600 or 1024 x 768. It is possible that your system has a video adapter that can generate video modes your monitor can't handle. Fractint assumes that if you attempt to use a particular video mode and your video adapter can handle it, then your monitor can handle it as well.

When a video adapter attempts to throw a monitor into a resolution beyond its capability, you get a generally trashed display, usually of moving diagonal lines. If this happens to you when you attempt to enter a high-resolution video mode, press ESC to get back into text mode and then select another video mode.

It is not a good idea to push a video monitor beyond its rated limits. A monitor can be damaged or destroyed when a video adapter attempts to run it at a higher resolution than it was built to handle. If you inadvertently attempt to use a video resolution beyond the capabilities of your monitor, don't spend a lot of time watching that weird display before pressing ESC.

VIDEO CHIPSET DETECTION PROBLEMS

Super VGA chipset autodetection is wonderful—when it works. Fractint's super VGA detection algorithms are, in the authors' humble opinions, among the best in the business, but sometimes even that isn't good enough. Unfortunately, the only way to detect the presence of some video chipsets is to write to certain adapter locations and read back the results, and there is always the possibility that some new chipset introduced by vendor "A" just happens to pass the detection logic Fractint uses for super VGA adapter "B"—or what's worse, locks up solid when that chipset detection logic is used. In the former case, all of Fractint's standard VGA video modes will work, but none of its super VGA modes. In the latter case, older versions of Fractint have been known to lock up the system entirely as soon as they were started. Fortunately, there are solutions for these cases.

If your video adapter came with a VESA VBE driver (usually a file named something like VESA.COM or VESA.EXE), invoking that driver before starting up Fractint will often solve any adapter-detection problems. VESA stands for *Video Electronics Standards Association*, and VBE stands for its *Video BIOS Extension* standard. This standard is one of the best things that ever happened to the super VGA adapter world, as it gives DOS-based programs a standard way to access super VGA video adapters regardless of the chipsets they use. During its super VGA detection routines, Fractint checks for VESA compliance first. If it finds that your video adapter responds to its VESA requests, Fractint never executes any of its chipset-specific logic.

If you don't have a VESA driver, but know what type of chipset your video adapter uses, you can use the `adapter=` command-line parameter to force Fractint to bypass its internal video chipset detection logic completely and assume the presence of a particular chipset. The `adapter=` command-line parameter is discussed in detail at the end of Chapter 5, *Fractint Reference*, at the beginning of the *command-line only commands* section. Try this parameter on your command-line first. If it works there, add it to your SSTOOLS.INI file so that Fractint uses it every time it runs. (Note that Fractint looks for the video chipset on your adapter rather than the name of the vendor who built or sold it. To the software program, it's the chipset that is important—and as it happens, several popular PC vendors have switched video chipsets several times.) The command-line parameter `adapter=vga` causes Fractint to assume the presence of a vanilla VGA adapter that handles no super VGA video modes.

GRAPHICS-TO-TEXT-TO-GRAPHICS SWITCH FAILURE

Fractint's default method of switching from graphics mode to text mode and back (such as when you press (TAB) to check on the status of your image and then press (ENTER) to return to that image) is extremely sophisticated—and makes some stringent assumptions about the VGA-compatibility of your video adapter. One could argue that these assumptions are particularly stringent (to the point of being absurd) when they are applied to a video adapter which is in a super VGA video mode that "real" VGA adapters can't get into in the first place.

If your graphics image is corrupted (usually a generally correct image but with a few rows of trashed pixels) when you switch from graphics mode to text mode and back, then your graphics adapter doesn't match all of those assumptions. If this happens to you when you return to your graphics image from a TAB, F1, or ESC sequence, Fractint's **textsafe=** command-line parameter will probably solve your problems. This command-line parameter causes Fractint to use one of several alternative approaches to handling its graphics-to-text-to-graphics switch. The **textsafe=** command-line parameter is discussed in detail at the end of Chapter 5, *Fractint Reference*, at the beginning of the *command-line only commands* section. Try this parameter and its different options on your command-line first. If one works there, add it to your SSTOOLS.INI file so that Fractint uses it every time it runs.

APPENDIX B

FRACTINT AND GIF FILES

Fractint stores its images in GIF (*Graphics Interchange Format*) files. The GIF standard was developed and is maintained by CompuServe to meet the needs of their on-line customer base. Because of this design goal (and the fact that CompuServe's subscribers access CompuServe using all kinds of computer hardware over relatively slow dial-up lines), the GIF standard offers several advantages over other formats.

Platform Independence—GIF images are stored in a format independent of the computer system on which they were generated. GIF files developed by Fractint are displayable on any system for which someone has written a GIF viewer, and GIF viewers are available for virtually every graphics-capable system on the market today.

Compression—GIF images are stored in an internally compressed format using a variant of the popular LZW algorithm. As a result, high-resolution images take up much less room on your hard disk when stored as GIF files than when stored in other formats—an important consideration for an image-oriented program such as Fractint.

Application-Specific Extensions—When CompuServe enhanced the GIF standard with the GIF89a version, GIF images were given the ability to include *application-specific extension blocks* as part of the image stream. Application-specific extension blocks include a name field identifying the application that created them, and are ignored by any other program accessing the GIF image. Fractint uses this capability (and the identifier "FRACTINT") to store any fractal-specific information about the image as part of the GIF file, and recovers that information along with the image when it reads that file later. This is how Fractint knows, for example, that FRACT123.GIF is a partially completed image of a Julia set with its parameters set at (0.25,0.58) and corners at (–1.4,0.06)(0.45,0.60) which was calculating line 143 using the second pass of the solid-guessing algorithm when it was saved—and is able to resume that calculation immediately after reloading the image.

Because some older GIF viewers can handle only the older GIF87a format, Fractint includes a command-line option (`gif87a=yes`) that forces it to store its images using that older format. Note that when it does so, it can't include its fractal-specific extension block as part of the image and, therefore, can only view the image as if it were one created by some other, nonfractal program if it reloads it later.

The Graphics Interchange Format© is the copyright property of CompuServe Incorporated. GIF^{SM} is a Service Mark property of CompuServe Incorporated.

APPENDIX C

COMPLEX AND HYPERCOMPLEX NUMBERS

Fractint uses a variety of different kinds of number systems to generate fractals. These number systems include complex numbers, quaternions, and hypercomplex numbers. You may not be familiar with these different number systems, so the basic facts about them are presented here.

Complex numbers, the extension of the algebra of real numbers to two dimensions, are the most familar of these various number systems. Complex numbers have desirable properties that make them ideal for use in generating fractals. You can add, subtract, multiply, and divide complex numbers just as you do familiar real numbers. Because complex numbers are two dimensional, the pixels on your computer screen can be mapped to complex numbers in a natural way. Another advantage of complex numbers for our purposes is that many of the functions that work with real numbers (such as the sine, cosine, exponential, and logarithm) can be extended to complex numbers, so we can use these functions to build fractal formulas.

The quaternions are a generalization of complex numbers to four dimensions. They are useful in the theoretical formulations of physics, and are familiar to students of abstarct algebra. Alan Norton (1982) and John Hart (1989) have generated intriguing three-dimensional images of quaternion Julia sets. Now you can do the same thing with Fractint's julibrot fractal type.

The hypercomplex numbers are similar to quaternions, but they are not well known because the nineteenth century mathematicians who first worked on the problem of generalizing complex numbers rejected hypercomplex numbers in favor of quaternions for use in physics. For our purposes, however, hypercomplex numbers have a big advantage over quaternions. We can build hypercomplex fractal formulas using all the functions we use to generate fractals with complex numbers. Hypercomplex numbers were brought to the attention of the Stone Soup Group by Clyde Davenport, who has written an expository extension of his master's thesis entitled *A Hypercomplex Calculus with Applications to Relativity*. To the best of our knowledge, the use of hypercomplex numbers to generate fractals is presented here in *Fractal Creations, Second Edition* and Fractint version 18 for the first time.

This appendix provides you with a summary of the basic rules governing the algebras of complex numbers, quaternions, and hypercomplex numbers. For each number system, you will learn how the arithmetic operations of addition, subtraction, multiplication, and division work, and how to compute transcendental functions applied to these different numbers. This information is intended as a reference for those who want to implement fractal generation software of their own or simply want to better understand the mathematics underlying Fractint's higher dimensional fractals. Space permits only a brief exposition of algebraic properties and not a full tutorial.

COMPLEX NUMBERS

You may remember from your high school days that it is illegal to take the square root of a negative number. The real number system is not closed under the operation of taking roots, and in particular, the square root of a negative number cannot be a real number. (A number system is said to be *closed* under an operation if the result of that operation on numbers from that system yields a result in the original number system.) The real number system can be extended to a larger number system that is closed under the operation of taking roots. The result of this extension is the complex number system.

The Number *i*

The complex number system is created from the real number system by defining a single additional element defined as a number i such that $i^2 = -1$. This $i = (-1)$. All the rules of addition, subtraction, multiplication, and division work with this extended number system, and any complex number z can be written in the form $z = a1 + bi$, where a and b are real numbers. (The "1" is usually not written, but we did it this once to emphasize that every complex number can be written as a linear combination of the two numbers 1 and i.) The number a is called the real part of z and the number b is the imaginary part of z. A number with no real component is called an imaginary number. We will sometimes write the complex number $a + bi$ as the ordered pair (a,b). The (a,b) notation emphasizes that the complex numbers form a two-dimensional space.

The term "imaginary" is unfortunate, because you might think from the name that imaginary numbers are somehow not legitimate numbers. Imaginary numbers are no less "real" or any more "imaginary" in the ordinary sense of those words than the real numbers. Complex numbers definitely apply to the "real" world in a very strong way. Engineering subjects ranging from spacecraft orbital dynamics to electric power transfer are heavily dependent on them.

Algebraic Properties of Complex Numbers

The complex numbers are a successful extension of the real numbers to two dimensions. The real numbers form what mathematicians call a field. A field has two operations, addition and multiplication, which are commutative, associative, and distributive. The associative law says that the order of adding or multiplying makes no difference, that for example $(ab)c$ is the same as $a(bc)$. The commutative law states that adding or multiplying from the left is the same as adding or multiplying from the right. For example, $a + b = b + a$. The two operations of addition and multiplication are related in a distributive law, stating that $a(b + c) = ab + ac$. Both operations have an identity element (0 for addition, 1 for multiplication) that when applied to an element result in no change: $a + 0 = a$ and $a1 = a$. Both operations have an inverse operation, so that for every a, there is an element $-a$ such that $a + (-a) = 0$. Similarly, for every nonzero a, there is an element $1/a$ such that $a\,(1/a) = 1$.

When the real numbers are extended to form the complex number system, every single one of the field properties still applies. The complex numbers obey the same algebraic rules as the real numbers.

Complex Arithmetic

All you need to know to do arithmetic with complex numbers is that $i^2 = -1$ and that the regular algebra rules of real number arithmetic apply.

To add or subtract two complex numbers, you just add or subtract the real and imaginary components. For example,

$$\begin{aligned}(2 + 3i) + (3 + 4i) &= (2 + 3) + (3 + 4)i \\ &= 5 + 7i\end{aligned}$$

and

$$\begin{aligned}(2 + 3i) - (3 + 4i) &= (2 - 3) + (3 - 4)i \\ &= -1 - i\end{aligned}$$

To multiply, treat i as a variable and use the rules of algebra, in particular the distributive law, which states that $a(b + c) = ab + ac$. Then replace any instances of i^2 with -1. Collect all the real and imaginary parts together. Therefore,

$$\begin{aligned}(2 + 3i)(4 + 5i) &= (2 + 3i)4 + (2 + 3i)5i \\ &= 2(4) + 12i + 10i + 15i^2 \\ &= 2(4) + 12i + 10i + 15(-1) \\ &= (8 - 15) + (12 + 10)i\end{aligned}$$

To divide, the trick is to simplify the demominator so that it is in the form $a + bi$. Then multiply the top and bottom of the fraction by the complex conjugate of the denominator. (The complex conjugate of $a + bi$ is $a - bi$; or the original number with the imaginary component negated.) The reason this trick works is that when you multiply a complex number by its conjugate, the result is a real number, so the denominator will not have an imaginary component. Then you can simplify the result to the $a + bi$ form. For example:

$$\begin{aligned}(2 + 3i)/(4 + 5i) &= ((2 + 3i)(4 - 5i))/((4 + 5i)(4 - 5i)) \\ &= (2(4) - 2(5i) + 3i(4) - (3i)(5i))i / 4(4) - (5i)(5i) \\ &= (8 - 10i + 12i - 15i^2)/(16 - 25i^2) \\ &= ((8 + 15) + (12 - 10)i)/ (16 - 25(-1)) \\ &= (23 + 2i)/41 \\ &= 23/41 + (2/41)i\end{aligned}$$

Transcendental Functions

Many functions that operate with real numbers can be generalized to complex numbers. Some examples are the trigonometric functions sine(x) and cosine(x), the exponential e^x, and the natural logarithm ln(x). Because computer languages and libraries usually provide only the real versions of these functions, the programmer who desires to write fractal programs involving transcendental functions needs formulas that reduce the complex version of functions to their real equivalents. This information is surprisingly difficult to find. Books about complex analysis are concerned about the development of analytical theory, and the derivation of the formulas for transcendental functions is a very low priority that is often not covered. We present here the information we wish we had had when Fractint was first programmed.

Name	Function	Formula
Exponential	e^{x+iy}	$= e^x\cos(y) + ie^x\sin(y)$
Sine	$\sin(x+iy)$	$= \sin(x)\cosh(y) + i\cos(x)\sinh(y)$
Cosine	$\cos(x+iy)$	$= \cos(x)\cosh(y) - i\sin(x)\sinh(y)$
Hyperbolic sine	$\sinh(x+iy)$	$= \sinh(x)\cos(y) + i\cosh(x)\sin(y)$
Hyperbolic cosine	$\cosh(x+iy)$	$= \cosh(x)\cos(y) + i\sinh(x)\sin(y)$
Natural logarithm	$\ln(x+iy)$	$= (1/2)\ln(x^2 + y^2) + i(\arctan(y/x) + 2k\text{pi})$ $(k = 0, +-1, +-2, +-....)$
Tangent	$\tan(x+iy)$	$= (\sin(2x)/(\cos(2x) + \cosh(2y)))$ $+ i\sinh(2y)/(\cos(2x) + \cosh(2y))$
Hyperbolic tangent	$\tanh(x+iy)$	$= (\sinh(2x)/(\cosh(2x) + \cos(2y)))$ $+ i\sin(2y)/(\cosh(2x) + \cos(2y))$
Cotangent	$\text{cotan}(x+iy)$	$= (\sin(2x) - i\sinh(2y))/(\cosh(2y) - \cos(2x))$
Hyperbolic cotangent	$\text{cotanh}(x+iy)$	$= (\sinh(2x) - i\sin(2y))/(\cosh(2x) - \cos(2y))$

Table C-1 Transcendental function formulas

In the formulas shown in Table C-1, the left hand of the equation is the complex valued function of a complex variable to be calculated. The right hand of the equation provides a formula in terms of real valued functions of a real variable operating on the real or imaginary components of the complex argument.

QUATERNIONS

When the real numbers are generalized to two dimensions to form the complex numbers, no algebraic properties are lost. The complex numbers obey all of the field properties discussed earlier. A natural question arises: why not extend the complex numbers further, to four dimensions? Alas, the nineteenth century mathematician Frobenius who investigated this problem proved that the quest for four-dimensional "complex" numbers cannot succeed. At least one field property must be sacrificed. The quaternions are such a four-dimensional extension of the complex numbers. They fail the communtative law of multiplication: sometimes *ab* and *ba* are not equal.

$ij = k$	$jk = i$	$ki = j$
$ji = -k$	$kj = -i$	$ik = -j$
$ii = jj = kk = -1$	$ijk = -1$	

Table C-2 Quaternion basis element multiplication rules

The Numbers *i*, *j*, and *k*

Just as the complex numbers can be obtained from the real numbers by adding the special element *i*, the quaternions can be obtained from the complex numbers by adding special elements *j* and *k*. Every quaternion *q* can be written as a linear combination of 1, *i*, *j*, and *k*. That is, for every quaternion *q*, there are real numbers x, y, z, and w such that $q = x + yi + zj + wk$.

Quaternion Arithmetic

Recall that complex number arithmetic boils down to the regular rules for real arithmentic plus the fact that $i^2 = -1$. You can use ordinary algebra to manipulate complex numbers, and whenever you encounter i^2 you can replace it with -1. The analogous idea works for quaternions, with two important differences. Instead of just the new element *i*, we have *i*, *j*, and *k*, so we need to give rules for multiplying these special elements together. Because, the commutative law of multiplication fails, when manipulating quaternion numbers, you cannot assume (as you do when working out real and complex number algebra) that $xy = yx$.

Table C-2 shows the rules for multiplying various combinations of 1, *i*, *j*, and *k* together. Notice that 1 and *i* behave as before, with $i^2 = -1$, so when the *j* and *k* components of a quaternion are zero, the quaternion behaves exactly like a complex number.

The failure of the commutative law is evident from Table C-2. For example, $ij = k$ but in the reverse order, $ji = -k$. The information in Table C-2 is sufficient to multiply any two quaternions together. However, for your convenience, the following formula works out the details.

Let $q_1 = x_1 + y_1 i + z_1 j + w_1 k$ and $q_2 = x_2 + y_2 i + z_2 j + w_2 k$. Then

$$\begin{aligned} q_1 q_2 = {} & 1(x_1x_2 + y_1y_2 - z_1z_2 - w_1w_2) + \\ & i(y_1x_2 + x_1y_2 + w_1z_2 - z_1w_2) + \\ & j(z_1x_2 - w_1y_2 + x_1z_2 + y_1w_2) + \\ & k(w_1x_2 + z_1y_2 - y_1z_2 + x_1w_2) \end{aligned}$$

Quaternion Fractals

Fractint can make both 2-D and 3-D fractals using quaternions. For 2-D cross sections of true four-dimensional fractals, try fractal types quat and quatjul. To see a 3-D solid fractal (actually a 3-D cross section of a 4-D fractal), select fractal type julibrot and choose orbit formula quatjul. (See Chapter 6, *Fractal Types*, for more information on fractal types quat, quatjul, and julibrot.)

HYPERCOMPLEX NUMBERS

Quaternions are not the only possible four-dimensional supersets of the complex numbers. William Hamilton, the discoverer of quaternions in the 1830s, considered an alternative called the hypercomplex number system. We have already mentioned that complex numbers cannot be extended to four dimensions with all the field properties intact. What field property must be sacrificed this time?

Unlike quaternions, the hypercomplex numbers satisfy the commutative law of multiplication: for any hypercomplex numbers a and b, we have $ab = ba$. The law that fails is the field property stating that all nonzero elements of a field have a multiplicative inverse. For nonzero hypercomplex numbers, a, there is no guarantee that there is another element a^{-1} such that $aa^{-1} = 1$.

Today we cannot say for sure why Hamilton decided to use quaternions instead of hypercomplex numbers in the emerging physics. We can only speculate that the inability to confidently divide by nonzero elements was a stumbling block in his mind. He prefered the messy consequences of losing the commutative law of multiplication instead. The rest is history. Today hypercomplex numbers are not well known, and the quaternion product is alive and well in undergraduate electricity and magnetism classes as the *div* and *curl* functions.

Now, a century and a half later, we can second guess Hamilton's decision. For, as we shall see momentarily, hypercomplex numbers have some real advantages for making fractals.

Hypercomplex Arithmetic

As with quaternions, we will define multiplication in terms of the elements 1, i, j, and k, but with subtly different rules. Table C-3 shows the rules for multiplying various combinations of 1, i, j, and k together. As before with quaternions, we still have $i^2 = -1$, so the complex numbers are embedded within the hypercomplex numbers just as they are within the quaternions.

$ij = k$	$jk = -i$	$ki = -j$
$ji = k$	$kj = -i$	$ik = -j$
$ii = jj = -kk = -1$	$ijk = 1$	

Table C-3 Hypercomplex basis element multiplication rules

You have to look very closely to see the difference between quaternion and hypercomplex multiplication. Compare Table C-2 with Table C-3. Just a few signs have changed. Compare the first two lines of each table. In Table C-3, reversing the order of multiplication has no effect. We have $ij = ji$, $jk = kj$, and $ki = ik$. The commutative law holds.

Once again, Table C-3 is sufficient to tell you how to multiply two hypercomplex numbers together. Just use normal algebra, and when you need to simplfy the product of two basic elements, use table C-3. However, the formulas below will help anyone who would like to implement hypercomplex mathematics in their own program.

Let $h_1 = x_1 + y_1i + z_1j + w_1k$ and $h_2 = x_2 + y_2i + z_2j + w_2k$. Then

$$\begin{aligned} h_1h_2 = {} & 1(x_1x_2 - y_1y_2 - z_1z_2 + w_1w_2) + \\ & i(y_1x_2 + x_1y_2 - w_1z_2 - z_1w_2) + \\ & j(z_1x_2 - w_1y_2 + x_1z_2 - y_1w_2) + \\ & k(w_1x_2 + z_1y_2 + y_1z_2 + x_1w_2) \end{aligned}$$

As an added bonus, we'll give you the formula for the reciprocal.

$$\begin{aligned} h^{-1} = {} & 1[x(x^2+y^2+z^2+w^2)-2w(xw-yz)]/[((x-w)^2+(y+z)^2)((x+w)^2+(y-z)^2)]+ \\ & i[-y(x^2+y^2+z^2+w^2)-2z(xw-yz)]/[((x-w)^2+(y+z)^2)((x+w)^2+(y-z)^2)]+ \\ & j[-z(x^2+y^2+z^2+w^2)-2y(xw-yz)]/[((x-w)^2+(y+z)^2)((x+w)^2+(y-z)^2)]+ \\ & k[w(x^2+y^2+z^2+w^2)-2x(xw-yz)]/[((x-w)^2+(y+z)^2)((x+w)^2+(y-z)^2)] \end{aligned}$$

A look at this formula shows the difficulty with hypercomplex numbers. In order to calculate h^{-1}, you have to divide by the quantity $[((x - w)^2 + (y + z)^2)((x + w)^2 + (y - z)^2)]$. So when this quantity is zero, the multiplicative inverse will not exist.

Serendipity Strikes: Generalizing Transcendental Functions

From what has been said so far, you may be wondering why bother with hypercomplex numbers since they are so similar to quaternions. The answer lies in the simplicity with which any complex valued function of a complex variable can be generalized to hypercomplex numbers.

Hypercomplex numbers can be represented as a pair of complex numbers in the following way.

Let $h = x + yi + zj + wk$.
Let $a = (x - w) + i(y + z)$ and
$b = (x + w) + i(y - z)$

The numbers a and b are complex numbers. We can represent h as the pair of complex numbers (a,b). Conversely, if we have a hypercomplex number given to us in the form (a,b), we can solve for x, y, z, and w. The solution to

$$c = c_x + ic_y = (x - w) + i(y + z)$$
$$d = d_x + id_y = (x + w) + i(y - z)$$

is

$$x = (c_x + d_x)/2$$
$$y = (c_y + d_y)/2$$
$$z = (c_y - d_y)/2$$
$$x = (d_x - c_x)/2$$

Define sin(h) as (sin(a),sin(b)). We know how to compute sin(a) and sin(b) (see Table C-1). Let

$$c = \sin(a)$$
$$d = \sin(b)$$

and use the equations above to solve for x, y, z, and w in terms of c and d. The beauty of this is that it really doesn't make any difference what function we use. Instead of sin(), we could have used cos, sinh, ln, or z^2

Using this technique, version 18 of Fractint can create 3-D fractals using the formula $h' = \text{fn}(h) + c$, where "fn" is any of the built-in functions sin, cos, tan, cot, sinh, cosh, log, and so forth. The potential use of hypercomplex numbers to create fractals is still greater. Consider this: every fractal you can generate using Fractint's powerful formula (parser) can be generalized to four dimensions!

ABOUT THE AUTHORS

Tim Wegner and Bert Tyler, two of the programmers of Fractint, have collaborated to bring you *Fractal Creations, Second Edition.* As of the time of this writing, they have never met, but have worked entirely via electronic mail and through CompuServe's GRAPHDEV conference.

TIM WEGNER

Tim Wegner considers himself more of a "math type" than a programmer, although some of the programming skills of the rest of the team may have rubbed off a little on him. He first discovered Bert's Fra386 program in late 1988, and remembers pestering Bert to alter the program so it would run on low-end PCs. Tim's reasons were entirely selfish: he wanted to modify Bert's code to add features, and he only had a lowly 80286-based PC. As soon as Fractint version 6 came out in January of 1989, Tim began to barrage Bert with ideas and code. These included support for super VGA graphics boards, the now-famous color-cycling feature, new fractal types, and 3-D transformation capabilities. Together, Tim and Bert hammered out the main outlines of Fractint's "StandardFractal" architecture and data structures. Tim has been labeled by his cohorts as being "obsessed with options," but he has now paid the price. As a co-author of this book, Tim had to document all the options he so enthusiastically added!

Tim has BA and MA degrees in mathematics from Carleton College and the University of California Berkeley. He worked for seven years overseas as a volunteer, doing things like working with Egyptian villagers building water systems. Since returning to the United States in 1982, he has written shuttle navigation software, a flight scheduling prototype, and supported strategic information planning, all at NASA's Johnson Space Center. He currently is a Member of the Technical Staff of MITRE Houston.

Tim is the author of *Image Lab* and a co-author of *The Waite Group's Fractal Creations (First Edition)* and *Fractals for Windows.*

Bert Tyler is Fractint's original author. He wrote the "blindingly fast" Intel 80386-specific integer math code and the original video mode logic that was the basis of the Fractint program. At one point, Bert understood every line of Fractint's source code, but these days there are so many goodies in there from so many developers that he now claims that he has no idea what some of those routines are doing. Bert's involvement with Fractint began when he downloaded a "Mandelbrot generator" program, fired it up on his brand-new 80386-based PC with no math coprocessor—and then broke out of the program two hours later when it had drawn only half of its first image. Bert remembered that a friend of his had written a Mandelbrot generator for a TI-based processor that used integer math and decided to try programming something similar for his 386—and Fractint was born.

When asked what his best contributions to Fractint have been, Bert answered that they were his decisions to distribute the program with full source code, and give full credit to anyone who sent him improvements. The authors receive major improvements from people they've never heard of before on an almost daily basis.

Bert has a BA in mathematics from Cornell University. He has been in programming since he got a job at the computer center in his sophomore year at college. In other words, he says, he hasn't done an honest day's work in his life. Bert has been know to pass himself off as a PC expert, a UNIX expert, a statistician, and even a financial modeling expert. He is currently passing himself off as an independent PC consultant, supporting PC-to-mainframe communications. Bert is a co-author of *Fractals for Windows*.

ACCESSING THE AUTHORS

Communication between the authors for development of the next version of Fractint takes place in GRAPHDEV (Graphics Developers) Section 4 (Fractal Sources) of CompuServe. Access to this area is open to any and all interested in computer generated images. Stop on by if you have any questions or just want to take a peek at what's getting tossed into the soup! This is a good way to get your Fractint questions answered. The authors are always happy to help Fractint users and to hear suggestions for improving the program.

Bert Tyler	[73477,433] on CompuServe
Timothy Wegner	[71320,675] on CompuServe

THE STONE SOUP GROUP

Fractint is the product of an informal association of programmers and fractal enthusiasts know as the Stone Soup group. Here is the explanation of the origin of that name and an introduction to the *Fractal Creations Second Edition* authors.

THE FABLE OF STONE SOUP

Once upon a time, somewhere in Eastern Europe, there was a great famine. People jealously hoarded whatever food they could find, hiding it even from their friends and neighbors. One day a peddler drove his wagon into a village, sold a few of his wares, and began asking questions as if he planned to stay for the night.

"There's not a bite to eat in the whole province," he was told. "Better keep moving on."

"Oh, I have everything I need," he said. "In fact, I was thinking of making some stone soup to share with all of you." He pulled an iron cauldron from his wagon, filled it with water, and built a fire under it. Then, with great ceremony, he drew an ordinary-looking stone from a velvet bag and dropped it into the water.

By now, hearing the rumor of food, most of the villagers had come to the square or watched from their windows. As the peddler sniffed the "broth" and licked his lips in anticipation, hunger began to overcome their skepticism.

"Ahh," the peddler said to himself rather loudly, "I do like a tasty stone soup. Of course, stone soup with *cabbage*—that's hard to beat."

Soon a villager approached hesitantly, holding a cabbage he'd retrieved from its hiding place, and added it to the pot. "Capital!" cried the peddler. "You know, I once had stone soup with cabbage and a bit of salt beef as well, and it was fit for a king."

The village butcher managed to find some salt beef...and so it went, through potatoes, onions, carrots, mushrooms, and so on, until there was indeed a delicious meal for all. The villagers offered the peddler a great deal of money for the magic stone, but he refused to sell and traveled on the next day. And from that time on, long after the famine had ended, they reminisced about the finest soup they'd ever had.

THE ORIGIN OF FRACTINT

Fractint has grown and developed just like the soup in the fable, with quite a bit of magic, although without the element of deception. You don't have to deceive programmers to make them think that hours of painstaking, often frustrating work is fun—they think that already!

The original "stone" was the program FRA386.EXE written by Bert Tyler, which may still be found on some computer bulletin boards. In some ways it is a little unfair to describe Bert's original program as a humble stone, since Fra386 was a highly polished and capable fractal generator. It's claim to fame was that it was "blindingly fast." But a comparison between the original program and the copy of Fractint packaged with this book shows why the "stone" metaphor is apt. If Fra386 was a tasty morsel, then Fractint is a gourmet feast! The reason is that for several years now fractal enthusiasts from around the world have been sending Bert and the other Stone Soup authors programming onions, potatoes, and spices to add to the soup! You are a beneficiary of this enthusiastic outpouring of creativity, because Fractint is the state of the art of PC fractal programming. It would take a mammoth software development project to duplicate Fractint's features in a commercial program, and by then Fractint would have added still more features.

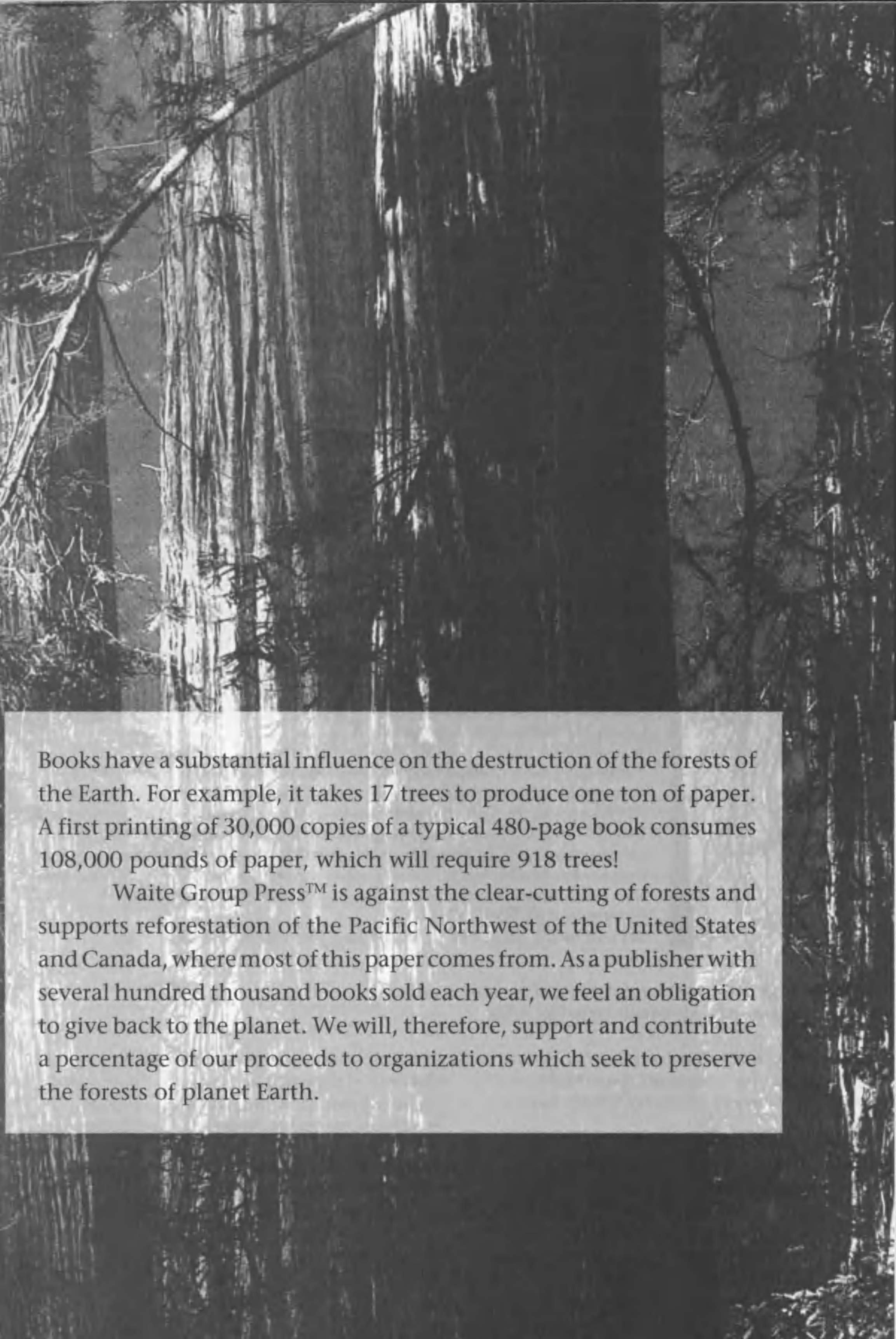
HOW THE STONE SOUP TEAM WORKS

Whenever anyone comes up with ideas for Fractint, those ideas are shared and passed around in the CompuServe GRAPHDEV forum. A feature makes it into the program if someone cares enough to incorporate the feature by writing the code. Because the source code is available, many ideas are sent to the authors as fully integrated source code. Other features start life as suggestions and are eventually coded by one of the authors.

One of the Fractint authors wrote this statement, that sums up the experience of being a "Stone Souper":

There is something unique about the way this group works. We get along well without a formal structure or responsibilities, without clashes, and somehow everyone's efforts come together. Don't ask me how—I think some kind of magic is involved, and certainly some good humor.

INDEX


ENVIRONMENTAL AWARENESS

LIMITED WARRANTY

The following warranties shall be effective for 90 days from the date of purchase: (i) The Waite Group, Inc. warrants the enclosed disk to be free of defects in materials and workmanship under normal use; and (ii) The Waite Group, Inc. warrants that the programs, unless modified by the purchaser, will substantially perform the functions described in the documentation provided by The Waite Group, Inc. when operated on the designated hardware and operating system. The Waite Group, Inc. does not warrant that the programs will meet purchaser's requirements or that operation of the programs will be uninterrupted or error-free. The programs warranty does not cover any programs that has been altered or changed in any way by anyone other than The Waite Group, Inc. The Waite Group, Inc. is not responsible for problems caused by changes in the operating characteristics of computer hardware or computer operating systems that are made after the release of the programs, nor for problems in the interaction of the programs with each other or other software.

THESE WARRANTIES ARE EXCLUSIVE AND IN LIEU OF ALL OTHER WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR OF ANY OTHER WARRANTY, WHETHER EXPRESS OR IMPLIED.

EXCLUSIVE REMEDY

The Waite Group, Inc. will replace any defective disk without charge if the defective disk is returned to The Waite Group, Inc. within 90 days from date of purchase.

This is Purchaser's sole and exclusive remedy for any breach of warranty or claim for contract, tort, or damages.

LIMITATION OF LIABILITY

THE WAITE GROUP, INC. AND THE AUTHORS OF THE PROGRAM SHALL NOT IN ANY CASE BE LIABLE FOR SPECIAL, INCIDENTAL, CONSEQUENTIAL, INDIRECT, OR OTHER SIMILAR DAMAGES ARISING FROM ANY BREACH OF THESE WARRANTIES EVEN IF THE WAITE GROUP, INC. OR ITS AGENT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

THE LIABILITY FOR DAMAGES OF THE WAITE GROUP, INC. AND THE AUTHORS OF THE PROGRAMS UNDER THIS AGREEMENT SHALL IN NO EVENT EXCEED THE PURCHASE PRICE PAID.

COMPLETE AGREEMENT

This Agreement constitutes the complete agreement between The Waite Group, Inc. and the authors of the programs, and you, the purchaser.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above exclusions or limitations may not apply to you. This limited warranty gives you specific legal rights; you may have others, which vary from state to state.

SATISFACTION REPORT CARD

Please fill out this card if you wish to know of future updates to *Fractal Creations, Second Edition,* or to receive our catalog.

Company Name: ____________________

Division: ____________________ Mail Stop: ____________________

Last Name: ____________________ First Name: ____________________ Middle Initial: ____________________

Street Address: ____________________

City: ____________________ State: ____________________ Zip: ____________________

Daytime Telephone: (   ) ____________________

Date product was acquired: Month ______ Day ______ Year ______ Your Occupation: ____________________

Overall, how would you rate *Fractal Creations, Second Edition?*

☐ Excellent ☐ Very Good ☐ Good
☐ Fair ☐ Below Average ☐ Poor

What did you like MOST about this product? ____________________

What did you like LEAST about this product? ____________________

How do you like the Fractint program? ____________________

How do you use this book (education, diversion, relaxation...)? ____________________

How did you find the pace of this book? ____________________

Please describe any problems you may have encountered with installing or using the programs: ____________________

What is your level of computer expertise?

☐ New ☐ Dabbler ☐ Hacker
☐ Power User ☐ Programmer ☐ Experienced Professional

Is there any program or subject you would like to see The Waite Group cover in a similar approach? ____________________

Please describe your computer hardware:

Computer ____________ Hard disk ____________
5.25" disk drives ____________ 3.5" disk drives ____________
Video card ____________ Monitor ____________
Printer ____________ Peripherals ____________

Where did you buy this book?

☐ Bookstore name: ____________________
☐ Discount store name: ____________________
☐ Computer store name: ____________________
☐ Catalog name: ____________________
☐ Direct from WGP ☐ Other ____________________

What price did you pay for this book? ____________________

What influenced your purchase of this book?

☐ Recommendation ☐ Advertisement
☐ Magazine review ☐ Store display
☐ Mailing ☐ Book's format
☐ Reputation of The Waite Group ☐ Topic

How many computer books do you buy each year? ____________

How many other Waite Group books do you own? ____________

What is your favorite Waite Group book? ____________________

Do you have the first edition of this book? ____________

Additional comments? ____________________

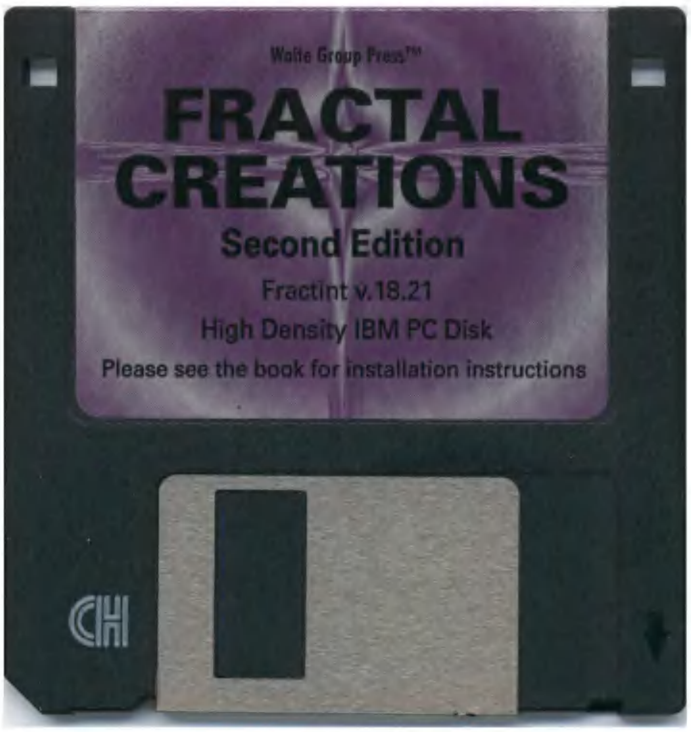
☐ **Check here for a free Waite Group Press™ catalog**

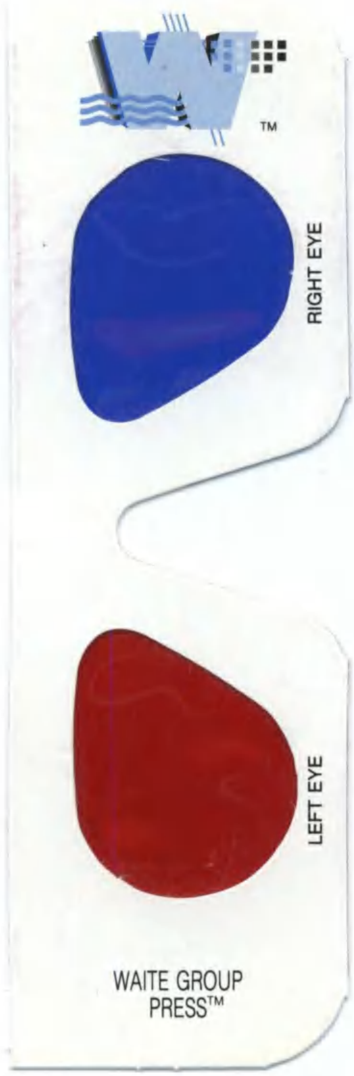
Send to Waite Group Press™, 200 Tamal Plaza, Corte Madera, CA 94925

PLACE
STAMP
HERE

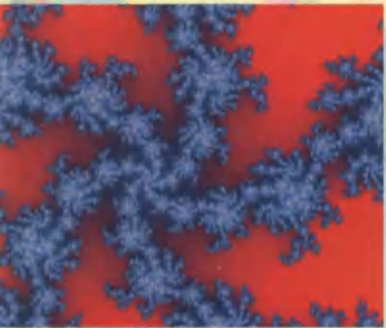
Waite Group Press, Inc.
Attention: *Fractal Creations, Second Edition*
200 Tamal Plaza, Suite 101
Corte Madera, CA 94925

FOLD HERE


Waite Group Press™
FRACTAL CREATIONS
Second Edition
Fractint v.18.21
High Density IBM PC Disk
Please see the book for installation instructions


TM
RIGHT EYE
LEFT EYE
WAITE GROUP PRESS™

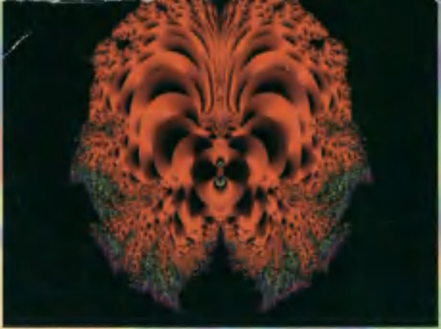
Fractals aren't just op art or psychedelic lava lights. You'll learn their history and scientific uses such as studying weather, geographical measurement, nature simulations, and cinematic special effects. And then there are the everyday benefits of fractals: learning about color theory, math, and geometry.



A new view of nature might be the last thing you expect from a PC, but that's what you get with Fdesign. This unique program is used for generating and modifying Iterated Fractal Systems (IFS) fractals, which reproduce the geometry of nature with patterns of mountains, veins, and leaves.

Fractal Creations Second Edition offers a myriad of options for creating and transforming fractals. Use the zoom box to see a fractal's microscopic structure, or render your vivid creations in 3-D. Fractal recipes show how to cook up a sea slug, octopus, or evil frog, and then number crunch a batch of gingerbread men. Adjust Fractint's huge range by tweaking corner parameters, altering color palettes, resizing, or animating the fractal. Spice up images with 3-D stereo vision: slip on the provided red/blue glasses and watch the images jump off the screen.

Fractint 18 now offers more fractal types than any other program (99, to be exact), including some unavailable anywhere else. New features include: 25 new fractals, new variations of original fractals, virtually unlimited graphics resolution, and hyper-text-style on-line help. And Fractint is still the world's fastest fractal generator.



THE WAITE GROUP®

FRACTAL CREATIONS

Second Edition

FRACTAL CREATIONS
Second Edition Includes:

- Fractint V18
- Accessible, hands-on tutorials and manual
- Red/Blue 3-D glasses
- Reference section with algorithms
- Modifiable Fractint source code
- CD with 1,800 GIF and .PAR Fractals files
- 16 pages of color plates
- Source code for DOS, Unix

WHAT YOU NEED:

- 512K of memory, IBM compatible PC, hard disk (recommended)

Fractals are intricate patterns of color and texture generated by deceptively simple mathematical formulas. With **Fractal Creations Second Edition** and a PC you can easily create these phenomenal images. But this isn't just a book—it's a super-creative book/disk/CD-ROM bundle: you'll get the latest version of the world's most eminent fractal generator, Fractint version 18; plus the powerful Fdesign, a program which draws fractals that mimic nature such as rain, clouds or waves. The awesome storage capacity of the CD allows us to include over 1,800 beautiful GIF fractal images and all the Fractint source code.

WAITE GROUP PRESS™
200 Tamal Plaza
Corte Madera, CA 94925

$39.95 USA
$48.95 CANADA

SHELVING: Computer Graphics
ISBN 1-878739-34-4
53495>
EAN
9 781878 739346