7-16-1985

A Recursive Fractal Design Generator for Dimensions Zero to Two Implemented within a Two Dimensional Core Graphics Package

Elizabeth Lehmann

Rochester Institute of Technology
School of Computer Science and Technology

A Recursive Fractal Design Generator for Dimensions Zero to Two Implemented within a Two Dimensional Core Graphics Package

by

Elizabeth A. Lehmann

A thesis, submitted to
The Faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree
Master of Science in Computer Science

Approved by:

_____
Professor Peter G. Anderson

_____
Professor Chris Comte

_____
Professor Guy Johnson

July 16, 1985

Title of Thesis: A Recursive Fractal Design Generator for Dimensions Zero to Two Implemented within a Two Dimensional Core Graphics Package.

Acknowledgements

The author wishes to thank all those individuals who helped make this thesis project possible. I especially appreciate the assistance and encouragement of my husband Jim Sutherland. I also wish to thank Mike Bentley and Jeff Duntemann for their assistance on various microcomputer graphics boards, and Dr. Peter Anderson for explaining the mathematics behind fractals.

In addition, I would like to thank my Xerox manager, Jim Miller, and the Xerox SOGP personnel who made all of this work possible.

1.2. Abstract.

This thesis incorporates the technique developed by Benoit Mandelbrot to describe recursive fractals into an interactive graphics package based on the Core Graphics System (Core) produced by an ACM SIGGRAPH Committee (1977, 1979). The graphics package encompasses simple standard geometric shapes as well as the recursive fractals.

To draw those fractals requires knowing both the basic shape or generator, and the points of recursion. These two pieces are acquired through the using of two windows which allow the generator and the points of recursion to be built. Once built, the fractal recursion level is chosen interactively on the main drawing.

The conclusion I reached as a result of this project is that it is possible to integrate fractals in a systematic way into a standard graphics package, much as rectangles and circles are today in most graphics systems.

1.3. Key Words and Phrases.

Core Graphics System, Fractals, Hausdorff Besicovitch dimension, Computer Graphics, Graphics on Microcomputers.

1.4. Computing Review Subject Codes.

I.3.4. Graphics Utilities - application package

I.3.5. Computation Geometry and Object Modeling - geometric algorithms, languages, and systems

1.5. Table of Contents.

2. Proposal.

The original proposal of this thesis described the functions to be performed by the basic graphics package, and the fractal generator. The complete proposal is on file with the Computer Science Department. What follows is the discription of changes to the project from the originial functional specification.

In the basic graphics package a number of minor modifications and one major change was made.
One of the minor changes included reordering the function, or "F", keys to make the numbers follow more closely the number of sides of the shape requested, such as F3 for the triangle and F4 for the rectangle. A second minor modification was the number of sides for the regular polygon followed the point selection rather than before. In addition, an added feature was that the program shows the coordinates of the mouse-selected points, truncated to the second decimal place.

Another minor difference from the original specification was that only four colors could be implemented on the microcomputer even though all eight colors were shown in the menu. The four were white, black, magenta (red, yellow also invoked this), and cyan (green and blue invoked this one, too). All of the line options except the erase line option were added. The erase was not included since the black color could be used instead.

The one major change to the specifications of the main drawing was that the ESC (escape) key, which allowed "escaping" from an undesired function, was only partially incorporated. This was mainly due to the fact that scanf (a standard C function) had trouble with the ESC key when it was expecting numeric input. Because the escape feature was not directly associated with the point of this thesis, this limitation was not corrected by rewriting the scanf function. Instead, the escape option was only allowed while drawing the figure, and at some places of the fractal generator.

Like the main drawing, the fractal generator had a few minor changes and one large modification. One of the minor differences was that the grid density was made into a separate screen after the selection of points. In addition, there were minor changes to the keys used in both the first and second windows. In the first fractal window, the selection of the fractal dimension number was changed to the F key since the D key was used for the delete option. One change in the second window was made to make it consistent with the first window: the eliminate option, E, was changed to the delete, D. Also, the random sub-generators key was changed to P for partial while the total random key, T, was retained.

Another minor change gave more flexibility. The duplication of the fractal to other points did not have to happen immediately after the first fractal was made.

Instead, the main menu gave the option to duplicate any of the fractals on the main drawing with the F1Ø key.

The one major change was that the modify option in the first window was not incorporated because of difficulties in calculating a good change to the fractal generator. Using simple figures, various modifications to them resulted in nonlinear changes in the fractal dimension. This is discussed in depth in appendix B of this thesis.

Outside of the drawing program, there was one major hardware issue. It was the need for high resolution for the fractals at the expense of limiting the colors to three (plus background) or just one color. This color restriction was inversely tied to the resolution of the microcomputer used; i.e., the more colors, the lower the resolution. The resolution at the three-color level (32Ø x 2ØØ pixels) turned out to be sufficient for this project. When a fractal was drawn at recursion levels that went beyond this resolution, the variable memory was usually exceeded. This memory limitation was due to the C package used which restricted variable memory space to 64k. Consequently, any further attempts to increase the resolution and the color by obtaining a high resolution graphics board were dropped.

3. Introduction and Background.

Recently, the computer graphics profession has become interested in the graphics potentials opened up by fractals--by their realistic representations of mountains and clouds, and by their intricate abstract designs. But what are fractals? How do they differ from standard geometry? How can a fractal be created?

The first part of this section will cover the basic fractal concepts. That introduction will be followed by a discussion of the different types of fractals, recent work in those areas, and reasons for selecting the recursive fractals for further development. The third part will cover in depth the mathematics of recursive fractals. Subsequently, Mandelbrot's visual algorithm for recursive fractals and the program's implementation of that algorithm will be explained. The last part will discuss the goal of this thesis project.

Fractal Concepts

The term "fractals" was coined by Benoit Mandelbrot from the Latin adjective fractus meaning "irregular" or "fragmented."(1) The term refers to a specific class of shapes outside of the standard Euclidean geometry of points, lines, circles, squares, planes, and other regular forms.

(1) The Fractal Geometry of Nature by Benoit Mandelbrot, W. H. Freeman and Company, 1982, p.4.

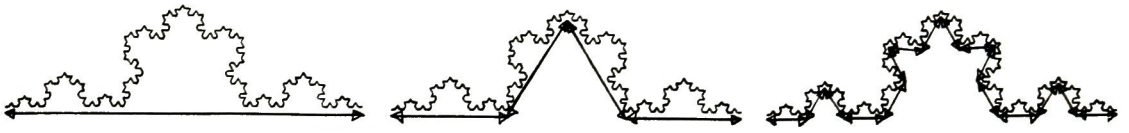
One feature of Euclidean geometry and its smooth curves is that the smaller the ruler used to measure a particular curve, the more accurate the measurement becomes. Finally, as the ruler becomes infinitely small, a well defined limit is reached: the length of the curve.

However, this is not the case with natural objects, nor with fractals. The classical example used to illustrate this fact is "how long is the coast of Britain?" Basically, this example considers the effect of different size rulers on the resulting measurements. For instance, if a kilometer long (Ø.6214 mile) ruler is used to measure the coast of Britain, the measurement would be one length. Then if the ruler is reduced in size to a meter in length (39.37 inches) the resulting picture of the coastline would be more accurate, but the total distance would be much greater than the measurement by kilometer-long ruler. Reducing the size of the ruler to one millimeter (Ø.Ø3937 inches) would result in an even more accurate picture, but once again the coast distance would grow still longer. In fact, there is no limit to that length; with an infinitely small ruler the coast would measure a length of infinity.(2)

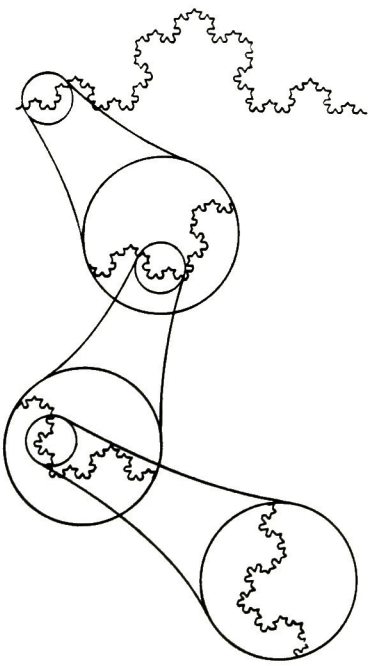
This characteristic of the natural world is the same for fractals. As with the coast of Britain example, the

(2) ibid. pp.25-26.

smaller the ruler used to measure a fractal, the larger the resulting measurement of the curve. The curve's length approaches infinity as the ruler becomes infinitely small.



From the above diagram, a mistaken idea could be developed that if the ruler became sufficiently small the shape's true length could be measured. However, the one characteristic of fractals is that the closer one looks, the more fractal that can be found. The picture below illustrates this concept by magnifying part of the original shape. This process of magnification, shown only to three levels, could go on indefinitely, producing a curve with an infinite length. The only reason the fractal looks measureable is due to the printing resolution on the page.



Types of fractals

Fractals can be classified in several different ways. One method is to categorize them by the process that generates them. Though this technique is not exhaustive, it has the advantage of highlighting the underlying computing process. According to this method, the three major types of fractals are: 1) recursive, 2) random, and 3) iterative. Each of these types are discussed in depth below, together with current work in the area.

1) Recursive.

These fractals have a computing characteristic of starting with a shape producing sequence that could be recursively applied as many times as necessary to an initial form. The result was a complex regular fractal which was defined by the shape producing sequence employed.

To date, a few computer graphic artists have employed recursive fractals as graphic designs by apparently programming each shape producing sequence separately, and keeping a library of them for later use.(3) I have grouped other references to recursive fractals into two categories, fixed and flexible, which are examined below.

Fixed fractals were those with a permanent algorithm, though changing input variables would modify some of the results. A Peano curve article, and a BASIC article using string manipulation were both in this category. The article

(3) "Geometrical Forms know as Fractals find Sense in Chaos" by Jeanne McDermott, Smithsonian, December 1983, Vol.12, No.9, pp.115-116.

on the Peano curve explained how to generate grey scales to convert a continuous tone picture to a halftone one.(4) The other article described the use of BASIC to emulate recursion through manipulating strings; it included "hard-coding" of the various shapes, such as the quadratic Koch curve.(5)

Flexible fractals allowed input of the algorithm as well as any input of the variables. The single article in this category used BASIC's string manipulating capabilities to attain this degree of flexibility. Unlike the previously mentioned BASIC article, it allowed different Peano shapes with fixed 90^{o} angles to be drawn based on a user-supplied input of right and left turns.(6) This particular method of generating fractals could be easily modified to employ a recursive language which could utilize different angles and line lengths. Though it was a more analytic method than Mandelbrot's visual algorithm, described below, the input string technique might be worth pursuing in another thesis.

For my thesis, I decided to use recursive fractals because they allowed the most flexibility to create new designs. In addition, recursive fractals have a straightforward method to calculate their fractal dimension, and a visual system, developed by Benoit Mandelbrot, for

(4) "Using Peano Curves for Bilevel Display of Continuous-Tone Images" by Ian H. Witten and Radford M. Neal, IEEE Computer Graphics and Applications, May 1982, Vol.2, No.3, pp.47-52.
(5) "Recursive Curves" by Eui In Lee, PC Tech Journal, April 1985, Vol.3, No.4, pp.171-174.
(6) "A Tiger Meets a Dragon" by Dan Rollins, Byte, December 1983, Vol.8, No.12, pp.457-478.

building the shape producing sequence. Both of these are expanded upon in the following sections.

2) Random.

Random fractals often used a recursive "base" but unlike the previous category randomness was incorporated into the placement of points generated. The result was often a natural looking form, such as a mountain, which varied in shape each time the fractal was drawn.

The most common illustration of this type of fractal was the mountain made from recursively generated distorted triangles.(7) Other more advanced three dimensional fractals have been generated using Gaussian distributions and Brownian conversion functions.(8) Textures have also been generated by displacing the angle of reflecting light instead of the placement of points to make a realistic surface.(9)

Random fractals were not incorporated into this thesis in order to limit the scope of the project. In addition, there exist problems in the calculation of the fractal dimension of these forms, and in methods for reasonably representing an algorithm to produce them. Adding random fractals to the existing program might be worth pursuing in another thesis.

(7) "3D Fractals" by Michiel van de Panne, Creative Computing, July 1985, Vol.11, No.2, pp.78-82.
(8) The Fractal Geometry of Nature by Benoit Mandelbrot, W. H. Freeman and Company, 1982, pp.232-271,C10-C12.
(9) "Using Stochastic Modeling for Texture Generation" by Shinichiro Haruyama and Brian A. Barsky, IEEE Computer Graphics and Applications, March 1984, Vol.4, No.3, pp.7-19.

3) Iterative.

Iterative fractals were based on the concept that points in the x-y plane were passed through a function many times until the results approached infinity (point of repulsion) or became stable near the original point (point of attraction). Interesting pictures could be seen when the speed of repulsion was color coded. The result was often a complex regular shape which is characteristic of the formula used.

These fractals were more commonly associated with chaotic or turbulent behavior of functions and natural phenonmenon. One article about "chaos" used iterative calculations on several fixed equations, such as c*sin(x+iy), to explore their properties, producing several impressive pictures in the process.(1Ø) A second article took iterative fractals a step further by exploring three dimensional shapes, and using a surface finding algorithm to speed the drawing process.(11)

There was a major problem with iterative fractals as a method to make fractal designs. Because each point in the total picture had to be repeatedly calculated, the designs were extraordinarily slow to generate. Iterative fractals were also tied to a particular equation and could not be

(1Ø) "Escape into Chaos" by Ivars Peterson, Science News, May 26, 1984, Vol.125, p.329.

(11) "Generation and Display of Geometric Fractals in 3-D" by Alan Norton, SIGGRAPH '82 Conference Proceedings. Ninth Annual Conference on Computer Graphics and Interactive Techniques, July 1982, Vol.16, No.3, pp. 61-67.

easily changed. Consequently, this type of fractal was not incorporated in this project.

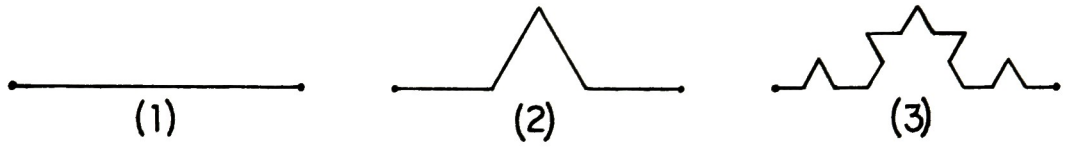
Calculating recursive fractals

All fractals, whether recursive, random, or iterative, constitute a set of equations or designs that have a Hausdorff Besicovitch dimension that strictly exceeds the topological dimension. The topological dimension is the standard one from Euclidean geometry: zero for points, one for lines, and two for planes. The Hausdorff or fractal dimension is the number that measures the "dustiness" of fragmented lines (numbers between zero and one) or the plane-filling characteristics of a line (numbers between one and two). For standard shapes, like circles and squares, the fractal dimension is the same as topological dimension.(12)

But how is the Hausdorff Besicovitch or fractal dimension of a fractal calculated? As seen in the coast of Britain example mentioned in the first section, the smaller the ruler used, the larger the resulting measurement of the fractal curve. The curve's length approaches infinity as the ruler becomes infinitely small. Rather than try to measure the curve directly, the "ratio" of one length with one ruler to a second length and its ruler is used. That "ratio" is the Haudorff dimension.

(12) The Fractal Geometry of Nature by Benoit Mandelbrot, W. H. Freeman and Company, 1982, p.15.

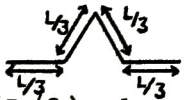
A good example of using this "ratio" is the Koch snowflake generator. The snowflake starts with a straight line (1) which then adds a triangle to its center (2). As in the first line, each of the sub-lines adds its own triangle (3). This process of adding triangles can go on indefinitely to make a fractal.



For the rest of this discussion, the second level of the snowflake generator will be employed, though higher levels would work just as well.

If a ruler of length L is chosen which is equal to the snowflakes's length, then the measure of it would be L.

Subsequently, if that ruler is made one-third as long (L/3), the generator would require 4 of those smaller rulers.



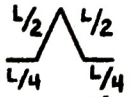
In other words, the length would go from L to 4*(L/3) when the size of the ruler is reduced by one-third. If a higher level of the snowflake was used, and the size of the ruler was reduced further by another third, L/9, the length of the snowflake would grow even larger, 16*(L/9).

To understand this geometric progression, the fractal dimension D is used to make the first measure equal to the second: $L^D = 4 * (L/3)^D$. When this equation is solved for D, it results in: $D = (\log 4)/(\log 3) = 1.262$. Because

this number is larger than the topological dimension of 1, which normal Euclidean geometry would dictate, the snowflake is a fractal.[13]

Benoit Mandelbrot in his book The Fractal Geometry of Nature used the regularity of recursive fractals to calculate the fractal dimension of the figures he made. The principle behind that calculation follows.

For recursive fractals with even length sides, this dimension can be quickly determined by computing the log of the number of sides and dividing it by the log of the inverse of the fractional length of a side. For the Koch snowflake, the number of sides is 4, and the fractional length of a side is 1/3 which has an inverse of 3. Consequently, the fractal dimension D is log 4 divided by log 3, the same as calculated above.

For fractals with uneven sides, this quick algorithm breaks down. In this case, each side must be added separately to obtain the total dimension. For instance, if the Koch snowflake had a longer point and shorter sides, 

, the calculation would be very different from the one above. The two ends that are one-quarter the length of the original line, and the point that is made up of two sides of one half the length result in the following equation:

$$L^D = (2 * (L/4)^D) + (2 * (L/2)^D)$$

(13) ibid. pp.36-38.

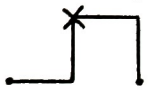
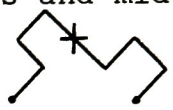
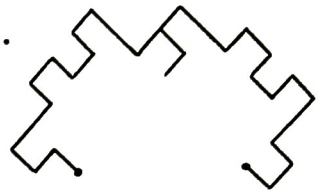
which gives a fractal dimension D of 1.45. If the quick algorithm had been used of log 6 divided by log 4, the dimension would have been incorrectly found to be 1.29.

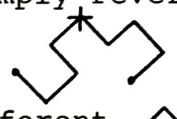
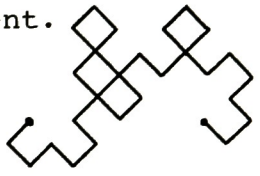
For this thesis, when the fractal dimension was calculated, Mandelbrot's quick agorithm was used to obtain an approximate number. Then, employing the full equation, D was repeatedly calculated and adjusted until the dimension was within a tolerance of Ø.ØØ1. For regular fractals such as Peano curves, the full equation calculation simply confirmed the original number obtained by the quick method. A sample calculation applying this process is given in Appendix A.

Visual algorithm for recursive fractals

To make other recursive fractals besides the Koch snowflake, Mandelbrot in his book The Fractal Geometry of Nature developed a visual algorithm. This algorithm combined an initiator (such as the straight line of the snowflake) and a generator (such as the triangle in the middle of the line) to form shapes that were piecewise similar to the overall shape, and thus were self-similar. This method of using a visual algorithm to generate fractals made them easy to understand. Though not all fractals could be invented in this manner, many rich and complex patterns could be generated.

Mandelbrot's algorithm for fractals involved the use of straight lines of known lengths (the initiator) and points on those lines where recursion took place (the generator).

For example, the Peano C curve starts with a step shape:  There is only one point of recursion, the corner of the step:  Using the two end points and middle point, recursion of the curve can take place:  When this recursion is carried to the fourth level, the C shape is readily apparent. 

A second part of this visual algorithm allows changes to the second recursion level of the basic fractal shape. For example, using the same step shape employed for the C curve, a dragon curve can be made by simply reversing part of curve starting at that second level:  The resulting fractal appears quite different. 

To implement Mandelbrot's visual algorithm in this thesis, I used two windows with grids. The first window allowed building of the initiator and the generator, while the second window permitted variations for the generator, such as inversion. An added feature in this second window was an option to have a random orientation for the generator. This orientation differed from a random displacement of the generator endpoints that a random fractal would have required.

Utilizing a grid for placing the fractal initiator and generator was a deviation from Mandelbrot's algorithm, but

it served a two-fold purpose. First, it simplified placement of points and lines for the user, rather than employing absolute line lengths as Mandelbrot had. Second, the grid was used in the calculation of the fractal dimension.

Purpose

My goal in this thesis has been to employ Mandelbrot's visual algorithm method to expand the category of flexible recursive fractals and thereby make them accessible to the graphic artist. To reach that goal, I hope to prove that a class of fractals (recursive fractals) can be systematically incorporated into the Core Graphics System, a standard graphics system.

4. Functional Specification.

This project has two major functional areas. The first is the basic graphics package, while the second is the fractal generator. The purpose of the former is to give a background in which the latter could be incorporated. As seen from the user, a description of each follows.

The basic graphics package allows a user to create points, lines, and standard shapes. Those shapes include the rectangle, triangle, circle, and regular polygons. The selection of a particular shape is by keyboard function keys, i.e., F1 for a point, F2 for a line, F3 for a triangle, F4, rectangle, F5, a regular polygon, and F6 circle. This is followed by the point selection to define the shape, either by typing in number coordinates or using the mouse to select points on the screen. An added feature with the mouse selection of points, is that the program shows the coordinates of those points, truncated to the second decimal place. If the last point selection was done by using the mouse, the program will wait for the right button before closing that option. The reason for this is to allow the user to inspect the mouse-generated coordinates if needed. For the regular polygon, F5, the number of sides desired must be typed in after the point selection. The minimum is one side for a polygon; there is no maximum except for memory limitations.

Also, the user can select color and line quality by typing in the desired option. For colors, the keys are M for magenta, C for cyan, Y for yellow, R for red, B for blue, G for green, W for white, and K for black. Though the menu shows all the colors, only four are implemented on the microcomputer. The four are white, black, magenta (red and yellow also invoke this), and cyan (green and blue generate this one). The line styles available are solid lines, S, dashed lines, D, and dotted lines, T. An erase option is not included since the black color can be used instead.

Clipping to one shape is supported by using the F7 key, i.e., a rectangle can be clipped to one circle, but not to two circles. The clipping works by "flashing" each object on the main drawing until one is selected for clipping or none. All subsequent shapes including fractals will be clipped that shape. Some shapes can not be used for clipping. They are points, lines, fractals, and any shape less than 2 lines. The reason is that the clipping algorithm finds the insides of a shape, and needs at least 2 sides to do that. In general, fractals are too convoluted to find the insides.

All of the above commands allow a user to create a main drawing with standard graphics tools. To obtain fractals, the second major functional area within this project, two keys are used. To make the fractal generator, F9 is pressed followed by other keys; to make a duplicate of a previous fractal mold, F1Ø is used. A description of how to make the

fractal from the user's perspective follows. After that, the duplicate fractal function is explained.

To make a new fractal, there are five steps. The first is to select the fractal endpoints on the main drawing. The second is to choose the grid density. The third and fourth steps are to make the fractal in the first and second windows, respectively. And the final step is to chose the level of recursion for the fractal on the main drawing. Each of these steps are covered in more depth below.

Once the F9 key is chosen to make a new fractal, two end points are selected on the main drawing, using either typed-in coordinates or mouse-selected points. Then the fractal generator goes to the second step which makes the first of two small "windows" on the screen. On the window are two end points of the future fractal, symbolically indicated by two red X's. The point on the left is always at grid point Ø,Ø. The second is at the <grid #>,Ø. The first window will always have the default grid of 4. At this stage, that grid density can be interactively changed; but once chosen, it would remain for the entire fractal session. The reason for the variable spacing is to allow for different fractals. The minimum grid is 1 and the maximum is 15, due to the resolution of the screen.

After the grid selection, the menu for the first window appears, the third step. The choices are to add, A, or delete, D, pairs of points on this first window to make a generator. The point selection could be made by typing in

point coordinates or by using a mouse to pick them. Both of these options continue until a "y" is entered at the "done" question, or both mouse buttons are pressed. As the fractal generator is built, the lines are shown in red in the first window. To select recursion points, type R. Now any adding or deleting of points would be show in blue, indicating recursion. Pressing R again would toggle back to the fractal generator mode. To find the fractal dimension of the generator, the F key is selected. This dimension is based on the recursion points alone, not the generator points, and does not take overlapping into account, which technically voids the dimension calculated. The end of the first window and the start of the second one is indicated by hitting the F1Ø key.

The completed generator and recursion points from the first window are then used to make a second level fractal on the second window, step number four in making a fractal. This second level is necessary if one wanted to reverse, R, invert, I, add, A, or delete, D, some of the sub-generators to make different fractals. To do this, the choice is made by pressing the appropriate key followed by the selection of those sub-generators by typing in or by picking with the mouse the desired end points. Random orientation of the sub-generators, P, or of the total figure, T, can also be selected at this time too. The randomness is toggled off by selecting the same key again and the same points.

At this point, if the first generator is unsatisfactory, the user can select the F9 to return to the first window to try again, essentially taking the user back to step number three. Any additions or deletions of the recursion points from the second window would be carried back to the first. Again, the user would have to go to the second level window, step four, by selecting F1Ø, before reaching the fifth step.

On the other hand, if the fractal is satisfactory in the second window, pressing the F1Ø key a second time will erase both of the windows, and take the user to the last menu of the fractal generator, the fifth step. At this time the level of recursion desired for the fractal at the original two end points (selected in step one) is entered. This level is chosen by simply typing in a positive integer. The fractal is then executed on the main drawing. When the fractal level is acceptable, a "y" or two mouse buttons at the "done?" question will end the session. If that particular level is not suitable, a carriage return or right mouse button at the "done?" will allow you to try another level. If none of the levels work, a level of -1 will take you back to the main menu and all work for that session is lost.

That concludes the five steps to make a new fractal. If a duplicate fractal is desired the user must go through only three steps, after pressing the F1Ø key at the main menu. The first step is to select which fractal to copy.

Then, the fractal endpoints are selected on the main drawing, just like when a new fractal is made. The third step is to choose the level of recursion for the fractal on the main drawing, as was done in the fifth step of a new fractal. As you can see, using the duplicate option essentially saves redoing steps two, three, and four of making a new fractal. Each of the steps for making a duplicate fractal is covered in more depth below.

The purpose of the first step is to select which fractal "mold" to use. This can be done by either entering the mold number or using the existing fractals to select one. For the former method, an N or a left mouse button is pressed and then a number is entered. This form of selection is only for fractal molds that were retrieved from a file and have no image hooked to them. To use the existing fractal pictures for selection, a method similar to the clipping options is employed. Each of the fractals are "flashed" in reverse order of creation, since the last one made is most likely to be chosen. To select one, a user enters a Y or both mouse buttons at the "flashing" fractal desired, or if the one "flashing" is not wanted, a carriage return or right mouse button is pressed. If none of the fractals is chosen or the mold number is not valid, the user is returned to the main menu.

Once the mold is chosen, the second step of picking end points is reached. This part is exactly like the first

step in making a new fractal. Two points are indicated through entering x-y coordinates or by mouse selection.

The third and last step of making a duplicate fractal is choosing the level of recursion. Again this is exactly like making a new fractal. A recursion level is interactively chosen until one is picked, a positive integer, or none, a level of -1.

Saving or retrieving previously saved files is supported through the file option, F8, on the main menu. A single fractal mold (save only), all of the fractal molds, just the main picture without the molds, or the picture with the fractal molds can be chosen. After the type of operation is picked, either save or retrieval, the user is prompted for the drive and the name of the file before executing the function. That file name is given a suffix by the program depending if the file is for a fractal, FRC, or picture, PIC. For example, a file name of "dragon1" would have a full name of DRAGON1.FRC for fractal molds, and DRAGON1.PIC for pictures. If a picture with fractal molds is selected (the BOTH option), both file types with the same name will be created or read from. When creating files, any previously existing files with that same name will be overwritten.

As a convenience for the user, a limited option to escape during a selection has been incorporated. While a line, triangle, square, polygon, or circle is being drawn on the screen, the escape key can be pressed to stop, which

allows partial shapes to be made easily. Unfortunately, using the escape key in other places, such as the point selections for the geometric shapes, can result in problems.

In addition, there are three points to "escape" the session both for making and duplicating fractals. For new fractals, the first point is at the point selection menu, step one, where all zeros or carriage returns will allow the user to return to the main menu. The second place is at both of the windows, steps three and four, where the escape key is a valid option to get out doing a fractal entirely. At the recursion level menu, step five, getting out is by using -1 instead of the escape key.

As mentioned above, the duplicating fractal option also allows three points to escape. The first is at the mold selection, step one. By using the carriage return to tab through all of the "flashing" fractals, the user is informed that none was selected and is returned to the main menu. Both the second and third escapes are the same as the first and last ones, respectively, for new fractals.

5. Architectural Design.

The hardware that I used was a Zenith 151 PC computer which was an IBM PC compatible microcomputer. The computer was based on the Intel 8088 16-bit microprocessor, and used the MS-DOS Version 2.0 operating system. The Intel 8087 arithmetic coprocessor chip was installed on this machine, though it was not necessay for the operation of any part of this thesis. The advantage of the math chip was that the program ran significantly faster (about twice as fast) and the compiled code was smaller (by about 4K).

In addition, I used the Microsoft Mouse, Bus Version, with the MOUSE.COM program to install it. Once installed, the mouse interrupts within my program both sent and received information. The mouse procedures that made the interrupts were written by me in assembly language ASM88 that came with the DeSmet C. Normally, the MOUSE.LIB which came with the Microsoft Mouse would be linked with the appropriate program. However, the object code of Desmet C was not compatible unless an extra converting library was used which I did not have.

For prints of the screen, the Okimate 20, which is a thermal color printer, was employed. The screen dump program, CSP.COM, ran in the background much like the mouse program until the PrtSc key was pressed.

The software that I used was DeSmet C which supported floats, structures, and over 120 C procedures. Except for

the driver section, the entire Core Graphics module that had been written in the Unix environment was compiled without changes. In addition, this version of C included a floating point math library which contained sin, cos, and log functions needed for this project. Depending on whether the computer had a math chip or not, different math libraries were used to complile the program.

As part of the C package, an assembly language interface, ASM88 mentioned above, was supported. Besides the mouse procedures, the assembly language was use to write two routines for the keyboard and the screen. Moreover, a library of PC dependent assembly procedures were supplied in DeSmet C, of which only a subset was used in the final project.

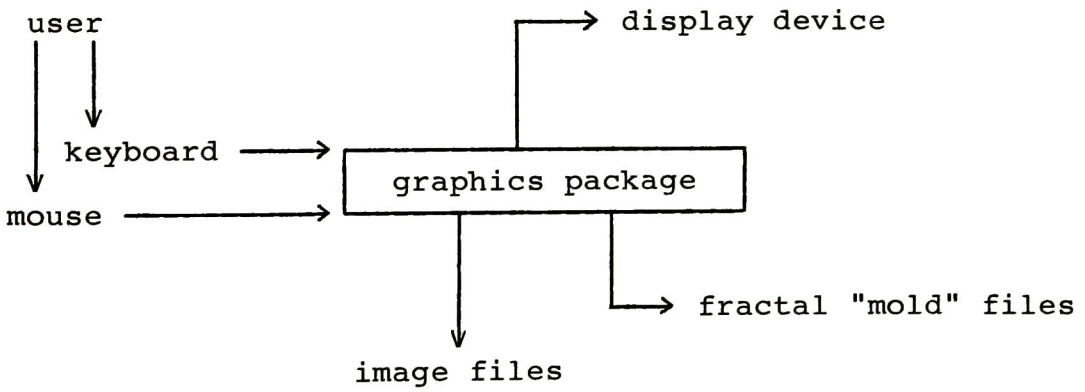
6. Interface Specification.

6.1. External Interfaces.

The inputs into the graphics system and the fractal generator were through both the keyboard and a mouse. The specific keys used are mentioned in section 4, Functional Specification.

The outputs from the system will be the image on the screen and files to store the main image and the fractal "molds." These are optional files that the user can make to save both the image on the screen and the fractal "mold" for later use. These files can be read back into the graphic package for further modification.

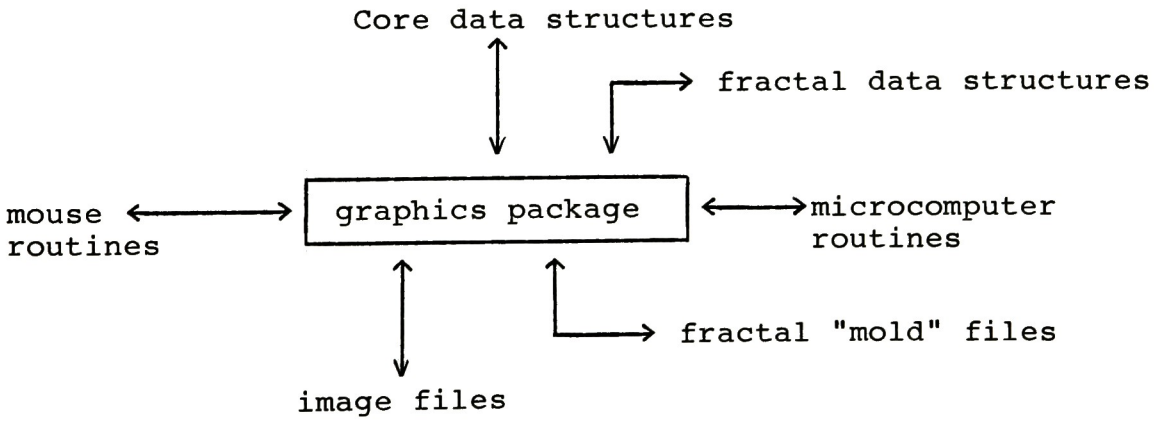
The relationship between the program and the external interfaces are show below:



6.2. Internal Interfaces.

The program interfaces with a number of structures, files, and routines internally. The two major structures are within the Core Graphics module and the fractal module, and are not connected to each other. The files, though they are initiated from the controlling module, are also separated into Core or fractal.

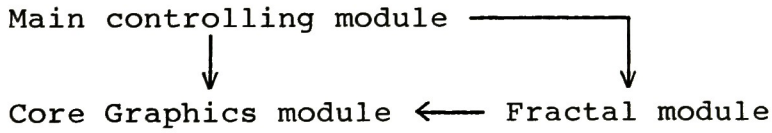
The two "routines," mentioned in the illustration below, are actually assembly language procedures that are hardware dependent. The mouse routines are specific to the Microsoft mouse, though similar ones could be written for other brands. The microcomputer routines are specific to the keyboard, display memory, and operating system of the Zenith 151 microcomputer.



7. Module Designs.

7.1. Processor Modules.

There are three main modules to this project. The first is the section based on the Core Graphics System which was written during the ICSS77Ø computer graphics class. The second module is the fractal section which uses the Core Graphics to draw the fractals. The third part is the main controlling module which invokes the fractal module and also uses the Core Graphics procedures to draw basic geometric shapes on the main drawing. The following picture shows their relationships:



Within each module, there are major sections which are in turn supported by a number of procedures. An outline of each of the modules, the sections, and supporting procedures follows.

Main controlling module: This module is located in the master.c file and contains all the geometric procedures.

1. controlling procedures: Controls options in main menu and initializes both Core and fractal modules.
   - main
   - initializeverything
   - initializemouse
   - main_menu

2. geometric shapes: The six standard shapes are supported.
   - pointdata
   - linedata
   - triangledata

drawtriangle
basic_triangle
squaredata
drawbox
basic_square
polygondata
drawpolygon
basic_polygon
circledata
drawcircle
basic_circle

3. file routines: Saving and retrieving files of both Core and fractal structures.
dofiles
getname
savefile
retrievefile
adjustnum

4. remaining routines: Miscellaneous routines.
clipdata (other clipping routines are in core.c)
(all fractal routines are in fractal.c)
repeatfractal (selects which fractal # to draw)
refreshscreen

5. utility procedures: Used by many of the routines above.
getxandy
getmouse
getcr
getfracnum

Fractal module: This module is located in the fractal.c file and contains all the fractal procedures.

1. callable routines: All of these are called by main.c to do fractals.
initializefractals
makefractal
findfractal
duplicatefractal
saveallfractals
savefractal
retrievefractal

2. opening & closing routines: Used by makefractal and duplicatefractal for start-up and closing.
selectpoints
startfracseg
deletefracseg
getgrid
replacescreen
resetcolor

3. level 1 routines: Used by makefractal (first window).

```
first_menu
first_options
drawfirstfractal
addpoints
deletepoints
fractalnumber
calculatefractal
```

4. level 2 routines: Used by makefractal (second window).

```
second_menu
second_options
drawsecondfractal
addrecur
deleterecur
invertrecur
reverserecur
randompoint
totalrandom
```

5. level 3 routines: Used by makefractal and duplicatefractal for last step, selecting the recursion level.

```
third_menu
third_options
drawthirdfractal
```

6. utility routines: Use by procedures above.

```
setup_outline
drawgrid
drawX
drawrecurpoints
bigpoint
getanswer
getpoints
getbothpoints
getmousel
getmouse2
round
insertfrac
insertrecur
removeline
findfrac
removefrac
findrecur
removerecur
undrawrecur
```

Core Graphics module: This module is located in the core.c file and contains all the Core Graphics procedures.

1. global routines: The procedures affect the whole screen.

initialize
terminate
refresh
clear
delete
window
viewport
setvisiblity

2. text routines: These are the Core text writing procedures, but they were not employed by this project.

text
celldim
cellspace

3. drawing routines: These procedures were modified to include clipping.

linestyle
color
moveabs2
moverel2
lineabs2
linerel2
pointabs2
pointrel2

4. segment routines: Routines to control segments and clipping.

startseg
endseg
clearseg
deleteseg
refreshseg
clippingseg

5. file routines: New routines for file I/O.

savecore
retrievcore

6. matrix routines: Control rotating, scaling, & moving.

identity
scale
rotate
translate
push
pop
restore

7. inaccessible routines: These procedures are internal to the Core module.

map
findcode

testline
clip
cliptoseg
clipline
setmatrixidentity
multiplymatrix
transform
setratioandcell
save
driver
drawline

7.2. Communications among Modules.

The only communication between the main controlling module and the fractal procedures is through the variables that were passed. To initialize the fractals, only the variable indicating whether the mouse was in effect is used. The variables passed to make a new fractal include the existing color and line styles, the segment number for the fractal, and the clipping segment number. The fractal procedure returns the fractal number to the main program. That fractal number is used for duplicating fractals. The main controlling procedure does not know about the fractal "mold" structures except through the fractal number, nor does the Core module. For saving and retrieving fractal files, the file pointer and the fractal number are sent and the status and fractal number (retrieving only) are returned.

Both the main controlling module and the fractal module call the Core Graphics procedures in the normal manner, passing the needed variables and checking any returning ones. The Core Graphics structures are hidden from the both

the main and the fractal procedures, and are accessable only through the segment numbers.

7.3. Module Design Philosophy.

There were several guiding philosophies that I used in making coding decisions. One was that all shapes would have a unit shape that was transformed through matrix manipulations into the correct size and orientation requested by the user. Consequently, the triangle and square procedures are unusually complex and do not directly draw these shapes from the given points, as would be expected. The reasoning behind this philosophy was that since the fractals would always have a "unit fractal," and since most of the gemetric shapes could have a "unit shape," I decided to be consistent across both. The only exceptions to this rule were the point and line procedures where unit shapes were considered unnecessary since they were the units or building blocks of the other figures.

Another guiding philosophy was due mainly to the data space limitations of C package I was using. Since building the prompt menus and keeping Core structures of them around would require using precious memory space needed for the drawings, I decided on the following coding philosophy. Everything in the drawing part of the screen, such as the fractal windows, would always go through the Core Graphics structures. Everything in the menu areas would be written directly to the screen. Consequently, if a later

implementation of this project allowed more data memory space, only the menu procedures would need to be rewritten: "scr_rowcol" and "printf" statements would have to be changed to "moveabs2" and "text," respectively, and the whole menu surrounded by "startseg" and "endseg" routines.

The third philosophy was based on the existing Core Graphics programs of hiding structures from the user. This idea was used heavily in the fractal module and to a lesser extent in the master controlling module. The one place where the hiding became impossible was when a fractal mold was retrieved from a file independently of any referencing fractal drawing. In that case, the program was force to tell the user the fractal number used to reference that particular mold.

A fourth philosophy was to keep the hardware dependent code separated from the portable code as much as possible. In this project, the hardware dependent procedures are in three spots: the driver procedure (at the end of core.c), the entire mouse file (mouse.a), and the whole screen I/O file (scrio.a). Once these procedures are rewritten for a particular machine, the remaining code could run as is, after recompiling, of course.

8. Verification and Validation.

8.1. Test Plan.

The plan on this project was to build up each module and each sub-module within it one at a time, testing each before going onto the next one. Each module and sub-module would be independent enough that dependency problems would be minimized and bugs easily traced. For example, in the module to include rectangles, triangles, circles, and polygons, each shape was a sub-module which was build independently and tested for correctness before being included.

The basic design of this system was top-down, but the building of modules and testing used a bottom-up methodology. This proved easier to handle and made testing and isolating problems much easier.

8.2. Test Procedures.

Using a bottom-up methodology meant writing each sub-module and testing it to the specifications in the proposal until that particular section worked properly. This method was used for most of the project. The sub-modules and the resulting modules are given below:

- Installing a new driver on the existing Core Graphics program which required an assembly language procedure to write a dot to the screen.
- This sub-module made up the first module of upgrading the Core module.

- Making the triangle, rectangle, polygon, and circle procedures.
- Building the keyboard routine and menu for the main drawing screen, which required writing an assembly language procedure to handle inputs from the keyboard.
- Writing the clipping algorithm.

- These sub-modules made up the second module of making the software interactive from the keyboard.
  - Building the fractal main controlling procedure.
  - Writing the options to add and delete points to the first window.
  - Writing the recursive point procedures for the first window.
  - Writing the inverse and reverse options for the second window, as well as including the add and delete.
  - Building the recursive level selection for the main drawing.
  - Adding the random options to the second window.
  - Calculating the fractal number in the first window.
  - Adding the option to duplicate the previous fractals.

- These sub-modules made up the third module of creating the fractal generator.
  - Write assembly language procedures to access the mouse and its buttons.
  - Incorporate the mouse into the main drawing module.
  - Add the mouse to the fractal module.

- These sub-modules made up the fourth module of installing the mouse.
  - Building the file option menu in the main module.
  - Incorporate the file saving and retrieving into the Core module.

- Add the file option to the fractal module.

- These sub-modules made up the fifth module of allowing files to be saved and reused.

8.3. Test Results.

The only mathematical part of the project is the fractal dimension calculation. The program numbers fit those computed on my hand calculator. For the fractal triadic Koch snowflake with a grid of 6, the program calculated: 1.33 and by hand: 1.332Ø. An explanation of the algorithm used for the calculation is given in Appendix A.

Since this thesis was mainly a graphics project, its results lay in the visual execution of the different shapes and fractals. The following pictures of the screen, using the Okimate 2Ø, show different aspects of running program.

The first picture, Fig.1, shows all the standard shapes that can be drawn: point, line, triangle, square, polygon, and circle. The second illustration, Fig. 2, shows the clipping option being used. First the squares are clipped to the large circle. Then the small circles are clipped to the squares. Figures 3 through 1Ø show a fractal being generated from the initial selecting of points to the final resulting fractal, a Peano dragon in this case.

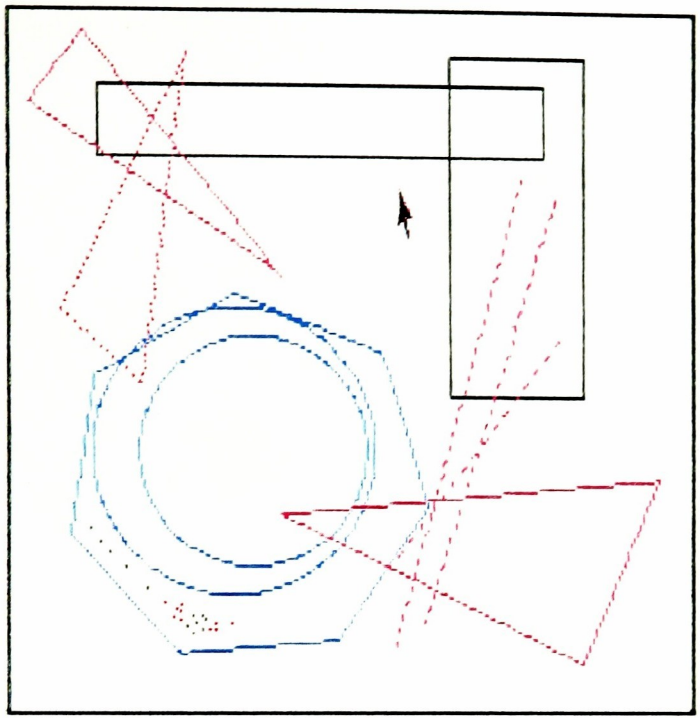


Figure 1. All 6 of the standard shapes.

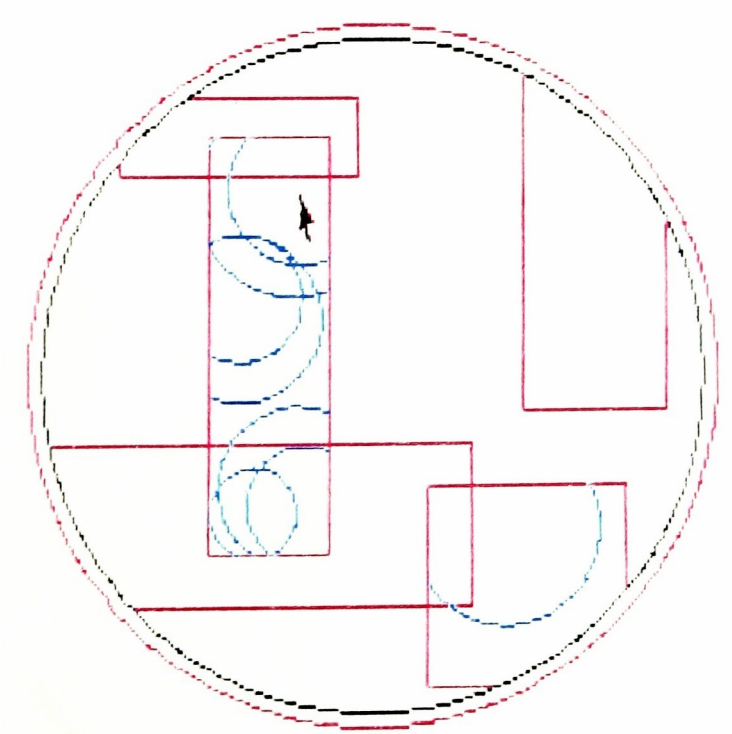


Figure 2. Clipping option.

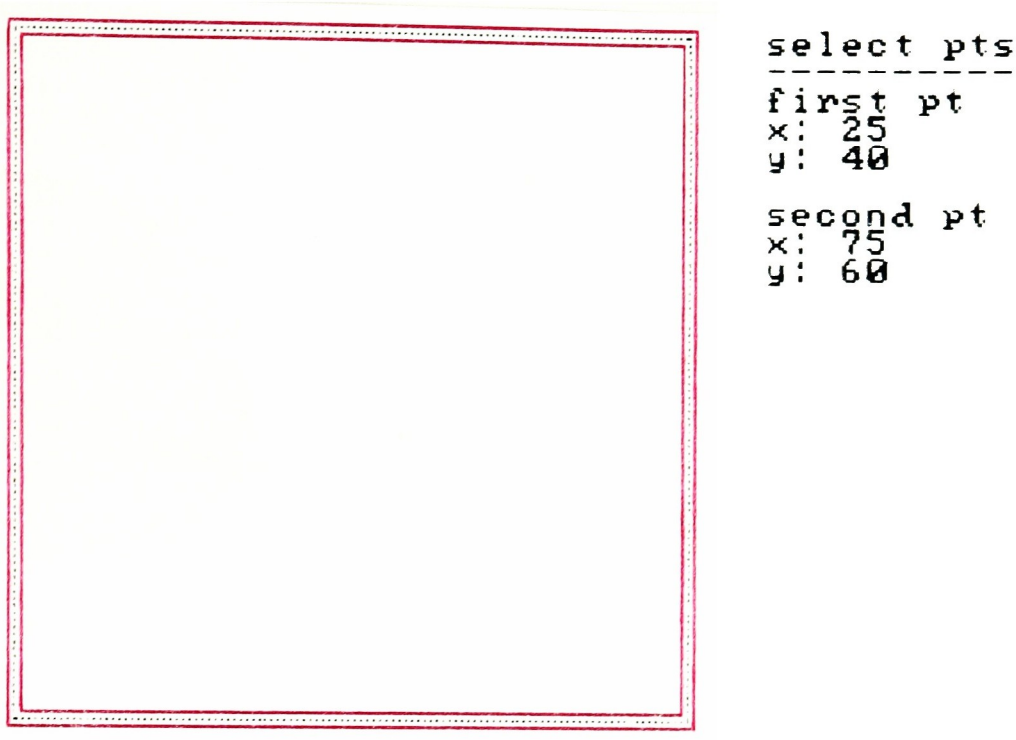


Figure 3. Point selection for the fractal.

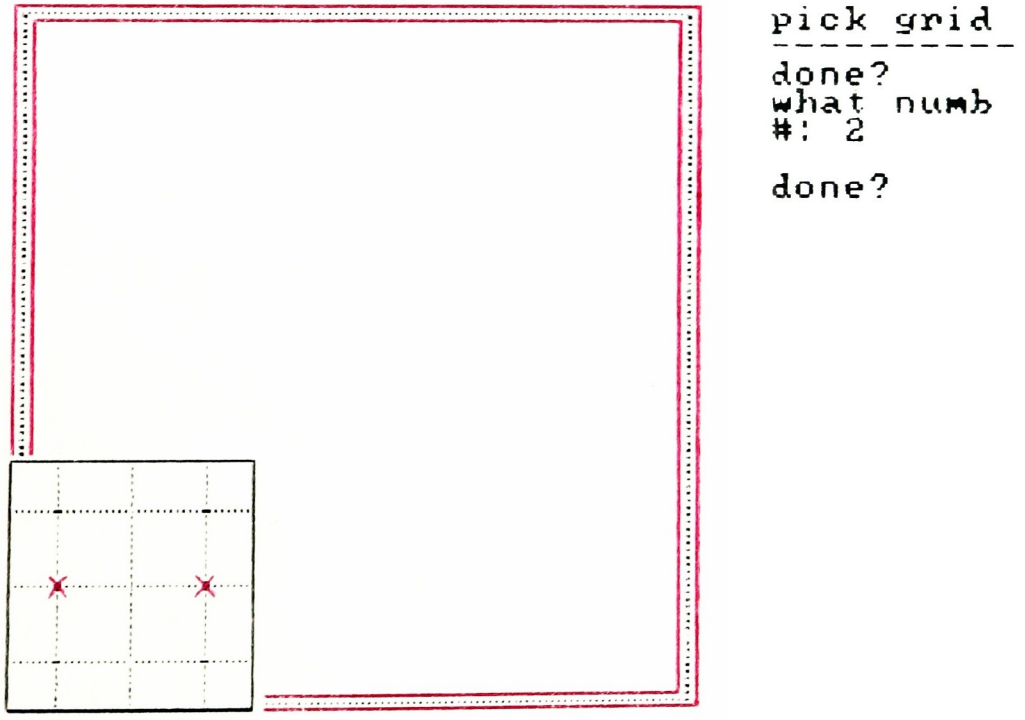


Figure 4. Grid selection.

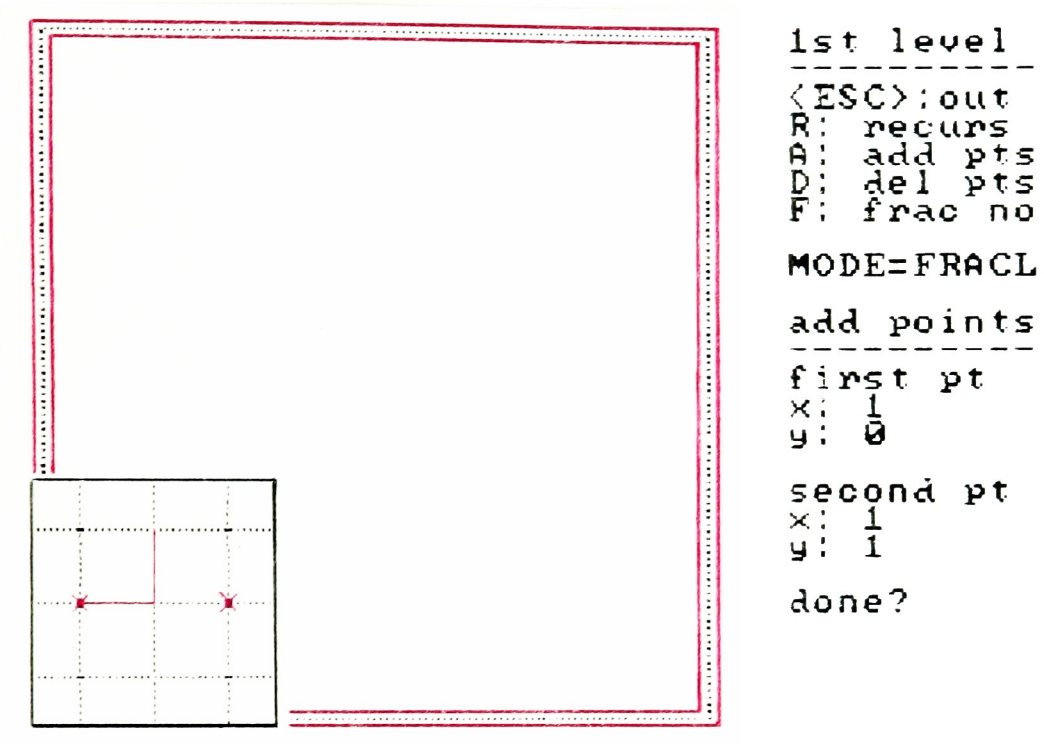


Figure 5. Adding points to the fractal generator.

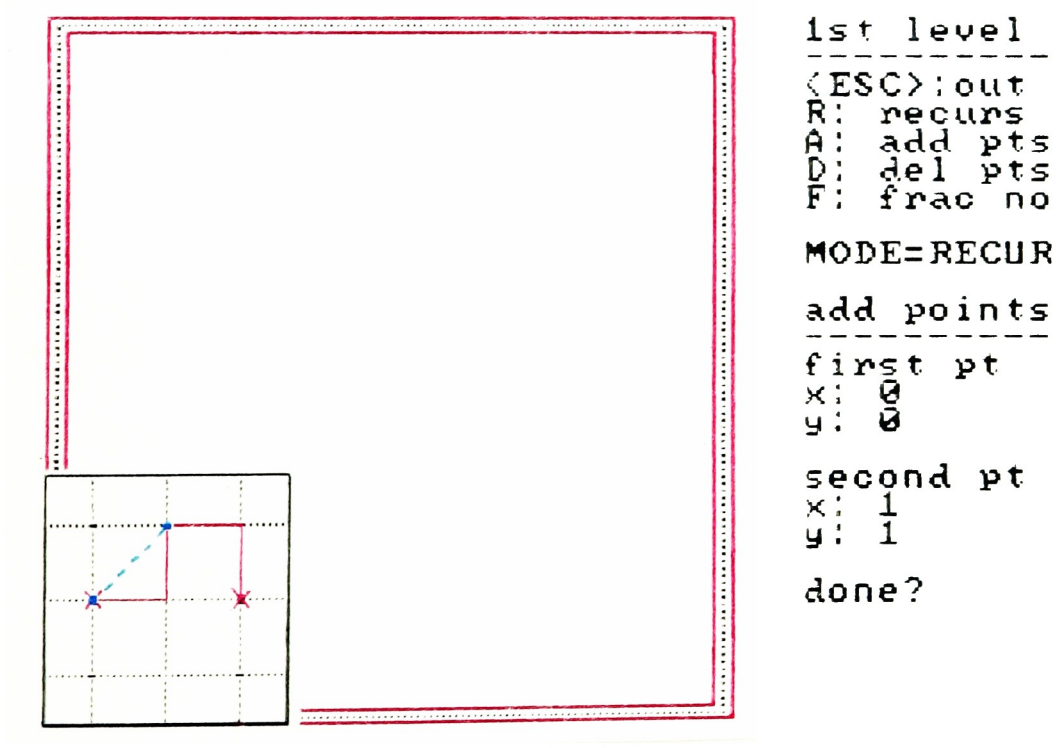


Figure 6. Adding points of recursion.

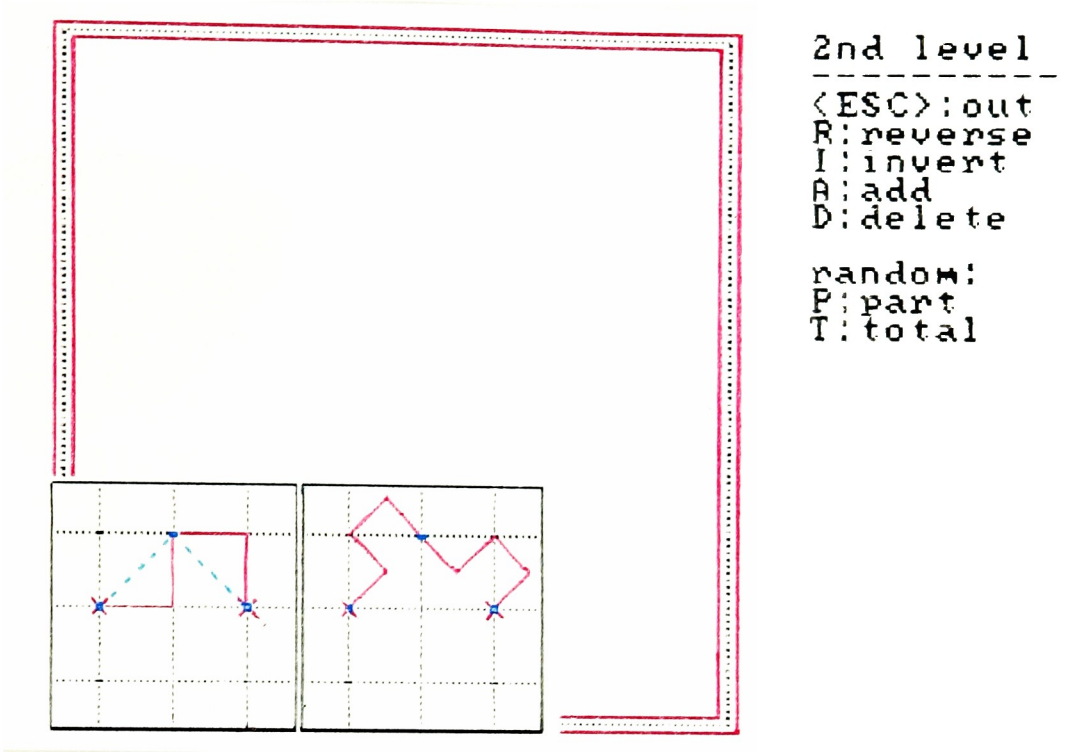


Figure 7. Second level generation.

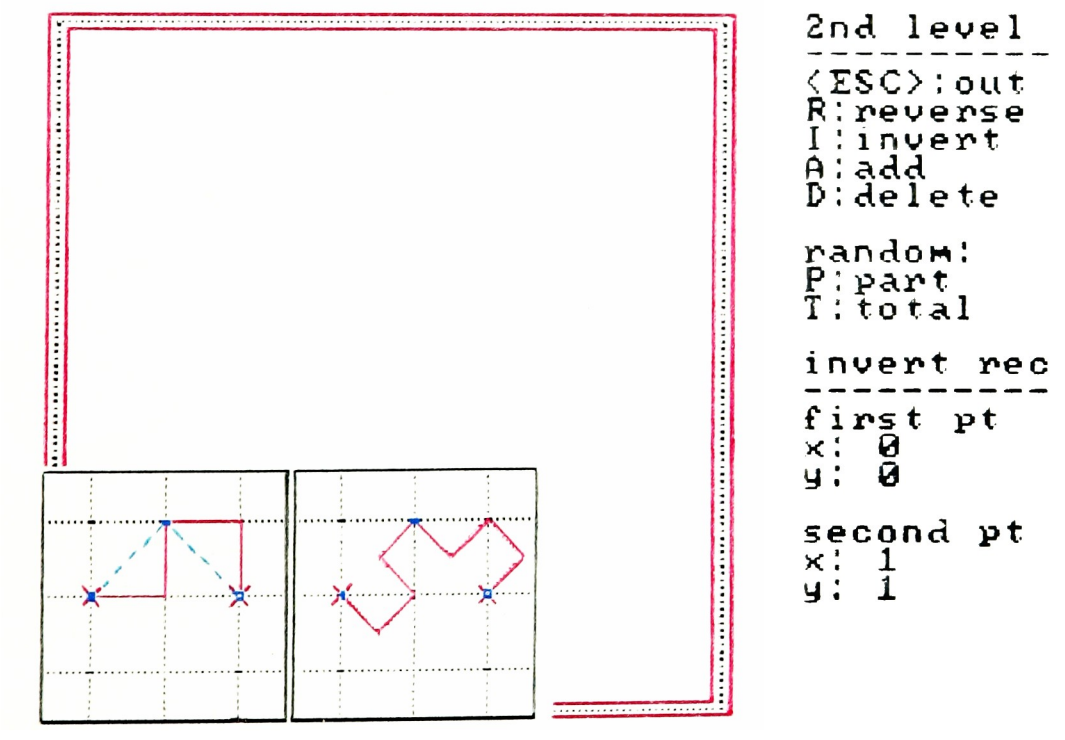


Figure 8. Sub-generator inversion.

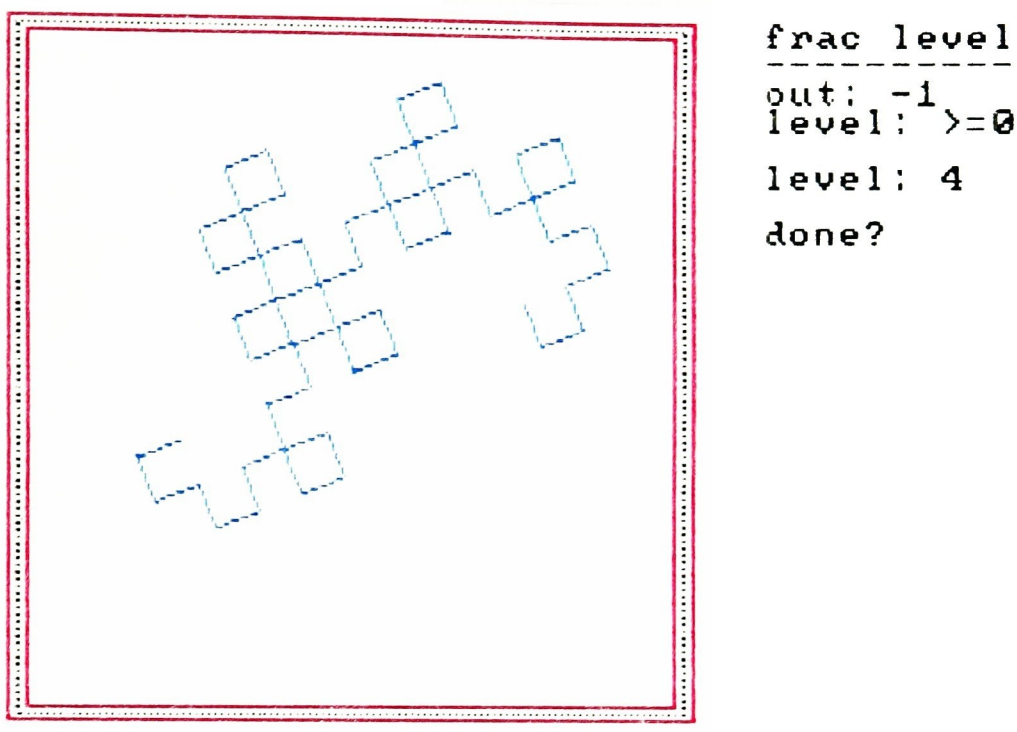


Figure 9. Level selection.

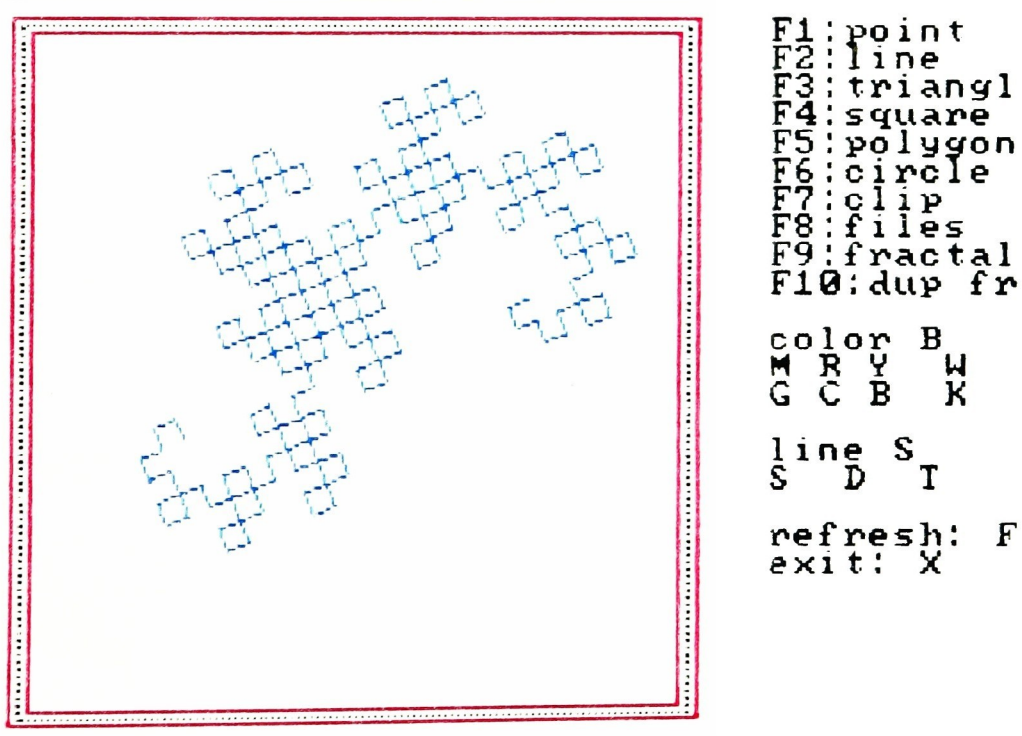


Figure 10. Resulting fractal in main drawing.

9. Conclusions.

9.1. Problems Encountered and Solved.

Problems that I encountered while working on this project and their solutions include:

- 64K memory limit for variables. This limit affected the how much I could use the Core Graphics commands. This was due to the fact that each time a call was made to either the text or drawing Core Graphics procedures, a Core structure was added in variable memory space. Since these structures filled memory so quickly, I decided was to restrict the use of the Core commands to only the graphic part of the display, the "drawing" part. The reason for this decision was that "drawing" commands had a direct impact on the main thrust of this thesis, while other commands, such as the menu prompts, were peripheral. Consequently, those other commands went directly to and from the microcomputer. For example, the text in the menus were placed and written directly to the screen, instead of employing the Core move and text writing procedures.

- Controlling the shapes generated. Since the clipping, fractal duplicating, and the file options all required some historical knowledge of the shapes drawn, a simple array system was used to keep track of the segment's number, and the shape's type. The array's index was the segment number, and the type was the integer within the array. The types were Ø to 6 for the standard shapes, the

number being the same as the F keys that created them, and for the fractals, any number greater than 99. Any drawing failures, such as a canceled fractal or an unavailable segment number, was indicated by a -1. The array size was set to 1000 since this was considered larger than any possible number of shapes, given the memory limit. This system could be easily changed to a link-list for a larger implementation.

- Clipping algorithm. I added a simple clipping algorithm which selected a segment shape and kept track of that segment's number. Then any subsequent lines would be clipped to each of the clipping segment's lines. Since triangles, rectangles, polygons, and circles have an interior, this was calculated by using the principle of cross multiplication on the first two lines of that shape. If a minimum of two lines were not found or they were parallel, the clipper would reject the request. An attempt to allow to clipping to either the inside or the outside of a selected shape did not work for two reasons. First, determining whether a line was on the outside or the inside of the clipping segment did not work well. Secondly, the existing line drawing algorithm did not handle a line clipping tht resulted in two separate lines. Since this area was not central to my thesis, the clipping was simply restricted to the interior of multi-sided shapes. In addition, fractals were not permitted as a clipping shape

because of their often convoluted forms made finding an interior difficult.

- Fractal dimension algorithm. Originally, I had thought that Mandelbrot's simple algorithm of taking the log of the number of sides and dividing it by the log of the grid would suffice in calculating the fractal dimension. Unfortunately this only worked when fractal lines stayed on the grid lines, not when any diagonals were drawn. Consequently, the full equation was necessary. A discussion of how this was implemented is covered in appendix A.

- Fractal adjust option. Having a user type in a fractal number and having the existing fractal drawing modify itself did not work as planned. After experimenting with several shapes to see how the fractal dimension changed, I came to the conclusion that a straight-forward algorithm could not be used. A complete discussion of this problem is covered in appendix B. Subsequently, this option was dropped.

- Color/resolution. I investigated several other video display boards beyond the one that came with the computer in an attempt to improve both color selection and resolution. Problems with those board's layout of the video memory or incompatability with the microcomputer eliminated most of them. Actually, the memory limitations for the program variables was a more constraining factor for fractal generation than the resolution of the screen. As a result,

further attempts to improve the resolution were discontinued.

- File saving and retrieving. My references on the Core Graphics System did not mention any standard way to store and retrieve files. I tried to come up on my own a Core-like version that reasonably hid the structures used, but still allowed linking them to the main controlling program. Also, there was a problem of conflict between fractal mold numbers from a retrieved file to numbers on already existing fractals. This was solved by checking the file mold number to see if it was used. If it was, a constant was incremented and added to the mold number until a number was found free. All subsequent fractal mold numbers from that file had that constant added to them. A third problem was that of retrieving a picture file without a fractal file to "hook" the fractal numbers. This problem was solved by adding 1ØØ to the fractal number when placing it into the array (mentioned above in the controlling shapes generated section). Doing this prevented these "unhooked" fractals from being accidently tied to good fractal molds, while still permitting identification of the shapes as fractals.

9.2. Discrepances and Shortcomings of the System.

Discrepanies include:

- Placement and output of the text for menus was done directly to the monitor through hardware specific calls instead of going through the Core Graphics move and text commands. This was unlike the main drawing area which used the Core commands. The main reason for this separation was the 64K memory limitation for variables. Storing those extra commands in the Core structures would have limited how much could be drawn in the main drawing, and so were not used.

- Mouse commands are direct instead of going through a Core pick command (see memory limitations for program variable mentioned above).

- The clipping algorithm does not generate a true polygon clip where the sides of the clipped shape (the clippee) follow the clipping shape (the clipper). Instead, the clipper acts like a odd shaped window against which the clippee shape is cut.

Shortcomings of the system include:

- No deleting of figures from the main drawing. Though this would have been relatively straight forward to implement, doing so would not have contributed considerably to this thesis.

- Limited color/resolution. Expanding this feature would have made the resulting drawings nicer (see discussion in 9.1 Problems Encountered and Solved).

- The escape key was not implemented fully. Installing this feature would have required writing a new version of scanf. Since doing so would have detracted from the main purpose of this thesis, I incorporated the escape key wherever it could be easily checked.

- No recovery when memory space was exceeded. When the number of variables exceeded the allowed memory space recovery consisted of rebooting the computer. To prevent this, memory space should be monitored and the drawing stopped before reaching this limit.

9.3. Lessons Learned.

I felt that I learned quite a bit, especially about microcomputers on this project. Other areas where I expanded by knowlege included:

- Learning about how to program the mouse.

- Discovering the layout of video display memory, as well as various means of accessing it.

- Investigating different video boards.

- Using and writing various assembly language procedures.

- Managing a large computer project, by breaking it into smaller pieces.

9.4. Future Development.

A possible future development of this project could be to implement it within the Unix operating system environment with output to either a Gigi or Masscomp terminal. This would involve rewriting the driver procedure (in core.c), the mouse routines (mouse.a) and the screen I/O routines (scrio.a).

Another possibility could be to extend the fractal options to include recursive textures, string formulas of turns and line lengths (much like Rollin's BASIC program), or random point offset to create "natural looking" fractals.

Extensions to the main drawing program could include deleting figures, adding a fill option, incorporating a rubberbanding feature to the figures when using the mouse, or including more geometric shapes.

10. Bibliography.

Conklin, Dick, PC Graphics: Charts, Graphs, Games, and Art on the IBM PC, John Wiley & Sons, Inc., New York, 1983.

Foley, James D. and Andries Van Dam, Fundamentals of Interactive Computer Graphics, Addison-Wesley Publishing Company, Reading, Massachusetts. 1983.

Harrington, Steven, Computer Graphics, a Programming Approach, McGraw-Hill Book Company, New York, 1983.

Haruyama, Shinichiro and Brian A. Barsky, "Using Stochastick Modeling for Texture Generation," IEEE Computer Graphics and Applications, Vol.4, No.3, pp.7-19, March 1984.

Kernighan, Brian W., and Dennis M. Ritchie, The C Programming Language, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1978.

Lee, Eui In, "Recursive Curves," PC Tech Journal, Vol.3, No.4, pp.171-174, April 1985.

Mandelbrot, Benoit B., The Fractal Geometry of Nature, W. H. Freeman and Company, San Francisco. 1982.

McClure, Matthew, "Computer Illusions," Popular Computing, Vol.4, No.3, pp.49-52. January 1985.

McDermott, Jeanne, "Geometrical Forms known as Fractals find Sense in Chaos," Smithsonian, Vol.14, No.9, pp.110-117. December 1983.

Norton, Alan, "Generation and Display of Geometric Fractals in 3-D," SIGGRAPH '82 Conference Proceedings. Ninth Annual Conference on Computer Graphics and Interactive Techniques, Vol.16, No.3, pp.61-67. July 1982.

Parsons, Vincent, and Ronald G. Parsons, "Let's Mouse Around," Dr. Dobb's Journal, Vol.10, Issue 4, pp.74-79, April 1985.

Peterson, Ivars, "Escape into Chaos," Science News, Vol.125, pp.328-329. May 26, 1984.

Purdum, Jack, C Programming Guide, Que Corporation, Indianapolis, Indiana, 1983.

Rollins, Dan, "A Tiger Meets a Dragon," Byte, Vol.8, No.12, pp.457-478. December 1983.

Rollins, Dan, IBM-PC 8088 MACRO Assembler Programming, Macmillan Publishing Company, 1985.

Sorensen, Peter R., "Fractals," Byte, Vol.9, No.10, pp.157-172. September 1984.

Stein, Kathleen, "The Fractal Cosmos," Omni, Vol.5, No.5, pp.63-71+, February 1983.

van de Panne, Michiel, "3D Fractals," Creative Computing, Vol.11, No.7, pp.78-82, July 1985.

Waite, Mitchell, Stephen Prata, and Donald Martin, C Primer Plus, Howard W. Sams & Co., Inc., Indianapolis, Indiana, 1984.

Wardop, Simon, "Plotting Fractals on your Computer," MICRO, No.70, pp.74-77. March 1984.

Witten, Ian H. and Radford M. Neal, "Using Peano Curves for Bilevel Display of Continuous-Tone Images," IEEE Computer Graphics and Applications, Vol.2, No.3, pp.47-52. May 1982.

Software and Hardware Manuals:

DeSmet C Development Package Manual, version 2.4, C Ware Corporation, 1984.

Microsoft Mouse for the IBM Personal Computer, Installation and Operation Manual, Microsoft Corporation, 1983.

Programmer's Reference Manual, Z-100 PC Series Computers, Zenith Data Systems Corporation, 1984.

11. Appendices.

A. Fractal dimension algorithm.

Initially, I had thought that taking the log of the number of sides and dividing it by the log of the grid, Mandelbrot's simple algorithm, would suffice in calculating the fractal dimension. However, this method only worked when the recursive fractal lines stayed on the grid lines, not when any diagonals were drawn. Consequently, the full equation was necessary to calculate the fractal dimension. This involved taking each side length divided by the grid and raising the result to the fractal dimension. Then the sum of these fractions was compared to 1.Ø which a correct dimension would produce.

As implemented in my project, the fractal dimension was calculated as follows. First, an approximation to the fractal dimension was done by using the simple algorithm first. This gave an approximate number that was accurate for figures that stayed on the grid. For example, if the Koch snowflake was drawn on a grid of 6, the simple algorithm would sum the sides as: $2 + \sqrt{5} + \sqrt{5} + 2 = 8.4721$. Then the log of the sides divided by the log of the grid would produce a first attempt at the dimension:
$\log 8.4721 / \log 6 = 1.1926$.

With this first dimension, the full equation was calculated: $(2/6)^{1.1926} + (\sqrt{5}/6)^{1.1926} + (\sqrt{5}/6)^{1.1926} + (2/6)^{1.1926} = 1.1558$. Since the answer to this equation was

higher than 1.Ø, the desired result, the fractal dimension would have to be adjusted.

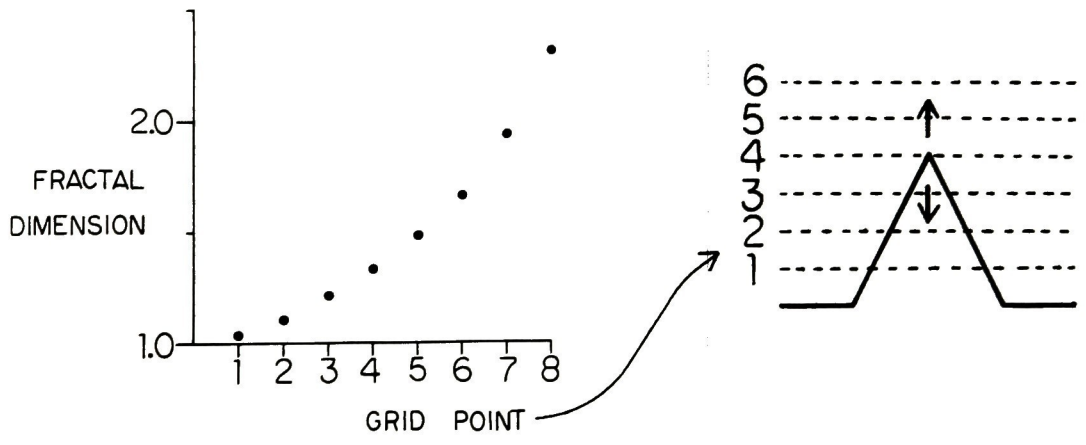
One effective way found was to take the difference of the answer and one, and then add that to the dimension: 1.1926 + (1.1558 - 1.ØØØØ) = 1.3484. This new fractal dimension was then used again in the full equation: $(2/6)^{1.3484} + (\sqrt{5}/6)^{1.3484} + (\sqrt{5}/6)^{1.3484} + (2/6)^{1.3484} =$ Ø.9831. Now the answer came closer to 1.Ø. Again, the dimension was adjusted: 1.3484 + (Ø.9831 - 1.ØØØØ) = 1.3315.

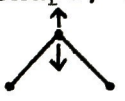
This process was continued until the difference between the old dimension and the new was less than Ø.ØØ1, usually in three or four iterations. That tolerance was hard-coded in the program and could be made lower, thus making the fractal dimension more accurate, at the cost of more iterations.

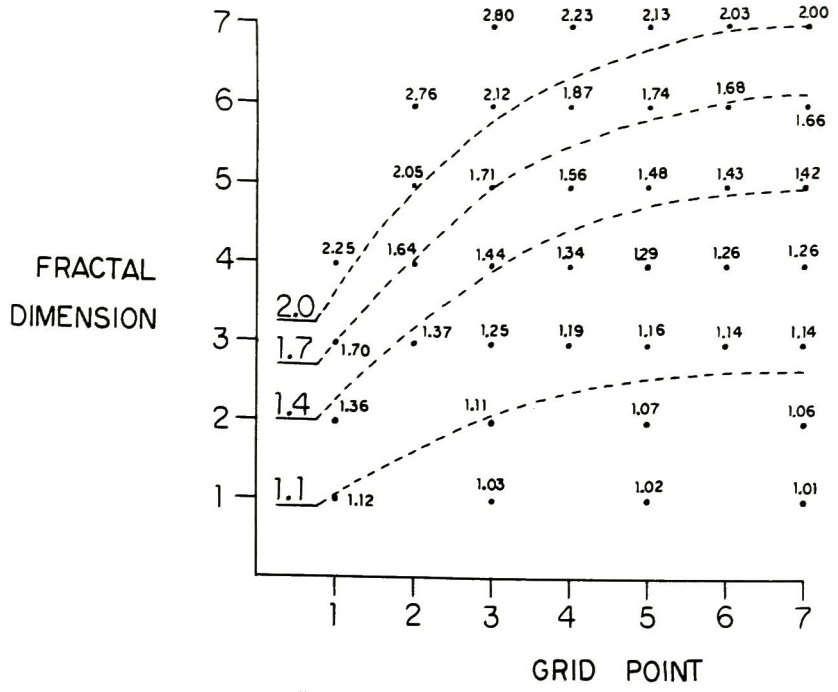
As mentioned earlier, the first attempt at the fractal dimension was accurate for figures that stayed on the grid, such as the Peano C curve. What would happen in the program was that the first dimension was used in the full equation and the answer would immediately equal to 1.Ø. The effect was that the full equation simply comfirmed the simple algorithm's calculation.

B. Fractal adjust option.

The fractal adjust was going to be an option in the first window of making a new fractal. This option would allow a user to type in a fractal number and have the existing fractal drawing modify itself. Unfortunately, this did not prove feasible. The main reason was that changes in the fractal shape resulted in non-linear changes in the dimension calculated. For example, if the Koch snowflake was built on a grid of 12, and point of the snowflake was moved up and down, the dimension changes would be quite large and non-linear. The following graph shows the results of this experiment.



A second experiment using an even simpler shape, a generator for the C or dragon curve, was tried.  By changing the middle point and keeping the end points constant, different fractal dimensions were calculated. The resulting dimensions for each point and an estimate of the iso-dimension lines are shown below. Again, the change in dimension was non-linear with changes in the fractal.



Other forms of shape modification produced similar results using the grid. Releasing the restriction of keeping the shape's points on the grid would have improved the chances of getting other fractal dimensions, but at the expense of loosing access to those points. The reason for the loss has to do with the fact that all input, including the mouse's, was rounded to the nearest integer.

On the other hand, if that rounding was turned off, trying to access points that were previously put on the figure with the mouse would be very difficult; the coordinates would have to be entered exactly to get a match.

Presently, any modification to the figure can be easily done by the user since both the add and delete options are available. Following any changes, the user could invoke the option for the fractal dimension and thereby see the consequences of those alterations.

12. Program Listings.

The listings are on file with the Rochester Institute of Technology School of Computer Science and Technology Department.

13. User Manual.

Overall intent: This program is meant as a simple graphics package with the added feature of fractals. This package will allow a person to draw the outline of different fractal and geometric shapes in various colors and lines. No filling or solid shapes are available.

Exit = to leave the program, type X with no carriage return.

Color = only three colors plus background are available: white which is invoked by W; magenta which is invoked by M, R (red), or Y (yellow); cyan, by C, G (green), or B (blue); and black (background) which is K. Any of these letters will change the color of the line drawn. No carriage return is necessary.

Line style = three line styles are implemented: solid can be obtained by using the S key; dashed lines by the D; and dotted lines by a T. No carriage return is necessary.

Refresh = an F will clear the picture area and redraw everything. This is useful if any point selection number overflowed into the drawing. No carriage return is necessary

Mouse:

Basic concepts: a left mouse press will generate both the x and y coordinates. If any keyboard key is pressed at the "x:" prompt instead of a mouse button, then the mouse can't be used until the next "x:".

Left button = point selection. Coordinates of the selected point will be shown in the menu. Usually a right button will have to be pressed to close a menu. In addition, when a duplicate fractal is to be selected, the left button represents a "N" for number.

Right button = carriage return. This is used to close a menu after the left button has chosen a point. This allows you to inspect the resulting coordinates before going on. Also, when a carriage return is requested before closing an option or when making a selection.

Both buttons = "Y" or "Yes". Some places need a response to determine whether to continue or not. If you don't want to continue, use the right button to indicate a null response. Both buttons mean a "yes."

Drawing:

Basic concepts: The x and y coordinates go from Ø.Ø to 1ØØ.Ø starting in the botton left corner. The center of the drawing screen is 5Ø.Ø, 5Ø.Ø. Either an integer or a floating point can be entered. Any non-numeric keys will give unexpected results, and program may have to be restarted.

Point = This is selected by using the F1 key. To indicate the location, either type in the coordinates, or press the left button of the mouse followed by the right button. The color will be determined by the last color selected. Linestyle has no effect on points.

Line = Using the F2 key will select the line option. To indicate the two end points, either type in the coordinates, or press the left button of the mouse. Both of these can be alternated. If the last point entered employed a mouse, you need to press the right mouse button to leave this option.

Triangle = This is chosen using the F3 key. To select the three end points, either type in the coordinates, or press the left button of the mouse. Both of these can be alternated. If the last point entered employed a mouse, you need to press the right mouse button to leave this option.

Rectangle = The F4 key gives this option. To select the two opposite corners of the rectangle, either type in the coordinates, or press the left button of the mouse. Both of these can be alternated. If the last point entered used a mouse, you need to press the right button to leave this option.

Regular Polygon = This is chosen using the F5 key. The first point is the center of the polygon and the second is the edge. Either type in the coordinates, or press the left button of the mouse to select two points. Both of these can be alternated. The number requested is number of sides for the polygon. At least 1 side is necessary. The number of sides must be entered by the keyboard. If the last point entered entered

employed a mouse, you need to press the right mouse button to leave this option.

Circle = The F6 key gives this option. The first point is the center of the circle and the second is the edge. Either type in the coordinates, or press the left button of the mouse to select two end points. Both of these can be alternated. If the last point entered was a mouse, you need to press the right mouse button to leave this option.

Fractals:

Basic concepts: This program does only recursive fractals, such as the Koch snowflake. By filling in each of the windows, you supply the program with information to build a fractal mold. That mold can be saved to and retrieved from a file, as well as invoked later for duplicating on the main drawing.

End points = These are the two end points for the fractal on the main drawing. Just like the line option, either type in the coordinates, or press the left button of the mouse to select two end points. Both of these can be alternated. If the last point entered was a mouse, you need to press the right mouse button to leave this option.

Grid = A default grid of four is brought up. If it is not satisfactory, either a carriage return or right button on the mouse will give you a chance to change the grid. A minimum of 1 and maximum of 15 is allowed. All subsequent point selection will be rounded to that grid.

First window = In this window you are building the fractal generator and part of the recursive algorithm. The left end red X is the Ø,Ø point and represents the first point you selected on the main drawing. To add points just type an A, and as many line endpoints as desired. To delete, type D, and the coordinates of any previously added lines. To end either selection, type a "Y" or use both mouse buttons when the "done?" prompt is made. Point selection can be made by the mouse or typed in. Selection between adding the generator (solid red) and the recursion points (dashed blue) is done by a toggle, typing an R. Which state you are in is reflected in the "MODE=" line. The F option is the fractal dimension as calculated on the recursion points. The calculation does not take overlapping of the final

fractal into account. To go on to the second window, you must use the F1Ø key.

Second window = This window shows the second level of the fractal that was built in the first window. At this point, more options are availabe to adjust the types of recursion: besides adding (A) and deleting (D), lines can be inverted (I), reversed (R), or randomized (P). The total figure can be made random by (T). The random option acts like a toggle. If a recursion line is already random, selecting it again will revert it back to an unrandom state. Those random lines will be indicated by changing the red lines to blue. If the fractal mold is unsatisfactory, you can go back to the first window by using the F9 key. Otherwise, to leave the second window, you use the F1Ø key again.

Recursion Levels = This is the last phase in making a fractal. At this time the level of recursion for the fractal at the original two end points is selected. Typing any number greater or equal to zero will produce a fractal, though a level of 7 or more tends to cause the program to run out of memory. A level of -1 will take you back to the main menu without any fractal being drawn and all work for that session being lost. After typing in the level, the program will ask if you are done. If so, typing a Y (no carriage return) or using both mouse buttons will return you to the main menu. If you want to try another recursion level, a carriage return or right mouse button will give you another chance.

Duplicating Fractals = This option is available at the main menu by using the F1Ø key (no relation to the previous use of F1Ø in making a fractal). Three phases will be necessary to make the fractal. The first is selecting a fractal to duplicate. Either selecting a fractal on the main drawing, or typing the number of the fractal mold that had been retrieved from a file can be used. To select a fractal on the main drawing, type a Y when that fractal is flashed. Otherwise, enter a carriage return to move the selection to the next fractal. Typing an N, for a number, will give a secondary prompt for which mold number you want used. After the fractal is selected the second phase of picking the fractals end points comes up. This phase acts the same as the first part of making a new fractal. The third and last phase is the same as the last step of making a fractal, selecting the recursion levels.

Miscellaneous commands:

Clipping = This option is available at the main menu by using the F7 key. To select a shape on the main drawing for clipping, type a Y when that figure is flashed. Otherwise, enter a carriage return to move the selection to the next figure. If one shape is selected, all drawings including fractals will be clipped to that shape. If the program comes back and says that you can't use that shape for clipping, it is because there is an insufficient number of lines to find the inside of the figure. To turn off the clipping, the F7 option must be selected again and either another shape must be chosen for clipping, or no shapes. No selection is done by entering a carriage return for all flashing figures.

Saving and Retrieving Files = Selecting the F8 key will give this option. It is entirely menu driven with a maximum of four questions. A null answer (just a carriage return) at any one question will return you to the main menu. The first is whether to save (S) or get (G) a file. The second question asks what type: a single fractal (S) (availabe only on a save), all fractal molds (F), just the picture without any fractal molds (P), or both a picture and the fractals (B). If a single fractal is chosen, the selection is made exactly like duplicating a fractal, either choosing a flashing fractal or entering a mold number. The next question asks which disk drive (A or B) to use. The last question is to enter a name of the file. If a save was requested, any files with the same name will be overwritten. If a retrieval of only a fractal file was invoked, the fractal mold number will be shown for each mold found. Those numbers are the only access you will have to those fractal molds.