

06/12/00

MUTATIONS OF THE MANDELBROT SET (Revised)

Mathematics and Computer Education, Vol. 35, No. 1, (2001), pp. 18-26.

Jae Hattrick - Simpers

Thomas J. Osler

Department of Mathematics

Rowan University

Glassboro, NJ 08028

Osler@rowan.edu

1. INTRODUCTION

The Mandelbrot set is the most famous of all fractals. It is easy to generate on a home computer and full of fascinatingly beautiful detail. The Mandelbrot set (shown in Figure 1) is sometimes called “the bug” for obvious reasons. In this paper, we show how to make simple modifications of the iterations which generate the bug in order to produce mutations. The possible variations are countless forming numerous families with various orders, classes, phyla, and kingdoms. Thus it becomes possible to classify the new creatures as they evolve from the original Mandelbrot set in the form of a family tree. Different mathematical iterations produce different families of descendants. (While we do not present such a family tree here, we do give the necessary programs and explanations so that students can discover their own.) We believe that this simple exercise can be an entertaining, informative and creative project for students in elementary computer science courses as well as in any course where iterations are introduced such as numerical analysis and dynamical systems.

We begin by reviewing the mathematical iterations which define the Mandelbrot set. Then a simple program in Quick BASIC is given along with a description of how it works. We also show how to get started easily in Visual Basic and how to modify the program so that it will run in this popular new language. Even if the reader is not familiar with the creation of the classical Mandelbrot set, the review material presented in sections

2 and 3 should be sufficient to get started. Next we describe how certain lines in the program can be modified to create mutations of the Mandelbrot image. An undergraduate mathematics major at Rowan University tried this project and we show some of his results. Since there are countless ways in which to vary the computer program, this project can be given to different students with different results.

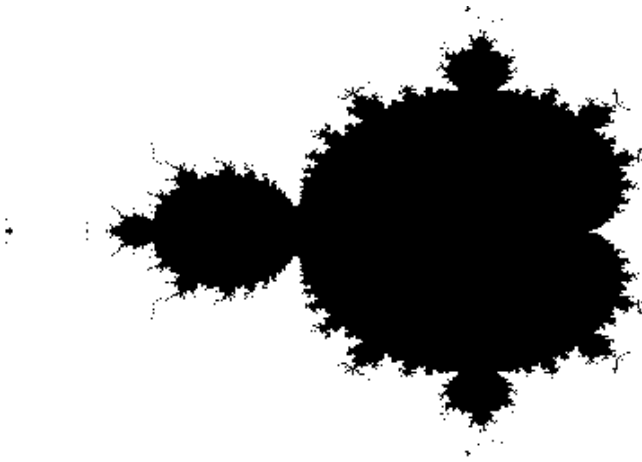


Figure 1: The Mandelbrot set

2. THE MANDELBROT SET

We begin by reviewing the mathematical apparatus by means of which the Mandelbrot set is created. An excellent book on this subject for beginners is by Devaney [2].

The Mandelbrot set emerges from the study of the iterations of the simple function

$$(2.1) \quad f(z) = z^2 + c.$$

It is remarkable that anything interesting (as the shape in Figure 1) can arise from such a simple expression. Here is how the iteration works:

1. Let c be a fixed number.
2. We will always start with $z_0 = 0$, and then generate a sequence

$$\begin{aligned} z_1 &= z_0^2 + c \\ z_2 &= z_1^2 + c \\ z_3 &= z_2^2 + c \\ &\dots \\ z_{n+1} &= z_n^2 + c \\ &\dots \end{aligned}$$

We call this sequence z_0, z_1, z_2, \dots the *orbit* of the above iteration.

Example 1: Let $c = 0$. Find the orbit.

Using the above iterations and starting with $z = 0$ we get

$$\begin{aligned} z_1 &= z_0^2 + c = 0^2 + 0 = 0 \\ z_2 &= z_1^2 + c = 0 \\ z_3 &= z_2^2 + c = 0 \\ &\dots \end{aligned}$$

Thus we get the constant orbit $0, 0, 0, \dots$. This sequence has the limit 0, but our interest will center on a weaker condition called *boundedness*. We say that the orbit is *bounded* if a number R exists such that $|z_n| < R$ for all elements of the orbit. This orbit is *bounded*.

Example 2: Let $c = 1$. Now, starting with $z_0 = 0$, we generate the following iterations:

$$\begin{aligned} z_1 &= z_0^2 + c = 0^2 + 1 \\ z_2 &= z_1^2 + c = 1^2 + 1 = 2 \\ z_3 &= z_2^2 + c = 2^2 + 1 = 5 \\ &\dots \end{aligned}$$

The orbit in this case is $0, 1, 2, 5, 26, \dots$ which approaches infinity. We say this orbit is *unbounded*.

Example 3: Let $c = -1$. Now we have (starting from $z_0 = 0$)

$$z_1 = z_0^2 + c = 0^2 - 1$$

$$z_2 = z_1^2 + c = (-1)^2 - 1 = 0$$

$$z_3 = z_2^2 + c = 0^2 - 1 = -1$$

...

Now the orbit is $0, -1, 0, -1, 0, -1, \dots$. This orbit is called *bounded*.

We are now ready to explain the Mandelbrot set. Let both $z = x + iy$ and $c = c_x + ic_y$ be complex variables having real and imaginary parts as shown. Our bug (shown in Figure 1) is created in the complex c -plane. For each point in the complex c -plane, we calculate the orbit of the iterations of $f(z)$ as shown above. If this orbit is unbounded, we imagine painting the point c white, otherwise we paint it black. This is how the Mandelbrot set in Figure 1 was created.

Thus the Mandelbrot set is the collection of all points in the complex c -plane where the orbit of the iterations of $f(z)$ is bounded.

We conclude by noting that the complex calculations for the orbit are

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

$$x_{n+1} + iy_{n+1} = (x_n + iy_n)^2 + (c_x + ic_y).$$

Expanding and collecting real and imaginary parts we get

$$(2.2) \quad x_{n+1} = x_n^2 - y_n^2 + c_x$$

$$(2.3) \quad y_{n+1} = 2x_n y_n + c_y.$$

These two relations will be needed in the computer program given in the next section.

3. A PROGRAM FOR THE MANDELBROT SET

The following program, written in Quick BASIC generates the Mandelbrot set using the ideas explained in the previous section.

```
' PROGRAM TO GENERATE THE MANDELBROT SET

' ENTER SCREEN SIZE FROM THE KEYBOARD

100 INPUT "ENTER XMIN, XMAX "; XMIN, XMAX
110 INPUT "ENTER YMIN, YMAX "; YMIN, YMAX

' READY THE SCREEN FOR HIGH RESOLUTION - VGA MODE

120 SCREEN 12: XPIX = 640: YPIX = 480
130 WINDOW (XMIN, YMAX)-(XMAX, YMIN)
140 STEPX = (XMAX - XMIN) / XPIX
150 STEPY = (YMAX - YMIN) / YPIX
160 CLS

' ESTABLISH NESTED LOOPS TO COLOR THE  CX, CY - PLANE
' ONE PIXEL AT A TIME

170 FOR CY = YMIN TO YMAX STEP STEPY
180  FOR CX = XMIN TO XMAX STEP STEPX

' START ORBIT AT  $Z = 0$  ( $Z = X + IY$ )

183  X = 0: Y = 0

' EXAMINE 25 POINTS ON THE ORBIT

185  FOR N = 1 TO 25
190  XX = X * X - Y * Y + CX: YY = 2 * X * Y + CY
200  X = XX: Y = YY

' IF THE ORBIT APPEARS TO BE GOING TO INFINITY, PAINT THE
' CURRENT PIXEL IN THE  CX, CY - PLANE

210  R = X * X + Y * Y: IF R > 4 THEN PSET (CX, CY), N: GOTO 230
220  NEXT N

230 NEXT CX
```

240 NEXT CY

Lines 100 and 110 allow the user to enter from the keyboard the portion of the complex c -plane to be viewed on the screen. The real and imaginary parts of the complex variable c are denoted in the program by CX and CY . The limitations of these variables are $XMIN < CX < XMAX$, and $YMIN < CY < YMAX$. To generate Figure 1 we used $XMIN = -2$, $XMAX = 2$, $YMIN = -2$, and $YMAX = 2$.

In line 120 the SCREEN 12 command allows high resolution graphics with 640 by 480 pixels. The WINDOW command in line 130 is Quick BASIC's way of describing the coordinates of our rectangular screen. Lines 140 and 150 determine the tiny distances separating pixels in the horizontal and vertical directions.

The double loops in lines 170 - 240 and 180 - 230 allow us to examine points on the complex c -plane pixel by pixel. We start with the bottom row ($CY=YMIN$) and go row by row to the top ($CY=YMAX$). On each row we start at the left ($CX=XMIN$) and proceed to the right ($CX=XMAX$).

Line 183 starts the iterations at $z = 0$. The iterations are repeated by the FOR-NEXT loop in lines 185-220 where the variable N counts the iterations. Line 190 is at the heart of the calculation. XX is the new value of X as given by relation (2.2) and YY is the new value of Y as given by (2.3). Line 200 simply renames the variables.

In line 210 we first calculate R which is the magnitude of $|z_n|^2$. We use this to see if the iterations are apparently becoming unbounded. It can be shown that if R exceeds the value 4, then the iterations will be *unbounded*, and the point in question is *not* in the Mandelbrot set. The PSET statement is Quick BASIC's way of coloring the pixel at $c = CX + iCY$. The color selected is determined arbitrarily by the number N which in this

case measures how rapidly the iterations are diverging. The GOTO 230 statement stops the iterations and we move on to the next pixel.

If the lines 185 to 220 are repeated the maximum possible 25 times, then we will assume that the iterations are remaining *bounded*. This is not always true, but we have to stop calculating sometime. In this case the pixel is not colored, and appears black on the screen indicating that it is in the Mandelbrot set.

This completes our description of how the program generates the Mandelbrot set.

4. RUNNING THE PROGRAM IN VISUAL BASIC

Quick BASIC interpreters are no longer available. Yet, Quick BASIC may have been the most popular programming language in the world ten years ago. In its place Microsoft now markets Visual Basic. Fortunately, Visual Basic code is identical to Quick BASIC code with only a few exceptions. Readers unfamiliar with Visual Basic can obtain the so called “Control Creation Edition” on CD as well as a book describing the language at any book store. For example we purchased [4] for only \$19.95 which included the CD.

Getting started with Visual Basic can be discouraging because there are so many options available. To get started quickly, read the first few pages of the book and perform the following simple steps.

1. Bring up a new form.
2. Place a command button on the form.
3. Double click on the command button and the window for the Basic code opens.
4. Simply type in our program with the following modifications:

Replace lines 100 and 110 with

```
100 XMIN = -2 : XMAX = 2
```

```
110 YMIN = -2 : YMAX = 2
```

Replace lines 120 and 130 with

```
120 XPIX = 640 : YPIX = 480
```

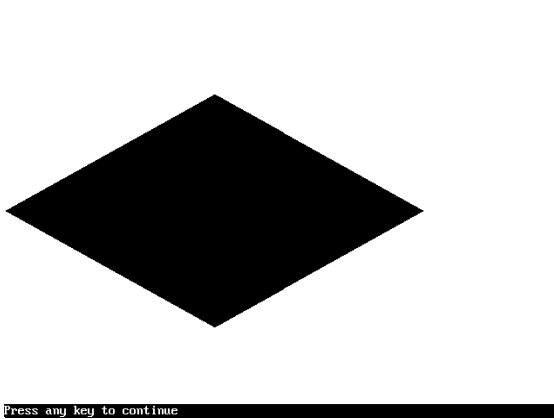
```
130 Scale (XMIN, YMAX)-(XMAX, YMIN)
```

Now click on “Run” on the tool bar and the form with the command button will appear. Simply click on the command button and the fractal will be generated on the form. We note that our inexpensive Control Creation Edition of Visual Basic draws the fractal on the screen in about the same time as our Quick BASIC interpreter.

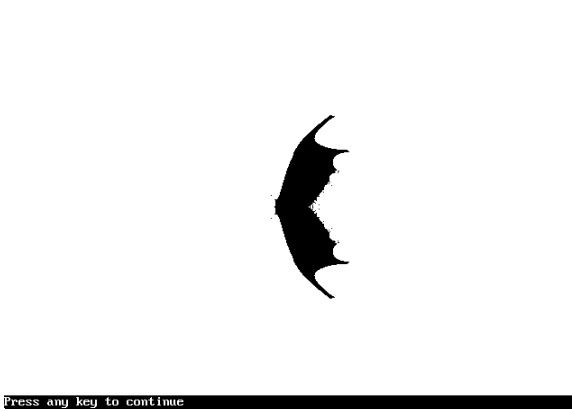
5. MUTATIONS OF THE MANDELBROT SET

A nice feature of this project lies in the simplicity of the changes that need to be made to the program to alter the shape of the set. Students with even the most rudimentary computer skills will be able to produce aesthetically pleasing fractals of great complexity. Students will also be able to see two reoccurring themes in dynamical systems, (1) complexity emerging from simplicity, (2) large changes resulting from small changes. What follows is a description of the process by which the students will discover mutations of the Mandelbrot set.

In this project, the “tinkering” is almost exclusively limited to line 190 of the Quick BASIC program. This tinkering is like altering the proteins that make up the DNA of the bug. Changes in the proteins can be made as arbitrarily as desired with occasional problems, to be described later. The following four figures show changes that have been made to line 190 of the program paired with the animals that are created.



**Figure 2: The result of changing lines 190 to
 $XX=X*X+Y*Y+CX$: $YY=2*X*Y+CY$.**



**Figure 3: The result of changing line 190 to
 $XX=\text{Sin}(X)-Y*Y+CX$: $Y*Y=2*X*Y+CY$.**

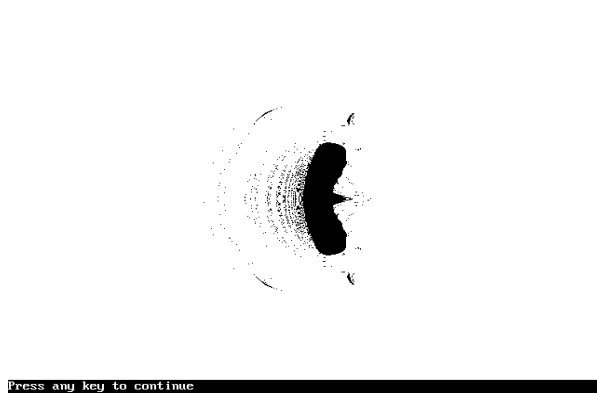


Figure 4: The result of changing line 190 to
 $XX=X/(X^5+1) - Y*Y+CX: YY=2*X*Y+CY.$

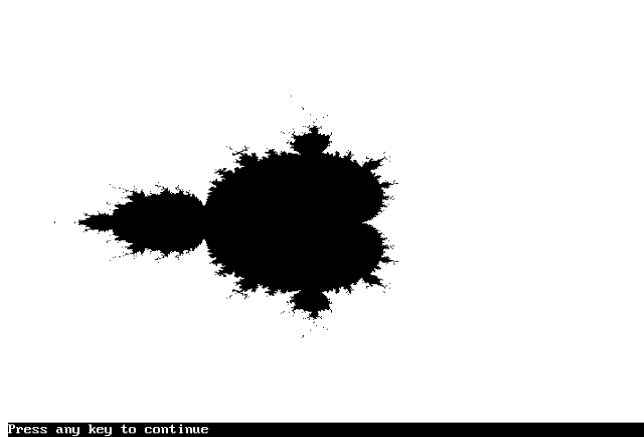


Figure 5: The result of changing line 190 to
 $XX= \text{Abs}(X)^{(5/3)} - Y*Y+CX: YY= 2*X*Y +CY.$

Some minor changes to the formula, as in Figure 2, produce major changes in the resulting picture. (Here only a minus sign is changed to a plus sign.) On the other hand, some changes, like the Figure 5, produce pictures that largely resemble the original fractal. Then there are creatures as in Figures 3 and 4, whose formula and pictures in no

way resemble that of the original Mandelbrot set. A portion of this project's charm lies in the unpredictability of how the changes will affect the shape of the set.

When assigning this project we discovered several problems that the students are likely to encounter. We discuss a few of these below.

First of all when dealing with fractional exponents or other manipulations that yield imaginary numbers, a certain provision needs to be made to prevent the program from crashing. An absolute value can be incorporated into the term being raised to a fractional power to make it valid. For instance, the line

$$190 \text{ XX} = X^{(5/3)} - Y * Y + CX: \text{ YY} = 2 * X * Y + CY$$

will certainly cause the program to crash, even though the root is odd. The line

$$190 \text{ XX} = (\text{Abs}(X))^{(5/3)} - Y * Y + CX: \text{ YY} = 2 * X * Y + CY$$

which incorporates an absolute value works, and Figure 5 is its output.

Another potential bug is the occasional problem of window size. Certain functions at particular points just can not be calculated by the program. A prime case where this occurs would be

$$190 \text{ XX} = \text{Tan}(X) + CX: \text{ YY} = 2 * X * Y + CY.$$

If the more generic window size of $-2 < X < 2$, $-2 < Y < 2$ is used then the program crashes for the simple reason that $\text{Tan}(X)$ is undefined at $-\pi / 2$ and $\pi / 2$. So instead one can use a window with X boundaries avoiding those numbers. To correct this problem we can input something like $-1.72 < X < 1.72$, and $-2 < Y < 2$. This works well, and retains most if not all of the behavior of $\text{Tan}(X)$.

The third possible source of difficulty occurs when functions are undefined right in the middle of the optimum viewing window. For instance if the line;

$$190 \quad XX = X*X - Y*Y + CX : YY = 2/X*Y + CY$$

is used with a totally unaltered program then it will crash while trying to generate the picture. However if we change the starting value for the iterations slightly by altering line 183 to look something like;

$$183 \quad X=.0001 : Y=0,$$

the program will work. This trick could come in handy for other functions that are undefined at the origin.

Finally, after students make many mutations, it should be possible for them to organize the different images into families with various orders, classes, phyla, and kingdoms. Thus it becomes possible to classify the new images as animal creatures as they evolve from the original Mandelbrot set in the form of a family tree.

This completes our discussion of mutations of the Mandelbrot set.

6. FINAL REMARKS

The diamond shaped mutation shown in Figure 2 has received some attention in the literature ([3] and [5]). This is because the altered line 190

$$190 \quad XX=X*X +Y*Y +CX: YY= 2*X*Y + CY$$

is obtained naturally when the complex numbers are replaced by spacetime numbers.

Spacetime numbers are not well known except to specialists in Clifford algebras.

However two references are available that make them accessible to even precalculus students. These are [1] and [6]. Spacetime numbers are also called hyperbolic numbers and are a fascinating subject in themselves that can be used to simplify the study of special relativity.

REFERENCES

- [1] Borota, N., Flores, E. and Osler, T. J., *Spacetime numbers the easy way*, Mathematics and Computer Education, 34(2000), pp. 159-168.
- [2] Devaney, Robert L. , *Chaos, Fractals and Dynamics*, Addison-Wesley, New York, 1990.
- [3] Metzler, Wolfgang, *The “mystery” of the quadratic Mandelbrot set*, Am. J. Phys., 62(1994), pp. 813-814.
- [4] Perry, Greg, *Sams’ Teach Yourself Visual Basic 5 in 24 Hours*, Sams Pub., Indianapolis, Indiana, 1997.
- [5] Senn, Peter, *The Mandelbrot set for binary numbers*, Am. J. Phys., 58(1990), p. 1018.
- [6] Sobczyk, G. *The hyperbolic number plane*, The College Mathematics Journal, 26 (1995), pp. 268-280.