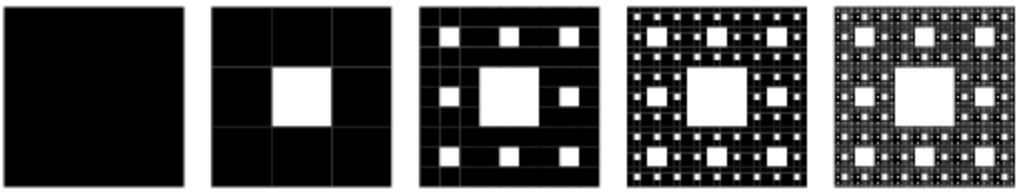
Nathan Romriell
Final Project

Fractals and the Importance of Linear Algebra in Computer Graphics

Introduction to Fractals:

Fractals from a non-mathematical point-of-view are a series of repeating shapes that appear over and over again potentially repeating into infinitely. From a mathematical point-of-view fractals are a mathematical set causing a repeating pattern. If you are looking to create fractals as a generated image there are several ways to go about it.

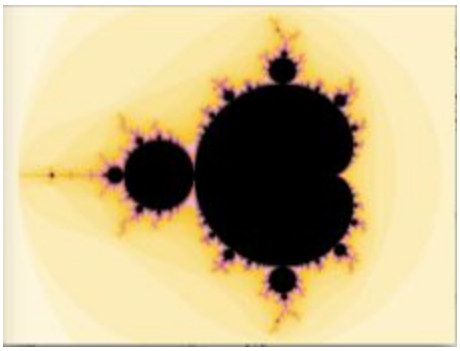
One way of creating fractals is to duplicate an existing image and translate it in a repeating pattern. An example of this method is the fractal call Sierpinksi carpet. The general form for the Sierpinksi Carpet is to take a copy of the original image scale it to one third of its original size and then translate its image to the left, top left, top third, top right, right, bottom right, bottom, and bottom left thirds of the last frame.



source: http://mathworld.wolfram.com/SierpinskiCarpet.html

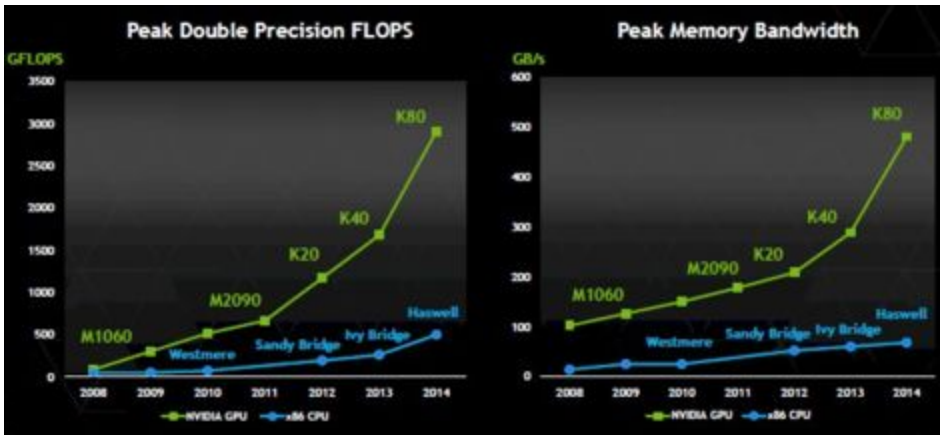
The second way of creating fractals is by using a mathematical set that can then be visualized using certain parameters. The most common of these is the Julia and Mandelbrot sets. The Julia and Mandelbrot sets are a set of fractals in the form of Z(n+1) = Z(n)^2 + C. This equation is given in the complex form Z(0), Z(n), and C are all complex numbers. To make this more readable for conversion to x,y space gives you z = x+yi where x is the real part of the number and y is the imaginary. When you put (x+yi) into the equation for the Mandelbrot set you get $(x+yi)(x+yi) = x^2 – y^2+2xyi$. If you break this into the real and imaginary parts, then you get that the real part equals $x^2 – y^2$ and the imaginary equals 2xyi. To draw the Mandelbrot on a computer you use how quickly the iterations of each point escape the disk of radius 2. The exciting parts you think of when looking at a Mandelbrot or Julia set are actually not a part of a set but part of the surrounding area which quickly escapes the closed disk of radius 2.

Basic Mandelbrot



Linear Algebra and Computer Graphics

Linear Algebra for the use with computer graphics is extremely important. If you imagine the entire xy visual plane of your computer screen as a giant matrix of pixel points height rows x width columns you can see why. Video games and 3D graphics in general are formed by applying transformations to a series of points presented to the user. This is also very important because as we have started to reach the max power input and processor speed for serial CPU processing. Our only way to continually increase power to produce better graphics and perform more complex algorithms in faster time is to use parallel processing spreading out the work between many processors simultaneously. GPU processors are the best equipped to handle processes like these with up to thousands of separate computing units that can perform operations simultaneously. From this graphic below that was presented by Nvidia shows just how much faster Graphics Processing Units are than standard CPUs.



source: http://www.enterprisetech.com/2014/11/17/nvidia-doubles-tesla-gpu-accelerators/

In this graphic you can see that a top of the line GPU is capable of computing more than 2.5 trillion more floating-point operations per second.

This is important because linear algebra allows you to break large computations down into much smaller pieces. As a simple example you could use a 3x3 matrix that you were then going

to reduce to row-echelon form. By hand or through serial processing you would have to find a pivot column and then perform an operation to reduce the other columns a single entry at a time so to reduce from the first pivot you would have to perform up to 6 operations one at a time. By running each of the operations at the same time you have now performed the same function in 1/6 the time.

Application to Fractal Display

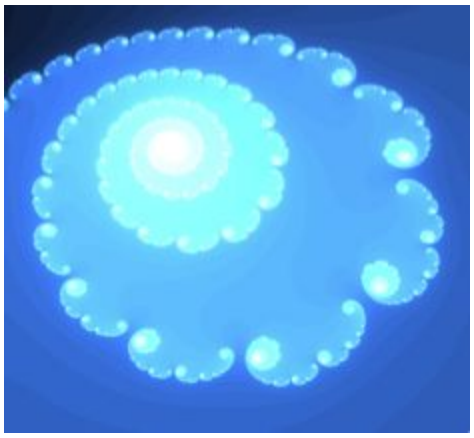
This ability to break down functions into smaller pieces becomes extremely apparent when you are trying to produce fractal images. A Mandelbrot or Julia set fractal has to perform an operation for each potential pixel on the screen. If you run this on serially on a CPU even on the fastest computers, you will find that there is a delay in producing that image. This is because you have to run through max iterations of into the hundreds for both real and imaginary parts for each pixel. For a 640-pixel by 640-pixel screen this is something around 800,000 separate operations. Since the Mandelbrot and other mathematical set fractals are essentially linear transformations in the real and imaginary planes you can apply that transformation to each point in Euclidian space in parallel without affecting other pixels drastically reducing computation time.

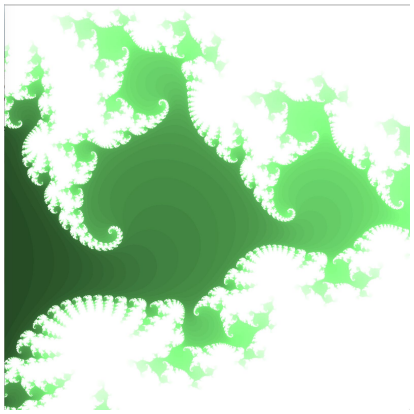
Drawing Fractals Julia Set

A Julia set is a subset of the Mandelbrot and can produce a much wider series of images with little change to the C real and imaginary components. This is why I have chosen this set to produce fractal images.

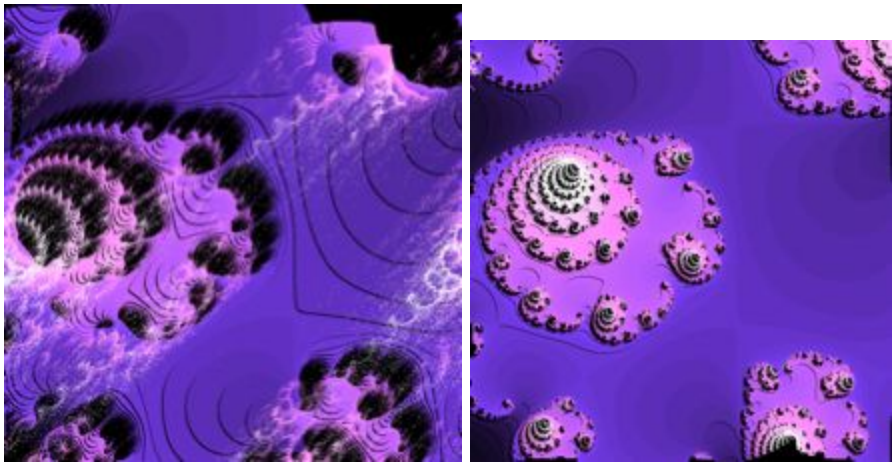
The basic code for producing a Julia set fractal is:
- Loop through each pixel
- For each pixel start with the starting positing for the x,y coordinate of that pixel plus transformations(such as scale) where x is the starting real number and y is the starting imaginary number
- Iterate through from the start values until (1) the iteration escapes the radius of 2 or (2) the max iteration is reached
- Color that pixel based on how fast it escaped the radius of 2

Generated Fractals from created program using OpenGL:





You can also apply interesting behaviors to fractals using linear transformations. For example these 2 dimensional fractals gain an interesting 3 dimensional appearance when you apply a transformation in the z axis based on the escape velocity of the point.



From the left the starting fractal, the right is a 2 point shift upwards, and the bottom is a 0.3 point shift down at max escape velocity.