Recursive Graphics

- Sierpinski Triangle - Koch Snowflake - Hilbert Curve -

By: Kiana Ross & Nick Goltsos

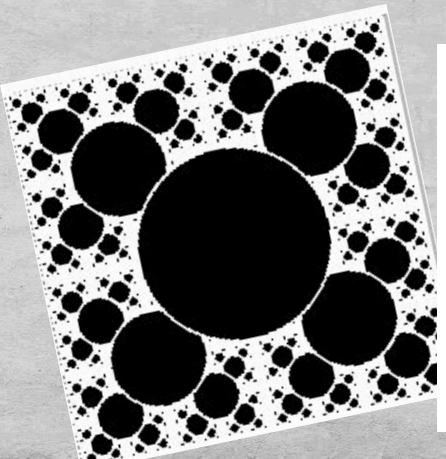
Github repository link

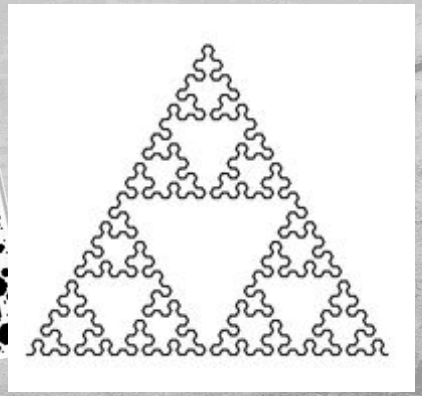
https://github.com/kianaross/CSC212Finalproject

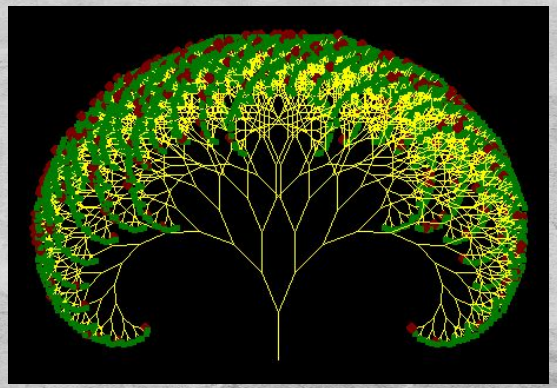
What is a Recursive Fractal?

- Recursive fractals vary in complexity.
- A fractal is a never-ending, recursion driven dynamic pattern created by repeating a simple process over and over in an ongoing feedback loop.
- The recursive calls create graphics by drawing the same image in different sizes and angles, layering to create a more complex image.

**below are many examples of recursive graphics*





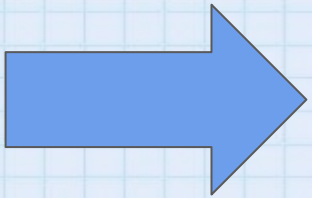


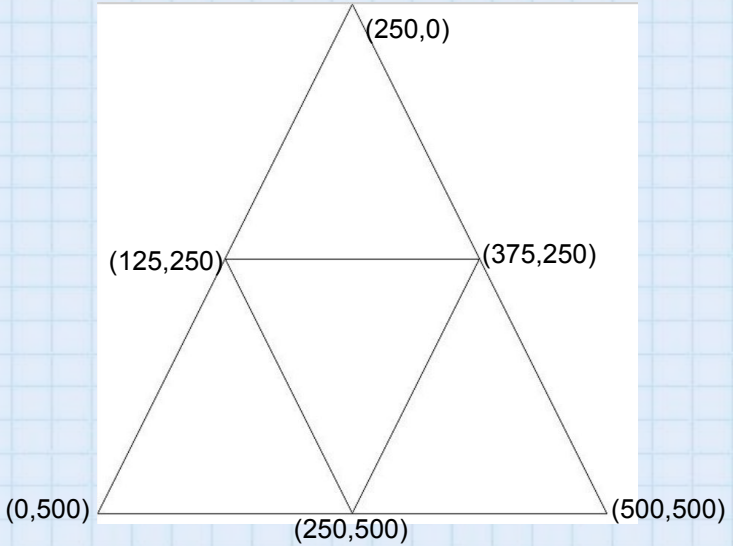
How To Draw Fractals?

Option 1: Use coordinate points to mark vertex's of the fractals and then draw lines from one point to the other using a python program

A .txt file containing these lines/points……..Would output this image

```
1 250 0 0 500
2 250 0 500 500
3 0 500 500 500
4 125 250 375 250
5 125 250 250 500
6 375 250 250 500
```

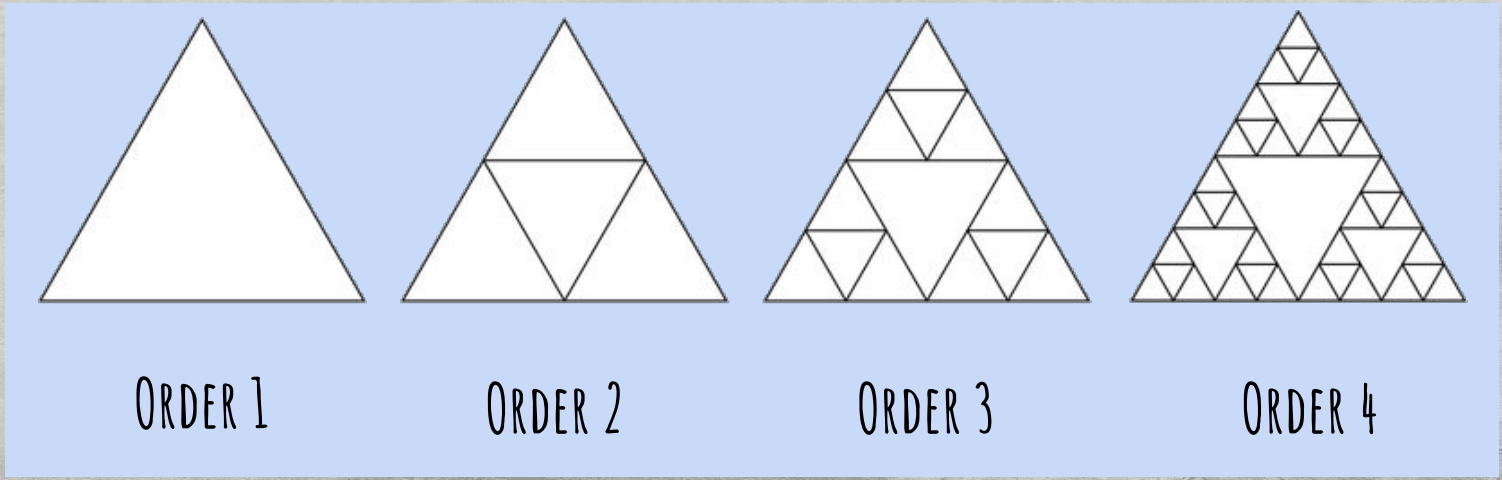




Sierpinski Triangle

HOW TO CREATE (Algorithm)

1. Begin by drawing initial equilateral triangle (with given vertices)
2. Create 3 smaller equilateral triangles inside the original, using the original vertices and their midpoints (ex. (x1+x2)/2, (y1+y2)/2)
3. Repeat process for all right-side up triangles (recursive step)



Sierpinski Triangle - Methods

CODE STRUCTURE

- Use fractal drawing option on the previous slide.
- 6 command line arguments (x1, y1, x2, y2, x3, y3),the vertices of starting triangle
- Use void functions, vectors, and integer points to find midpoints, create new vertices, and store these numbers.
- Write from a vector to the out file (.txt)
- To print image, use a python program and pass the your txt file with the data into this.
- Result is a PDF with the image

Void Triangle function(Recursion happens here)

- Base case- if length of triangle <10, stop
- Else- Call draw function (x3), pushback 4 new vertex points.
- Recursively call triangle function, with 2 new midpoints and 1 prior vertice (3 times to account for each new triangle).

Void Draw function(called in Triangle Function)

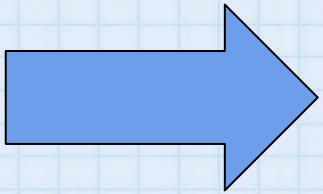
- takes 4 points and a vector (passed by reference as parameters
- set the 4 points to the first 4 values in vector (the 4 points to draw each line)

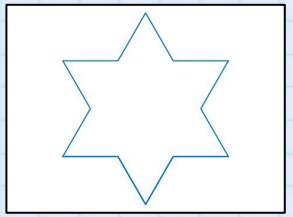
How To Draw Fractals?

Option 2: Use L-Systems, a process of creating instructions which state (F) draw a line (-) turn to the left by some degree (+) turn to the right by some degree. Operates by having a “turtle” walk in a space knowing its current position and orientation, as well as the next instruction. Involves replacement.

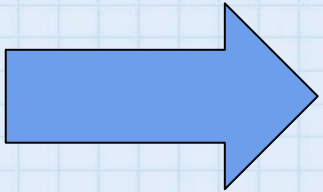
These l-system operations ……..... Print these images

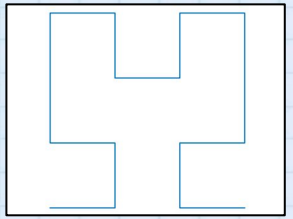
F-F++F-F++F-F++F-F++
F-F++F-F++F-F++F-F++
(with 60 degree turns)





+-F+F+F-F-+F-F-F+F+F-F-F
+-F-F+F+F-+
(with 90 degree turns)

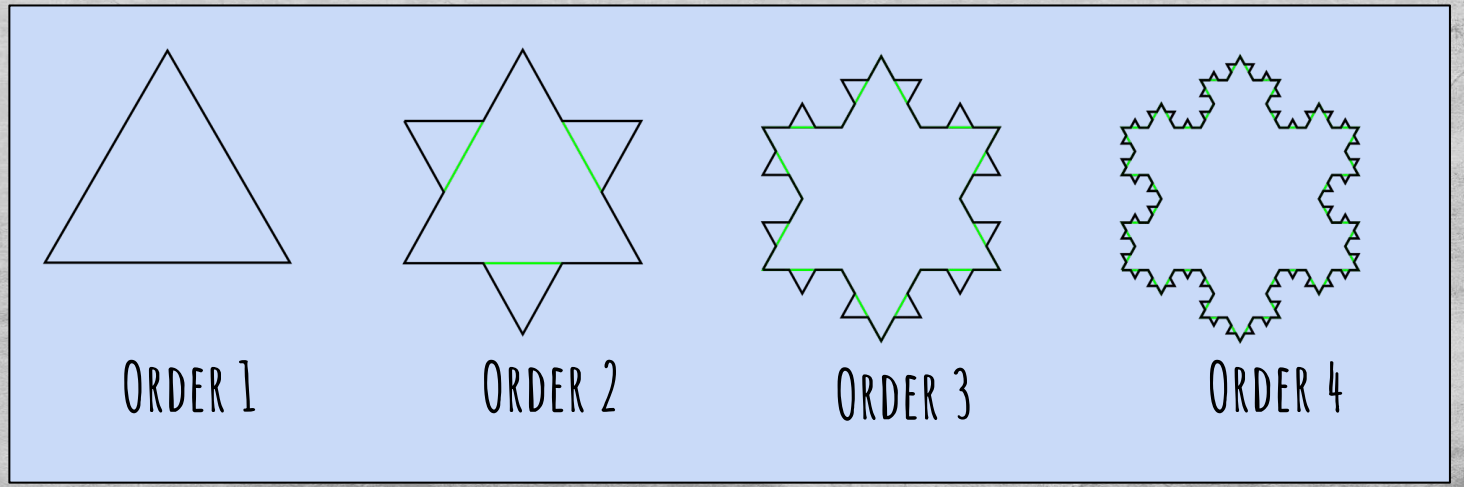




Koch Snowflake

HOW TO CREATE (Algorithm)

1. The Koch Snowflake begins as an equilateral triangle
2. The following iterations are formed by splitting segments into thirds and repeatedly adding smaller triangles in the middle third.
3. The snowflake can be split into 3 equivalent sides, thus we focused on creating 1/3 and then replicating through a 120 degree right hand turn.



Koch Snowflake - Methods

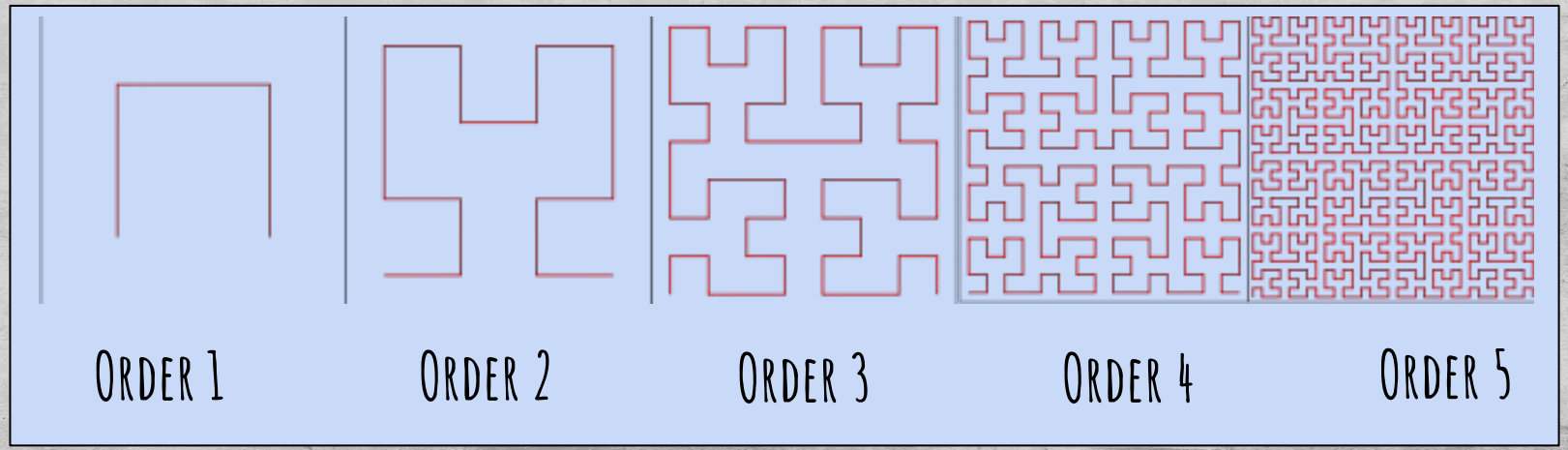
CODE STRUCTURE

- Created using the L-systems options for drawing fractals which was discussed prior.
- The L-system initial string used F->F-F++F-, after each iteration rewrite with F as F.
- **Main function**- takes order as a command line argument and prepares to output info. Calls koch_snowflake()
- **String koch_snowfake function-** function takes order as parameter and returns a string with all the commands. Use a loop to draw the 3 sides of the snowflake, calling snowflake() and making a 120 degree right turn after each iteration.
- **Void snowflake**(recursion happens here)- takes order and string (passed by reference) as parameters.
    - Base case- if order of 0, return
    - Else- append the commands from out string above. Before each line drawn, call snowflake recursively fr order-1.
- PDF image created using python file where user enters degree and output file is read

Hilbert Curve

HOW TO CREATE (Algorithm)

1. The Hilbert Curve initially begins as an open box (open down in this case)
2. This box is then replicated 3 ways so there is a box in each quadrant
   a. across y axis (that of the closed sides)
   b. across the x-axis and turned right 90 degrees
   c. Across x, moved right and turned left 90 degrees
3. These replicas are connected and the process is then repeated recursively



Hilbert Curve - Methods

CODE STRUCTURE

- Created using L-systems option discussed prior.
- Unlike Koch snowflake, where we only use one string of commands, Hilbert curve requires 2.
  - A->+BF-AFA-FB
  - B->-AF+BFB+FA-
- Main function- receives order from user, calls string hilbertcurve function and writes result to output file.
- **String hilbertcurve function-** add moves for A to a string, calling astring function and bstring function where an A or B appears.
- **Void astring & bstring -** take order and string passed by reference as parameters
  - Base case- order = 1, return
  - Else- append commands for the string, calling astring and bstring functions accordingly for order-1.
  - PDF image created using python file where user enters degree and output file is read

Now we'll run our programs so you can see for yourself