COSC 231 Program #4 GUI for Fractals

Distributed: 16 November 2011 **Due**: 12 April 2011-

Write a GUI that will display, with certain user-controlled parameters, any or all of the following:

1 the Sierpinski gasket,
2 the 'dragon' fractal (http://en.wikipedia.org/wiki/Dragon_curve)
3 any other space-filling curve (http://en.wikipedia.org/wiki/Space_filling_curve)
e.g., Hilbert curve, Peano curve
4 the Mandelbrot set,
5 any Julia set (http://en.wikipedia.org/wiki/Julia_set),

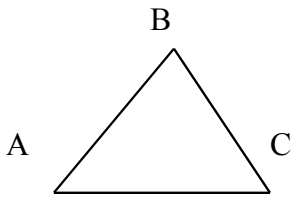
1 – 3 are linear fractals, you are drawing a line. 4 – 5 are continuous: each point on the drawing surface is given a value, and all values are updated at every iteration.

Extra credit is given for implementing more than one of the above five options.

Sierpinski triangle (gasket)
The Sierpinski triangle generation algorithm is given in Savitch's book (3rd edition) on pp 1061-2. That text is reproduced here:

Write a program that draws a Sierpinski gasket. A Sierpinski gasket or triangle is a type of fractal. It is an example of how an orderly structure can be created as a result of random, chaotic behavior.

The creation of a Sierpinski gasket is fairly simple. There are three points that form the corners of a triangle. In the figure below, they are labeled as A, B, and C.



To draw the Sierpinski gasket follow the algorithm:

1. Randomly pick one of the three corners. Call this the current point. (Alternatively, you could also randomly select any point inside the triangle).

2. Randomly pick one of the three corners. Call this the target point.

3. Calculate the midpoint between the current point and the target point.

4. Draw a single pixel at the midpoint.

5. Set the current point to the midpoint.

6. Go back to step 2 and repeat many times (e.g., 100 iterations).

It would seem like you should get a random mess of dots since the algorithm randomly picks a target corner each time. Instead, if this process is repeated many times, you will get a very orderly picture with a repeating structure: < picture from Savitch is not reproduced here. Look at

http://en.wikipedia.org/wiki/Sierpinski_gasket >

To draw a single pixel at coordinate (x, y) use the `drawline` method where the start and endpoints are both (x, y).

Mandolbrot set
There are several explanations of the Mandelbrot set. A simple one, including the iterative algorithm for determining if a number is a member of the set, is given at http://www.ddewey.net/mandelbrot/
Here is another simple explanation:
http://t16web.lanl.gov/Kawano/gnuplot/fractal/mandelbrot-e.html

Each point, C, in the complex plane is a member of the Mandelbrot set if, after iterating (forever), the magnitude of Z (Z = a + i*b), magnitude(Z) = sqrt(a^2 + b^2) is less than 2.

That is, each point C has its own Z value.

Z iterates as follows:

$Z_0 = 0.0 + i * 0.0$

$Z_{n+1} = Z_n * Z_n + C = (a_n + i*b_n) * (a_n + i*b_n) + (x + i*y)$

Represent a point C = x + iy on the drawing surface as (x, y).

Khan Academy on youtube has a nice and sufficient explanation for complex numbers that will take a little over 20 minutes to view.

http://www.youtube.com/watch?v=kpywdu1afas	Complex numbers (part 1)
http://www.youtube.com/watch?v=EQviquyrDxA&feature=related	Complex numbers (part 3
http://www.youtube.com/watch?v=dWjg6fiKNOw&feature=related	Complex numbers (part 11)

/***/
So, in pseudo-code:

```
// initialize z
for (int row=0; row <=WIDTH; row++)
   for (int col = 0; col <= HEIGHT; col++) {
      (z[row][col]).real = 0;
      (z[row][col]).img = 0;
      }

// iterate for long enough
for (int t = 0; t < MAX_ITERATION; t++) {
   for (int row=0; row <=WIDTH; row++)
   for (int col = 0; col <= HEIGHT; col++) {

      compute real and imaginary values of z for point (row, col);

      if ( sqrt (
                  ( (z[row][col]).real ^2) +
                  ( (z[row][col]).img  ^2) )
                ) < 2.0) color (row,col) INSET;
      else color (row, col) OUTSET;

      }   // for int col
   }      // for int t
```

The GUI interface will have a drawing surface and the following controls:

1. Clear – to clear the drawing surface. This can be done while the curve is drawing. The curve continues from its current state (do not start the curve from the initial state).

2. Set Boundary – activate the mouse listener to select a bounding box for the curve. If the user does not choose a bounding box, then use a programmer specified default. This cannot be done while the curve is drawing. Do not allow the user to change this setting during execution of a curve.

3. Set Initial points – some curves will accept starting locations (e.g., the Sierpinski gasket can have the corners of the triangle specified by the user). If the user does not choose initial points, then use a programmer specified default. This cannot be done while the curve is drawing. Do not allow the user to change this setting during execution of a curve.

4. Choose curve (a drop down box to choose the curve being drawn). Only permit the user to choose among implemented curves. This cannot be done while the curve is drawing. Do not allow the user to change this setting during execution of a curve.

5. Start – Clears the screen and starts the current curve with the current settings.

6. Stop – Halts the current graphic generation. The current state of the fractal generation is saved so that if the user subsequently hits the Start button, the fractal generation continues from the current state. Do NOT clear the drawing surface.

7. Speed – Lets the user choose the speed of the iteration. The speed can be changed during curve generation. The slowest speed should be such that you can easily watch the curve "grow". Curve generation will not cease or reset.

7. Color choice – A user can select a colors for the fractal. (A ColorPicker to actually select colors if you wish). For line fractals (dragon, Sierpinski triangle, space-filling curve), color choice can be changed during curve generation. For continuous fractals (Mandelbrot, Julia) the color choice relates to the number of iterations

The buttons must be arranged in a rational and user-friendly way. Nicely aligned. Related functionalities grouped together in geography and in style.

Important: Any non-functional user controls should be "greyed out". The user should not be allowed to set controls that are not implemented in your code.

Deliverables:

- URL
- Hardcopy of GUI and fractal code
- Demo and code walk-thru

Grade based on:

- Satisfying the program specs given here	75%
- GUI layout	5%
- Elegance of code:	10%
- Coding standards	10%

Extra credit calculation: 5% for each additional option. The added option must work completely and correctly for extra credit.