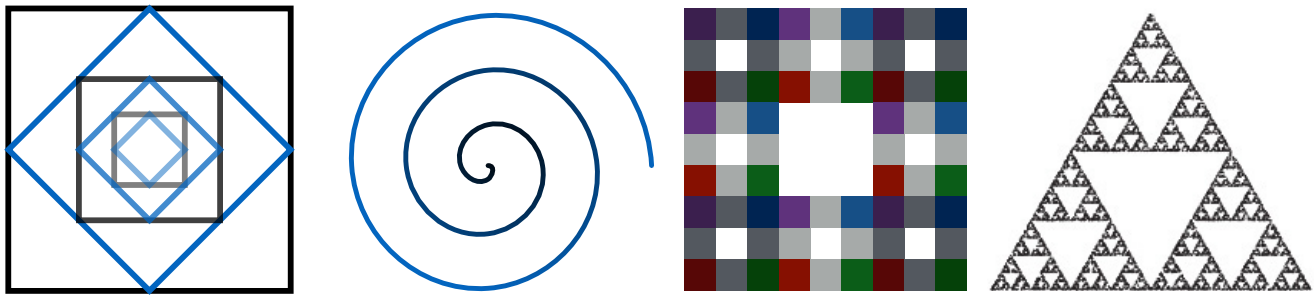
Points, Lines, and Triangles

Assignment #1

CPSC 453 • Fall 2018 • University of Calgary



Overview & Objectives

The purpose of the first assignment in CPSC 453 is to provide you opportunities to familiarize yourself with C++ programming and the OpenGL environment by drawing simple, yet interesting two-dimensional geometry. Source code for a minimalistic and pedagogical boilerplate C++/OpenGL application is provided for you to use as a starting point, though you may create your own template if you prefer.

The most important outcome is to understand how algorithms (shaders) operate on data (buffers) in the OpenGL system to generate images, and how to set these objects up to work together within the OpenGL and GLFW framework. The bulk of your work will be to write code in C++ to generate the correct geometry for the polygonal, parametric, and fractal shapes you're asked to draw. Only minimal modification of the shaders and OpenGL function calls provided in the template will be necessary.

This assignment consists of a written component and a programming component. The programming part is described first, but **note that written answers to the assignment questions are due ahead of the program submission.** You may work on both components with a single partner of your choosing, but remember that you may not work with the same partner for any future assignments in this course.

Due Dates

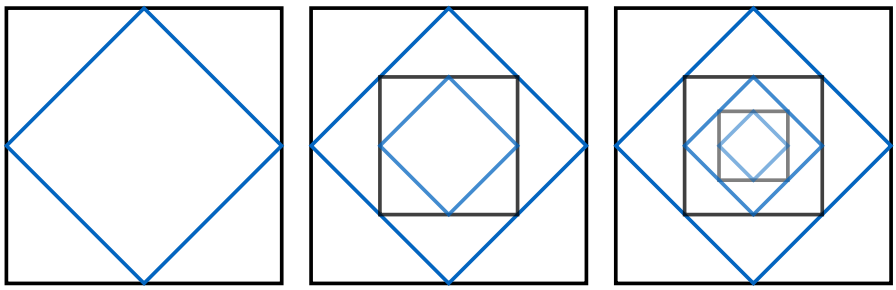
Written component	**Monday, September 17 at 11:59 PM**
Programming component	**Tuesday, September 25 at 11:59 PM**

Programming Component

There are a total of four parts to this programming assignment with an additional requirement to integrate the four scenes into a single application. This component is worth a total of 20 points, with the point distribution as indicated in each part, and an addition possibility of earning a "bonus" designation. Since the bonus is a binary state, it will only be awarded to submissions that do an exemplary job of meeting the requirements.

Part I: Squares & Diamonds (4 points)

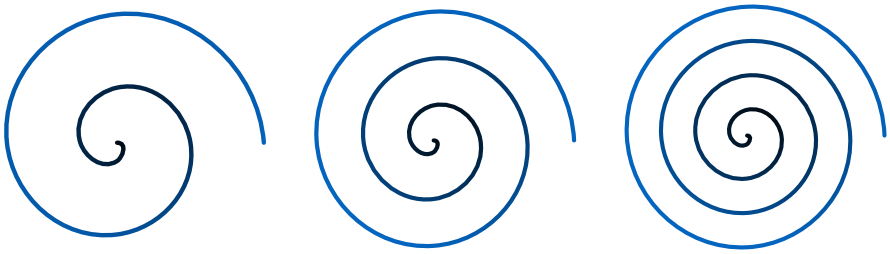
Create a program, or modify the provided template, to draw a pattern of nested squares and diamonds as shown in the diagram below. Choose different colours for the squares and diamonds, and alter the shade of the colours between geometry at different nested levels. Use keyboard input to provide a means for interactively changing the levels of nesting drawn (*i.e.* number of squares/diamonds). Allow for at least 6 levels of nesting.



Level 1 Level 2 Level 3

Part II: Parametric Spiral (4 points)

Add the ability to draw a second scene, a spiral, to your program from Part I. Construct your spiral with a colour gradient so that the colour of the curve at the centre of your window smoothly transitions to a different colour at the edge as you trace the curve's path, as shown below. Provide a means for interactively changing the number of revolutions the spiral makes as it goes from the centre to the edge of the window.

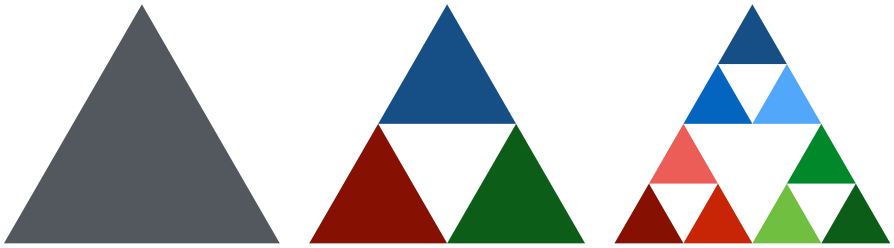


2 Revolutions 3 Revolutions 4 Revolutions

Part III: Sierpinski Triangle or 2D Menger Sponge (4 points)

Add a third scene to your program to draw a Sierpinski triangle or a two-dimensional Menger sponge. (You may do both if you'd like.) Both are well-known fractal shapes with very interesting mathematical properties.

Start with an equilateral triangle for the Sierpinski triangle and centre it in your window. The next iteration of the Sierpinski triangle is generated by splitting the triangle into three smaller, equally-sized triangles, with side lengths half of that of the original, and placing them within the space of the original so that their corners just touch, as shown below. Repeat this process for all triangles in the fractal to generate additional levels.



Base Triangle 1 Iteration 2 Iterations

The Menger sponge is constructed in a similar fashion, but by starting with a square. The next iteration is generated by splitting the square into eight smaller squares, each with sides one-third the length of the original, and placing them on the inner perimeter of the original. The result should look as though a single small square were removed from the centre. Again, additional levels of the fractal are generated by repeating this process.

Base Square 1 Iteration 2 Iterations

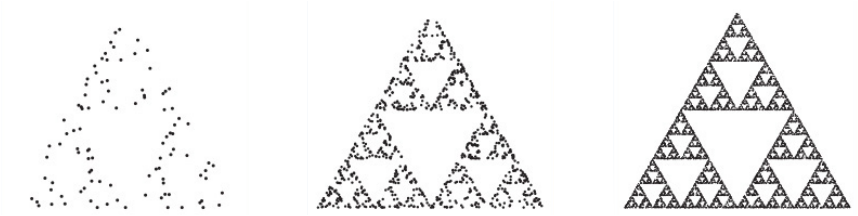
Assign colours to your triangles or squares to distinguish them from each other, but choose your colours in such a way that your final fractal image is aesthetically pleasing. Provide a means to interactively change the number of fractal iterations applied, and allow for at least 6 levels of division.

Part IV: Sierpinski Triangle Reloaded (4 points)

It may surprise you that the Sierpinski triangle can be drawn in a completely different manner, through a random process! Consider the following process:

1. Choose any one of the three vertices of the equilateral triangle and plot it.
2. Take another of the three original vertices at random, find the midpoint between it and your last position, and plot that midpoint.
3. Take the midpoint as your new position and repeat from step 2.

After running this process over hundreds or thousands of iterations, you should see something that resembles the diagram below.



A Few Iterations Many Iterations Lots of Iterations

Add a fourth and final scene to your program to draw the Sierpinski triangle using points with this iterative random process. You will no longer be able to control the recursive level of the triangle. Instead, provide a means to interactively adjust the number of points drawn, effectively controlling how sparse the triangle looks. You may use any colours you like.

Integration & Control (4 points)

Use keyboard input to provide a means for switching between the three scenes in your program. Ensure that your program does not produce rendering anomalies, or crash, when switching between scenes or numbers of iterations in each scene. Efficiency is important too! Programs that fail to clean up resources or generate excessive amounts of unnecessary geometry may bog down the system or eventually crash, and will not receive full credit.

Bonus Part: Fractal Geometries

Add two additional scenes to your program: a Barnsley fern and a Hilbert space-filling curve. As before, provide a means for interactively changing the number of points or fractal iterations drawn. Choose colours to make both scenes aesthetically pleasing. If you can think of other interesting shapes or scenes that you would like to explore, fractal or otherwise, feel free to include them. The bonus designation will only be awarded to submissions that do an exemplary job of rendering these additional scenes.

Submission

We encourage you to learn the course material by discussing concepts with your peers or studying other sources of information. However, all work you submit for this assignment must be your own, or explicitly provided to you for this assignment. *Submitting source code you did not author yourself is plagiarism!* If you wish to use other template or support code for this assignment, please obtain permission from the instructors first. Cite any sources of code you used to a large extent for inspiration, but did not copy, in completing this assignment.

Please upload your source file(s) to the appropriate drop box on the course Desire2Learn site, and indicate any late days used here. Include a "readme" text file that briefly explains the keyboard controls for operating your program, the platform and compiler (OS and version) you built your submission on, and specific instructions for compiling your program if needed. If your program does not compile and run on the graphics lab computers in MS 239, *the onus is on you to ensure that your submission runs on your TA's grading environment for your platform!* Broken submissions will be returned for repair at a minimum penalty of one late day, and may not be accepted if the problem is severe. Ensure that you upload any supporting files (*e.g.* makefiles, project files, shaders, data files) needed to compile and run your program, but please do not upload compiled binaries or object files.

Your program must also conform to the OpenGL 3.2+ Core Profile, meaning that you should not be using any functions deprecated in the OpenGL API, to receive credit for this part of the assignment. We highly recommend using the official OpenGL 4 reference pages as your definitive guide, located at: https://www.opengl.org/sdk/docs/man/.

Assignment Questions

The written component consists of three questions, each with several parts, corresponding to the parts of the programming assignment, and worth 10 points in total. When answering the questions, show enough of your work, or provide sufficient explanation, to convince the instructors that you arrived at your answer by means of your own.

Question 1: Squares & Diamonds (3 points)

A. If your outermost square has a side length of 1.0, what is the side length of the diamond nested immediately inside?

B. What is the side length of the square nested immediately inside the diamond of 1A?

C. If your outer square is drawn 1000 pixels wide on your display, how many levels can you draw in this nested pattern before the smallest square is less than one pixel wide?

Question 2: Parametric Spiral (3 points)

A. Write a parametric equation for a spiral curve that starts at the origin and makes three complete turns as it moves radially outward. Express x and y as functions of a parameter u, and indicate the appropriate range for values of u.

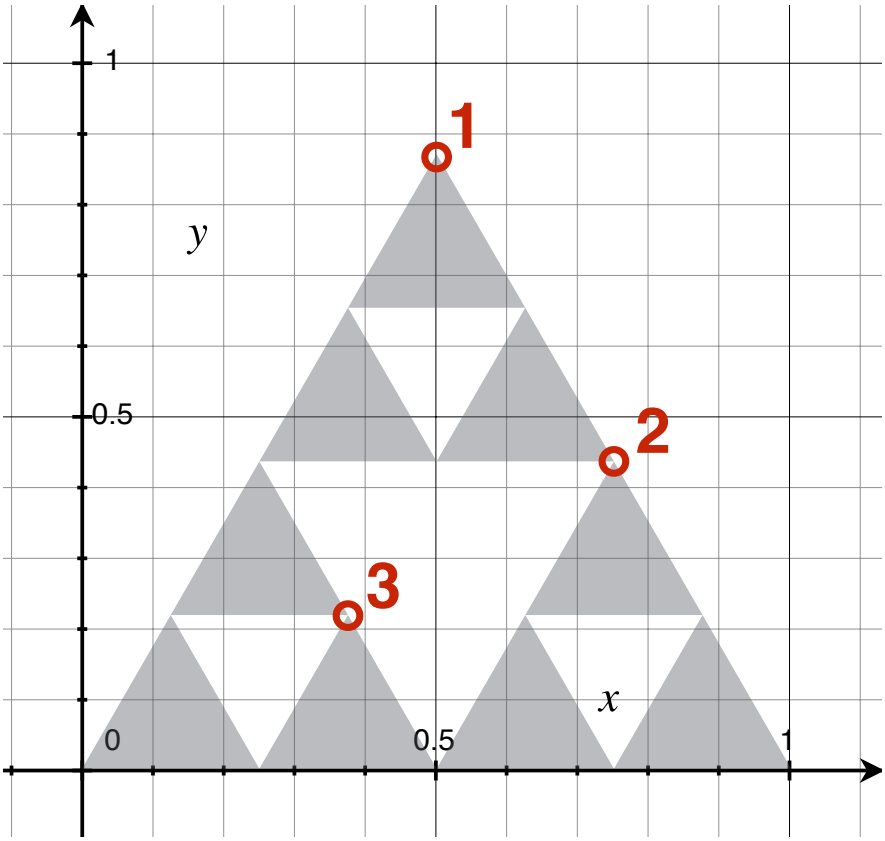
B. Recall that the most effective method for drawing a curve on a raster display using OpenGL is to approximate it with line segments. If you were to approximate a 1000-pixel diameter circle using equal-length line segments, how many segments would you need to draw to ensure that you are no more than one pixel off the true circle?

C. If the spiral of 2A were drawn to fill a 1000×1000 pixel window, what is the minimum number of line segments needed to approximate the spiral to an accuracy of one pixel? (It is incredibly difficult to arrive at an exact answer for this question, but full credit will be given to reasonably accurate answers.)

Question 3: Sierpinski Triangle & Menger Sponge (4 points)

A. Examine the Sierpinski triangle on the next page. It originated from an equilateral triangle of side length 1.0 with its bottom-left corner at the origin, (0,0). Calculate the positions, as (x,y) coordinate pairs, of the vertices numbered 1, 2, and 3 in the diagram.

B. If your system is capable of storing and rendering a maximum of 1 000 000 (one million) squares, how many iterations of the 2D Menger sponge can you draw before you exceed your maximum capacity?



You may submit your answers in digital form (typed or scanned) in the appropriate drop box on the course Desire2Learn site, or as hard copy directly to your TA in the tutorial of or before the due date. Remember that late days may not be used for the written component!