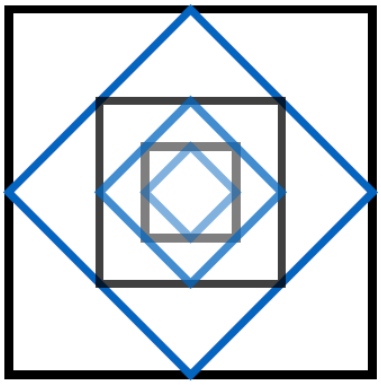
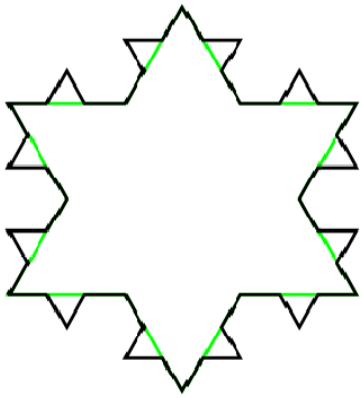
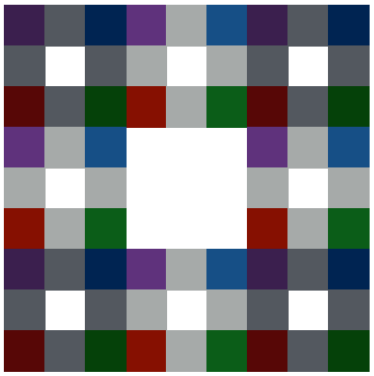
CPSC 453 Assignment 1 Points, Lines and Triangles *

Fall 2021, University of Calgary

1 Overview and 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 to create geometry in your C++ program and send that data to the GPU to be rendered using 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.

*Assignment specification and imagery taken from previous offerings of CPSC 453 by Sonny Chan and Faramarz Samavati. Please notify drlarsen@ucalgary.ca of typos or mistakes.

2 Due Date

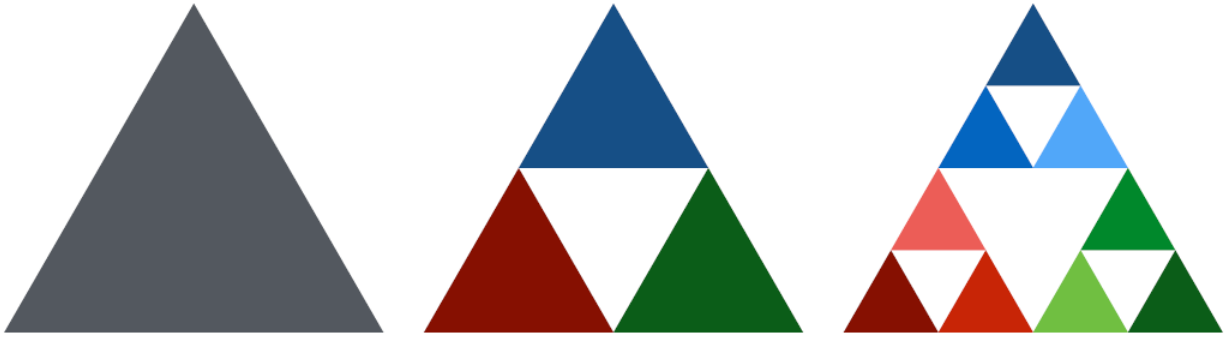
October 3rd at 11:59 PM

3 Programming Assignment

There are a total of three parts to this programming assignment with an additional requirement to integrate the scenes into a single application. This assignment is worth a total of 20 points, with the point distribution as indicated in each part, and an additional possibility of earning two "bonus" designations. Note that the bonuses may require going beyond what is presented in the lecture and tutorial components of the course and may require student initiative to find online resourses to complete successfully.

3.1 Part I: Sierpinski Triangle or 2D Menger Sponge (4 Points)

Create a program, or modify the provided template, 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.

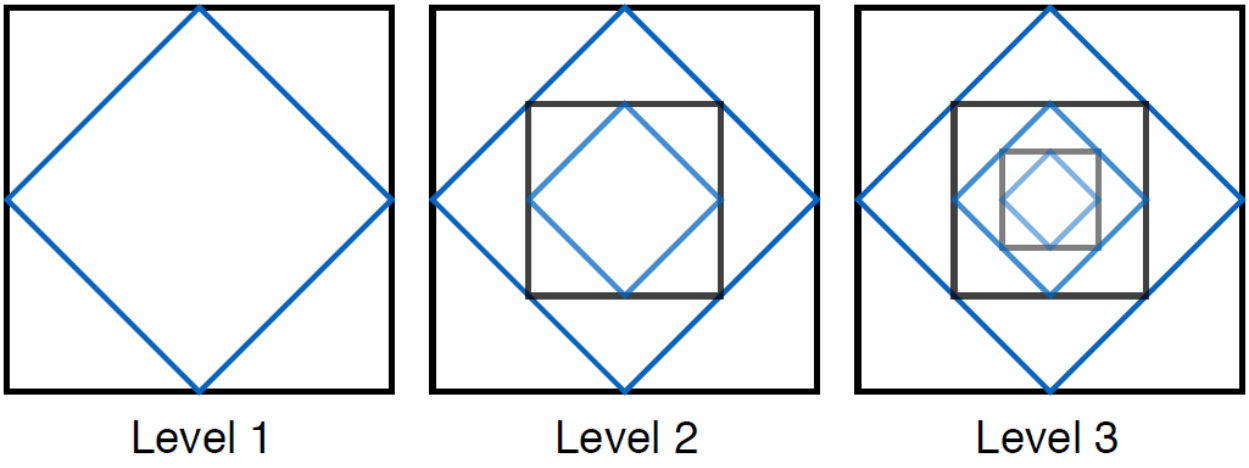


The Menger sponge is constructed in a similar fashion, but by starting with a square. The next iteration is generated by splitting the square into eight smaller squares, each with sides one-third the length of the original, and placing them on the inner perimeter of the original. The result should look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

Assign colours to your triangles or squares to distinguish them from each other. Provide a means to interactively change the number of fractal iterations applied, and allow for at least 6 levels of subdivision.

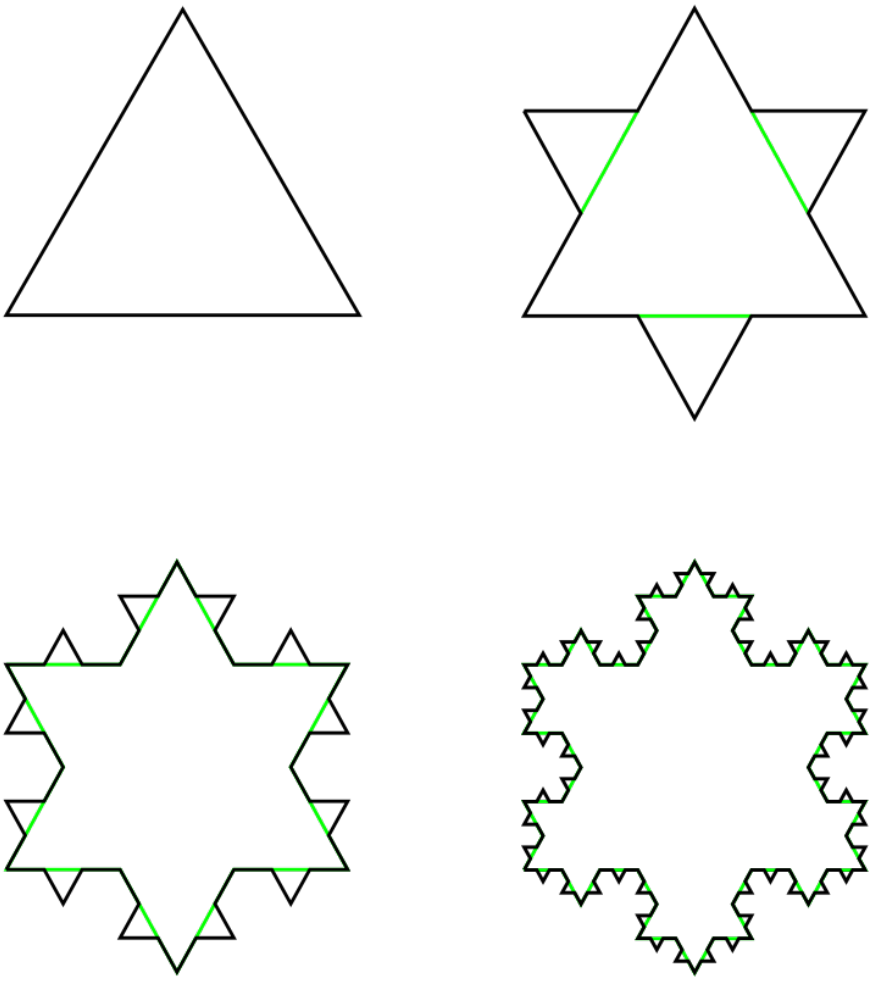
3.2 Part II: Squares and Diamonds (4 Points)

Add a scene 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.



3.3 Part III: Koch Snowflake (4 Points)

Add the ability to draw a third scene, a koch snowflake, to your program. Choose any colors you like for this final fractal. Provide keyboard controls for interactively changing the number of subdivision levels of your dragon curve. Note that the dotted lines in the diagram are to give you an idea of how the geometry is created and do not need to be drawn in your program.



3.4 Part IV: Dragon Curve (4 Points)

Add the ability to draw a fourth scene, a dragon curve, to your program. Choose colors that allow you see the added pieces of geometry as you increase subdivision levels (not shown in the figure). Provide keyboard controls for interactively changing the number of subdivision levels of your snowflake. Note that the green lines in the diagram are to give you an idea of how the geometry is created and do not need to be drawn in your program.

3.5 Integration and Control (4 Points)

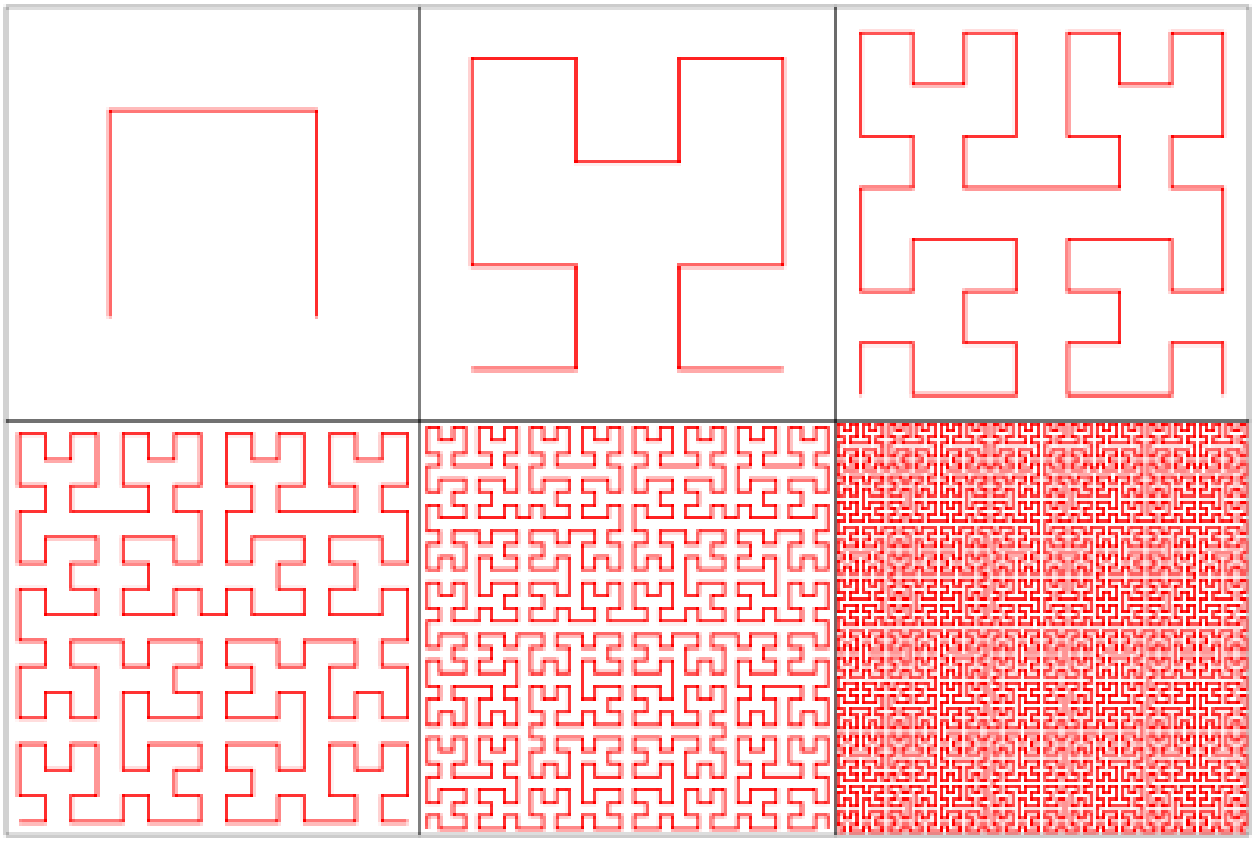
Use keyboard input to provide a means for switching between the three scenes in your program. Ensure that your program does not produce rendering anomalies, or crash, when switching between scenes or numbers of iterations in each scene. Efficiency is important too! Programs that fail to clean up resources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

3.6 Bonus Part I: GUI Integration (2 Points)

Add a graphical user interface to your program as a way to change scenes and subdivision levels within scenes (however, continue to allow keyboard controls for full credit for the non-bonus portion of the assignment). For this bonus, downloading a C++ library that provides tools for creating the GUI is recommended. For full credit, you can download the GUI library, integrate it into your code and use the API provided by the library to draw your GUI elements. It is recommended that you use Imgui as your GUI library (`https://github.com/ocornut/imgui`). You have to look through the documentation provided on the website for an understanding of how to integrate it into your program and how to use the API.

3.7 Bonus Part II: Fractal Geometries (4 Points)

Add an additional scene to your program, a Hilbert 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. Completing this bonus may require some additional online research to understand how the Hilbert Curve is defined in order to produce the geometry in your program.



4 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. 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. In general, the onus is on you to ensure that your submission runs on your TA's grading environment for your platform! It is recommended that you submit a test assignment to ensure it works on the environment used by your TA for grading. Your TAs are happy to work with you to ensure that your submissions run in their environment before the due date of the assignment. Broken submissions may be returned for repair 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. 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/.