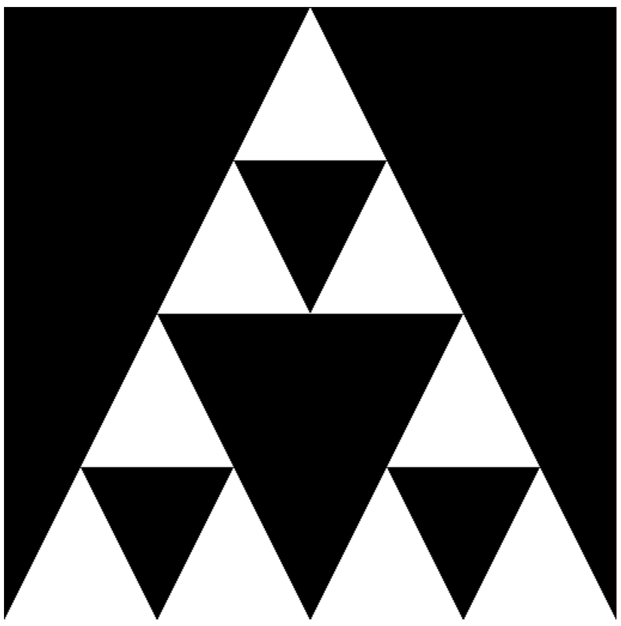
INF3320 - OBLIGATORY EXERCISE 1

SIERPINSKI GASKET

BART SIWEK, REVISED BY ANDRÉ R. BRODTKORB

Revision: 24/08/2014



In this exercise we will construct and display a approximation to a Sierpinski Gasket fractal. The main objectives of this exercise are setting up OpenGL environment, constructing the geometry of Sierpinski Gasket and displaying it.

THE SIERPINSKI GASKET

The Sierpinski Gasket is an example of a simple fractal. One of the ways of constructing it is to start with a triangle and then proceed as follows:

1. Split each edge of the triangle in half to obtain a new point halfway between the edge endpoints. Connect that newly obtained points with each other. This way we have split our original triangle into four smaller ones (one is upside down).
2. Remove the small triangle in the middle.
3. Apply the algorithm to each of the three smaller triangles left.

From the mathematical standpoint of view nothing prevents us to apply the above algorithm infinitely many times to obtain the Gasket, however due to the finite nature of the computer we must stop our calculations at some point. If we consider the original triangle to be the level zero of our approximation to the Gasket, then one application of the above algorithm will yield level one approximation, two applications will yield level two approximation and so on. In our code we will use a global variable `level` that will denote the approximation level our application is suppose to display.

The Application

The goal of the application is as follows - compute the Sierpinski Gasket approximation of level zero, place the results in a Vertex Array Object and corresponding Vertex Buffer Objects and use them to display the result. Element array buffer (also known as index buffer) will also be required in order to avoid duplication of geometry data. When user presses the '+' button on the keyboard the application needs to increase the `level` variable by one, recompute the approximation to the gasket accounting for the new level, transfer new data to the Vertex Buffer Objects and display the result. When user presses the '-' button on the keyboard the application needs to decrease the `level` variable by one and then perform the same actions as if user pressed the '+' button.

A following Vertex structure is suggested:

```
struct Vertex {
  Vertex(float x, float y)
      : position(x, y) {
  }

  Vertex(const glm::vec2 &p)
      : position(p) {
  }

  glm::vec2 position;
};
```

Details and Hints

Use the `std::vector<Vertex>` to store vertices and `std::vector<unsigend int>` to store the indinces into vertex vector - this way you can easily describe the geometry without duplications (remember to specify the vertex indices in a counter clockwise order).
Lookup the following functions and constants for reference:

- `glGenVertexArrays`
- `glDeleteVertexArrays`
- `glGenBuffers`
- `glDeleteBuffers`
- `glBindVertexArray`

- `glBindBuffer`
- `glBufferData`
- `glEnableClientState`
- `glVertexPointer`
- `GL_ARRAY_BUFFER`
- `GL_ELEMENT_ARRAY_BUFFER`
- `glDrawElements`

Use the following values as the positions of initial triangle vertices.

- $(-1, -1)$.
- $(1, -1)$.
- $(0, 1)$.

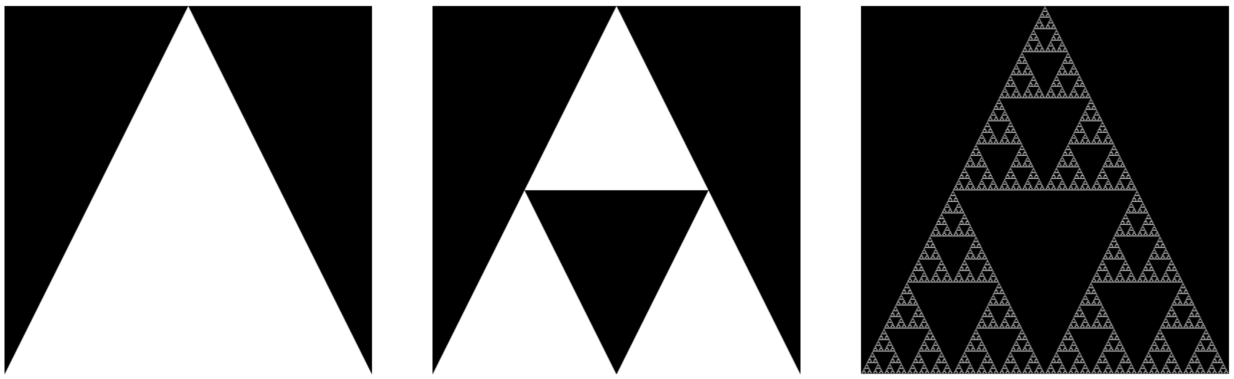
With default values for OpenGL model-view matrix and projection matrix this will produce a triangle that is fit to the window.
Finally we would suggest following approach:

1. Compile and run the template code without any changes. The template is designed in such a way that it should display a empty back window.
2. Start with a level zero of the gasket which is just a triangle. Work on setting up Vertex Array Object, Vertex Buffer Objects and displaying the initial triangle. The results should be like in the figure below.
3. Work on adding the first level of the gasket. Make sure it works.
4. Test the code with higher levels of the gasket. If everything was done correctly in previous steps this should be just a formality.

HANDING IN THE EXERCISE

The assignment is individual and everyone shall create their own program. If you choose to use code or derivations of code that is not your own, its source and author shall be explicitly cited. We will normally accept some degree of unoriginal work, but we may require that parts of the unoriginal code must be rewritten.

We reserve the right to do an oral examination of the student for each submitted exercise. The failing or passing of this examination will decide whether you pass this exercise, and this decision is final.



(a) Level 0 (b) Level 1 (c) Level 8

Figure 1: Different approximation levels of Sierpinski Gasket

The program should preferably be a C++ program, and the visualisation shall be done using OpenGL. The choice of windowing toolkit is the student's. Document your efforts by commenting the code.

All source code as well as a Makefile shall be included. The Makefile should have at least a target to build the program as well as a clean-target that deletes all object-files as well as the executable. Assume that your user name is `foo` (exchange "foo" with your user name). Put all files in a directory called `foo-1`.

Handing in the assignment is performed by creating a private repository at either GitHub (sign up as student for five private repositories on `https://github.com/edu`) or Bitbucket, and then sending an email to `mariuek@ifi.uio.no` with the clone URL and a short description detailing the files involved in your hand-in. Make sure the program compiles and runs cleanly on the computers at IFI, and that it fulfils the requirements outlined in the evaluation guidelines.

Good luck!