Lab 2

- *Shaders*: *Vertex-Shader* and *Fragment-Shader*
- GLSL – *OpenGL Shading Language*
- Basic organization of an application using *shaders*
- Rendering 2D primitives

1.1 Rendering a triangle — First *shaders* example

Analyze example **WebGL_example_06.html**.

Notice how the file contents are structured.

Identify the main changes regarding the previous examples:

- Code organization.
- Defining, compiling and linking the *shaders*.
- Calling the *shaders* and passing an array of vertices as argument to the *vertex-shader*.

Questions:

- What are the coordinates of the triangle's vertices?
- How is a color assigned to the triangle?
- How is defined the *clipping-window*?
- What happens if we change the dimensions of the *viewport*?

Suggestions:

- Render a rectangle using two triangles.
- Render a complex shape, made up of several triangles.
- Render triangles which are partially outside the *clipping-window*.

1.2 The WebGL graphical primitives

Analyze the incomplete example **WebGL_example_07.html**.

Notice how some information is presented for each WebGL graphical primitive.

Questions:

- What are the vertices' coordinates?
- What is their definition order?
- How are defined the different primitives?

Task:

- Render the several types of graphical primitives, but always using the same sequence of vertices.
- The user selects the type of primitive to be rendered: `LINES`, `LINE_STRIP`, `LINE_LOOP`, `POINTS`, `TRIANGLES`, `TRIANGLE_STRIP` or `TRIANGLE_FAN`.

1.3 Rendering a triangle — Assigning different colors to the vertices

Analyze the incomplete example **WebGL_example_08.html**.

Identify the main change regarding the previous examples:

- Assigning a color to each vertex.
- Calling the *shaders* and passing two (global) arrays storing the coordinates and the color attribute of each vertex, as arguments, to the *vertex-shader*.

Questions:

- How is a color assigned to each triangle vertex?
- What happens when each triangle vertex is assigned a different color?

Task:

- Complete the example, so that a different color can be assigned to each triangle vertex.

1.4 Rendering triangles: *The Sierpinski Gasket*

[Wikipedia]

Analyze example **WebGL_example_09.html**.

Notice how the file contents are structured.

Analyze the recursive algorithm that defines the fractal.

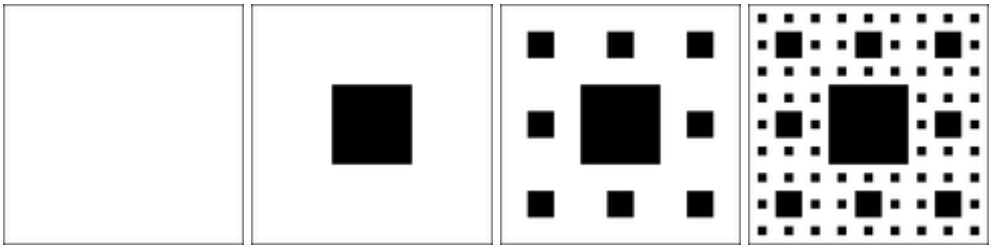
Questions:

- How are the coordinates computed for the successive vertices?
- Which coordinates are stored in the vertices array?

Suggestions:

- Allow the user to set the height of the recursion tree.
- Allow the user to set the coordinates of the initial triangle's vertices.
- Store in a file the coordinates of the vertices defining the rendered fractal, so that it can (later) be rendered from file.

1.5 *The Sierpinski Carpet* – OPTIONAL

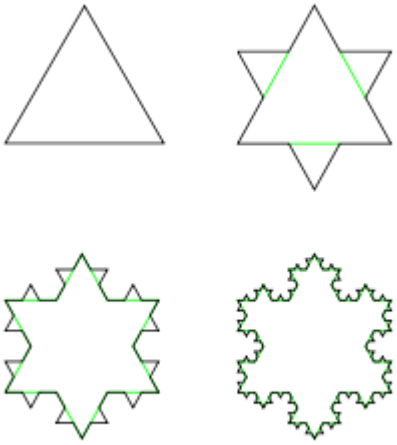


[Wikipedia]

Based on the previous example, develop a new one (**WebGL_example_10.html**) which allows rendering the Sierpinski Carpet, with different levels of recursivity.

At first, set the rules for the subdivision process. Note that each square is made up of two triangles.

1.6 Rendering line segments: *The Koch Snowflake*



[Wikipedia]

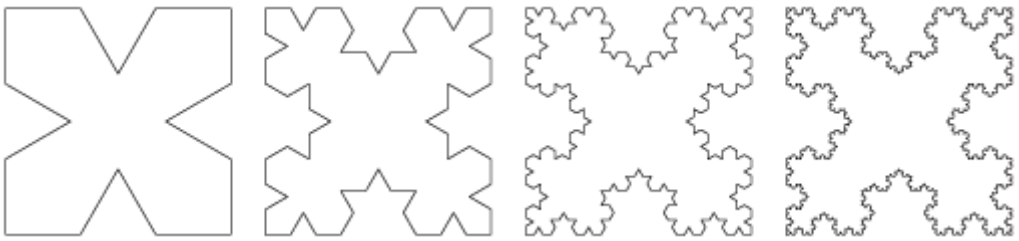
Analyze example **WebGL_example_11.html**.

Analyze the recursive algorithm that defines the fractal.

Questions:

- How are the coordinates computed for the successive vertices?
- Which coordinates are stored in the vertices array?

1.7 *The Cesàro Fractal* – OPTIONAL



[Mathworld]

Based on the previous example, develop a new one (**WebGL_example_12.html**) which allows rendering the Cesàro Fractal, with different levels of recursivity.

At first, set the rules for the subdivision process.