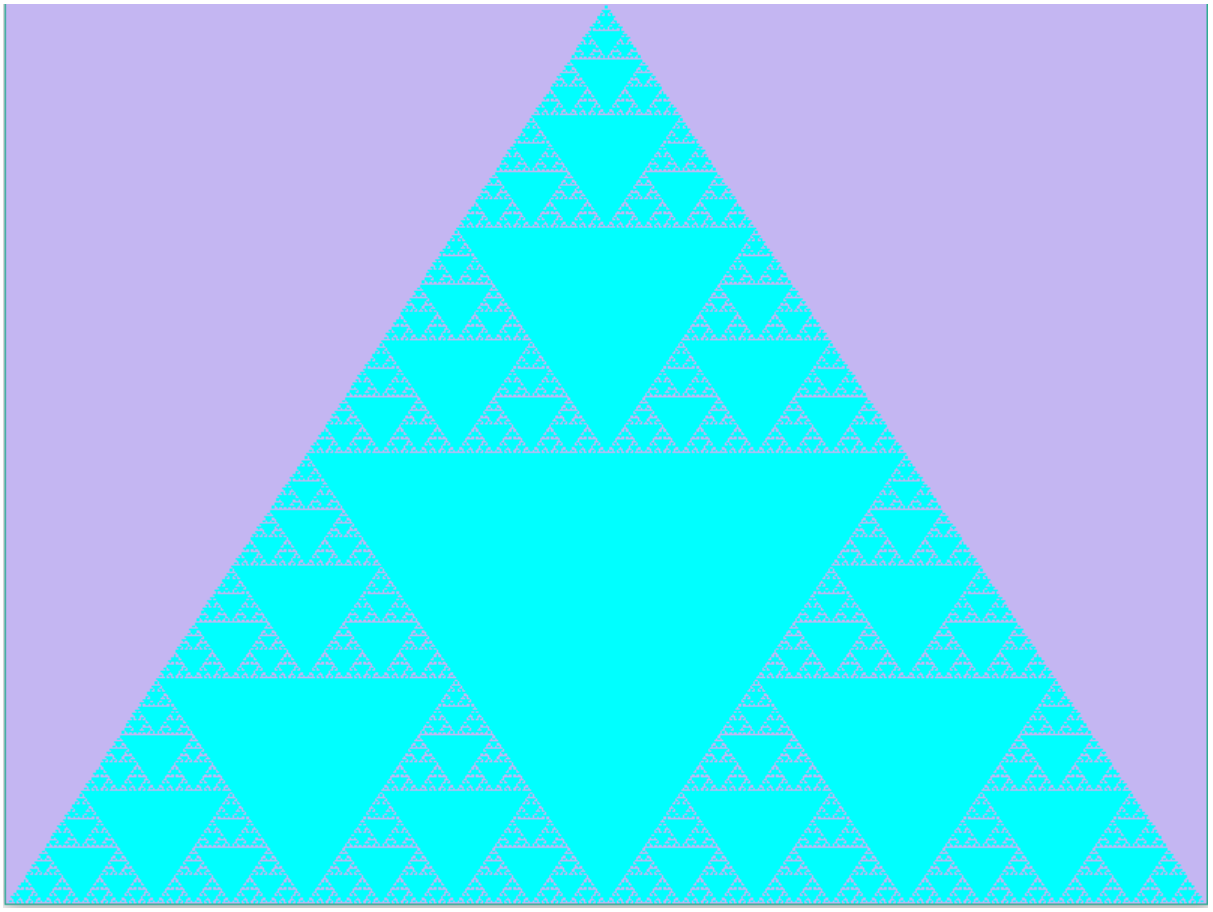
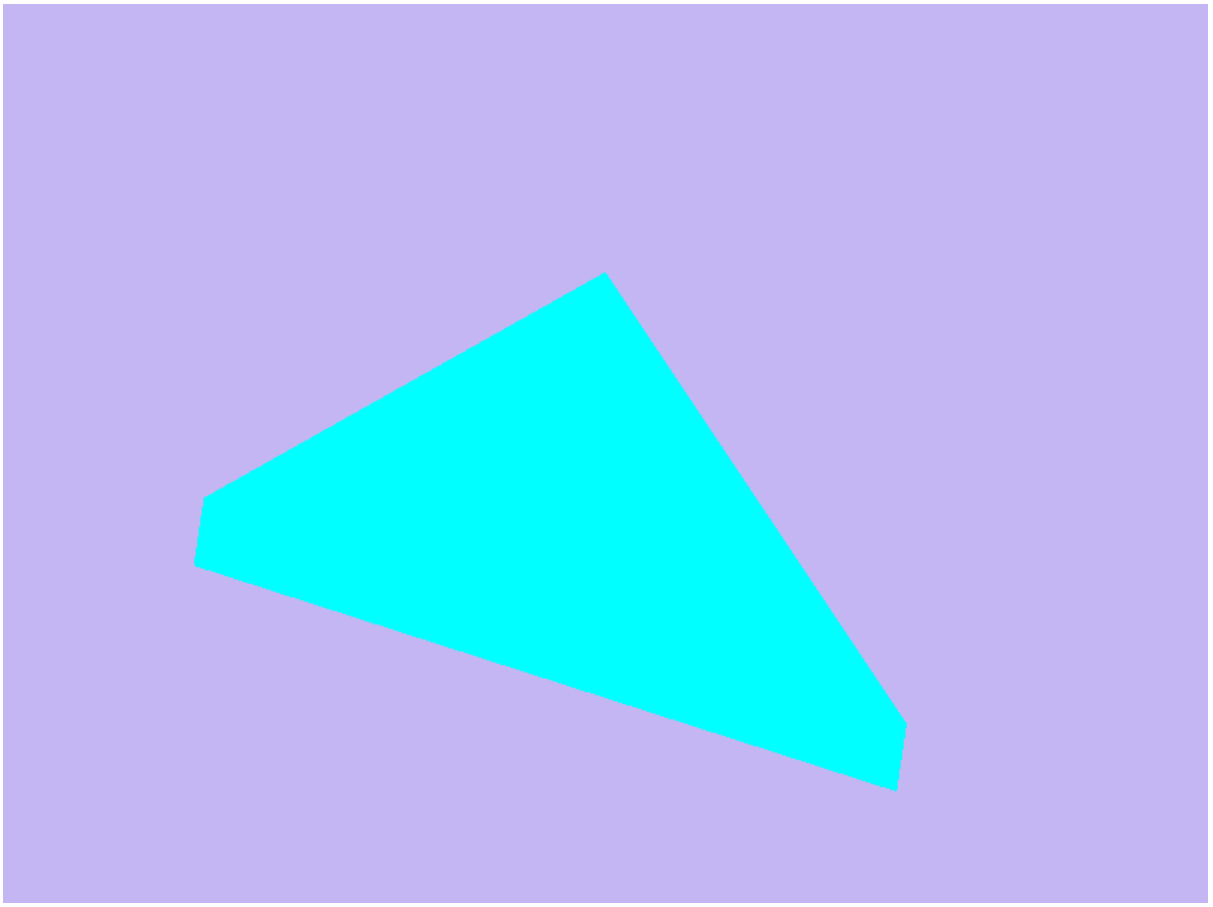
Computer Graphics - Assignment 1 - Erik Galler

Task 1: Drawing your first triangle



Task 2: Geometry and Theory

a) **Draw a single triangle passing through the following vertices:**
v0 = [0.6, −0.8, −1.2] , v1 = [0, 0.4, 0] , v2 = [−0.8, −0.2, 1.2]
Put a screenshot of the result in your report. This shape does not entirely look like a triangle. Explain in your own words:
i) What is the name of this phenomenon?
This phenomenon is called “clipping”
ii) When does it occur?
It occurs when primitives are outside of the 2x2x2 clipbox defined from -1 to 1 for x, y and z. The triangle is being clipped because of the v0[2] and v2[2] vertex.
iii) What is its purpose?
The purpose is that we have to define which part of the data we want to visualize, and therefore need a process of throwing away all geometry which exceeds the bounds of the screen.



b) [0.4 point] [report] While drawing one or more triangles, try to change the order in which the vertices of a single triangle are drawn by modifying the index buffer. A significant change in the appearance of the triangle(s) should occur. Show a screenshot in your report.

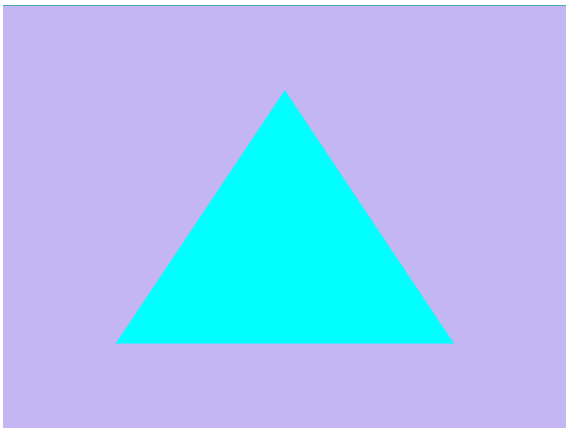
i)What happens?

vertex coordinates: [
-0.6, -0.6, 0.0,
 0.6, -0.6, 0.0,
 0.0, 0.6, 0.0,
]

index buffer:

counter-clockwise
rendered: [0, 1, 2], [2, 0, 1], [1, 2, 0]

clockwise
not rendered: [0, 1, 2], [2, 0, 1], [1, 2, 0]

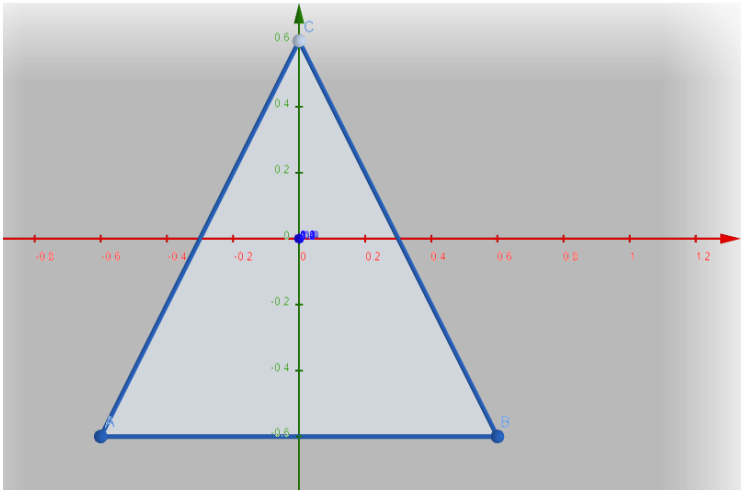


ii) Why does it happen?

There is no need to render the back-face of a surface (relative to the camera) because it won't be visible due to the front-face, resulting in a better performance.

iii) What is the condition under which this effect occurs? Determine a rule.

If we plot the different points in a 3D graph and look at the triangle from the -z axis and focus on the vertex order the reason becomes clear. OpenGL needs to know if the triangle is front-facing or back-facing and determines this based on the winding order of the triangles. In our example, if the winding order is counter-clockwise, that means it is front-facing, and it gets rendered. If its winding order is clockwise, meaning back-facing, it gets discarded, known as face-culling.



c) [0.5 point] [report] Explain the following in your own words:

i) Why does the depth buffer need to be reset each frame?

Because the depth test (which uses the depth buffer as storage) is dependent on the camera's position, which could change in between frames, rendering the old data in the depth buffer's useless.

ii) In which situation can the Fragment Shader be executed multiple times for the same pixel?

If multiple primitives have overlapping vertices this could happen.

iii) What are the two most commonly used types of Shaders? What are the responsibilities of each of them?

The two most common shaders are vertex shaders and fragment shaders. Vertex shaders transform shape positions into 3D drawing coordinate, whilst fragment shaders assign colors to rasterized pixels.

iv) Why is it common to use an index buffer to specify which vertices should be connected into triangles, as opposed to relying on the order in which vertices are specified in the vertex buffer(s)?

Because primitives often share vertices, and therefore the is no need to

render them multiple times, by only doing this once you also save memory.

v) While the last input of gl::VertexAttribPointer() is a pointer, usually a null pointer is passed in. Describe a situation in which you would pass a non-zero value into this function.

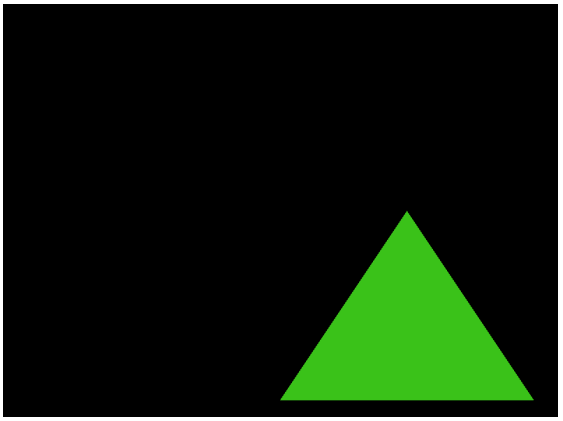
If you would like to define an array of generic vertex attribute data with only texture coordinates, for example given (x, y, z, u, v), where u and v are texture coordinates, pointer would be byte 3 (x, y, z) · 4 (bytes per coordinate) = 12bytes.

d) [0.2 point] [report] Modify the source code of the shader pair to:

i) Mirror the whole scene horizontally and vertically at the same time.

I created a linear combination of the horizontal mirror matrix and the vertical mirror matrix by performing a matrix multiplication (Order does not matter). Afterwards I matrix multiply the linear combination with my coordinates (NB correct order) which produces the requested transformation. I applied this by creating a global Shader variable with uniform.

Before transformation:



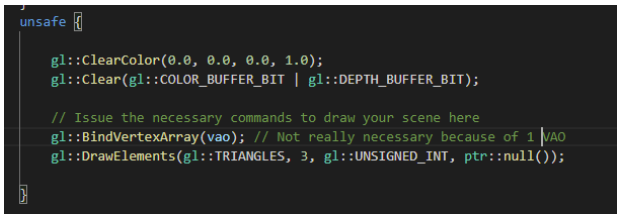
After transformation:

```
unsafe { let shaders = shader::ShaderBuilder::new()
    .attach_file("./shaders/simple.frag")
    .attach_file("./shaders/simple.vert")
    .link();
    shaders.activate();

    let x_mirror= glm::Mat4::from([
        [-1.0, 0.0, 0.0, 0.0],
        [0.0, 1.0, 0.0, 0.0],
        [0.0, 0.0, 1.0, 0.0],
        [0.0, 0.0, 0.0, 1.0]]);

    let y_mirror= glm::Mat4::from([
        [1.0, 0.0, 0.0, 0.0],
        [0.0, -1.0, 0.0, 0.0],
        [0.0, 0.0, 1.0, 0.0],
        [0.0, 0.0, 0.0, 1.0]]);

    let xy_mirror = x_mirror * y_mirror;
    let name = String::from("mirror");
    let mirror_location: gl::types::GLint = gl::GetUniformLocation(shaders.program_id,  name.as_ptr() as *const gl::types::GLchar);
    gl::UniformMatrix4fv(mirror_location, 1, gl::FALSE, xy_mirror.as_ptr());

};
```

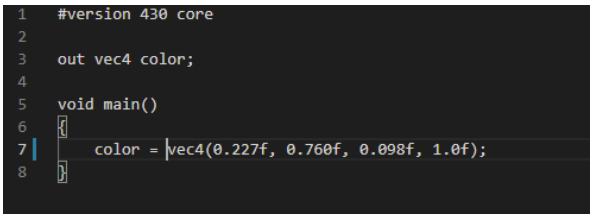
```
#version 430 core

in vec3 position;

uniform mat4 mirror;

void main()
{
    gl_Position = mirror * vec4(position, 1.1f);
}
```

ii) Change the colour of the drawn triangle(s) to a different colour. Put a screenshot of each effect in your report. Explain briefly how you accomplished each effect.

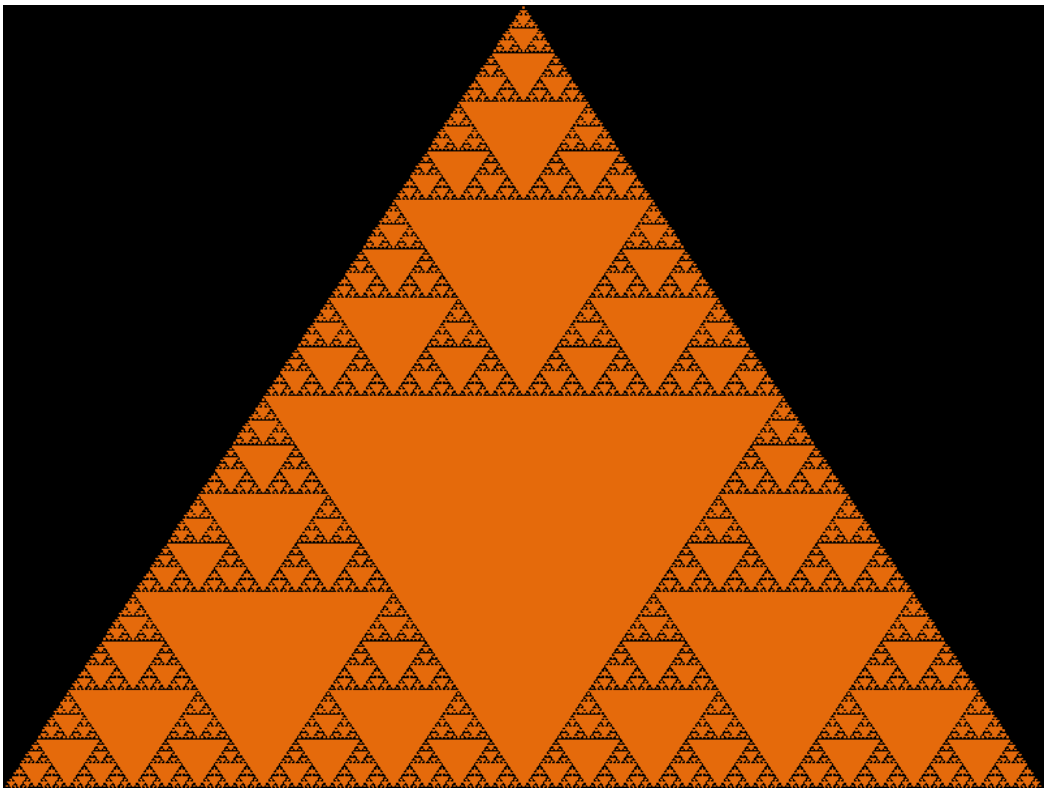
Change float values in simple.frag file (RGB Normalized decimal) and for background change gl::ClearColor() input.

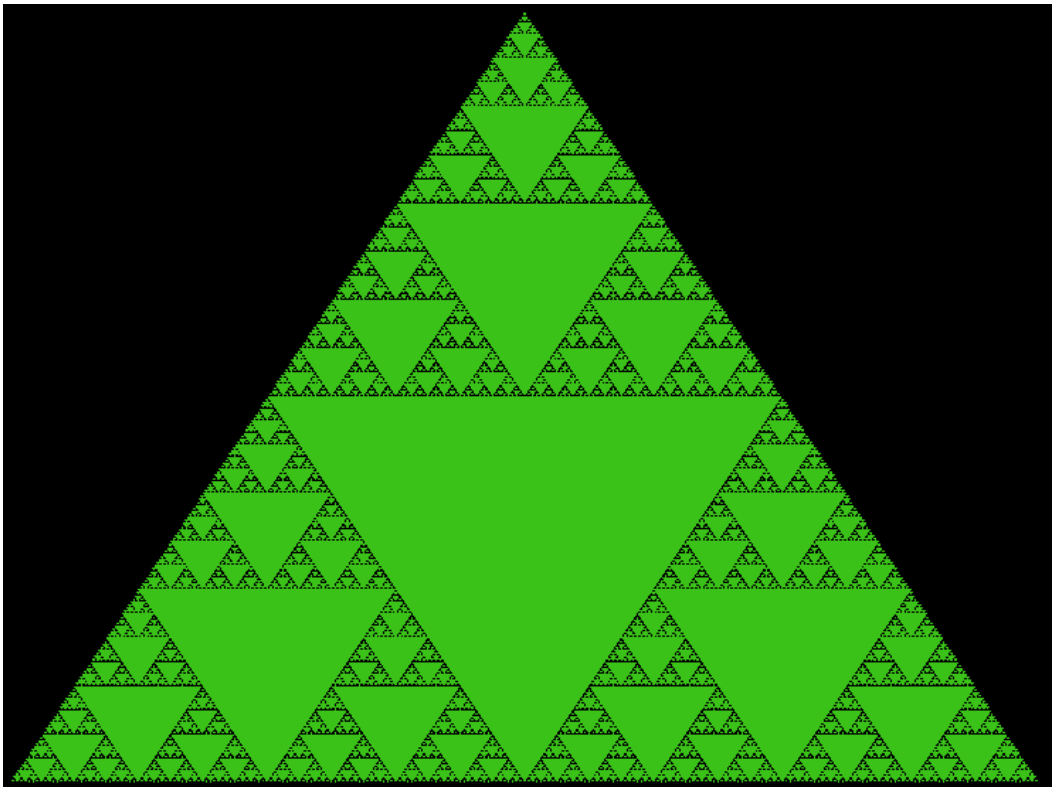


```
unsafe {

    gl::ClearColor(0.0, 0.0, 0.0, 1.0);
    gl::Clear(gl::COLOR_BUFFER_BIT | gl::DEPTH_BUFFER_BIT);

    // Issue the necessary commands to draw your scene here
    gl::BindVertexArray(vao); // Not really necessary because of 1 VAO
    gl::DrawElements(gl::TRIANGLES, 3, gl::UNSIGNED_INT, ptr::null());
}
```



```
#version 430 core

out vec4 color;

void main()
{
    color = vec4(0.227f, 0.760f, 0.098f, 1.0f);
}
```





Task 3: Optional Bonus Challenges [up to 0.2 bonus points]

Draw a circle:

```
// == // Set up your VAO here
let vao: u32;

unsafe {
    // let vertex_coordinates: Vec<f32> = vec![
    //     0.5, 0.0, 0.0,
    //     0.0, -1.0, 0.0,
    //     1.0, -1.0, 0.0
    // ];

    // let array_of_indices: Vec<u32> = vec![0, 1, 2];


    let mut vertex_coordinates: Vec<f32> = Vec::new();
    let mut array_of_indices: Vec<u32> = Vec::new();
    let radius: f32 = 0.9;

    for theta in 1..720 {
        let mut angle: f32 = theta as f32;
        angle = angle/360.0*2.0*std::f32::consts::PI;
        let x = radius*angle.cos();
        let y = radius*angle.sin();
        vertex_coordinates.push(x);
        vertex_coordinates.push(y);
        vertex_coordinates.push(0.0);
        array_of_indices.push(theta);
    }
    vao = create_vao(&vertex_coordinates , &array_of_indices);
}
```