CS 470 Introduction to Computer Graphics

Fall 2019
Instructor: Gianfranco Doretto gidoretto@mix.wvu.edu

Assignment 1

Learning Vertex Attributes

Software Requirements

1. CMake
2. C++ Compiler and IDE
    1. GCC (5.0 and higher) + CLion/NetBeams/Eclipse/KDevelop/<you name it> on Linux
    2. XCode on macOS
    3. Microsoft Visual Studio (I'd recommend 2015) on Windows

b. Installed graphics driver for the graphics card or integrated graphics.

c. Python 2.7/3.* (to run gl3w_gen.py)

General Rules

You should not use the deprecated functions of OpenGL in this or any other programming assignment. Any assignment that uses old, deprecated OpenGL will not be graded.

All code must be portable, the instructor should be able to compile it on any major platform, no matter on which one you were developing it.

You'll be given a skeleton code, which resembles the examples shown in class.

How to Submit Your Work

Submit a zip file containing a folder named with your last name, followed by underscore, then the initial of your first name, then underscore and the assignment number. So, if the instructor were to submit an assignment, the folder would be named

```
doretto_g_a1
```

and the instructor would submit a zip archive named `doretto_g_a1.zip`
All your files should be inside the folder in the archive.

You should put the zip archive in your MIX Google Drive, and you should share that file with the instructor, identified by the email gidoretto@mix.wvu.edu

Submission Deadline

The assignment is due on **September 27, 2019**, before the beginning of the class.

Grading Scheme

1. Code compiles and the window is displayed: 10
2. Vertex buffer object (VBO) and index buffer object (IBO) is created and filled with data according to the assignment: 25 points.
3. Vertex attributes are set correctly: 25 points.
4. The shape stored in VBO in polar coordinates is rendered correctly: 25 points
5. The shape is rotating without modifying VBO, but with a uniform variable: 15 points

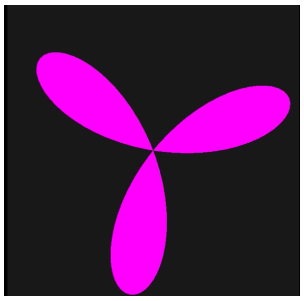
Total 100 points.

Assignment Description

Topics learned:

- Vertex buffer object (VBO), index buffer object (IBO)
- Vertex attributes
- Drawing primitives

In this assignment, you are going to visualize an animated scene, where the shape is stored in polar coordinate in the vertex buffer and rotates over time.



You should:

- Create and fill the VBO for an object with vertex structure described below
- Each vertex should have two attributes, each of which is one float-point number
- In the vertex shader you should compute the final position of the vertex and assign it to gl_Position
- Pass a time parameter to the vertex shader, which should be used to rotate the shape.

Detailed Description

Each vertex should consist of two attributes. Each attribute is one float. The size of the vertex should be 2 * sizeof(float), which is 8 bytes.

The first attribute should be the angle of the vertex in polar coordinates.
The second attribute should be the radius of the vertex in polar coordinates.

The final position of the vertex should be computed by the vertex shader as follows:

x = cos(angle + rotation_speed * time) * radius
y = sin(angle + rotation_speed * time) * radius

where **angle**, **radius** are the attributes of the vertex.

The vertex buffer should consist of 200 vertexes. Attributes for all vertices (except the first one) should be assigned as follows:

angle = i * 2 * Pi / (vertex_count - 2)
radius = cos(angle * 3);

where i is the index of the vertex.

The first vertex should always be in the center, because we are going to render them using the primitive **`GL_TRIANGLE_FAN`**. So, you will have 199 vertices on the perimeter.

You need to take care of how you pass the data to the VBO.

In the function call:

```
glBufferData(GL_ARRAY_BUFFER, <SIZE_OF_DATA>, vertices, GL_STATIC_DRAW);
```

make sure that SIZE_OF_DATA has the appropriate size of the buffer in bytes.

You will need to get the location of the two attributes:

```
m_attrib_angle = glGetAttribLocation(m_program, "a_angle");
m_attrib_radius = glGetAttribLocation(m_program, "a_radius");
```

The two lines above, assume that you've named the three attributes in the vertex shader as `a_angle` , and `radius`.

You will need to get the location for the uniform parameter that will be used as a time parameter for rotation:

```
m_uniform_time = glGetUniformLocation(m_program, "u_time");
```

The line above assumes that the uniform parameter in the vertex shader is named **`u_time`**.

The vertex shader should compute x and y position of the vertex and assign in to the gl_Position variable. So, if you have stored the final position in a variable called **position** of type **vec2,** then you can do:

gl_Position = vec4(position, 0.0, 1.0);

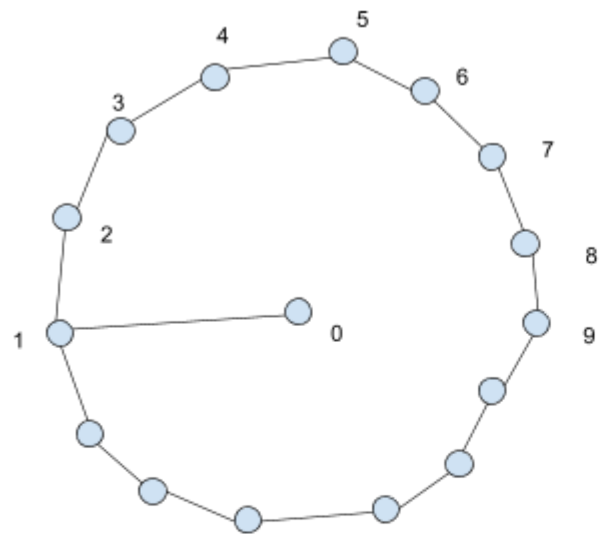
In the skeleton code, in the `main.cpp` file the time is computed since the start of the program and it is passed to the Draw method of the Application as a float point number.

```
#include <chrono>
…

auto start = std::chrono::steady_clock::now();
/* Loop until the user closes the window */
while (!glfwWindowShouldClose(window))
{
      auto current_timestamp = std::chrono::steady_clock::now();
      std::chrono::duration<float> elapsed_time = (current_timestamp - 
start);
      app->Draw(elapsed_time.count());
      /* Swap front and back buffers */
      glfwSwapBuffers(window);
      /* Poll for and process events */
      glfwPollEvents();
}
```

You can use that value to set the uniform parameter as follows:

```
glUseProgram(m_program);

glUniform1f(m_uniform_time, time);
```

The primitives should be drawn using the **GL_TRIANGLE_FAN** method. That means, that the ordering of the vertices should be the following:



So, the first vertex should be at 0,0.

Make sure that you draw triangles using the **GL_TRIANGLE_FAN** constant:

```
glDrawElements(GL_TRIANGLE_FAN, vertexCount, GL_UNSIGNED_SHORT, 0);
```

Make sure that you call the function **`glEnableVertexAttribArray`** to enable each of the thwo attributes and then, after rendering you also disable them (**glDisableVertexAttribArray**).

Make sure that for each of the thwo attributes, you call **`glVertexAttribPointer`** with proper arguments.

```
glVertexAttribPointer(<attribute>, 1, GL_FLOAT, GL_FALSE, <stride>,
<offset>);
```

Alternatively, you can set only one attribute, of type **vec2**, using the following call:

```
glVertexAttribPointer(<attribute>, 2, GL_FLOAT, GL_FALSE, <stride>,0);
```

In this case, you will need to change vertex shader to reflect the change, you will have only one attribute:

```
    attribute vec2 a_polar_coord;
```