IM420 Advanced Computer Graphics – WS 2020
Assignment 1- Drawing 2D scene

Sahabaj Barbhuiya, Faezeh Asgharkermani

9th November, 2020

1.1 Task

Based on the lecture notes and project template from the lecture, the task was to draw a 2D scene based on different triangles. This first assignment uses the concept of Vertex Buffer Objects (VBOs) and Vertex Array Objects (VAOs). In order to draw the 2D scene, the program needs to be able to output all the triangles used to draw the 2D scene that is rendered with simple OpenGL shaders.

The IDE which was used to create the 2D scene for this first assignment is Visual Studio 2019 using C++. This first assignment was not difficult and this serves as an introduction to the basic project structure and basic 2D rendering in OpenGL. This first assignment also demonstrates the complex practices where we learn to work out simpler solutions in OpenGL.

1.2 Approach to the solution

For this assignment we focus on the use of *GL_TRIANGLES* while other primitives are *GL_POINTS*, *GL_TRIANGLE_STRIP*, *GL_TRIANGLE_FAN* and so on. The triangular shapes were defined with coordinates respectively vertex values for x, y and z. As well as setting a colour for each vertex with values for red, green, blue and the alpha channel which is shown in the code snippet below:

```
// define  vertices
GLfloat vertices[] = {
  //TREE 1
  -0.6f, -0.6f, 0.0f,
  -0.6f, -0.4f, 0.0f,
  -0.4f, -0.6f, 0.0f,
    };

// define  colors
GLfloat colors[] = {
  //Brown
  0.5f, 0.35f, 0.05f, 1.0f,
  0.5f, 0.35f, 0.05f, 1.0f,
  0.5f, 0.35f, 0.05f, 1.0f,
  };
```

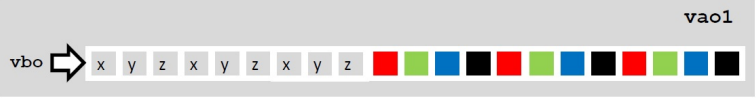

Figure 1.1: The schematic for vertex buffer object which holds the values of vertex data (position, normal vector, color, etc)

1.3 VBOs and VAOs

To able to process the geometry data for all the triangles to draw the 2D scene, we have to store the data. We can use the vertex buffer object (VBOs) to store the vertex position, vertex color data in the VBOs. From this values an array is generated that stores the data of a geometric object and is able to process these attributes accordingly.

The following code can be used generate and bind the data to the array buffer:

```
1  glGenBuffers(1, &vbo);
2  glBindBuffer(GL_ARRAY_BUFFER, vbo);
```

Similarly, an vertex array object (VAOs) must be generated and binded using the following code:

```
1    glGenVertexArrays(1, &vao);
2    glBindVertexArray(vao);
```

To store the data in the VBO, the VBO needs to know its size, as well as where and what data to hold. The function *glBufferData* defines all the values needed to store this data, such as:

- **target:** specifies the target that the buffer object is bound to
- **size:** specifies how many bytes the buffer object needs for example, *vertices * values * sizeOfFloat*
- **data:** specifies the data the buffer is initialized with

In order to store two arrays in the buffer data, *glBufferSubData* method can be used, which allows you to define an offset in the buffer. The following code snippet shows the usage of *glBufferData* and *glBufferSubData*:

```
1  glBufferData(GL_ARRAY_BUFFER, sizeof(vertices) + sizeof(colors), NULL,
     GL_STATIC_DRAW);
2  glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(vertices), vertices);
3  glBufferSubData(GL_ARRAY_BUFFER, sizeof(vertices), sizeof(colors), colors);
```

The *GL_STATIC_DRAW* is used and indicates that the geometry will not need updates during the runtime.

For our simple shaders to work correctly, pointers have been defined that provide an array of vertex attribute data which will be processed by the shaders. The following code snippet shows the usage of vertex attribute data:

```
1  positionID = glGetAttribLocation(shaderProgram, "inVertex");
2  colorID = glGetAttribLocation(shaderProgram, "inColor");
3
4  glVertexAttribPointer(positionID, 3, GL_FLOAT, GL_FALSE, 0, NULL);
5  glVertexAttribPointer(colorID, 4, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(sizeof(
     vertices)));
```

1.4 Drawing the 2D scene

An online tool https://www.geogebra.org/ can be used to draw a 2D scene (or a tangram) which is very helpful to find the exact Cartesian coordinates. For example, Figure. 1.2 and Figure. 1.3 shows the grid coordinate adaptation which will be used to draw and render our 2D scene.

There are 24 triangles used to complete this assignment and hence 72 vertices() are needed to draw and render the 2D scene. Finally, we use *glDrawArrays* function to draw the 2D scene. As discussed earlier, *GL_TRIANGLES* geometric primitive is used to draw all the triangles for the 2D scene.

```
1    glDrawArrays(GL_TRIANGLES, 0, 72);
```

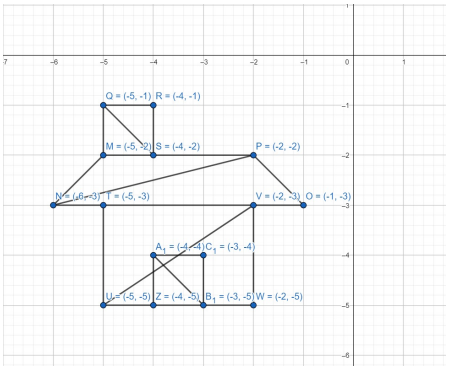

Figure 1.2: Grid coordinates for the house

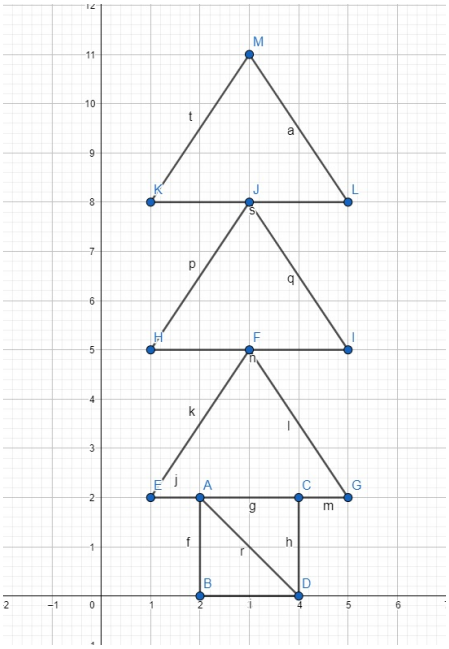

Figure 1.3: Grid coordinates for the tree

1.5 Result

At the beginning of this assignment the challenge was getting the correct size of the buffer and array. The final result can be seen in Figure 1.4 as the output screen shows the fully rendered 2D scene with two different houses and two trees. After completing this assignment of drawing a 2D scene, we can say that we have learned how Vertex Buffer Objects (VBOs) and Vertex Array Objects (VAOs) works.

Before rendering any scene we must update the number of vertices according to the total number of triangles which is used to render the complete 2D scene. Moreover,

Figure 1.4: A screenshot of the final 2D scene in the OpenGL application

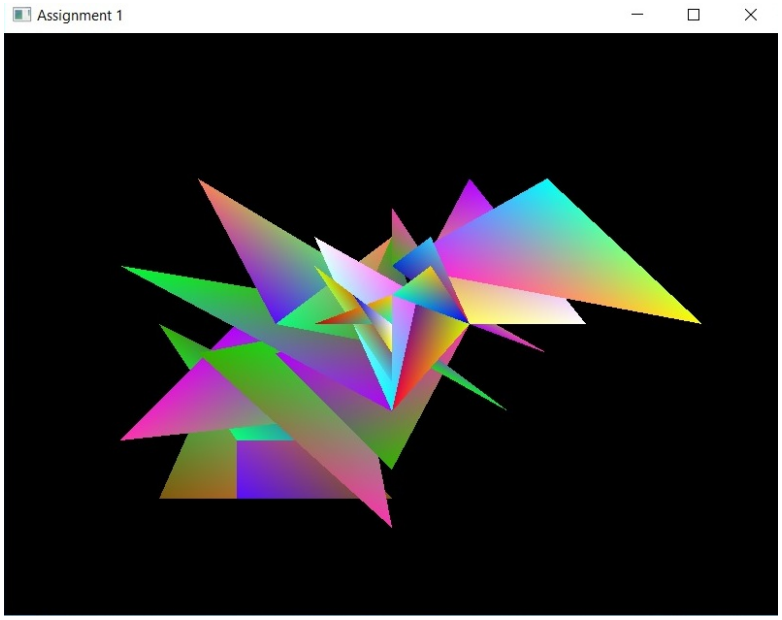

Figure 1.5: A screenshot of distorted scene due to wrong vertex attribute pointers values

we also have adjust the buffer offset in the vertex attribute pointers before rendering else we might have distorted scene as output. Let's take the following code snippet as example:

```
1 glVertexAttribPointer(positionID, 2, GL_FLOAT, GL_FALSE, 0, NULL);
```

```
2 glVertexAttribPointer(colorID, 3, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(sizeof(
      vertices)));
```

In this code, the value of *positionID* is changed to 2 and the value for *colorID* is changed to 3. This will result in the distorted scene which is shown in Figure 1.5. As we know we need three vertices (x, y, z) to draw a triangle, hence if we can change the value of *positionID* from 3 to 2 in *glVertexAttribPointer* we will see a distorted scene. Moreover, in OpenGL we use R, G, B and A values for colors. Hence, changing the value of *colorID* from 4 to 3 in *glVertexAttribPointer* will result in strange colors in our scene.

For this assignment a lot of repetitive work has to be done such as setting the triangles' data in the initialization array. However, this repetition can be overcome by using Index Buffer Objects (IBOs) by encapsulating several VBOs. This will ultimately reduce the total memory size depending on the number of the triangles in a scene.