COMP 371 Computer Graphics

Assignment 2

Winter 2025

This assignment aims to introduce you to OpenGL and its basic functions. You will learn how to set up an OpenGL context, draw basic shapes, and apply simple transformations. You will create a pyramid and control its animation by using the keyboard.

This assignment must be done in a group of a **minimum 2 and a maximum 4**.

Due date: as indicated on Moodle.

Requirements:

- Set up an OpenGL window using GLFW and GLEW.
- Draw a colored Pyramid.
- Apply transformations (translation, rotation, and scaling) to the shape.
- Use keyboard input to control the transformations.

Tools:

- C++
- GLFW (for window management and input)
- GLEW (for loading OpenGL functions)
- GLM

Steps:

Step 1: Setting Up Your Environment

1. Install GLFW and GLEW:
    - Download and install GLFW from GLFW website https://www.glfw.org/ .
    - Download and install GLEW from GLEW website https://glew.sourceforge.net/install.html .

2. Set Up a Basic OpenGL Window:
   Create a new C++ project and include the necessary headers for GLFW and GLEW. Initialize a window and an OpenGL context.

   There is another document on Moodle with more details on setting up your environment.

Step 2: Drawing Basic Shapes

1. Define the Vertex Shader and Fragment Shader:
   Create **a simple vertex and fragment shader** to render the shapes.

   This could be something **similar to**:

```
const char* vertexShaderSource = R"glsl(
    #version 330 core
    layout (location = 0) in vec3 aPos;
    uniform mat4 transform;
    void main() {
        gl_Position = transform * vec4(aPos, 1.0);
    }
)glsl";
```

2. Compile and Link the Shaders:
   Compile the vertex and fragment shaders and link them into a shader program.

```
// Vertex Shader
unsigned int vertexShader = glCreateShader(GL_VERTEX_SHADER);
```

```
// Fragment Shader
unsigned int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
```

3. Define the Vertex Data for a Pyramid. The code below defines a triangle. You need to define pyramid:

```
// Define vertices for a triangle
float verticesTriangle[] = {
    -0.5f, -0.5f, 0.0f,
     0.5f, -0.5f, 0.0f,
     0.0f,  0.5f, 0.0f
};
```

4. Set Up the Vertex Array Object (VAO), Vertex Buffer Object (VBO), and Element Buffer Object (EBO):

```
unsigned int VBO[2], VAO[2], EBO;
glGenVertexArrays(2, VAO);
glGenBuffers(2, VBO);
glGenBuffers(1, &EBO);
```

```
// Bind the Vertex Array Object for the triangle
glBindVertexArray(VAO[0]);
```

5. Render the Shapes:

```
// Render loop
while (!glfwWindowShouldClose(window)) {
    processInput(window);

    // Rendering commands here
    glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    glUseProgram(shaderProgram);
```

Step 3: Applying Transformations

1. Create Transformation Matrices:
   Use GLM (OpenGL Mathematics) library to create transformation matrices.

2. Add Uniforms for Transformation Matrices in the Shader:
   Update the vertex shader to accept a transformation matrix.

3. Apply Transformations:

```
// Transformation for the triangle
glm::mat4 transform = glm::mat4(1.0f);
transform = glm::translate(transform, translation);
```

Here you are the keyboard keys you need to support and their functionalities:
w → translate up by a certain distance "**d**" that is hardcoded in your code. You choose d.
s → translate down by distance "**d**"
a → translate to the left by distance "**d**"
d → translate to the right by distance "**d**"

q → rotate around the z axis by 30 degrees anticlockwise
e → rotate around the z axis by 30 degrees clockwise

r → scale in the +ve z direction by a factor **"s"**. You hardcode s in your program
f → scale in the -ve z direction by a factor **"s"**.

Step 4: Implement Keyboard Input for Transformations

1. Modify `processInput` function to handle keyboard inputs.

2. Apply the Transformations based on Input:

```
void processInput(GLFWwindow* window) {
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);

    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
        translation.y += 0.001f;
```

Grading:

- Quality of the code is very important 20%
- Good Design 10%
- Functionality and proper use of C++ constructs and OpenGL functions 70%

Demos:

- A demo will be given to a TA. No credit will be given without the demo.
- Double booking demo slots will result in a 30% deduction. Only one demo slot per group.

- Missing your demo time will result in a 30% deduction of your mark.
- All team members must be present during the demo.
- During the demo, the TA may ask a specific student a specific question. Only that student need to answer the question. Different grades may be given to different students based on their knowledge of the code and the different functionalities of the OpenGL function.
- Every student is responsible for the whole code. Do not say, I did not do this part. You should be aware of all parts of the project.

Submission:

On Moodle, you need to submit:

- A zip that contains the whole project. You need to go to the root folder of the project, right click, send to zipped folder. This will create a zip that contains all the files and folders used by Visual C++ project. Then upload the zip.
- A sample run in pdf format that shows all the functionality of your code.
- A report on who has done what. This report must list all the tasks that have been performed by each group member.

Only **one submission per group**. Make sure to write the names and ID numbers of all team members on all submitted documents.

Have fun 😊