3D Pyramid Rendering using OpenGL, Glumpy, and Texture Mapping

---

Project Description

- This project is a 3D graphics implementation using Glumpy and PyOpenGL.
- It renders a rotating pyramid using an index buffer and applies texture mapping on its surface.
- The project demonstrates the modern OpenGL pipeline and uses custom vertex and fragment shaders.
- The objective is to apply core computer graphics concepts like transformations, shading, and texturing in a practical OpenGL environment

Tools & Technologies Used

- Python.
- Python Packages:
  - wheel - for package installation.
  - pillow - for image manipulation.
  - numpy 1.26.0 - for numerical operations.
  - PyOpenGL - OpenGL python binding.
  - glumpy - visualization library.
  - glfw - context generation library.
- Visual Studio Code.

Source Code

1. Imports

```
from glumpy import app, gloo, gl
from glumpy.transforms import Rotate
import numpy as np
from PIL import Image
```

- **from glumpy import app, gloo, gl**: These lines import necessary modules from the Glumpy library.
  - app: Provides functions for creating windows and managing the application.
  - gloo: Offers an object-oriented interface to OpenGL objects like meshes, buffers, and shaders.
  - gl: Provides direct access to OpenGL constants and functions.

- **from glumpy.transforms import Rotate**: This statement imports the Rotate class from Glumpy's transforms module, used to rotate the pyramid.
- **import numpy as np**: Imports the NumPy library and assigns it the alias np. NumPy is used for numerical operations, especially for creating arrays that represent vertices and coordinates.
- **from PIL import Image**: This statement imports the Image module from the Pillow (PIL) library, which is used for loading and processing the image used as a texture.

2. Texture Loading

```
img = Image.open('pyramid.jpeg')
img_data = np.array(img).astype(np.uint8)
texture = gloo.Texture2D(img_data)
```

- **img = Image.open('pyramid.jpeg')**: This line opens the image file 'pyramid.jpeg' using Pillow's Image.open() function.
- **img_data = np.array(img).astype(np.uint8)**: This line converts the image data into a NumPy array of unsigned 8-bit integers. This is necessary because OpenGL requires texture data to be in this format.
- **texture = gloo.Texture2D(img_data)**: This line creates a Glumpy Texture2D object from the NumPy array. This object represents the texture in OpenGL.

3. Vertex Shader

```
vertex = """
#version 440 core
attribute vec3 position;
attribute vec2 texcoord;
varying vec2 v_texcoord;

void main() {
    gl_Position = vec4(<transform(position * 0.7)>, 1.0);
    v_texcoord = texcoord;
}
"""
```

- **The vertex shader** is a program that runs on the GPU for each vertex of the 3D model.
- **#version 440 core**: Specifies the version of the OpenGL Shading Language (GLSL) used.

- **attribute vec3 position**: Declares an input attribute named position that represents the 3D coordinates of each vertex.
- **attribute vec2 texcoord**: Declares an input attribute named texcoord that represents the 2D texture coordinates for each vertex.
- **varying vec2 v_texcoord**: Declares a varying variable named v_texcoord. Varying variables are used to pass data from the vertex shader to the fragment shader. In this case, it passes the texture coordinates.
- **gl_Position = vec4(<transform(position * 0.7)>, 1.0)**: Calculates the final position of the vertex.
  - position * 0.7: Multiplies the vertex position by 0.7 to scale it down.
  - <transform(...)>: Applies a transformation (rotation, in this case).
  - vec4(..., 1.0): Constructs a 4D vector (x, y, z, w) required for OpenGL, where w is set to 1.0.
- **v_texcoord = texcoord**: Assigns the input texture coordinates to the varying variable v_texcoord to be passed to the fragment shader.

4. Fragment Shader

```
fragment = """
#version 440 core
uniform sampler2D u_texture;
varying vec2 v_texcoord;

void main() {
    gl_FragColor = texture2D(u_texture, v_texcoord);
}
"""
```

- **The fragment shader** is a program that runs on the GPU for each pixel (fragment) of the rendered primitive.
- **#version 440 core**: Specifies the version of GLSL used.
- **uniform sampler2D u_texture**: Declares a uniform variable named u_texture that represents the 2D texture. Uniform variables are values that are constant across all vertices and fragments for a single draw call.
- **varying vec2 v_texcoord**: Declares a varying variable named v_texcoord that receives the interpolated texture coordinates from the vertex shader.
- **gl_FragColor = texture2D(u_texture, v_texcoord)**: Sets the color of the fragment. texture2D() is a GLSL function that samples the texture u_texture at the given texture coordinates v_texcoord and returns the color at that location.

5. Window and Rendering Setup

```
app.use("glfw")
window = app.Window(width=512, height=512, title="GREAT
PYRAMID OF OPENGL")
```

- **app.use("glfw"):** Tells Glumpy to use GLFW for window and context creation. GLFW is an open-source library for writing applications that use OpenGL or OpenGL ES graphics.
- **window = app.Window(width=512, height=512, title="GREAT PYRAMID OF OPENGL"):** Creates a Glumpy window with a width and height of 512 pixels and sets the window title.

6. Geometry and Buffers

```
pyramid = gloo.Program(vertex, fragment, count=5)

pyramid['position'] = [(0, 1, 0), (0, 0, -1), (-1, 0, 1),
(1, 0, 1), (0, 0, -1)]
pyramid['texcoord'] = [(0, 0), (0.5, 1), (0, 1), (1, 1),
(0, 1)]

index_buffer = np.array([0, 1, 2, 0, 2, 3, 0, 1, 4, 0, 3,
4], dtype=np.uint32).view(gloo.IndexBuffer)
```

- **pyramid = gloo.Program(vertex, fragment, count=5):** Creates a Glumpy Program object, which represents the compiled and linked vertex and fragment shaders. count=5 specifies that there are 5 vertices.
- **pyramid['position'] = [(0, 1, 0), (0, 0, -1), (-1, 0, 1), (1, 0, 1), (0, 0, -1)]:** Sets the vertex positions for the pyramid. Each tuple represents the (x, y, z) coordinates of a vertex.
- **pyramid['texcoord'] = [(0, 0), (0.5, 1), (0, 1), (1, 1), (0, 1)]:** Sets the texture coordinates for each vertex. Each tuple represents the (u, v) coordinates, which determine how the texture is mapped onto the pyramid.
- **index_buffer = np.array([0, 1, 2, 0, 2, 3, 0, 1, 4, 0, 3, 4], dtype=np.uint32).view(gloo.IndexBuffer):** Creates an index buffer. The index buffer defines the order in which the vertices should be connected to form triangles. This is more efficient than repeating vertex data.

7. Apply Transformation and Texture

```
pyramid['transform'] = Rotate(axis=(0.0, 1.0, 0.0))
pyramid['u_texture'] = texture
```

- **pyramid['transform'] = Rotate(axis=(0.0, 1.0, 0.0)):** Creates a Rotate transform that will rotate the pyramid around the y-axis (vertical axis).
- **pyramid['u_texture'] = texture:** Assigns the loaded texture to the 'u_texture' uniform variable in the shaders.

8. Rendering Function

```
@window.event
def on_draw(dt):
    window.clear()
    gl.glEnable(gl.GL_DEPTH_TEST)
    gl.glClear(gl.GL_DEPTH_BUFFER_BIT)
    pyramid.draw(gl.GL_TRIANGLES, index_buffer)
    pyramid['transform'].angle += 1
```

- **"@window.event def on_draw(dt):":** This is a Glumpy event handler that is called every time the window needs to be redrawn. dt represents the time elapsed since the last frame (delta time).
- **window.clear():** Clears the color buffer (the screen).
- **gl.glEnable(gl.GL_DEPTH_TEST):** Enables depth testing. Depth testing is a technique that ensures that objects are rendered in the correct order based on their distance from the viewer, preventing objects from being drawn on top of closer objects.
- **gl.glClear(gl.GL_DEPTH_BUFFER_BIT):** Clears the depth buffer.
- **pyramid.draw**(gl.GL_TRIANGLES, index_buffer): Draws the pyramid using the specified vertex data, shaders, and index buffer. gl.GL_TRIANGLES tells OpenGL to draw triangles.
- **pyramid['transform'].angle += 1:** Increments the rotation angle of the pyramid, causing it to rotate.

9. Run the Application

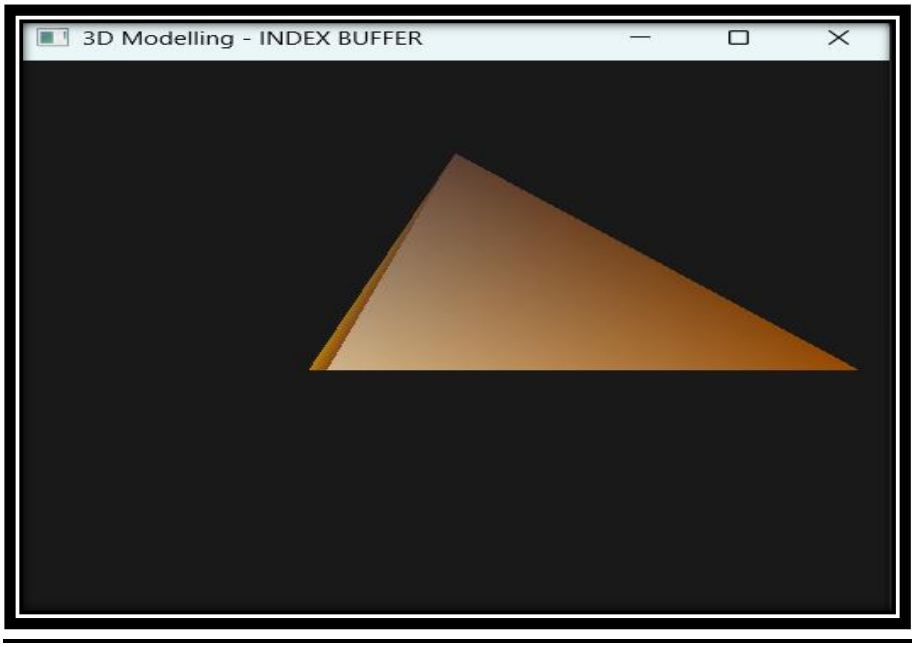
```
app.run()
```

- **app.run():** Starts the Glumpy application loop, which handles events (like drawing) and keeps the window open until it is closed.

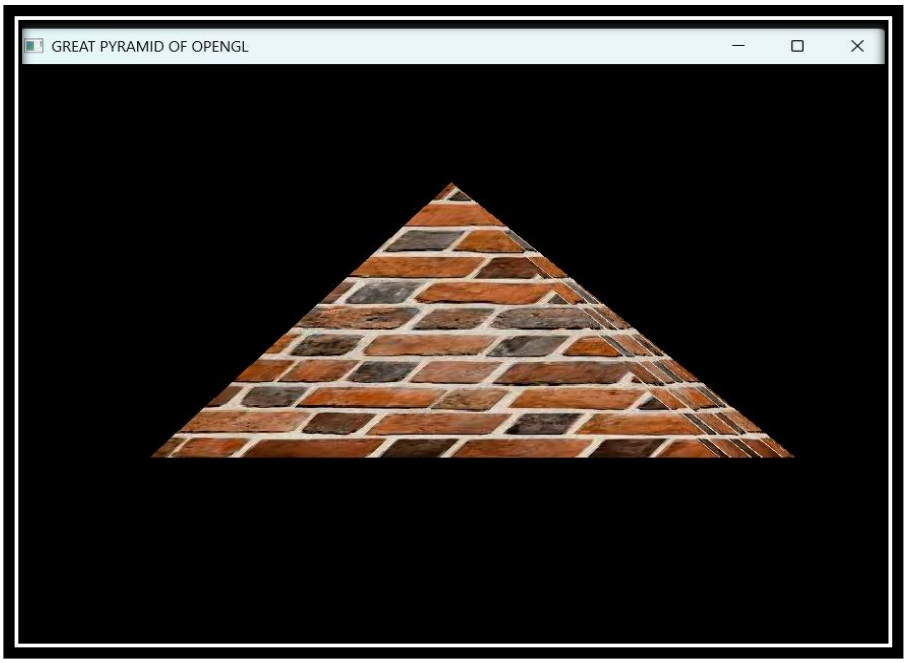
Screenshots/Recording

Insert a screenshot of the program window showing the 3D rotating textured pyramid. To **capture a screenshot:**

Screen Recording



Screen Recording



Source code