Lab 1: Introduction to OpenGL

Instructor: Ojaswa Sharma , TAs: Aarya Patel , Hardik Sanjeevkumar Jain

Due date: 14:00, 30 August 2024

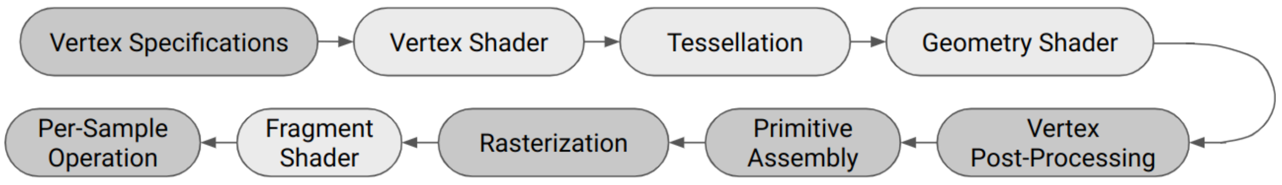
Introduction

OpenGL (Open Graphics Library) is a cross-platform, open standard API for rendering 2D and 3D vector graphics. It is widely used in video games, CAD applications, and virtual reality. Understanding OpenGL is crucial for computer graphics programming, and it forms the backbone for rendering in many applications, including those using ImGui.

This tutorial is designed to introduce you to the basics of OpenGL, with a focus on understanding the graphics pipeline, storing vertex data on the GPU, and rendering simple geometries. By the end of this tutorial, you should be able to create basic OpenGL applications that can draw simple shapes using different rendering methods.

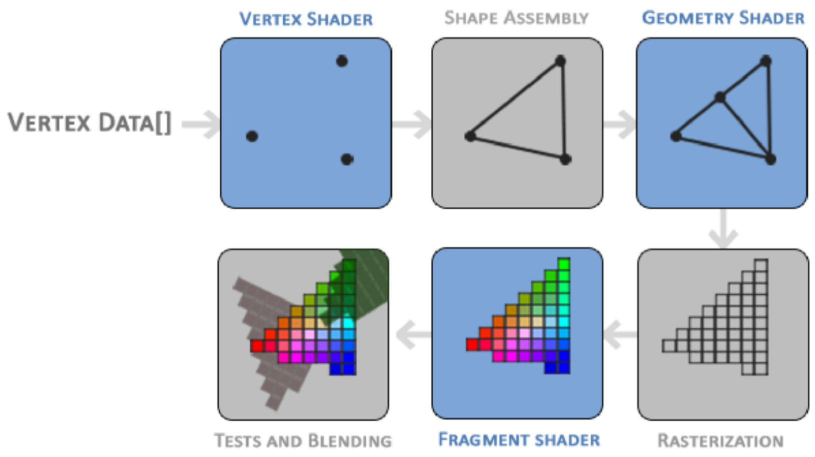
Graphics Pipeline

The Graphics Pipeline in OpenGL is a conceptual model that describes the steps required to transform 3D coordinates into a 2D image on the screen. It is divided into several stages:



Graphics Pipeline

1. **Vertex Processing:**
   Each vertex retrieved from the vertex arrays (as defined by the VAO) is acted upon by a **Vertex Shader.** Each vertex in the stream is processed in turn into an output vertex.
    - Optional primitive **tessellation** stages



Graphics Pipeline Visualization

2. **Vertex Post-Processing:** The outputs of the last stage are adjusted or shipped to different locations.
   - Transform Feedback happens here.
   - Primitive Assembly
   - Primitive Clipping, the perspective divide, and the **viewport transform** to window space.

3. Scan conversion and primitive parameter interpolation, which generates a number of Fragments.

4. A Fragment Shader processes each fragment. Each fragment generates a number of outputs.

5. **Per-Sample-Processing:** Including but not limited to
   - Scissor Test
   - Stencil Test
   - Depth Test
   - Blending
   - Logical Operation
   - Write Mask

Storing Vertex Data on GPU Memory

To render any shape, we need to store vertex data on the GPU. GPU is designed to handle massive amounts of data in parallel. This is done using Vertex Buffer Objects (VBOs) and Vertex Array Objects (VAOs).

Vertex Buffer Object (VBO):

- A VBO is used to store vertex data in GPU memory.
- The vertex data can include positions, normals, texture coordinates, etc.
- Storing vertex data in VBOs makes the rendering process faster as the data is directly accessed from the GPU memory.

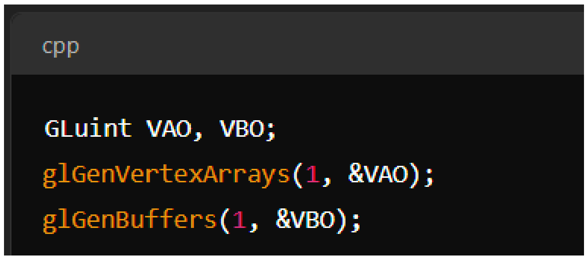
Vertex Array Object (VAO):

- A VAO is an object that stores the state of various VBOs and their associated attributes.
- It makes it easier to switch between different sets of vertex data.

Binding VBO to VAO:

This involves:

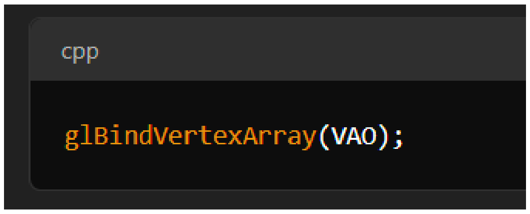
Generating a VAO and VBO:



cpp

```cpp
GLuint VAO, VBO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
```

Binding the VAO:



cpp

```cpp
glBindVertexArray(VAO);
```

Binding the VBO and copying vertex data:

```
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

Configuring vertex attribute pointers:

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
```

Using Different Render Types

OpenGL provides several rendering modes that determine how vertices are interpreted and drawn on the screen. Here, we'll cover three basic render types:

GL_POINTS:

- Renders each vertex as a point on the screen.

```
glDrawArrays(GL_POINTS, 0, 3);
```

GL_TRIANGLES:

- Interprets every three vertices as a triangle.

```
glDrawArrays(GL_TRIANGLES, 0, 6);
```

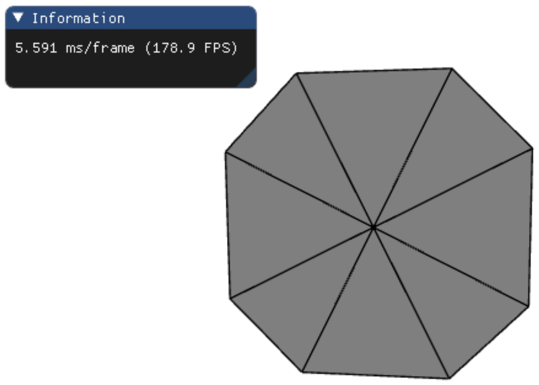
GL_LINE_STRIP:

- Renders a connected group of lines from the vertices.

```
glDrawArrays(GL_LINE_STRIP, 0, 4);
```

Exercise

The given code renders a rectangle using two triangles.
As shown in the images below, you must edit the code to render an octagon using triangles. You must define all the vertices of the octagon and a center point. You must define the order in which the vertices must be used to form all the triangles required to complete the octagon. Your outputs must look like this:



```
5.591 ms/frame (178.9 FPS)
```

Deliverables

Upload the zip file of **code** and **image of output**.
Name the zip file as lab01_<name>_<roll number>.zip
Example: lab01_aarya_2320156.zip

References

https://www.opengl.org/documentation/
https://www.khronos.org/opengl/wiki/Rendering Pipeline Overview
https://www.khronos.org/registry/OpenGL-Refpages
https://www.glfw.org/documentation.html
https://www.khronos.org/opengl/wiki/Framebuffer

Note: Your code should be written by you and be easy to read. You are NOT permitted to use any code that is not written by you. (Any code provided by the instructor/TA can be used with proper credits within your program). Theory questions need to be answered by you and not copied from other sources. Please refer to IIIT-Delhi's Policy on Academic Integrity here.