FLAME UNIVERSITY

EVERLASTING learning

FUNDAMENTALS OF COMPUTER GRAPHICS (CSIT304)

INTRODUCTION TO BASIC OPENGL AND GEOMETRIC PRIMITIVES

CHIRANJOY CHATTOPADHYAY

Associate Professor,

FLAME School of Computation and Data Science

INTRODUCTION

- A strong foundation in OpenGL sets the stage for advanced graphics programming.
- Our focus: Creating and manipulating basic geometric shapes.
- Hands-on exercises will reinforce your understanding.
- Expectation: comfortable drawing points, lines, and triangles in OpenGL.

UNVEILING THE WORLD OF OPENGL!

- OpenGL, or **Open Graphics Library**, is a powerful cross-platform graphics API. It serves as a standardized interface for rendering 2D and 3D graphics.
- **Widespread Use:**
  - Widely adopted across industries, including gaming, simulations, scientific visualization, and more.
  - Facilitates efficient communication with graphics hardware.
- **Open-Source Nature:**
  - An open-source library, fostering collaboration and continuous improvement.
- **Versatility:**
  - From embedded systems to high-performance computing, OpenGL adapts seamlessly.

GEOMETRIC PRIMITIVES

- Geometric primitives are fundamental shapes used in graphics programming.
- They serve as the building blocks for creating more complex visual elements.
- Examples of geometric primitives include points, lines, and triangles.
- These basic shapes form the foundation for constructing more intricate scenes.
- As we delve into OpenGL, mastering these geometric primitives will empower you to craft compelling visuals.

NAVIGATING COORDINATE SYSTEMS IN OPENGL

- Coordinate systems define the space in which graphics are rendered.
- In OpenGL, we use a right-hand coordinate system.
- In a right-hand coordinate system:
  - X-axis points right.
  - Y-axis points up.
  - Z-axis points out of the screen towards the viewer.
- Accurate placement of objects in a scene requires understanding coordinate systems.

NAVIGATING THE OPENGL RENDERING PIPELINE (1)

- The OpenGL rendering pipeline is a series of stages that transforms 3D data into 2D images.

- Key stages include the application, geometry, rasterization, fragment, and framebuffer stages.

NAVIGATING THE OPENGL RENDERING PIPELINE (2)

Application Stage:

- In the application stage, the CPU sends data and instructions to the GPU.
- This includes defining geometric data and shaders.

Geometry Stage:

- The geometry stage processes vertices and primitive assembly.
- Transformation matrices are applied to position objects in the 3D space.

Rasterization Stage:

- Rasterization converts geometric primitives into fragments.
- These fragments are the building blocks for pixels in the final image.

Fragment Stage:

- The fragment stage applies shaders and calculates the final color of each pixel.
- Lighting, texturing, and other effects are implemented here.

Framebuffer Stage:

- The framebuffer stage displays the final image on the screen.
- Double buffering is often used to avoid flickering.

BASIC STRUCTURE OF AN OPENGL PROGRAM

Initialization:

- The program begins with initialization.
- Setup tasks include setting up the window, loading shaders, and initializing OpenGL states.

Rendering Loop:

- The heart of an OpenGL program is the rendering loop.
- It iterates, continuously updating and rendering the scene.

Rendered Output:

- During each iteration, the scene is rendered based on the current state.
- This results in the continuous display of graphics on the screen.

Cleanup:

- Proper cleanup is essential at the end of the program.
- This involves freeing resources, deallocating memory, and closing the rendering context..

DRAWING POINTS

- To draw points in OpenGL, we use the **glPointSize** function.
- This function sets the size of the rendered points.

```
# Example code for drawing a point
glPointSize(10.0)  # Set point size to 10 pixels
glBegin(GL_POINTS)
glVertex2f(0.0, 0.0)  # Coordinates of the point
glEnd()
```

DRAWING LINES

- To draw lines in OpenGL, we use **glBegin(GL_LINES)** and **glEnd()** to specify the line segments.

```
# Example code for drawing a line
glBegin(GL_LINES)
glVertex2f(-50.0, -50.0)   # Starting point of the line
glVertex2f(50.0, 50.0)     # Ending point of the line
glEnd()
```

DRAWING TRIANGLES

- Drawing triangles in OpenGL involves using **glBegin(GL_TRIANGLES)** and **glEnd()** to define the vertices of the triangles.

```
# Example code for drawing a filled triangle
glBegin(GL_TRIANGLES)
glVertex2f(-50.0, -50.0)   # Vertex 1
glVertex2f(50.0, -50.0)    # Vertex 2
glVertex2f(0.0, 50.0)      # Vertex 3
glEnd()
```

TRANSFORMATION OF GEOMETRIC PRIMITIVES

- Let us apply transformation concepts on these geometric primitives
- Understand the nature of transformation

TASK: CREATE A SIMPLE SCENE

I was planning to do this in the class. Try to do this on your own and show it to me on Tuesday.

OBJECTIVE:

- Apply your knowledge of OpenGL basics to create a simple and visually appealing scene.
- Use points, lines, and triangles to construct the elements of the scene.

REQUIREMENTS:

Background:

- Set the background color of the window to something other than white.
- Experiment with different background colors to enhance the visual appeal.

Celestial Bodies:

- Draw at least three celestial bodies (e.g., sun, moon, stars) using points.
- Experiment with sizes and colors to distinguish the celestial bodies.

Connecting Lines:

- Draw connecting lines between the celestial bodies using lines.
- Explore different line styles and colors.

Landform:

- Add a simple landform using triangles.
- The landform could be a hill, mountain, or any creative shape.

FLAME UNIVERSITY | EVERLASTING learning

THANK YOU