**Computer Graphics LAB**

**(**Experiment 1**)**

|  |
| --- |
| **Name -** Ashish Sharma  **SAP ID -** 500087115  **Batch -** B4 |

**Ques 1)** What is OpenGL? Discuss its uses and properties.

OpenGL(Open Graphics Library ) is a cross-language (language independent), cross-platform (platform independent) API for rendering 2D and 3D vector graphics. The OpenGL API is designed mainly in hardware.

**Uses of OpenGL:**

OpenGL is a C-language API for controlling hardware-accelerated graphics. Most computers have a GPU, and OpenGL is one way to use it. Basically, for anything graphic-related. OpenGL is likely to be used in your favourite mobile game. The majority of iPhone animations also make use of OpenGL. The majority of games on your PC make use of OpenGL.

**Features of OpenGL:**

1. Low-Level :

One critical goal of OpenGL is to provide device independence while still allowing full hardware contact. As a result, the API allows graphics operations at the lowest level while maintaining device independence.

1. Fine-Grained Control :

The API suggests that you state entity parts of geometric entities and perform operations on them. This fine-grained control is required so that these mechanisms and operations can be defined in any order and rendering operations can be controlled comfortably to meet the needs of various applications.

1. Not Programmable :

There is no programming language provided by OpenGL. Its function can be organised by turning actions on and off or by assigning factors to operations, but the rendering algorithms are mostly fixed.

1. Geometry and Images :

OpenGL gives support for managing both 3D and 2D geometry.

1. Frame buffer :

The majority of OpenGL requires that the graphics hardware have a frame buffer. This is a reasonable scenario because almost all interactive graphics are run on systems with frame buffers. Some OpenGL actions are only accomplished while exposing their execution via a frame buffer.

1. Modal
2. Based on IRIS GL

**Ques 2)** What are different popular libraries in OpenGL? Describe them

Out of many libraries, 3 most popular libraries are as follows ;

1. **GL:** This OpenGL library is used for ‘Primitives’, i.e. for creation of point, line, polygons. Also for shading and colouring, translation, rotation, scaling, viewing, clipping, texture, hidden surface removal and many such purposes.
2. **GLU:** Viewing; perspective, orthographic, images

The GLU is built using low-level OpenGL commands and is included with OpenGL. It includes routines for generating viewing and projection matrices, as well as polygonal tesselation and surface rendering.

When using the GLU library you will need to #include <GL/glu.h> in your program. GLU routines begin with the prefix **glu**.

1. **GLUT:** OpenGL Utility toolkit

Windowing toolkit; (Key, Mouse Handler, Window Events etc…)

The GLUT is a window system-independent toolkit designed to hide the complexities of various window system APIs. Window definition, window control, and keyboard and mouse input monitoring are among the functions performed. GLUT also provides limited support for the creation of pop-up menus.

to use the GLUT library you will need use #include <GL/glut.h> instead of #include <GL/gl.h> in your program. GLUT routines begin with the prefix **glut**.

1. **GLX:** OpenGL Extention to the X Window System

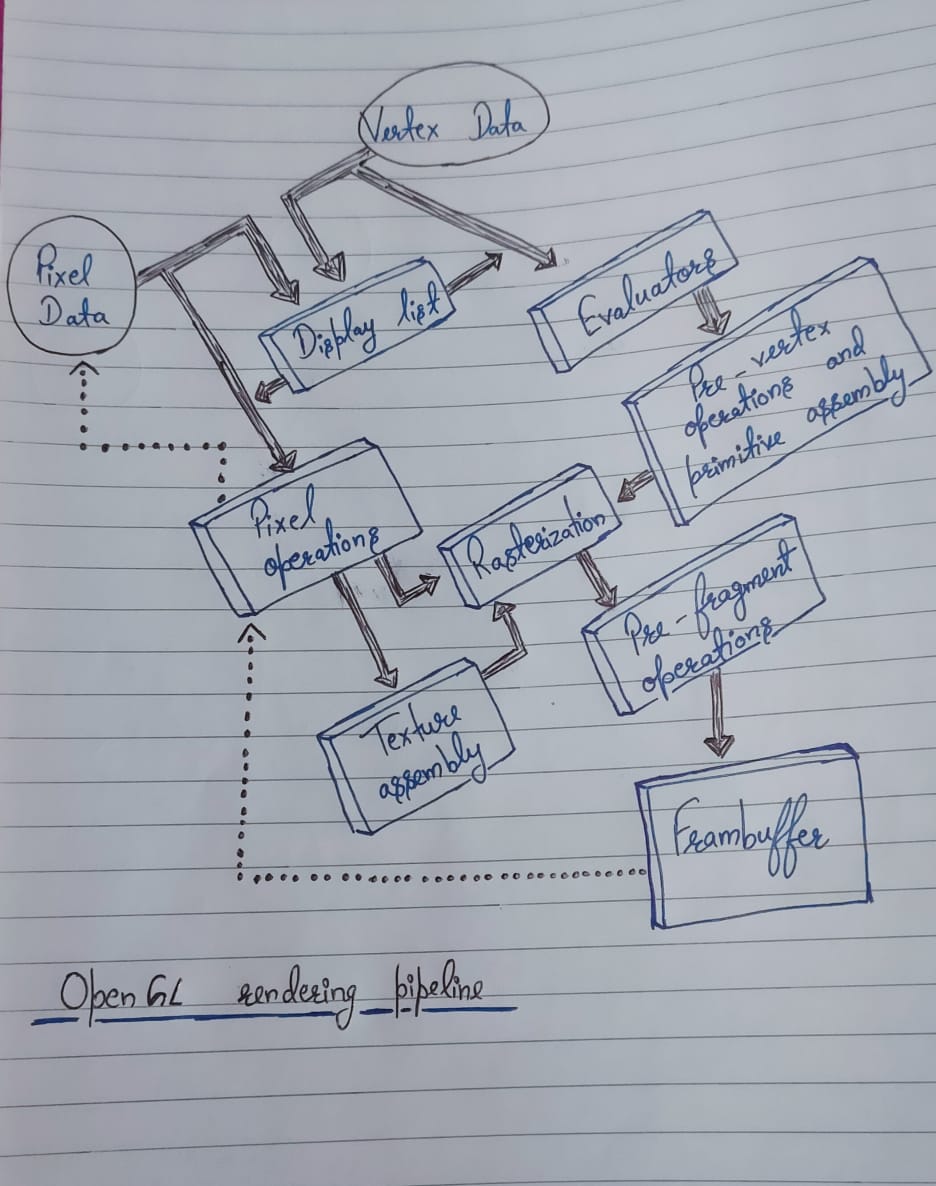
Because OpenGL lacks routines for handling windowing operations, the GLX was created to fill this void in the X Window System. We

can use GLX to enable your OpenGL programmes to draw into an X Window System

window.

GLX routines begin with the prefix **glx**.

**Ques 3)** Discuss about OpenGL rendering pipeline.



* **Display Lists -**

All data, regardless of whether it describes geometry or pixels, can be saved in a display list for immediate or future use. When a display list is executed, the retained data is sent from the display list in the same way that it would be sent by an application in immediate mode.

**{***The alternative to retaining data in a display list is processing the data irnmediately-also known as immediate mode.***}**

* **Evaluators**

Evaluators provide a method for deriving the surface vertices from the control points. The method is polynomial mapping and is used to generate surface normals, texture coordinates, colours, and spatial coordinate values from control points.

* **Per-Vertex Operations**

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives.

* **Primitive Assembly**

Clipping, a key component of primitive assembly, is the removal of geometry that extends beyond a half-space defined by a plane. Some other operation of Primitive Assembly are “line or polygon clipping”, “viewport and depth(i.e. z coordinate )”. This stage yields complete geometric primitives, which are transformed and clipped vertices with associated colour, depth, and, in some cases, texture-coordinate values, as well as guidelines for the “*rasterization step”*.

* **Pixel Operations**

While geometric data takes one path through the OpenGL rendering pipeline, Pixels from an array in system memory are first unpacked into the appropriate number of components from one of several formats. The data is then scaled, skewed, and processed with a pixel map. Clamping occurs before the results are either written into texture memory or transmitted to the rasterization stage.

Pixel-transfer procedures (scale, bias, mapping, and clamping) are done when pixel data is read from the frame buffer. These findings are packaged in a suitable format and returned to a system memory array. To copy data in the framebuffer to other areas of the framebuffer or to the texture memory, special pixel copy procedures are available. Before the data is written to the texture memory or returned to the framebuffer, a single pass through the pixel transfer operations is performed.

* **Texture Assembly**

To make geometric objects look more realistic, an OpenGL application may wish to apply texture pictures to them. If various texture images are used, it is best to put them into texture objects so that we can easily swap between them. Some OpenGL implementations may include additional resources to improve texturing performance. It's possible that there's specialised, high-performance texture memory.

* **Rasterization**

Rasterization is the process of conversion of both geometric and pixel data into fragments. Each square fragment corresponds to a pixel in the framebuffer. Each fragment square is given a colour and depth value.

* **Fragment Operations**

Before values are placed in the framebuffer, a sequence of actions are carried out that may alter or even discard fragments. All of these operations have the option of being enabled or disabled. The first operation that may be encountered is texturing, which involves generating a texel (texture element) from texture memory for each fragment and applying it to the fragment. After that, fog calculations might be used, followed by the scissor test, alpha test, stencil test, and depth-buffer test (the depth buffer is for hidden-surface removal). Failure to pass an enabled test may result in the termination of further processing of a fragment's square. Blending, dithering, logical operations, and bitmask masking can then be performed.

Finally, the thoroughly processed fragment is inserted into the proper buffer, where it has advanced to the level of a pixel and found its final resting place.