# INTRODUCTION

OpenGL is an open specification for an applications program interface for defining 2D and3D objects. The specification is cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. It renders 3D objects to the screen, providing the same set of instructions on different computers and graphics adapters. Thus it allows us to write an application that can create the same effects in any operating system using any OpenGL-adhering graphics adapter.

Computer graphics, a 3-dimensional primitive can be anything from a single point to an nsided polygon. From the software standpoint, primitives utilize the basic 3-dimensional rasterization algorithms such as Bresenham's line drawing algorithm, polygon scan line fill, texture mapping and so forth. OpenGL's basic operation is to accept primitives such as points, lines and polygons, and convert them into pixels. This is done by a graphics pipeline known as the OpenGL state machine. Most OpenGL commands either issue primitives to the graphics pipeline, or configure how the pipeline processes these primitives.

OpenGL is a low-level, procedural API, requiring the programmer to dictate the exact steps required to render a scene. OpenGL's low-level design requires programmers to have a good knowledge of the graphics pipeline, but also gives a certain amount of freedom to implement novel rendering algorithms.

* 1. **Purpose:**

The aim of this project is to develop a 2-D graphics package which supports basic operations which include creating objects like lines, circles, polygons, spirals, etc and also transformation operations like translation, rotation, etc on such objects. The package must also have a user-friendly interface that may be menu-oriented, iconic or a combination of both.

* 1. **Introduction on OpenGL :**

OpenGL provides the programmer with an interface to graphics hardware. It is a powerful, low-level rendering and modeling software library, available on all major platforms, with wide hardware support. It is designed for use in any graphics applications, from games to modeling to CAD.

OpenGL intentionally provides only low-level rendering routines, allowing the programmer a great deal of control and flexibility. The provided routines can easily be used to build high-level rendering and modeling libraries, and in fact, the OpenGL Utility Library (GLU), which is included in most OpenGL distributions, does exactly that. Note also that OpenGL is just a graphics library; unlike DirectX, it does not include support for sound, input, networking, or anything else not directly related to graphics.

**1.2.1 OpenGL History**

OpenGL was originally developed by Silicon Graphics, Inc. (SGI) as a multi-purpose, platform-independent graphics API. Since 1992, the development of OpenGL has been overseen by the OpenGL Architecture Review Board (ARB), which is made up of major graphics vendors and other industry leaders, currently consisting of ATI, Compaq, Evans & Sutherland, Hewlett-Packard, IBM, Intel, Intergraph, nVidia, Microsoft, and Silicon Graphics. The role of the ARB is to establish and maintain the OpenGL specification, which dictates which features must be included when one is developing an OpenGL distribution.

Because OpenGL is designed to be used with high-end graphics workstations, it has, until recently, included the power to take full advantage of consumer-level graphics hardware. Furious competition over the last couple of years, however, has brought features once available only on graphics workstations to the consumer level; as a result, there are more and more video cards of which OpenGL can't take full advantage. Eventually, these extensions may become official additions to the OpenGL standard. OpenGL 1.2 was the first version to contain support for features specifically requested by game developers (such as multitexturing), and it is likely that future releases will be influenced by gaming as well.

**1.2.2 OpenGL Architecture**

OpenGL is a collection of several hundred functions providing access to all the features offered by your graphics hardware. Internally, it acts as a state machine--a collection of states that tell OpenGL what to do. Using the API, you can set various aspects of the state machine, including such things as the current color, lighting, blending, and so on. When rendering, everything drawn is affected by the current settings of the state machine. It's important to be aware of what the various states are, and the effect they have, because it's not uncommon to have unexpected results due to having one or more states set incorrectly.

At the core of OpenGL is the rendering pipeline, as shown in Figure 2.1. You don't need to

understand everything that happens in the pipeline at this point, but you should at least be aware

that what you see on the screen results from a series of steps. Fortunately, OpenGL handles most

of these steps for you.

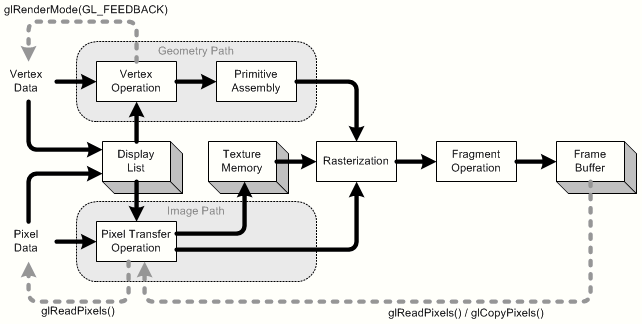
****

Fig 1.1 The OpenGL rendering pipeline.

Under Windows, OpenGL provides an alternative to using the Graphics Device Interface (GDI). GDI architects designed it to make the graphics hardware entirely invisible to Windows programmers. This provides layers of abstraction that help programmers avoid dealing with device-specific issues.

However, GDI is intended for use with applications and thus lacks the speed required for games. OpenGL allows you to bypass GDI entirely and deal directly with graphics hardware. Figure 2.2 illustrates the OpenGL hierarchy under Windows.

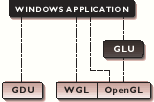


Fig 1.2 OpenGL API hierarchy under Windows systems.

**1.2.3 The OpenGL Utility Library**

The OpenGL Utility Library, or GLU, supplements OpenGL by providing higher-level functions. GLU offers features that range from simple wrappers around OpenGL functions tocomplex components supporting advanced rendering techniques. Its features include:

* 2D image scaling
* Rendering 3D objects including spheres, cylinders, and disks
* Automatic mipmap generation from a single image
* Support for curves surfaces through NURBS
* Support for tessellation of non-convex polygons
* Special-purpose transformations and matrices

**1.2.4 What Is GLUT?**

GLUT, short for OpenGL Utility Toolkit, is a set of support libraries available on every major platform. OpenGL does not directly support any form of windowing, menus, or input. That's where GLUT comes in. It provides basic functionality in all of those areas, while remaining platform independent, so that you can easily move GLUT-based applications from, for example, Windows to UNIX with few, if any, changes.

GLUT is easy to use and learn, and although it does not provide you with all the functionality the operating system offers, it works quite well for demos and simple applications.

* 1. **Significance of OpenGL :**
* With different 3D accelerators, by presenting the programmer To hide the complexities of interfacing with a single, uniform API.
* To hide the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set (using software emulation if necessary).
* OpenGL is a well documented API.
* OpenGL is also a clean API and much easier to learn and program.
* OpenGL has the best demonstrated 3D performance for any API.
* OpenGL has a conformance suite to validate that OpenGL implementations correctly implement OpenGL.
  1. **Design Aspects of the Project Using OpenGL :**

The Graphics Package is designed using the in built graphics library. The objects, which can be drawn using the editor, are stored as functions that can be used according to the requirements.

We can say that based on the design philosophy used during their implementation, the graphics editors can be of two main types:

One is an Object oriented editor where in each thing drawn in the view port is an object. Such objects can be selected individually and can be subjected to any of the transformations provided in the editor. The advantage of such editors is that the code can be easily written in using an Object Oriented Programming language like C++. Also undo functionalities can be easily implemented because all that the editor has to do is to keep a stack of objects being drawn on the screen. The disadvantage is that the user can only select objects and not a part of the screen.

The other kind of an editor is a pixel-based editor where in drawing anything on the view port is like painting on a canvas. Once an object is drawn it cannot be individually selected. Instead only a rectangular portion of the screen can be usually selected and subjected to various transformations or other operations. In other words the smallest object that can be selected and modified is a pixel. The basic advantage is that of the simplicity in code of such an editor where in the smallest unit is a pixel. The disadvantage being that an individual object cannot be selected and subjected to transformations.

Given the advantages and disadvantages of the two ways of implementing, the programmer is free to choose one that is more appealing to him .The end product needs to be a user-friendly interface. Ease of understanding and speed of working are two main requirements for it, which should be kept in mind during each phase of design and implementation.