TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 OPENGL

OpenGL is the abbreviation for Open Graphics Library. It is a software interface for graphics hardware. This interface consists of several hundred functions that allow you, a graphics programmer, to specify the objects and operations needed to produce high-quality color images of two-dimensional and three-dimensional objects. Many of these functions are actually simple variations of each other, so in reality there are about 120 substantially different functions. The main purpose of OpenGL is to render two-dimensional and three-dimensional objects into the frame buffer. These objects are defined as sequences of vertices (that define geometric objects) or pixels (that define images). OpenGL performs several processes on this data to convert it to pixels to form the final desired image in the frame buffer.

1.2 HISTORY

As a result, SGI released the **OpenGL** standard in the 1980s, developing software that could function with a wide range of graphics hardware was a real challenge. Software developers wrote custom interfaces and drivers for each piece of hardware. This was expensive and resulted in much duplication of effort.

By the early 1990s, Silicon Graphics (SGI) was a leader in 3D graphics for workstations. Their IRIS GL API was considered the state of the art and became the de facto industry standard, overshadowing the open standards-based PHIGS. This was because IRIS GL was considered easier to use, and because it supported immediate mode rendering. By contrast, PHIGS was considered difficult to use and outdated in terms of functionality.

SGI's competitors (including Sun Microsystems, Hewlett-Packard and IBM) were also able to bring to market 3D hardware, supported by extensions made to the PHIGS standard.

This in turn caused SGI market share to weaken as more 3D graphics hardware suppliers entered the market. In an effort to influence the market, SGI decided to turn the Iris GL API into an open standard.

SGI considered that the Iris GL API itself wasn't suitable for opening due to licensing and patent issues. Also, the Iris GL had API functions that were not relevant to 3D graphics. For example, it included a windowing, keyboard and mouse API, in part because it was developed before the X Window System and Sun's NEWS systems were developed.

In addition, SGI had a large number of software customers; by changing to the OpenGL API, they planned to keep their customers locked onto SGI (and IBM) hardware for a few years while market support for OpenGL matured. Meanwhile, SGI would continue to try to maintain their customers tied to SGI hardware by developing the advanced and proprietary Iris Inventor and Iris Performer programming APIs.

1.3 FEATURES OF OPENGL

- **Industry standard**

  An independent consortium, the OpenGL Architecture Review Board, guides the OpenGL specification. With broad industry support, OpenGL is the only truly open, vendor-neutral, multiplatform graphics standard.

- **Stable**

  OpenGL implementations have been available for more than seven years on a wide variety of platforms. Additions to the specification are well controlled, and proposed updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure that existing applications do not become obsolete.

- **Reliable and portable**

  All OpenGL applications produce consistent visual display results on any OpenGL API- compliant hardware, regardless of operating system or windowing system.

- **Evolving**

  Because of its thorough and forward-looking design, OpenGL allows new hardware

innovations to be accessible through the API via the OpenGL extension mechanism. In this way, innovations appear in the API in a timely fashion, letting application developers.

- **Scalable**

  OpenGL API-based applications can run on systems ranging from consumer electronics to PCs, workstations, and supercomputers. As a result, applications can scale to any class of machine that the developer chooses to target.

- **Easy to use**

  OpenGL is well structured with an intuitive design and logical commands. Efficient OpenGL routines typically result in applications with fewer lines of code than those that make up programs generated using other graphics libraries or packages. In addition, OpenGL drivers encapsulate information about the underlying hardware, freeing the application developer from having to design for specific hardware features.

- **Well-documented**

  Numerous books have been published about OpenGL, and a great deal of sample code is readily available, making information about OpenGL inexpensive and easy to obtain.

1.4 LIBRARIES

Most of our applications will be designed to access OpenGL directly through functions in three libraries. They are

- **GL – Graphics Library**

  Functions in the main GL (or OpenGL in Windows) library have names that begin with the letters gl and are stored in a library usually referred to as GL (or OpenGL in Windows).

- **GLU – Graphics Utility Library**

This library uses only GL functions but contain code for creating common objects and simplifying viewing. All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begins with the letters glu.

- **GLUT – OpenGL Utility Toolkit**

To interface with the window system and to get input from external devices into our programs we need at least one more library. For the X window System, this library is called GLX, for Windows, it is wgl, and for the Macintosh, it is agl. Rather than using a different library for each system, we use a readily available library called the OpenGL Utility Toolkit (GLUT), which provides minimum functionality that should be expected in any modern windowing system.

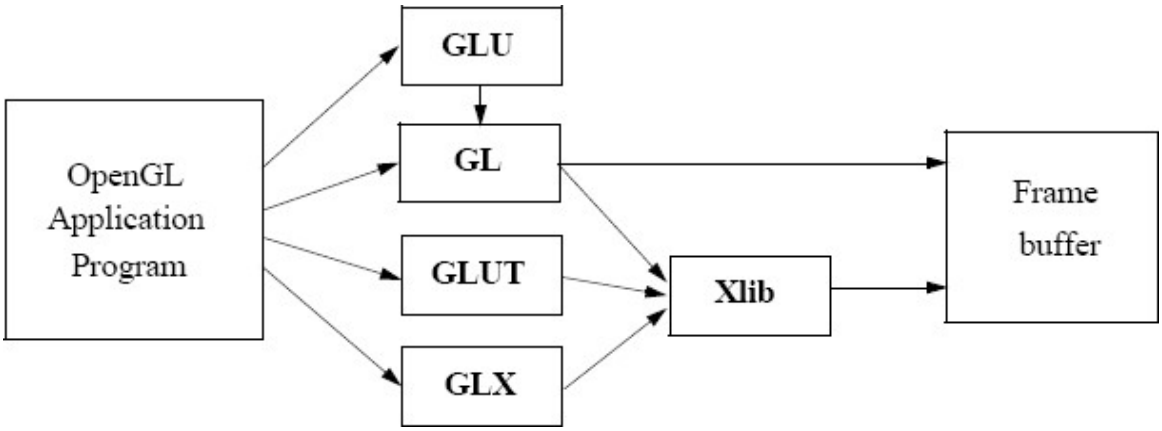

Fig: 1.5 Library Organization

The above figure shows the organization of the libraries for an X Window System environment.

#include<GL/glut.h> is sufficient to read in glut.h, gl.h and glu.h.

1.5 GRAPHICS FUNCTIONS

Our basic model of a graphics package is a black box, a term that engineers use to denote a system whose properties are described only by its inputs and outputs; we may

know nothing about its internal workings.

OpenGL functions can be classified into seven major groups:

- **Primitive function**

The primitive functions define the low-level objects or atomic entities that our system can display. Depending on the API, the primitives can include points, lines, polygons, pixels, text, and various types of curves and surfaces.

- **Attribute functions**

If primitives are the what of an API – the primitive objects that can be displayed- then attributes are the how. That is, the attributes govern the way the primitive appears on the display. Attribute functions allow us to perform operations ranging from choosing the color with which we display a line segment, to picking a pattern with which to fill inside of a polygon.

- **Viewing functions**

The viewing functions allow us to specify various views, although APIs differ in the degree of flexibility, they provide in choosing a view.

- **Transformation functions**

One of the characteristics of a good API is that it provides the user with a set of transformations functions such as rotation, translation and scaling.

- **Input functions**

For interactive applications, an API must provide a set of input functions, to allow users to deal with the diverse forms of input that characterize modern graphics systems.

We need functions to deal with devices such as keyboards, mice and data tablets.

- **Control functions**

These functions enable us to communicate with the window system, to initialize our programs, and to deal with any errors that take place during the execution of our programs.

- **Query functions**

If we are to write device independent programs, we should expect the implementation of the API to take care of the differences between devices, such as how many colors are supported or the size of the display. Such information of the particular implementation should be provided through a set of query functions.

CHAPTER 4

IMPLEMENTATION

4.1 OVERVIEW

This project is a demonstration of "**Maze Game**". We have taken the help of built in functions present in the header file. To provide functionality to our project we have written sub functions. These functions provide us the efficient way to design the project. In this chapter we are describing the functionality of our project using these functions.

Keyboard interactions are provided where, when a Enter button is pressed, menu displays and we can select options from menu displayed.

4.2 BUILTIN FUNCTIONS

- void glColor3f(GLfloat red, GLfloat green, GLfloat blue): This function sets present RGB colours. Different colour is given to object using the colours parameters such as red, green, blue. The maximum and minimum values of the floating point types are 1.0 and 0.0 respectively.
- void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far): Defines the orthographic volume with all the parameters measured from the centre of projection of the plane.
- void glLineWidth(width): Line width is set in OpenGL with function glLineWidth(width). Floating point value is assigned to parameters width, and this value is rounded to nearest non-negative integer.
- void LoadIdentity(): Sets the current transformation matrix to an identity matrix.
- void glClear(GL_COLOR_BUFFER_BIT): Clears all buffer whose bits are set in mask. The mask is formed by the logical OR of values defined in gl.h GL_COLOR_BUFFER_BIT refers to colour buffers.

- void glBegin( ) and void glEnd( ): The glBegin and glEnd functions delimit the vertices of a primitive or a group of like primitives. Syntax of glBegin is as follows: glBegin(GLenum mode);
- void glFlush(void): The glFlush function forces execution of OpenGL function in finite time. This function has no parameters. This function does not return a value.
- void MatrixMode(GL_PROJECTION): Projection matrixes are stored inOpenGL projection mode. So to set the projection transformation matrix. That mode is invoked through the statement, glMatrixMode(GL_PROJECTION);
- void glutDisplayFunc(): Sets the display call back for the current window.
- void glutPostRedisplay(): Mark the normal plane of current window as needing to be redisplayed. The next iteration through glutMainLoop, the window's display call back will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display call back opportunity generates only a single redisplay call back. GlutPostRedisplay may be called within a window's display or overlay display call back to remark that windows for redisplay.
- void glutInit(int *argc, char **argv): The glutInit will initialize the GLUT library, the arguments form main are passed in and can used by the application. This will negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to user if GLUT cannot be properly initialized.
- void glutInitDisplayMode(unsigned int mode): The glutInitDisplayMode sets the initial display mode. It requests a display with the properties in mode. The value of mode is determined by the logical OR of option including the colour model(GLUT_RGB), buffering (GLUT_DOUBLE) and (GLUT_DEPTH).
- void glutInitWindowSize(int width, int height): This function specifies the initial height and width of the width in pixels.
- void glutInitWindowPosition(int x, int y): This function specifies the initial position of the top-left corner of the windows in pixels.
- void glutCreateWindow(char *title): The glutCreateWindow create a window on the display. The string title can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.
- void glutMainLoop(): Cause the program to enter an event-processing loop. It should be the last statement in the main.

- void glutKeyboardFunc(void(*func)(unsigned char key,int x,int y)): This function is used to tell the windows system which function we want to process the normal key events.

4.3 USER-DEFINED FUNCTIONS:

Explain user defined functions like this

- **myinit():** This function is used to initialize the graphics window.glMatrixMode(GL_PROJECTION), glLoadIdentity() are used to project the output on to the graphics window.

- **display ():** If df==10, i.e., it will call the frontscreen(), else if df==0 then startscreen() is called, Now the game has been started with the timer of 60sec displaying the MAZE to be solved by the player.

- **panel():** This function is used to display the Wall forming the Maze.

- **mainscreen ():** This is the function which helps in opening the home page of the game. This page is linked to all other pages described before. After clicking enter in this page the Main page is opened.

- **idle () :** This function is the major criteria of this game as it sets a timer for the player which limits the player to finish his game within 60sec else the player loses the game.

- **point() :** This function is used to define the navigation point,

- **flag1() :** This function is used to draw a flag, which indicates the starting point of the game.
- **flag2() :** This function is used to draw a flag, which indicates the ending point of the game.

- **output() :** This function is used to print the string on the display window.

- **drawstring() :** This function is also used print the characters on the display window and is also same as the output() function, but it prints the contents with the different font style.

- **winscreen() :** This function is written for the page that is displayed after the player wins the game.

- **startscreen() :** This function is written for the page that is displayed next to the mainscreen page.

- **instructions() :** This function is used to display the instructions of the game.

- **timeover() :** This function is written for the page that is displayed if the player loses the game.

- **specialkey() :** The code in this function is written to specify the movements of the navigation point.

- **keyboard() :** This function is used to add the keyboard interactions to the project.

- **myreshape() :** This function defines what to do when the window is resized.