**R. V. COLLEGE OF ENGINEERING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**(Autonomous Institution under VTU)**

|  |
| --- |
| **Bangalore – 560 059** |



A Project report on

**“Graphics Editor Package using Open GL”**

Submitted in partial fulfilment of the requirements for the award of the degree of

Bachelor of Engineering

in

## Computer Science & Engineering

By

Anusha Ramachandra (1RV10CS023)

Harshita Megotia (1RV10CS039)

Under the guidance of

#### Mr Krishanappa Ms**. Chaitra**

#### Assistant Professor Assistant Professor

#### Department of CSE Department Of CSE

*Carried out at*

**Department Of Computer Science & Engineering**

**R. V. College of Engineering**

**Bangalore – 560 059**

**(Academic Year: 2012-13)**

**R. V. COLLEGE OF ENGINEERING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**(Autonomous Institution under VTU)**

|  |
| --- |
| **Bangalore – 560 059** |



**CERTIFICATE**

This is to certify that the project entitled **“Graphics Editor using OpenGL”** has been successfully carried out at R.V.C.E., Bangalore in partial fulfilment of the requirements for the award of degree in Computer Science and Engineering of Vishveshwarayya Technological University, under autonomous scheme, during the academic year 2013-2014.

By

Anusha Ramachandra (1RV10CS023)

Harshita Megotia (1RV10CS039)

**Lab Incharge HOD, Dept CSE, RVCE**

**Names of lab Examiners**   **Signature**

1. 1.

2. 2.

DECLARATION

We hereby declare that the project report entitled **"Graphics Package”** is the bonafide record of the project carried out at **R.V.COLLEGE OF ENGINEERING** in partial fulfilment of the requirements for the award of degree **Bachelor of Engineering in Computer Science and Engineering during the academic year 2013-14.**

We further declare that the project report is not submitted to any other university fulfilments of the requirements for the award of any degree.

Anusha Ramachandra (1RV10CS023)

Harshita Megotia (1RV10CS039)

**ACKNOWLEDGMENTS**

This project represents the fruits of months of work with the help, guidance and suggestions of many. Firstly, we would like to thank the Head of Department, Computer Science and Engineering, **Prof. Shobha**  for providing us the requisite digital and physical infrastructure to implement this project. We would like to thank **Ms.Krishnappa**  and **Mrs Chaitra** for their timely help and clarification. We would also like to thank the technical staff for the upkeep of the system in the laboratory. We would also like to thank the R. V. College of Engineering (Autonomous under V.T.U) for the opportunity to undertake this exercise as a part of the academic course. Lastly, we thank our parents and friends for the support and goodwill.

Anusha Ramachandra (1RV09CS023)

Harshita Megotia (1RV09CS039)

Table of Contents

[Computer Science & Engineering i](#_Toc373339448)

[1.INTRODUCTION 1](#_Toc373339449)

[1.1 Purpose: 2](#_Toc373339450)

[1.2 Introduction on OpenGL : 2](#_Toc373339451)

[1.2.1 OpenGL History 2](#_Toc373339452)

[1.2.2 OpenGL Architecture 3](#_Toc373339453)

[1.2.3 The OpenGL Utility Library 4](#_Toc373339454)

[1.2.4 GLUT 4](#_Toc373339455)

[1.3 Significance of OpenGL : 5](#_Toc373339456)

[1.4 Design Aspects of the Project Using OpenGL : 5](#_Toc373339457)

[2.SOFTWARE REQUIREMENT SPECIFICATIONS 7](#_Toc373339458)

[2.1 Overall Description: 7](#_Toc373339459)

[2.1.1 Product Perspective: 7](#_Toc373339460)

[2.1.2 Product Functions: 7](#_Toc373339461)

[2.1.3 User Characteristics: 7](#_Toc373339462)

[2.2 Specific Requirements: 7](#_Toc373339463)

[2.2.1 Software Requirements: 7](#_Toc373339464)

[2.2.2 Hardware Requirements: 8](#_Toc373339465)

[2.2.3 Supportability 8](#_Toc373339466)

[2.3 Design Constraints: 8](#_Toc373339467)

[2.4 Interfaces : 9](#_Toc373339468)

[2.4.1 User Interfaces: 9](#_Toc373339469)

[3. DETAILED DESIGN 10](#_Toc373339470)

[3.1 Window design: 10](#_Toc373339471)

[3.2 Main Algorithm: 10](#_Toc373339472)

[3.3 Data Structures Used: 10](#_Toc373339473)

[3.4 Other Algorithms used 11](#_Toc373339474)

[3.4.1 Free- hand Drawing: 11](#_Toc373339475)

[3.4.2 Translation: 11](#_Toc373339476)

[3.4.3 Scaling: 11](#_Toc373339477)

[4. IMPLEMENTATION 12](#_Toc373339478)

[4.1 Structures used 12](#_Toc373339479)

[4.1.1 Buffer: 12](#_Toc373339480)

[4.2 Text: 12](#_Toc373339481)

[4.3 Button : 12](#_Toc373339482)

[4.4 Color palette : 13](#_Toc373339483)

[4.5 File options: 13](#_Toc373339484)

[4.6 Function to draw objects using basic primitives: 13](#_Toc373339485)

[4.6.1 Function to draw a Pencil 13](#_Toc373339486)

[4.6.2 Function to draw a Line 13](#_Toc373339487)

[4.6.3 Function to draw a Rectangle 13](#_Toc373339488)

[4.6.4 Function to Render a Text on the Screen 14](#_Toc373339489)

[4.6.5 Function to draw a Circle 14](#_Toc373339490)

[4.7 2D drawing : 14](#_Toc373339491)

[4.8 Clipping : 14](#_Toc373339492)

[4.8.1 Inside Clipping: 14](#_Toc373339493)

[4.8.2 Outside Clipping: 14](#_Toc373339494)

[4.9 Window Management : 14](#_Toc373339495)

[4.10 The Display Callback : 15](#_Toc373339496)

[Running the Program : 16](#_Toc373339497)

[4.11 OpenGL Geometric Drawing Primitives : 16](#_Toc373339498)

[5. CONCLUSION 17](#_Toc373339499)

[5.1 Summary : 17](#_Toc373339500)

[5.2 Limitations : 17](#_Toc373339501)

[5.3 Future Enhancements : 17](#_Toc373339502)

[6. BIBLIOGRAPHY 18](#_Toc373339503)

[APPENDIX -A SNAPSHOTS 19](#_Toc373339504)

**ABSTRACT**

The Graphics Editor is developed using Open GL libraries. The Graphics Editor is aimed at supporting the requirements of the Computer Graphics package. It is a mini Graphics Editor project which is made up by using various algorithms like circle drawing algorithm, clipping algorithm etc.

The editor has been developed using Visual Studio 2010. The various features have been supported by an easy to understand mouse interface. All the tools have been organized under the toolbar. The color palette has been organized under the color palette. The drawing work is carried out in the sub window which is a part of the main window. The drawing features supported are Line, Rectangle, Brush, freehand drawing, circle, clipping and eraser.

This project aims to develop a 2-D graphics package which supports basic operations which include creating objects like lines, circles, polygons etc. and also transformation operations like translation, scaling, 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

**OpenGL** (**Open G**raphics **L**ibrary) is a [cross-language](http://en.wikipedia.org/wiki/Language-independent_specification), [multi-platform](http://en.wikipedia.org/wiki/Cross-platform) [application programming interface](http://en.wikipedia.org/wiki/Application_programming_interface) (API) for [rendering](http://en.wikipedia.org/wiki/Rendering_(computer_graphics)) [2D](http://en.wikipedia.org/wiki/2D_computer_graphics) and [3D computer graphics](http://en.wikipedia.org/wiki/3D_computer_graphics). The API is typically used to interact with a [Graphics processing unit](http://en.wikipedia.org/wiki/Graphics_processing_unit) (GPU), to achieve [hardware-accelerated](http://en.wikipedia.org/wiki/Hardware_acceleration) [rendering](http://en.wikipedia.org/wiki/Rendering_(computer_graphics)). OpenGL is an open specification for an applications program interface for defining 2D and3D objects. 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 n sided 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.

The OpenGL specification describes an abstract [API](http://en.wikipedia.org/wiki/Application_programming_interface) for drawing 2D and 3D graphics. Although it's possible for the API to be implemented entirely in software, it's designed to be implemented mostly or entirely [in hardware](http://en.wikipedia.org/wiki/Hardware_acceleration).

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. In addition to being language-independent, OpenGL is also platform-independent. The specification says nothing on the subject of obtaining, and managing, an OpenGL context, leaving this as a detail of the underlying [windowing system](http://en.wikipedia.org/wiki/Windowing_system). For the same reason, OpenGL is purely concerned with rendering, providing no APIs related to input, audio, or windowing.

## 1.1 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.

## 1.2 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, we can set various aspects of the state machine, including such things as the current colour, 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 1.1. OpenGL handles most of these steps for users.



**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 to complex 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 GLUT

GLUT (pronounced like the glut in gluttony) is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that works across all PC and workstation OS platforms.

GLUT is designed for constructing small to medium sized OpenGL programs. While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits. GLUT is simple, easy, and small.

The toolkit supports:

* Multiple windows for OpenGL rendering
* Callback driven event processing
* Sophisticated input devices
* An 'idle' routine and timers
* A simple, cascading pop-up menu facility
* Utility routines to generate various solid and wire frame objects
* Support for bitmap and stroke fonts
* Miscellaneous window management functions

## 1.3 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.

## 1.4 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.

# 

# 2.SOFTWARE REQUIREMENT SPECIFICATIONS

## 2.1 Overall Description:

### 2.1.1 Product Perspective:

This is a graphics editor that enables the user to input graphical data. Using the editor, the user can also save the information input by him/her into files or open existing files for editing. This editor provides a graphical user - friendly interface to create and edit the files.

### 

### 2.1.2 Product Functions:

As mentioned previously, the main objective of the editor is to help the user to input graphical data and edit it conveniently. For ease in input and to help the user to traverse through the text easily, the editor provides functionality through the mouse.

### 2.1.3 User Characteristics:

The editor provides a very easy-to-use interface and does not expect any extra technical knowledge from the user. A basic understanding of all the options provided in the editor would facilitate him in using the editor to the best possible extent. Since it is a mouse-driven interface it is sufficiently easy for any kind of end user to run it.

## 2.2 Specific Requirements:

### 2.2.1 Software Requirements:

* An MS-DOS based operating system like Windows 98, Windows 2000 or Windows XP is the platform required to develop the 2D and 3D graphics applications.
* A Visual C/C++ compiler is required for compiling the source code to make the executable file which can then be directly executed.
* A built in graphics library like glut and glut32, and header file like glut.h and also dynamic link libraries like glut and glut32 are required.

### 2.2.2 Hardware Requirements:

The hardware requirements are very minimal and the software can run on most of the machines.

* Processor - Intel 486/Pentium processor or above.
* Processor Speed - 500 MHz or above
* RAM - 64MB or above Storage Space - 2 MB or above
  + Monitor resolution - A color monitor with a minimum resolution of 640\*480.

### 2.2.3 Supportability

* Good coding standards must be followed. The naming convention must be such that the names of the variables and functions used should indicate their purpose.
* One or two lines of documentation must be provided along with the functions to indicate what they are trying to achieve. Documentation must be provided for every module.

## 2.3 Design Constraints:

* As the software is being built to run on a DOS platform, which gives access to a maximum of only 640kB of conventional memory, efficient use of the memory is very important.
* As the software needs to be run even on low-end machines the code should be efficient and optimal with the minimal redundancies.
* Needless to say, the editor should also be robust and fast.
* It is assumed that the standard output device, namely the monitor, supports colors.
* One of the assumptions made in the file saving and retrieval process is that the required file is in the current directory.
* The user's system is required to have the C++ compiler of the appropriate version.
* The system is also expected to have a mouse connected since most of the drawing and other graphical operations implemented assume the presence of a mouse.

## 2.4 Interfaces :

#### This section deals with the interfaces that must be supported by the application. It includes the user interfaces that are to be implemented by the system.

### 2.4.1 User Interfaces:

The interface for the editor requires for the user to have a mouse connected, and the corresponding drivers installed. This is because most of the implementation details require and presume the presence of a mouse. For the convenience of the user, there are palettes and icons displayed on the screen.

**Icons:**

The icons consists of the file saving and opening options. It also has the 'exit' option for termination of the editor's execution.

**Palettes:**

* Color palette, which displays the different colors available to the user. He/she can pick the required color by clicking on the particular block.
* Shapes palette, which displays the various shapes that can be drawn by the user. This typically consists of line, circle, rectangle and freehand. User can choose the required shape by clicking on the icon representing the shape.
* 2D Transformations palette gives options like translation and rotation for the 2D objects created on the canvas. Again the user has to click on the icon to select the particular transformation.

# 

# 3. DETAILED DESIGN

## 

## 3.1 Window design:

Graphics editor uses only one window for all its purposes. The part of the window is used as canvas for drawing, container for holding many tools like color palette, writing tools like pencil, brush etc…

One of the most important things is that window is portable and flexible. The positioning of controls inside the window is generalized. The controls are placed with respect to the relative values of the window width and height. This means that the editor is portable to any system having different resolutions.

## 3.2 Main Algorithm:

The main algorithm for the graphics editor is the one shown below.

1. Initiate the Opengl graphics mode.

2. Initiate the mouse interface and keyboard interfaces.

3. Draw the toolbar icons, menus, and the main window.

4. Perform any drawing or editing on the canvas using any of the tools.

5. Close the application after releasing any dynamically used resources are released.

As we see above specific functions are registered for mouse and keyboard events. These functions are generally call back functions which are called upon receiving specific events from mouse and keyboard. Upon receiving inputs from these functions , specific actions are performed based on the mouse position.

## 3.3 Data Structures Used:

There are no other data structures explicitly used. This is mainly because, the editor is designed for area based operations and not object-based operations. So it is not necessary maintain information about objects drawn on the drawing area.

## 3.4 Other Algorithms used

Various algorithms have been used in this editor to provide the functionalities it boasts of. Few of them have been explained here.

### 3.4.1 Free- hand Drawing:

Free- hand drawing is a special case of polyline drawing where we have a large number of points obtained by continuously poling the mouse position during the time the mouse' left button is clicked.

### 3.4.2 Translation:

Translation is done by adding the required amount of translation quantities to each of the points of the objects in the selected area. If P(x,y) be the a point and (tx, ty) translation quantities then the translated point is given by P'(x,y) = p(x+tx,y+ty)

### 3.4.3 Scaling:

The scaling operation on an object can be carried out for an object by multiplying each of the points (x,y) by the scaling factors sx, sy.

Newx = oldx\*sx

Newy = oldy\*sy

# 

# 

# 4. IMPLEMENTATION

## 4.1 Structures used

There is only one structure used for designing our editor which is used to store the screen co-ordinates values. The objects of these basic structures are used in various part of the design. The following listing shows them.

### 4.1.1 Buffer:

Color is a basic structure for color parameters such as red, green, blue. The fig 4.1 shows the color structure. These structures objects are used in other basic functions to store color values required for the objects.



**Fig 4.1 Color structure**

## 4.2 Text:

The text is written on the drawing canvas using one of the library function available in Open Gl

“glutBitmapCharacter()”.

Before drawing the text, the pinter is set to proper position using the opengl function call

“glRasterPos2i()”

## 4.3 Button :

The buttons in this project are actually state-less objects. They do not graphically respond to the user input. But there input has been recognized by mouse callback function based on the position where user clicks the button.

## 4.4 Color palette :

Color palette is designed using the number of color buttons. The Fig 4.1 shows the color palette. The colors are generated randomly by varying the values of floating point numbers in the function “glColor3f()”. The color chosen by the user is recognized by the mouse callback function based on the co-ordinate points of the mouse.



**Fig 4.1 Color palette**

## 4.5 File options:

File menu have the options like New, Open, Save, and exit. The New option will create a new blank window for drawing.

The Open option allows opening the file by accepting the filename in command prompt. The file is read by using the “fread” function to a buffer and the buffer is drawn to a screen by using “glDrawPixels” function.

The save option will basically save the contents of the buffer to a file using the “fwrite” system call.

The exit option is used to quit the editor.

## 4.6 Function to draw objects using basic primitives:

### 4.6.1 Function to draw a Pencil

Pencil is used to draw a stream of points on the canvas. A function called “draw\_point” has been developed which basically draws a point. So this function is called repeatedly to imitate it as a pencil.

### 4.6.2 Function to draw a Line

Line drawing is done by using Opengl library functions and by passing the parameter “GL\_LINES” to “glBegin()” function.

### 4.6.3 Function to draw a Rectangle

The above function is achieved by using the opengl library function to draw a line loop.

### 4.6.4 Function to Render a Text on the Screen

The text is written on the drawing canvas using one of the library function available in Open Gl “glutBitmapCharacter()”.

### 4.6.5 Function to draw a Circle

Circle drawing is achieved by drawing points based on the angle and using circle’s parametric equation.

X=rcosθ

Y=rsinθ

## 4.7 2D drawing :

2D drawings are done in screen window that is used as canvas. The canvas is nothing but an empty space in a window where the drawing and various other actions are carried out.

## 4.8 Clipping :

There are two types of clipping implemented

* Inside Clipping
* Outside Clipping

### 4.8.1 Inside Clipping:

The area inside a selected region must be removed. And the rest of the area must be retained.

## Outside Clipping:

The area inside a selected region must be retained, rest all should be removed.

## 4.9 Window Management :

Five routines perform tasks necessary to initialize a window.

* **glutInit**(int \**argc*, char \*\**argv*) initializes GLUT and processes any command line arguments (for X, this would be options like -display and -geometry). **glutInit()** should be called before any other GLUT routine.
* **glutInitDisplayMode**(unsigned int *mode*) specifies whether to use an *RGBA* or color-index color model. You can also specify whether you want a single- or double-buffered window. (If you’re working in color-index mode, you’ll want to load certain colors into the color map; use **glutSetColor()** to do this.) Finally, you can use this routine to indicate that you want the window to have an associated depth, stencil, and/or accumulation buffer. For example, if you want a window with double buffering, the RGBA color model, and a depth buffer, you might call **glutInitDisplayMode**(*GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH*).
* **glutInitWindowPosition**(int *x*, int *y*) specifies the screen location for the upper-left corner of your window.
* **glutInitWindowSize**(int *width*, int *size*) specifies the size, in pixels, of your window.
* **glutCreateWindow**(char \**string*) creates a window with an OpenGL context. It returns a unique identifier for the new window. Until **glutMainLoop()** is called the window is not yet displayed.

## 4.10 The Display Callback :

* **glutDisplayFunc**(void (\* *func*)(void)) is the first and most important event callback function you will see. Whenever GLUT determines the contents of the window need to be redisplayed, the callback function registered by **glutDisplayFunc()** is executed. Therefore, you should put all the routines you

need to redraw the scene in the display callback function.

* **glutPostRedisplay**(void), which gives **glutMainLoop()** a nudge to call the registered display callback at its next opportunity. If your program changes the contents of the window, sometimes you will have to call this function.

# Running the Program :

The very last thing you must do is call **glutMainLoop**(void). All windows that have been created are now shown, and rendering to those windows is now effective. Event processing begins, and the registered display callback is triggered. Once this loop is entered, it is never exited!

## 4.11 OpenGL Geometric Drawing Primitives :

Now that you’ve seen how to specify vertices, you still need to know how to tell OpenGL to create a set of points, a line, or a polygon from those vertices. To do this, you bracket each set of vertices between a call to glBegin() and a call to glEnd(). The argument passed to glBegin() determines what sort of geometric primitive is constructed from the vertices.

**Point Details**

To control the size of a rendered point, use **glPointSize()** and supply the desired size in pixels as theargument.

* *void* ***glLoadMatrix****{fd}(const TYPE \*m);*

*Sets the sixteen values of the current matrix to those specified by m.*

* *void* ***glMultMatrix****{fd}(const TYPE \*m);*

*Multiplies the matrix specified by the sixteen values pointed to by m by the current matrix and stores the result as the current matrix.*

* *void* ***glLoadIdentity****(void);*

*Sets the currently modifiable matrix to the 4 × 4 identity matrix.*

* *void* ***glMatrixMode****(GLenum mode);*

*Specifies whether the modelview, projection, or texture matrix will be modified, using the mode attribute. Projecton matrix is used for 2-D pictures.*

# 5. CONCLUSION

## 5.1 Summary :

The editor requires no detailed knowledge about computer on the part of the user other than the ability to operate a computer i.e. entering the data from the keyboard. The mouse interface and keyboard interface makes it possible for even a novice user to use the graphics editor. Various predefined functions in opengl are used to provide options to users in the editor in a simpler way. Colour palette enhances the capability of users.

## 5.2 Limitations :

* Multiple workspaces are not provided.
* Limited features are provided.
* More than one file cannot be opened simultaneously.
* Concentrated more on 2D.

## 5.3 Future Enhancements :

* Pattern filling can be improved.
* Making the package more user friendly.
* Filling algorithms can be implemented for any polygon.
* Can be enhanced for the efficient usage of memory.
* More options on file can be provided

# 6. BIBLIOGRAPHY

[1]<http://www.files32.com/Types-Of-Clipping-In-Computer-Graphics.asp>

[2]Interactive Computer Graphics-A Top-Down Approach Using OpenGL by Edward Angel, Pearson Education, 5th Edition.

[3]Donald Hearn, M. Pauline Baker : Computer Graphics ‘C’ version Pearson Education, 2nd Edition.

[4]F. S. Hill, Jr: Computer Graphics Using OpenGL Pearson Education, 3rd Edition. Foley Van Dam Feiner Hughes : Computer Graphics Principles and practice Pearson Education, 2nd Edition in ‘C’.

* [5]<http://www.opengl.org/>

[6] <http://open.gl/>

* [7]<http://www.glprogramming.com/>

# APPENDIX -A SNAPSHOTS







