**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 pointer 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 4.7 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 to open 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.

**4.8.2 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.

**4.11 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!

**4.12 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 purposes whereas Modelview is used for 3-D.*