CHAPTER 5

IMPLEMENTATION

5.1 Header files Used

#include<stdio.h>

The standard input/output library requires the use of a header file called stdio.h. This include command is a directive that tells the compiler to use the information in the header file called stdio.h.The initial stdio stands for standard input output, and stdio.h file contains all the instructions the compiler needs to work with disk files and send information to the output device.

#include<GL/glut.h>

GL is the fundamental OpenGL library.it provides functions that are permanent part of OpenGL.The function starts with characters ‘gl’.

GLUT,the GL utility tool kit supports developing and managing menus and managing events. The functions start with characters ‘glut’.

GLU,the GL utility Library provides high-level routines to handle certain matrix operations, drawing of quadric surfaces such as sphere and cylinders. The functions start with characters ‘glu’.

#include<stdlib.h>

The ISO C standard this header file as a place to declare certain standard library fu8nctions.These include the memory management functions (malloc,free) communication with the environment(aqblort,exit)and others, not yet all the standard functions of this header file are supported, If a declaration is present in the supplied header file, then UCR supports it and will continue to support it.If a function is not there, it will be added in time.

5.2 Functions for the project

The Algorithm is constructed in a set of functions. We have used several user defined functions and built-in functions.

The following are the functions used in this project and their purpose **glutInitDisplayMode (GLUT_DOUBLE|GLUT_RGB)** specifies whether to use an *RGB* 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**).

5.3Display Callback

The **glutDisplayFunc** (void (*func)(void)) is the first and most important event callback in the code. Whenever GLUT determines the contents of the window need to be redisplayed, the callback function registered by **glutDisplayFunc**() is executed. Therefore, it registers display callback function.

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

5.4 Running the program

The very last thing to be done 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!

The various other functions used are described in more detail below:

1. **glutBitmapCharacter (font, string[size])** it renders the character in the named bitmap font and advances the current raster position.
2. **glutInit (&argc, char **argv):** this command initializes GLUT. The argument from main are passed in and be used by the application.
3. **glutCreateWindow ("name")** creates a window on the display. The string title can be used to label the window.
4. **glutInitWindowSize()** specifies the initial height and width of the window in pixels.
5. **glutMainLoop ()** causes the program to enter an event processing loop.
6. **glutPostRedisplay()** marks the current window as needing to be redisplayed.
7. **glutSwapBuffers**() swaps the buffers of the current window if double buffered.
8. **glutTimerFunc()** registers the timer callback function to be triggered in atleast milliseconds
9. **glutDisplayFunc()** Registers the display function that is executed when the window needs to be redrawn.
10. **glRasterPosition()** specifies a raster position.
11. **glMatrixMode()** specifies which matrix will be affected by subsequent transformations, Mode can be GL_MODEL_VIEW.
12. **glViewport()** specifies the affine transformation of x and y from normalized device coordinates to window coordinates.
13. **glVertex2f()** specifies x and y co-ordinates of the vertex.
14. **glColor3f()** sets the current color.
15. **glFlush()** forces any buffered OpenGL commands to execute.