CHAPTER 1

INTRODUCTION

MS-Paint program comes pre-installed on Microsoft's 'Windows' series of operating systems. It is the default graphics program for windows systems. It's mainly used for basic image editing purposes. This project aims to simulate the working of this MS-Paint program. For creating the GUI and implementing various functionalities, OpenGL graphics library has been used in C language. The project has been implemented in Microsoft Visual Studio Professional Edition 2008 and 2013 which uses C and/or C++ as the language tool.

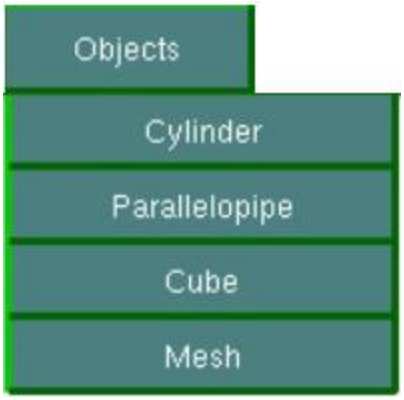
OpenGL provide various viewing function that helps us to develop various views of single object, and the way in which it appears on screen. Orthographic projection is the default view. OpenGL also provide various transformation functions with the help of these functions user can render its object at the desired location on the screen. In OpenGL we obtain viewing and modeling functionality through a small set of transformation functions. We can even rotate the object along desired locations and with the desired angle on the screen.

OpenGL provides a set of commands to render a three dimensional scene. OpenGL is a hardware- and system-independent interface. An OpenGL-application will work on every platform, as long as there is an installed GLUT library.

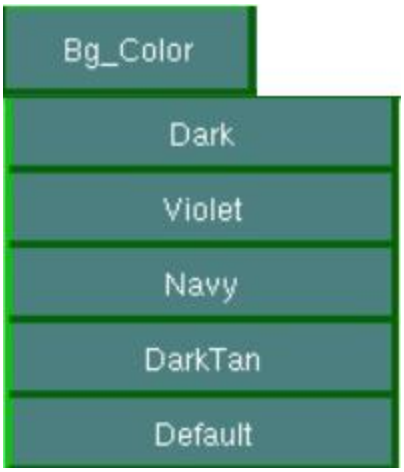
GLUT is a complete API written by Mark Kilgard which allows us to create windows and render the 2D or 3D scenes. It exists for several platforms, that means that a program which uses GLUT can be compiled on many platforms without (or at least with very few) changes in the code.

The various features implemented in this paint program are-

File: This option can be used to open any previously saved image or drawing by the user.



Object: This option can be used by the user to save any drawing.



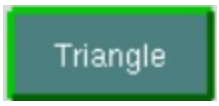
Background Color: This option changes the background screen color.

2D: This option can be used to draw 2D figures.

3D: This option can be used to draw the 3D figures.

Point: This tool can be used for drawing points.

Line: This tool can be used to draw any straight line.



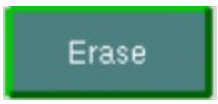
Triangle: This tool can be used to draw any type of outlined triangle.

Rectangle: This tool can be used to draw rectangle of any dimension.

Polygon: This tool can be used to draw polygon of any kind.

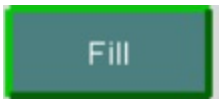
Circle: This tool can be used to draw a circle of specified radius.

Zoom-In & Zoom-Out: This tool can be used to zoom-in or zoom-out any drawing.

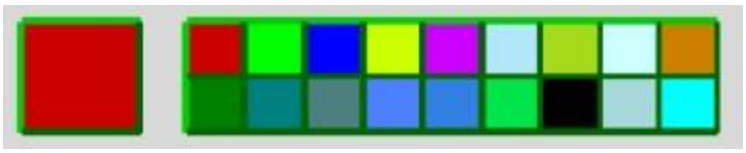


Eraser: This tool can be used to erase any drawing or a part of it.

Environment Property: This option changes the environment (theme color) of the window.

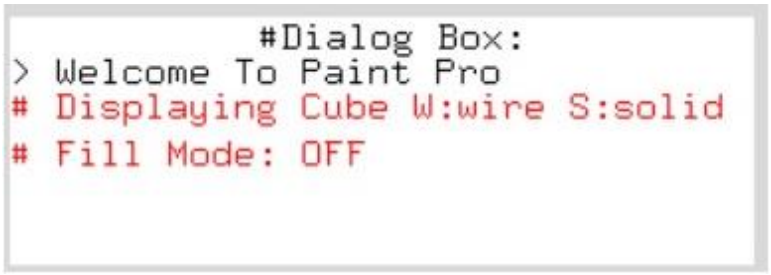


Fill: This tool can be used to fill any closed object with selected color.



Color Palette: This option can be used to select any particular color.

Toggle Fill:This mode enables to switch back and forth between Fill mode and Normal mode.If this mode is set, all closed figures drawn wiil be in solid color.



Dialog Box: This window shows various information related to drawing/selecting a tool. It also has options to enter filename for saving a file and to open it.

Help: This option pops up another window which contains various help topics.

Close: This option can be used to quit the application.

CHAPTER 2

REQUIREMENTS ANALYSIS

2.1 Software Requirements

Operating system like Windows XP, Windows 7 and Windows 8 is the platform required to develop 2D & 3D graphics applications.

A Visual C++ compiler is required for compiling the source code to make the executable file (.exe) which can then be directly executed.

A built-in graphics library like glut, glut32 & header file like glut.h and DLL i.e. dynamic link libraries like OpenGL32.dll, glut32.dll are required for creating the 3D layout.

2.2 Hardware requirements

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

Processor: Above x86

Processor speed: 500 MHz and above

RAM: 64 MB or above storage space 4GB and more

Monitor Resolution: A color monitor with a minimum resolution of 640*480

2.3 Platform

The package is implemented using Microsoft visual C++ under the windows programming environment. OpenGL and associated toolkits are used for the package development.

CHAPTER 3

DESIGN

3.1 Header files and their descriptions

3.1.1 Standard header files

```
#include <Windows.h>
#include <GL/glut.h>
#include <iostream>
#include <math.h>
#include <string.h>
```

- **Windows.h:** It's a windows-specific header file for the C/C++ programming language which contains the declarations for all of the functions in the windows API. It defines a very large no of window specific functions that can be used in C.

- **GL/glut.h:** The OpenGL utility toolkit (glut) handles most of the system dependent actions required to display a window, put OpenGL graphics in it, and accept mouse and keyboard input. It focuses on programming graphics.

- **iostream.h:** C++ input/output streams are primarily defined by iostream. A header file that is a part of the C++ standard library.

- **math.h:** It's a library in the standard library of C programming language designed for basic mathematical operations.

- **String.h:** It's a library in the standard library in C/C++ which defines several functions for manipulating C strings and arrays.

3.1.2 User defined header files

```
#include "Resource.h"
#include "Button.h"
#include "Camera.h"
#include "Menu.h"
```

- **Resource.h:** This header file holds various data structures (linked lists) to store the points of a geometric entity. It also contains additional functions as well as various global variables.

- **Button.h:** This header file contains two structures to store the button attributes and mouse clicks attributes. It also contains various additional functions to support all the button related functionalities.

- **Camera.h:** It contains functions which returns the type of camera user wants. The two kinds of available camera are 2D camera and 3D camera.

- **Menu.h:** It contains functions to handle all the menu related functionalities.

3.2 Description of OpenGL functions

Glut library functions used are:

main(): Execution of any program always starts with main function irrespective of where it is written in a program.

glutSwapBuffers(void): Performs a buffer swap on the *layer in use* for the *current window*. Specifically, glutSwapBuffers promotes the contents of the back buffer of the *layer in use* of the *current window* to become the contents of the front buffer. The contents of the back buffer then become undefined. The update typically takes place during the vertical retrace of the monitor, rather than immediately after glutSwapBuffers is called.

glutPostRedisplay(): marks the plane of current window as needing to be redisplayed. The next iteration through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display callback opportunity generates only a single redisplay callback.

glutInit(&argc,char ** argv): glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized.

glutInitDisplayMode(unsigned int mode): The *initial display mode* is used when creating top-level windows, sub windows, and overlays to determine the OpenGL display mode for the to-be-created window or overlay.

GLUT_RGB specifies the structure of each pixel.
GLUT_DEPTH is a buffer to hold the depth of each pixel.

glutCreateWindow(char *title): The glutCreateWindow creates a top-level window. The name will be provided to the window system as the window's name. The intent is that the window system will label the window with the name.

glutReshapeFunc():glutReshapeFunc sets the reshape callback for the *current window*. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established. The width and height parameters of the callback specify the new window size in pixels. Before the callback, the *current window* is set to the window that has been reshaped.

glutMainLoop():glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

glutBitmapCharacter(font,c): Without using any display lists, glutBitmapCharacter renders the character ‘c’ in the named bitmap font.

glPointSize(size): glPointSize specifies the rasterized diameter of points.

glFlush(): Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. glFlush empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

glRasterPos3f(x , y, z): This function is used to set the cursor at the x, y, z location.

glBegin(enum) and glEnd(): glBegin and glEnd delimit the vertices that define a primitive or a group of like primitives. glBegin accepts a single argument that specifies in which often ways the vertices are interpreted.

glColor3f(R,G,B): This function is used to pick a color of specified R,G,B value.

glClear(GLbitfield): glClear sets the bit plane area of the window to values previously selected by glClearColor, glClearIndex, glClearDepth, glClearStencil, and glClearAccum. Multiple color buffers can be cleared simultaneously by selecting more than one buffer at a time using glDrawBuffer.

glVertex2f(x,y): This function is used to draw a vertex at location x,y.

glEnable(GLenum): This function is used to enable the various functionalities like enabling the light, enabling the Z buffer.

glDisable(GLenum): This function is used to disable the various functionalities like enabling the light, enabling the Z buffer.

glRectf(x_0,y_0,x_1,y_1): This function is used to draw rectangle using the two diagonal points.

glClearColor(R,G,B,A): glClearColor specifies the red, green, blue, and alpha values used by glClear to clear the color buffers. Values specified by glClearColor are clamped to the range 0 to1 .

glMaterialfv(GLenum,GLenum,GLfloat *): glMaterial assigns values to material parameters. There are two matched sets of material parameters. One, the *front-facing* set, is used to shade points, lines, bitmaps, and all polygons (when two-sided lighting is disabled), or just front-facing polygons (when two-sided lighting is enabled).

glutSolidTeapot(size): glutSolidTeapot and glutWireTeapot render a solid or wireframe teapot respectively. Both surface normals and texture coordinates for the teapot are generated. The teapot is generated with OpenGL evaluators

glutSolidCube(size): glutSolidCube and glutWireCube render a solid or wireframe cube respectively. The cube is centered at the modeling coordinate's origin with sides of length size.

glutDisplayFunc(void (*func)()): glutDisplayFunc sets the display callback for the *current window*. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called. Before the callback, the *current window* is set to the window needing to be redisplayed and (if no overlay display callback is registered) the *layer in use* is set to the normal plane. The display callback is called with no parameters.

glutKeyboardFunc(void (*func)(unsigned char key, int x, int y)): glutKeyboardFunc sets the keyboard callback for the *current window*. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character

glutMouseFunc(void (*func)(int button, int state, int x, int y)): glutMouseFunc sets the mouse callback for the *current window*. When a user presses and releases mouse buttons in the window, each press and each release generates a mouse callback. The button parameter is one of GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, or GLUT_RIGHT_BUTTON.

glutMotionFunc(void (*func)(int x, int y)): glutMotionFunc set the motion callback for the *current window*. The motion callback for a window is called when the mouse moves within the window while one or more mouse buttons are pressed.

glutPassiveMotionFunc(void(*func)(int x, int y)): glutPassiveMotionFunc set the passive motion callback for the *current window*. The passive motion callback for a window is called when the mouse moves within the window while *no* mouse buttons are pressed.

glMatrixMode(GLenum) mode: Specifies which matrix stack is the target for subsequent matrix operations. Values accepted are: GL_MODELVIEW, GL_PROJECTION.The initial value is GL_MODELVIEW. Additionally, if the ARB tension extension is supported, GL_COLOR is also accepted.

gluPerspective(fovy, aspect, near, far): gluPerspective specifies a viewing frustum into the world coordinate system. In general, the aspect ratio in gluPerspective should match the aspect ratio of the associated viewport. For example, aspect = 2.0means the viewer's angle of view is twice as wide in *x* as it is in *y*.

glOrtho(left, right, top, bottom, Znear, Zfar): glOrtho describes a transformation that produces a parallel projection. The current matrix is multiplied by this matrix and the result replaces the current matrix.

3.3 Description of User-defined functions

Various user-defined functions used are:

Void Init(): This function is used to Initialize the basic color property of the window.

void display(): This function is the main display function which is called by the main function. Here all the objects are drawn for the continuous display.

Void keyboard(unsigned char key, int x, int y): This function is used to handle the Keyboard interaction.

Void resize(GLsizei w,GLsizei h): This function is called whenever the window is resized.

Void MouseButton(int button, int state, int x, int y): This function handles all the mouse interaction.

Void MouseMotion(int x, int y): This function is called whenever the position of the mouse cursor is changed and it gives the new mouse position.

Void MousePassiveMotion(int x, int y): This function is called at a high frequency and it continuously returns the last position of the mouse cursor.

int CreateButton(char *label, ButtonCallback cb, int x, int y, int w, int h, float r, float g, float b):This function is used to create a button at a specific location as specified in the function parameters.

Void ButtonDraw(): This function is used to draw the button on the screen.

int ButtonClickTest(Button *b, int x, int y): This function is used to check whether the particular button 'b' is clicked or not based on that it returns the value.

int ButtonRelease(int x, int y): This function is used to check that the pressed button is released or not based on that it will return the value.

Void ButtonPress(int x, int y): This function is used to check which button is pressed based on the value of x and y.

Void ButtonPassive(int x, int y): This function continuously check whether the cursor is above the button based on that the button is highlighted.

Void font(void *font, char *text, int x, int y): This function is used to print a message on the output screen. It is also used for displaying the label of button.

int DeleteById(int id): This function is used to delete a particular button based on the value of id.
Void ResetAll(): This function resets all the value stored in the data structure and the variables and initializes it to NULL.

Void Camera3D(int width, int height): This function is used to set up a 3D camera to render the scene.

Void Camera2D(int width , int Height): This function is used to set up a 2D camera to render the 2D scene.

Void MenuInit(): This function initializes the menu by putting all the needed menu entry in the linked list and the linked is further displayed.

Void DisplayMenu(int x , int y): This function is used to display the menu at the position x and y specified in the parameters.

float GetCurRed(): This function returns the current red value.

float GetCurGreen(): This function returns the current green value.

float GetCurBlue(): This function returns the current blue value.

Void SetMessage(const char *msg): This function is used to set the message to be displayed on the screen.

Char* GetMessage(): This function is used to get the current value of the message.

Void DrawMesh(): This function is used to draw mesh like structure on the output screen.

Void DrawCylinder(): This function is used to draw cylinder on to the output screen.

Void DrawParallelopipe(): This function is used to draw parallelopipe on to the output screen.

Viod Drawcube(): This function is used to draw cube at specified location on the output screen.

Void Hou2D(): This function draw a 2D house on the screen.

Void Hou3D(): This function draw a3D house on the screen.

Void gasket2D(): This function draws the 2d gasket on the output screen.

Void gasket3D(): This function draws the 3d gasket on the output screen..

Void OutLine(): This function draws the basic layout of the Application.

CHAPTER 4

IMPLEMENTATION

4.1 Setting up camera

```
/*Camera.h*/

#ifndef CAMERA_H
#define CAMERA_H

void Camera3D(int wwidth, int wheight);
void Camera2D(int wwidth, int wheight);

#endif

/*Camera.cpp*/

#include "Camera.h"

void Camera2D(int width,int height)
{
   //set up a 2d camera.
}

void Camera3D(int width,int height)
{
  //set up a 3d camera.
}
```

4.2 setting up Button

```
/*Button.h*/

#ifndef BUTTON_H
#define BUTTON_H

typedef void(*ButtonCallBack)();

/*Structure to hold button information*/
struct button
{
        int x;
```

```
        int y;
        int w;
        int h;
        int id;
        float r;
        float g;
        float bb;
        char *label;
        int state;
        int highlighted;
        ButtonCallBack callbackfunction;
        struct button *nxtbtn;
};
typedef struct button Button;

/*Structure to Hold Mouse Information*/
struct mouse
{
        int x;
        int y;
        int lmb;
        int mmb;
        int rmb;
        int xpress;
        int ypress;
};
typedef struct mouse Mouse;

void MouseButton(int button, int state, int x, int y);
void MouseMotion(int x, int y);
void MousePassiveMotion(int x, int y);
int CreateButton(char *label, ButtonCallBack cb, int x, int y, int w, int h, float r, float g, float
bb);
void ButtonDraw();
int ButtonClickTest(Button *b, int x, int y);
void ButtonRelease(int x, int y);
void ButtonPress(int x, int y);
void ButtonPassive(int x, int y);
void font(void *font, char *text, int x, int y);
int DeleteById(int id);

#endif


/*Button.cpp*/
```

```
#include "Button.h"

int CreateButton(char *label, ButtonCallBack cb, int x, int y, int w, int h, float r, float g, float bb)
{
  /*allocate memory for the button */
    Button *p=AllocateMemory();
 /*Inset attribute in the linked list of Buttons*/
      p->x = x;
         p->y = y;
         p->w = w;
         p->h = h;
         p->r = r;
         p->g = g;
         p->bb = bb;...
 /*update the linked list*/
      p->nxtbtn = pButtonList;
         pButtonList = p;
 /*return id of button*/
      return id;
}


int ButtonClickTest(Button *b, int x, int y)
{
  /* if button b is clicked.*/
     return 1;
  /* if button b is not clicked.*/
     return 0;
}

void ButtonPress(int x, int y)
{
   /*if x and y matches the x and y of button */
   /* button was pressed change the color and call the registered call bak function */
}

void ButtonPassive(int x, int y)
{
  /* check the x and y matches the x and y of button if true */
  highlight=1;
  Buttoncolor=newcolor;
}

void ButtonRelease(int x, int y)
{
  /* check the button is released or not if true */
```

```
    Buttoncolor = oldcolor;
}

void ButtonDraw()
{
  /* traverse the linklist a draw the button for each entry. */
}

void MouseButton(int button, int state, int x, int y)
{
   if (state == GLUT_DOWN)
        {
              /* update the button click */
              case GLUT_LEFT_BUTTON:
                    leftclick=true;
                    CheckForButtonClick();
              case GLUT_MIDDLE_BUTTON:
                    middleclick = true;
              case GLUT_RIGHT_BUTTON:
                    rightclick = true;
        }
        else
        {
         /* update the button click */
              case GLUT_LEFT_BUTTON:
                    leftclick=true;
                    CheckForButtonRelease();
              case GLUT_MIDDLE_BUTTON:
                    middleclick=true;
              case GLUT_RIGHT_BUTTON:
                    rightclick = true;
                    RemoveTheMenu();
        }
  /*redisplay the buttons */
}
```

4.3 Setting up menus.

```
/* Menu.h */

#ifndef MENU_H
#define MENU_H

void MenuInit();
void DisplayMenu(int x,int y);
```

```
#endif

/* Menu.cpp */

#include "Menu.h"

void MenuInit()
{
  /* Initialize all the menus */
    Menu *m=AllocateMemory();
      m=GetAttributes();
  /* stores the attribute in the list */
          update the list...

}

void DisplayMenu(int x,int y)
{
   /* Display the menu at position x and y */
}
```

4.4 Resource file

```
/* Resource.h */

#ifndef RESOURCE_H
#define RESOURCE_H


//structure to hold the line points.
struct LinePoints
{
        int x1, y1;
        int x2, y2;
        float red, green, blue;
        struct LinePoints *next;
};

//structure to hold the cube points.
struct CubePoints
{
        int x1, y1;
        int x2, y2;
        float red, green, blue;
        struct CubePoints *next;
```

```
};

//structure to hold the Parallelopipe points.
struct Parallelopipe
{
        int x1, y1;
        int x2, y2;
        float red, green, blue;
        struct Parallelopipe *next;
};

//structure to hold the Polygon points.
struct PolygonPoint
{
        int x1[10];
        int x2[10];
        int isfill;
        float red, green, blue;
        struct PolygonPoint *next;
};

//structure to hold the cylinder points.
struct CylinderPoint
{
        int x1, y1;
        int x2, y2;
        float red, green, blue;
        struct CylinderPoint *next;
};

//structure to hold the mesh points.
struct MeshPoint
{
        int x1, y1;
        int x2, y2;
        float red, green, blue;
        struct MeshPoint *next;
};

//structure to hold the points.
struct Points
{
        int x1, y1;
        float red, green, blue;
        struct Points *next;
};
```

```
//structure to hold the circle points.
struct CirclePoints
{
        int x1, y1;
        int x2, y2;
        int isfill;
        float red, green, blue;
        struct CirclePoints *next;
};

//structure to hold the triangle points.
struct TrianglePoints
{
        int x1, y1;
        int x2, y2;
        int x3, y3;
        int isfill;
        float red, green, blue;
        struct TrianglePoints *next;
};

//structure to hold the rectangle points.
struct RectPoints
{
        int x1, y1;
        int x2, y2;
        int isfill;
        float red, green, blue;
        struct RectPoints *next;
};

struct ErasePoint
{
        int x1, y1;
        struct ErasePoint *next;
};


void Outline();
char* getMessage();
void SetMessage(char *msg);
void Hou2D();
void Hou3D();
void drawcube(int a1, int b1, int a2, int b2);
void drawParallelopipe(int a1, int b1, int a2, int b2);
```

```
void drawCylinder(int a1, int b1, int a2, int b2);
void DrawMesh(int a1, int b1, int a2, int b2);

#endif

/* Resource.cpp */

#include "Resource.h"

void Outline()
{
  /*display the outline of the interface */
}

char* getMessage()
{
  return current_message;
}

void SetMessage(char *msg)
{
  current_message=msg;
}

void drawcube(int a1, int b1, int a2, int b2)
{
  /* Display The wire Cube */
}

void drawParallelopipe(int a1, int b1, int a2, int b2)
{
   /* dispaly the parallelopipe. */
}

void drawCylinder(int a1, int b1, int a2, int b2)
{
  /* display the cylinder */
}

void DrawMesh(int a1, int b1, int a2, int b2)
{
 /* display the mesh */
}


//main.cpp
```

4.5 include all the required header files.

```
#include <iostream>
#include <GL/glut.h>
#include <Windows.h>
#include <math.h>
#include <string>
#include "Resource.h"
#include "Button.h"
#include "Camera.h"
#include "Menu.h"
using namespace std;


void Init()
{
 /* initialize the window property. */
 /* enable the light, depth_test */
 /* Initialize the menu */
   MenuInit();
}


void display()
{
  /* clear the color buffer */

   if (welcome == 0 || welcome == 1)
   {
        /*display the welcome screen*/
        glFlush();
   }
   if(welcome == 2)
   {
        /* clear the screen */
        /* clear the color buffer */
        /* get a viewport of specified width and height */
        Camera2D(mwwidth,mwheight);
        display2D();
        Camera3D(mwwidth,mwheight);
        display3D();

        /* display the outline, buttons continously */
        Camera2D(mwwidth, mwheight);
        Outline();
```

```
            xizmo();
            ButtonDraw();
            glFlush();
            glutSwapBuffers();
        }
}

void keyboard(unsigned char key, int x, int y)
{
     switch(key)
     {
        /* Perform action based on the key Pressed */
            /* if save is true get the filename        */
            /* if open is true get the filename        */
     }
   /* redisplay based on changeg */
}

void resize(GLsizei w,GLsizei h)
{
            GetNewWidth();
            GetNewHeight();
            UpdateViewPort();
}

int main(int argc,char *argv[])
{
            /* initialize the glut library */
            glutInit(&argc, argv);
            /* set the dispaly mode */
            glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
            /* set the window position */
            glutInitWindowPosition(100, 100);
            /* set the windwo width and height */
            glutInitWindowSize(mwwidth, mwheight);
            /* create the windows */
            glutCreateWindow("Paint Pro");
            /* Initialize The basic window property */
            Init();
            /* regidter the display callback function */
            glutDisplayFunc(display);
            /* register the keyboard callback function */
            glutKeyboardFunc(keyboard);
            glutFullScreen();
            /*register the reshape callback function */
            glutReshapeFunc(resize);
```

```
        /*register the mouse callback function */
        glutMouseFunc(MouseButton);
        /*register the motion callback function */
        glutMotionFunc(MouseMotion);
        /*register the Passivemotion callback function */
        glutPassiveMotionFunc(MousePassiveMotion);
        /* loop the program untill encounters exit */
        glutMainLoop();
        return 0;
}
```

CHAPTER 5

RESULTS AND SNAPSHOTS

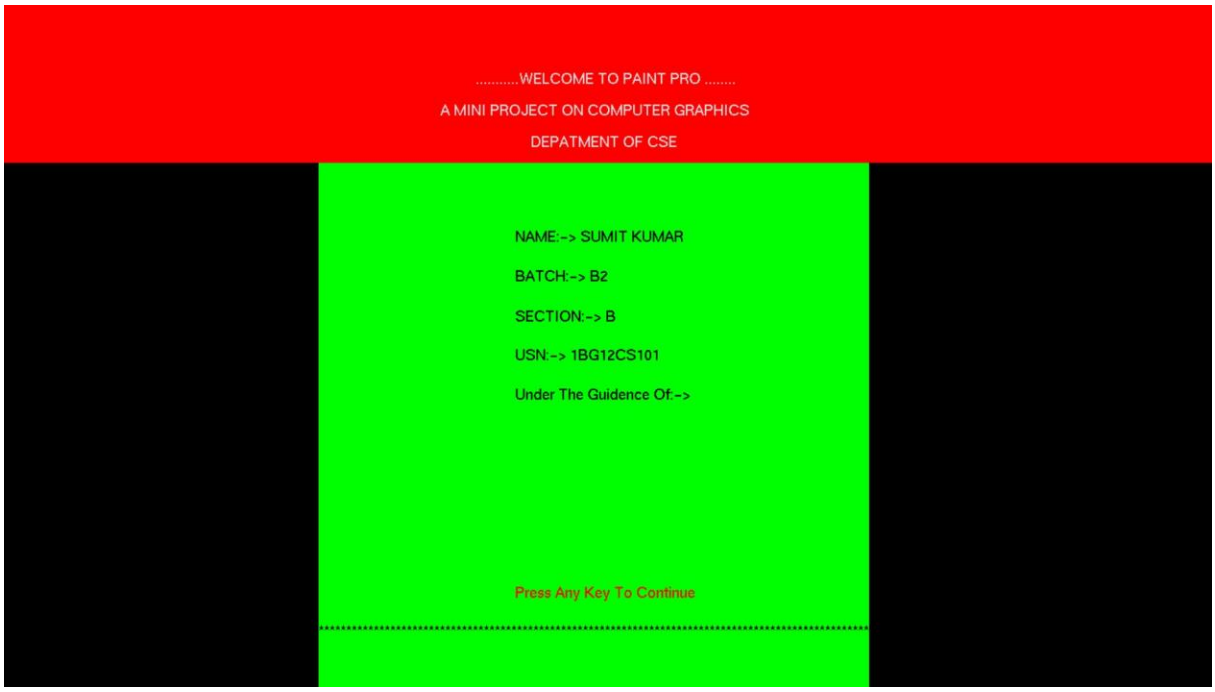


Fig 1: Welcome Screen

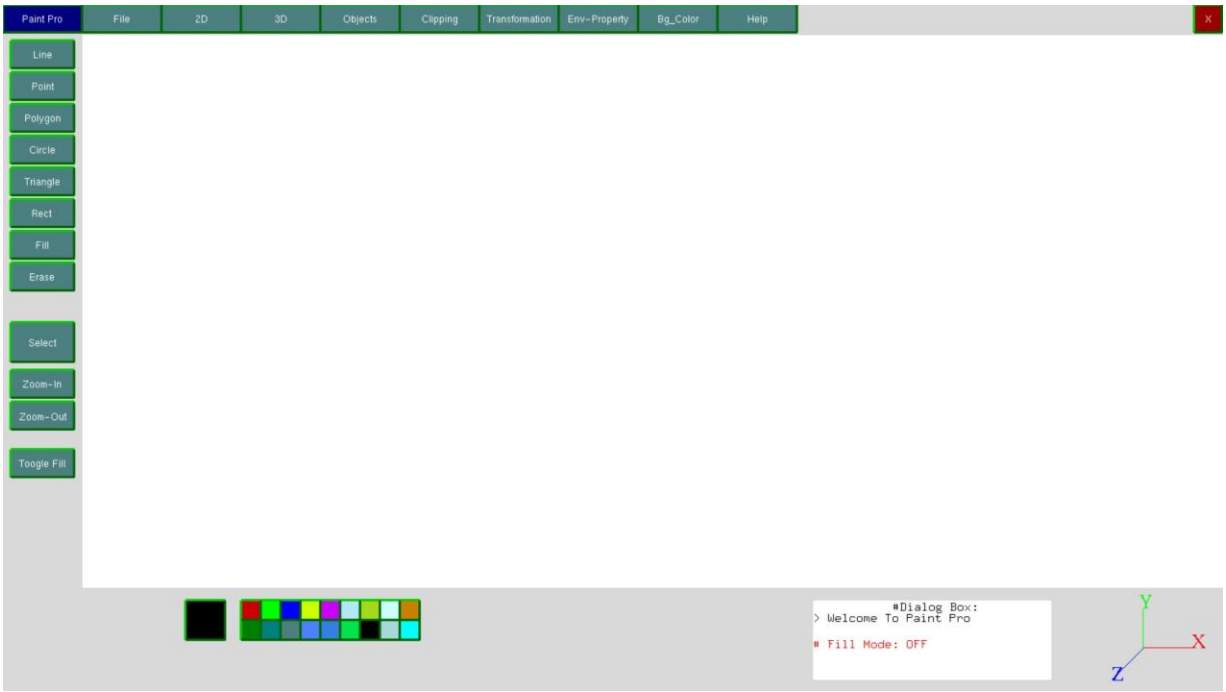


Fig 2: Default Layout of application

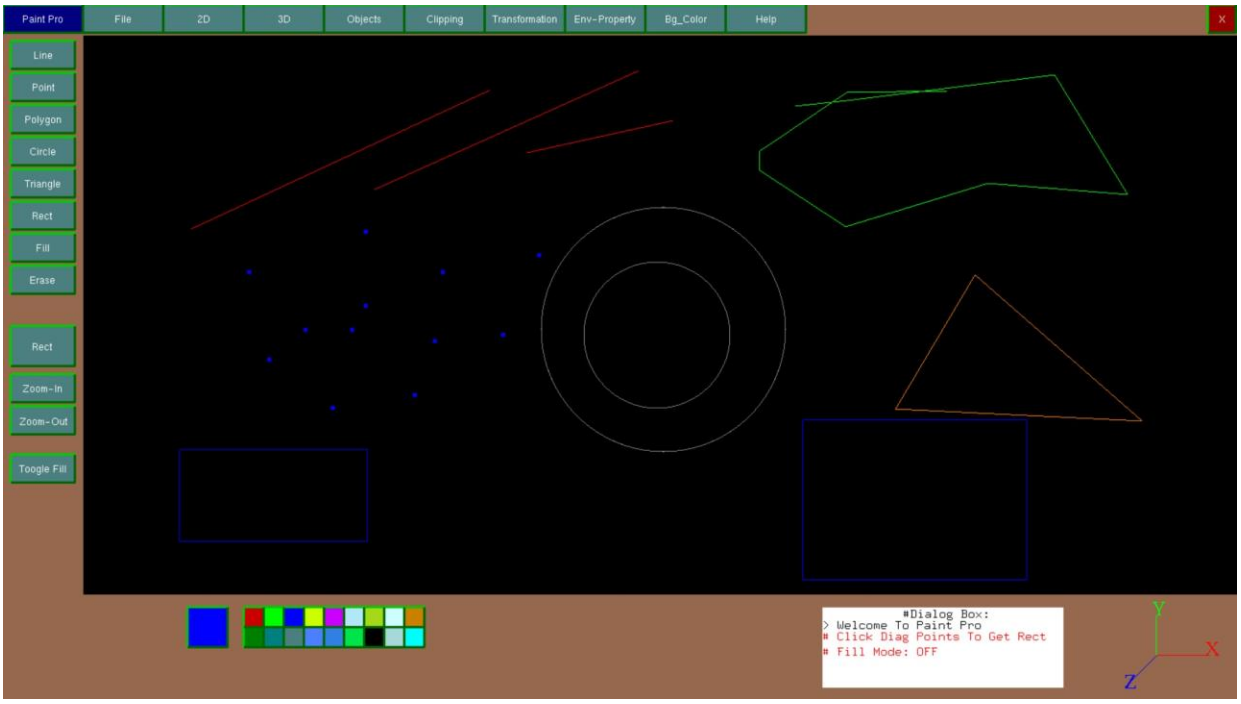


Fig 3: Application with brown theme and black background

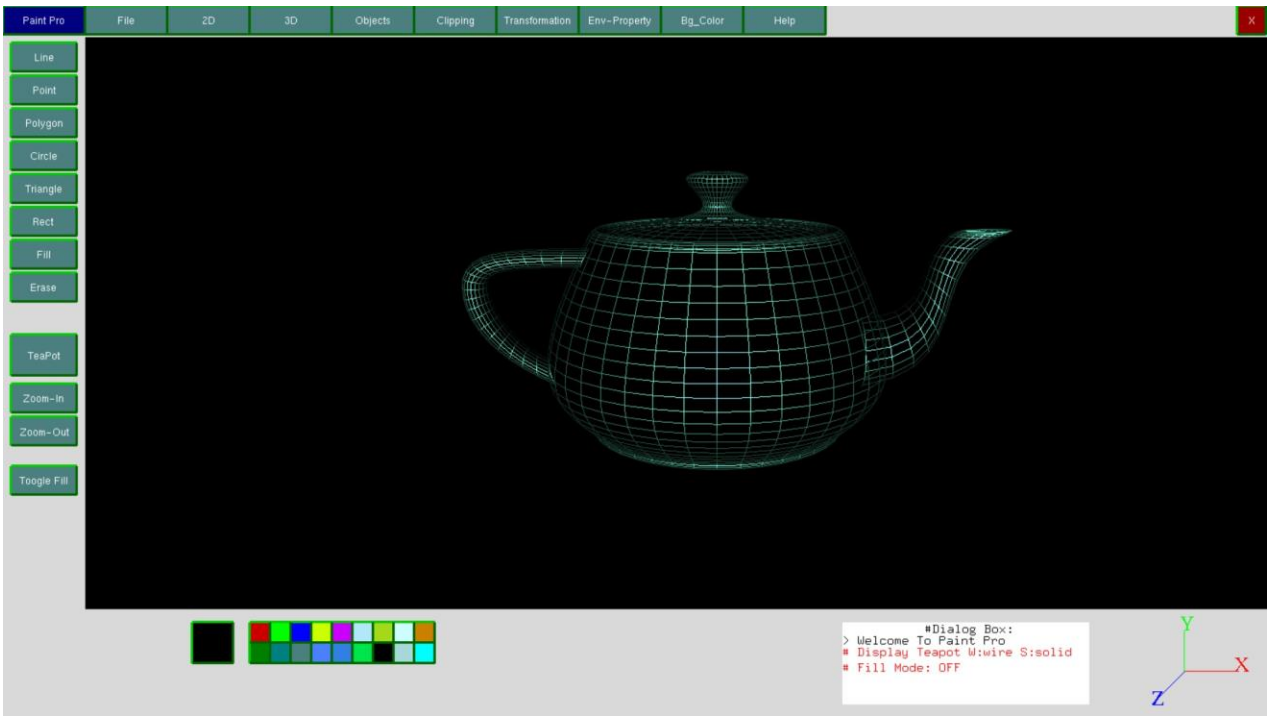


Fig 4: Wireframe teapot

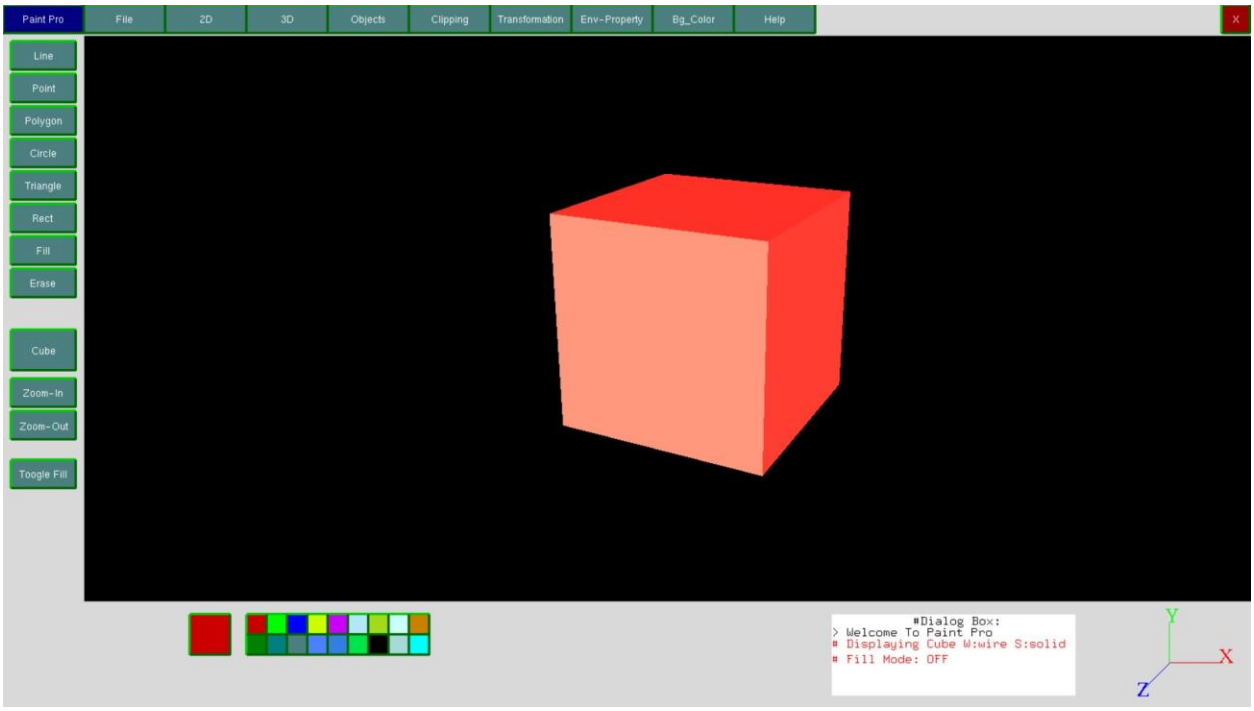


Fig 5: Application showing menu on right click

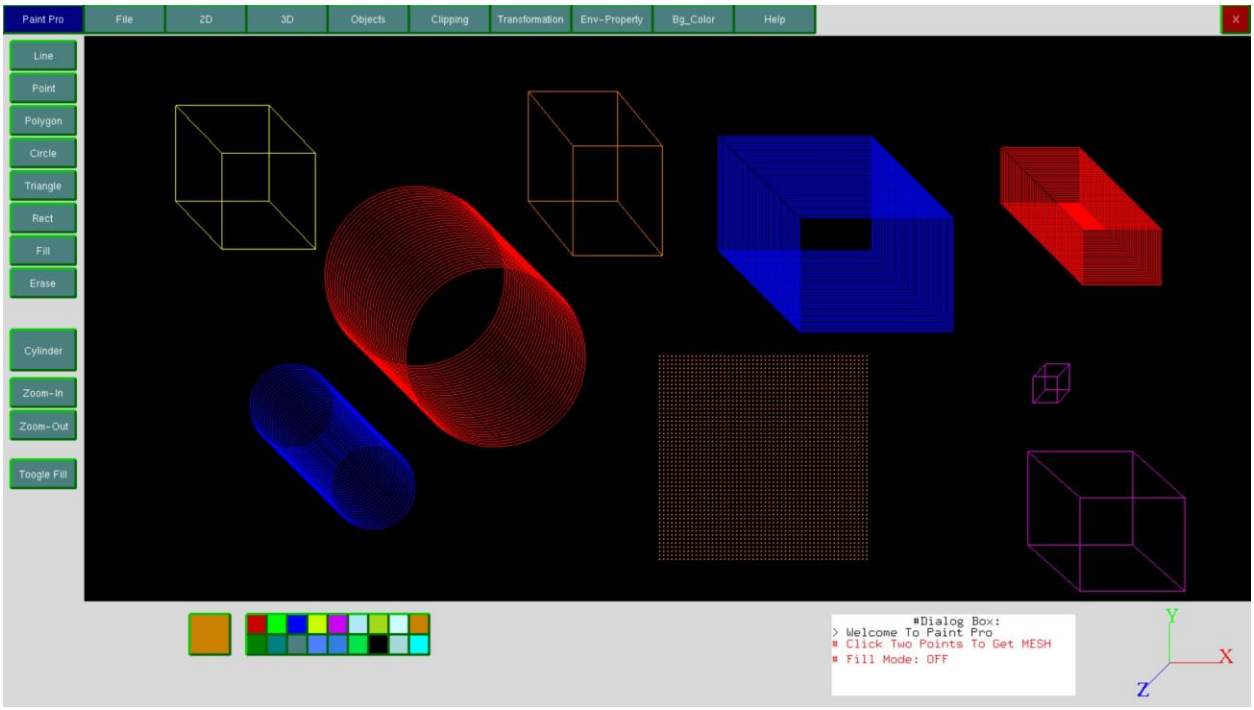


Fig 6: Various objects drawn

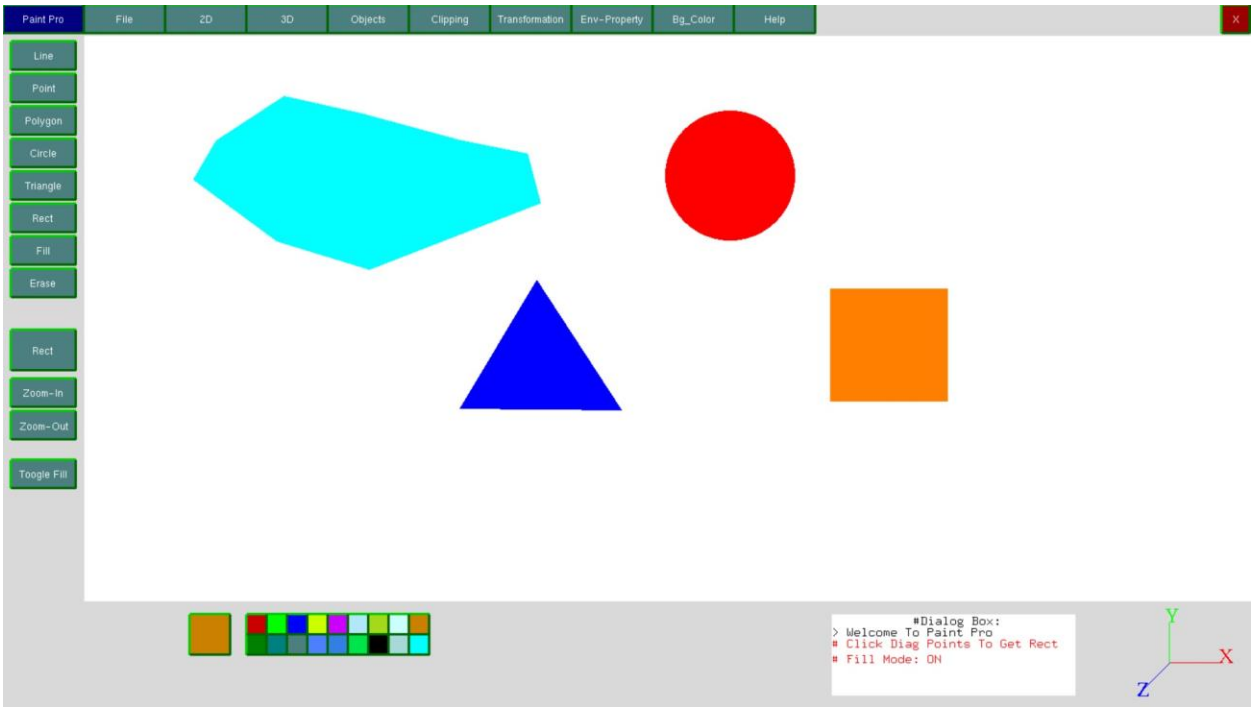


Fig 7: Various free hand in Fill Mode

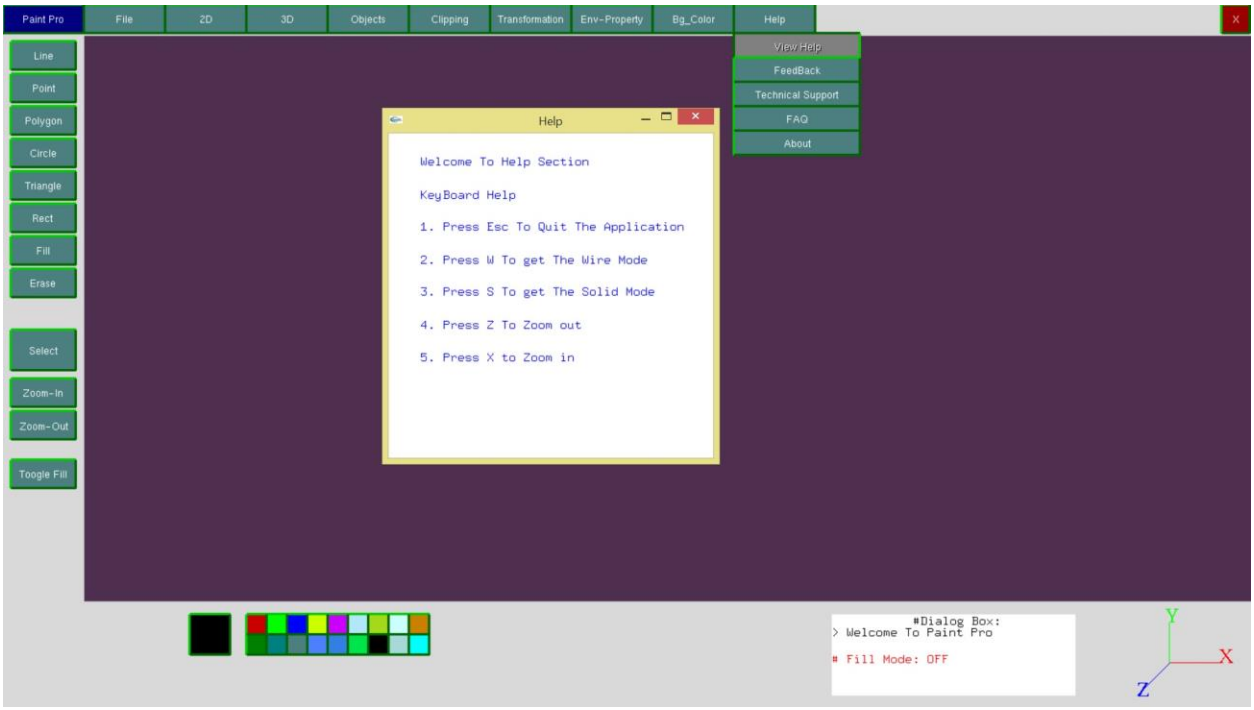


Fig 8: Application showing the Help window

CHAPTER 6

FUTURE ENHANCEMENTS

In future versions this project, addition of some more in-built 2D and 3D models of geometric figures like cone, and pyramids, option to change line thickness, pick up color of line and various options to set light properties according to users' requirements is a feasible idea

Making the User-interface of this program simpler and more user friendly will certainly help beginners in using this program more easily. The added functionality of giving information about an object on mouse hover over the object as well as 'Undo' and 'Redo' options will be also be a good choice.

Perhaps the most challenging task will be to implement the functionality of 'Layers' which are used extensively by Professionals in graphics industry.

CHAPTER 7

CONCLUSION

This project is well suited for designing 2D and 3D objects, as well as for carrying out basic graphics functionalities like drawing a simple line, creating a cube, circle, square, erasing and filling them. However, if implemented on large scale with sufficient resources, it has the potential to become a standard stand-alone GUI based application for Windows Operating System.

Out of the many features available, the project demonstrates some popular and commonly used features of OpenGL such as Rendering, Transformation, Rotation, Lighting, Scaling etc. These graphic functions will also work in tandem with various rules involved in the game. Since this project works on dynamic values, it can be used for real time computation.

The project enabled to work with mid-level. OpenGL complexity and through this project we realize the scope of OpenGL platform has a premier game developing launch pad. Hence, it has indeed been useful in developing many games. OpenGL in its own right is good for low cost and simple game development. It serves as an important stepping stone for venturing into other fields of Computer Graphics design and applications.

BIBLIOGRAPHY

1. Interactive Computer Graphics (a top down approach) by Edward Angel

2. OpenGL Programming Guide by Addison-Wesley

Online Resources

1. www.opengl.org
2. www.cs.unm.edu/~angel
3. www.aw.com/cssupport