CONTENTS

SL.NO	particulars	PAGE.NO
1	Abstract	2
2	System Specifications	3
3	Introduction to openGL	4
5	Implementation	7
6	Interaction	9
7	Source Code	10
8	Output	27
9	Conclusion	29
10	Bibliography	30

Abstract

- Main aim of this Mini Project is to illustrate the concepts and usage of pre-built functions in OpenGL.
- Creating objects and games like StuntPlane using Opengl library.
- Functions like glutSolidSphere and glutSolidCube are used to create the plane.
- The rings were created using torus function provided in glut library.
- We have used input devices like mouse and key board to interact with program

System specifications

➢ **SOFTWARE REQUIREMENTS :**

- MICROSOFT VISUAL C++
- OPENGL

➢ **HARDWARE REQUIREMENT :**

- GRAPHICS SYSTEM,
- Pentium P4 with 256 of Ram(Min)

Introduction to openGL

As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer.

These objects are described as sequences of vertices or pixels.

OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

OpenGL Fundamentals

This section explains some of the concepts inherent in OpenGL.

Primitives and Commands

OpenGL draws primitives—points, line segments, or polygons—subject to several selectable modes.

You can control modes independently of each other; that is, setting one mode doesn't affect whether other modes are set .Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls.

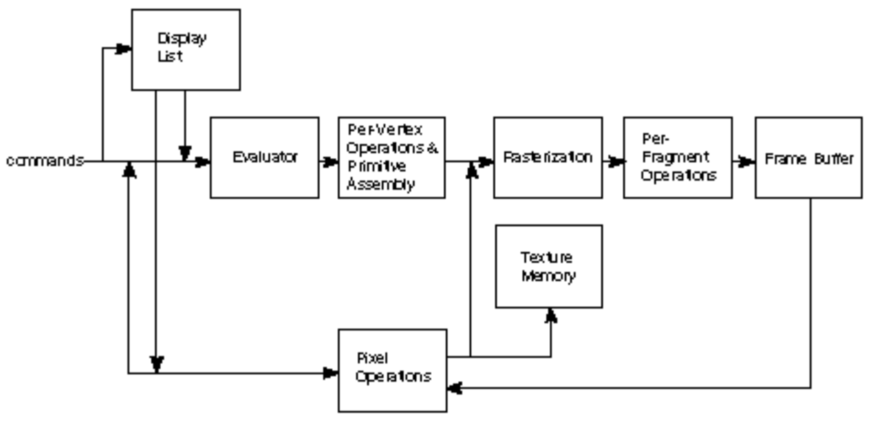
Primitives are defined by a group of one or more vertices. A vertex defines a point, an endpoint of a line, or a corner of a polygon where two edges meet. Data is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way. The type of clipping depends on which primitive the group of vertices represents.

Commands are always processed in the order in which they are received, although there may be an indeterminate delay before a command takes effect. This means that each primitive is drawn completely before any subsequent command takes effect. It also means that state-querying commands return data that's consistent with complete execution of all previously issued OpenGL commands.

Basic OpenGL Operation

The figure shown below gives an abstract, high-level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.

Figure . OpenGL Block Diagram



As shown by the first block in the diagram, rather than having all commands proceed immediately through the pipeline, you can choose to accumulate some of them in a display list for processing at a later time.

Rasterization produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon.

Each fragment so produced is fed into the last stage,

per-fragment operations, which performs the final operations on the data before it's stored as pixels in the frame buffer. These operations include conditional updates to the frame buffer based on incoming and previously stored z-value s (for z-buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values.

All elements of OpenGL state, including the contents of the texture memory and even of the frame buffer, can be obtained by an OpenGL application.

Implementation

This program is implemented using various openGL functions which are

shown below.

Various functions used in this program.

- glutInit() : interaction between the windowing system and OPENGL is initiated

- glutInitDisplayMode() : used when double buffering is required and depth information is required

- glutCreateWindow() : this opens the OPENGL window and displays the title at top of the window
- glutInitWindowSize() : specifies the size of the window
- glutInitWindowPosition() : specifies the position of the window in screen co-ordinates
- glutKeyboardFunc() : handles normal ascii symbols
- glutSpecialFunc() : handles special keyboard keys
- glutReshapeFunc() : sets up the callback function for reshaping the window
- glutIdleFunc() : this handles the processing of the background
- glutDisplayFunc() : this handles redrawing of the window
- glutMainLoop() : this starts the main loop, it never returns
- glViewport() : used to set up the viewport
- glVertex3fv() : used to set up the points or vertices in three dimensions
- glColor3fv() : used to render color to faces
- glFlush() : used to flush the pipeline

- glutPostRedisplay() : used to trigger an automatic redrawal of the object
- glMatrixMode() : used to set up the required mode of the matrix
- glLoadIdentity() : used to load or initialize to the identity matrix
- glTranslatef() : used to translate or move the rotation centre from one point to another in three dimensions
- glRotatef() : used to rotate an object through a specified rotation angle

Interaction with program

- This program includes interaction through keyboard.

- S → Start the Project
- Use keys A,D,W,S to control the moment of PacMan.
- Q-> Quit

Source Code

```
#include <windows.h>

#include<string.h>

#include<stdarg.h>

#include<stdio.h>

#include <glut.h>
```

```
static double x1=0.0;

static double x2=0.0;  // Rotates Blades

static double a1=0.0;

static double r1=0.0;     //tilt plane

static double r2=0.0;

static double r3=0.0;     //tilt plane

static double r4=0.0;


static double move=0.0;     // moves planes

static double move_y=0.0;     // moves planes


static double z1=0.0;   // Moves Cirlces


void

stroke_output(GLfloat x, GLfloat y, char *format,...)

{

        va_list args;

        char buffer[200], *p;

        va_start(args, format);

        vsprintf(buffer, format, args);

        va_end(args);

        glPushMatrix();

        glTranslatef(-2.5, y, 0);
        glScaled(0.003, 0.005, 0.005);
        for (p = buffer; *p; p++)
   glutStrokeCharacter(GLUT_STROKE_ROMAN, *p);
        glPopMatrix();
}

void Circle(){


glColor3f(1.0,1.0,1.0);
glutSolidTorus(0.4,3.5,50,50);


}




void drawPlane(){




/***********  PLANE CONSTRUCTION  *******************/
        glPushMatrix();




        // Main Body
glPushMatrix();
```

```
glScalef(.3,0.3,1.5);
glColor3f(1,1.0,0.5);
glutSolidSphere(2.0,50,50);
glPopMatrix();

 glPushMatrix();
 glTranslatef(0.0,0.1,-1.8);
glScalef(1.0,1,1.5);
glColor3f(0,0,1);
glutSolidSphere(0.5,25,25);
glPopMatrix();


//Left Fin

glPushMatrix();
glTranslatef(-1.0,0,0);
glScalef(1.5,0.1,0.5);
glColor3f(0,0,0);
glutSolidSphere(1.0,50,50);
glPopMatrix();

// Right Fin
```

```
glPushMatrix();


glTranslatef(1.0,0,0);
glScalef(1.5,0.1,0.5);
glColor3f(0,0,0);
glutSolidSphere(1.0,50,50);
glPopMatrix();


//right Tail fin
glPushMatrix();


glTranslatef(0.8,0,2.4);
glScalef(1.2,0.1,0.5);
glColor3f(0.0,0,0);
glutSolidSphere(0.4,50,50);
glPopMatrix();


//left Tail fin
glPushMatrix();
glTranslatef(-0.8,0,2.4);
glScalef(1.2,0.1,0.5);
glColor3f(0.0,0,0);
glutSolidSphere(0.4,50,50);
```

```
glPopMatrix();

//Top tail fin
glPushMatrix();
glTranslatef(0,0.5,2.4);
glScalef(0.1,1.1,0.5);
glColor3f(0.0,0,0);
glutSolidSphere(0.4,50,50);
glPopMatrix();

// Blades
glPushMatrix();
glRotatef(x2,0.0,0.0,1.0);
glPushMatrix();
glTranslatef(0,0.0,-3.0);
glScalef(1.5,0.2,0.1);
glColor3f(0.0,0,0);
glutSolidSphere(0.3,50,50);
glPopMatrix();

//blades
glPushMatrix();
glRotatef(90,0.0,0.0,1.0);
```

```
glTranslatef(0,0.0,-3.0);
glScalef(1.5,0.2,0.1);
glColor3f(0.0,0,0);
glutSolidSphere(0.3,50,50);
glPopMatrix();

glPopMatrix();
/* Blased End */

/*   Wheels   */
//Front



glPushMatrix();
glTranslatef(0.0,-0.8,-1.5);
glRotatef(90,0.0,1,0);
glScaled(0.3,0.3,0.3);
glutSolidTorus(0.18,0.5,25,25);
glColor3f(1,1,1);
glutSolidTorus(0.2,0.1,25,25);

glPopMatrix();
```

```
glPushMatrix();
glTranslatef(0.0,-0.4,-1.5);
glRotatef(20,0.0,1,0);
glScaled(0.05,0.3,0.05);
glutSolidSphere(1.0,25,25);
glPopMatrix();



//Rear


glPushMatrix();
glTranslatef(0.3,-0.8,0.7);
glRotatef(90,0.0,1,0);
glScaled(0.3,0.3,0.3);
glColor3f(0,0,0);
glutSolidTorus(0.18,0.5,25,25);
glColor3f(1,1,1);
glutSolidTorus(0.2,0.1,25,25);
glPopMatrix();

glPushMatrix();
glTranslatef(0.3,-0.4,0.7);
glRotatef(20,0.0,1,0);
```

```
glScaled(0.05,0.3,0.05);
glutSolidSphere(1.0,25,25);
glPopMatrix();

//rear 2
glPushMatrix();
glTranslatef(-0.3,-0.8,0.7);
glRotatef(90,0.0,1,0);
glScaled(0.3,0.3,0.3);
glColor3f(0,0,0);
glutSolidTorus(0.18,0.5,25,25);
glColor3f(1,1,1);
glutSolidTorus(0.2,0.1,25,25);
glPopMatrix();

glPushMatrix();
glTranslatef(-0.3,-0.4,0.7);
glRotatef(20,0.0,1,0);
glScaled(0.05,0.3,0.05);
glutSolidSphere(1.0,25,25);
glPopMatrix();
glPopMatrix();




}


void plane(){


        glClearColor(0.3,0.3,0.3,0.0);

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glLoadIdentity();

  glTranslatef(0.0f,0.0f,-13.0f);


        glPushMatrix();

        glRotatef(x1,0.0,1.0,0.0);

        drawPlane();

        glPopMatrix();


glFlush();

glutSwapBuffers();


}
```

```
void gameOver(){
        glClearColor(1.0,0.0,0.0,0.0);


                glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();
   glTranslatef(0.0f,0.0f,-13.0f);
        stroke_output(-2,2,"you crashed my plane");









glBegin(GL_POLYGON);
        glColor3f(0.0,1.0,0.0);
glVertex3f(-100,-5,100);
glVertex3f(100,-5,100);
glVertex3f(100,-5,-100);
glVertex3f(-100,-5,-100);
glEnd();
```

```
        glFlush();


}


void fighterPlane()

{




        glClearColor(0.3,0.3,0.3,0.0);

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   glLoadIdentity();

   glTranslatef(0.0f,0.0f,-13.0f);

if(move_y<=-4)

{

        gameOver();


}

else{


//Floor


glBegin(GL_POLYGON);

        glColor3f(0.0,1.0,0.0);
```

```
glVertex3f(-100,-5,100);

glVertex3f(100,-5,100);

glVertex3f(100,-5,-100);

glVertex3f(-100,-5,-100);

glEnd();



// draw the obstacles

glPushMatrix();

glTranslatef(0,0,-20+z1);

Circle();

glPopMatrix();



glPushMatrix();

glTranslatef(-3,0,-60+z1);

Circle();

glPopMatrix();


glPushMatrix();

glTranslatef(3,0,-100+z1);

Circle();

glPopMatrix();
```

```
glPushMatrix();

glTranslatef(0,0,-160+z1);

Circle();

glPopMatrix();



glPushMatrix();

glTranslatef(3,0,-320+z1);

Circle();

glPopMatrix();



glPushMatrix();

glTranslatef(-3,0,-380+z1);

Circle();

glPopMatrix();
```

```
// Call drawPlane

glPushMatrix();
        glTranslatef(move,-1.0+move_y,0);
        glRotatef(r1,0.0,0.0,1.0);
        glRotatef(r2,1.0,0.0,0.0);
        drawPlane();
        glPopMatrix();

}
        glFlush();
  glutSwapBuffers();
}






void s()
{x1+=0.3;
```

```
fighterPlane();
}

void start(){
x1+=0.3;
x2+=5.0;
z1+=0.5;
fighterPlane();
}

void p1(){

x2+=10.0;
plane();
}



void doInit()
{

        /* Background and foreground color */
  glClearColor(0.0,0.0,0.0,0.0);
  glColor3f(.0,1.0,1.0);
```

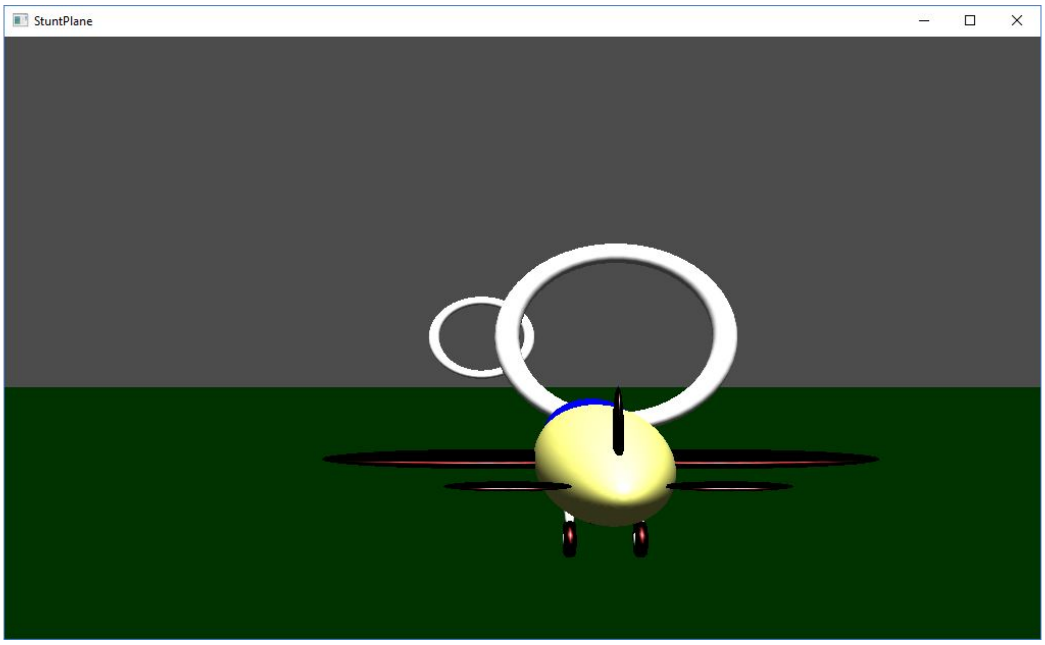
```
   glViewport(0,0,640,480);

         /* Select the projection matrix and reset it then
    setup our view perspective */
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(30.0f,(GLfloat)640/(GLfloat)480,0.1f,200.0f);
   /* Select the modelview matrix, which we alter with rotatef() */
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glClearDepth(2.0f);
         glEnable(GL_COLOR_MATERIAL);
   glEnable(GL_DEPTH_TEST);

   glDepthFunc(GL_LEQUAL);
}

void doDisplay()
{
         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
         glLoadIdentity();
         glTranslatef(0.0f,0.0f,-13.0f);
glPushMatrix();
```

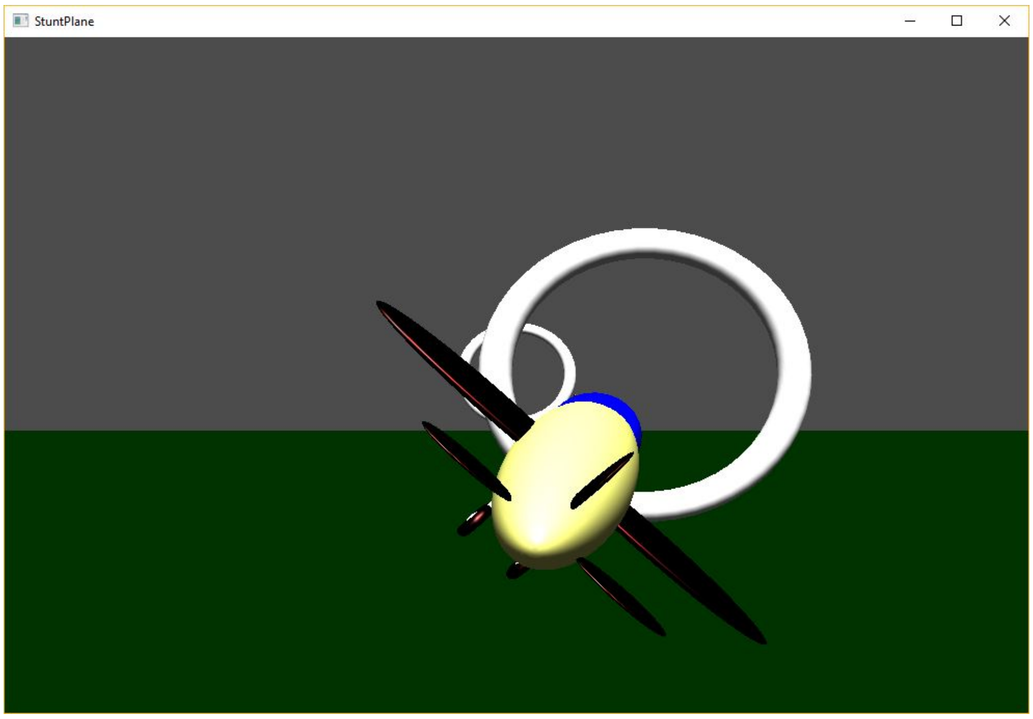
```
        glScaled(0.7,0.7,0.7);
        stroke_output(-2.0, 1.7, "Wel Come");
        stroke_output(-2.0, 0.9, "To");
        stroke_output(-2.0, 0.0, "Project Created");
        stroke_output(-2.0, -0.9, "By");
        stroke_output(-2.0, -1.8, "Candidate Name");
glPopMatrix();
        GLfloat mat_ambient[]={0.0f,1.0f,2.0f,1.0f};
        GLfloat mat_diffuse[]={0.0f,1.5f,.5f,1.0f};
        GLfloat mat_specular[]={5.0f,1.0f,1.0f,1.0f};
        GLfloat mat_shininess[]={50.0f};
        glMaterialfv(GL_FRONT,GL_AMBIENT,mat_ambient);
        glMaterialfv(GL_FRONT,GL_DIFFUSE,mat_diffuse);
        glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular);
        glMaterialfv(GL_FRONT,GL_SHININESS,mat_shininess);

        GLfloat lightIntensity[]={1.7f,1.7f,1.7f,1.0f};
        GLfloat light_position3[]={0.0f,5.0f,5.0f,0.0f};
        glLightfv(GL_LIGHT0,GL_POSITION,light_position3);
        glLightfv(GL_LIGHT0,GL_DIFFUSE,lightIntensity);



        glFlush();
        glutSwapBuffers();


}




void mykey(unsigned char key,int x,int y)

{

        if(key=='s')

        {

                glutIdleFunc(start);

        }



        if(key=='S')

        {

                glutIdleFunc(s);

        }




        if(key=='a'|| key=='A')

        {
```

```
        r1=45;

        r2=10;

        move-=0.1;

}



if(key=='d'|| key=='D')

{

        r1=-45;

        r2=10;

        move+=0.1;

}


if(key=='w'|| key=='W')

{

        r1=0;

        r2=0;


        move_y-=0.4;

}



if(key=='p'|| key=='P')
```

```
        {
                x1+=1.3;
                glutIdleFunc(p1);


        }


        if(key=='o'|| key=='O')
        {
                x1-=1.3;
                glutIdleFunc(p1);


        }





        if(key=='q'|| key=='Q'){
        exit(0);


        }


}


static void specialKey(int key,int x,int y){
        if(key==GLUT_KEY_UP){

        r1=0;

        r2=-10;

         //lower the nose of plane

                move_y-=0.5;


        }



if(key==GLUT_KEY_DOWN){

        move_y+=0.5;

        r1=0;

        r2=10;

         // raise the nose of plane

        }
if(key==GLUT_KEY_LEFT){

        r1=45;

                r2=10;

                move-=0.1;


        }
if(key==GLUT_KEY_RIGHT){
        r1=-45;
                r2=10;
                move+=0.1;
        }




}



int main(int argc, char *argv[])
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
   glutInitWindowSize(640,480);
   glutInitWindowPosition(30,0);
   glutCreateWindow("StuntPlane");
   glutDisplayFunc(doDisplay);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glShadeModel(GL_SMOOTH);
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_NORMALIZE);
        glutKeyboardFunc(mykey);
        glutSpecialFunc(specialKey);
        doInit();
  glutMainLoop();
        return 0;
}
```

OUTPUT OF THE PROGRAM





Conclusions

The project “Stunt Plane” clearly demonstrates the usage of inbuilt functions of OpenGL library.

Finally we conclude that this program clearly illustrate the concepts of computer graphics using openGL and has been completed successfully and is ready to be demonstrated.

Bibliography

WE HAVE OBTAINED INFORMATION FROM MANY RESOURCES TO DESIGN AND IMPLEMENT OUR PROJECT SUCCESSIVELY. WE HAVE ACQUIRED MOST OF THE KNOWLEDGE FROM RELATED WEBSITES. THE FOLLOWING ARE SOME OF THE RESOURCES :

- TEXT BOOKS :
  INTERACTIVE COMPUTER GRAPHICS A TOP-DOWN APPROACH
  -By Edward Angel.

- COMPUTER GRAPHICS,PRINCIPLES & PRACTICES
  - Foley van dam
  - Feiner hughes

- WEB REFERENCES:
  http://jerome.jouvie.free.fr/OpenGl/Lessons/Lesson3.php
  http://google.com
  http://opengl.org