International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 07 | July 2020 www.irjet.net p-ISSN: 2395-0072

A PROJECT ON 3D CAR SIMULATION

Punyaslok Sarkar[1], Mr. Shivaraj V B[2],

[1]4th (Final)Year student, Computer Science & Engineering, CMR Institute of Technology, Karnataka, India
[2]Asst. Professor, Computer Science & Engineering, CMR Institute of Technology, Karnataka, India

---

Abstract - *The main aim of the 3D Car Simulation Computer Graphics Mini Project is to illustrate the concepts and usage of pre-built functions in OpenGL. Simulation of a 3d Car Simulation is being done using computer graphics. 3D Car Simulation is a project where we can also change the weather & color of the car and other graphics.*

Key Words: car, graphics, simulation, openGL, color

1. INTRODUCTION

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.

1.1 OpenGL Fundamentals

This section explains some of the concepts inherent in OpenGL.

1.1.1 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.

1.1.2 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.

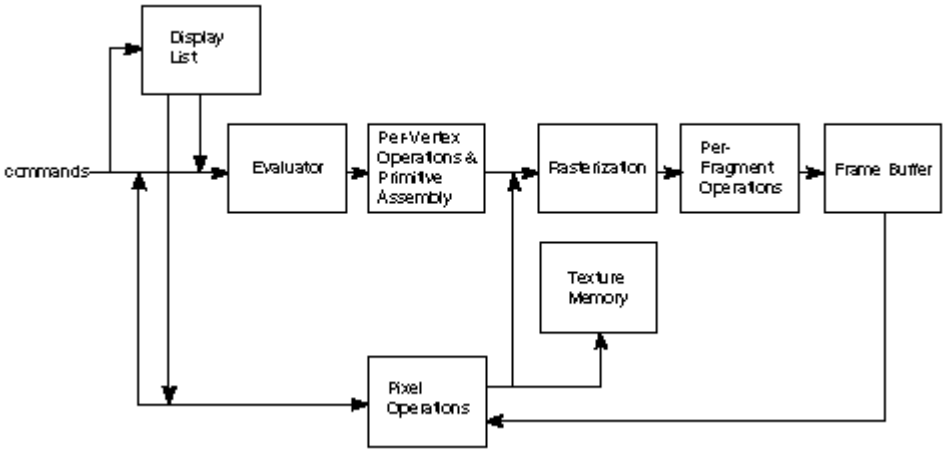

Fig.1-OpenGL Block Diagram

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.

2. System Requirement

2.1 Code::Blocks :

Code::Blocks is a free, open-source cross platform IDE that supports multiple compilers including GCC, Clang and Visual C++. It is developed in C++ using WxWidgets as the GUI toolkit. Using a plug-in architecture, its capabilities and features are defined by the provided plug-in. Currently, Code::Blocks is oriented towards C, C++, and Fortran. It has a custom build system and optional Make support.

2.2 OpenGL: Open Graphics Library (OpenGL) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering.

3. Design

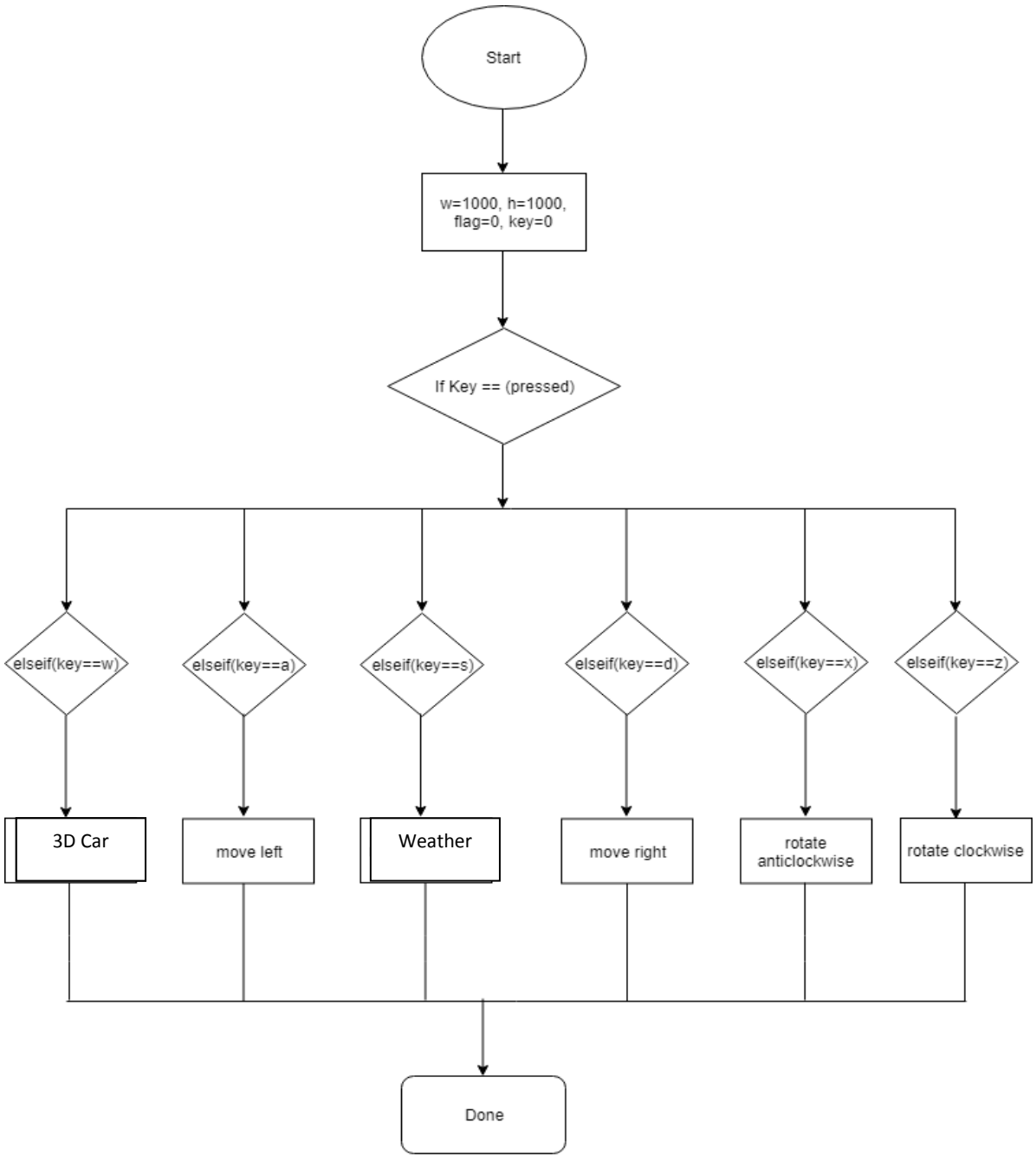

Fig.2- Flowchart of Application Program in abstract

4. Implementation

This program is implemented using various OpenGL functions which are shown below.

4.1 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
- glPushMatrix() : set current matrix on the stack
- glPopMatrix() :pop the old matrix without the transformations.
- glutKeyboardFunc() : handles normal ascii symbols
- glutSpecialFunc() : handles special keyboard keys
- glLightfv() :gives light in an area
- 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.

4.2 Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>
#include <math.h>
#include <string.h>
```

```
/* ASCII code for the escape key. */
#define ESCAPE 27

GLint window;
GLint window2;
GLint Xsize=1000;
GLint Ysize=800;
float i,theta;
GLint nml=0,day=1;

char name3[]="PROJECT:  3D CAR  ANIMATION";

GLfloat xt=0.0,yt=0.0,zt=0.0,xw=0.0;   /* x,y,z translation */
GLfloat tx=295,ty=62;
GLfloat xs=1.0,ys=1.0,zs=1.0;

GLfloat xangle=0.0,yangle=0.0,zangle=0.0,angle=0.0;   /* axis angles */

GLfloat r=0,g=0,b=1;
GLint light=1;
int count=1,flg=1;
int view=0;
int flag1=0,aflag=1;          //to switch car driving mode
int flag2=0,wheelflag=0;   //to switch fog effect
GLUquadricObj *t;

static void SpecialKeyFunc( int Key, int x, int y );

/* Simple  transformation routine */
GLvoid Transform(GLfloat Width, GLfloat Height)
{
 glViewport(0, 0, Width, Height);            /* Set the viewport */
 glMatrixMode(GL_PROJECTION);               /* Select the projection matrix */
 glLoadIdentity();                                  /* Reset The Projection Matrix */
 gluPerspective(45.0,Width/Height,0.1,100.0);  /* Calculate The Aspect Ratio Of The Window */
 glMatrixMode(GL_MODELVIEW);                /* Switch back to the modelview matrix */
}


/* A general OpenGL initialization function.  Sets all of the initial parameters. */
GLvoid InitGL(GLfloat Width, GLfloat Height)
{

 glClearColor(1.0, 1.0, 1.0, 1.0);
 glLineWidth(2.0);            /* Add line width,   ditto */
 Transform( Width, Height ); /* Perform the transformation */
 //newly added
 t=gluNewQuadric();
     gluQuadricDrawStyle(t, GLU_FILL);

glEnable(GL_LIGHTING);
```

```
glEnable(GL_LIGHT0);

// Create light components
GLfloat ambientLight[] = { 0.2f, 0.2f, 0.2f, 1.0f };
GLfloat diffuseLight[] = { 0.8f, 0.8f, 0.8, 1.0f };
GLfloat specularLight[] = { 0.5f, 0.5f, 0.5f, 1.0f };
GLfloat position[] = { 1.5f, 1.0f, 4.0f, 1.0f };

// Assign created components to GL_LIGHT0
glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);
glLightfv(GL_LIGHT0, GL_SPECULAR, specularLight);
glLightfv(GL_LIGHT0, GL_POSITION, position);

}

/* The function called when our window is resized  */
GLvoid ReSizeGLScene(GLint Width, GLint Height)
{
 if (Height==0)    Height=1;              /* Sanity checks */
 if (Width==0)     Width=1;
 Transform( Width, Height );              /* Perform the transformation */
}

void init()
{
   glClearColor(0,0,0,0);
          glPointSize(5.0);
          glMatrixMode(GL_PROJECTION);
          glLoadIdentity();
          glOrtho(0.0,900.0,0.0,600.0,50.0,-50.0);
          glutPostRedisplay();              // request redisplay
}


/* The main drawing function

  In here we put all the OpenGL and calls to routines which manipulate
  the OpenGL state and environment.

  This is the function which will be called when a "redisplay" is requested.
*/

void display_string(int x, int y, char *string, int font)
{
   int len,i;
          glColor3f(0.8,0.52,1.0);
          glRasterPos2f(x, y);
   len = (int) strlen(string);
   for (i = 0; i < len; i++) {
   if(font==1)
          glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,string[i]);
```

```
        if(font==2)
  glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,string[i]);
        if(font==3)
            glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12,string[i]);
        if(font==4)
  glutBitmapCharacter(GLUT_BITMAP_HELVETICA_10,string[i]);
        }

}

void display1(void)
{

        glClearColor(1.0,1.0,0.1,1.0);
        display_string(180,540,"NAME OF THE ENGINEERING COLLEGE",1); //correct cordinate according to
name
        display_string(215,500,name3,1);
        display_string(390,470,"HELP",2);
        display_string(10,450,"MOUSE",2);
        display_string(10,410,"PRESS RIGHT BUTTON FOR MENU",3);
        display_string(10,370,"KEYBOARD",2);
        display_string(10,340,"X-Y-Z KEYS FOR CORRESPONDING ROTATION",3);
        display_string(10,310,"A-S-Q CAR CUSTOM SIZE SELECTION",3);
        display_string(10,280,"U-F FOR CAMERA VIEW SETTINGS",3);
        display_string(10,250,"USE LEFT ARROW(<-) AND RIGHT ARROW(->) TO MOVE CAR",3);
        display_string(10,220,"ESCAPE TO EXIT",3);
        display_string(250,150,"PRESS SPACE BAR TO ENTER",2);
        glutPostRedisplay();
        glutSwapBuffers();

}

GLvoid DrawGLScene()
{


 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); /* Clear The Screen And The Depth Buffer */
if(view==0)
{
init();
display1();
}
else
{
 if(count==1)
        InitGL(Xsize,Ysize);
 if(aflag==1)/* Initialize our window. */
 glClearColor(1,1,1,1);
 else
         glClearColor(0.1,0.1,0.1,0);
 glPushMatrix();
 glLoadIdentity();
```

```
 glTranslatef(-1.0,0.0,-3.5);
 glRotatef(xangle,1.0,0.0,0.0);
 glRotatef(yangle,0.0,1.0,0.0);
 glRotatef(zangle,0.0,0.0,1.0);
 glTranslatef(xt,yt,zt);
 glScalef(xs,ys,zs);
 glEnable(GL_COLOR_MATERIAL);
 glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

 if(flag2==1)
 {
 GLfloat fogcolour[4]={1.0,1.0,1.0,1.0};

 glFogfv(GL_FOG_COLOR,fogcolour);          /* Define the fog colour */
 glFogf(GL_FOG_DENSITY,0.1);              /* How dense */
 glFogi(GL_FOG_MODE,GL_EXP);               /* exponential decay */
 glFogf(GL_FOG_START,3.0);               /* Where wwe start fogging */
 glFogf(GL_FOG_END,100.0);                /* end */
 glHint(GL_FOG_HINT, GL_FASTEST);          /* compute per vertex */
 glEnable(GL_FOG);/* ENABLE */
 }
 if(flag2==0)
 {
        glDisable(GL_FOG);
 }

if(!aflag){
 glBegin(GL_POINTS);
 glColor3f(1,1,1);
 glPointSize(200.0);
 int ccount=0;
 float x=10,y=10;
 while(ccount<20)
 {
        glVertex2f(x,y);
        x+=10;
        y+=10;
        if(y>Ysize) y-=10;
        if(x>Xsize) x-=10;
        ccount++;
 }
 glEnd();}

 glColor3f(1.0,.75,0.0);
 glPointSize(30.0);
 glBegin(GL_POINTS);
 glVertex3f(0.2,0.3,0.3);
 glVertex3f(0.2,0.3,0.5);
 glEnd();
 glPointSize(200.0);
```

```
 glBegin(GL_QUADS);

//*********************ENTER WINDOW*********************************
 glColor3f(0.3,0.3,0.3);
 glVertex3f( 0.77, 0.63,0.2);
 glVertex3f(0.75, 0.5,0.2);       //quad front window
 glVertex3f(1.2, 0.5, 0.2);
 glVertex3f( 1.22,0.63,0.2);

 glVertex3f(1.27,0.63,.2);
 glVertex3f(1.25,0.5,0.2);       //quad back window
 glVertex3f(1.65,0.5,0.2);
 glVertex3f(1.67,0.63,0.2);

 glColor3f(r,g,b);
 glVertex3f(0.7,0.65,0.2);
 glVertex3f(0.7,0.5,.2);      //first separation
 glVertex3f(0.75,0.5,0.2);
 glVertex3f(0.77,0.65,0.2);

 glVertex3f(1.2,0.65,0.2);
 glVertex3f(1.2,0.5,.2);      //second separation
 glVertex3f(1.25,0.5,0.2);
 glVertex3f(1.27,0.65,0.2);

 glVertex3f(1.65,0.65,0.2);
 glVertex3f(1.65,0.5,.2);    //3d separation
 glVertex3f(1.7,0.5,0.2);
 glVertex3f(1.7,0.65,0.2);

 glVertex3f( 0.75, 0.65,0.2);
 glVertex3f(0.75, 0.63,0.2);       //line strip
 glVertex3f(1.7, 0.63, 0.2);
 glVertex3f( 1.7,0.65,0.2);

 glVertex3f( 0.75, 0.65,0.6);
 glVertex3f(0.75, 0.63,0.6);       //line strip
 glVertex3f(1.7, 0.63, 0.6);
 glVertex3f( 1.7,0.65,0.6);

 glColor3f(0.3,0.3,0.3);
 glVertex3f( 0.77, 0.63,0.6);
 glVertex3f(0.75, 0.5,0.6);       //quad front window
 glVertex3f(1.2, 0.5, 0.6);
 glVertex3f( 1.22,0.63,0.6);

 glVertex3f(1.27,0.63,.6);
 glVertex3f(1.25,0.5,0.6);       //quad back window
 glVertex3f(1.65,0.5,0.6);
 glVertex3f(1.67,0.63,0.6);

 glColor3f(r,g,b);
```

```
glVertex3f(0.7,0.65,0.6);
glVertex3f(0.7,0.5,.6);      //first separation
glVertex3f(0.75,0.5,0.6);
glVertex3f(0.77,0.65,0.6);

glVertex3f(1.2,0.65,0.6);
glVertex3f(1.2,0.5,.6);      //second separation
glVertex3f(1.25,0.5,0.6);
glVertex3f(1.27,0.65,0.6);

glVertex3f(1.65,0.65,0.6);
glVertex3f(1.65,0.5,.6);
glVertex3f(1.7,0.5,0.6);
glVertex3f(1.7,0.65,0.6);
glEnd();

glBegin(GL_QUADS);

/* top of cube*/
glColor3f(0.3,0.3,0.3);
glVertex3f( 0.6, 0.5,0.6);
glVertex3f(0.6, 0.5,0.2);       //quad front window
glVertex3f(0.7, 0.65, 0.2);
glVertex3f( 0.7,0.65,0.6);

glVertex3f(1.7,0.65,.6);
glVertex3f(1.7,0.65,0.2);       //quad back window
glVertex3f(1.8,0.5,0.2);
glVertex3f(1.8,0.5,0.6);

if(flag1)
{
        glPushMatrix();
        glTranslatef(xw,0,0);
glColor3f(0,1,0);
glVertex3f(-100,0.1,-100);
glVertex3f(-100,0.1,0);        //a green surroundings
glVertex3f(100,0.1,0);
glVertex3f(100,0.1,-100);

glColor3f(0.7,0.7,0.7);
glVertex3f(-100,0.1,0);
glVertex3f(-100,0.1,0.45);       //a long road
glVertex3f(100,0.1,0.45);
glVertex3f(100,0.1,0);

glColor3f(1.0,0.75,0.0);
glVertex3f(-100,0.1,0.45);      //a median
glVertex3f(-100,0.1,0.55);
glVertex3f(100,0.1,0.55);
glVertex3f(100,0.1,0.45);
```

```
 glColor3f(0.7,0.7,0.7);
 glVertex3f(-100,0.1,0.55);
 glVertex3f(-100,0.1,1);        //a long road
 glVertex3f(100,0.1,1);
 glVertex3f(100,0.1,0.55);

 glColor3f(0,1,0);
 glVertex3f(-100,0.1,1);
 glVertex3f(-100,0.1,100);       //a green surroundings
 glVertex3f(100,0.1,100);
 glVertex3f(100,0.1,1);
  glPopMatrix();
 }
 glEnd();

 if(wheelflag)
 {
        glPushMatrix();
        glTranslatef(xw,0,0);
        glColor3f(0.5,.2,0.3);
        glBegin(GL_QUADS);
        for(i=0;i<200;i+=0.2)
        {
                glVertex3f(-100+i,0,1);
                glVertex3f(-99.9+i,0,1);
                glVertex3f(-99.9+i,0.2,1);
                glVertex3f(-100+i,0.2,1);
                i+=0.5;
        }
        for(i=0;i<200;i+=0.2)
        {
                glVertex3f(-100+i,0,0);
                glVertex3f(-99.9+i,0,0);
                glVertex3f(-99.9+i,0.2,0);
                glVertex3f(-100+i,0.2,0);
                i+=0.5;
        }
        glEnd();
        glPopMatrix();
 }
glBegin(GL_TRIANGLES);            /* start drawing the cube.*/

 /* top of cube*/
 glColor3f(0.3,0.3,0.3);
 glVertex3f( 0.6, 0.5,0.6);
 glVertex3f( 0.7,0.65,0.6);     //tri front window
 glVertex3f(0.7,0.5,0.6);

 glVertex3f( 0.6, 0.5,0.2);
 glVertex3f( 0.7,0.65,0.2);     //tri front window
 glVertex3f(0.7,0.5,0.2);
```

```
 glVertex3f( 1.7, 0.65,0.2);
 glVertex3f( 1.8,0.5,0.2);      //tri back window
 glVertex3f( 1.7,0.5,0.2);

 glVertex3f( 1.7, 0.65,0.6);
 glVertex3f( 1.8,0.5,0.6);      //tri back window
 glVertex3f(1.7,0.5,0.6);

glEnd();

//************IGNITION SYSTEM**********************************

glPushMatrix();
  glColor3f(0.7,0.7,0.7);
  glTranslatef(1.65,0.2,0.3);
  glRotatef(90.0,0,1,0);
  gluCylinder(t,0.02,0.03,.5,10,10);
glPopMatrix();

//********************WHEEL*********************************************

glColor3f(0.7,0.7,0.7);
glPushMatrix();
 glBegin(GL_LINE_STRIP);
          for(theta=0;theta<360;theta=theta+20)
          {
 glVertex3f(0.6,0.2,0.62);
 glVertex3f(0.6+(0.08*(cos(((theta+angle)*3.14)/180))),0.2+(0.08*(sin(((theta+angle)*3.14)/180))),0.62);
          }
glEnd();

glBegin(GL_LINE_STRIP);
          for(theta=0;theta<360;theta=theta+20)
          {
 glVertex3f(0.6,0.2,0.18);
 glVertex3f(0.6+(0.08*(cos(((theta+angle)*3.14)/180))),0.2+(0.08*(sin(((theta+angle)*3.14)/180))),0.18);
          }
glEnd();

glBegin(GL_LINE_STRIP);
for(theta=0;theta<360;theta=theta+20)
          {
 glVertex3f(1.7,0.2,0.18);
 glVertex3f(1.7+(0.08*(cos(((theta+angle)*3.14)/180))),0.2+(0.08*(sin(((theta+angle)*3.14)/180))),0.18);
          }
glEnd();

glBegin(GL_LINE_STRIP);
for(theta=0;theta<360;theta=theta+20)
          {
 glVertex3f(1.7,0.2,0.62);
glVertex3f(1.7+(0.08*(cos(((theta+angle)*3.14)/180))),0.2+(0.08*(sin(((theta+angle)*3.14)/180))),0.62);
```

```
            }
glEnd();
 glTranslatef(0.6,0.2,0.6);
 glColor3f(0,0,0);
 glutSolidTorus(0.025,0.07,10,25);

 glTranslatef(0,0,-0.4);
 glutSolidTorus(0.025,0.07,10,25);

 glTranslatef(1.1,0,0);
 glutSolidTorus(0.025,0.07,10,25);

 glTranslatef(0,0,0.4);
 glutSolidTorus(0.025,0.07,10,25);
 glPopMatrix();
 glPopMatrix();
 glEnable(GL_DEPTH_TEST);
 glutPostRedisplay();
 glutSwapBuffers();
 }
}

void NormalKey(GLubyte key, GLint x, GLint y)
{
   switch ( key )   {
    case ESCAPE : printf("escape pressed. exit.\n");
                  glutDestroyWindow(window);          /* Kill our window */
                  exit(0);
                      break;

     case ' ':view=1;
              DrawGLScene();
              break;

     case 'x': xangle += 5.0;
                glutPostRedisplay();
                break;

      case 'X':xangle -= 5.0;
                glutPostRedisplay();
                break;

     case 'y':
       yangle += 5.0;
       glutPostRedisplay();
       break;

    case 'Y':
       yangle -= 5.0;
       glutPostRedisplay();
       break;
```

```
case 'z':
  zangle += 5.0;
  glutPostRedisplay();
  break;

case 'Z':
  zangle -= 5.0;
  glutPostRedisplay();
  break;

 case 'u':                    /* Move up */
  yt += 0.2;
  glutPostRedisplay();
  break;

 case 'U':
  yt -= 0.2;                  /* Move down */
  glutPostRedisplay();
  break;

 case 'f':                    /* Move forward */
  zt += 0.2;
  glutPostRedisplay();
  break;

 case 'F':
  zt -= 0.2;                  /* Move away */
  glutPostRedisplay();
  break;

        case 's':zs+=.2;
                 glutPostRedisplay();
                 break;

        case 'S':zs-=0.2;
                 glutPostRedisplay();
                 break;

        case 'a':ys+=.2;
                 glutPostRedisplay();
                 break;

        case 'A':ys-=0.2;
                 glutPostRedisplay();
                 break;

        case 'q':xs+=.2;
                 glutPostRedisplay();
                 break;

        case 'Q':xs-=0.2;
                 glutPostRedisplay();
```

```
                    break;

    default:
            break;
   }

}

static void SpecialKeyFunc( int Key, int x, int y )
{
            switch ( Key ) {
            case GLUT_KEY_RIGHT:
                        if(!wheelflag)
                        xt += 0.2;
                        if(wheelflag)
                        {
                        angle+=5;
                        xw+=0.2;
                        }
                  glutPostRedisplay();
                        break;

                   case GLUT_KEY_LEFT:
                        if(!wheelflag)
                        xt -= 0.2;
                        if(wheelflag)
                        {
                        angle+=5;
                        xw-=0.2;
                        }
       glutPostRedisplay();
                        break;
            }
}

void myMenu(int id)
{
                        if (id==1)
            {
                        flag1=0;
                        wheelflag=0;
            glutPostRedisplay();

            }
            if(id ==2)
            {
                        flag1=1;
                        flag2=0;
                        wheelflag=0;
                        xangle += 5.0;
                        glutPostRedisplay();
            }
```

```
if(id==3)
{
        flag2=1;
        wheelflag=0;
        xangle += 5.0;
        glutPostRedisplay();
}
if (id==4)
{
         wheelflag=1;
         glutPostRedisplay();
}
if (id==5)
{
if(day)
 {

        if(light)
        {
                count++;
                glDisable(GL_LIGHTING);
                glDisable(GL_LIGHT0);
                light=0;
        }
        else
        {
                count--;
                light=1;
                glEnable(GL_LIGHTING);
                glEnable(GL_LIGHT0);
        }
        glutPostRedisplay();
}
else
 {

  if(nml==0 && flag2==2)
    {
     flag2=0;
     nml=1;
    }
   else
    {
       flag2=2;
 nml=0;

 aflag=0;
 day=0;

      glClearColor(0.1,0.1,0.1,0);
      GLfloat fogcolour[4]={0.0,0.0,0.0,1.0};
```

```
            glFogfv(GL_FOG_COLOR,fogcolour);
            glFogf(GL_FOG_DENSITY,0.5);              /* How dense */
            glFogi(GL_FOG_MODE,GL_EXP);
           glHint(GL_FOG_HINT, GL_FASTEST);
           glEnable(GL_FOG);

                glutPostRedisplay();
            }
         }

          }

     if(id==12)
          {
          aflag=1;
      day=1;
          glClearColor(1,1,1,1);
          glDisable(GL_FOG);
          glutPostRedisplay();
          }

          if(id==13)
          {
          aflag=0;
      day=0;
          flag2=2;
          glClearColor(0.1,0.1,0.1,0);
          GLfloat fogcolour[4]={0.0,0.0,0.0,1.0};

  glFogfv(GL_FOG_COLOR,fogcolour);          /* Define the fog colour */
  glFogf(GL_FOG_DENSITY,0.5);              /* How dense */
  glFogi(GL_FOG_MODE,GL_EXP);               /* exponential decay */
                  /* end */
  glHint(GL_FOG_HINT, GL_FASTEST);          /* compute per vertex */
  glEnable(GL_FOG);

  glutPostRedisplay();
     }
}

void colorMenu(int id)
{
                    if (id==6)
          {
                    r=g=0;
                    b=1;
          glutPostRedisplay();

          }
          if(id ==7)
          {
                    r=0.8;
```

```
                b=g=0;
                glutPostRedisplay();
        }
        if(id==8)
        {
          g=1;
                r=b=0;
                glutPostRedisplay();
        }
        if (id==9)
        {
                 r=b=g=0;
                 glutPostRedisplay();
        }
        if(id==10)
        {
                b=0;
                r=g=1;
    glutPostRedisplay();
        }
        if(id==11)
        {
                b=r=g=.7;
    glutPostRedisplay();
        }

}

void myreshape(int w,int h)
{
    glViewport(0,0,w,h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if(w<=h)
        glOrtho(-2.0,2.0,-2.0*(GLfloat)h/(GLfloat)w,2.0*(GLfloat)h/(GLfloat)w,-10.0,10.0);
    else
        glOrtho(-2.0*(GLfloat)w/(GLfloat)h,2.0*(GLfloat)w/(GLfloat)h,-2.0,2.0,-10.0,10.0);
    glMatrixMode(GL_MODELVIEW);
    glutPostRedisplay();
}


//*************************** Main************************

int main(int argc, char **argv)
{

/* Initialisation and window creation */

 glutInit(&argc, argv);            /* Initialize GLUT state. */

 glutInitDisplayMode(GLUT_RGBA |     /* RGB and Alpha */
```

```
            GLUT_DOUBLE|    /* double buffer */
            GLUT_DEPTH);    /* Z buffer (depth) */

 glutInitWindowSize(Xsize,Ysize);    /* set initial window size. */
 glutInitWindowPosition(0,0);        /* upper left corner of the screen. */

 glutCreateWindow("3D CAR ANIMATION"); /* Open a window with a title. */

/* Now register the various callback functions */

 glutReshapeFunc(myreshape);
 glutDisplayFunc(DrawGLScene);       /* Function to do all our OpenGL drawing. */
 glutReshapeFunc(ReSizeGLScene);
 glutKeyboardFunc(NormalKey);        /*Normal key is pressed */
 glutSpecialFunc( SpecialKeyFunc );
 InitGL(Xsize,Ysize);
 int submenu=glutCreateMenu(colorMenu);
 glutAddMenuEntry("blue", 6);
        glutAddMenuEntry("red", 7);
        glutAddMenuEntry("green",8);
        glutAddMenuEntry("black",9);
        glutAddMenuEntry("yellow",10);
        glutAddMenuEntry("grey",11);
   glutCreateMenu(myMenu);
        glutAddMenuEntry("car model mode", 1);
        glutAddMenuEntry("car driving mode", 2);
        glutAddMenuEntry("fog effect",3);
        glutAddMenuEntry("wheel effect",4);
        glutAddMenuEntry("toggle light",5);
        glutAddSubMenu("car colors",submenu);
        glutAddMenuEntry("daymode",12);
        glutAddMenuEntry("Night mode",13);
        glutAttachMenu(GLUT_RIGHT_BUTTON);

 /* Now drop into the event loop from which we never return */

 glutMainLoop();
 return 1;
}
```

5. Screenshots

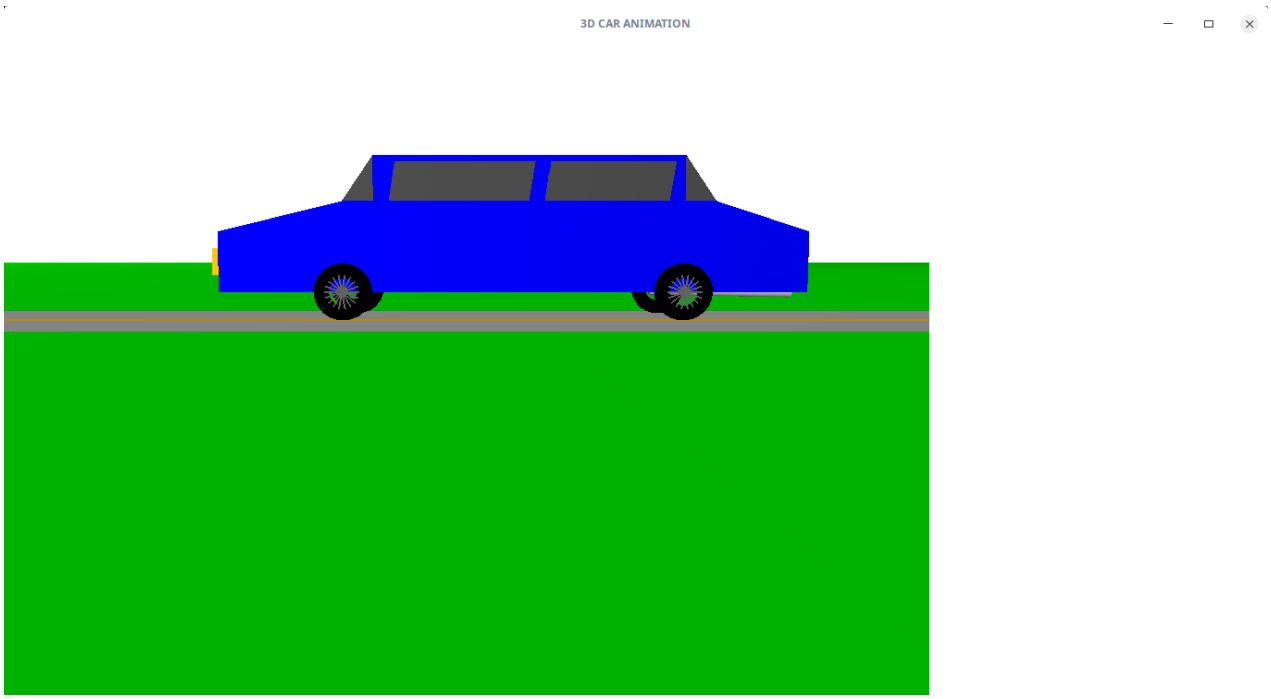

Fig.3-First Graphics Page

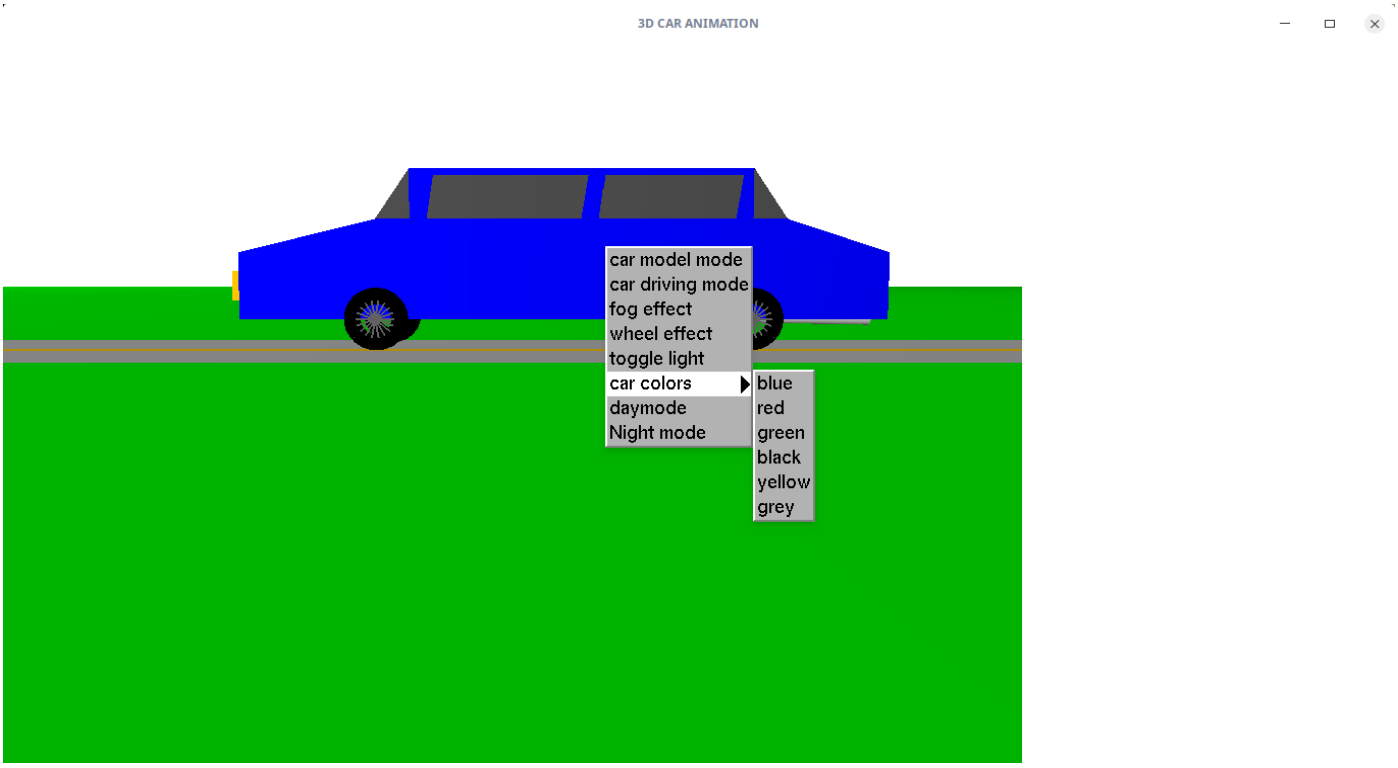

Fig.4-Second Graphics Page

Fig.5-Third Graphics page

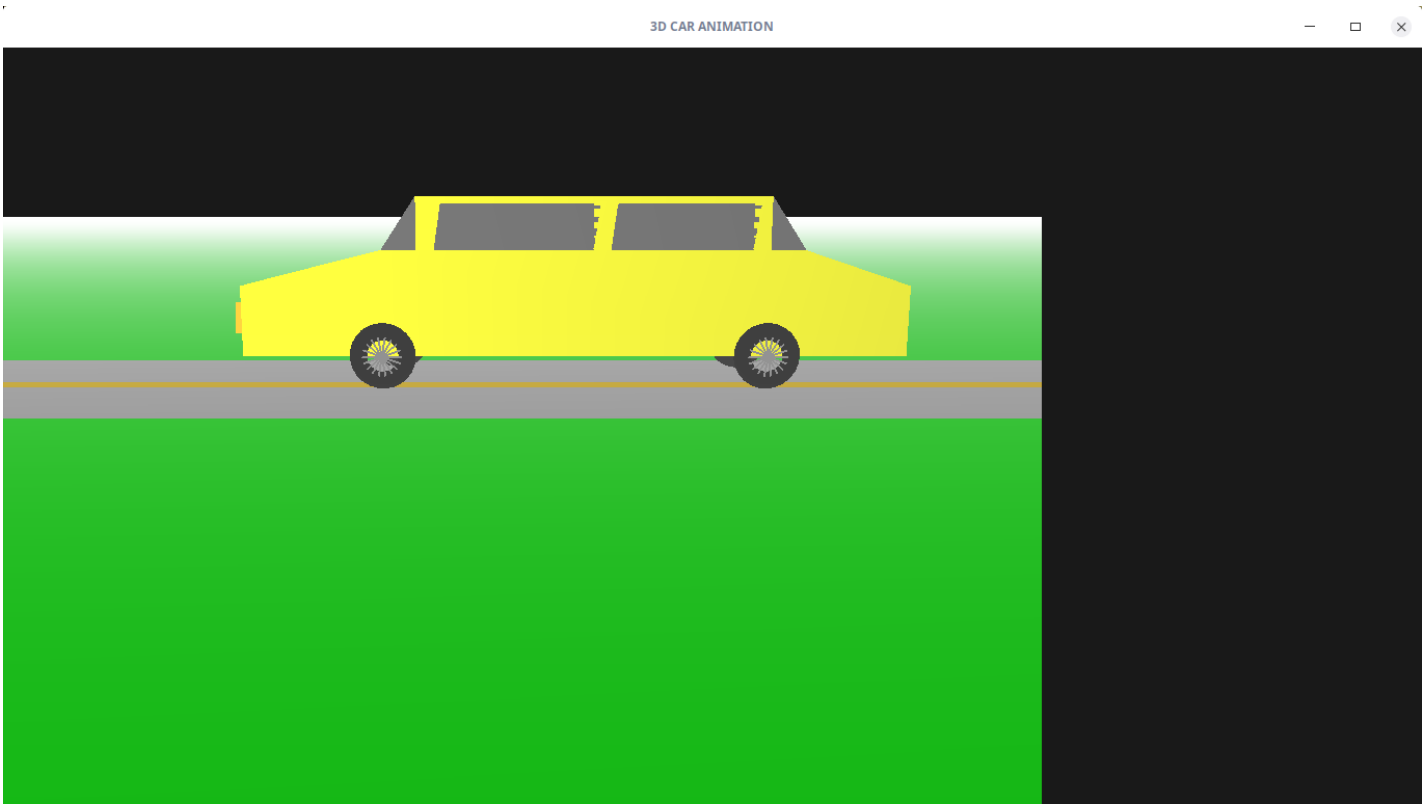

Fig.6-Fourth Graphics Page

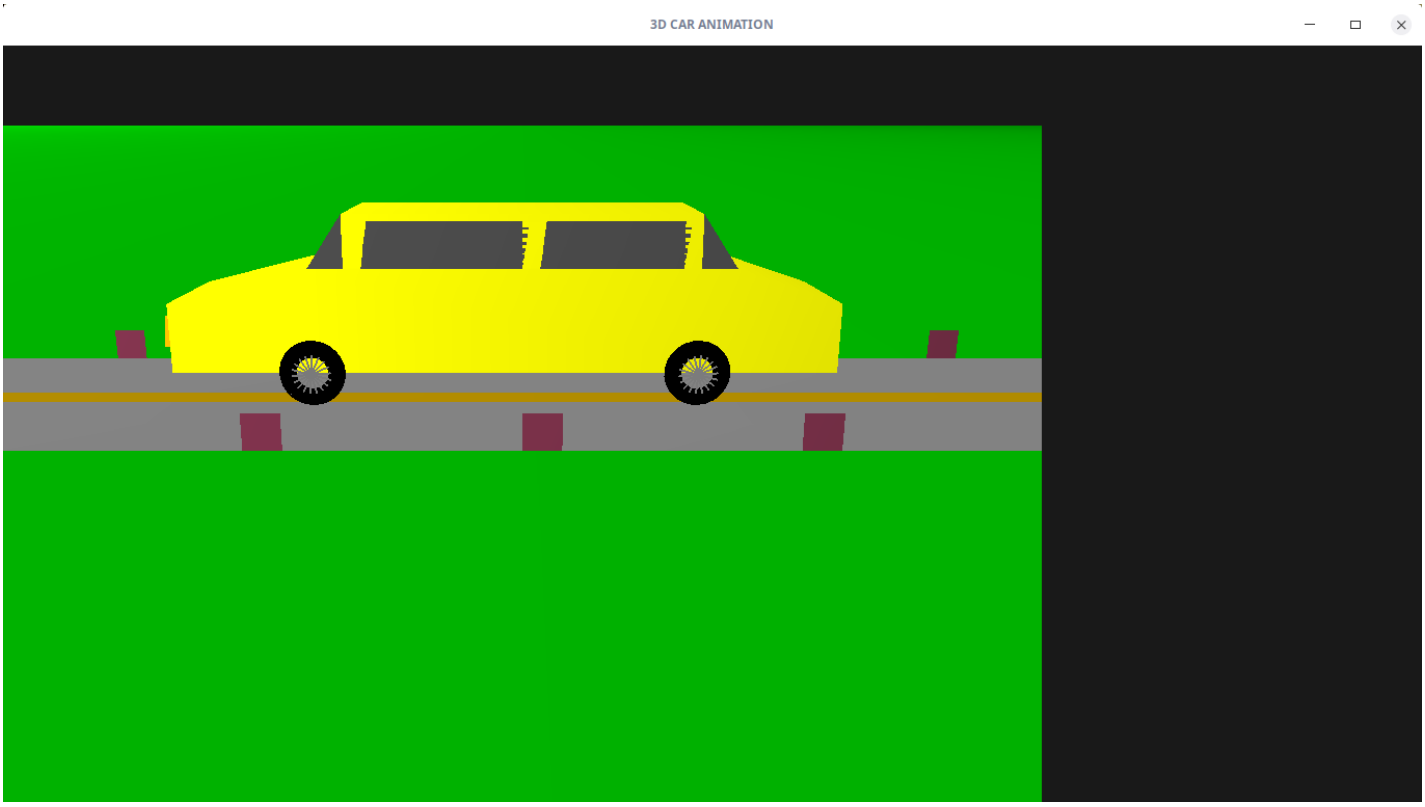

Fig.7-Fifth Graphics Page

6. CONCLUSION

The project "3d Car Simulation" clearly demonstrates the simulation of 3d Car Simulation using OpenGL. Finally we conclude that this program clearly illustrate the 3d Car Simulation using OpenGL and has been completed successfully and is ready to be demonstrated.

7. REFERENCES

[1].James D Foley, Andries Van Dam, Steven K Feiner, John F Huges Computer graphics with OpenGL: pearson education.

[2].Kelvin Sung, Peter Shirley, Steven Baer: Interactive Computer Graphics, concepts and applications, Cengage Learning.