VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"Jnana Sangama", Belagavi-590018

A
Mini Project Report on

"3D CAR ANIMATION"
Submitted in partial fulfillment of the requirement for the COMPUTER GRAPHICS AND IMAGE

PROCESSING LABORATORY [21CSL66] course of VI semester

Bachelor of Engineering Computer Science and Engineering
in

Submitted by:

Umesh Laxman Nayak
2JH21CS115

Under the Guidance of

Prof. Rajeshwari V Patil
Assistant Professor, Department of CSE

Department of Computer Science and Engineering

JAIN COLLEGE OF ENGINEERING AND TECHNOLOGY, HUBBALLI.
Hebballi Road, Sainagar, Hubballi-580031

2023-2024

ABSTRACT

3D computer graphics or three-dimensional computer graphics (in contrast with 2D computer graphics) are graphics that use a three dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering 2D images. Such images may be stored for viewing later or displayed in real time.

3D computer graphics rely on many of the same algorithms as 2D computer vector graphics in the wire frame model and 2D computer raster graphics in the final rendered display. In computer graphics software, 2D applications may use 3D techniques to achieve effects such as lighting, and 3D may use 2D rendering techniques.

3D computer graphics are often referred to as 3D models. Apart from the rendered graphic, the model is contained within the graphical data file. However, there are differences: a 3D model is mathematical representation of any three-dimensional object. A model is not technically a graphic until it is displayed. A model can be displayed visually as a two-dimensional image through a process called 3D rendering or used in non-graphical computer simulations and calculations. With 3D printing, 3D models are similarly rendered into a 3D physical representation of the model, with limitations to how accurate the rendering can match the virtual model.

CONTENTS

LIST OF FIGURES

CHAPTER-1

INTRODUCTION

1.1   COMPUTER GRAPHICS

In this era of computing, computer graphics has become one of the most powerful and interesting fact of computing. It all started with display of data on hardcopy and CRT screen. Now computer graphics is about creation, retrieval, manipulation of models and images.

Graphics today is used in many different areas. Graphics provides one of the most natural means of communicating within a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and effectively. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results.

OpenGL is an application program interface (API) offering various functions to implement primitives, models and images. This offers functions to create and manipulate render lighting, coloring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL offers translation, rotation and scaling of objects.

FIG:1.1   OVERVIEW OF COMPUTER GRAPHICS

1.2 OpenGL CONCEPT

1.2.1 INTERFACE

OpenGL is an application program interface (API) offering various functions to implement primitives, models and images. This offers functions to create and manipulate render lighting, coloring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL o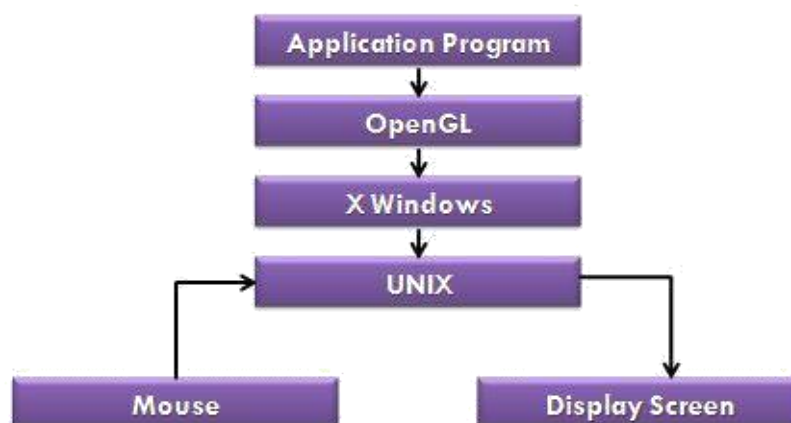ffers translation, rotation and scaling of objects. Functions in the main GL library have names that begins with *gl* and are stored in a library usually referred to as GL. The second is the **OpenGL Utility Library(GLU)**. The library uses only GL functions but contains code for creating common objects and simplifying viewing.

All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begin with the letter *glu*. Rather than using a different library for each system we use a readily available library called OpenGL Utility Toolkit(GLUT),which provides the minimum functionality that should be expected in any modern windowing system.

1.2.2 OVERVIEW

- OpenGL(Open Graphics Library) is the interface between a graphics program and graphics hardware. *It is streamlined*. In other words, it provides low-level functionality. For example, all objects are built from points, lines and convex polygons. Higher level objects like cubes are implemented as six four-sided polygons.
- OpenGL supports features like 3-dimensions, lighting, anti-aliasing, shadows, textures, depth effects, etc.
- It is system-independent. It does not assume anything about hardware or operating system and is only concerned with efficiently rendering mathematically described scenes. As a result, it does not provide any windowing capabilities.
- It is a state machine. At any moment during the execution of a program there is a current model transformation.
- It is a rendering pipeline. The rendering pipeline consists of the following steps:
    * Defines objects mathematically.
    * Arranges objects in space relative to a viewpoint.
    * Calculates the color of the objects.

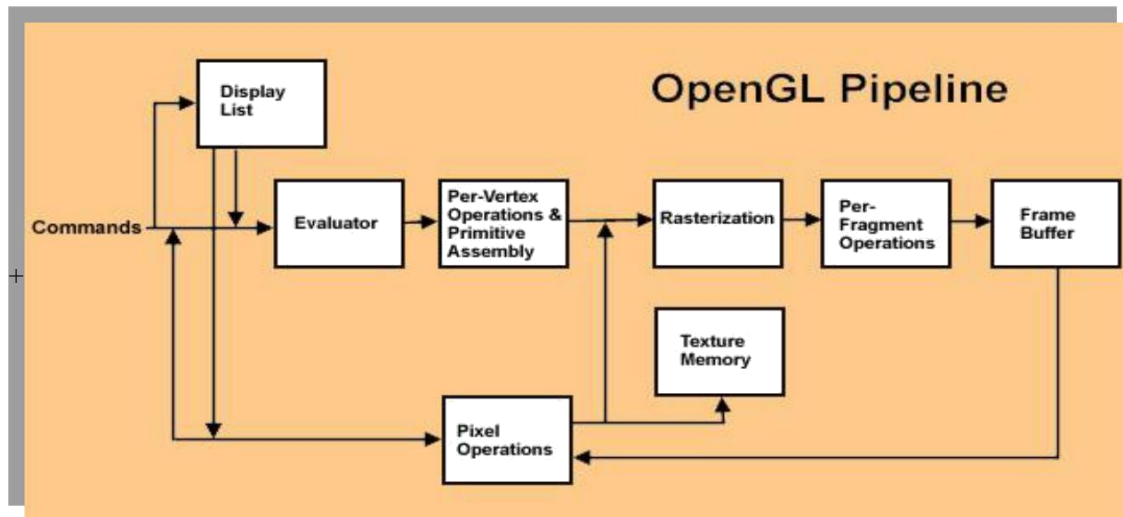## 1.2.3 OPENGL ARCHITECTURE:

1) Pipeline Architectures

FIG:1.2.3.1 OPENGL PIPELINE ARCHITECTURE

2) OpenGL Engine And Drivers

FIG:1.2.3.2 OPENGL ENGINE AND DRIVERS

3) Application Development-API's
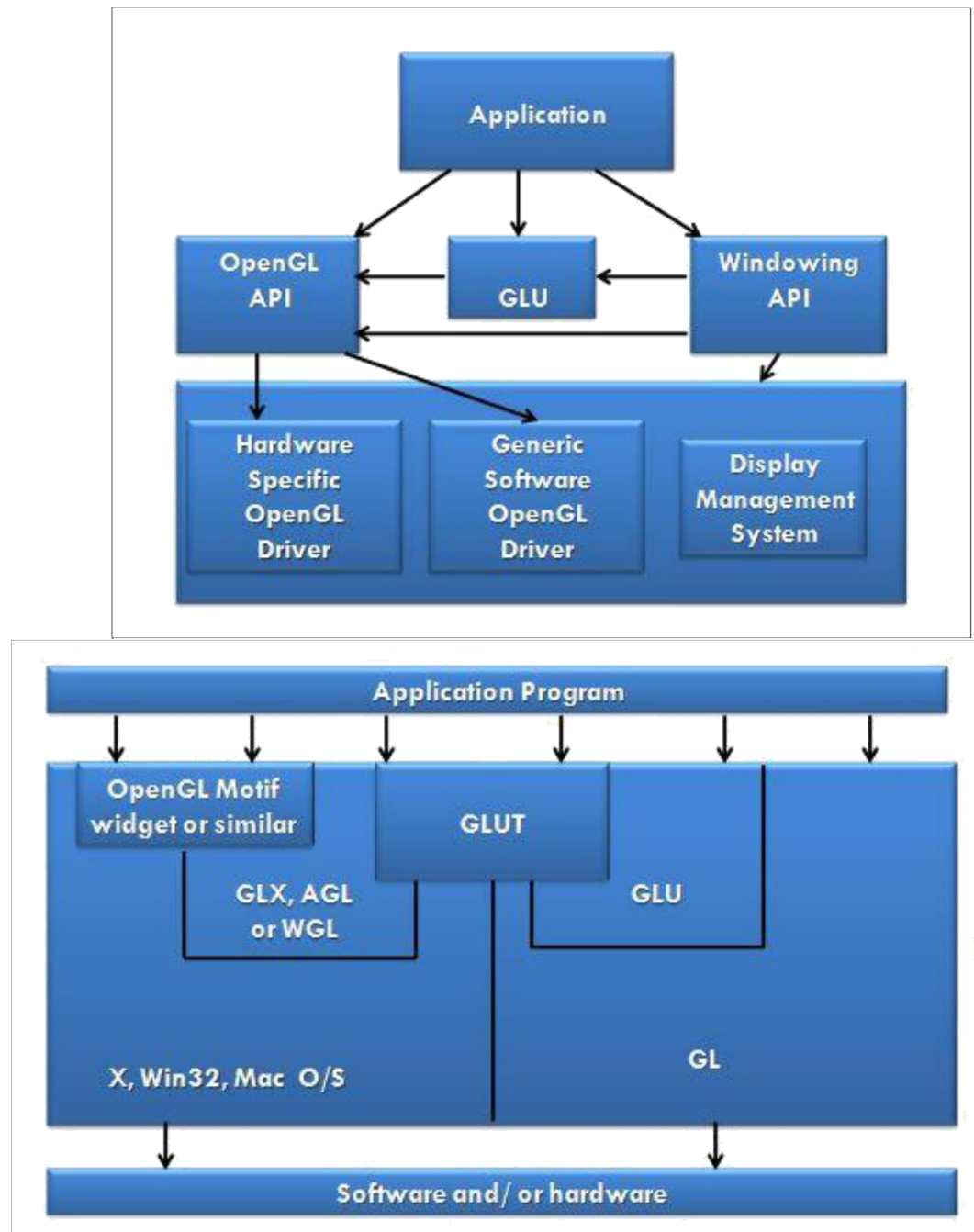
FIG:1.2.3.3 APPLICATIONS DEVELOPMENT(API'S)

The above diagram illustrates the relationship of the various libraries and window system components.

Generally, applications which require more user interface support will use a library designed to support those types of features (i.e. buttons, menu and scroll bars, etc.) such as Motif or the Win32 API.

Prototype applications, or ones which do not require all the bells and whistles of a full GUI, may choose to use GLUT instead because of its simplified programming model and window system independence.

Display Lists:

All data, whether it describes geometry or pixels, can be saved in a *display list* for current or later use. (1 alternative to retaining data in a displaylist is processing the data immediately - also known as *immediate mode*.) When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

Evaluators:

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions. Evaluators provide a method to derive the vertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colors, and spatial coordinate values from the control points.

Per-Vertex Operations

For vertex data, next is the "per-vertex operations" stage, which converts. The vertices into primitives. Some vertex data (for example, spatial coordinates) are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen.

If advanced features are enabled, this stage is even busier. If texturing is used, texture coordinates may be generated and transformed here. If lighting is enabled, the lighting calculations are performed using the transformed vertex, surface normal, light source position, material properties, and other lighting information to produce a color value.

Primitive Assembly

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half-space, defined by a plane. Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending upon how the line or polygon is clipped.

In some cases, this is followed by perspective division, which makes distant geometric objects appear smaller than closer objects. Then view port

and depth (z coordinate) operations are applied. If culling is enabled and the primitive is a polygon, it then may be rejected by a culling test. Depending upon the polygon mode, a polygon may be drawn as points or lines. The results or this stage are complete geometric primitives, which are the transformed and clipped vertices with related color, depth, and sometimes texture-coordinate values and guidelines for the rasterization step.

Pixel Operations

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step. If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory. There are special pixel copy operations to copy data in the frame buffer to other parts of

the frame buffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the frame buffer.

Texture Assembly

An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. If several texture images are used, it's wise to put them into texture objects so that you can easily switch among them. Some OpenGL implementations may have special resources to accelerate texture performance. There may be specialized, high-performance texture memory. If this memory is available, the texture objects may be prioritized to control the use of this limited and valuable resource.

Rasterization

Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the frame buffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support ant aliasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square.

CHAPTER-2

REQUIREMENT SPECIFICATION

2.1 SOFTWARE REQUIREMENTS SPECIFICATION

This section attempts to bring out the requirements and specifications as given out by the Visvesvaraya Technological University for the completion of the package. Minimum requirements expected are cursor movement, editing picture objects like point, line, circle, ellipse and polygons. Transformations on objects/selected area should be possible. User should be able to open the package do the required operations and exit from the environment.

2.2 EXTERNAL INTERFACE REQUIREMENTS

User Interface:

The interface for the 2D package requires for the user to have a mouse connected, and the corresponding drivers software and header files installed. For the convenience of the user, there are menus and sub -menus displayed on the screen.

Menus:

The Menus consists of various operations related to drawing on the area specified. It also has the 'clear' option for clearing the screen and also changes the background color.

Hardware Interface:

The standard output device, as mentioned earlier has been assumed to be a color monitor. It is quite essential for any graphics package to have this, as provision of color options to the user is a must. The mouse, the main input device, has to be functional. A keyboard is also required.

Apart from these hardware requirements, there should be sufficient hard disk space and primary memory available for proper working of the package.

Software Interface:

The editor has been implemented on the DOS platform and mainly requires an appropriate version of the compiler to be installed and functional. Though it has been implemented on DOS, it is pretty much platform independent with the

Hardware Requirements:

System	**:**	**Intel**
Frequency	**:**	**3.0 GHz**
RAM	**:**	**4 GB**
Disk Capacity	**:**	**1 TB**

Software Requirements:

Operating System :   WINDOWS 7/8/10
Compiler, IDE   :   DEV C++

CHAPTER-3

SYSTEM DESIGN

3.1 Program Structure : The program consist of many functions such as display function, reshape function, text display function, call back function and the main function.
Logically, the program consists of multiple *"View"*. The view in the main window which provides the user a list of choices to display the number.

3.2 Start Screen : As you execute the program, first thing come is the start screen where you will see the details of the project. The start screen have name of college and the name of projects. It also have the user interaction   instructions about the use of mouse and keyboard functionality. There is option of going to main screen, by pressing 'space' key.

3.3 Main Screen: After you get in to main screen from the start screen you will see the 3D Car drawn in blue colour. If you press right mouse button you will get the menu options to select for particular functionality to execute. Details is mentioned in below user interaction.

3.4 User Interaction: The user interaction is one of the most important part of any program. This 3D OpenGL program also have user interaction both using keyboard as well as mouse. Below is the description of the uses and functionality of keys and menus in this 3D OpenGL Progam.

3.5 Mouse Interaction:

Press right mouse button to get the mouse. Following is the menus:-
Car model mode - This is default mode of car display which will display the only car.
Car driving mode - This will display the driving mode, which includes the long road and green filed.
Fog effect - This will apply the fog around the environment of the car.

Wheel effect - This is one of the finest effect, it will animate the car while it moved as it is moving in the car.

Toggle light - This will apply the light effect on/off when selected.

Car colors - This menu have submenu which allow to select the color of car. The submenu have the following options - blue, red, green, black, yellow and grey.

Daymode - By default we have Daymode on, in this projects so while in

Night mode you can select this to toggle to Daylight mode.Night mode - This menu will let you switching to the Night mode, by showing darkness around.

3.6 Keyboard Interaction:

Below is complete description against each key and what they do when pressed.
x- Rotate the car in 'x' direction

y- Rotate the car in 'y' direction

z- Rotate the car in 'z' direction

a- Increase the size of car in'x' direction

s- Increase the size of car in 'y' direction

q- Increase the size of car in 'z' direction

u- Camera top view

f- Camera side view

left arrrow key - Move car in forward direction

right arrow key- Move car in backward direction

esc - Exit from the program

spacebar - Enter the main screen from start screen.

CHAPTER-4   <u>IMPLEMENTATION SOURCE CODE</u>

```c
#include <stdio.h>
#include <stdlib.h>
#include<GL/glut.h>
#include <math.h>
#include <string.h>
#include <graphics.h>
/* ASCII code for the escape key. */
#define ESCAPE 27

GLint window;
GLint window2;
GLint Xsize=1366;
GLint Ysize=768;
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_MODELVIEWodelew 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,"SRI SAIRAM COLLEGE OF ENGINEERING",1);
//correct cordinate according to name
      display_string(215,500,"GOVIND KUMAR &  ASHISH",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);
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);              /* OBJECT MODULE*/

/* top of cube*/
//*********************FRONT
BODY*************************************
glColor3f(r,g,b);
glVertex3f( 0.2, 0.4,0.6);
glVertex3f(0.6, 0.5,0.6);
glVertex3f(0.6, 0.5,0.2);
glVertex3f( 0.2,0.4,0.2);

/* bottom of cube*/
glVertex3f( 0.2,0.2,0.6);
glVertex3f(0.6,0.2,0.6);
glVertex3f(0.6,0.2,0.2);
glVertex3f( 0.2,0.2,0.2);

/* front of cube*/
glVertex3f( 0.2,0.2,0.6);
glVertex3f(0.2, 0.4,0.6);
glVertex3f(0.2,0.4,0.2);
glVertex3f( 0.2,0.2,0.2);

/* back of cube.*/
glVertex3f(0.6,0.2,0.6);
glVertex3f(0.6,0.5,0.6);
glVertex3f(0.6,0.5,0.2);
glVertex3f( 0.6,0.2,0.2);

/* left of cube*/
glVertex3f(0.2,0.2,0.6);
glVertex3f(0.6,0.2,0.6);
glVertex3f(0.6,0.5,0.6);
glVertex3f(0.2,0.4,0.6);

/* Right of cube */
glVertex3f(0.2,0.2,0.2);
glVertex3f( 0.6,0.2,0.2);
glVertex3f( 0.6,0.5,0.2);
glVertex3f( 0.2,0.4,0.2);
```

```
//***************************************************************
*****
  glVertex3f(0.7,0.65,0.6);
  glVertex3f(0.7,0.65,0.2);
  glVertex3f(1.7,0.65,0.2);        //top cover
  glVertex3f(1.7,0.65,0.6);
//**********************back guard**************************
  glColor3f(r,g,b);           /* Set The Color To Blue*/
  glVertex3f( 1.8, 0.5,0.6);
  glVertex3f(1.8, 0.5,0.2);
  glVertex3f(2.1, 0.4, 0.2);
  glVertex3f(2.1,0.4,0.6);

  /* bottom of cube*/
  glVertex3f( 2.1,0.2,0.6);
  glVertex3f(2.1,0.2,0.2);
  glVertex3f(1.8,0.2,0.6);
  glVertex3f( 1.8,0.2,0.6);

  /* back of cube.*/
  glVertex3f(2.1,0.4,0.6);
  glVertex3f(2.1,0.4,0.2);
  glVertex3f(2.1,0.2,0.2);
  glVertex3f(2.1,0.2,0.6);

  /* left of cube*/
  glVertex3f(1.8,0.2,0.2);
  glVertex3f(1.8,0.5,0.2);
  glVertex3f(2.1,0.4,0.2);
  glVertex3f(2.1,0.2,0.2);

  /* Right of cube */
  glVertex3f(1.8,0.2,0.6);
  glVertex3f(1.8,0.5,0.6);
  glVertex3f(2.1,0.4,0.6);
  glVertex3f(2.1,0.2,0.6);
//****************MIDDLE BODY**********************************
  glVertex3f( 0.6, 0.5,0.6);
  glVertex3f(0.6, 0.2,0.6);
  glVertex3f(1.8, 0.2, 0.6);
  glVertex3f(1.8,0.5,0.6);

  /* bottom of cube*/
  glVertex3f( 0.6,0.2,0.6);
  glVertex3f(0.6,0.2,0.2);
  glVertex3f(1.8,0.2,0.2);
  glVertex3f( 1.8,0.2,0.6);

  /* back of cube.*/
  glVertex3f(0.6,0.5,0.2);
```

```
    glVertex3f(0.6,0.2,0.2);
    glVertex3f(1.8,0.2,0.2);
    glVertex3f(1.8,0.5,0.2);
//*******************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);


//***************************roa              and          surrounding
development********************************
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);
   (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)*3.1
4)/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();
//************************************************************

  glEnable(GL_DEPTH_TEST);
  glutPostRedisplay();
  glutSwapBuffers();
 }
}


/*  The function called whenever a "normal" key is pressed. */
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);          /* 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();
            }
        }

        }


    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();                  /* Start Event Processing Engine. */
   return 1;
   }
```
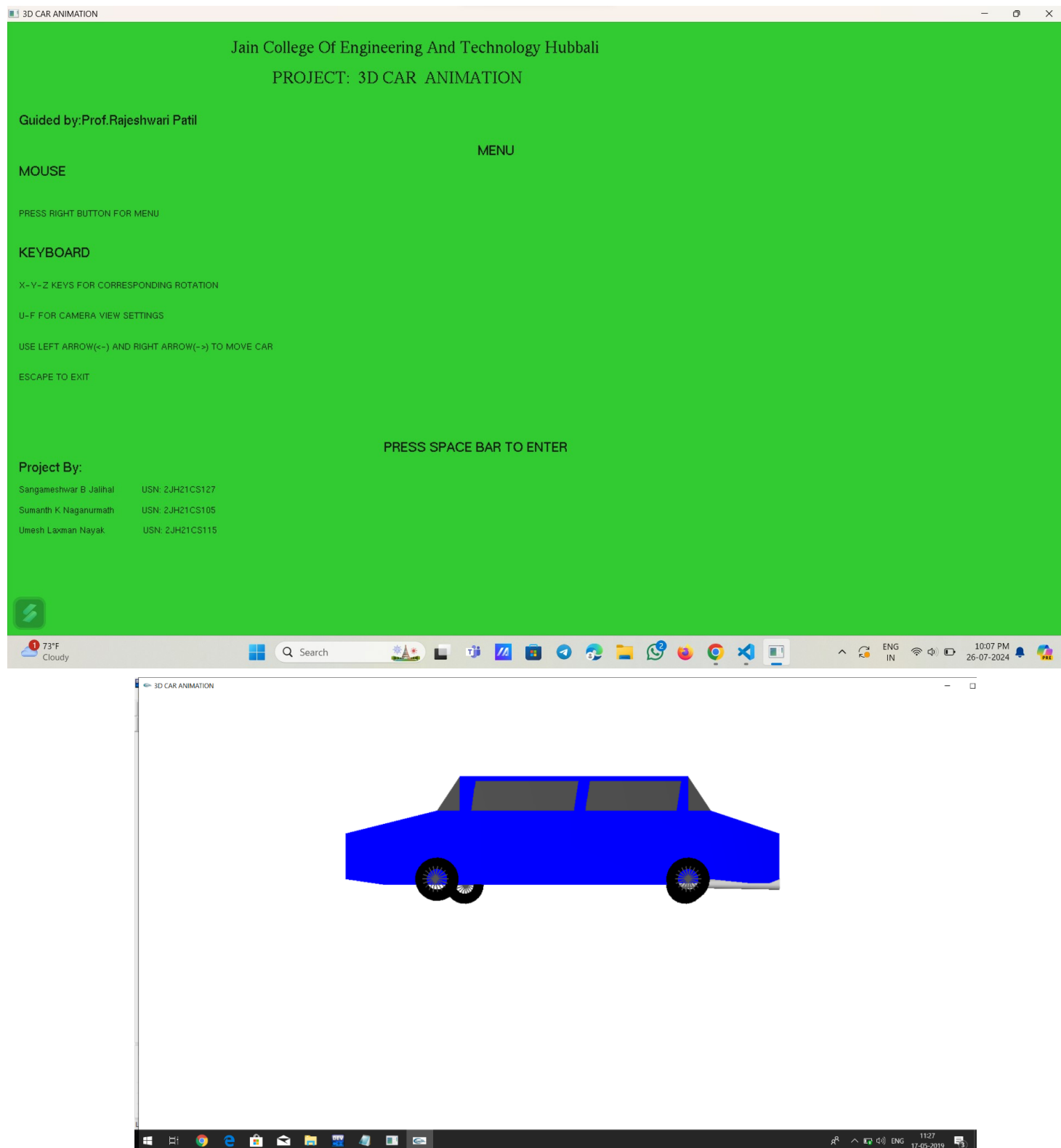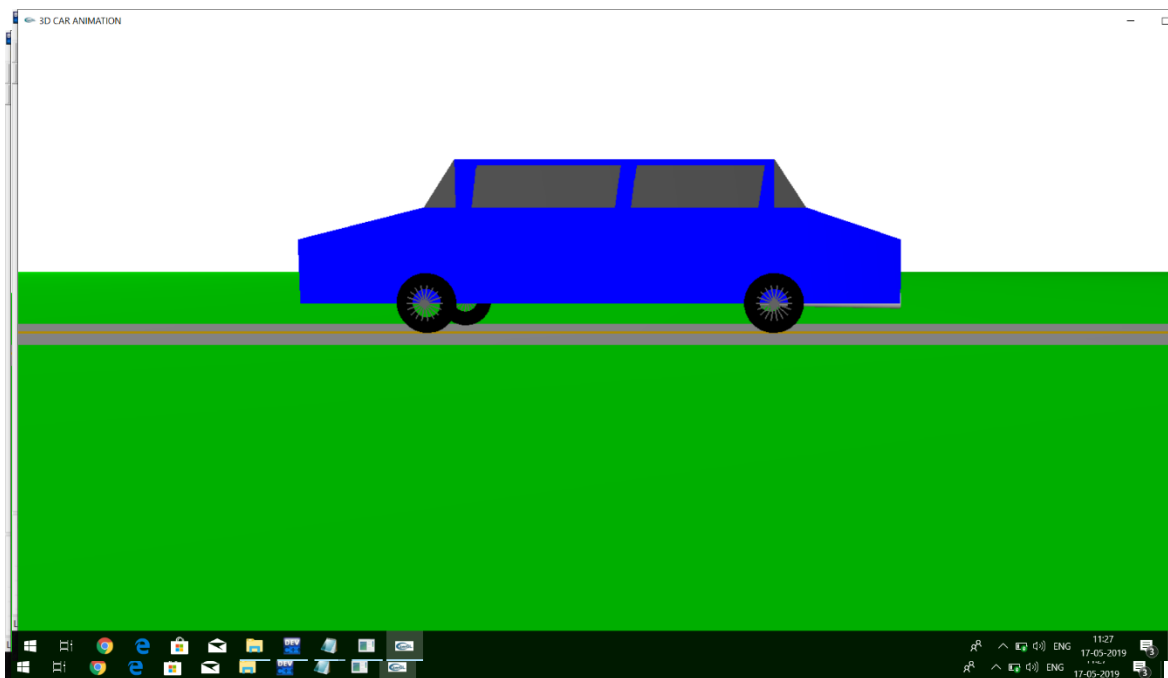
CHAPTER-5

RESULTS

FIG 5.2 *Inside the project*
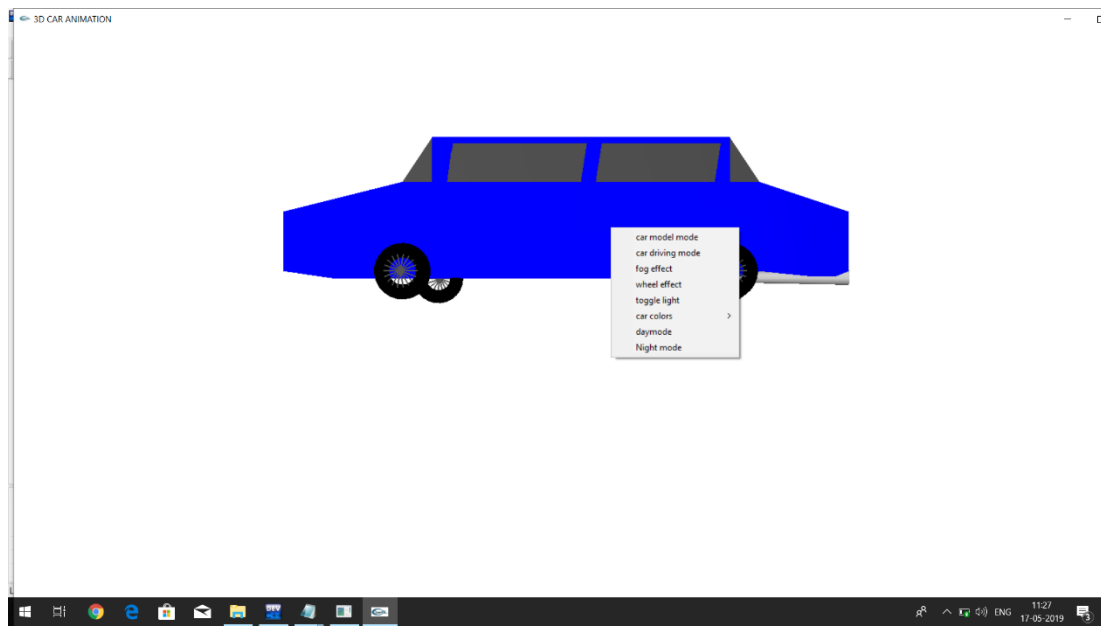
FIG 5.3   *Different car options*
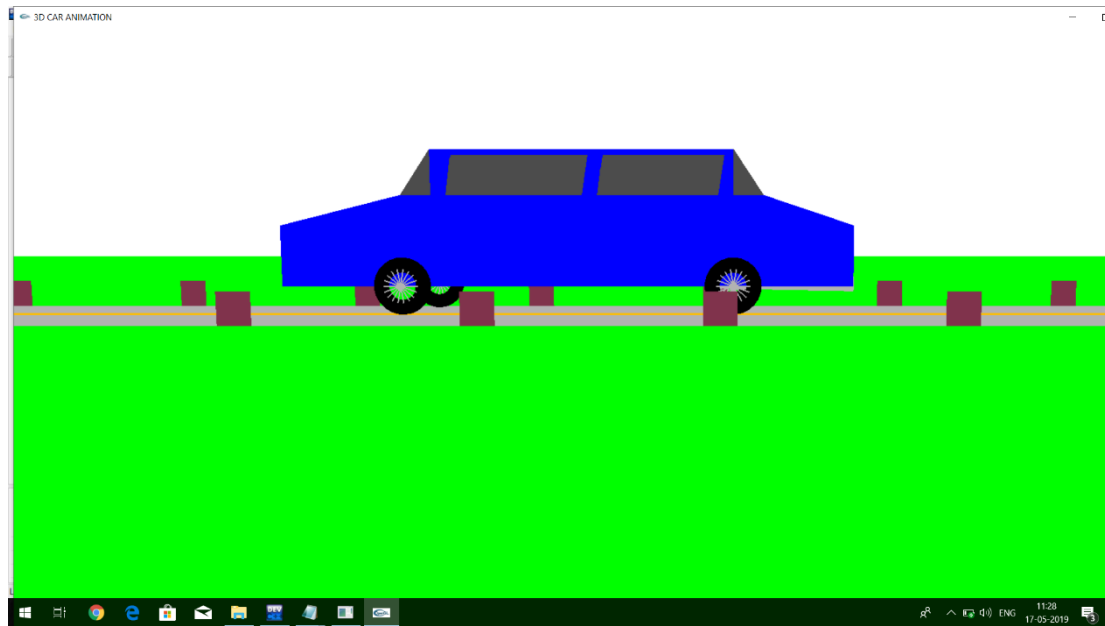

FIG 5.4   *Car Driving Mode*

FIG.5.6.*Wheel Effect*

Fig 5.7 Night mode

CONCLUSION

It has been an interesting journey through the development of this project. At the beginning we used our limited knowledge to implement only the basic features. However, through the months of development, new issues and bugs led to new ideas which led to newer methods of implementation which, in turn, led to us learning even more features of the OpenGL and apply more creative and efficient ways to perform the older functions. New methods helped us in adding flexibility to the various parts of the program, making the further addition of newer features easier and less time consuming which, again, led to the possibility of adding even more features. This sequential chain reaction of progress and ideas has enabled to learn so much through the months of working on this project and we have done our best to add as many features as we could and provide a user interface that is easy and intuitive to use.

Before concluding, it is worth mentioning that this project would never have been possible without the tremendous amount of encouragement by the staff and guides of our department.

We are content with the outcome of this project and are hopeful that it meets the requirements expected and we wish that it may inspire others to be creative and critical in the field of computer graphics and have a newfound appreciation for the Open Graphics Library.

REFERENCES

REFERENCE BOOKS:

[1] Donal hearn & Paulime Baker :Computer Graphics with OpenGL version, 3rd/4th Edition, Pearson Education, 2011.

[2] Edward angel: Interactive computer graphics A TOP-DOWN Approach with OpenGL, 2nd edition, Addison-Wesley, 2000.

[3] F.S. Hill, Jr.: computer graphics using OpenGL, 2nd edition, Pearson education, 2001.

WEB URL'S:

Http://msdn.microsoft.com
http://codeproject.com
http://stackoverflow.com