ABSTRACT

Computer Graphics has grown into a very important topic in the branch of Computer Science. This is due to an effective and rapid communication formed between man and the machine. Human eye can absorb the information in a displayed diagram or perspective diagram much faster than it can scan a page or a table of contents.

This project demonstrates the creation of a moving racing car along with a circular race track and scenery. The user is also given the option to see the car from different views. OpenGL is used to make this possible by virtue of its various functionalities.

We make use of simple geometric figures like triangle, circle and polygons to construct the parts of racing car and the track Cones, spheres and cylinders are used to generate the trees. We also include lighting and material properties.

The code implemented makes use of various OpenGL functions for translation, rotation and keyboard callback function, built-in functions for solids and many more.

The concepts of computer graphics and OpenGL stand a backbone to achieve the aforementioned idea. Primitive drawing, event driven interactions and basic animation have been the important concepts brought out by this application.

The report is chalked out into sections describing the basic requirements superseded by the briefing on functions used. Following this, the detailed description of how the implementation is done effectively using these functions and C language is presented. The source code is provided along with necessary comments to enhance readability of code.

CONTENTS

CHAPTER 1

INTRODUCTION

Introduction to OpenGL

OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics.

The interface consists of function calls, which can be used to draw complex two-dimensional and three-dimensional scenes from simple primitives. The header file <GL/glut.h> is included in order to utilize the inbuilt functions needed in the implementation. C programming language is used for the project coding.

OpenGL serves two main purposes:

- To hide the complexities of interfacing with different 3D accelerators, by presenting the programmer with a single, uniform API.
- To hide the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set.

OpenGL's basic operation is to accept primitives such as points, lines and polygons, and convert them into pixels.

This is done by a graphics pipeline known as OpenGL state machine. Most OpenGL commands either issue primitives to the graphics pipeline, or configure how the pipeline processes these primitives.

Basic Functionalities in OpenGL include:

- Rasterised points, lines and polygons as basic primitives
- Transform and lighting pipeline
- Z-buffering
- Texture mapping
- Alpha Blending

OPENGL(OPEN GRAPHICS LIBRARY)

Most of our applications will be designed to access openGL directly through functions in three libraries. Function in the GL library have name that begin with letter gl and stored in the library.
The second is the openGL utility Library (GLU).
Library uses only GL function but contains codes for creating common object and viewing.
Rather than using a different library for each system we used available library called openGL utility toolkit (GLUT). It is used as #include<glut.h>.

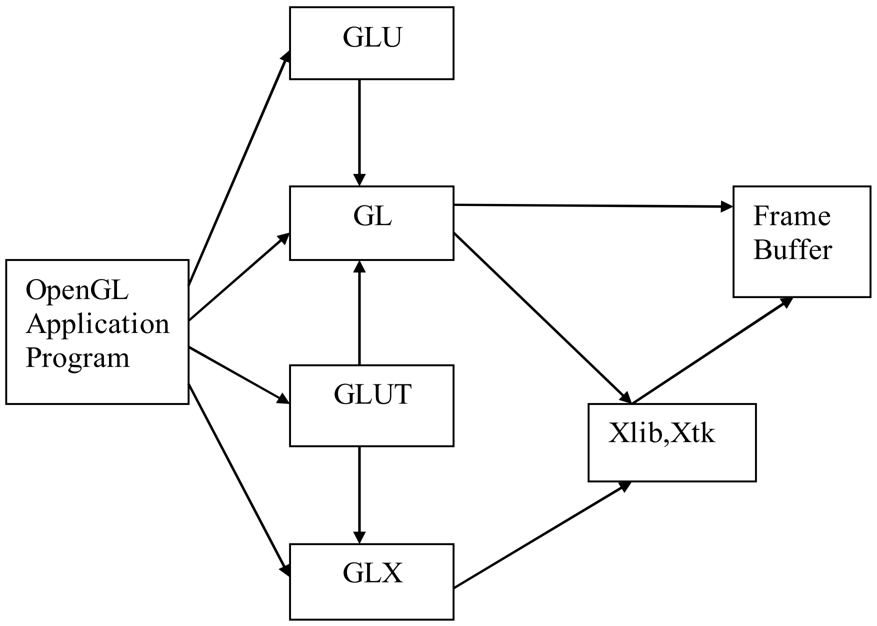


Fig: Library Organisation.

CHAPTER 2 LITERATURE SURVEY

The program begins with all necessary header files and by making global declarations of vertices of quadrilaterals used to generate the car, co ordinates of the initial position of the car. We also define constants required to make conversions from radians to degrees and vice-versa. An enumerated data type is created to define a variable holding the current view of the car.

LIBRARY FUNCTIONS:

glutInitDisplayMode()

Used to define the display mode. GLUT_RGB specifies *RGB* colour mode. GLUT_DOUBLE specifies a double buffered window. GLUT_DEPTH is used for hidden surface removal using depth buffer.

glutInitWindowPosition() specifies the screen location for the upper-left corner of the window.

glutInitWindowSize() specifies the size, in pixels, of the window.

glutCreateWindow() creates a window with an OpenGL context. It returns a unique identifier for the new window.

glutDisplayFunc() registers the display call-back.

glutKeyboardFunc() registers keyboard call-back.

glutIdleFunc() registers the idle call-back.

glutReshapeFunc() indicates what action should be taken when the window is resized.The name of function to be called when window is resized is passed as parameter. glutMainLoop() gets the application in a never ending loop, always waiting for next event to process.

glBegin() initiates a new primitive and begins collection of vertices.

glEnd() terminates a list of vertices.

glVertex3f() specifies the position of a vertex in 3 dimensions.

glClearColor() sets the present RGBA clear color used when clearing the color buffer.

glClear() indicates that the appropriate buffer is to be cleared for redisplaying purpose. GL_COLOR_BUFFER_BIT specifies color buffer has to be cleared. GL_DEPTH_BUFFER_BIT specifies depth buffer has to be cleared.

glMatrixMode() specifies which matrix will be affected by subsequent transformations. Mode can be GL_PROJECTION or GL_MODELVIEW.

glLoadIdentity() sets the current transformation matrix to the identity matrix.

gluPerspective() defines a perspective viewing volume using y direction field of view measured in degrees, aspect ratio of front clipping plane, and near and far distances.

gluLookAt() post multiplies the current matrix by a matrix determined by a viewer at eye point looking at the at point with a specified up direction.

glSwapBuffers() swaps front and back buffers.

glColor3f() sets the present RGB colors.

glViewport() specifies a width x height viewport in pixels .

glutPostRedisplay() requests that the display callback be executed after the current callback returns.

glPushMatrix() and glPopMatrix() pushes to and pops from the matrix stack corresponding to the current matrix mode.

glRotatef() alters the current matrix by a rotation of specified angle about the specified axis.

glTranslatef() alters the current matrix by a specified displacement.

glLightf() sets light properties.

glMaterialf() sets material properties.

glutSolidSphere() generates a solid sphere.

USER DEFINED FUNCTIONS:

main()

The execution first starts with the main() function.

It's an entry point for GLUT OpenGL application.

init()

This function is defined to initialize window parameters.

display() is the display call back function.

keys() is the keyboard call back function.

idle() is idle call back function

tri() is used to draw triangles.

cylinder() is used to draw cylinders.

circle() generates circles.

cone() generates a cone.

rect() is used to draw quadrilaterals.

chassis() generates the chassis of the car.

wheels() generates the car wheels.

alloy() generate the sides of the tyres.

actall() generates the spokes of the wheel.

car() generates the car along with chassis and the wheels.

driver() generates the car driver using cylinders and a sphere.

track() generates the track.

tree() generates trees with cone shaped tree tops.

tree2() generates trees with sphere shaped tree tops.

scenery() generates the scenery .

reshape() is function called when window's size is changed.

view() is function that generates the scene from a view chosen by the user.

CHAPTER 3 REQUIREMENT SPECIFICATIONS

The requirements can be broken down into 2 major categories namely hardware and software requirements. The former specifies the minimal hardware facilities expected in a system in which the project has to be run. The latter specifies the essential software needed to build and run the project.

HARDWARE REQUIREMENTS:

The Hardware requirements are very minimal and the program can be run on most of the machines.

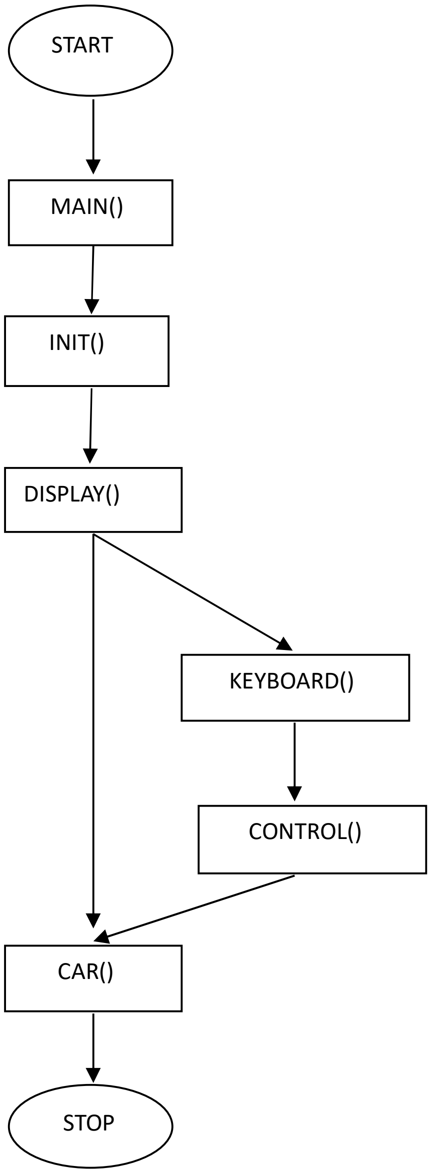
• Processor	- Intel 486/Pentium processor or better
• Processor Speed	- 500 MHz or above
• Hard Disk	- 20GB(approx)
• RAM	- 64MB or above
• Storage Space	- Approx. 2MB

SOFTWARE REQUIREMENTS:

1. Operating System	- UBUNTU	3. Development Tool	-Eclipse
2. Language	-C/C++	4. Graphics Libraries	-OpenGL

CHAPTER 4 ANALYSIS AND ALGORITHM

FLOWCHART:



ALGORITHM DESIGN:

The chassis of the car has been designed using polygons like triangles and quadrilaterals. The car is generated in the X-Y plane with it's height along the positive Z axis. The wheels of the car along with the axle are generated using cylinders. The rim and the spokes are designed using circles and user defined primitive respectively. The driver is designed using cylinders for body and limbs and sphere for the head.

There two types of trees generated in this project. Trees with cone shaped and sphere shaped tree tops. They are generated using cylinders, cones and spheres. A hollow cylinder is used as a background resembling the sky. A circle is used as the ground.

We achieve the motion of the car using translation and rotation. User is given the option to start and stop the car using keyboard control.

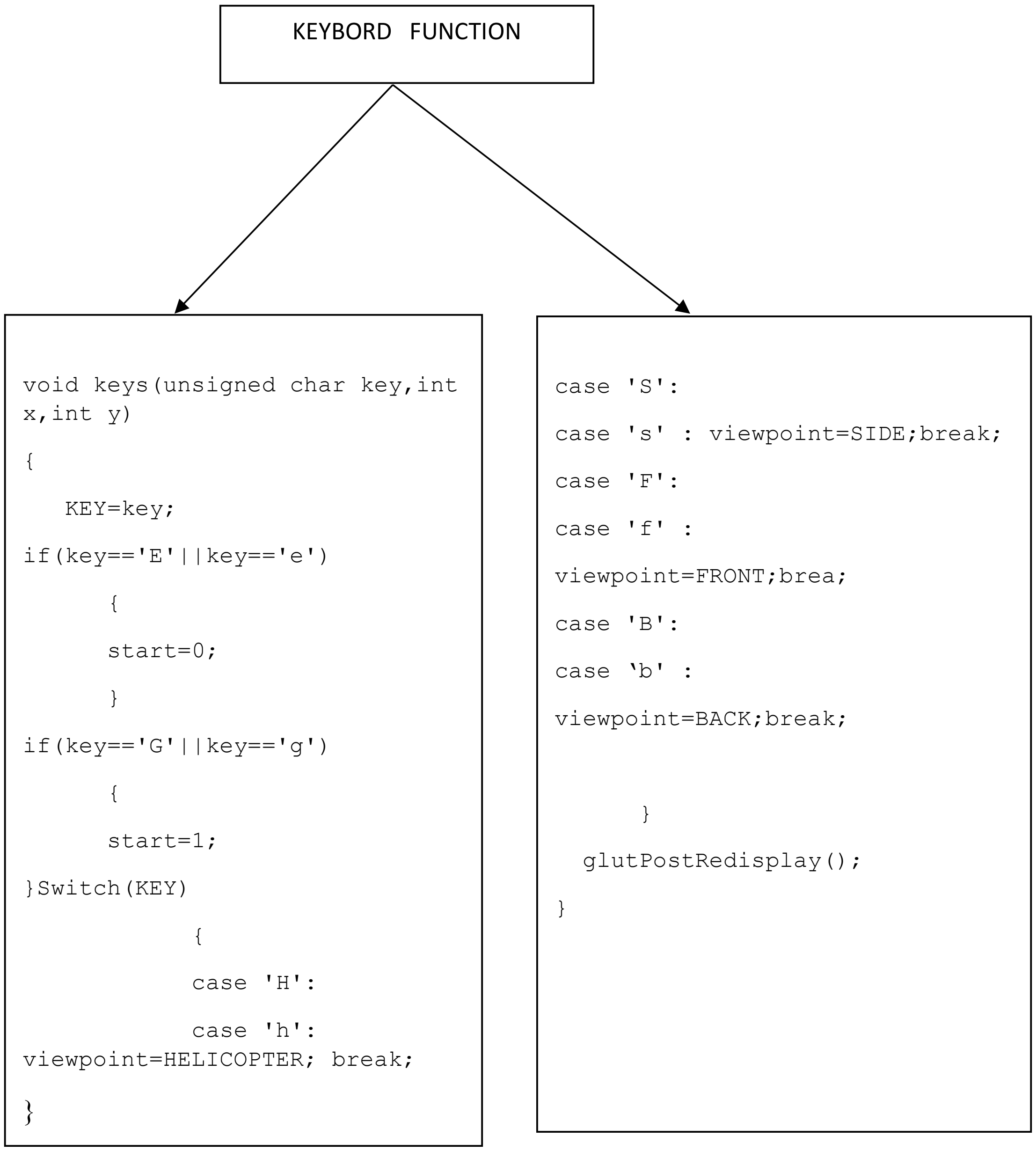
The car can be seen from four views when in motion:

- HELICOPTER VIEW
- FRONT VIEW
- SIDE VIEW
- BACK VIEW

The user can select the view using the keyboard control.
To make the scene realistic lighting has been used.

CHAPTER 5 IMPLEMENTATION

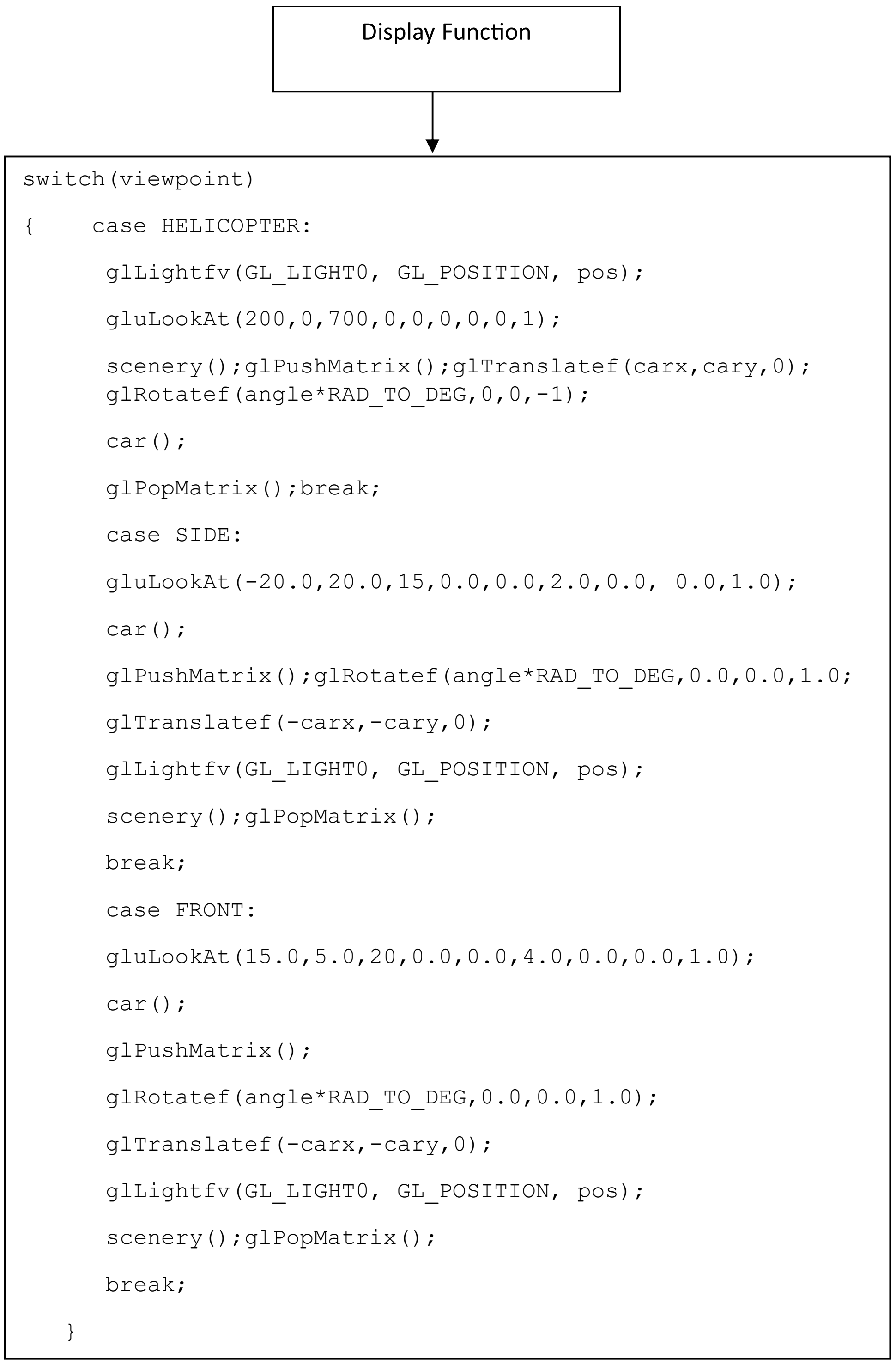
KEYBOARD FUNCTION:



```
void keys(unsigned char key,int
x,int y)

{

   KEY=key;

if(key=='E'||key=='e')

      {

      start=0;

      }

if(key=='G'||key=='g')

      {

      start=1;

}Switch(KEY)

            {

            case 'H':

            case 'h':
viewpoint=HELICOPTER; break;

}
```

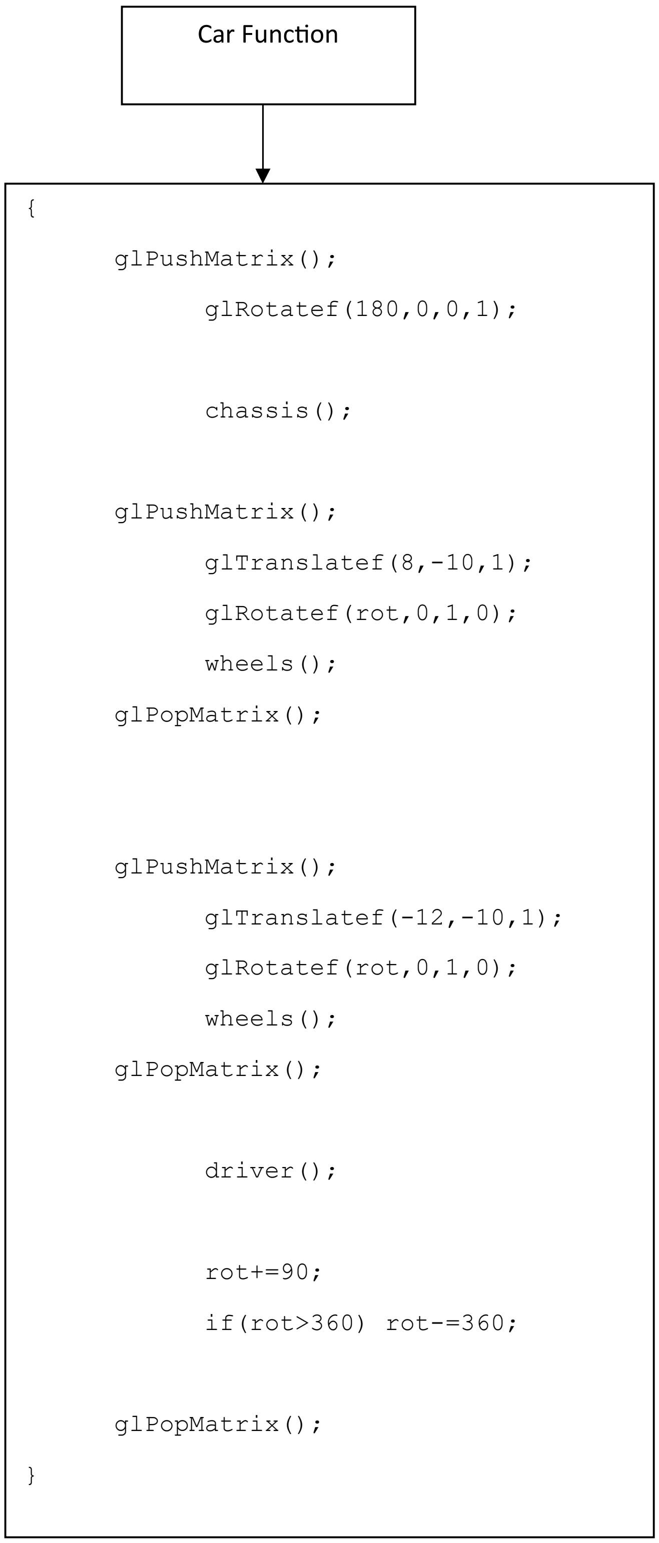
```
case 'S':

case 's' : viewpoint=SIDE;break;

case 'F':

case 'f' :

viewpoint=FRONT;brea;

case 'B':

case 'b' :

viewpoint=BACK;break;


      }

  glutPostRedisplay();

}
```

DISPLAY FUNCTION:



```
switch(viewpoint)
{    case HELICOPTER:
      glLightfv(GL_LIGHT0, GL_POSITION, pos);
      gluLookAt(200,0,700,0,0,0,0,0,1);
      scenery();glPushMatrix();glTranslatef(carx,cary,0);
      glRotatef(angle*RAD_TO_DEG,0,0,-1);
      car();
      glPopMatrix();break;
      case SIDE:
      gluLookAt(-20.0,20.0,15,0.0,0.0,2.0,0.0, 0.0,1.0);
      car();
      glPushMatrix();glRotatef(angle*RAD_TO_DEG,0.0,0.0,1.0;
      glTranslatef(-carx,-cary,0);
      glLightfv(GL_LIGHT0, GL_POSITION, pos);
      scenery();glPopMatrix();
      break;
      case FRONT:
      gluLookAt(15.0,5.0,20,0.0,0.0,4.0,0.0,0.0,1.0);
      car();
      glPushMatrix();
      glRotatef(angle*RAD_TO_DEG,0.0,0.0,1.0);
      glTranslatef(-carx,-cary,0);
      glLightfv(GL_LIGHT0, GL_POSITION, pos);
      scenery();glPopMatrix();
      break;
   }
```

CAR FUNCTION:



```
{
      glPushMatrix();
            glRotatef(180,0,0,1);

            chassis();

      glPushMatrix();
            glTranslatef(8,-10,1);
            glRotatef(rot,0,1,0);
            wheels();
      glPopMatrix();



      glPushMatrix();
            glTranslatef(-12,-10,1);
            glRotatef(rot,0,1,0);
            wheels();
      glPopMatrix();

            driver();

            rot+=90;
            if(rot>360) rot-=360;

      glPopMatrix();
}
```

CHAPTER 6

DISCUSSIONS & SNAP SHOTS

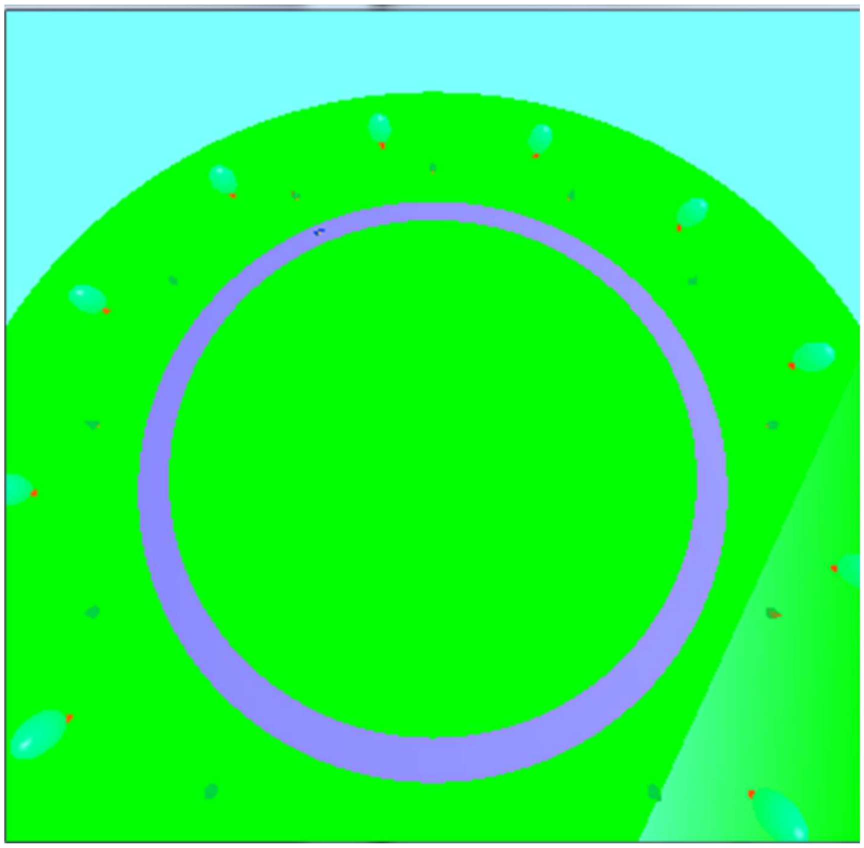


Fig 1: HELICOPTER VIEW

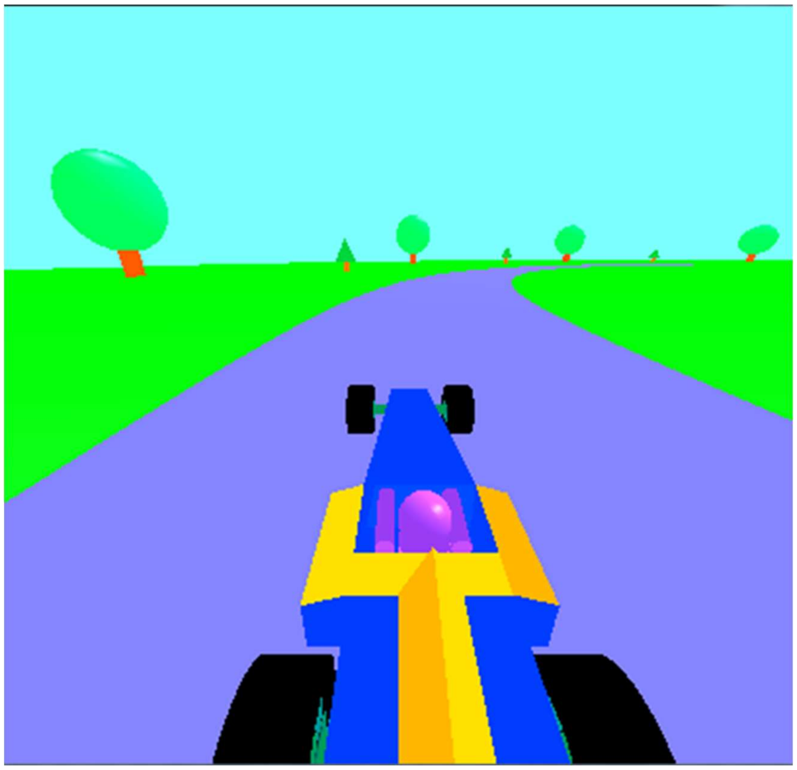


FIG 2: BACK VIEW

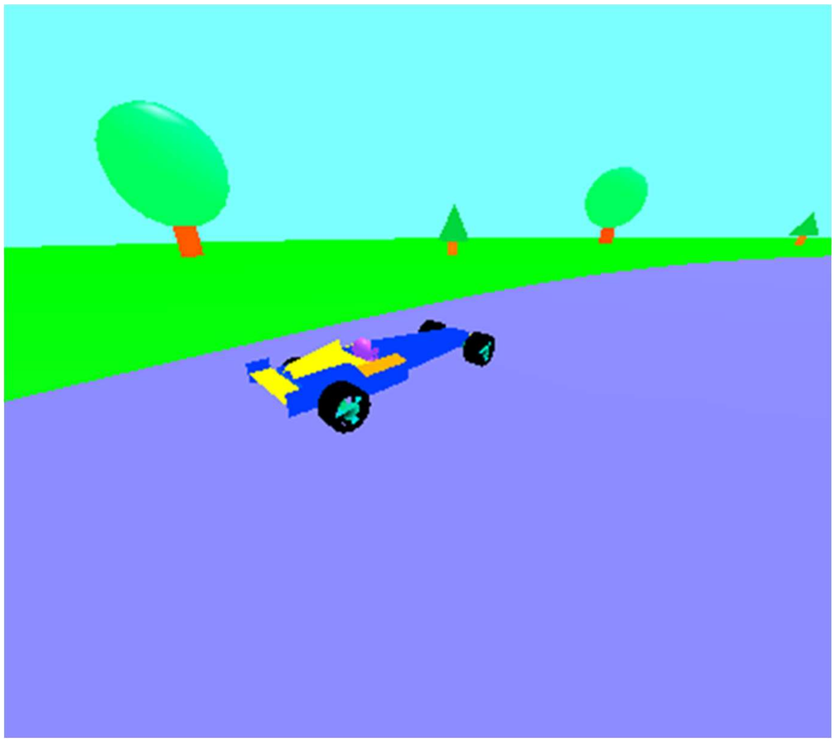


FIG 3: SIDE VIEW

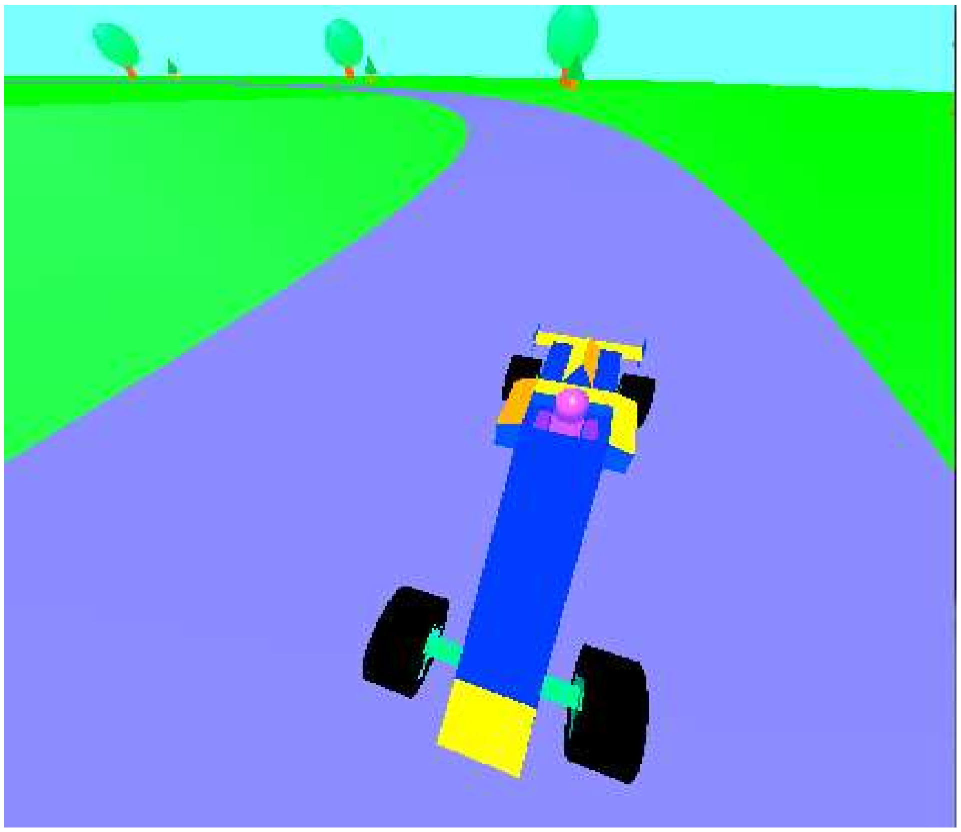


Fig 4: FRONT VIEW

CHAPTER 7

CONCLUSION & FUTURE SCOPE

This project has been a successful learning experience to practically use the functions defined in OpenGL and to understand a variety of features and options present in it. This project clearly demonstrated the richness of graphics library and its ability to make complex 3D images in a very simple way. We intended to make maximum use of the OpenGL functions which we have done to our satisfaction.

This project showed us a moving car with different views. The inclusion of textures, complex shading and lighting models can make the planes more clear and colourful. The project can be developed into a game by allowing the user to move the car using direction keys. More number of cars can be added to get a real feeling of a race. This project is also helpful for students of computer science to study the OpenGL functions in a better way.

CHAPTER 8

CODE

```
#include<GL/glut.h>

#include<math.h>

#include<stdio.h>


#define c 3.14/180

#define PI  3.14

#define TWO_PI  2.0 * PI

#define RAD_TO_DEG  180.0 / PI


//Coordinates for the chassis of the car


float p[]={5.5,-2.5,1},q[]={5.5,-7.5,1},r[]={10.7,-7.5,1},s[]={10.7,-2.5,1};


float p1[]={10.7,-9,3},s1[]={12.7,-9,3},q1[]={10.7,-1,3},r1[]={12.7,-1,3};


float p2[]={0.5,-1,1},s2[]={5.5,-1,1},q2[]={0.5,-9,1},r2[]={5.5,-9,1};


float p3[]={-15,-6.5,1},q3[]={-15,-3.5,1},r3[]={0.5,-2.5,1},s3[]={0.5,-7.5,1};


float p4[]={-13,-6.5,1},q4[]={-13,-6.5,2.5},r4[]={0.5,-7.5,3.5},s4[]={0.5,-7.5,1};


float p5[]={-13,-3.5,1},q5[]={-13,-3.5,2.5},r5[]={0.5,-2.5,3.5},s5[]={0.5,-2.5,1};


float p6[]={5.5,-2.5,1},q6[]={5.5,-2.5,3.5},r6[]={10.7,-2.5,3.5},s6[]={10.7,-2.5,1};


float p7[]={5.5,-7.5,1},q7[]={5.5,-7.5,3.5},r7[]={10.7,-7.5,3.5},s7[]={10.7,-7.5,1};


float p8[]={5.5,-7.5,3.5},q8[]={10.7,-7.5,3.5},r8[]={10.7,-6,3.5},s8[]={5.5,-6,3.5};
float p9[]={5.5,-2.5,3.5},q9[]={5.5,-4,3.5},r9[]={10.7,-4,3.5},s9[]={10.7,-2.5,3.5};

float p10[]={5.5,-4,3.5},q10[]={10.7,-4,3.5},r10[]={10.7,-5,4.5},s10[]={5.5,-5,5.5};

float p11[]={5.5,-6,3.5},q11[]={10.7,-6,3.5},r11[]={10.7,-5,4.5},s11[]={5.5,-5,5.5};

float p12[]={10.7,-9,2},q12[]={10.7,-9,4},r12[]={12.7,-9,4},s12[]={12.7,-9,2};

float p13[]={10.7,-1,2},q13[]={10.7,-1,4},r13[]={12.7,-1,4},s13[]={12.7,-1,2};

float p14[]={0.5,-1,1},q14[]={0.5,-1,3},r14[]={5.5,-1,3},s14[]={5.5,-1,1};

float p15[]={0.5,-9,1},q15[]={0.5,-9,3},r15[]={5.5,-9,3},s15[]={5.5,-9,1};

float p16[]={0.5,-1,1},q16[]={0.5,-1,3},r16[]={0.5,-2.5,3.5},s16[]={0.5,-2.5,1};

float p17[]={0.5,-7.5,1},q17[]={0.5,-7.5,3.5},r17[]={0.5,-9,3},s17[]={0.5,-9,1};

float p18[]={5.5,-1,1},q18[]={5.5,-1,3},r18[]={5.5,-2.5,3.5},s18[]={5.5,-2.5,1};

float p19[]={5.5,-7.5,1},q19[]={5.5,-7.5,3.5},r19[]={5.5,-9,3},s19[]={5.5,-9,1};

float p20[]={10.7,-7.5,1},q20[]={10.7,-7.5,3.5},r20[]={10.7,-2.5,3.5},
            s20[]={10.7,-2.5,1};

float p21[]={4,-2.5,3.5},q21[]={5.5,-2.5,3.5},r21[]={5.5,-7.5,3.5},s21[]={4,-7.5,3.5};

enum
{       // Constants for different views
        HELICOPTER,FRONT,SIDE,BACK
} viewpoint = HELICOPTER;
```

```
int MID=570; //Distance of the car on the track from the centre of the track
int start=0;

char KEY; //Variable that stores key pressed by user

float angle; //Rotation angle for car

float carx=0,cary=570; //Variables that specify position of the car

int rot=0; //rotation angle for the wheels

//Function to generate a cone
void cone()
{
    float i,x,y,r=10;

    glColor3f(0.0,0.7,0.2);
    glBegin(GL_TRIANGLE_FAN);
    glVertex3f(0,0,20);
    for(i=0;i<=361;i+=2)
    {
        x= r * cos(i*c);
        y= r * sin(i*c);
        glVertex3f(x,y,0);
    }
    glEnd();
}



//Fuction to draw the track
void track(float R1,float R2)
```

```
{
        float X,Y,Z;
             int  y;
        glBegin(GL_QUAD_STRIP);
        for( y=0;y<=361;y+=1)
        {
                    X=R1*cos(c*y);
                    Y=R1*sin(c*y);
                    Z=-1;
                    glVertex3f(X,Y,Z);

                    X=R2*cos(c*y);
                    Y=R2*sin(c*y);
                    Z=-1;
                    glVertex3f(X,Y,Z);
        }
        glEnd();
}

//Function that generates a cylinder
void cylinder(float r,float l)
{
        float x,y,z; int d;
        glBegin(GL_QUAD_STRIP);
        for( d=0;d<=362;d+=1)
        {
                    x=r*cos(c*d);
                    z=r*sin(c*d);
                    y=0;
                    glVertex3f(x,y,z);

                    y=l;
```

```
                glVertex3f(x,y,z);
        }
        glEnd();
}
//Function that generates tree with cone shaped tree top
void tree(float a,float b)
{       //Tree trunk
        glColor3f(0.9,0.3,0);
        glPushMatrix();
                glTranslatef(a,b,-1);
                glRotatef(90,1,0,0);
                cylinder(3,15);
        glPopMatrix();

        //Cone shaped tree top
        glPushMatrix();
                glTranslatef(a,b,8);
                cone();
        glPopMatrix();

}

//Functin that generates tree with sphere shaped tree top
void tree2(float a,float b)
{
        //Tree trunk
        glColor3f(1,0.2,0);
        glPushMatrix();
                glTranslatef(a,b,-1);
                glRotatef(90,1,0,0);
                cylinder(6,25);
        glPopMatrix();
        for(p=0;p<=360;p+=30)
        {
                x=700*cos(c*p);
                y=700*sin(c*p);
                tree(x,y);
       }



        //Sphere shaped trees
        for( p=100;p<=460;p+=30)
        {
                x=800*cos(c*p);
                y=800*sin(c*p);
                tree2(x,y);
        }

}
//Function to draw triangles
void tri(float a[],float b[],float z[])
{
        glBegin(GL_TRIANGLES);
                glVertex3fv(a);
                glVertex3fv(b);
                glVertex3fv(z);
        glEnd();
}

//Function that has calls to other functions to generate wheels along with axle
void wheels()
```

```
        glPushMatrix();
                glTranslatef(0,10,0);
                actall(1.4,0.8);
                alloy(2,1.4);
                glColor3f(0,0.5,0.4);
                circle(0.8);
        glPopMatrix();

}

//Function that generates the chassis of the car
void chassis()
{
        //Parameters For glMaterialfv() function
        GLfloat specular[] = { 0.7, 0.7, 0.7, 1.0 };
        GLfloat ambient[]={1,1,1,1},diffuse[]={0.7,0.7,0.7,1};
        GLfloat full_shininess[]={100.0};

        //Material Properties
        glMaterialfv(GL_FRONT,GL_AMBIENT,ambient);
        glMaterialfv(GL_FRONT,GL_SPECULAR,specular);
        glMaterialfv(GL_FRONT,GL_DIFFUSE,diffuse);
        glMaterialfv(GL_FRONT,GL_SHININESS, full_shininess);

        glColor3f(0,0.2,0.9);

        rect(p,q,r,s);
        rect(p2,q2,r2,s2);
        rect(p3,q3,r3,s3);
        rect(p4,q4,r4,s4);
        rect(p5,q5,r5,s5);
        rect(q5,q4,r4,r5);
        rect(p6,q6,r6,s6);

        rect(p7,q7,r7,s7);

        rect(p8,q8,r8,s8);

        rect(p9,q9,r9,s9);


        glColor3f(1,0.6,0);


        rect(p1,q1,r1,s1);

        rect(q5,q4,p3,q3);

        tri(p4,q4,p3);

        tri(p5,q5,q3);

        rect(p10,q10,r10,s10);

        rect(p11,q11,r11,s11);

        rect(r16,r18,q18,q16);

        rect(q17,q19,r19,r17);

        rect(p21,q21,r21,s21);


        glColor3f(0,0.2,0.9);


        rect(p12,q12,r12,s12);

        rect(p13,q13,r13,s13);                                          rect(p14,q14,r14,s14);

        rect(p15,q15,r15,s15);

        rect(p16,q16,r16,s16);

        rect(p17,q17,r17,s17);

        rect(p18,q18,r18,s18);

        rect(p19,q19,r19,s19);

        rect(r18,q19,p19,s18);

        rect(p20,q20,r20,s20);

    }


    //Function that that has function calls to chassis(),tyrea(),

    //tyreb(),driver() to generate the car with wheels rotating
```

```
void car()
{
        glPushMatrix();
                glRotatef(180,0,0,1);

                chassis();

                glPushMatrix();
                        glTranslatef(8,-10,1);
                        glRotatef(rot,0,1,0);
                        wheels();
                glPopMatrix();


                glPushMatrix();
                        glTranslatef(-12,-10,1);
                        glRotatef(rot,0,1,0);
                        wheels();
                glPopMatrix();
            glPopMatrix();
}

//Keyboard Callback Function
void keys(unsigned char key,int x,int y)
{
KEY=key;
void init()
{
        GLfloat amb[]={1,1,1,1},diff[]={1,1,1,1},spec[]={1,1,1,1};
        glLoadIdentity();

        glLightfv(GL_LIGHT0, GL_AMBIENT, amb);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, diff);
        glLightfv(GL_LIGHT0, GL_SPECULAR, spec);

        glLightModeli(GL_LIGHT_MODEL_TWO_SIDE,GL_TRUE);

            glEnable(GL_COLOR_MATERIAL);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_DEPTH_TEST);

        glClearColor(1,1,1,1);

}      //Display Callback Function
void display()
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        view();
        glutSwapBuffers();     }
//Reshape Function

void reshape(int w, int h)  {
 glViewport (0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluPerspective(100, (GLfloat) w/(GLfloat) h, 1, 2000.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

```
//Main Fuction


void main(int argc,char **argv)

{

        glutInit(&argc,argv);

        glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);


        printf("\t\t**********RACING CAR IN A RACE TRACK***********\n");

        printf("\n\tPRESS:\n");

        printf("\n\tG or g:To Start or Continue\n");

        printf("\n\tE or e:To Stop\n");

        printf("\n\tH or h :Helicopter View\n");

        printf("\n\tB or b :Back View\n");

        printf("\n\tS or s :Side View\n");

        printf("\n\tF or f :Front View\n");


        glutInitWindowPosition(500,500);

        glutInitWindowSize(500,500);

        glutCreateWindow("Computer Graphics");

        glutDisplayFunc(display);

        glutIdleFunc(idle);

        glutKeyboardFunc(keys);

        glutReshapeFunc(reshape);

        init();

        glutMainLoop();

}
```