VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belgaum-590018

A Computer Graphics & Visualization Mini Project Report
on

"CAR RACING"

Submitted in Partial fulfillment of the Requirements for VI Semester of the Degree of

Bachelor of Engineering
In
Computer Science & Engineering
By

SHAILAV SHRESTHA
(1CR16CS154)

RAJENDRA GUPTA
(1CR16CS127)

Under the Guidance of

Mr. SHIVARAJ V B
Asst. Professor, Dept. of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CMR INSTITUTE OF TECHNOLOGY

#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI,

BANGALORE-560037

CMR INSTITUTE OF TECHNOLOGY

#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI,

BANGALORE-560037

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Computer Graphics & visualization project work entitled **"Car Racing"** has been carried out by **Shailav Shrestha (1CR16CS154)** and **Rajendra Gupta (1CR16CS127)** bonafide students of CMR Institute of Technology in partial fulfillment for the award of **Bachelor of Engineering** in **Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year **2018-2019**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. This CG project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

-----------------------	**----------------------**
Signature of Guide	**Signature of HOD**
Mr. Shivaraj V B	**Dr. Jhansi Rani P**
Asst. Professor	**Professor & Head**
Dept. of CSE, CMRIT	**Dept. of CSE, CMRIT**

External Viva

Name of the examiners	Signature with date
1.	
2.	

ABSTRACT

OpenGL provides a set of commands to render a three dimensional scene. That means you provide them in an Open GL-useable form and Open GL will show this data on the screen (render it). It is developed by many companies and it is free to use. You can develop Open GLapplications without licensing.

Open GL is a hardware- and system dependent interface. An Open GL-application will work on every platform, as long as there is an installed implementation, because it is system independent, there are no functions to create windows etc., but there are helper functions for each platform. A very useful thing is GLUT.

This project includes the concepts of transformation, motion in objects. I have tried to create a simple game called “CAR RACING”.

Car Racing game, its basic concept is to race around a circular path. The view of the car can be adjusted with the provided keystrokes .The main goal of our program is to simulate a car as it moves through a racing track. Simulation is an immensely helpful approach in Engineering design. It allows us to make better designs and asses how situations are handled by our product.

ACKNOWLEDGEMENT

Behind every success there is a master hand. A master hand will create unperturbed concentration, dedication and encouragement in everything good and bad, without whose blessing this would have never come into existence.

Firstly, I thank God for showering the blessings on me. I am grateful to my institution CMRIT for providing me a congenial atmosphere to carry out the project successfully.

I would like to express my heartfelt gratitude to **Dr. Sanjay Jain,** Principal, CMRIT, Bangalore, for extending his support.

I am highly thankful to **Dr. Jhansi Rani,** HOD of Computer Science and Engineering, CMRIT, Bangalore for her support and encouragement given to carry out the project.

I am very grateful to my guide, **Mr. Shivaraj V B & Mr. Kartheek G C R,** Assistant Professor, Department of Computer Science, for his able guidance and valuable advice at early stage of my project which helped me in successful completion of my project.

Finally, I would like to thank my parents and friends who helped me with the content of this report, without which the project would not have become a reality.

SHAILAV SHRESTHA (1CR16CS154)
RAJENDRA GUPTA (1CR16CS127)

LIST OF FIGURES

CONTENTS

REFERENCES

[1] Edward Angel, “Interactive Computer Graphics”, 5th edition, Pearson Education, 2005

[2] **“**Computer Graphics”, Addison-Wesley 1997 James D Foley, Andries Van Dam, Steven K Feiner, John F Hughes.

[3] F.S.Hill and Stephen M.Kelly, “Computer Graphics using OpenGl “, 3rd edition

[4] https://www.opengl.org/documentation/

[5] http:// www.openglprojects.in

[6] http://www.openglprogramming.com/

[7] http://www.google.com/

CHAPTER 1

INTRODUCTION

1.1 Computer Graphics

- Computer graphics is the study of manipulation of visual and geometric information using computational techniques. It focuses on the mathematical and computational foundations of image generation and processing rather than purely aesthetic issues.

- Computer Graphics is a subfield of Computer Science which studies methods for digitally synthesizing and manipulating visual content.

- Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D or 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly.

- Computer graphics is often differentiated from the field of visualization although the two fields have many similarities. The field not only encompasses the real world objects but also of abstract objects such as mathematical surfaces on 4D and modelling of data that have no inherent geometry.

- ++The field of computer graphics can be classified into following sub fields.
    - Geometry : study ways to represent and process surfaces
    - Animation : study ways to represent and manipulate motion
    - Rendering : studies algorithms to reproduce light transport
    - Imaging : studies image acquisition or image editing
    - Topology : studies the behaviour of spaces and surfaces.

- A computer graphics system is a computer system with all the components of the general purpose computer system. There are five major elements in system: input devices, processor, memory, frame buffer, output devices.

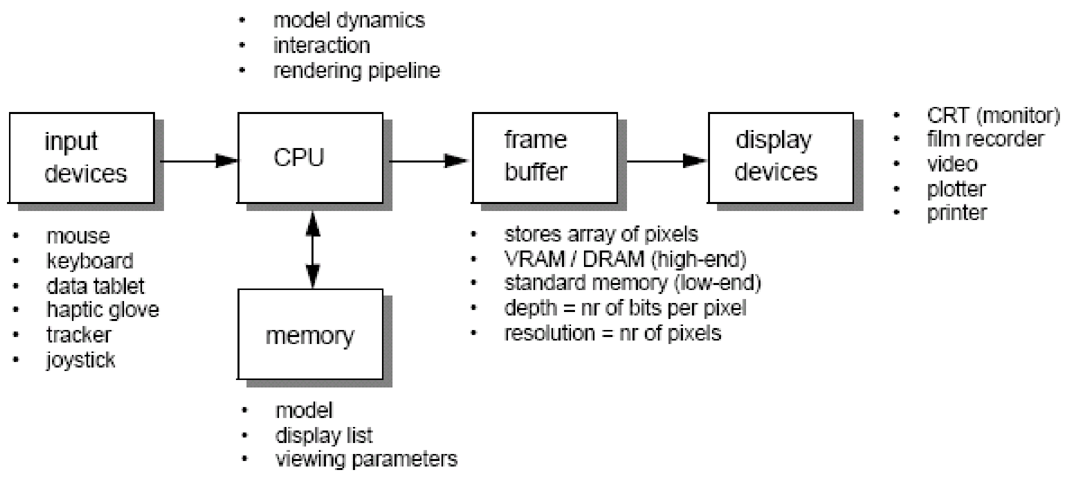


Fig 1.1: Graphic System

1.2 Areas of Application of Computer Graphics

Modern Society needs no exhortation on the relevance of Computer Graphics. It’s application ranges both wide and deep.

As Engineering Students, we understand the importance of simulation and modelling of information first hand. Modelling and Simulation of system with graphical tools not only gives us solid understanding of the working of the system but also gives insights to innovation and design flaws.s the premier environment for developing portable, interactive 2D and 3D graphics applications

Some fields that heavily apply computer graphics are :

- Computational Biology
- Computational Physics
- Information Graphics
- Scientific Visualization
- Graphic Design
- Computer-aided Design
- Web Design
- Digital Art
- Video Games
- Virtual Reality
- Computer Simulation
- Education
- Information Visualization
- Cartography

1.3 OpenGL

1.3.1 Introduction to OpenGL

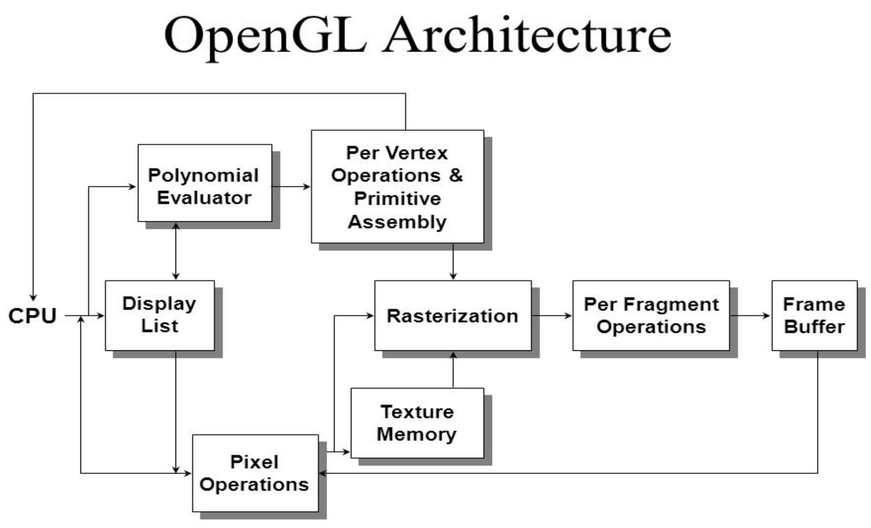
- **OpenGL** a cross-language, cross-platform library for rendering 2D/3D vector graphics.

- **OpenGL** is the industry's most widely used, supported and best documented 2D/3D graphics API making it inexpensive & easy to obtain information on implementing OpenGL in hardware and software.

- The **API** is typically used to interact with a graphics processing unit (GPU), to achieve hardware accelerated rendering.

- 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 (which define geometric objects) or pixels (which define images). OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

1.3.2 OpenGL Architecture and Organization



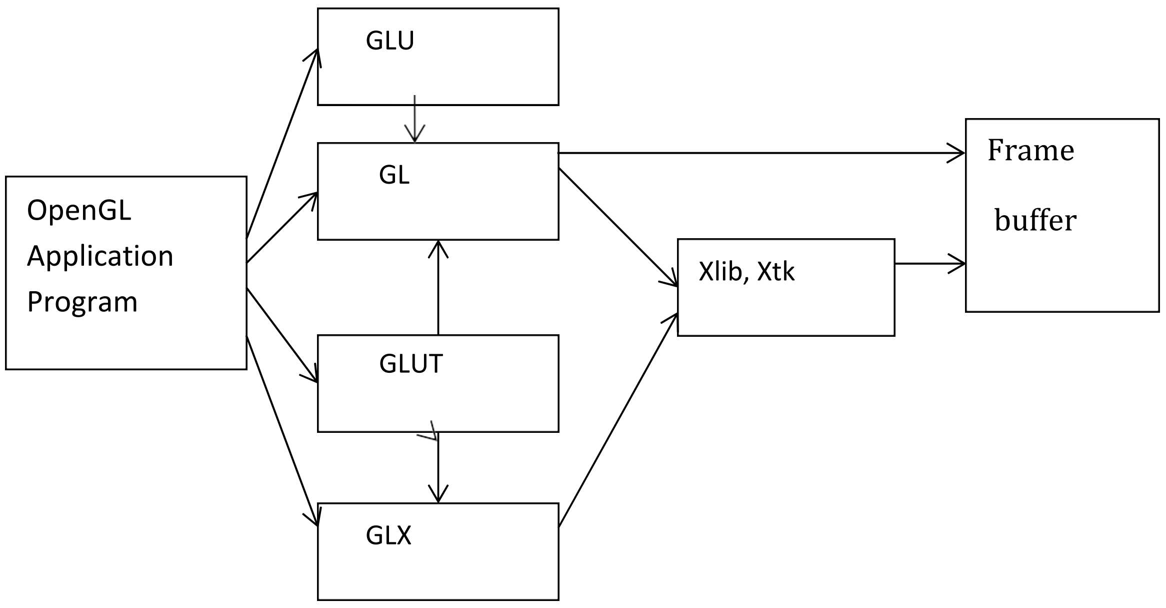


Fig 1.2: Library organization of OpenGL

The OpenGl Library is organized as follows :

GL	: core graphics capability
GLU	: utilities on top of GL
GLUT	: input and windowing functions

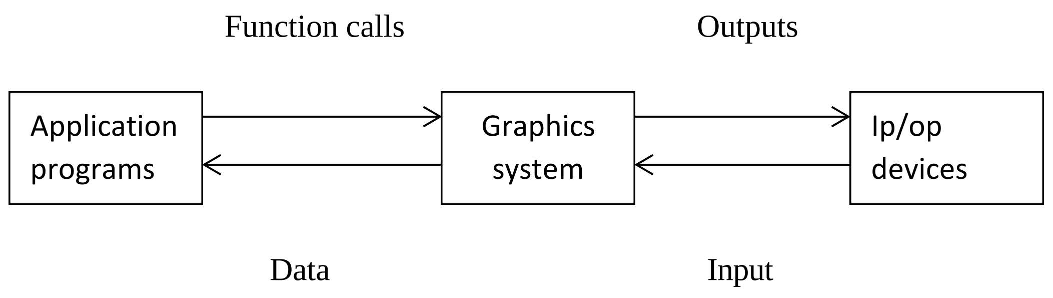


Fig 1.3: Graphics system as a black box.

Our basic model of a graphics package is a black box, a term that engineers use to denote a system whose properties are described only by its inputs and outputs. We describe an API through the functions in its library.

1.3.3 Fundamental Concepts in OpenGL

Some of the concepts inherent in OpenGL are :

- **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 (although many modes may interact to determine what eventually ends up in the frame buffer). Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls. Commands are always processed in the order in which they are received, although there may be an indeterminate delay before a command takes effect.

- **Procedural versus Descriptive**

  OpenGL provides you with fairly direct control over the fundamental operations of two- and three-dimensional graphics. This includes specification of such parameters as transformation matrices, lighting equation coefficients, antialiasing methods, and pixel update operators. However, it doesn't provide you with a means for describing or modeling complex geometric objects. OpenGL is fundamentally procedural rather than descriptive. Because of this procedural nature, it helps to know how OpenGL works—the order in which it carries out its operations, for example—in order to fully understand how to use it

- **Execution Model**

  The model for interpretation of OpenGL commands is client-server. An application (the client) issues commands, which are interpreted and processed by OpenGL (the server). The server may or may not operate on the same computer as the client. In this sense, OpenGL is network-transparent. A server can maintain several GL *contexts*, each of which is an encapsulated GL state. A client can connect to any one of these contexts. The effects of OpenGL commands on the frame buffer are ultimately controlled by the window system that allocates frame buffer resources.

CHAPTER 2

REQUIREMENTS SPECIFICATION

2.1 Purpose of the requirements document

The software requirement specification is the official statement of what is required for development of particular project. It includes both user requirements and system requirements. This requirement document is utilized by variety of users starting from project manager who gives project to the engineer responsible for development of project.

It should give details of how to maintain, test, verify and what all the actions to be carried out through life cycle of project.

2.1.1 Scope of the project

The scope is to use the basic primitives defined in openGL library creating complex objects. We make use of different concepts such as glColor() , gluOrtho2D(), timer function.

2.1.2 Definition

The project **RACING CAR** is created to demonstrate OpenGL's concepts. It encompasses some of the skills learnt in our OpenGL classes such as glColor() , gluOrtho2D(), timer function.

.

2.1.3 Acronyms & Abbreviations

OpenGL provides a powerful but primitive set of rendering command, and all higher-level design must be done in terms of these commands.

OpenGL Utility Toolkit(GLUT): -windows-system-independent toolkit.

2.1.4 References

OpenGL tutorials

Interactive Computer Graphics (Edward Angel)

2.2 Specific requirements

2.2.1 User Requirement:

- Easy to understand and should be simple.
- The built-in functions should be utilized to maximum extent.
- OpenGL library facilities should be used.

2.2.2 Software Requirements:

- Platform : UNIX
- Distribution : Ubuntu
- Packages : OpenGL Library, Editor
- Language : C
- Compiler : gcc

2.2.3 Hardware Requirements:

- Processor-Intel or AMD(Advanced Micro Devices)
- RAM-512MB(minimum)
- Hard Disk-1MB(minimum)
- Mouse
- Keyboard
- Monitor

CHAPTER 3

DESIGN

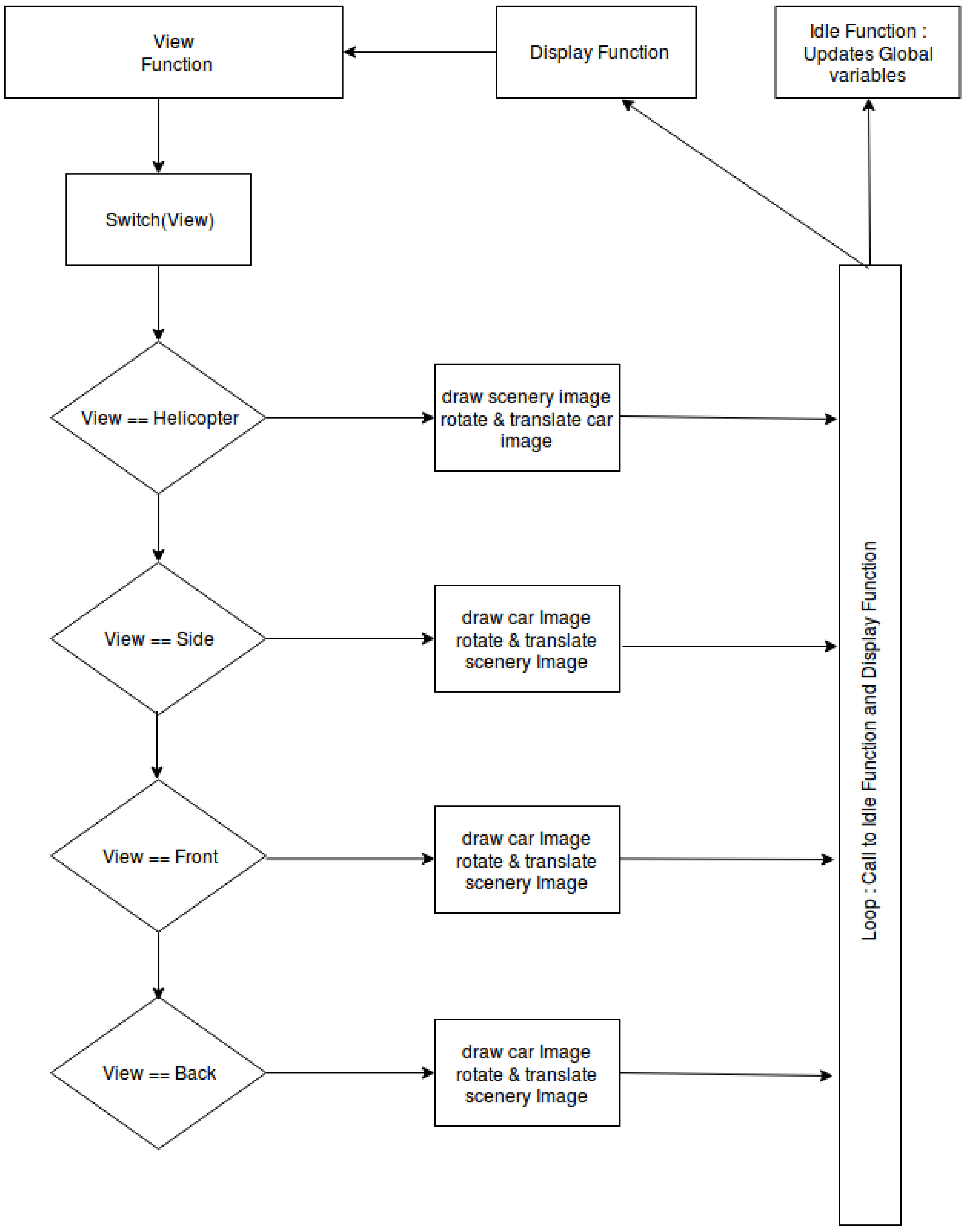

View Function
Display Function
Idle Function : Updates Global variables
Switch(View)
View == Helicopter
draw scenery image rotate & translate car image
View == Side
draw car Image rotate & translate scenery Image
View == Front
draw car Image rotate & translate scenery Image
View == Back
draw car Image rotate & translate scenery Image
Loop : Call to Idle Function and Display Function

Fig 3.1: Implementation Design

3.1 Components

The components of the visualizations are divided into 2 key components.

1. Car
2. Scenery

1. Car consists of the following parts :

    1. Chassis : The base of the car
    2. Wheels : Tyres on 4 sides along with axle

2. Scenery consists of

1. Background
2. Ground
3. Track
4. Trees

All of these elements are drawn independently and then appropriately translated rotated

An idle function is called by the Event Processing Loop.

We use this function to change the angle of the car or the scenery as per the user inputs.
The images are then appropriately translated and rotated by the view function

CHAPTER 4

IMPLEMENTATION

4.1 OpenGL Function Details

- **GlutInitDisplayMode —** sets the initial display mode.
  - Declaration: void glutInitDisplayMode (unsigned int mode);
  - Remarks: The initial display mode is used when creating top-level windows, sub windows, and overlays to determine the OpenGL display mode for the to-be-created window or overlay.
- **glutInitWindowposition ---** set the initial window position.
  - Declaration: void glutInitWindowPosition(int x, int y);

  x: Window X location in pixels.
  y: Window Y location in pixels.
- **glutInitWindowSize ---** set the initial window size.
  - Declaration: void glutInitWindowSize(int width,int height);

  width: Width in pixels
  height: Height in pixels.
- **glutCreateWindow** --- set the title to graphics window.
  - Declaration: Int glutCreateWindow(char *title);
  - Remarks: This function creates a window on the display. The string title can be used to label the window.The integer value returned can be used to set the current window when multiple windows are created.
- **glutDisplayFunc**
  - Declaration: void glutDisplayFunc(void(*func)void));
  - Remarks: This function registers the display function that is executed when the window needs to be redrawn.

- **glClear:**
  - Declaration: void glClear();
  - Remarks: This function clears the particular buffer.

- **glclearColor:**
  - Declaration:
    void glClearColor(GLfloat red, GLfloat green, Glfloat blue, Glfloat alpha);
  - Remarks: This function sets the color value that is used when clearing the color buffer.

- **glEnd**
  - Declaration: void glEnd();
  - Remarks: This function is used in conjunction with glBegin to delimit the vertices of an opengl primitive.

- **glMatrixMode**
  - Declaration: void glMatrixMode(GLenum mode);
  - Remarks: This function specifies which matrix will be affected by subsequent transformations mode can be GL_MODELVIEW,GL_PROJECTION or GL_TEXTURE..

- **gluOrtho2D**
  - Declaration:
    void glOrtho(GLdouble left, GLdouble right, GLdouble bottom,GLdouble top);
  - Remarks: It defines an orthographic viewing volume with all parameters measured from the center of the projection plane.

- **glutInit**
  - Declaration:
    Void glutInit(int *argc, char **argv);
  - Remarks: To start thru graphics system, we must first call glutInit (),glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT

4.2 Code in C Language

```
#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();

      //Sphere shaped tree top
      glColor3f(0,1,0.3);
      glPushMatrix();
            glTranslatef(a,b,45);
            glutSolidSphere(30,10,10);
      glPopMatrix();
}
```

```
//Function to generate the sides of the tyres
void alloy(float R1,float R2)
{
      float X,Y,Z;int y;
      glColor3f(0,0,0);
      glBegin(GL_QUAD_STRIP);
      for(y=0;y<=361;y+=1)
      {
            X=R1*cos(c*y);
            Z=R1*sin(c*y);
            Y=0;
            glVertex3f(X,Y,Z);

            X=R2*cos(c*y);
            Z=R2*sin(c*y);
            Y=0;
            glVertex3f(X,Y,Z);

      }
      glEnd();  }
//Function to draw the spokes of the wheel
void actall(float R1,float R2)
{
      float X,Y,Z; int i;
      glBegin(GL_QUADS);
      for(i=0;i<=361;i+=120)
      {
            glColor3f(0,0.5,0.5);
            X=R1*cos(c*i);
            Y=0;
            Z=R1*sin(c*i);
            glVertex3f(X,Y,Z);

            X=R1*cos(c*(i+30));
            Y=0;
            Z=R1*sin(c*(i+30));
            glVertex3f(X,Y,Z);

            X=R2*cos(c*(i+30));
            Y=0;
            Z=R2*sin(c*(i+30));
            glVertex3f(X,Y,Z);

            X=R2*cos(c*i);
            Y=0;
            Z=R2*sin(c*i);
            glVertex3f(X,Y,Z);
      }
      glEnd();
}
//Function to draw a circle
void circle(float R)
{
      float X,Y,Z;int i;

      glBegin(GL_POLYGON);
      for(i=0;i<=360;i++)
      {
            X=R*cos(c*i);
            Z=R*sin(c*i);
            Y=0;
            glVertex3f(X,Y,Z);
      }
      glEnd();
}


//Function to draw a quadrilateral
void rect(float p[],float q[],float r[],float s[])
{
      glBegin(GL_POLYGON);
            glVertex3fv(p);
            glVertex3fv(q);
            glVertex3fv(r);
            glVertex3fv(s);
      glEnd();
  }
//Function to generate car driver
void driver()
{
      glColor3f(0.5,0.2,0.8);
      //Legs
      glPushMatrix();
            glTranslatef(3,-3.5,1.5);
            glRotatef(90,0,0,1);
            cylinder(0.4,3);
      glPopMatrix();

      glPushMatrix();
            glTranslatef(3,-6.5,1.5);
            glRotatef(90,0,0,1);
            cylinder(0.4,3);
      glPopMatrix();

      //Hands
      glPushMatrix();
            glTranslatef(3,-3.5,2.5);
            glRotatef(90,0,0,1);
            cylinder(0.4,3);
      glPopMatrix();

      glPushMatrix();
            glTranslatef(3,-6.5,2.5);
            glRotatef(90,0,0,1);
            cylinder(0.4,3);
      glPopMatrix();

      //Head
      glPushMatrix();
            glTranslatef(3,-5,4);
            glutSolidSphere (1.0, 20, 16);
      glPopMatrix();


//Body
      glPushMatrix();
            glTranslatef(3,-5,1);
            glRotatef(90,1,0,0);
            cylinder(1,2);
      glPopMatrix();

      //Circle
      glPushMatrix();
            glTranslatef(3,-5,3);
            glRotatef(90,1,0,0);
            circle(1);
      glPopMatrix();
 }
//Function generating scenery using functions track( ),tree( ),tree2( )
void scenery()
{
      float x,y; int p;
      //Background
      glColor3f(0.4,0.9,0.9);
      glPushMatrix();
            glRotatef(90,1,0,0);
            cylinder(1000,1000);
      glPopMatrix();

      //Ground
      glColor3f(0,1,0);
      glPushMatrix();
            glTranslatef(0,0,-1.1);
            glRotatef(90,1,0,0);
            circle(1100);
      glPopMatrix();

      //Track
      glColor3f(0.3,0.3,0.6);
      track(600,540);

      //Cone shaped trees
      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()
{
      //axle
      glColor3f(0,0.5,0.3);
      cylinder(0.4,9);

      //1st Wheel
      glColor3f(0,0,0);
      cylinder(2,2);
```

```
     alloy(2,1.4);
     actall(1.4,0.8);
     glColor3f(0,0.5,0.4);
     circle(0.8);

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

     //2nd Wheel
     glPushMatrix();
           glTranslatef(0,8,0);
           glColor3f(0,0,0);
           cylinder(2,2);
           alloy(2,1.4);
           actall(1.4,0.8);
           glColor3f(0,0.5,0.4);
           circle(0.8);
     glPopMatrix();

     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();

		driver();

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

	glPopMatrix();
}

//Keyboard Callback 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;
      }

}

//Function  that generates a particular view of scene depending on view
selected by //user
void view()
{
      float pos[]={1000,1000,2000,1};//Position of the light source

      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;

                  case BACK:

                  gluLookAt(-12.0,6.0,13,15.0,6.0,2.0,0.0,0.0,1.0);
                  car();
                  glPushMatrix();
```

```
                      glRotatef(RAD_TO_DEG * angle, 0.0, 0.0, 1.0);
                      glTranslatef(-carx,-cary,0);
                      glLightfv(GL_LIGHT0, GL_POSITION, pos);
                      scenery();
                glPopMatrix();

                break;

      }

}

//Idle Callback Function
void idle()
{

      if(start==1)
      {
            angle+=0.05;
            if(angle==TWO_PI)
            {
                  angle-=TWO_PI;
            }

            carx=MID*sin(angle);
            cary=MID*cos(angle);

            switch(KEY)
            {
            case 'H':
            case 'h':viewpoint=HELICOPTER;break;

            case 'S':
            case 's':viewpoint=SIDE;break;

            case 'F':
            case 'f':viewpoint=FRONT;break;

            case 'B':
            case 'b':viewpoint=BACK;break;

            }
            glutPostRedisplay();
      }
}


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();
}
```

CHAPTER 5

DESCRIPTION AND SNAPSHOTS

5.1 DESCRIPTION

- The Car racing can be implemented in a multitude of ways. Our approach is only one of the many approaches that could have been taken.
- The main goal of our program is to simulate a car as it moves through a racing track.
- Simulation is an immensely helpful approach in Engineering design. It allows us to make better designs and asses how situations are handled by our product.
- Our Car race program is an illustration of how we can use simulation to see how our model behaves in a pseudo environment along with the ability to view and assess from multiple perspectives.
- The viewer has to option to view the car from different angles by pressing the appropriate keys specified in the program.

5.2 SCREEN SNAPSHOTS

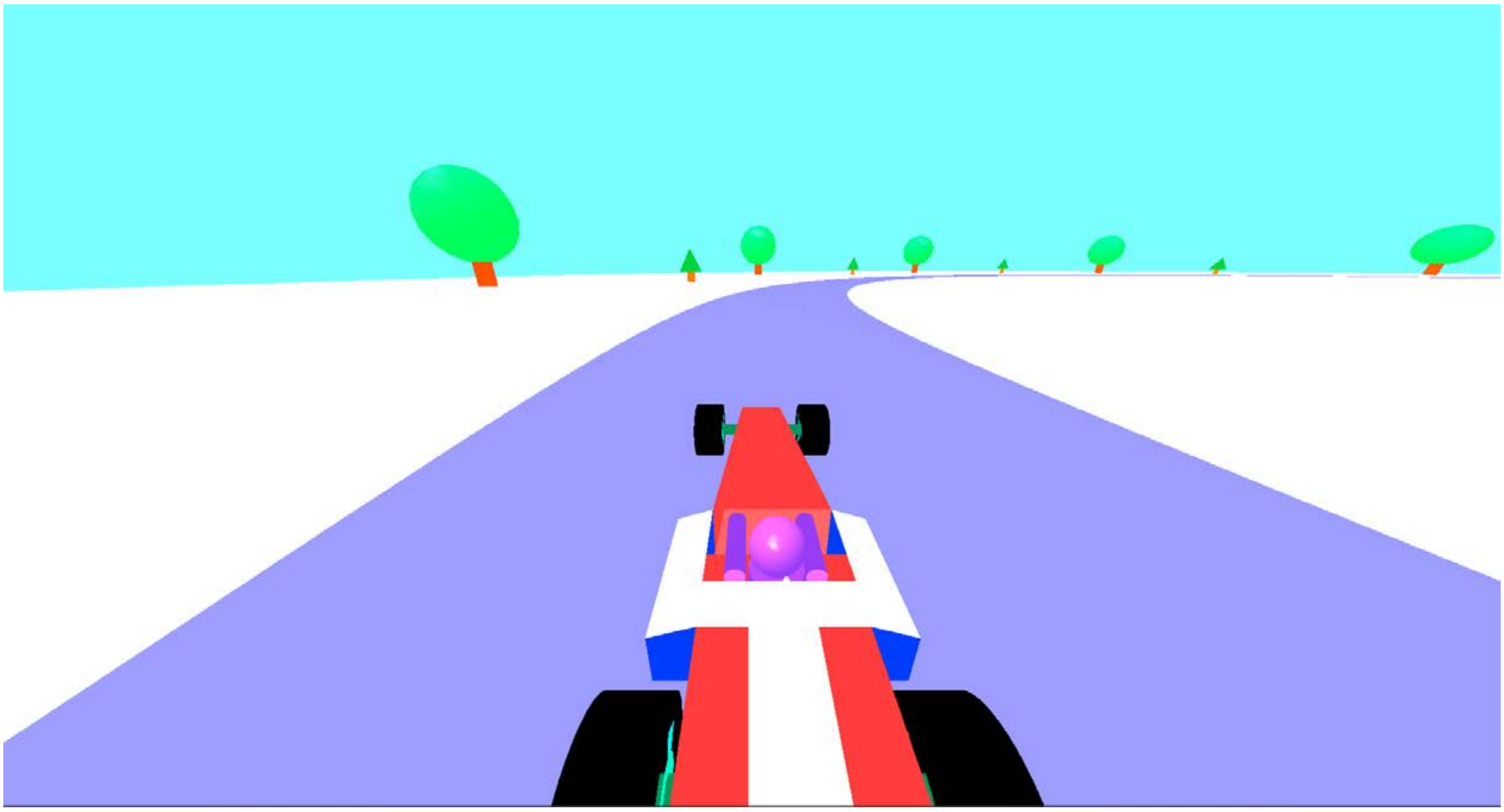


Fig 5.2.1: Back View

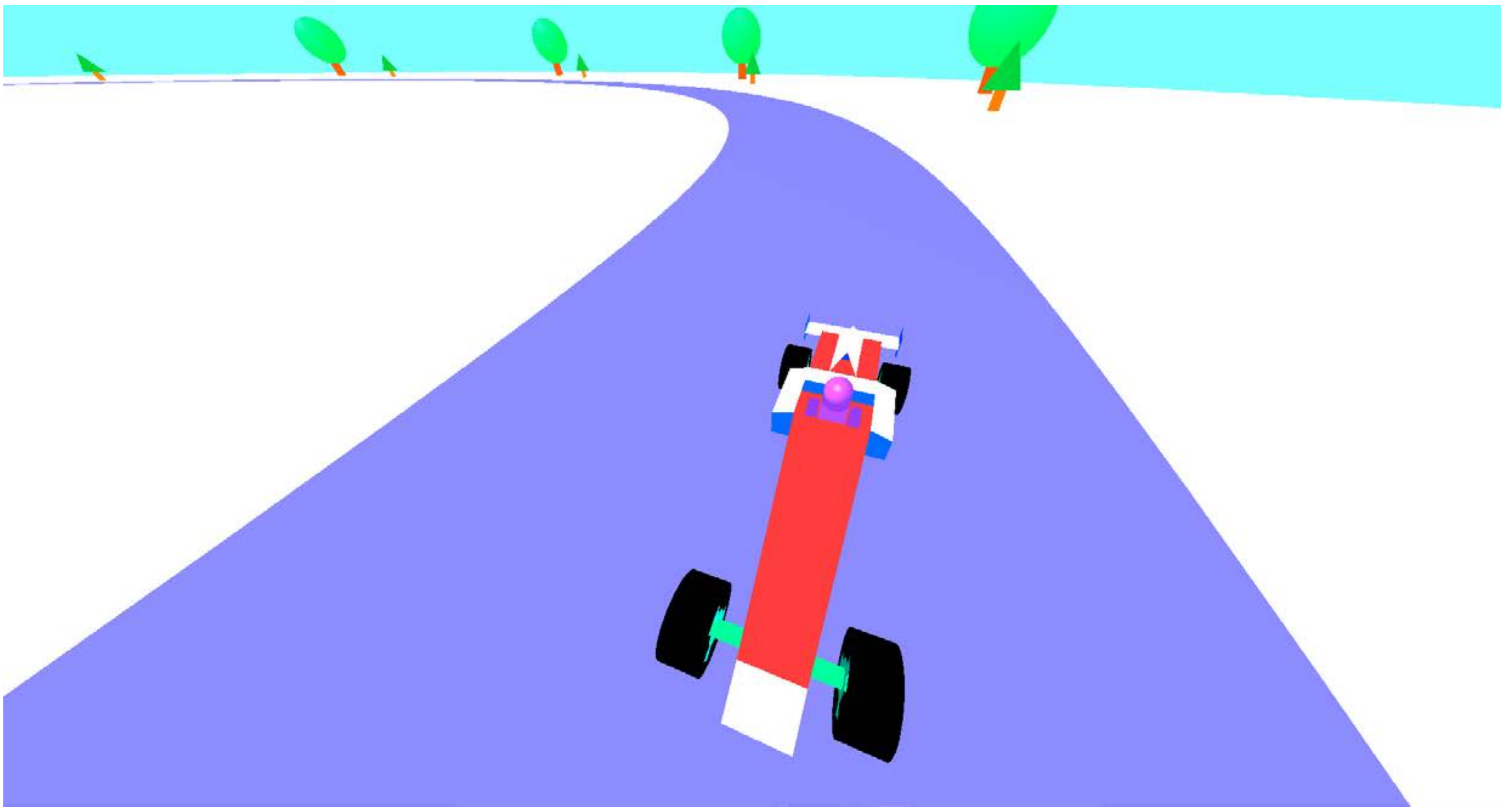


Fig 5.2.2: Front View

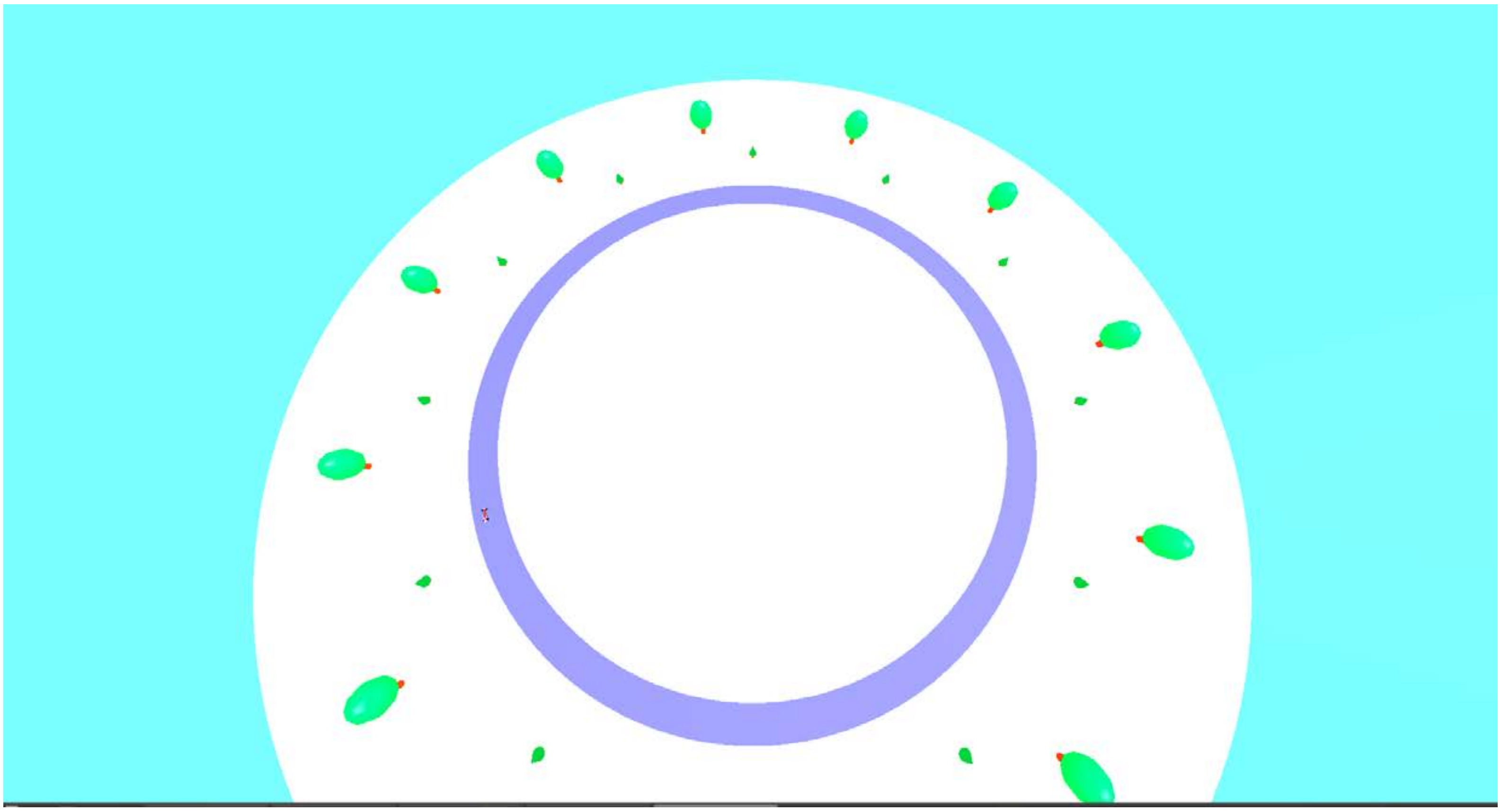


Fig 5.2.3: Helicopter View

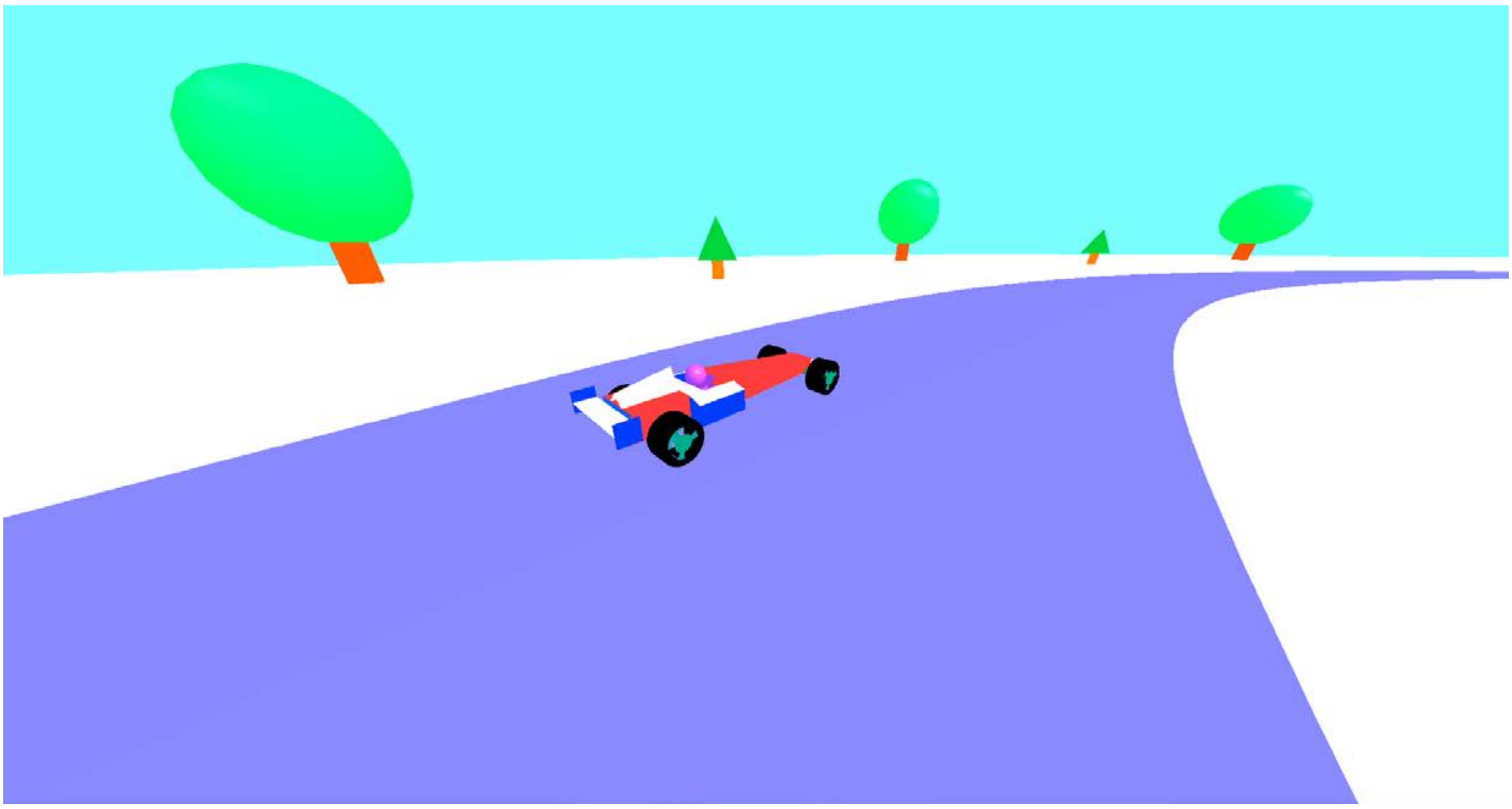
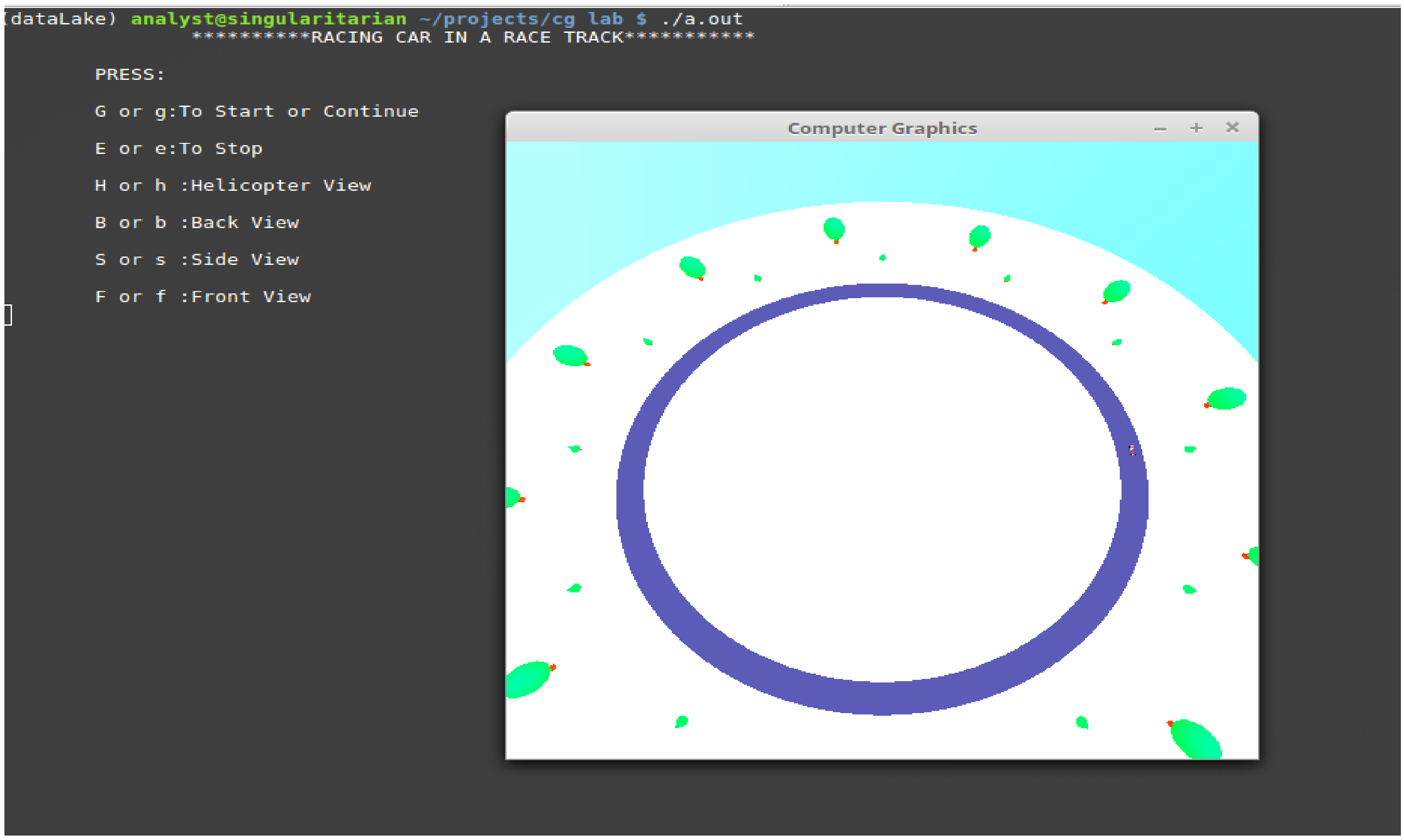


Fig 5.2.4 : Side View



```
(dataLake) analyst@singularitarian ~/projects/cg lab $ ./a.out
                **********RACING CAR IN A RACE TRACK***********

        PRESS:

        G or g:To Start or Continue

        E or e:To Stop

        H or h :Helicopter View

        B or b :Back View

        S or s :Side View

        F or f :Front View
```

Fig 5.2.5: Terminal View

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

By implementing this project I got to know how to use some of the built in functions effectively and how to interact efficiently and easily. I got a good exposure of how games, animation and simulations are developed, by working on this project.

The OpenGL Utility Toolkit (GLUT) is a programming interface with ANSI C bindings for writing window system independent OpenGL programs.

One of the major accomplishments in the specification of OpenGL was the isolation of window system dependencies from OpenGL's rendering model. The result is that OpenGL is window system independent. Window system operations such as the creation of a rendering window and the handling of window system events are left to the native window system to define. Necessary interactions between OpenGL and the window system such as creating and binding an OpenGL context to a window are described separately from the OpenGL specification in a window system dependent specification.

The GLUT application-programming interface (API) requires very few routines to display a graphics scene rendered using OpenGL. The GLUT API (like the OpenGL API) is stateful. Most initial GLUT state is defined and the initial state is reasonable for simple programs. The GLUT routines also take relatively few parameters. No pointers are returned. The only pointers passed into GLUT are pointers to character strings (all strings passed to GLUT are copied, not referenced) and opaque font handles.