**RV COLLEGE OF ENGINEERING® BENGALURU – 560059**

(Autonomous Institution Affiliated to VTU, Belagavi)

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



**“CAR PARK”**

# COMPUTER GRAPHICS LAB (16CS73)

**OPEN ENDED EXPERIMENT REPORT**

# VII SEMESTER

**2020-2021**

# Submitted by

**Santhosh K M- 1RV17CS140**

**Raghu -1RV17CS118**

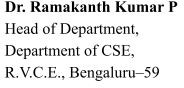
**Under the Guidance of Prof.Mamatha Department of CSE, R.V.C.E., Bengaluru - 560059**

**RV COLLEGE OF ENGINEERING®, BENGALURU - 560059**

**(Autonomous Institution Affiliated to VTU, Belagavi) DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**C ERTIFICATE**

Certified that the **Open-Ended Experiment** titled “**Car Park**” has been carried out by **Santhosh K M (1RV17CS140) and Raghu (1RV17CS118),** bonafide students of RV College of Engineering, Bengaluru, have submitted in partial fulfillment for the **Internal Assessment of Course: COMPUTER GRAPHICS LAB (16CS73)** during the year 2020-2021. It is certified that all corrections/suggestions indicated for the internal Assessment have been incorporated in the report.

**Prof.Mamatha** Faculty Incharge, Department of CSE,

## R.V.C.E., Bengaluru –59

**RV COLLEGE OF ENGINEERING® , BENGALURU - 560059**

**(Autonomous Institution Affiliated to VTU) DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**DECLARATION**

We, **Santhosh K M (1RV17CS140) and Raghu (1RV17CS118)** the students of Seventh Semester B.E., Computer Science and Engineering, R.V. College of Engineering, Bengaluru hereby declare that the mini-project titled **“CAR PARK”** has been carried out by us and submitted in partial fulfillment for the **Internal Assessment of Course: COMPUTER GRAPHICS LAB (16CS73) - Open-Ended Experiment** during the year 2020-2021. We do declare that the matter embodied in this report has not been submitted to any other university or institution for the award of any other degree or diploma.

**Place: Bengaluru Santhosh K M**

**Date: Raghu**

**A BSTRACT**

The aim of this project is to create a 3-D/VIRTUAL CAR PARK. The viewer is allowed to roam around in the parking area and see the cars closely and to drive a car and park it in the car park area. The parking area is surrounded by a number of houses.

First the co-ordinates of the car is calculated and then using the OPENGL PRIMITIVES the car is constructed. In a similar way the 3D house is constructed. A display list is constructed for each of these objects. These display lists are used everytime a car or house has to be constructed.

So, to create the 36 cars in the parking lot the “carr\_display\_list “ is called 36 times from within a loop and are translated each time by suitable values to place them correctly. Similarly, to construct the houses “house\_display\_list” is called and are suitably translated, scaled and rotated to place them properly. For the movement of camera GluLookAt ( )function is used .

OpenGL is the software interface to graphics hardware. This interface consists of aboutnumerous commands, which you use to specify the objects and operations needed to produce interactive applications.We have OPENGL PRIMITIVES the car is constructed. The PRIMITIVES used are:- GL\_LINES , GL\_POLYGON , GL\_QUADS, GL\_TRIANGLE

This project allows the user to rove in the parking lot and can even enter the houses that are present along the parking area. So, it’s like a virtualization of that area.

**ACKNOWLEDGMENT**

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped me in carrying out this project work. I would like to take this opportunity to thank them all.

I deeply express my sincere gratitude to my guide **Dr. Mamatha**, Associate Professor, Department of CSE, RVCE, Bengaluru, for his able guidance, regular source of encouragement and assistance throughout this project.

I would like to thank **Dr.Ramakanth Kumar P**, Head of Department, Computer Science & Engineering, R.V.C.E, Bengaluru, for his valuable suggestions and expert advice.

First and foremost I would like to thank **Dr. Subramanya. K. N**, Principal, R.V.C.E, Bengaluru, for his moral support towards completing my project work.

I thank my Parents, and all the Faculty members of Department of Computer Science & Engineering for their constant support and encouragement.

Last, but not the least, I would like to thank my peers and friends who provided me with valuable suggestions to improve my project.

**T ABLE OF CONTENTS**

1. Introduction
   1. Computer graphics
   2. OpenGL
      1. OpenGL Graphics architecture
      2. Primitives and attributes
      3. Color, viewing and control functions
   3. Proposed System
      1. Objective of the project
      2. Methodology
      3. Scope
2. Requirements Specifications
   1. Hardware Requirements
   2. Software Requirements
3. System Design and Implementation
   1. Modular Description
   2. Data Flow Diagram
4. Results and Snapshots
5. Conclusion
6. Bibliography
7. APPENDIX A - Source Code
8. **Introduction**
   1. **Computer Graphics**

Computer graphics is the pictorial representation and manipulation of data by a computer. A great deal of specialized hardware and software has been developed, with the displays of most devices being driven by computer graphics hardware. The term “computer graphics” it was coined by the William Fetter in the year of 1960 and he was a graphic designer for Boeing. In 1970’s many of the ancient and vital breakthroughs brought a change in the field of graphics and were introduced at the University of Utah. In 1980’s the sophistication of computer graphics were seen and in 1990’s the 3D graphics were introduced. In 2000’s the computer graphics were utilized in video games and in cinemas too, in 2010’s the texture mapping has evolved with many layers of processing.

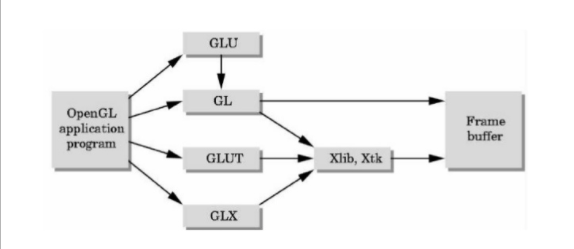
In these days everyone is fond of playing the video games on the phones, with a video game controller on the home television set, in laptops etc. It is involved with all aspects of generating images availing a computer and also involves with the pictorial representation of genuine or imaginary objects from the computer based models. This can be done by using the tools and with the help of that tools everyone is able to represent the data in the desired way and it is nothing but the computer graphics, along with that it furnishes the material to represent the data in a unique way.

The computer graphics are divided into two types and they are as Non-interactive computer graphics and Interactive computer graphics Non-interactive computer graphics: The non-interactive computer graphics are also called as passive computer graphics, in this type of graphics the observer has no power over the images. The examples of non-interactive graphics are the images or pictures on TV, other computer arts. Interactive computer graphics: The interactive computer graphics furnishes the exchange of information between the computer and user or observer. In this, the user has power over the image by furnishing the observer or user with an input machine and an example of this is a video game controller.

* 1. **OpenGL**

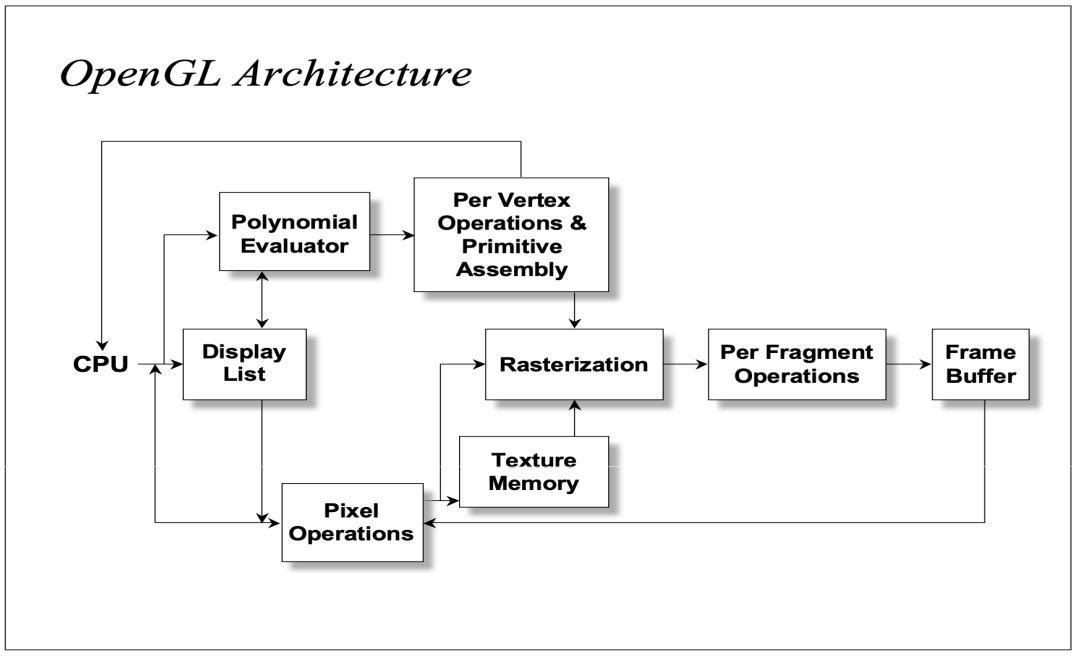
OpenGL **(Open Graphics Library) is the computer industry's standard application program interface ( API ) for defining 2-D and 3-D graphic images.OpneGL provide various viewing fu**nction that helps us to develop various views of sinle object and the way it appears on screen. Orthographic projection is the default view. OpenGL provide various transformation functions with the help of these functions user can render its object at the desired location on the screen. In OpenGL we obtain viewing and modelling functionality through a small set of transformation fucntions. We can even roate the object along desired locations and with the desired angle on the screen.

OpenGL provides a set of commands to render a three dimensional scene. OpenGL is a hardware and system independent interface. An OpenGL-application will work on every platform as long as there is an GLUT library.



shows the organization of the libraries for an X Window System environment. For this window system, GLUT will use GLX and the X libraries. The application program, however, can use only GLUT functions and thus can be recompiled with the GLUT library for other window systems

* + 1. **OpenGL Graphics Architecture**

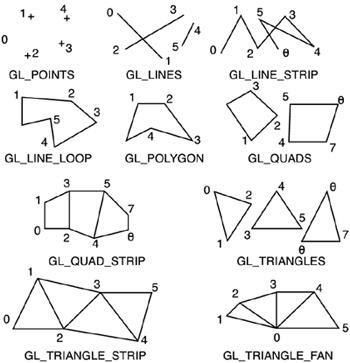


The architecture of OpenGL is based on a client-server model. An application program written to use the OpenGL API is the "client" and runs on the CPU. The implementation of the OpenGL graphics engine is the "server" and runs on the GPU. Geometry and many other types of attributes are stored in buffers called Vertx Buffer Objects. These buffers are allocated on the GPU and filled by your CPU program.

Modelling, rendering, and interaction is very much a cooperative process between the CPU client program and the GPU server programs written in GLSL. An important part of the design process is to decide how best to divide the work and how best to package and communicate the required information from the CPU to the GPU.

* + 1. **Primitives and Attributes**

OpenGL supports 2 types of primitives : Geometric primitives (vertices, line segments..) – they pass through the geometric pipeline and Raster primitives (arrays of pixels) – passes through a separate pipeline to the frame buffer.



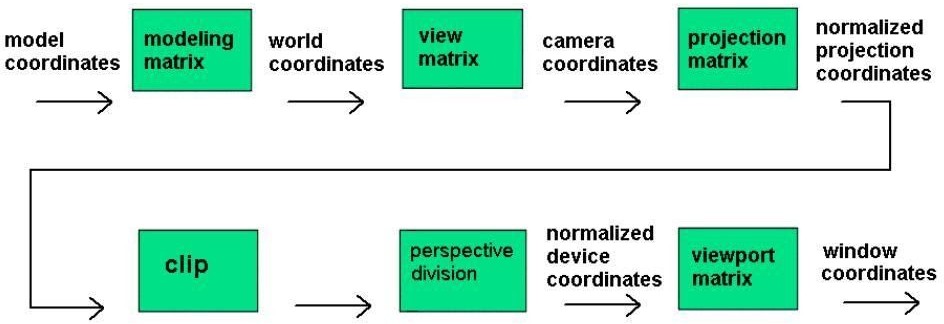
Similarly, for the GL\_TRIANGLES primitive, every third vertex causes a triangle to be drawn. Note that for the GL\_TRIANGLE\_STRIP and GL\_TRIANGLE\_FAN primitives, a new triangle is produced for every additional vertex. All of the closed primitives shown below are solid-filled, with the exception of GL\_LINE\_LOOP, which only draws lines connecting the vertices.

In general, a parameter that affects the way a primitive is to be displayed is referred to as an attribute parameter. Some attribute parameters, such as colour and size, determine the fundamental characteristics of a primitive. Other attributes specify how the primitive is to be displayed under special conditions. Text can appear reading from left to right, slanted diagonally across the screen, or in vertical columns. Individual characters can be displayed in different fonts, colours, and sizes. And we can apply intensity variations at the edges of objects to smooth out the raster stair-step effect.

* + 1. **Colour, Viewing and Control Functions**

OpenGL maintains a current drawing colour as part of its state information. The glColor() function calls are used to change the current drawing colour. assigned using the glColor function call. Like glVertex(), this function exists in various instantiations. Colour components are specified in the order of red, green, blue.

Colour component values are in the range [0...1], where 1 corresponds to maximum intensity. For unsigned bytes, the range corresponds to [0...255]. All primitives following the fragment of code given below would be drawn in green, assuming no additional glColor() function calls are used.



The coordinates we specify using the glVertex\* commands are the model coordinates. TheglRotate, glTranslate and glScale commands are used to transform the model into the desired orientation and size. After applying the modelling transformations to the model coordinates what we get are world coordinates. The Modelling transformations give rise to 4×4 matrices.

T he control functions are:

Window – A rectangular area of our display.

glutInit allows the application to get command line arguments and initializes the system.

glutInit(int \*argc, char \*\*argv) initializes GLUT and processes any command-line arguments (for X, this would be options like -display and -geometry). glutInit() should be called before any other GLUT routine. gluInitDisplayMode requests properties for the window (the rendering context)

* + - * RGB colour
      * Single buffering
      * Properties logically ORed together

glutInitDisplayMode(unsigned int mode) specifies whether to use an RGBA or color index color model. You can also specify whether you want a single- or double-buffered window.

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH). If you want a window with double buffering, the RGBA colour model, and a depth buffer, you might call

* + - * glutWindowSize(int width, int size) in pixels
      * glutWindowPosition(int x, int y) specifies the screen location for the upper-left corner of your window
      * glutCreateWindow(char \*string) create a window with a particular title
  1. **Proposed System**

The system consists of 36 cars in a parking lot and houses placed around it. The layout can be viewed differently by the use of arrow keys or other keys such as a, s, d, w, t. There will be option to engage or disable the view of the houses. One car will be programmed to move around using the similar keys.

* + 1. **Objective of the project**

The aim of this project is to create a 3-D/VIRTUAL CAR PARK. The viewer is allowed to roam around in the parking area and see the cars closely and to drive a car and park it in the car park area. The parking area is surrounded by a number of houses.

* + 1. **Methodology**

First the co-ordinates of the car is calculated and then using the OPENGL PRIMITIVES the car is constructed. The PRIMITIVES used are:-

1. GL\_LINES

2. GL\_POLYGON

3. GL\_QUADS

4. GL\_TRIANGLE

In a similar way the 3D house is constructed. A display list is constructed for each of these objects. These display lists are used everytime a car or house has to be constructed. So, to create the 36 cars in the parking lot the “carr\_display\_list “ is called 36 times from within a loop and are translated each time by suitable values to place them correctly. Similarly, to construct the houses “house\_display\_list” is called and are suitably translated, scaled and rotated to place them properly. For the movement of camera GluLookAt ( )function is used .

The controls are:-

1. UP KEY - > to move the viewer in forward direction.

2. DOWN KEY - > to move the viewer in backwards direction.

3. LEFT KEY - > to rotate the camera to the left of the viewer.

4. RIGHT KEY - > to rotate the camera to the right of the viewer.

5. T - >top view.

6. S - > to move away.

7. W - >to move near.

8. D - > to move right.

9. A - > to move left.

10. Q - >quit.

* + 1. **Scope**

1. 3-D MAP: This project can be modified and a lot of other objects can be added for example:- Trees , boundary walls, multiplexes ,roads etc. THUS A WHOLE CITY CAN BE CONSTRUCTED.

2. GAME: This program can be developed in to a fully-fledged game like Counter Strike,IGI,etc

1. **Requirement Specifications**
   1. **Hardware Requirements**
      * + Monitor
        + Keyboard
        + Mouse
        + CPU
        + Processor : Pentium PC
        + RAM : 512MB
        + Hard Disk : 20GB(approx)
        + Display : VGA Color Monitor
   2. **Software Requirements**
      * + Operating System : Windows 98SE/2000/XP/Vista/UBUNTU
        + OpenGL
        + Xcode
        + GCC
        + G++
        + Compiler : Eclipse/Microsoft Visual studio 2005
        + OpenGL Utility Toolkit
2. **System Design and Implementation**

First the co-ordinates of the car is calculated and then using the OPENGL PRIMITIVES the car is constructed. The PRIMITIVES used are:-

1. GL\_LINES

2. GL\_POLYGON

3. GL\_QUADS

4. GL\_TRIANGLE

In a similar way the 3D house is constructed. A display list is constructed for each of these objects. These display lists are used everytime a car or house has to be constructed. So, to create the 36 cars in the parking lot the “carr\_display\_list “ is called 36 times from within a loop and are translated each time by suitable values to place them correctly. Similarly, to construct the houses “house\_display\_list” is called and are suitably translated, scaled and rotated to place them properly. For the movement of camera GluLookAt ( )function is used . The controls are:- 1. UP KEY - > to move the viewer in forward direction.

2. DOWN KEY - > to move the viewer in backwards direction.

3. LEFT KEY - > to rotate the camera to the left of the viewer.

4. RIGHT KEY - > to rotate the camera to the right of the viewer.

5. T - >top view.

6. S - > to move away.

7. W - >to move near.

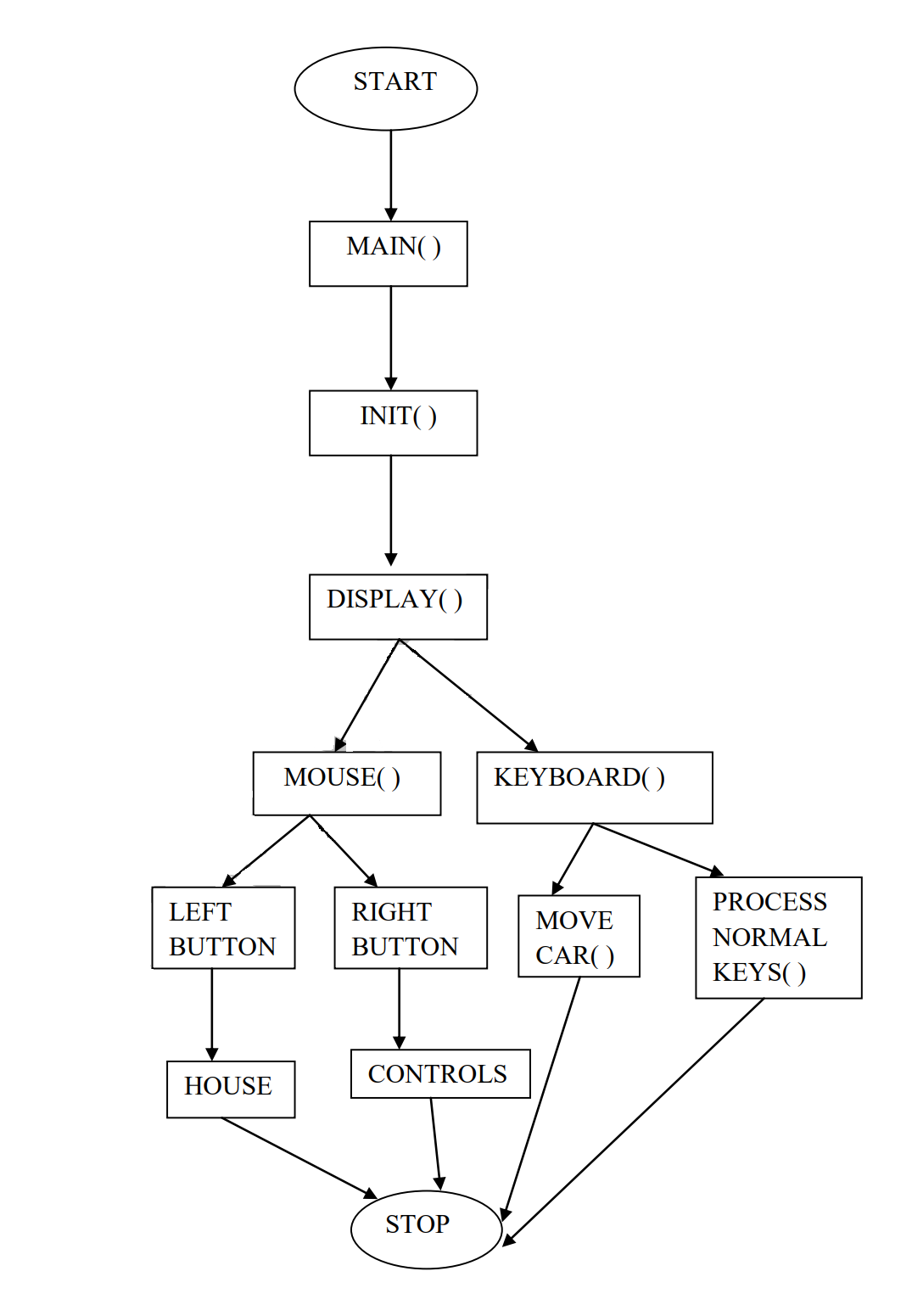
8. D - > to move right.

9. A - > to move left.

10. Q - >quit.

* 1. **Modular Description:**
* **void changeSize(int w, int h):** To define window size and other components accordingly
* **void drawcarr():** To design and color cars
* **void drawhouse():** To design and place houses
* **GLuint createDL():** Create and position the cars
* **GLuint createDL2().:** Create and position the houses
* **void initScene():** To define the scene
* **void processNormalKeys(unsigned char key, int x, int y):** To give controls to keys
* **void ProcessMenu1(int value):** Define menu on right click
* **void movecar(int key, int x, int y):** To define the move ment of car
* **void ProcessMenu(int value) :** Reset flags as appropriate in response to menu selections
* **void inputKey(int key, int x, int y):** To accept key values and controls
* **void renderScene(void):** to define how the scene will be viewed
* **void orientMe(float ang):** define the orientation view
* **void menu():** define the main menu on click
* **void moveMeFlat(int i):** To define forward view
  + 1. **User Defined Functions:**
* Keyboard function.
* Display function.
* Reshape function.

* 1. **Data Flow Diagram:**



**Results and Snapshots**

**INITIAL SCENE:**

This is the first scene which appears when the program is executed.



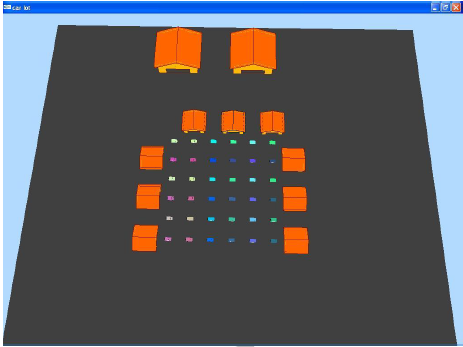
**FRONT VIEW :**

On pressing the “S” key the camera moves backwards and upwards simultaneously. The user can press “W” key to move the camera in the front direction in the same way but the path traversed is exactly opposite.



**TOP VIEW :**

On pressing the “t” key the camera changes to a position so that the user can see the top view of the whole parking area.



1. **Conclusion**

As the project was progressive in nature. The simulation has been developed using OpenGL functions and primitives such as polygons and provides user interaction through mouse and keyboard. The concepts and usage of pre-built functions in OpenGL was used to achieve the aim and was illustrated using the same. The user can view the car in any direction and simulate the direction of the car parking lot using these interactions and this project allows the user to rove in the parking lot and can even enter the houses that are present along the parking area.

1. **Bibliography Textbooks**

* Computer Graphics with OpenGL, Donald D. Hearn, M. Pauline Baker, Warren Carithers, 4th Edition, 2010, Pearson Education, ISBN-13: 978-0136053583.
* Interactive Computer Graphics: A Top-Down Approach Using OpenGL, Edward Angel, 5th Edition, 2010, Pearson Education, ISBN: 978131725306.
* Computer Graphics, Zhigang Xiang and Roy Plastock, 2nd Edition, 2007, ASIN: 0070601658, Tata McGraw-Hill, ISBN-13: 978-0070601659

**External Links**

* [h ttps://www.opengl.org/](https://www.opengl.org/)
* : [**http://www.cs.rutgers.edu/~decarlo/428/glman.html**](http://www.cs.rutgers.edu/~decarlo/428/glman.html)
* **http://nehe.gamedev.net OpenGL tutorials.**
* **http://www.opengl.org/sdk/docs/man**

**Appendix A - Source Code**

#include <GL/glut.h>

#include <math.h>

#include <stdlib.h>

static float angle=0.0,ratio;

static float x=0.0f,y=1.75f,z=5.0f;

static float lx=0.10f,ly=0.10f,lz=-1.0f;

static GLint carr\_display\_list,house\_display\_list;

float theta=0.01,fxincr=0.1,fzincr=0,temp,theta1,fx=-10,fz=80;

int xxxx=0,yyyy=0,kk=0,housevisible=0,movecarvar=0;

int a[36]={55,97,44,152,55,171,108,86,168,99,147,207,238,55,233,167,105,80,134,29,253,130,32,240,110,199,224,121,93,199,180,61,110,251,77,237};

int b[36]={102,194,110,152,153,184,137,113,55,138,104,43,240,255,203,8,100,53,88,64,127,64,87,5,2,144,211,128,10,89,27,11,175,185,157,241};

int c[36]={159,243,133,253,233,228,141,18,46,195,75,52,253,204,169,30,78,94,68,117,4,2,33,12,2,25,195,76,26,54,98,103,205,173,65,242};

void changeSize(int w, int h)

{

if(h == 0) // Prevent a divide by zero, when window is too short

// (you cant make a window of zero width).

h = 1;

ratio = 1.0f \* w / h; // Reset the coordinate system before modifying

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glViewport(0, 0, w, h); // Set the viewport to be the entire window

gluPerspective(45,ratio,1,1000);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(x, y, z,x + lx,y + ly,z + lz,0.0f,1.0f,0.0f);

}

void drawcarr()

{

glTranslatef(.0,0.8,0.0);

glEnable(GL\_BLEND); //TRANCPARENCY1

glBlendFunc(GL\_ONE, GL\_ZERO);//TRANCPARENCY2

glBegin(GL\_LINE\_LOOP);

glVertex3f(-1.12,-.48,0.7);//a

glVertex3f(-0.86,-.48,0.7);//b

glVertex3f(-.74,-0.2,0.7);//c

glVertex3f(-.42,-.2,0.7);//d

glVertex3f(-0.3,-.48,0.7);//e

glVertex3f(.81,-0.48,0.7);//f

glVertex3f(.94,-0.2,0.7);//g

glVertex3f(1.24,-.2,0.7);//h

glVertex3f(1.38,-.48,0.7);//i

glVertex3f(1.52,-.44,0.7);//j

glVertex3f(1.52,.14,0.7);//k

glVertex3f(1.14,0.22,0.7);//l

glVertex3f(0.76,.22,0.7);//m

glVertex3f(.52,0.56,0.7);//n

glVertex3f(-0.1,0.6,0.7);//0

glVertex3f(-1.02,0.6,0.7);//p

glVertex3f(-1.2,0.22,0.7);//q

glVertex3f(-1.2,-.28,0.7);//r

glEnd();

glBegin(GL\_LINE\_LOOP);

glVertex3f(-1.12,-.48,-0.7);//a'

glVertex3f(-0.86,-.48,-0.7);//b'

glVertex3f(-.74,-0.2,-0.7);//c'

glVertex3f(-.42,-.2,-0.7);//d'

glVertex3f(-0.3,-.48,-0.7);//e'

glVertex3f(.81,-0.48,-0.7);//f'

glVertex3f(.94,-0.2,-0.7);//g'

glVertex3f(1.24,-.2,-0.7);//h'

glVertex3f(1.38,-.48,-0.7);//i'

glVertex3f(1.52,-.44,-0.7);//j'

glVertex3f(1.52,.14,-0.7);//k'

glVertex3f(1.14,0.22,-0.7);//l'

glVertex3f(0.76,.22,-0.7);//m'

glVertex3f(.52,0.56,-0.7);//n'

glVertex3f(-0.1,0.6,-0.7);//o'

glVertex3f(-1.02,0.6,-0.7);//p'

glVertex3f(-1.2,0.22,-0.7);//q'

glVertex3f(-1.2,-.28,-0.7);//r'

glEnd();

glBegin(GL\_LINES);

glVertex3f(-1.12,-.48,0.7);//a

glVertex3f(-1.12,-.48,-0.7);//a'

glVertex3f(-0.86,-.48,0.7);//b

glVertex3f(-0.86,-.48,-0.7);//b'

glVertex3f(-.74,-0.2,0.7);//c

glVertex3f(-.74,-0.2,-0.7);//c'

glVertex3f(-.42,-.2,0.7);//d

glVertex3f(-.42,-.2,-0.7);//d'

glVertex3f(-0.3,-.48,0.7);//e

glVertex3f(-0.3,-.48,-0.7);//e'

glVertex3f(.81,-0.48,0.7);//f

glVertex3f(.81,-0.48,-0.7);//f'

glVertex3f(.94,-0.2,0.7);//g

glVertex3f(.94,-0.2,-0.7);//g'

glVertex3f(1.24,-.2,0.7);//h

glVertex3f(1.24,-.2,-0.7);//h'

glVertex3f(1.38,-.48,0.7);//i

glVertex3f(1.38,-.48,-0.7);//i'

glVertex3f(1.52,-.44,0.7);//j

glVertex3f(1.52,-.44,-0.7);//j'

glVertex3f(1.52,.14,0.7);//k

glVertex3f(1.52,.14,-0.7);//k'

glVertex3f(1.14,0.22,0.7);//l

glVertex3f(1.14,0.22,-0.7);//l'

glVertex3f(0.76,.22,0.7);//m

glVertex3f(0.76,.22,-0.7);//m'

glVertex3f(.52,0.56,0.7);//n

glVertex3f(.52,0.56,-0.7);//n'

glVertex3f(-0.1,0.6,0.7);//0

glVertex3f(-0.1,0.6,-0.7);//o'

glVertex3f(-1.02,0.6,0.7);//p

glVertex3f(-1.02,0.6,-0.7);//p'

glVertex3f(-1.2,0.22,0.7);//q

glVertex3f(-1.2,0.22,-0.7);//q'

glVertex3f(-1.2,-.28,0.7);//r

glVertex3f(-1.2,-.28,-0.7);//r'

glEnd();

glBegin(GL\_POLYGON); // top filling

glVertex3f(-0.1,0.6,0.7);//o

glVertex3f(-0.1,0.6,-0.7);//o'

glVertex3f(-1.02,0.6,-0.7);//p'

glVertex3f(-1.02,0.6,0.7);//p

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-0.1,0.6,0.7);//o

glVertex3f(-0.1,0.6,-0.7);//o'

glVertex3f(.52,0.56,-0.7);//n'

glVertex3f(.52,0.56,0.7);//n

glEnd();

glBegin(GL\_POLYGON); //back filling

glVertex3f(-1.2,0.22,0.7);//q

glVertex3f(-1.2,0.22,-0.7);//q'

glVertex3f(-1.2,-.28,-0.7);//r'

glVertex3f(-1.2,-.28,0.7);//r

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(1.52,.14,0.7);//k

glVertex3f(1.14,0.22,0.7);//l

glVertex3f(1.14,0.22,-0.7);//l'

glVertex3f(1.52,.14,-0.7);//k'

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(0.76,.22,0.7);//m

glVertex3f(0.76,.22,-0.7);//m'

glVertex3f(1.14,0.22,-0.7);//l'

glVertex3f(1.14,0.22,0.7);//l

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-1.12,-.48,0.7);//a

glVertex3f(-0.86,-.48,0.7);//b

glVertex3f(-.74,-0.2,0.7);//c

glVertex3f(-0.64,0.22,0.7);//cc

glVertex3f(-1.08,0.22,0.7);//dd

glVertex3f(-1.2,0.22,0.7);//q

glVertex3f(-1.2,-.28,0.7);//r

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-.74,-0.2,0.7);//c

glVertex3f(-0.64,0.22,0.7);//cc

glVertex3f(-0.5,0.22,0.7);//hh

glVertex3f(-0.5,-0.2,0.7);//pp

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(0.0,0.22,0.7);//gg

glVertex3f(1.14,0.22,0.7);//l

glVertex3f(1.24,-.2,0.7);//h

glVertex3f(0.0,-0.2,0.7);//oo

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-1.12,-.48,-0.7);//a'

glVertex3f(-0.86,-.48,-0.7);//b'

glVertex3f(-.74,-0.2,-0.7);//c'

glVertex3f(-0.64,0.22,-0.7);//cc'

glVertex3f(-1.08,0.22,-0.7);//dd'

glVertex3f(-1.2,0.22,-0.7);//q'

glVertex3f(-1.2,-.28,-0.7);//r'

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-.74,-0.2,-0.7);//c'

glVertex3f(-0.64,0.22,-0.7);//cc'

glVertex3f(-0.5,0.22,-0.7);//hh'

glVertex3f(-0.5,-0.2,-0.7);//pp'

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(0.0,0.22,-0.7);//gg'

glVertex3f(1.14,0.22,-0.7);//l'

glVertex3f(1.24,-.2,-0.7);//h'

glVertex3f(0.0,-0.2,-0.7);//oo'

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-1.2,0.22,0.7);//q

glVertex3f(-1.08,0.22,0.7);//dd

glVertex3f(-0.98,0.5,0.7);//aa

glVertex3f(-1.02,0.6,0.7);//p

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-1.02,0.6,0.7);//p

glVertex3f(-0.98,0.5,0.7);//aa

glVertex3f(0.44,0.5,0.7);//jj

glVertex3f(.52,0.56,0.7);//n

glVertex3f(-0.1,0.6,0.7);//0

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-0.64,0.5,0.7);//bb

glVertex3f(-0.64,0.22,0.7);//cc

glVertex3f(-0.5,0.22,0.7);//hh

glVertex3f(-0.5,0.5,0.7);//ee

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(0.0,0.5,0.7);//ff

glVertex3f(0.0,0.22,0.7);//gg

glVertex3f(0.12,0.22,0.7);//ll

glVertex3f(0.12,0.5,0.7);//ii

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(.52,0.56,0.7);//n

glVertex3f(0.44,0.5,0.7);//jj

glVertex3f(0.62,0.22,0.7);//kk

glVertex3f(0.76,.22,0.7);//m

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-.42,-.2,0.7);//d

glVertex3f(.94,-0.2,0.7);//g

glVertex3f(.81,-0.48,0.7);//f

glVertex3f(-0.3,-.48,0.7);//e

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(1.14,0.22,0.7);//l

glVertex3f(1.52,.14,0.7);//k

glVertex3f(1.52,-.44,0.7);//j

glVertex3f(1.38,-.48,0.7);//i

glVertex3f(1.24,-.2,0.7);//h

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-1.2,0.22,-0.7);//q'

glVertex3f(-1.08,0.22,-0.7);//dd'

glVertex3f(-0.98,0.5,-0.7);//aa'

glVertex3f(-1.02,0.6,-0.7);//p'

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-1.02,0.6,-0.7);//p'

glVertex3f(-0.98,0.5,-0.7);//aa'

glVertex3f(0.44,0.5,-0.7);//jj'

glVertex3f(.52,0.56,-0.7);//n'

glVertex3f(-0.1,0.6,-0.7);//0'

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-0.64,0.5,-0.7);//bb'

glVertex3f(-0.64,0.22,-0.7);//cc'

glVertex3f(-0.5,0.22,-0.7);//hh'

glVertex3f(-0.5,0.5,-0.7);//ee'

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(0.0,0.5,-0.7);//ff'

glVertex3f(0.0,0.22,-0.7);//gg'

glVertex3f(0.12,0.22,-0.7);//ll'

glVertex3f(0.12,0.5,-0.7);//ii'

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(.52,0.56,-0.7);//n'

glVertex3f(0.44,0.5,-0.7);//jj'

glVertex3f(0.62,0.22,-0.7);//kk'

glVertex3f(0.76,.22,-0.7);//m'

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-.42,-.2,-0.7);//d'

glVertex3f(.94,-0.2,-0.7);//g'

glVertex3f(.81,-0.48,-0.7);//f'

glVertex3f(-0.3,-.48,-0.7);//e'

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(1.14,0.22,-0.7);//l'

glVertex3f(1.52,.14,-0.7);//k'

glVertex3f(1.52,-.44,-0.7);//j'

glVertex3f(1.38,-.48,-0.7);//i'

glVertex3f(1.24,-.2,-0.7);//h'

glEnd();

glBegin(GL\_POLYGON); // door1 body- rear, near

glVertex3f(-0.5,0.22,0.7);//hh

glVertex3f(0.0,0.22,0.7);//gg

glVertex3f(0.0,-0.2,0.7);//oo

glVertex3f(-0.5,-0.2,0.7);//pp

glEnd();

glBegin(GL\_POLYGON); // door body- rear, far

glVertex3f(-0.5,0.22,-0.7);//hh'

glVertex3f(0.0,0.22,-0.7);//gg'

glVertex3f(0.0,-0.2,-0.7);//oo'

glVertex3f(-0.5,-0.2,-0.7);//pp'

glEnd();

glBegin(GL\_POLYGON); // door2 body- near, driver

glVertex3f(0.12,0.22,0.7);//ll

glVertex3f(0.62,0.22,0.7);//kk

glVertex3f(0.62,-0.2,0.7);//mm

glVertex3f(0.12,-0.2,0.7);//nn

glEnd();

glBegin(GL\_POLYGON); // door2 body- far, driver

glVertex3f(0.12,0.22,-0.7);//ll'

glVertex3f(0.62,0.22,-0.7);//kk'

glVertex3f(0.62,-0.2,-0.7);//mm'

glVertex3f(0.12,-0.2,-0.7);//nn'

glEnd();

glBegin(GL\_POLYGON);//front\*\*

glVertex3f(1.52,.14,0.7);//k

glVertex3f(1.52,.14,-0.7);//k'

glVertex3f(1.52,-.44,-0.7);//j'

glVertex3f(1.52,-.44,0.7);//j

glEnd();

glTranslatef(-.58,-.52,0.7); //translate to 1st tyre

glColor3f(0.09,0.09,0.09); // tyre color\*\*\*\*\*\*\*\*

glutSolidTorus(0.12f, .14f, 10, 25);

glTranslatef(1.68,0.0,0.0); //translate to 2nd tyre

glutSolidTorus(0.12f, .14f, 10, 25);

glTranslatef(0.0,0.0,-1.4); //translate to 3rd tyre

glutSolidTorus(0.12f, .14f, 10, 25);

glTranslatef(-1.68,0.0,0.0); //translate to 4th tyre which is behind 1st tyre rearback

glutSolidTorus(0.12f, .14f, 10, 25);

glTranslatef(.58,.52,0.7); //translate to origin

glRotatef(90.0,0.0,1.0,0.0);

glTranslatef(0.0,0.0,-1.40);

glutSolidTorus(0.2f, .2f, 10, 25);

glTranslatef(0.0,0.0,1.40);

glRotatef(270.0,0.0,1.0,0.0);

glBegin(GL\_POLYGON); //bottom filling

glColor3f(0.25,0.25,0.25);

glVertex3f(-0.3,-.48,0.7);//e

glVertex3f(-0.3,-.48,-0.7);//e'

glVertex3f(.81,-0.48,-0.7);//f'

glVertex3f(.81,-0.48,0.7);//f

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-.42,-.2,0.7);//d

glVertex3f(-.42,-.2,-0.7);//d'

glVertex3f(-0.3,-.48,-0.7);//e'

glVertex3f(-0.3,-.48,0.7);//e

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-1.2,-.28,0.7);//r

glVertex3f(-1.2,-.28,-0.7);//r'

glVertex3f(-1.12,-.48,-0.7);//a'

glVertex3f(-1.12,-.48,0.7);//a

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-1.12,-.48,0.7);//a

glVertex3f(-1.12,-.48,-0.7);//a'

glVertex3f(-0.86,-.48,-0.7);//b'

glVertex3f(-0.86,-.48,0.7);//b

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-0.86,-.48,0.7);//b

glVertex3f(-0.86,-.48,-0.7);//b'

glVertex3f(-.74,-0.2,-0.7);//c'

glVertex3f(-.74,-0.2,0.7);//c

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(-.74,-0.2,0.7);//c

glVertex3f(-.74,-0.2,-0.7);//c'

glVertex3f(-.42,-.2,-0.7);//d'

glVertex3f(-.42,-.2,0.7);//d

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(.81,-0.48,0.7);//f

glVertex3f(.81,-0.48,-0.7);//f'

glVertex3f(.94,-0.2,-0.7);//g'

glVertex3f(.94,-0.2,0.7);//g

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(.94,-0.2,0.7);//g

glVertex3f(.94,-0.2,-0.7);//g'

glVertex3f(1.24,-.2,-0.7);//h'

glVertex3f(1.24,-.2,0.7);//h

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(1.24,-.2,0.7);//h

glVertex3f(1.24,-.2,-0.7);//h'

glVertex3f(1.38,-.48,-0.7);//i'

glVertex3f(1.38,-.48,0.7);//i

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(1.38,-.48,0.7);//i

glVertex3f(1.38,-.48,-0.7);//i'

glVertex3f(1.52,-.44,-0.7);//j'

glVertex3f(1.52,-.44,0.7);//j

glEnd();

glBegin(GL\_LINE\_LOOP); // door outline- rear, front

glColor3f(1.0,1.0,1.0);

glVertex3f(-0.5,0.22,0.7);//hh

glVertex3f(0.0,0.22,0.7);//gg

glVertex3f(0.0,-0.2,0.7);//oo

glVertex3f(-0.5,-0.2,0.7);//pp

glEnd();

glBegin(GL\_LINE\_LOOP); // door2 outline- near, driver

glVertex3f(0.12,0.22,0.7);//ll

glVertex3f(0.62,0.22,0.7);//kk

glVertex3f(0.62,-0.2,0.7);//mm

glVertex3f(0.12,-0.2,0.7);//nn

glEnd();

glColor3f(0.0,0.0,0.0);

glBegin(GL\_LINE\_LOOP); // door2 outline- far, driver

glVertex3f(0.12,0.22,-0.7);//ll'

glVertex3f(0.62,0.22,-0.7);//kk'

glVertex3f(0.62,-0.2,-0.7);//mm'

glVertex3f(0.12,-0.2,-0.7);//nn'

glEnd();

glBegin(GL\_LINE\_LOOP); // door outline- rear, far

glVertex3f(-0.5,0.22,-0.7);//hh'

glVertex3f(0.0,0.22,-0.7);//gg'

glVertex3f(0.0,-0.2,-0.7);//oo'

glVertex3f(-0.5,-0.2,-0.7);//pp'

glEnd();

glBegin(GL\_POLYGON); //front\*\*

glVertex3f(1.52,.14,0.7);//k

glVertex3f(1.52,.14,-0.7);//k'

glVertex3f(1.52,-.44,-0.7);//j'

glVertex3f(1.52,-.44,0.7);//j

glEnd();

glColor3f(0.0,0.0,1.0);

// transparent objects are placed next ..

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA); //TRANCPARENCY3

//windscreen

glBegin(GL\_POLYGON);

glColor4f(0.0,0.0,0.0,0.7); //COLOR =WHITE TRANSPARENT

glVertex3f(0.562,.5,.6);//AAA

glVertex3f(.562,.5,-.6);//AAA'

glVertex3f(.76,.22,-.6);//MMM'

glVertex3f(.76,.22,.6);//MMM

glEnd();

glBegin(GL\_POLYGON); //rear window

//COLOR =WHITE TRANSPARENT

glVertex3f(-1.068,0.5,0.6);//pp

glVertex3f(-1.068,0.5,-0.6);//pp'

glVertex3f(-1.2,0.22,-0.6);//qq'

glVertex3f(-1.2,0.22,0.6);//qq

glEnd();

glBegin(GL\_POLYGON); //leftmost window front

glVertex3f(-0.98,0.5,0.7);//aa

glVertex3f(-0.64,0.5,0.7);//bb

glVertex3f(-0.64,0.22,0.7);//cc

glVertex3f(-1.08,0.22,0.7);//dd

glEnd();

glBegin(GL\_POLYGON); //leftmost window back

glVertex3f(-0.98,0.5,-0.7);//aa

glVertex3f(-0.64,0.5,-0.7);//bb

glVertex3f(-0.64,0.22,-0.7);//cc

glVertex3f(-1.08,0.22,-0.7);//dd

glEnd();

glBegin(GL\_POLYGON); //middle window front

glVertex3f(-0.5,0.5,0.7);

glVertex3f(0.0,0.5,0.7);

glVertex3f(0.0,0.22,0.7);

glVertex3f(-0.5,0.22,0.7);

glEnd();

glBegin(GL\_POLYGON); //middle window back

glVertex3f(-0.5,0.5,-0.7);

glVertex3f(0.0,0.5,-0.7);

glVertex3f(0.0,0.22,-0.7);

glVertex3f(-0.5,0.22,-0.7);

glEnd();

glBegin(GL\_POLYGON); //rightmost window front

glVertex3f(0.12,0.5,0.7);//ii

glVertex3f(0.44,0.5,0.7);//jj

glVertex3f(0.62,0.22,0.7);//kk

glVertex3f(0.12,0.22,0.7);//ll

glEnd();

glBegin(GL\_POLYGON); //rightmost window back

glVertex3f(0.12,0.5,-0.7);//ii'

glVertex3f(0.44,0.5,-0.7);//jj'

glVertex3f(0.62,0.22,-0.7);//kk'

glVertex3f(0.12,0.22,-0.7);//ll'

glEnd();

glColor3f(0.0,0.0,1.0);

}

void drawhouse()

{

glBegin(GL\_LINE\_LOOP);

glVertex3f(-2.6,-.84,2.5);//m

glVertex3f(-2.6,0.84,2.5);//n

glVertex3f(-3.04,0.84,2.8);//o

glVertex3f(0,1.95,2.8);//p

glVertex3f(3.04,0.84,2.8);//w

glVertex3f(2.6,0.84,2.5);//q

glVertex3f(2.6,-0.84,2.5);//r

glVertex3f(1.59,-0.84,2.5);//s

glVertex3f(1.59,0.16,2.5);//t

glVertex3f(-1.59,0.16,2.5);//u

glVertex3f(-1.59,-0.84,2.5);//v

glEnd();

glBegin(GL\_LINES);

glVertex3f(1.59,-0.84,2.5);//s

glVertex3f(-1.59,-0.84,2.5);//v

glEnd();

glBegin(GL\_LINE\_LOOP);

glVertex3f(-2.6,-.84,-2.5);//m'

glVertex3f(-2.6,0.84,-2.5);//n'

glVertex3f(-3.04,0.84,-2.8);//o'

glVertex3f(0,1.95,-2.8);//p'

glVertex3f(3.04,0.84,-2.8);//w'

glVertex3f(2.6,0.84,-2.5);//q'

glVertex3f(2.6,-0.84,-2.5);//r'

glVertex3f(1.59,-0.84,-2.5);//s'

glVertex3f(1.59,0.16,-2.5);//t'

glVertex3f(-1.59,0.16,-2.5);//u'

glVertex3f(-1.59,-0.84,-2.5);//v'

glEnd();

glBegin(GL\_LINES);

glVertex3f(-2.6,-.84,2.5);//m

glVertex3f(-2.6,-.84,-2.5);//m'

glVertex3f(-2.6,0.84,2.5);//n

glVertex3f(-2.6,0.84,-2.5);//n'

glVertex3f(-3.04,0.84,2.8);//o

glVertex3f(-3.04,0.84,-2.8);//o'

glVertex3f(0,1.95,2.8);//p

glVertex3f(0,1.95,-2.8);//p'

glVertex3f(3.04,0.84,2.8);//w

glVertex3f(3.04,0.84,-2.8);//w'

glVertex3f(2.6,0.84,2.5);//q

glVertex3f(2.6,0.84,-2.5);//q'

glVertex3f(2.6,-0.84,2.5);//r

glVertex3f(2.6,-0.84,-2.5);//r'

glVertex3f(1.59,-0.84,2.5);//s

glVertex3f(1.59,-0.84,-2.5);//s'

glVertex3f(-1.59,-0.84,2.5);//v

glVertex3f(-1.59,-0.84,-2.5);//v'

glEnd();

glColor3ub(255,185,1); //\*\*\*\*\*\*\*\*\*\*\*\*\*

glBegin(GL\_QUADS);

glVertex3f(-2.6,-.84,2.5);//m

glVertex3f(-2.6,0.16,2.5);//uu

glVertex3f(-1.59,0.16,2.5);//u

glVertex3f(-1.59,-0.84,2.5);//v

glVertex3f(-2.6,0.16,2.5);//uu

glVertex3f(-2.6,0.84,2.5);//n

glVertex3f(2.6,0.84,2.5);//q

glVertex3f(2.6,0.16,2.5);//tt

glVertex3f(1.59,-0.84,2.5);//s

glVertex3f(1.59,0.16,2.5);//t

glVertex3f(2.6,0.16,2.5);//tt

glVertex3f(2.6,-0.84,2.5);//r

glVertex3f(-2.6,-.84,-2.5);//m'

glVertex3f(-2.6,0.16,-2.5);//uu'

glVertex3f(-1.59,0.16,-2.5);//u'

glVertex3f(-1.59,-0.84,-2.5);//v'

glVertex3f(-2.6,0.16,-2.5);//uu'

glVertex3f(-2.6,0.84,-2.5);//n'

glVertex3f(2.6,0.84,-2.5);//q'

glVertex3f(2.6,0.16,-2.5);//tt'

glVertex3f(1.59,-0.84,-2.5);//s'

glVertex3f(1.59,0.16,-2.5);//t'

glVertex3f(2.6,0.16,-2.5);//tt'

glVertex3f(2.6,-0.84,-2.5);//r'

glVertex3f(-2.6,-.84,2.5);//m

glVertex3f(-2.6,-.84,-2.5);//m'

glVertex3f(-2.6,0.84,-2.5);//n'

glVertex3f(-2.6,0.84,2.5);//n

glVertex3f(2.6,0.84,2.5);//q

glVertex3f(2.6,0.84,-2.5);//q'

glVertex3f(2.6,-0.84,-2.5);//r'

glVertex3f(2.6,-0.84,2.5);//r

glEnd();

glBegin(GL\_TRIANGLES);

glVertex3f(0,1.95,2.5);//p

glVertex3f(3.04,0.84,2.5);//w

glVertex3f(-3.04,0.84,2.5);//o

glVertex3f(0,1.95,-2.5);//p'

glVertex3f(3.04,0.84,-2.5);//w'

glVertex3f(-3.04,0.84,-2.5);//o'

glEnd();

glColor3ub(255,102,0); //\*\*\*\*\*\*\*\*\*\*\*top color

glBegin(GL\_QUADS);

glVertex3f(0,1.95,2.8);//p

glVertex3f(0,1.95,-2.8);//p'

glVertex3f(3.04,0.84,-2.8);//w'

glVertex3f(3.04,0.84,2.8);//w

glVertex3f(-3.04,0.84,2.8);//o

glVertex3f(-3.04,0.84,-2.8);//o'

glVertex3f(0,1.95,-2.8);//p'

glVertex3f(0,1.95,2.8);//p

glEnd();

glColor3ub(116,18,0); //\*\*\*\*\*\*\*base color

glBegin(GL\_QUADS);

glVertex3f(-2.6,-.84,2.5);//m

glVertex3f(2.6,-0.84,2.5);//r

glVertex3f(2.6,-0.84,-2.5);//r'

glVertex3f(-2.6,-.84,-2.5);//m'

glEnd();

}

GLuint createDL()

{

GLuint carrDL;

carrDL = glGenLists(1); // Create the id for the list

glNewList(carrDL,GL\_COMPILE); // start list

drawcarr(); // call the function that contains the rendering commands

glEndList(); // endList

return(carrDL);

}

GLuint createDL2() //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

{

GLuint houseDL;

houseDL = glGenLists(1); // Create the id for the list

glNewList(houseDL,GL\_COMPILE); // start list

drawhouse(); // call the function that contains the rendering commands

glEndList(); // endList

return(houseDL);

} //\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void initScene()

{

glEnable(GL\_DEPTH\_TEST);

carr\_display\_list = createDL();

house\_display\_list= createDL2(); //\*\*\*\*\*\*\*\*\*\*\*

}

void renderScene(void)

{

int i,j;

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glClearColor(.7,0.85,1.0,1.0);

glColor3f(0.25f, 0.25f, 0.25f); // Draw ground

glBegin(GL\_QUADS);

glVertex3f(-100.0f, 0.0f, -100.0f);

glVertex3f(-100.0f, 0.0f, 100.0f);

glVertex3f( 100.0f, 0.0f, 100.0f);

glVertex3f( 100.0f, 0.0f, -100.0f);

glEnd();

for( i = -3; i < 3; i++) // Draw 36 car

for( j=-3; j < 3; j++)

{

glPushMatrix();

glTranslatef((i)\*10.0,0,(j) \* 10.0);

glColor3ub(a[i],b[j],c[i]);

glCallList(carr\_display\_list);

glPopMatrix();

}

if(housevisible)

{

glPushMatrix();

glScalef(2.0,2.0,2.0);

glTranslatef(0.0,.85,-20.0);

glCallList(house\_display\_list);

glTranslatef(10.0,0.0,0.0);

glCallList(house\_display\_list);

glTranslatef(-20.0,0.0,0.0);

glCallList(house\_display\_list);

glRotatef(90,0.0,1.0,0.0);

glTranslatef(-10.0,0.0,-10.0);

glCallList(house\_display\_list);

glTranslatef(-10.0,0.0,0.0);

glCallList(house\_display\_list);

glTranslatef(-10.0,0.0,0.0);

glCallList(house\_display\_list);

glPopMatrix();

glPushMatrix();

glTranslatef(10.0,3.4,-80.0);

glScalef(4.0,4.0,4.0);

glCallList(house\_display\_list);

glTranslatef(-10.0,0.0,0.0);

glCallList(house\_display\_list);

glPopMatrix();

glPushMatrix();

glRotatef(90,0.0,1.0,0.0);

glScalef(2.0,2.0,2.0);

glTranslatef(0.0,0.85,15.0);

glCallList(house\_display\_list);

glTranslatef(10.0,0.,0.0);

glCallList(house\_display\_list);

glTranslatef(-20.0,0.,0.0);

glCallList(house\_display\_list);

glPopMatrix();

}

if(fxincr!=0)

theta1=(atan(fzincr/fxincr)\*180)/3.141;

else if(fzincr>0)

theta1=-90.0;

else theta1=90.0;

if(fxincr>0&&fzincr<0)

{

theta1=-theta1;

}

else if(fxincr<0&&fzincr<0)

{

theta1=180-theta1;

}

else if(fxincr<0&&fzincr>0)

{

theta1=-180-theta1;

}else if(fxincr>0&&fzincr>0)

{

theta1=-theta1;

}

glPushMatrix();

glTranslatef(fx,0,fz);

glRotatef(theta1,0,1,0);

glColor3f(0.8,0.8,0);

glCallList(carr\_display\_list);

glPopMatrix();

glutSwapBuffers();

}

void orientMe(float ang)

{

lx = sin(ang);

lz = -cos(ang);

glLoadIdentity();

gluLookAt(x, y, z, x + lx,y + ly,z + lz,0.0f,1.0f,0.0f);

}

void moveMeFlat(int i)

{

if(xxxx==1)

y=y+i\*(lz)\*0.1; //\*\*\*\*\*\*\*\*\*

if(yyyy==1)

{

x=x+i\*(lz)\*.1;

}

else

{

z = z + i\*(lz)\*0.5;

x = x + i\*(lx)\*0.5;}

glLoadIdentity();

gluLookAt(x, y, z,x + lx,y + ly,z + lz,0.0f,1.0f,0.0f);

}

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

{

glLoadIdentity();

if (key == 'q')

exit(0);

if(key=='t')

gluLookAt(1,190,50,0,0 ,-10,0.0,1.0,.0);

if(key=='a')

moveMeFlat(4);xxxx=1,yyyy=0;

if(key=='s')

moveMeFlat(-4);xxxx=1,yyyy=0;

if(key=='w')

moveMeFlat(4);yyyy=1;xxxx=0;

if(key=='d')

moveMeFlat(-4);yyyy=1;xxxx=0;

}

void inputKey(int key, int x, int y)

{

switch (key)

{

case GLUT\_KEY\_LEFT : angle -= 0.05f;orientMe(angle);break;

case GLUT\_KEY\_RIGHT : angle +=0.05f;orientMe(angle);break;

case GLUT\_KEY\_UP : moveMeFlat(2);xxxx=0,yyyy=0;break;

case GLUT\_KEY\_DOWN : moveMeFlat(-2);xxxx=0,yyyy=0;break;

}

}

void movecar(int key, int x, int y)

{

switch (key)

{

case GLUT\_KEY\_LEFT :temp=fxincr;

fxincr=fxincr\*cos(theta)+fzincr\*sin(theta);

fzincr=-temp\*sin(theta)+fzincr\*cos(theta);

fx+=fxincr;

fz+=fzincr;

break;

case GLUT\_KEY\_RIGHT :temp=fxincr;

fxincr=fxincr\*cos(-theta)+fzincr\*sin(-theta);

fzincr=-temp\*sin(-theta)+fzincr\*cos(-theta);

fx+=fxincr;

fz+=fzincr;

break;

case GLUT\_KEY\_UP :fx+=fxincr;

fz+=fzincr;break;

case GLUT\_KEY\_DOWN :fx-=fxincr;

fz-=fzincr; break;

}

glutPostRedisplay();

}

void ProcessMenu(int value) // Reset flags as appropriate in response to menu selections

{

glutPostRedisplay();

}

void ProcessMenu1(int value)

{

switch(value)

{

case 1:if(housevisible==0)

housevisible=1;

else

housevisible=0;

glutPostRedisplay();

break;

case 2:if(movecarvar==0)

{

glutSpecialFunc(movecar);

movecarvar=1;

}

else

{

glutSpecialFunc(inputKey);

movecarvar=0;

}

break;

}

}

void menu()

{

int control;

int control1;

control= glutCreateMenu(ProcessMenu);

glutAddMenuEntry("\*\*CONTROLS\*\*",1);

glutAddMenuEntry("1) UP KEY:to move in Forward Direction.",1);

glutAddMenuEntry("2) DOWN KEY:to move in Backward Direction.",1);

glutAddMenuEntry("3) LEFT KEY:to Turn Left .",1);

glutAddMenuEntry("4) RIGHT KEY:to Turn Right .",1);

glutAddMenuEntry("5) d:moves Towards Right. ",1);

glutAddMenuEntry("6) a:moves Towards Left.",1);

glutAddMenuEntry("7) s:moves Away.",1);

glutAddMenuEntry("8) w:moves Near.",1);

glutAddMenuEntry("9) t:Top view.",1);

glutAddMenuEntry("10) q:Quit.",1);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

control1=glutCreateMenu(ProcessMenu1);

glutAddMenuEntry("HOUSE",1);

glutAddMenuEntry("MOVE CAR",2);

glutAttachMenu(GLUT\_LEFT\_BUTTON);

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowPosition(0,0);

glutInitWindowSize(1010,710);

glutCreateWindow("car lot");

initScene();

glutKeyboardFunc(processNormalKeys);

glutSpecialFunc(inputKey);

menu();

glutDisplayFunc(renderScene);

glutIdleFunc(renderScene);

glutReshapeFunc(changeSize);

glutMainLoop();

return(0);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*