VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi – 590 018

A Computer Graphics Project Report On

"CAR PARKING"

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

Bachelor of Engineering

In

Computer Science & Engineering

By

K PRATHVI RAO (4MW20CS032)

KRITHIKA (4MW20CS040)

Under the Guidance of

Ms. Soundharya R

Asst. Prof, Dept. of CSE

SMVITM

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Shri Madhwa Vadiraja Institute of Technology & Management

Vishwothama Nagar, Bantakal-574115

July, 2022

SHRI MADHWA VADIRAJA

INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(A Unit of Shri Sode Vadiraja Mutt Education Trust ®, Udupi)

Vishwothama Nagar, BANTAKAL – 574 115, Udupi District, Karnataka, INDIA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the Computer Graphics Project work entitled **"Car Parking"** has been carried out by **K Prathvi Rao (4MW20CS032) and Krithika (4MW20CS040)** respectively bonafide student of Shri Madhwa Vadiraja Institute of Technology and Management in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belagavi during the year **2022-2023**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. The Graphics Project Report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said Degree.

Ms. Soundharya R	**Dr. Sowmya J Bhat**
Project Guide	Professor and Head
Dept. of CSE	Dept. of CSE

External Viva

Name of the examiners	Signature with date
1.	
2.	

ABSTRACT

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. Computers have become a powerful medium for the rapid and economic production of pictures. There is virtually no area in which graphical displays cannot be used to some advantage.

Graphics provide a natural means of communicating with the computer that they have become widespread. Interactive graphics have been the most important means of producing pictures since the invention of photography and television.

This computer graphics project aims to create a 3D/virtual car park, providing users with an interactive and immersive experience. The virtual environment allows users to freely navigate within the parking area, enabling them to closely examine cars and experience the process of driving and parking a car in the designated spaces. The simulation includes realistic visual representations of cars and houses surrounding the parking area. By utilizing computer graphics techniques, users can explore the virtual car park, interact with the environment, and gain a realistic understanding of car parking. The project combines elements of 3D modeling, rendering, and user interaction to create an engaging and educational simulation.

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. This project uses the GLUT pre-built models' sub-API in openGL.

ACKNOWLEDGEMENT

It is our duty to acknowledge the help rendered to us by various persons in completing this Computer Graphics Project titled “Car Parking.”

We would like to express our heartfelt thanks to **Dr. Thirumaleshwara Bhat, Principal, SMVITM, Bantakal,** for extending his support.

We would also like to express our deepest gratitude to **Dr. Sowmya J Bhat, HOD, Computer Science and Engineering** whose guidance and support was truly invaluable in completing this project on time**.**

It is our privilege to express our sincerest regards to our project coordinator, **Ms. Soundharya R**, for her valuable inputs, guidance, encouragement and whole-hearted cooperation throughout the duration of our project.

We take this opportunity to thank all our teaching and non-teaching staff of the Department of CSE who have directly or indirectly helped the completion of our project. We pay our respects and love to our parents and all other family members and friends for their love and encouragement throughout our student life.

Thanking you all,

Krithika

K Prathvi Rao

TABLE OF CONTENTS

LISTS OF TABLES AND FIGURES

Chapter 1

INTRODUCTION

1.1 About Computer Graphics

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. Computers have become a powerful medium for the rapid and economical production of pictures. There is virtually no area in which graphical displays cannot be used to some advantage. Graphics provide a natural means of communicating with the computer that they have become widespread. Interactive graphics have been the most important means of producing pictures since the invention of photography and television. We can make pictures of not only the real-world objects but also of abstract objects such as mathematical surfaces on 4D and of data that have no inherent geometry. A computer graphics system is a computer system with all the components of the general-purpose computer system. There are five major elements in a system: input devices, processor, memory, frame buffer, output devices.

1.2 OpenGL Technology

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms. OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment. OpenGL Available Everywhere: Supported on all UNIX® workstations, and shipped standard with every Windows 95/98/2000/NT and MacOS PC, no other graphics API operates on a wider range of hardware platforms and software environments. OpenGL runs on every major operating system including Mac OS, OS/2, UNIX, Windows 95/98, Windows 2000, Windows NT,

Linux, OPENStep, and BeOS; it also works with every major windowing system, including Win32, MacOS, Presentation Manager, and X-Window System. OpenGL is callable from Ada, C, C++, Fortran, Python, Perl and Java and offers complete independence from network protocols and topologies. The OpenGL interface: Our application will be designed to access OpenGL directly through functions in three libraries namely: gl,glu,glut .

Chapter 2

REQUIREMENT SPECIFICATIONS

2.1 Hardware Requirements

The standard output device is assumed to be a Color Monitor. It is quite essential for any graphics package to have this, as provision of color options to the user is a must. The mouse, the main input device, has to be functional i.e. used to give input in the game. A keyboard is used for controlling and inputting data in the form of characters, numbers i.e. to change the user views.

Processor	Pentium III or higher
RAM	16 MB or higher
CD-ROM	Speed 48x and above
Cache memory	256 KB
Display	800x600 or higher

2.2 Software Requirements

The editor has been implemented on the OpenGL platform and mainly requires an appropriate version of eclipse compiler to be installed and functional in ubuntu. Though it is implemented in OpenGL, it is very much performed and independent with the restriction that there is support for the execution of C and C++ files. Text Modes are recommended.

Operating System	Windows 11
Language Used	C/C++
Used Application	Codeblocks 16.01

2.2.1 HeaderFiles

- GL/glut.h
- stdio.h
- stdlib.h
- math.h

Chapter 3

DESIGN

3.1 Transformation Functions

Matrices allow arbitrary linear transformations to be represented in a consistent format, suitable for computation. This also allows transformations to be concatenated easily (by multiplying their matrices). Linear transformations are not the only ones that can be represented by matrices. Using homogeneous coordinates,both affine transformation and perspective projection on R n can be represented as linear transformations on RPn+1 (that is, n+1- dimensional real projective space). For this reason, 4x4 transformation matrices are widely used in 3D computer graphics. 3-by-3 or 4-by-4 transformation matrices containing homogeneous coordinates are often called, somewhat improperly, "homogeneous transformation matrices". However, the transformations they represent are, in most cases, definitely non-homogeneous and nonlinear (like translation, roto-translation or perspective projection). And even the matrices themselves look rather heterogeneous, i.e. composed of different kinds of elements (see below). Because they are multi-purpose transformation matrices, capable of representing both affine and projective transformations, they might be called "general transformation matrices", or, depending on the application, "affine transformation" or "perspective projection" matrices. Moreover, since the homogeneous coordinates describe a projective vector space, they can also be called "projective space transformation matrices".

Chapter 4

IMPLEMENTATION

4.1 Header Files

4.1.1 GL/glut.h

The OpenGL Utility Toolkit (GLUT) handles most of the system dependent actions required to display a window, put OpenGL graphics in it, and accept mouse and keyboard input. glut.h includes gl.h and glu.h.

4.1.2 Stdio.h

The C Programming language provides many library functions of standard form for file input and output. It consists of operations such as file access, unformatted input and/or output, file positioning and error handling.

4.1.3 Stdlib.h

is the header of the general purpose standard library of C programming language which includes functions involving memory allocation, process control, conversions and others. It is compatible with C++ and is known as cstdlib.h in C++. The name "stdlib" stands for "standard library".

4.1.4 Math.h

are C mathematical operations as a group of functions in the standard library of the C programming language implementing basic mathematical functions

4.2 Functions Used

4.2.1 glColor3f();

This function is used to mention the color in which the pixel should appear. The number 3 specifies the number of arguments that the function would take. 'f ' gives the type that is float. The arguments are in the order RGB(Red, Green, Blue). The color of the pixel can be specified as the combination of these 3 primary colors.

4.2.2 glClearColor();

This function is used to clear the color of the screen. The 4 values that are passed as arguments for this function are (Red, Green, Blue, Alpha) where the red green and blue components are taken to set the background color and alpha is a value that specifies depth of the window. It is used for 3D images.

4.2.3 glutKeyboardFunc();

Void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y)); where func is the new keyboard callback function. glutKeyboardFunc sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed.

4.2.4 glFlush();

Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. glFlush empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

4.2.5 glMatrixMode();

where mode Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: GL_MODELVIEW, GL_PROJECTION, and GL_TEXTURE. The initial value is GL_MODELVIEW. glMatrixMode sets the current matrix mode. mode can assume one of three values:

GL_MODELVIEW: Applies subsequent matrix operations to the modelview matrix stack.

GL_PROJECTION: Applies subsequent matrix operations to the projection matrix stack.

4.2.6 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 program with an error message to the user if GLUT cannot be properly initialized. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command line options.glutInit also processes command line options, but the specific options parse are window system dependent. void glutReshapeFunc(void (*func)(int width, int height)); glutReshapeFunc sets the reshape callback for the current window. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established. The width and height parameters of the callback specify the new window size in pixels.

4.2.7 glViewport() ;

Where x, y Specify the lower left corner of the viewport rectangle, in pixels. The initial value is (0, 0). width, height Specify the width and height of the viewport. When a GL context is first attached to a surface, width and height are set to the dimensions of that surface. glViewport specifies the affine transformation of x and y from normalized device coordinates to window coordinates.

4.2.8 glClear();

The glClear function clears buffers to preset values.

SYNTAX: glClear(GLbitfield mask);

where 'mask' is Bitwise OR operators of masks that indicate the buffers to be cleared. The masks used are as follows:

GL_COLOR_BUFFER_BIT: The buffers currently enabled for color writing.
GL_DEPTH_BUFFER_BIT: The depth buffer.

4.2.9 glMatrixMode();

The glMatrixMode function specifies which matrix is the current matrix. SYNTAX: void glMatrixMode(GLenum mode);

where the 'mode' indicates the matrix stack that is the target for subsequent matrix operations. The mode parameter can assume values like GL_PROJECTION which applies subsequent matrix operations to the projection matrix stack or GL_MODELVIEW etc.

4.2.10 glClearColor();

Sets the present RGBA clear color used when clearing the color buffer. Variables of type GLclampf are floating-point numbers between 0.0 and 1.0.

SYNTAX: void glClearColor(GLclampf r, GLclampf g, GLclampfb,GLclampf a);

4.2.11 glFlush();

Forces any buffered OpenGL command to execute. It ensures that points are rendered to the screen as soon as possible.

SYNTAX: void glFlush( );

4.2.12 glutSwapBuffers();

Swaps the front and back buffers.

SYNTAX: void glutSwapBuffers( );

4.2.14 glutInitDisplayMode();

glutInitDisplayMode sets the initial display mode.

SYNTAX: void glutInitDisplayMode(unsigned int mode);

where ‘mode’ is normally the bitwise OR-ing of GLUT display mode bit masks.

4.2.15 glutInitWindowPosition() and glutInitWindowSize();

glutInitWindowPosition and glutInitWindowSize set the initial window position and size respectively.

SYNTAX: void glutInitWindowSize(int width, int height);

void glutInitWindowPosition(int x, int y);

4.2.16 glutCreateWindow();

glutCreateWindow creates a top-level window with the name contained in the string ‘title’. SYNTAX:intglutCreateWindow()

4.2.17 glutMainLoop();

glutMainLoop enters the GLUT event processing loop.

SYNTAX: void glutMainLoop();

4.2.18 glutKeyboardFunc();

Registers the keyboard callback function f in main. The callback function returns the ASCII code of the key pressed and the position of the mouse.

SYNTAX: void glutKeyboardFunc(void *f(int width, int height));

4.2.19 glutDisplayFunc();

glutDisplayFunc sets the display callback for the current window.

SYNTAX: void glutDisplayFunc(void (*func)(void));

4.2.20 glPushMatrix() and glPopMatrix();

They are used to push the transformation matrix onto the stack and to recover later the original condition respectively.

4.2.21 glTranslatef();

This is used to displace an object by a fixed distance in a given direction. dx, dy, dz are the components of the displacement vector.

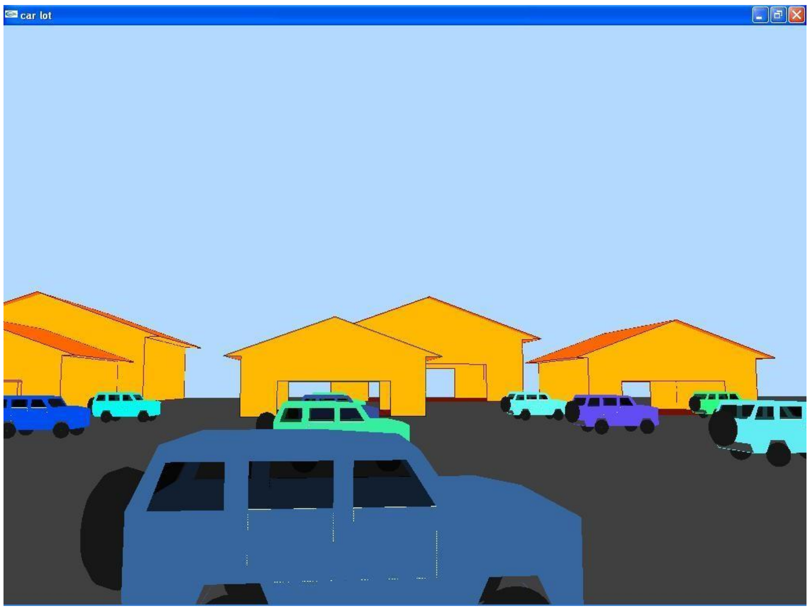
4.2.22 glutPostRedisplay();

Requests that the display callback be executed after the current callback returns.

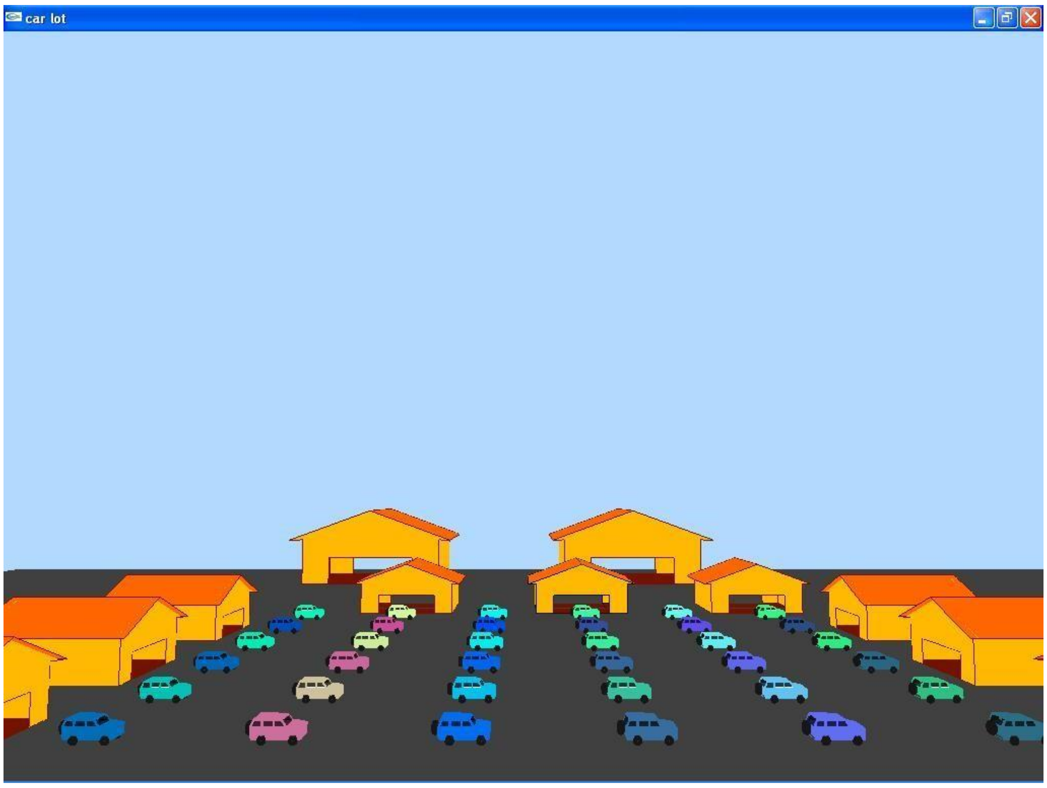
Chapter 5

RESULTS

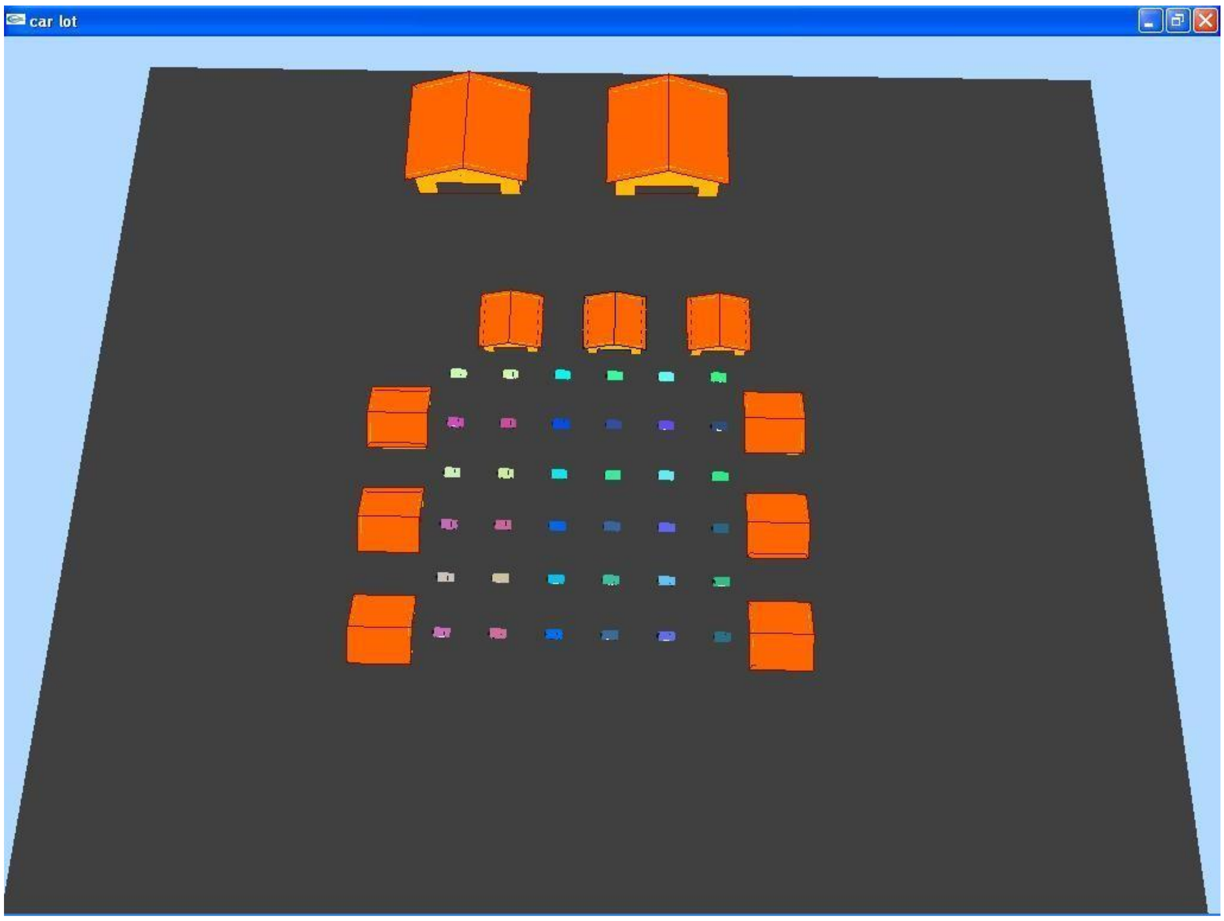
This is the first scene which appears when the program is executed.



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.



Chapter 6

CONCLUSION

The development of a 3D/virtual car park project using computer graphics techniques has provided an immersive and interactive experience for users. The project aimed to create a realistic simulation of a car parking environment, allowing users to navigate the area, closely examine cars, and experience the process of driving and parking a car in designated spaces.

By utilizing the GLUT pre-built models' sub-API in OpenGL, we were able to leverage powerful rendering capabilities and user interaction functionalities. The project successfully incorporated 3D modeling, rendering, and user interaction to create an engaging and educational simulation.

Throughout the project, careful attention was given to the design and layout of the parking area, the creation of realistic car models, and the surrounding environment with houses. The implementation of user controls enabled users to freely explore the virtual car park, while the car parking mechanism allowed them to experience the challenges and techniques involved in parking a car accurately.

The project also focused on providing a visually appealing experience by leveraging OpenGL's rendering capabilities. Realistic visuals, such as shadows, reflections, and lighting, were incorporated to enhance the overall realism and immersion of the simulation.

Performance optimization techniques were employed to ensure smooth and efficient performance, allowing users to interact with the virtual car park seamlessly.

Overall, the 3D/virtual car park project has achieved its objective of creating an interactive and immersive simulation. It serves as an educational tool, providing users with a realistic understanding of car parking and offering an engaging experience in a virtual environment.

Future enhancements to the project could include additional features, such as different parking scenarios, weather conditions, and more realistic physics simulations. User feedback and testing should be incorporated to refine the simulation and improve the user experience further.

In conclusion, the project successfully combines the elements of 3D modeling, rendering, and user interaction to create an educational and engaging virtual car park simulation.

BIBLIOGRAPHY

[1] Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version, 3rd / 4th Edition, Pearson Education, 2011

[2] Edward Angel: Interactive Computer Graphics- A Top-Down approach with OpenGL, 5th edition, Pearson Education, 2008

USER GUIDE

In order to use this simple openGL program, here are the operations:

- Right click to open the pop-up menu or just click on the menu.
- Select any of the options:

1. Click ‘UP KEY’ to move the viewer in forward direction.
2. Click ‘DOWN KEY’ to move the viewer in backwards direction.
3. Click ‘LEFT KEY’ to rotate the camera to the left of the viewer.
4. Click ‘RIGHT KEY’ to rotate the camera to the right of the viewer.
5. Click T for top view.
6. Click S to move away.
7. Click W to move near.
8. Click D to move right.
9. Click A to move left.
10. Click Q to quit.

APPENDIX

Program of Car Parking Game using OpenGL

```
#include<windows.h>
#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]=
{55,97,44,152,55,171,108,86,168,99,147,207,238,55,233,1,105,80,134,29,253,130,32,240,110,199,224,121,93,
199,180,61,110,251,77,237};
int b[36]=
{102,194,110,152,153,184,137,113,55,138,104,43,240,255,203,8,100,53,88,64,127,64,87,5,2,144,211,128,10,8
9,27,11,175,185,157,241};
int c[36]=
{159,243,133,253,233,228,141,18,46,195,75,52,253,204,169,30,78,94,68,117,4,2,33,12,2,25,195,76,26,54,98,10
3,205,173,65,242};
void changeSize(int w, int h)
{
if(h == 0)
h = 1;
ratio = 1.0f * w / h;
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glViewport(0, 0, w, h);
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);
glBlendFunc(GL_ONE, GL_ZERO);
glBegin(GL_LINE_LOOP);
glVertex3f(-1.12,-.48,0.7);
glVertex3f(-0.86,-.48,0.7);
glVertex3f(-.74,-0.2,0.7);
glVertex3f(-.42,-.2,0.7);
glVertex3f(-0.3,-.48,0.7);
glVertex3f(.81,-0.48,0.7);
glVertex3f(.94,-0.2,0.7);
```

```
glVertex3f(1.24,-.2,0.7);
glVertex3f(1.38,-.48,0.7);
glVertex3f(1.52,-.44,0.7);
glVertex3f(1.52,.14,0.7);
glVertex3f(1.14,0.22,0.7);
glVertex3f(0.76,.22,0.7);
glVertex3f(.52,0.56,0.7);
glVertex3f(-0.1,0.6,0.7);
glVertex3f(-1.02,0.6,0.7);
glVertex3f(-1.2,0.22,0.7);
glVertex3f(-1.2,-.28,0.7);
glEnd();
glBegin(GL_LINE_LOOP);
glVertex3f(-1.12,-.48,-0.7);
glVertex3f(-0.86,-.48,-0.7);
glVertex3f(-.74,-0.2,-0.7);
glVertex3f(-.42,-.2,-0.7);
glVertex3f(-0.3,-.48,-0.7);
glVertex3f(.81,-0.48,-0.7);
glVertex3f(.94,-0.2,-0.7);
glVertex3f(1.24,-.2,-0.7);
glVertex3f(1.38,-.48,-0.7);
glVertex3f(1.52,-.44,-0.7);
glVertex3f(1.52,.14,-0.7);
glVertex3f(1.14,0.22,-0.7);
glVertex3f(0.76,.22,-0.7);
glVertex3f(.52,0.56,-0.7);
glVertex3f(-0.1,0.6,-0.7);
glVertex3f(-1.02,0.6,-0.7);
glVertex3f(-1.2,0.22,-0.7);
glVertex3f(-1.2,-.28,-0.7);
glEnd();
glBegin(GL_LINES);
glVertex3f(-1.12,-.48,0.7);
glVertex3f(-1.12,-.48,-0.7);
glVertex3f(-0.86,-.48,0.7);
glVertex3f(-0.86,-.48,-0.7);
glVertex3f(-.74,-0.2,0.7);
glVertex3f(-.74,-0.2,-0.7);
glVertex3f(-.42,-.2,0.7);
glVertex3f(-.42,-.2,-0.7);
glVertex3f(-0.3,-.48,0.7);
glVertex3f(-0.3,-.48,-0.7);
glVertex3f(.81,-0.48,0.7);
glVertex3f(.81,-0.48,-0.7);
glVertex3f(.94,-0.2,0.7);
glVertex3f(.94,-0.2,-0.7);
glVertex3f(1.24,-.2,0.7);
glVertex3f(1.24,-.2,-0.7);
glVertex3f(1.38,-.48,0.7);
```

```
glVertex3f(1.38,-.48,-0.7);
glVertex3f(1.52,-.44,0.7);
glVertex3f(1.52,-.44,-0.7);
glVertex3f(1.52,.14,0.7);
glVertex3f(1.52,.14,-0.7);
glVertex3f(1.14,0.22,0.7);
glVertex3f(1.14,0.22,-0.7);
glVertex3f(0.76,.22,0.7);
glVertex3f(0.76,.22,-0.7);
glVertex3f(.52,0.56,0.7);
glVertex3f(.52,0.56,-0.7);
glVertex3f(-0.1,0.6,0.7);
glVertex3f(-0.1,0.6,-0.7);
glVertex3f(-1.02,0.6,0.7);
glVertex3f(-1.02,0.6,-0.7);
glVertex3f(-1.2,0.22,0.7);
glVertex3f(-1.2,0.22,-0.7);
glVertex3f(-1.2,-.28,0.7);
glVertex3f(-1.2,-.28,-0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-0.1,0.6,0.7);
glVertex3f(-0.1,0.6,-0.7);
glVertex3f(-1.02,0.6,-0.7);
glVertex3f(-1.02,0.6,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-0.1,0.6,0.7);
glVertex3f(-0.1,0.6,-0.7);
glVertex3f(.52,0.56,-0.7);
glVertex3f(.52,0.56,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-1.2,0.22,0.7);
glVertex3f(-1.2,0.22,-0.7);
glVertex3f(-1.2,-.28,-0.7);
glVertex3f(-1.2,-.28,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(1.52,.14,0.7);
glVertex3f(1.14,0.22,0.7);
glVertex3f(1.14,0.22,-0.7);
glVertex3f(1.52,.14,-0.7);
glEnd(); glBegin(GL_POLYGON);
glVertex3f(0.76,.22,0.7);
glVertex3f(0.76,.22,-0.7);
glVertex3f(1.14,0.22,-0.7);
glVertex3f(1.14,0.22,0.7);
glEnd();
glBegin(GL_POLYGON);
```

```
glVertex3f(-1.12,-.48,0.7);
glVertex3f(-0.86,-.48,0.7);
glVertex3f(-.74,-0.2,0.7);
glVertex3f(-0.64,0.22,0.7);
glVertex3f(-1.08,0.22,0.7);
glVertex3f(-1.2,0.22,0.7);
glVertex3f(-1.2,-.28,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-.74,-0.2,0.7);
glVertex3f(-0.64,0.22,0.7);
glVertex3f(-0.5,0.22,0.7);
glVertex3f(-0.5,-0.2,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(0.0,0.22,0.7);
glVertex3f(1.14,0.22,0.7);
glVertex3f(1.24,-.2,0.7);
glVertex3f(0.0,-0.2,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-1.12,-.48,-0.7);
glVertex3f(-0.86,-.48,-0.7);
glVertex3f(-.74,-0.2,-0.7);
glVertex3f(-0.64,0.22,-0.7);
glVertex3f(-1.08,0.22,-0.7);
glVertex3f(-1.2,0.22,-0.7);
glVertex3f(-1.2,-.28,-0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-.74,-0.2,-0.7);
glVertex3f(-0.64,0.22,-0.7);
glVertex3f(-0.5,0.22,-0.7);
glVertex3f(-0.5,-0.2,-0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(0.0,0.22,-0.7);
glVertex3f(1.14,0.22,-0.7);
glVertex3f(1.24,-.2,-0.7);
glVertex3f(0.0,-0.2,-0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-1.2,0.22,0.7);
glVertex3f(-1.08,0.22,0.7);
glVertex3f(-0.98,0.5,0.7);
glVertex3f(-1.02,0.6,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-1.02,0.6,0.7);
glVertex3f(-0.98,0.5,0.7);
```

```
glVertex3f(0.44,0.5,0.7);
glVertex3f(.52,0.56,0.7);
glVertex3f(-0.1,0.6,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-0.64,0.5,0.7);
glVertex3f(-0.64,0.22,0.7);
glVertex3f(-0.5,0.22,0.7);
glVertex3f(-0.5,0.5,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(0.0,0.5,0.7);
glVertex3f(0.0,0.22,0.7);
glVertex3f(0.12,0.22,0.7);
glVertex3f(0.12,0.5,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(.52,0.56,0.7);
glVertex3f(0.44,0.5,0.7);
glVertex3f(0.62,0.22,0.7);
glVertex3f(0.76,.22,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-.42,-.2,0.7);
glVertex3f(.94,-0.2,0.7);
glVertex3f(.81,-0.48,0.7);
glVertex3f(-0.3,-.48,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(1.14,0.22,0.7);
glVertex3f(1.52,.14,0.7);
glVertex3f(1.52,-.44,0.7);
glVertex3f(1.38,-.48,0.7);
glVertex3f(1.24,-.2,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-1.2,0.22,-0.7);
glVertex3f(-1.08,0.22,-0.7);
glVertex3f(-0.98,0.5,-0.7);
glVertex3f(-1.02,0.6,-0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-1.02,0.6,-0.7);
glVertex3f(-0.98,0.5,-0.7);
glVertex3f(0.44,0.5,-0.7);
glVertex3f(.52,0.56,-0.7);
glVertex3f(-0.1,0.6,-0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-0.64,0.5,-0.7);
```

```
glVertex3f(-0.64,0.22,-0.7);
glVertex3f(-0.5,0.22,-0.7);
glVertex3f(-0.5,0.5,-0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(0.0,0.5,-0.7);
glVertex3f(0.0,0.22,-0.7);
glVertex3f(0.12,0.22,-0.7);
glVertex3f(0.12,0.5,-0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(.52,0.56,-0.7);
glVertex3f(0.44,0.5,-0.7);
glVertex3f(0.62,0.22,-0.7);
glVertex3f(0.76,.22,-0.7);
glEnd();  glBegin(GL_POLYGON);
glVertex3f(-.42,-.2,-0.7);
glVertex3f(.94,-0.2,-0.7);
glVertex3f(.81,-0.48,-0.7);
glVertex3f(-0.3,-.48,-0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(1.14,0.22,-0.7);
glVertex3f(1.52,.14,-0.7);
glVertex3f(1.52,-.44,-0.7);
glVertex3f(1.38,-.48,-0.7);
glVertex3f(1.24,-.2,-0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-0.5,0.22,0.7);
glVertex3f(0.0,0.22,0.7);
glVertex3f(0.0,-0.2,0.7);
glVertex3f(-0.5,-0.2,0.7);
glEnd(); glBegin(GL_POLYGON);
glVertex3f(-0.5,0.22,-0.7);
glVertex3f(0.0,0.22,-0.7);
glVertex3f(0.0,-0.2,-0.7);
glVertex3f(-0.5,-0.2,-0.7);
glEnd(); glBegin(GL_POLYGON);
glVertex3f(0.12,0.22,0.7);
glVertex3f(0.62,0.22,0.7);glVertex3f(0.62,-0.2,0.7);
glVertex3f(0.12,-0.2,0.7);
glEnd(); glBegin(GL_POLYGON);
glVertex3f(0.12,0.22,-0.7);
glVertex3f(0.62,0.22,-0.7);
glVertex3f(0.62,-0.2,-0.7);
glVertex3f(0.12,-0.2,-0.7);
glEnd(); glBegin(GL_POLYGON);
glVertex3f(1.52,.14,0.7);
glVertex3f(1.52,.14,-0.7);
```

```
glVertex3f(1.52,-.44,-0.7);
glVertex3f(1.52,-.44,0.7);
glEnd();
glTranslatef(-.58,-.52,0.7);
glColor3f(0.09,0.09,0.09);
glutSolidTorus(0.12f, .14f, 10, 25);
glTranslatef(1.68,0.0,0.0);
glutSolidTorus(0.12f, .14f, 10, 25);
glTranslatef(0.0,0.0,-1.4);
glutSolidTorus(0.12f, .14f, 10, 25);
glTranslatef(-1.68,0.0,0.0);
glutSolidTorus(0.12f, .14f, 10, 25);
glTranslatef(.58,.52,0.7);
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);
glColor3f(0.25,0.25,0.25);
glVertex3f(-0.3,-.48,0.7);
glVertex3f(-0.3,-.48 ,-0.7);
glVertex3f(.81,-0.48,-0.7);
glVertex3f(.81,-0.48,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-.42,-.2,0.7);
glVertex3f(-.42,-.2,-0.7);
glVertex3f(-0.3,-.48,-0.7);
glVertex3f(-0.3,-.48,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-1.2,-.28,0.7);
glVertex3f(-1.2,-.28,-0.7);
glVertex3f(-1.12,-.48,-0.7);
glVertex3f(-1.12,-.48,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-1.12,-.48,0.7);
glVertex3f(-1.12,-.48,-0.7);
glVertex3f(-0.86,-.48,-0.7);
glVertex3f(-0.86,-.48,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-0.86,-.48,0.7);
glVertex3f(-0.86,-.48,-0.7);
glVertex3f(-.74,-0.2,-0.7);
glVertex3f(-.74,-0.2,0.7);
glEnd();
glBegin(GL_POLYGON);
```

```
glVertex3f(-.74,-0.2,0.7);
glVertex3f(-.74,-0.2,-0.7);
glVertex3f(-.42,-.2,-0.7);
glVertex3f(-.42,-.2,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(.81,-0.48,0.7);
glVertex3f(.81,-0.48,-0.7);
glVertex3f(.94,-0.2,-0.7);
glVertex3f(.94,-0.2,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(.94,-0.2,0.7);
glVertex3f(.94,-0.2,-0.7);
glVertex3f(1.24,-.2,-0.7);
glVertex3f(1.24,-.2,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(1.24,-.2,0.7);
glVertex3f(1.24,-.2,-0.7);
glVertex3f(1.38,-.48,-0.7);
glVertex3f(1.38,-.48,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(1.38,-.48,0.7);
glVertex3f(1.38,-.48,-0.7);
glVertex3f(1.52,-.44,-0.7);
glVertex3f(1.52,-.44,0.7);
glEnd();
glBegin(GL_LINE_LOOP);
glColor3f(1.0,1.0,1.0);
glVertex3f(-0.5,0.22,0.7);
glVertex3f(0.0,0.22,0.7);
glVertex3f(0.0,-0.2,0.7);
glVertex3f(-0.5,-0.2,0.7);
glEnd();
glBegin(GL_LINE_LOOP);
glVertex3f(0.12,0.22,0.7);
glVertex3f(0.62,0.22,0.7);
glVertex3f(0.62,-0.2,0.7);
glVertex3f(0.12,-0.2,0.7);
glEnd();
glColor3f(0.0,0.0,0.0);
glBegin(GL_LINE_LOOP);
glVertex3f(0.12,0.22,-0.7);
glVertex3f(0.62,0.22,-0.7);
glVertex3f(0.62,-0.2,-0.7);
glVertex3f(0.12,-0.2,-0.7);
glEnd();
glBegin(GL_LINE_LOOP);
```

```
glVertex3f(-0.5,0.22,-0.7);
glVertex3f(0.0,0.22,-0.7);
glVertex3f(0.0,-0.2,-0.7);
glVertex3f(-0.5,-0.2,-0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(1.52,.14,0.7);
glVertex3f(1.52,.14,-0.7);
glVertex3f(1.52,-.44,-0.7);
glVertex3f(1.52,-.44,0.7);
glEnd();
glColor3f(0.0,0.0,1.0);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glBegin(GL_POLYGON);
glColor4f(0.0,0.0,0.0,0.7);
glVertex3f(0.562,.5,.6);
glVertex3f(.562,.5,-.6);
glVertex3f(.76,.22,-.6);
glVertex3f(.76,.22,.6);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-1.068,0.5,0.6);
glVertex3f(-1.068,0.5,-0.6);
glVertex3f(-1.2,0.22,-0.6);
glVertex3f(-1.2,0.22,0.6);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-0.98,0.5,0.7);
glVertex3f(-0.64,0.5,0.7);
glVertex3f(-0.64,0.22,0.7);
glVertex3f(-1.08,0.22,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(-0.98,0.5,-0.7);
glVertex3f(-0.64,0.5,-0.7);
glVertex3f(-0.64,0.22,-0.7);
glVertex3f(-1.08,0.22,-0.7);
glEnd();
glBegin(GL_POLYGON);
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);
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);
```

```
glVertex3f(0.12,0.5,0.7);
glVertex3f(0.44,0.5,0.7);
glVertex3f(0.62,0.22,0.7);
glVertex3f(0.12,0.22,0.7);
glEnd();
glBegin(GL_POLYGON);
glVertex3f(0.12,0.5,-0.7);
glVertex3f(0.44,0.5,-0.7);
glVertex3f(0.62,0.22,-0.7);
glVertex3f(0.12,0.22,-0.7);
glEnd();
glColor3f(0.0,0.0,1.0);
}
void drawhouse()
{
glBegin(GL_LINE_LOOP);
glVertex3f(-2.6,-.84,2.5);
glVertex3f(-2.6,0.84,2.5);
glVertex3f(-3.04,0.84,2.8);
glVertex3f(0,1.95,2.8);
glVertex3f(3.04,0.84,2.8);
glVertex3f(2.6,0.84,2.5);
glVertex3f(2.6,-0.84,2.5);
glVertex3f(1.59,-0.84,2.5);
glVertex3f(1.59,0.16,2.5);
glVertex3f(-1.59,0.16,2.5);
glVertex3f(-1.59,-0.84,2.5);
glEnd();
glBegin(GL_LINES);
glVertex3f(1.59,-0.84,2.5);
glVertex3f(-1.59,-0.84,2.5);
glEnd();
glBegin(GL_LINE_LOOP);
glVertex3f(-2.6,-.84,-2.5);
glVertex3f(-2.6,0.84,-2.5);
glVertex3f(-3.04,0.84,-2.8);
glVertex3f(0,1.95,-2.8);
glVertex3f(3.04,0.84,-2.8);
glVertex3f(2.6,0.84,-2.5);
glVertex3f(2.6,-0.84,-2.5);
glVertex3f(1.59,-0.84,-2.5);
glVertex3f(1.59,0.16,-2.5);
glVertex3f(-1.59,0.16,-2.5);
glVertex3f(-1.59,-0.84,-2.5);
glEnd();
glBegin(GL_ LINES);
glVertex3f(-2.6,-.84,2.5);
glVertex3f(-2.6,-.84,-2.5);
glVertex3f(-2.6,0.84,2.5);
glVertex3f(-2.6,0.84,-2.5);
```

```
glVertex3f(-3.04,0.84,2.8);
glVertex3f(-3.04,0.84,-2.8);
glVertex3f(0,1.95,2.8);
glVertex3f(0,1.95,-2.8);
glVertex3f(3.04,0.84,2.8);
glVertex3f(3.04,0.84,-2.8);
glVertex3f(2.6,0.84,2.5);
glVertex3f(2.6,0.84,-2.5);
glVertex3f(2.6,-0.84,2.5);
glVertex3f(2.6,-0.84,-2.5);
glVertex3f(1.59,-0.84,2.5);
glVertex3f(1.59,-0.84,-2.5);
glVertex3f(-1.59,-0.84,2.5);
glVertex3f(-1.59,-0.84,-2.5);
glEnd();
glColor3ub(255,185,1);
glBegin(GL_QUADS);
glVertex3f(-2.6,-.84,2.5);
glVertex3f(-2.6,0.16,2.5);
glVertex3f(-1.59,0.16,2.5);
glVertex3f(-1.59,-0.84,2.5);
glVertex3f(-2.6,0.16,2.5);
glVertex3f(-2.6,0.84,2.5);
glVertex3f(2.6,0.84,2.5);
glVertex3f(2.6,0.16,2.5);
glVertex3f(1.59,-0.84,2.5);
glVertex3f(1.59,0.16,2.5);
glVertex3f(2.6,0.16,2.5);
glVertex3f(2.6,-0.84,2.5);
glVertex3f(-2.6,-.84,-2.5);
glVertex3f(-2.6,0.16,-2.5);
glVertex3f(-1.59,0.16,-2.5);
glVertex3f(-1.59,-0.84,-2.5);
glVertex3f(-2.6,0.16,-2.5);
glVertex3f(-2.6,0.84,-2.5);
glVertex3f(2.6,0.84,-2.5);
glVertex3f(2.6,0.16,-2.5);
glVertex3f(1.59,-0.84,-2.5);
glVertex3f(1.59,0.16,-2.5);
glVertex3f(2.6,0.16,-2.5);
glVertex3f(2.6,-0.84,-2.5);
glVertex3f(-2.6,-.84,2.5);
glVertex3f(-2.6,-.84,-2.5);
glVertex3f(-2.6,0.84,-2.5);
glVertex3f(-2.6,0.84,2.5);
glVertex3f(2.6,0.84,2.5);
glVertex3f(2.6,0.84,-2.5);
glVertex3f(2.6,-0.84,-2.5);
glVertex3f(2.6,-0.84,2.5);
glEnd();
```

```
glBegin(GL_TRIANGLES);
glVertex3f(0,1.95,2.5);
glVertex3f(3.04,0.84,2.5);
glVertex3f(-3.04,0.84,2.5);
glVertex3f(0,1.95,-2.5);
glVertex3f(3.04,0.84,-2.5);
glVertex3f(-3.04,0.84,-2.5);
glEnd();
glColor3ub(255,102,0);
glBegin(GL_QUADS);
glVertex3f(0,1.95,2.8);
glVertex3f(0,1.95,-2.8);
glVertex3f(3.04,0.84,-2.8);
glVertex3f(3.04,0.84,2.8);
glVertex3f(-3.04,0.84,2.8);
glVertex3f(-3.04,0.84,-2.8);
glVertex3f(0,1.95,-2.8);
glVertex3f(0,1.95,2.8);
glEnd();
glColor3ub(116,18,0);
glBegin(GL_QUADS);
glVertex3f(-2.6,-.84,2.5);
glVertex3f(2.6,-0.84,2.5);
glVertex3f(2.6,-0.84,-2.5);
glVertex3f(-2.6,-.84,-2.5);
glEnd();
}
GLuint createDL()
{
  GLuint carrDL;
carrDL = glGenLists(1);
glNewList(carrDL,GL_COMPILE);
drawcarr();
glEndList();
return(carrDL);
}
GLuint createDL2()
{
  GLuint houseDL;
houseDL = glGenLists(1);
glNewList(houseDL,GL_COMPILE);
drawhouse();
glEndList();
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);
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++)
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)
{
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("UP KEY : Move in Forward Direction.",1);
glutAddMenuEntry("DOWN KEY: Move  in Backward Direction.",1);
glutAddMenuEntry("  LEFT KEY:to Turn Left .",1);
glutAddMenuEntry("  RIGHT KEY:to Turn Right .",1);
glutAddMenuEntry("  d:moves Towards Right. ",1);
glutAddMenuEntry(" a:moves Towards Left.",1);
glutAddMenuEntry("  s:moves Away.",1);
glutAddMenuEntry(" w:moves Near.",1);
glutAddMenuEntry(" t:Top view.",1);
glutAddMenuEntry(" 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);
}
```