VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"Jnana Sangama", Belagavi-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY
K. R. Road, V. V. Puram, Bengaluru-560 004

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Computer Graphics Laboratory With Mini Project Report-18CSL67
on

"CAR PARKING "

Submitted By

1BI20CS153	**SATYAM KUMAR**

for the academic year 2022-23

Under the guidance of

Dr. Bhanushree K J	**Prof. Tejashwini P S**
Associate Professor	Assistant Professor

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
“Jnana Sangama”, Belagavi-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY
K. R. Road, V. V. Puram, Bengaluru-560 004

Department of Computer Science & Engineering

Certificate

This is to certify that the implementation of the **Computer Graphics Laboratory With Mini Project (18CSL67)** entitled **“CAR PARKING ”** has been successfully completed by

1BI20CS153	**SATYAM KUMAR**

of VI semester B.E. for the partial fulfillment of the requirements for the Bachelor's degree in **Computer Science & Engineering** of the **Visvesvaraya Technological University** during the academic year **2022-2023**.

Lab In charges:

Dr. Bhanushree K.J	**Prof. Tejashwini P S**	**Dr. Girija J.**
Associate Professor	Assistant Professor	Professor and Head
Dept. of CSE, BIT	Dept. of CSE, BIT	Dept. of CSE, BIT

Examiners: 1) 2)

ACKNOWLEDGEMENT

The knowledge & satisfaction that accompany the successful completion of any task would be incomplete without mention of the people who made it possible, whose guidance and encouragement crowned my effort with success. I would like to thank all and acknowledge the help I have received to carry out this Mini Project.

I would like to convey my sincere thanks to **Dr. M. U. Aswath**, Principal, BIT, and **Dr.Girija J .**, HOD, Department of CS&E, BIT for being kind enough to provide the necessary support to carry out the mini project.

I am most humbled to mention the enthusiastic influence provided by the lab in-charges **Dr. Bhanushree K. J. and Prof. Tejashwini P S,** on the project for their ideas, time to-time suggestions for being a constant guide and co-operation showed during the venture and making this project a great success.

I would also take this opportunity to thank my friends and family for their constant support and help. I'm very pleased to express my sincere gratitude to the friendly co-operation of all the **staff members** of Computer Science Department, BIT.

SATYAM KUMAR
1BI20CS153

Table of contents

Chapter -1

INTRODUCTION

1.1 Computer Graphics

Computer graphics is the art of drawing pictures, lines, and charts, using computers with the help of programming. Computer graphics is made up of a number of pixels. A pixel is the smallest graphical picture or unit represented on the computer screen. Basically, there are 2 types of computer graphics namely,

Interactive Computer Graphics involves two-way communication between the computer and user. The observer is given some control over the image by providing him with an input device. This helps him to signal his request to the computer.

Non-Interactive Computer Graphics, otherwise known as passive computer graphics, is the computer graphics in which the user has no control over the image. Image is merely the product of a static stored program and will work according to the instructions given in the program linearly. The image is totally under the control of program instructions not under the user. Example: screen savers.

1.2 Applications of Computer Graphics

Scientific Visualization

Scientific visualization is a branch of science, concerned with the visualization of three-dimensional phenomena, such as architectural, meteorological, medical, biological systems.

Graphic Design

The term graphic design can refer to a number of artistic and professional disciplines which focus on visual communication and presentation

Computer-aided Design

Computer-aided design (CAD) is the use of computer technology for the design of objects, real or virtual. The design of geometric models for object shapes, in particular, is often called computer-aided geometric design (CAGD). The manufacturing process is tied into the computer description of the designed objects so that the fabrication of a product can be automated using methods that are referred to as CAM, computer-aided manufacturing.

Web Design

Web design is the skill of designing presentations of content usually hypertext or hypermedia that is delivered to an end-user through the World Wide Web, by way of a Web browser.

Digital Art

Digital art most commonly refers to art created on a computer in digital form.

Video Games

A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a raster display device.

Virtual Reality

Virtual reality (VR) is a technology which allows a user to interact with a computer simulated environment. The simulated environment can be similar to the real world. This allows the designer to explore various positions of an object. Animations in virtual reality environments are used to train heavy equipment operators or to analyse the effectiveness of various cabin configurations and control placements.

Computer Simulation

A computer simulation, a computer model or a computational model is a computer program, or network of computers, that attempts to simulate an abstract model of a particular system.

Education and Training

Computer simulations have become a useful part of mathematical modelling of many natural systems in physics, chemistry and biology, human systems in economics, psychology, and social science and in the process of engineering new technology, to gain insight into the operation of those systems, or to observe their behaviour. Most simulators provide screens for visual display of the external environment with multiple panels is mounted in front of the simulator.

Image Processing

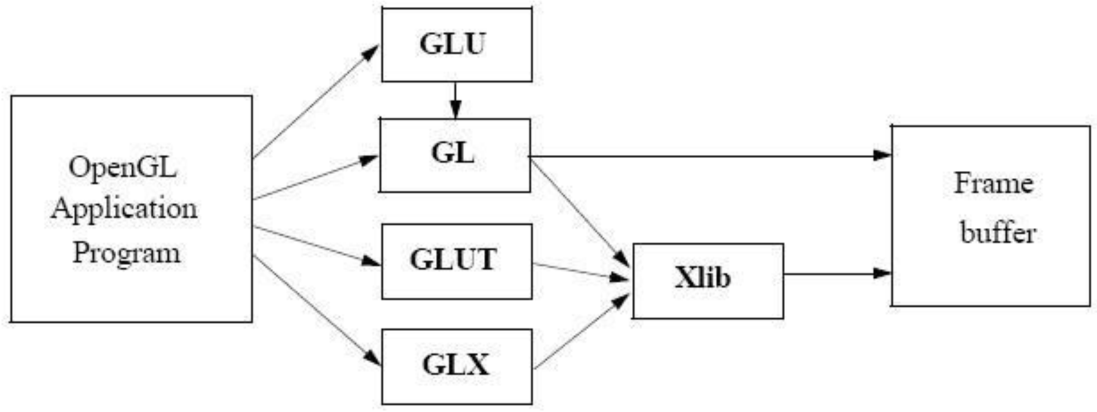
The modification or interpretation of existing pictures such as photographs and TV scans, is called image processing. In computer graphics, a computer is used to create a picture. Image processing techniques, on the other hand, are used to improve picture quality, analyse images, or recognize visual patterns for robotics applications

1.3 OpenGL

OpenGL has become a widely accepted standard for developing graphics applications. Most of our applications will be designed to access OpenGL directly through functions in the three libraries. Functions in main GL libraries have names that begin with the letters gl and are stored in a library usually referred to as GL.

The second is the OpenGL Utility Library (GLU). This library uses only GL functions but contains code for creating common objects and simplifying viewing. All function in GLU can be created from the core GL library. The GLU library is available in all OpenGL implementations. Functions in the GLU library starts with the letters glu.

The third is the OpenGL Utility Toolkit (GLUT). It provides the minimum functionality



that should be formulated in modern windowing systems.

Figure 1.1 Basic block diagram of OpenGL

1.4 Problem Statement

Design an efficient car parking system that manages parking spaces, assigns slots, and tracks availability for smooth vehicle entry and exit.

1.5 Objectives of the Project

- Optimize parking space utilization by efficiently assigning and managing parking slots.
- Implement a user-friendly interface for drivers to check parking availability and book slots.
- Ensure smooth flow of vehicles by providing real-time updates on occupied and available spaces.
- Generate reports and analytics to analyze parking patterns and improve overall system efficiency.

1.6 Organisation of the Project

The project was organized in a systematic way. First, we analyzed what are the basic features to be included in the project to make it acceptable. As it is an graphics-oriented project, we made the sketches prior, so as to have an idea like how our output must look like. After all these, the source code was formulated as a paperwork. All the required software were downloaded. Finally, the successful implementation of the project.

Chapter -2

SYSTEM SPECIFICATION

2.1 Hardware Requirements

- Main Processor: PENTIUM IV
- Processor Speed: 800 MHz
- RAM Size: 128 MB DDR
- Keyboard: Standard qwerty serial or PS/2 keyboard
- Mouse: Standard serial or PS/2 mouse
- Compatibility: AT/T Compatible
- Cache memory: 256 KB
- Diskette drive: 1,44MB,3.5 inches

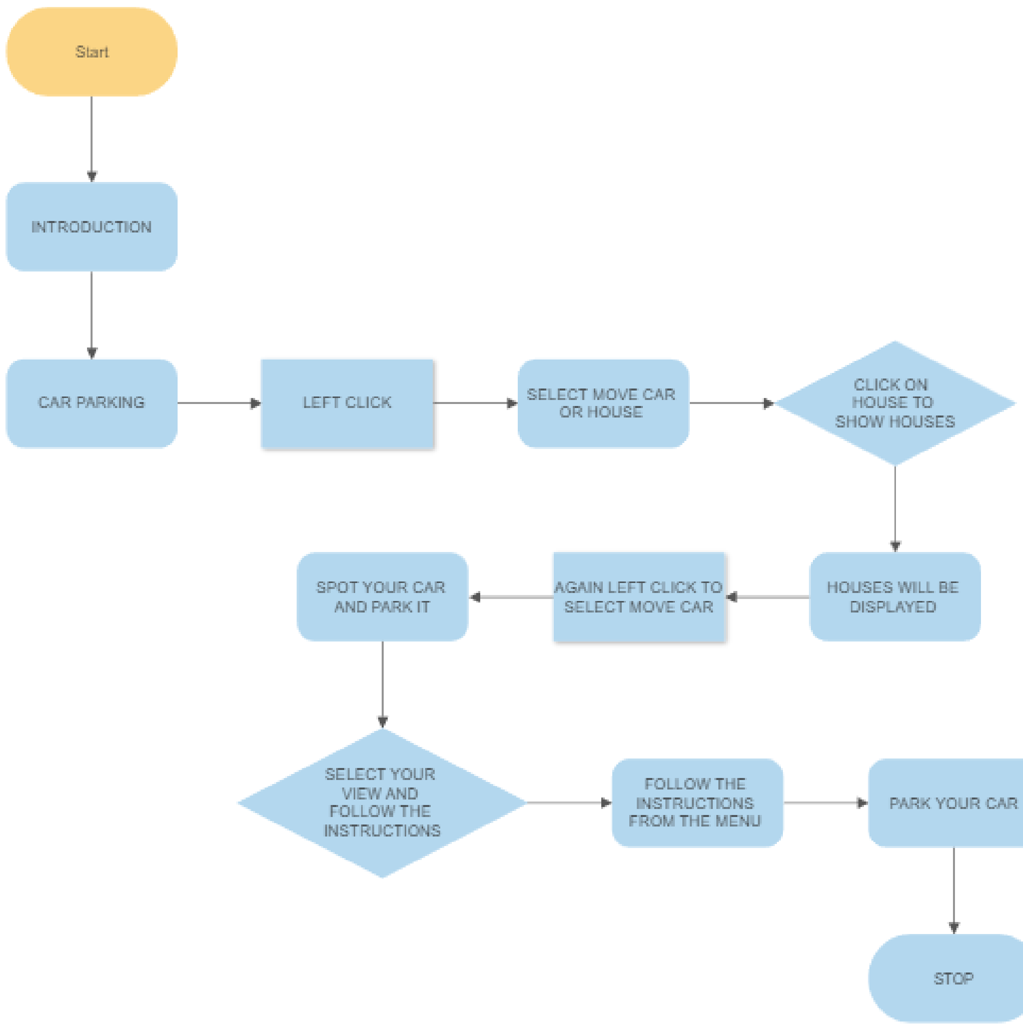
2.2 Software Requirements

- Operating System: Windows 10 or Linux (Fedora) or macOS
- Hypervisor used: Docker
- Compiler used: g++
- Language used: C++ language
- Terminal : Ubuntu
- Toolkit: GLUT Toolkit

Chapter -3

DESIGN

3.1 Flow Diagram


Start
INTRODUCTION
CAR PARKING
LEFT CLICK
SELECT MOVE CAR OR HOUSE
CLICK ON HOUSE TO SHOW HOUSES
HOUSES WILL BE DISPLAYED
AGAIN LEFT CLICK TO SELECT MOVE CAR
SPOT YOUR CAR AND PARK IT
SELECT YOUR VIEW AND FOLLOW THE INSTRUCTIONS
FOLLOW THE INSTRUCTIONS FROM THE MENU
PARK YOUR CAR
STOP

3.2 Description of Flow Diagram

The description of the flow diagram is as follows:

Step 1: Start

Step 2: The user is presented with the car parking and the user needs to park his car.

Step 3: The user presses left click to show the arrangement of houses.

Step 4: The user is asked to select the view in which he would like to see the parking space and park his car.

Step 5: The user is promted to search for his car and drive it to parking space

Step 6: On right Click various other menu options are being displayed which will help the user to drive the car.

Step 7: After carefully examining the ground and instructions the driver starts driving .

Step 8: With the help of instructions the user is able to park his vehicle.

Step 9: Make sure to park the vehicle without damaging the other vehicle.

Step 10: Stop.

Chapter -4

IMPLEMENTATION

4.1 Built-in Functions

1. glutInit()

glutInit is used to initialize the GLUT library.
Usage: void glutInit (int *argc, char **argv);
Description: glutInit will initialize the GLUT library and negotiate a session with the window system.

2. glutInitDisplayMode()

glutInitDisplayMode sets the initial display mode.
Usage: void glutInitDisplayMode (unsigned int mode);
Mode-Display mode, normally the bitwise OR-ing GLUT display mode bit masks.
Description: 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.

3. glutCreateWindow()

glutCreateWindow creates a top-level window.

Usage: intglutCreateWindow (char *name); Name-ASCII character string for use as window name .

Description: glutCreateWindow creates a top-level window. The name will be provided to the window system as the window's name. The intent is that the window system will label the window with the name. Implicitly, the current window is set to the newly created window.

Each created window has a unique associated OpenGL context.

4. glutDisplayFunc()

glutDisplayFunc sets the display callback for the current window.
Usage: void glutDisplayFunc (void(*func)(void));
Description: glutDisplayFunc sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called. Before the callback, the current window is set to the window needing to be redisplayed and the layer in use is set to the normal plane. The display callback is called with no parameters. The entire normal plane region should be redisplayed in response to the callback.

5. glutMainLoop()

glutMainLoop enters the GLUT event processing loop.

Usage: void glutMainLoop(void);

Description: glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

6. glMatrixMode()

The two most important matrices are the model-view and projection matrix. At many times, the state includes values for both of these matrices, which are initially set to identity matrices. There is only a single set of functions that can be applied to any type of matrix. Select the matrix to which the operations apply by first set in the matrix mode, a variable that is set to one type of matrix and is also part of the state.

7. glTranslate(GLfloat X, GLfloat Y, GLfloat Z)

glTranslate produces a translation by x y z. If the matrix mode is either GL_MODEL_VIEW or GL_PROJECTION, all objects drawn after a call to glTranslate are translated.

8. glRotatef(GLdouble angle, GLdouble X, GLdouble Y, GLdouble Z)

glRotatef produces a rotation of angle degrees around the vector x y z. If the matrix mode is either GL_MODEL_VIEW or GL_PROJECTION, all objects drawn after glRotatef is called are rotated. Use glPushMatrix() and glPopmatrix() to save and restore the unrotated coordinate system.

9. glPushMatrix()

There is a stack of matrices for each of the matrix mode. In GL_MODELVIEW mode, the stack depth is atleast 32. In other modes, GL_COLOR, GL_PROJECTION, and GL_TEXTURE, the depth is atleast 2. The current matrix in any mode is the matrix on the top of the stack for that mode.

10. glPopMatrix()

glPopMatrix pops the current matrix stack, replacing the current matrix with the one below it on the stack. Initially, each of the stack contains one matrix, an identity matrix. It is an error to push a full matrix stack or pop a matrix stack that contains only a single matrix. In either case, the error flag is set and no other change is made to GL state.

11. glutSwapBuffers()

Usage**:** void glutSwapBuffers(void);

Description: Performs a buffer swap on the layer in use for the current window. Specifically, glutSwapBuffers promotes the contents of the front buffer. The contents of the back buffer then become undefined.

12. glPointSize(GLfloat size)

glPointSize specifies the rasterized diameter of points. This value will be used rasterize points. Otherwise, the value written to the shading language built-in variable gl-PointSize will be used. The point size specified by glPointSize is always returned when GL_POINT_SIZE is queried.

13. glutKeyboardFunc()

Usage**:** void glutKeyboardFunc(void(*func)(unsigned char key, int x, int y)

Func**:** The new keyboard callback function

Description**:** 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.

14. glLineWidth(GLfloat width)

Parameters**:** width- Specifies the width of rasterized lines. The initial value is 1. Description**:** glLineWidth specifies the rasterized width of lines. The actual width is determined by rounding the supplied width to the nearest integer. i pixels are filled in each column that is rasterized, where I is the rounded value of width.

15. glLoadIdentity(void)

glLoadIdentity replaces the current matrix with the identity matrix. It is semantically equivalent to calling glLoadMatrix with the identity matrix.

16.glutSolidTorus

glut Solid Torus is a function in OpenGL that generates a solid torus shape in computer graphics**.**

4.2 User-Defined Functions

1. void changeSize(int w,int h)

This code handles window resizing, adjusts rendering settings, sets perspective, and positions the camera in OpenGL.

2. void drawcarr()

The code snippet appears to be drawing a 3D polygon with transparency and multiple vertices using OpenGL.

3. void drawhouse()

The code is using OpenGL to draw a 3D polygon with various vertices, lines, and colors for visualization purposes.

4. GLuint createDL()

The code creates a display list in OpenGL, generates a unique ID for it, and compiles the rendering commands for a car model into the list.

5. void initScene()

The code initializes the scene by enabling depth testing and creating display lists for a car and a house model.

6. void renderScene(void)

The code renders the scene by clearing buffers, drawing the ground, cars, and houses, and applying transformations and rotations.

7. void orientMe(float ang)

The function sets the orientation of the viewer by updating the viewing direction and using gluLookAt to position the viewer.

8. void moveMeFlat(int i)

The function moves the viewer horizontally or vertically based on input, adjusting the viewer's position and updating the view using gluLookAt.

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

The function handles normal keyboard input and performs corresponding actions, such as exiting the program or moving the viewer's position.

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

The function handles special keyboard input (arrow keys) and performs corresponding actions, such as rotating the viewer or moving them forward/backward.

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

The function moves a car in the scene based on the arrow key input by updating its position and orientation.

12. void ProcessMenu(int value)

The function posts a redisplay event to update the screen in response to a menu selection

13. void ProcessMenu1(int value)

The function posts a redisplay event to update the screen in response to a menu selection

14. void menu()

The code creates two menus, one for controls and another for selecting actions related to a house or moving a car.

15. int main(int argc, char **argv)

The code initializes the GLUT library, creates a window, sets up callbacks, and enters the main event loop.

4.3 PSUEDOCODE

main.cpp

```
#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, 11, 7, 4, 2, 33, 12, 2, 25, 195, 76, 26, 54, 98,
103, 205, 173, 65};

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 yre
    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);
}
```

Chapter -5

SNAPSHOTS

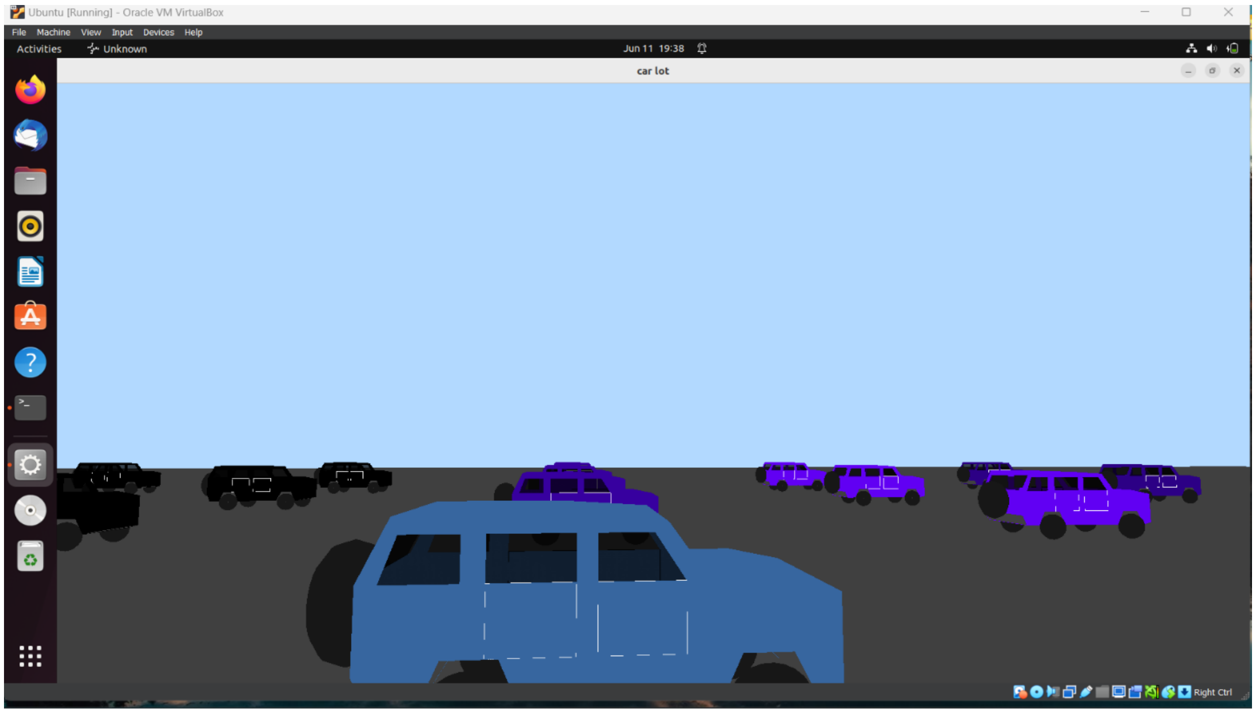

Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Unknown
Jun 11 19:38
car lot
Right Ctrl

Figure 5.1 : Intro scene

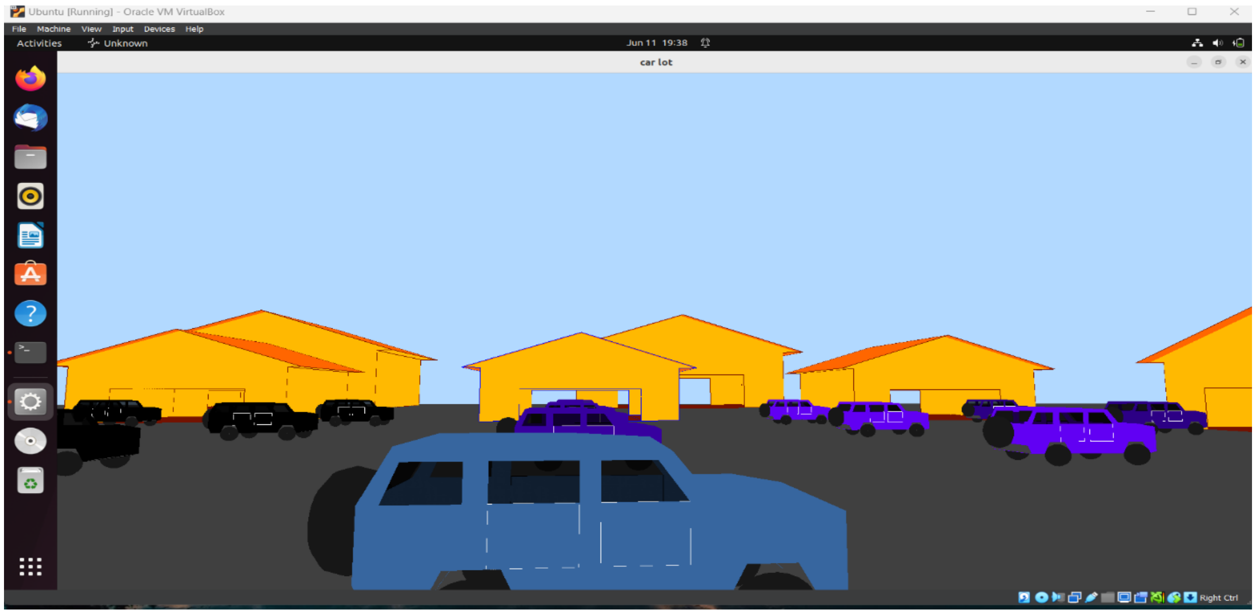

Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Unknown
Jun 11 19:38
car lot
Right Ctrl

Figure 5.2 Cars with Houses

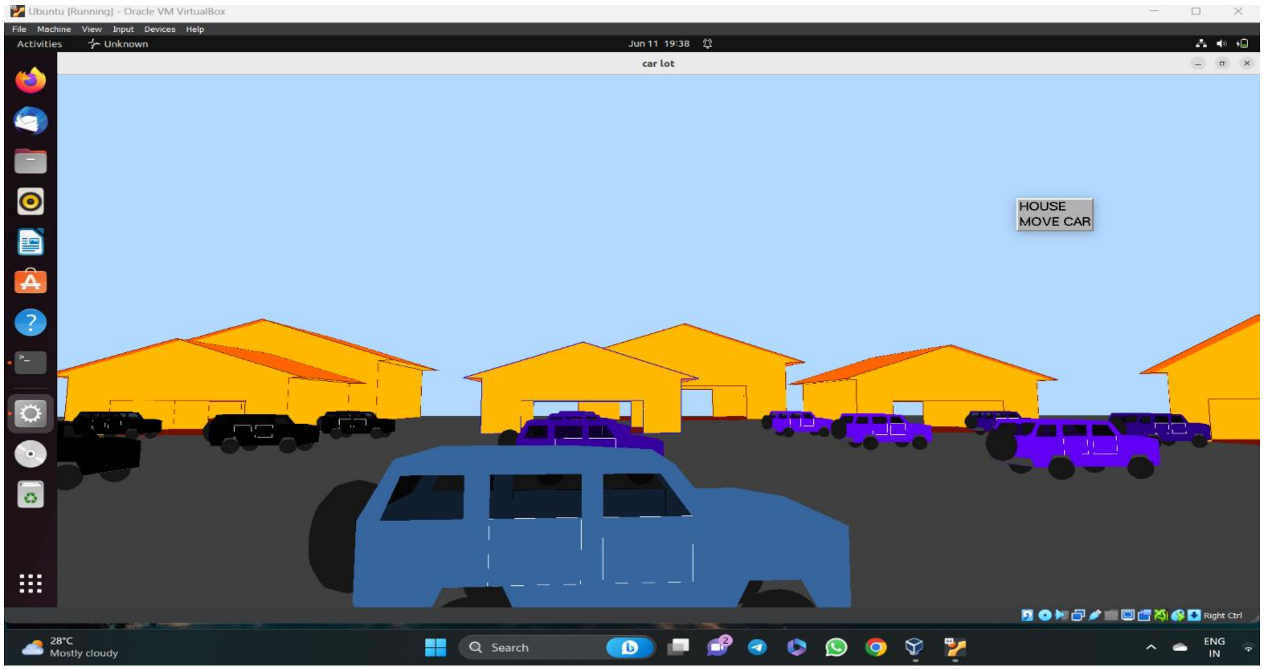
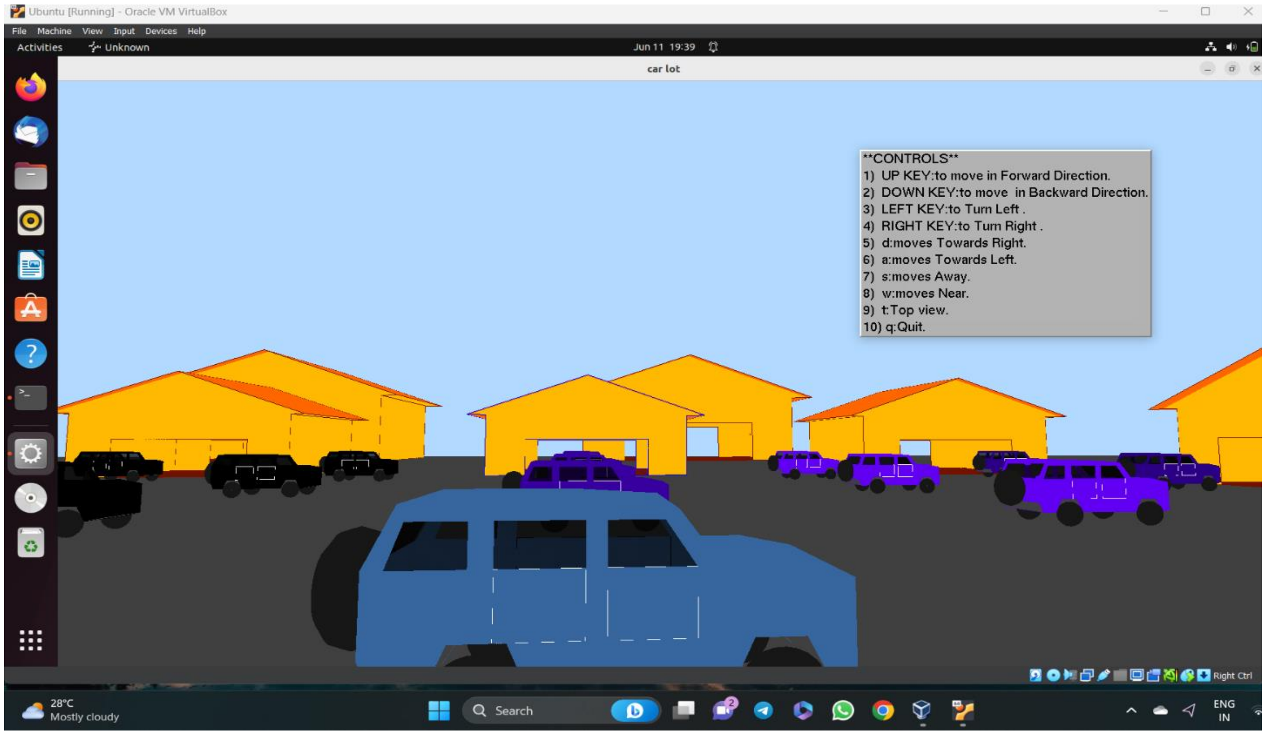
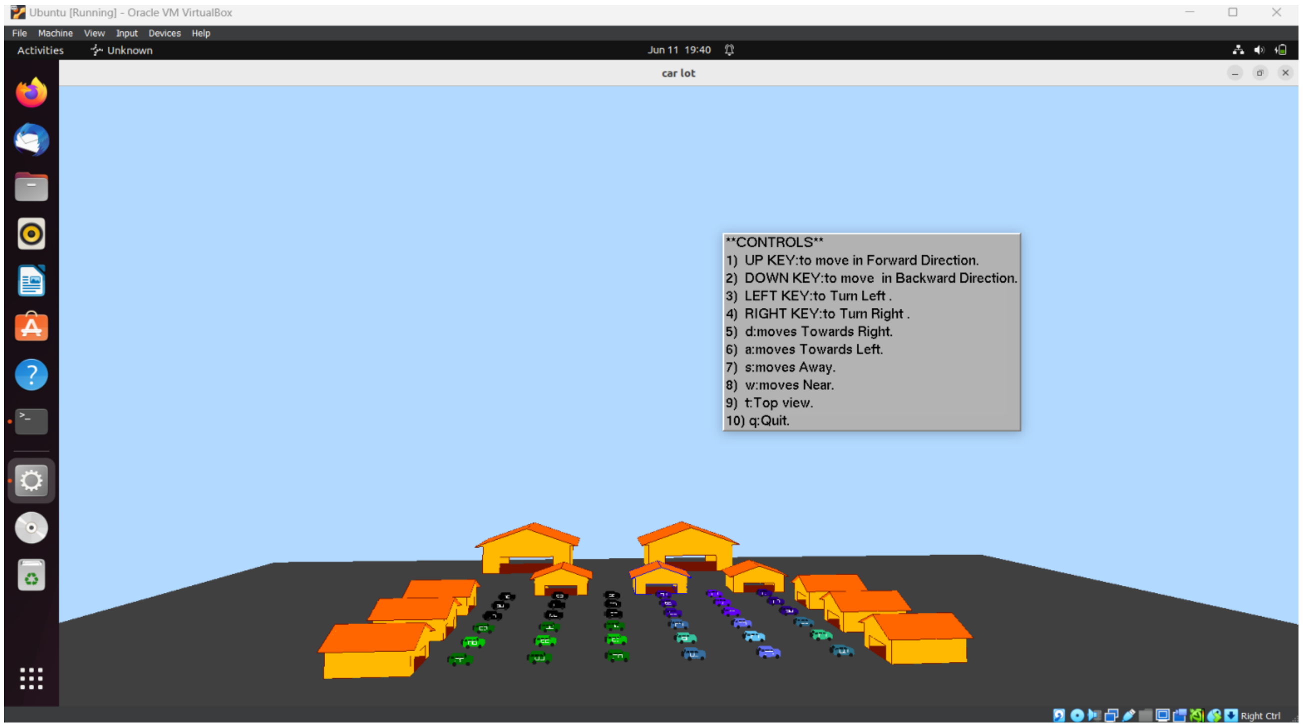


Figure 5.3 Car Parking with Menu



```
**CONTROLS**
1)  UP KEY:to move in Forward Direction.
2)  DOWN KEY:to move  in Backward Direction.
3)  LEFT KEY:to Turn Left .
4)  RIGHT KEY:to Turn Right .
5)  d:moves Towards Right.
6)  a:moves Towards Left.
7)  s:moves Away.
8)  w:moves Near.
9)  t:Top view.
10) q:Quit.
```

Figure 5.4 Car Parking with driving instructions


Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Unknown
Jun 11 19:40
car lot
Right Ctrl

CONTROLS
1) UP KEY:to move in Forward Direction.
2) DOWN KEY:to move in Backward Direction.
3) LEFT KEY:to Turn Left .
4) RIGHT KEY:to Turn Right .
5) d:moves Towards Right.
6) a:moves Towards Left.
7) s:moves Away.
8) w:moves Near.
9) t:Top view.
10) q:Quit.

Figure 5.5 Car Parking with Front View

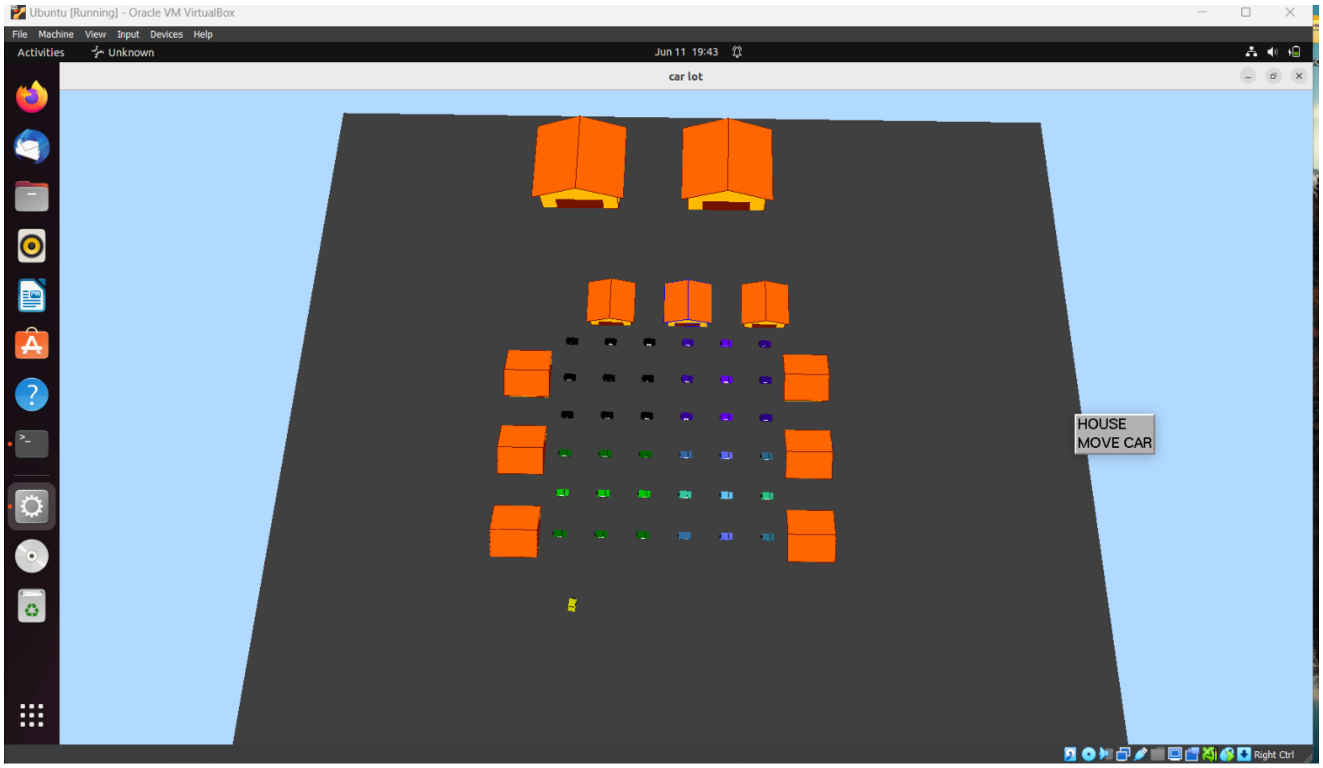

Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Unknown
Jun 11 19:43
car lot
HOUSE
MOVE CAR
Right Ctrl

Figure 5.6 Car Parking with Top View

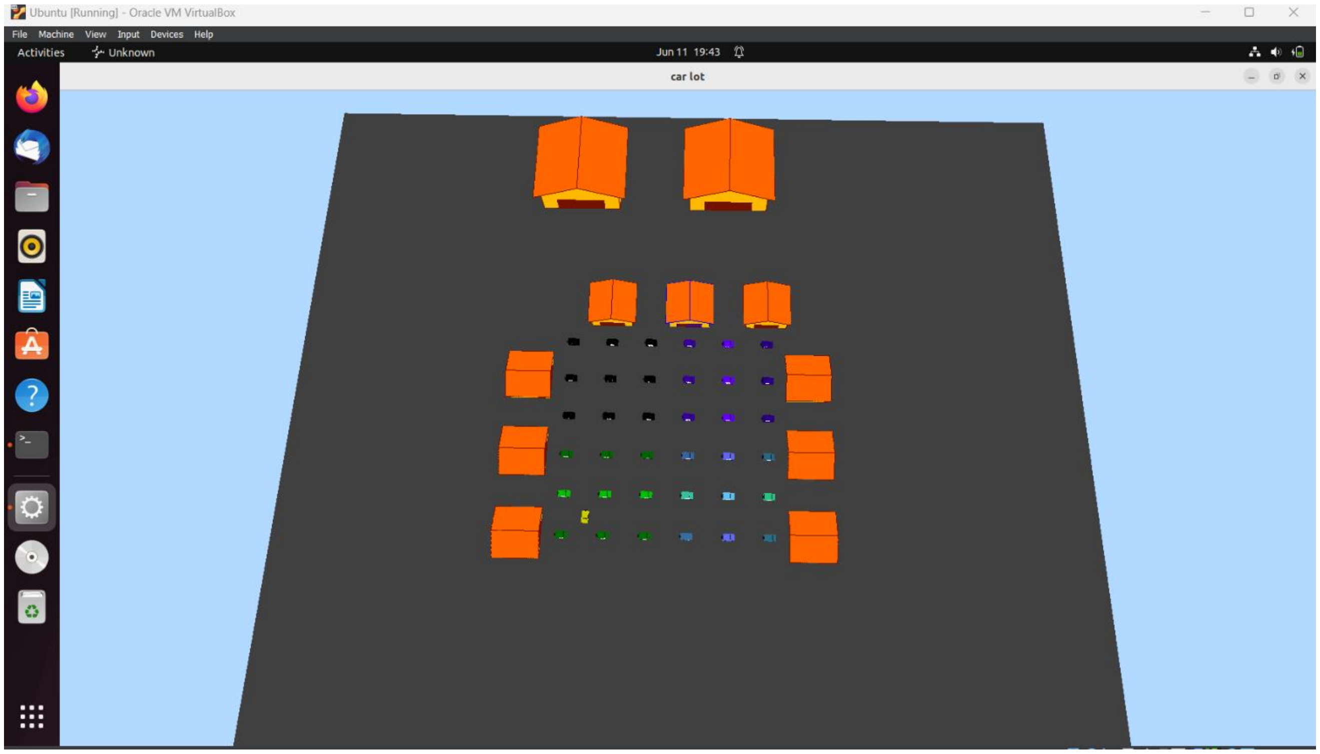

Ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Unknown
Jun 11 19:43
car lot

Figure 5.7 Car Parked(Top View)

Chapter -6

CONCLUSION

The car parking problem necessitates smart parking solutions that leverage technology to optimize parking spaces, improve efficiency, and enhance the user experience. Collaboration between stakeholders is essential, and factors like scalability and affordability must be considered. By adopting sustainable practices, we can create a future with hassle-free parking and streamlined parking management, contributing to efficient urban mobility. Continuous innovation and collective effort are key in transforming the way we park our vehicles.

We thus would like to emphasize the importance of this Project to many other perspectives of Technical , mathematical ,graphical and software concepts which we were unaware of.

6.1 Future Enhancements

- In future the same project can be enhanced in such a way that we can interact more with the project. Also the project can be implemented in 3D Space.
- A vast amount of future work can be possible by following investigations and strategies.
- More features can be included and can be modified in a more versatile way.

BIBLIOGRAPHY

Reference Books

[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

[3] James D Foley, Andries Van Dam, Steven K Feiner, John F Huges Computer Graphics with OpenGL, Pearson Education

[4] Macro Cantu: Mastering Delphi

Websites

[1] https://www.opengl.org/

[2] https://learnopengl.com/