ABSTRACT

The process of creating a design, 2D, 3D, and animation of an object is known as computer graphics. Modeling, simulation, and visualization of an object or a problem are all possible using computer graphics. Graphics is one of the most natural ways to communicate with a computer, because highly developed 2D and 3D pattern recognition abilities allow for rapid and efficient perception and processing of visual data. Since the invention of photography and television, interactive computer graphics has been the most essential method of making images. OpenGL is a cross-platform and cross-language application programming interface (API) for rendering 2D and 3D vector graphics. It is the study of applying computational approaches to manipulate visual and geometric data. Rather than solely aesthetic considerations, it focuses on the mathematical and computational foundations of picture production and processing. To examine the parking from various angles, virtual car parking is employed. To observe these multiple perspectives, different keyboard controls are used.

Table of Contents

List of Figures

Chapter 1

INTRODUCTION

1.1 Overview

Computer graphics is the study of applying computational approaches to manipulate visual and geometric data.

Various methods and techniques are used to create computer graphics.

They are becoming a widespread feature in user interfaces and commercial motion films on television. A picture is the end product of computer graphics; it could be a business graph, drawing, or engineering.

Two-dimensional or three-dimensional images for research can be made using computer graphics.

With the passage of time, many hardware devices algorithms have been developed to improve the speed of picture generation. It entails the generation and storing of object models and images. These models can be used in a variety of fields, including engineering, mathematics, and so on.

Application of Computer Graphics:

- Education and Training: Computer-generated model of the physical, financial and economic system is often used as educational aids. Example- Flight Simulator.
- Use in biology: Molecular biologist can display a picture of molecules and gain insight into their structure with the help of computer graphics.
- Computer-Generated Maps: Town planners and transportation engineers can use computer-generated maps which display data useful to them in their planning work.
- Architect: At an interactive graphics terminal, an architect can experiment with alternate design solutions. They can test a lot more solutions this way, something they couldn't do without the computer.
- Presentation Graphics: An illustration of a presentation Bar charts, line graphs, pie charts, and other visual representations that indicate correlations between various parameters are known as graphics. Presentation graphics are frequently used to summaries information.

1.2 Problem Statement

The goal of this project is to develop a virtual/three-dimensional car park. The viewer is permitted to go about the parking lot and look at the automobiles up close.

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

- GL_LINES
- GL_POLYGON
- GL_QUADS
- GL_TRIANGLE

A display list is constructed for each of these objects.

These display lists are used every time a car 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.

1.3 Motivation

Computer graphics is the art of utilising computers and programming to create images, lines, charts, and other objects. The input interactions and OpenGL transformations add to the graphics. This prompted me to start working on a project to simulate the virtual car parking.

1.4 Computer Graphics

Since they enable non-linear, self-learning settings that are particularly suited to abstract notions and technical information, computer graphics and multimedia technologies are increasingly being used in educational applications. Computer graphics are computer-generated images and films. Typically, the phrase refers to visual data generated by a computer with the assistance of specialized graphics gear and software. It's a broad and relatively new field in computer science. Verne Hudson and William Fetter of Boeing, computer graphics experts, created the word in 1960.It's commonly abbreviated as CG, while it's sometimes referred to as CGI. User interface

design, sprite graphics, vector graphics, 3D modelling, shaders, GPU design, implicit surface visualization with ray tracing, and computer vision are all important issues in computer graphics.

The fundamental sciences of geometry, optics, and physics are heavily used in the entire technique. Computer graphics is in charge of effectively and meaningfully showing art and visual data to the user. It's also utilized to process picture data that comes in from the real world.

Computer graphics development has changed animation, cinema, advertising, video games, and graphic design in general, and has had a huge impact on many sorts of media.

1.5 OpenGL API

Most of our application will be designed to access OpenGL directly through functions in three libraries. Functions in the main GL (or OpenGL in windows) library have names that begin with the letters gl and are stored in a library usually referred to as GL (or OpenGL in windows). 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 functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begin with letters glu.

To interface with the window system and to get input from external devices into our programs, we need at least one more system-specific library that provides the “glue” between the window system and OpenGL. For the X window system, this library is functionality that should be expected in any modern windowing system.

The application program, however, can use only GLUT functions and thus can be recompiled with the GLUT library for other window systems.

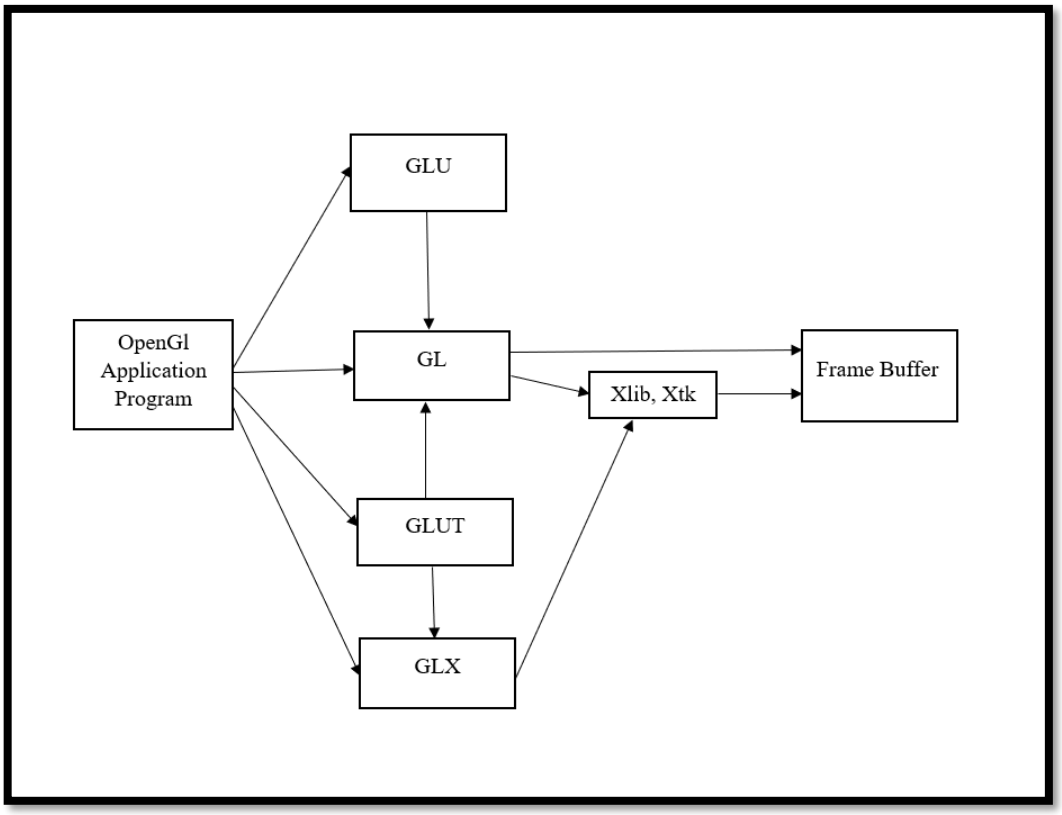

Fig 1.1 Library Organization

OpenGL Command Syntax:

OpenGL commands use the prefix gl and initial capital letters for each word making up the command name. Similarly, OpenGL defined constants begin with GL_, use all capital letters and use underscores to separate words.

Example:

GL_COLOR_BUFFER_BIT

OpenGL API Architecture:

- Display Lists:

  All information, whether it pertains to geometry or pixels, can be kept in a display list for immediate or future use. When a display list is performed, the retained data is delivered from the display list as if it had been sent in immediate mode by the application.

- Evaluators:

  Vertices eventually describe all geometric primitives. Control points and polynomial functions termed basis functions can be used to characterize parametric curves and surfaces at first.

- Per Vertex Operations:

  The "per-vertex operations" stage, which turns vertices into primitives, comes next for vertex data. 4 x 4 floating-point matrices are used to convert some vertex data. The projection of spatial coordinates from a place in the 3D world to a position on your screen.

- Primitive Assembly:

  Clipping is the process of removing pieces of geometry that fall outside of a half space defined by a plane. It is an important aspect of basic assembly.

- Pixel operation:

  Pixel data follows a different path through the OpenGL rendering pipeline than geometric data. Pixels from a system memory array are first unpacked into the correct number of components from one of several formats. A pixel map then scales, biases, and processes the data. The output is capped before being written to texture memory or delivered to the rasterization stage.

- Rasterization:

  The process of converting geometric and pixel data into fragments is known as rasterization. A pixel in the frame buffer corresponds to each fragment square. Each fragment square is given a color and depth value.

- Fragment Operation:

  A sequence of procedures are done before values are actually saved in the frame buffer, which may change or even throw out fragments. All of these functions can be turned on or off.

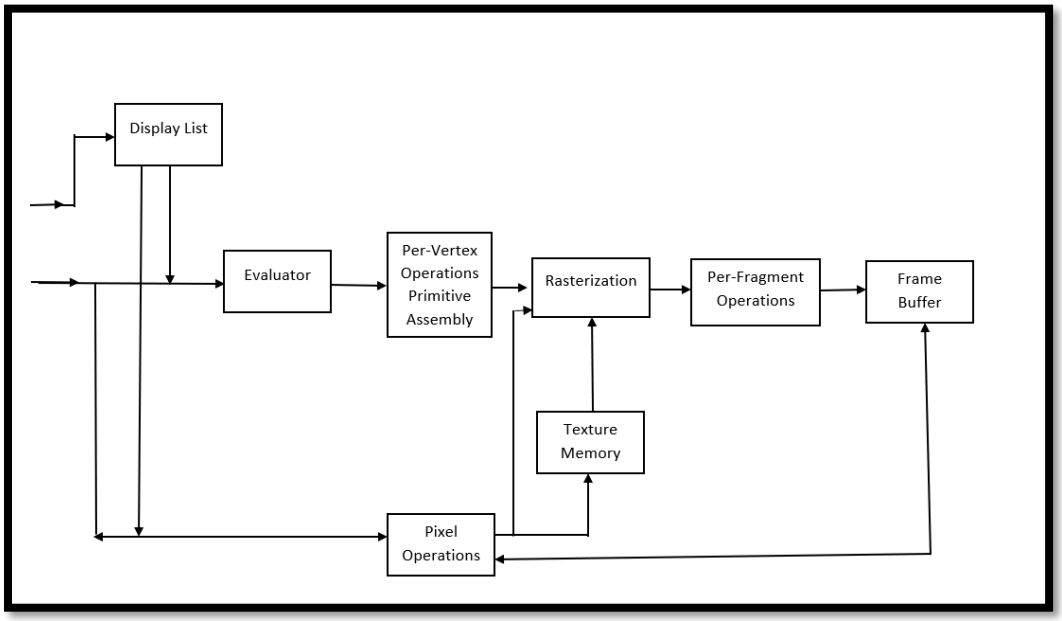

Fig 1.2 An illustration of the graphics pipeline process in OpenGL Architecture

1.6 Applications of Computer Graphics

- Display of Information:

  Graphics has long been related with information display. On 4000-year-old Babylonian stone tablets, examples of the use of orthographic projections to illustrate architectural floorplans may be found. During the Renaissance, mechanical means for making perspective drawings were created. Hand drafting is being replaced by computer-based drafting systems using plotters and workstations in sectors like architecture and mechanical design. Computer graphics are used in medical imaging in a variety of ways.

- Design:

  Engineering and architecture are two professions that deal with design. Despite the fact that their applications differ, most designers face comparable challenges and employ similar methods. As a result, the designer will evaluate a potential design and then tweak it, potentially several times, in order to come up with a superior option. In this iterative process, computer graphics has become an essential component.

- Simulation:

  Both the visual appeal of computer graphics and our ability to generate sophisticated pictures in real time are demonstrated in video games. Flight simulators, which have become the primary way for training pilots, use computer-generated graphics as well.

Chapter 2

LITERATURE SURVEY

2.1 History of Computer Graphics

The first computer graphics design system was developed by Evan Sutherland as his PhD thesis at MIT in 1963. It was called Sketchpad and allowed a user to sketch a mechanical part on a computer screen, place constraints on the part, and have the computer calculate the exact design of the part. It was revolutionary and computer graphics took off "as a thing" from that time forward.

In the early days, if you wanted to create computer graphics you had to write programs that directly talked to the graphics hardware. As new hardware designs were created, the software had to be completely re-written to work with the new hardware. Throughout the 1970's many "graphics libraries" were created to solve this problem, but most of the libraries worked with individual operating systems and could not be easily ported to new operating systems or new hardware. In the 1980's the ISO (International Organization for Standardization) attempted to create a standard computer graphics library called PHIGS. It was claimed to be the best and final answer for creating computer graphics across all computing platforms. However, it was a complete failure for one simple reason: not only did PHIGS provide a library of functionality for drawing 3D graphics, it specified exactly how you had to store and organize your graphics data. Programmers hated it.

The premiere computer graphics company in the late 1980's and throughout the 1990's was Silicon Graphics, Inc. They were a hardware company, but they shipped their computers with a proprietary computer graphics application programmer interface (API) known as IRIS Graphics Language (IRIS GL). Programmers of the day were searching for an alternative to PHIGS and in 1992 Silicon Graphics release their graphics library (GL) as an "open" standard called OpenGL.

OpenGL is a hardware specification that requires fairly powerful hardware to run. The software drivers for the API are very large and complex. As mobile, lower power devices starting appearing on the market, a 3D graphics solution was desired for these devices. Thus OpenGL ES (Embedded Subset) was created. From the opengl web site, OpenGL ES "slims down the rather large OpenGL API to the bare essentials, so that it can be implemented on devices with simpler,

cheaper hardware, and above all, low enough power requirements to run on batteries. For example, it is available as standard on smartphones running both Apple's IOS and Google's Android operating system." OpenGL ES has continually evolved from version 1.0 in 2003 to the most recent version 3.2.

To understand the many issues in today's modern computer graphics, you need to know how developed computer graphics from its beginnings to this day.

- 1950′s - the first graphic displays, military applications.
- 1962 - the first graphics station (sketchpad) consisting of a monitor, light pen and
- software for interactive operation constructed by Ivan Sutherland
- 1964 - research team working on the algorithms in computer graphics employed at the
- University of Utah (including Ivan Sutherland, James Blinn, Edwin Catmull).
- 1965 - the first commercial graphics station: IBM 2250 Display Unit and IBM P1130
- 1969 - beginning of a group SIGGRAPH (Special Interest Group on Graphics) in the organization of ACM (Association for Computing Machinery).
- 1974 - creation of graphics laboratory at the New York Institute of Technology
- 1980 - Turner Whitted published article about creating realistic images, beginning of method of ray tracing.
- 1982 - TRON, the first film that uses computer graphics. The first completely computer-generated scene in the movie Star Trek II: The Wrath of Khan
- 1983 - development of fractal techniques and their use in computer graphics. Fractals are used for example in the movie Star Trek II: The Wrath of Khan
- 1984 - work of C. Goral, K.Torrance, D.Greenberg and B.Battaile and proposing a new approach for visualization – the method of radiosity
- 1988 – the first film sequence with morphing in willow
- 1989 - the first character created using 3D graphics in the studio Industrial Light &Magic (ILM)
- 1993 - dinosaurs in Jurassic Park – the first complete and detailed living organisms generated digital technology
- 1995 - Toy Story implemented complete using computer graphics, the first photo realistic hair and fur computer generated

- 1999 – The first character of the complete human anatomy in a computer-generated +studio ILM
- 2001 - photon mapping as the development of ray tracing method
- 2009 - film Avatar – 3D cinema revolution
- 2009 - decision to create specialty Modern Computer Graphics for Applied Computer Science at the University of Science and Technology in Cracow
- 2013: Now, it is possible to create graphics on a home computer.
- 2015: Big data is being used to create animations.
- 2018: Now, we can create "realistic" graphics on mobile phones. We can also create a completely CGI-based human face in real-time.

2.1 Related Work

3D Graphics "Three-Dimensional graphics" which has parameters like Height, Length and Depth. As this consists of various parameters, ultimate resulting image looks like objects. Applications of 3D Graphics are TV Shows, Video Games, movie we watch often.

- Computer Aided Design (CAD):

Most of engineering and Architecture students are concerned with Design. CAD is used to design various structures such as Computers, Aircrafts, Building, in almost all kind of Industries (where designing is necessary) After making a full diagram, we can even see its animation (Operation and working of a Product)

Chapter 3

SYSTEM REQUIREMENTS

Software and Hardware Requirements

3.1 Software Requirements

- Operating System : Windows 10, 64 bit
- Language Tool : Win32 API for interface and Open GL for the functionalities
- Compiler : GNU GCC Compiler/C++ Compiler
- Library Files : OpenGL files-glut32.h, glut.dll
- Software Used : Microsoft Visual Studio 2019

3.2 Hardware Requirements

- Processor : Intel Core I7 10 Generation
- RAM : 8GB or above
- Hard Disk : 20 GB or above

Chapter 4

SYSTEM DESIGN

4.1 SOFTWARE DESIGN

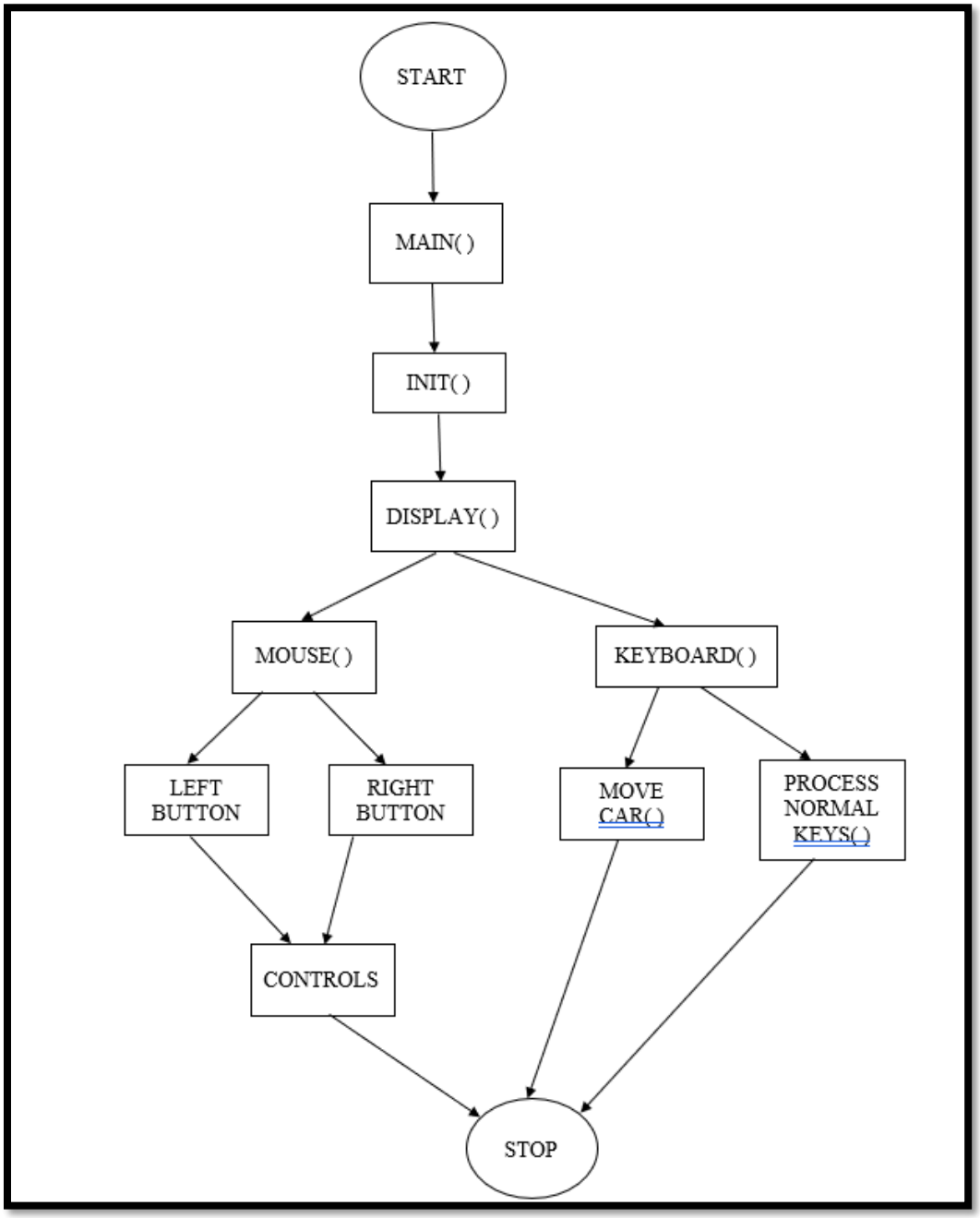

Fig 4.1 Flowchart of Virtual Car Parking

Chapter 5

IMPLEMENTATION

5.1Module Description

- void glScalef(TYPE sx, TYPE sy, TYPE sz)

  alters the current matrix by a scaling of (sx, sy, sz). TYPE here is GLfloat. Here in the above considered example we use scaling to minimize the length of the curve at each iteration. For this curve we use the scale factor to be 3 units because we substitute a line by 4 lines in each iteration.

- void glRotatef(TYPE angle, TYPE dx, TYPE dy, TYPE dz)

  alters the current matrix by a rotation of angle degrees about the axis(dx, dy, dz). TYPE here is GLfloat.

  For a Koch curve we rotate by 60° about the z-axis.

- void glTranslatef(TYPE x, TYPE y, TYPE z)

  alters the current matrix by a displacement of (x, y, z). TYPE here is GLfloat.

  We need to translate to display the new position of the line from the old position and also to go out to the beginning of the next side while drawing.

- void glLoadIdentity()

  sets the current transformation matrix to an identity matrix.

- void glPushMatrix()

  pushes to the matrix stack corresponding to the current matrix mode.

- void glPopMatrix()

  pops from the matrix stack corresponding to the current matrix mode.

- void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)

  defines a two-dimensional viewing rectangle in the plane z=0.

- void glutMouseFunc(myMouse)

  refers to the mouse callback function. The function to callback is defined as

```
void myMouse(int button, int state, int x
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
      if (glutGetModifiers() & GLUT_ACTIVE_SHIFT)
          decrease a level of recursion
      else
          increase a level of recursion
}
```

Here mouse interface is given to increase a level of recursion by clicking mouse button and also to decrease a level of recursion by doing the same holding the shift on the keyboard.

- void glutKeyboardFunc(myKey)

  refers to the keyboard callback function. The function to callback is defined as

```
void myKey(unsigned char key, int x, int y)
{
    if (c == 'q')
        exit
   if (c == 'n')
        //STATEMENTS and repeat when finished
}
```

Here keyboard interface is given to quit, the user can quit by pressing ‘q’ and to see next example of the implementation, the user should press ‘n’.

- void glutSwapBuffers()

  swaps the front and back buffers.

User defined functions are used to color the curves in a standard cycle rainbow manner which becomes very easy for the user to identify the levels of recursion for the curves.

- void glutInit(int *argc, char**argv)

  Initializes GLUT< the arguments from main are passed in and can by the application.

- void glutCreateWindow(char *title)

  Creates a window on the display. The string title can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

- void glutInitDisplaymode(unsigned int mode)

  Requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model (GLUT_RGB<GLUT_INDEX) and buffering (GLUT_SINGLE<GLUT_DOUBLE).

- void glutInitWindowSize(int width,int heights)

  Specifies the initial height and width of the window in pixels.

- void glutInitWindowPosition(int x,int y)

  Specifies the initial position of the top-left corner of the window in pixels.

- void glViewport(int x,int y,GLsizei width,GLsizei height)

  Specifies a width * height viewport in pixels whose lower-left corner is at (x,y) measured from he origin of the window.

- void glutMainLoop()

  Cause the program to enter an event –processing loop.it should be the statement in main.

- void glutPostRedisplay()

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

- void gluLookAt(GLdouble eyex,GLdouble eyey, GLdouble eyez, GLdouble atx, GLdouble aty, GLdouble atz, GLdouble upx, GLdouble upy, GLdouble upz)

  Postmultiplies the current matrix determined by the viewer at the eye point looking at the point with specified up direction.

- void gluPerscpective(GLdouble fov, GLdouble aspect, GLdouble near, GLdouble far)

  Defines a perspective viewing volume using the y direction field of view fov measured in degree,the aspect ratio of the front clipping plane, and the near and far distance.

5.2 High-Level Code

Built-in Functions

- glutInit(init *argc, char **argv):Initialize GLUT and OpenGL. Pass in command-line arguments.
- glutInitDisplayMode(unsigned int mode): Set GLUT's display mode.
- glutInitWindowSize(int width, int height): Set the window size.
- glutCreateWindow(char *title): Create window with the given title.
- glutMainLoop(): This starts the main loop.
- glutDisplayFunc(void(*)void):Callthe given function to redisplay everything.
- glutReshapeFunc(void (*)(int width,int height)): Call the given function when window is resized (and originally created).The function glViewport(), which typically should be called after each resizing.
- glutMouseFunc(void (*)(int button, int state, int x, int y):Call the given function when amouse button is clicked.The button argument can be GLUT_LEFT_BUTTON,GLUT_MIDDLE_BUTTON,GLUT_RIGHT_BUTT

ON, and the state is GLUT_DOWN or GLUT_UP. The (x,y)coordinate give the current mouse position.

- glutPostRedisplay(): Request that the image be redrawn.
- glutSwap Buffers(void): Swap the front and back buffers.
- glClearColor():Specifies the background color to be used by glClear().
- glFlush(void): OpenGL normally saves or "buffers" drawing commands for greater efficiency. This forces the image to be redrawn.
- glRasterPos*(…..): Set the current raster position for subsequent
- glutBitmapCharacter() and other pixel copying operations.
- glBegin(GLenum)…glEnd(void):Define an object, where mode may be any of:
- glVertex*(…) : Specify the coordinates of a vertex.
- glColor*(…..): Specify the color of subsequent vertices. This is normally used if lighting is not enabled. Otherwise, colors are defined by glMaterial().
- glutCreateMenu(void (*)(int)): Creates an empty menu. The argument is the callback function, which is of the form void myfunc(int value), where value holds the index of the menu item which was selected.
- glutAddMenuEntry(char *name, int value): Adds a menu entry to the bottom of the current menu..
- glutAttachMenu(int button): Attaches the current menu to the specified mouse button.
- glLoadIdentity(void): Set the current matrix to identity.
- glPushMatrix(void): Make a copy of the current matrix and push it onto the stack.
- glPopMatrix(void): Pop the top of the current matrix.

User Implementation 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]={55,97,44,152,55,171,108,86,168,99,147,207,238,55,233,167,105,80,134,29,253,1
30,32,240,110,199,224,121,93,199,180,61,110,251,77,237};
int
b[36]={102,194,110,152,153,184,137,113,55,138,104,43,240,255,203,8,100,53,88,64,12
7,64,87,5,2,144,211,128,10,89,27,11,175,185,157,241};
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
        {
                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 6

RESULTS

Fig 6.1 Output of Virtual Car Parking

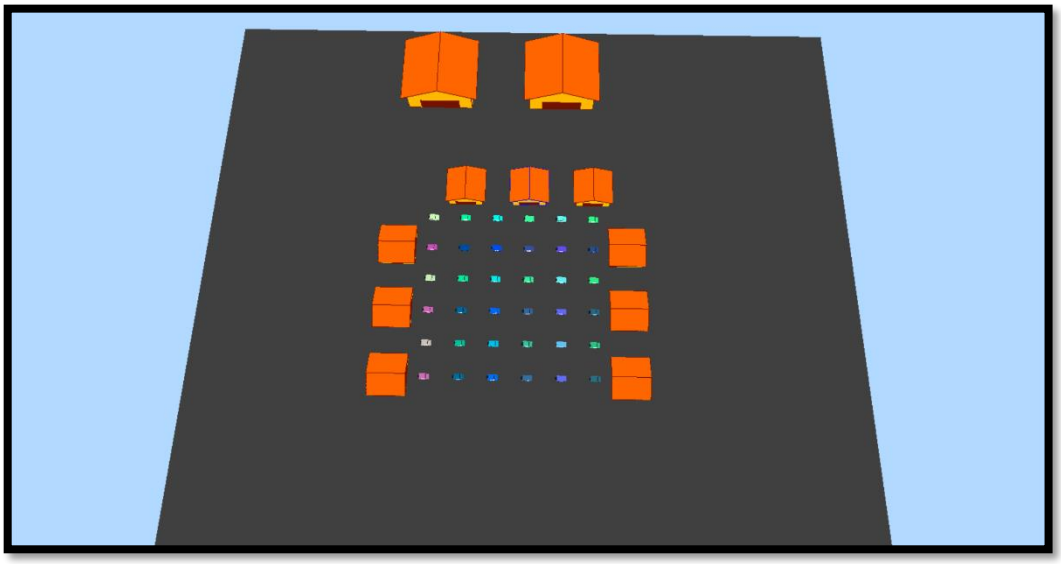

Fig 6.2 Top View of Virtual Car Parking

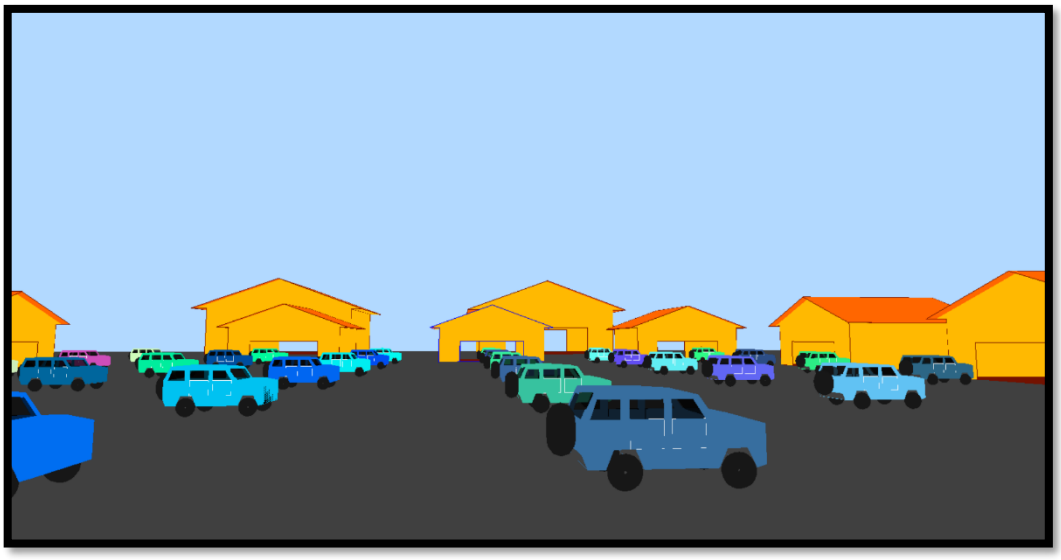

Fig 6.3 Front View of Virtual Car Parking

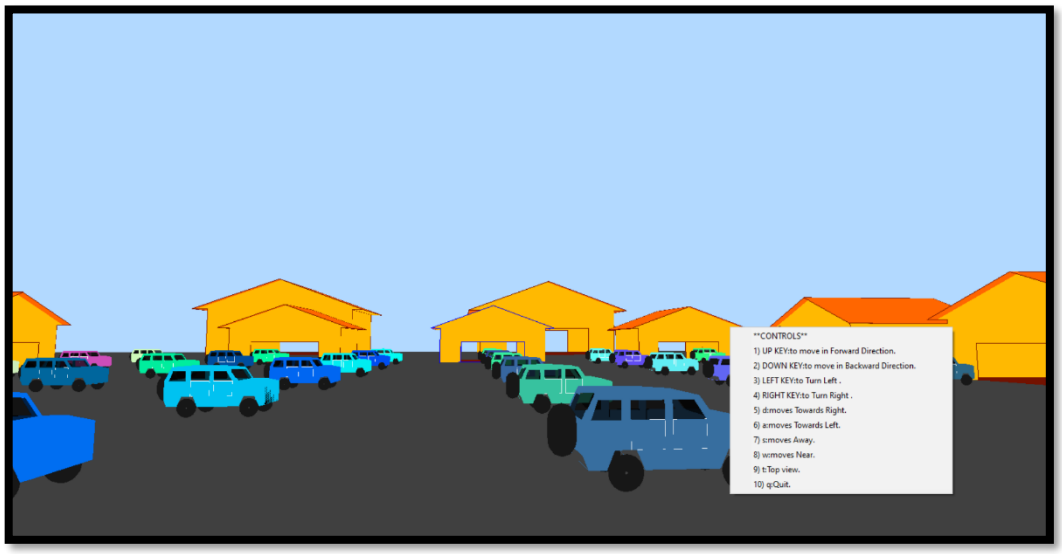

Fig 6.4 Controls in Virtual Car Parking

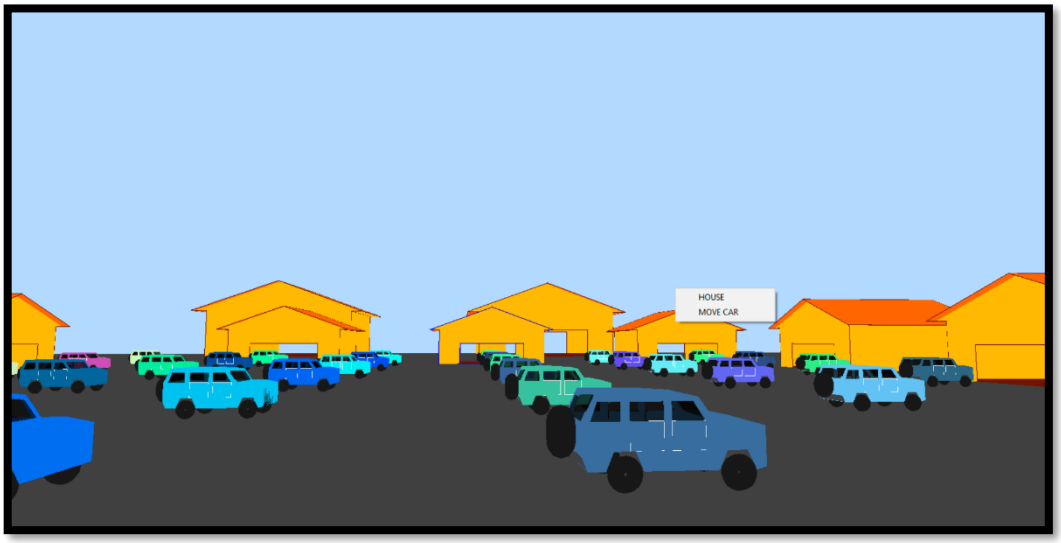

Fig 6.5 Mouse Controls in Virtual Car Parking

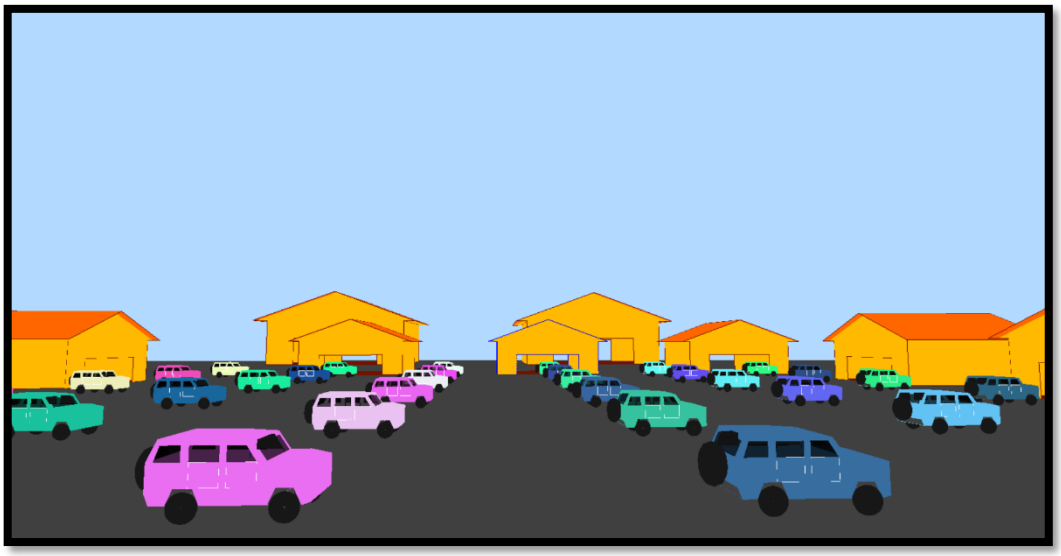

Fig 6.6 Moving Cars in Virtual Car Parking

CONCLUSION

From this project, we can examine the parking from various angles of car parking which is employed. We also observed these multiple perspectives, where different keyboard controls are used. The viewer is permitted to go about the parking lot and look at the automobiles up close. Several keyboard and mouse controls are used. Using OpenGL, in Computer Graphics and Visualization, this project has been developed. Several OpenGL Primitives are used to display the cars and house. Some of the keyboard controls used are UP, DOWN, LEFT, RIGHT keys along with d, a, s, w, t and q. All the keys have different controls. Graphics is one of the most natural ways to communicate with a computer, because highly developed 2D and 3D pattern recognition abilities allow for rapid and efficient perception and processing of visual data. Since the invention of photography and television, interactive computer graphics has been the most essential method of making images.

BIBLIOGRAPHY

- Interactive Computer Graphics A Top-Down Approach with OpenGL - Edward Angel, 5th Edition, Addison-Wesley, 2008
- The Official Guide to Learning OpenGL, by Jackie Neider, Tom Davis, Mason Woo (THE RED BOOK)
- OpenGL Super Bible by Richard S. Wright, Jr. and Michael Sweet
- http://www.cs.rutgers.edu/~decarlo/428/glman.html online man pages.
- http://www.opengl.org/sdk/docs/man online man pages.
- http://nehe.gamedev.net OpenGL tutorials.
- GOOGLE.