

## CHAPTER 1

### 1.1: INTRODUCTION TO CAR PARK.

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

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

1. GL\_LINES
2. GL\_POLYGON
3. GL\_QUADS
4. GL\_TRIANGLE

In a similar way the 3D house is constructed.

A display list is constructed for each of these objects. These display lists are used everytime a car or house has to be constructed.

So, to create the 36 cars in the parking lot the “carr\_display\_list” is called 36 times from within a loop and are translated each time by suitable values to place them correctly.

Similarly, to construct the houses “house\_display\_list” is called and are suitably translated, scaled and rotated to place them properly.

For the movement of camera GluLookAt () function is used .

The controls are:-

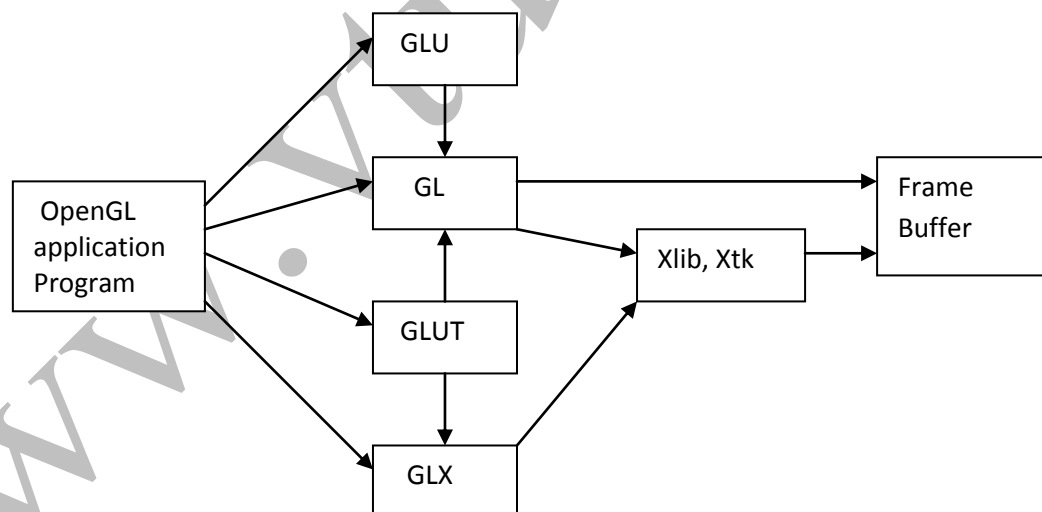
1. UP KEY - > to move the viewer in forward direction.
2. DOWN KEY - > to move the viewer in backwards direction.
3. LEFT KEY - > to rotate the camera to the left of the viewer.
4. RIGHT KEY - > to rotate the camera to the right of the viewer.
5. T - > top view.
6. S - > to move away.
7. W - > to move near.
8. D - > to move right.
9. A - > to move left.
10. Q - > quit.

## 1.2:INTRODUCTION TO OPENGL

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.

Fig 2.1 shows the organization of the libraries for an X Window System environment. For this window system, GLUT will use GLX and the X libraries. The application program, however, can use only GLUT functions and thus can be recompiled with the GLUT library for other window systems.



**Fig 1.2 Library organization**

### 1.2.1: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 (like `GL_COLOR_BUFFER_BIT`).

## CHAPTER 2

### LITERATURE SURVEY.

This project makes extensive use of translations, rotations and scaling for creating .

1. THE LINKS : <http://www.cs.rutgers.edu/~decarlo/428/glman.html> online man pages.
2. <http://www.opengl.org/sdk/docs/man> online man pages.
3. <http://nehe.gamedev.net> OpenGL tutorials.

Provides the description of the following functions.

**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 the 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 gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far)**

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

## CHAPTER 3

# REQUIREMENTS AND SPECIFICATION.

### Hardware Constraints

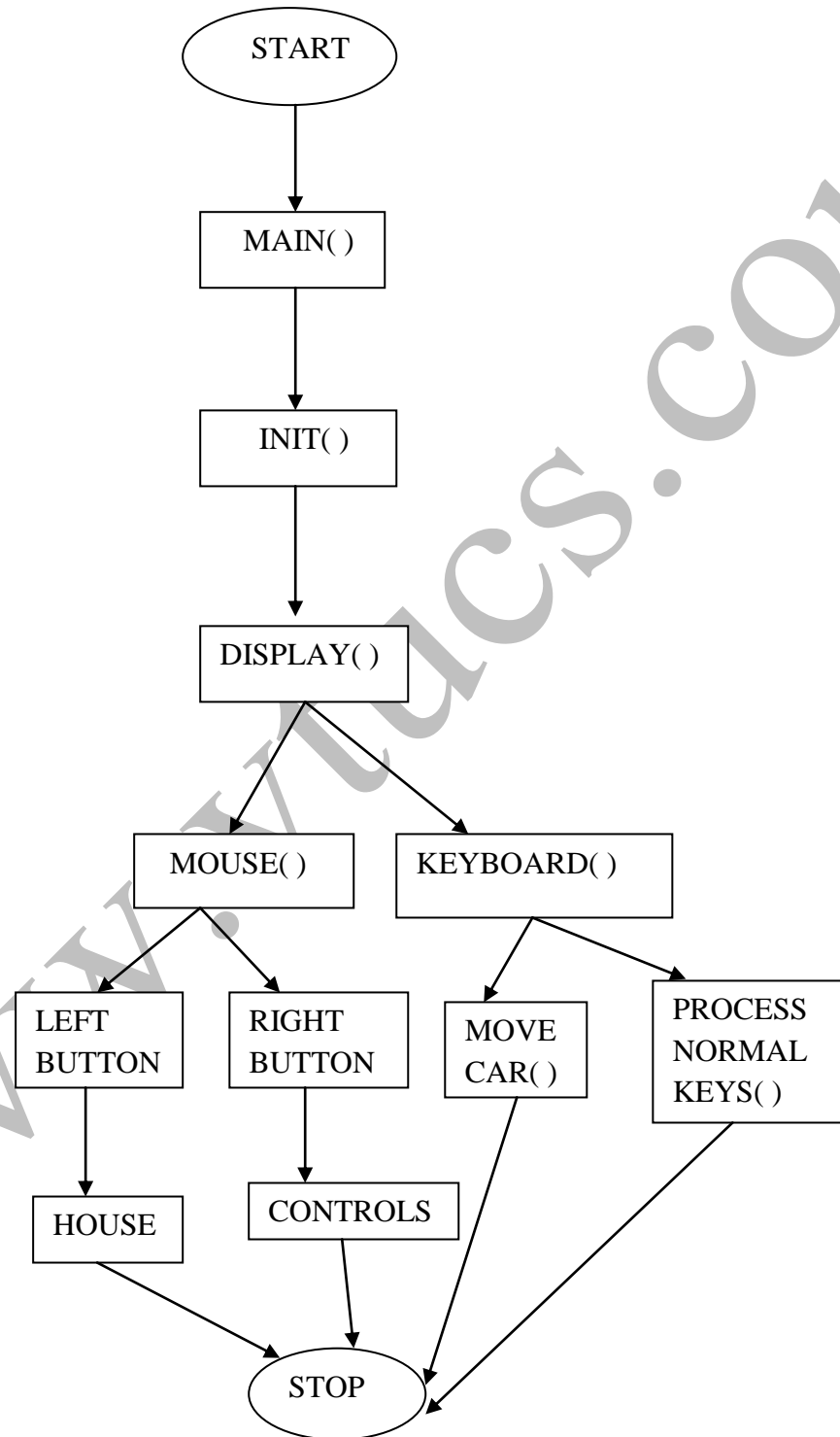
- Processor : Pentium PC
- RAM : 512MB
- Hard Disk : 20GB(approx)
- Display : VGA Color Monitor

### Software Constraints

- Operating System : Windows 98SE/2000/XP/Vista/UBUNTU
- Language : Open Gl
- Compiler : Eclipse/Microsoft Visual studio 2005

## CHAPTER 4

### SOFTWARE DESIGN.



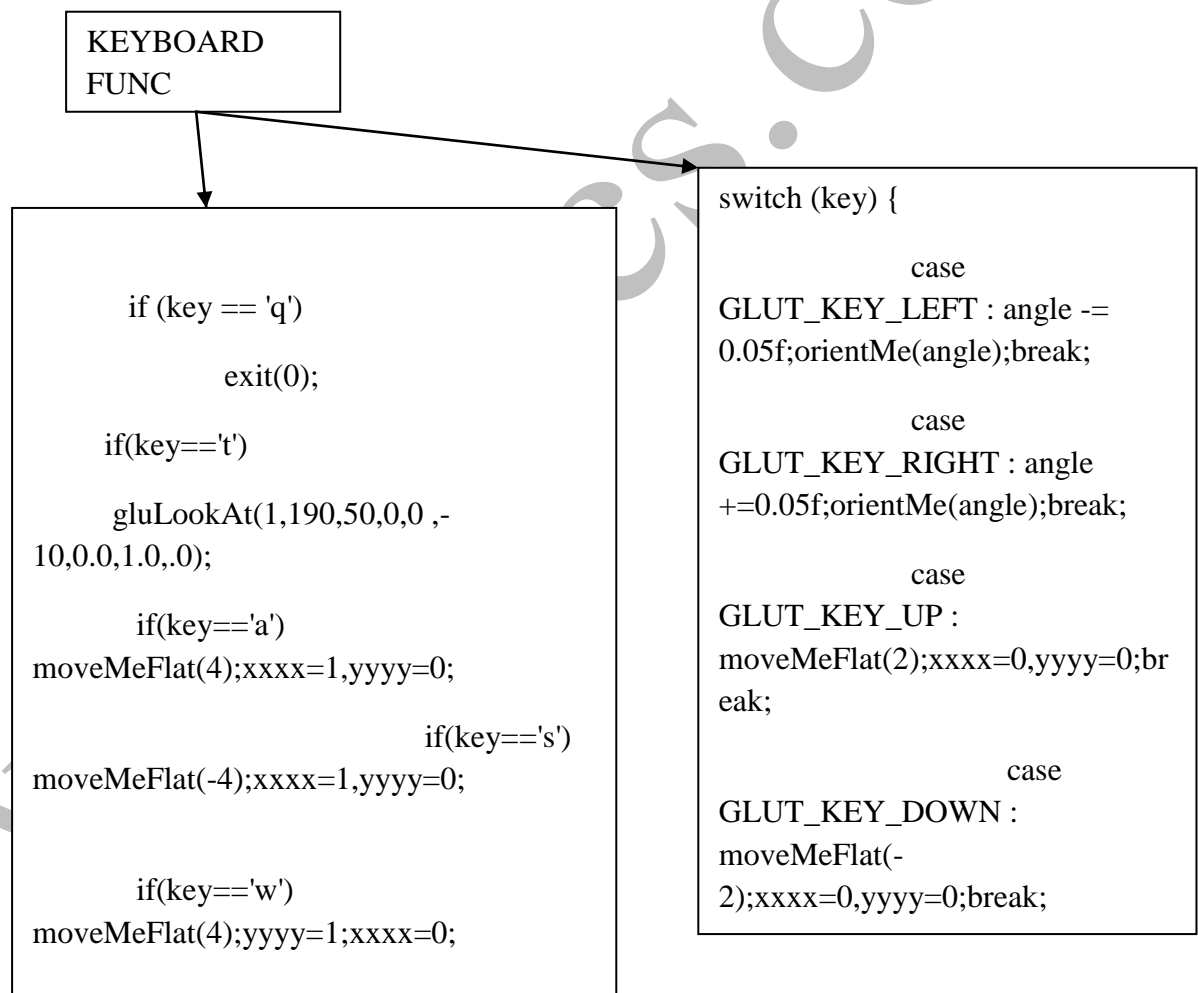
**CHAPTER 5****IMPLEMENTATION.**

5.1:Keyboard function.

5.2:Display function.

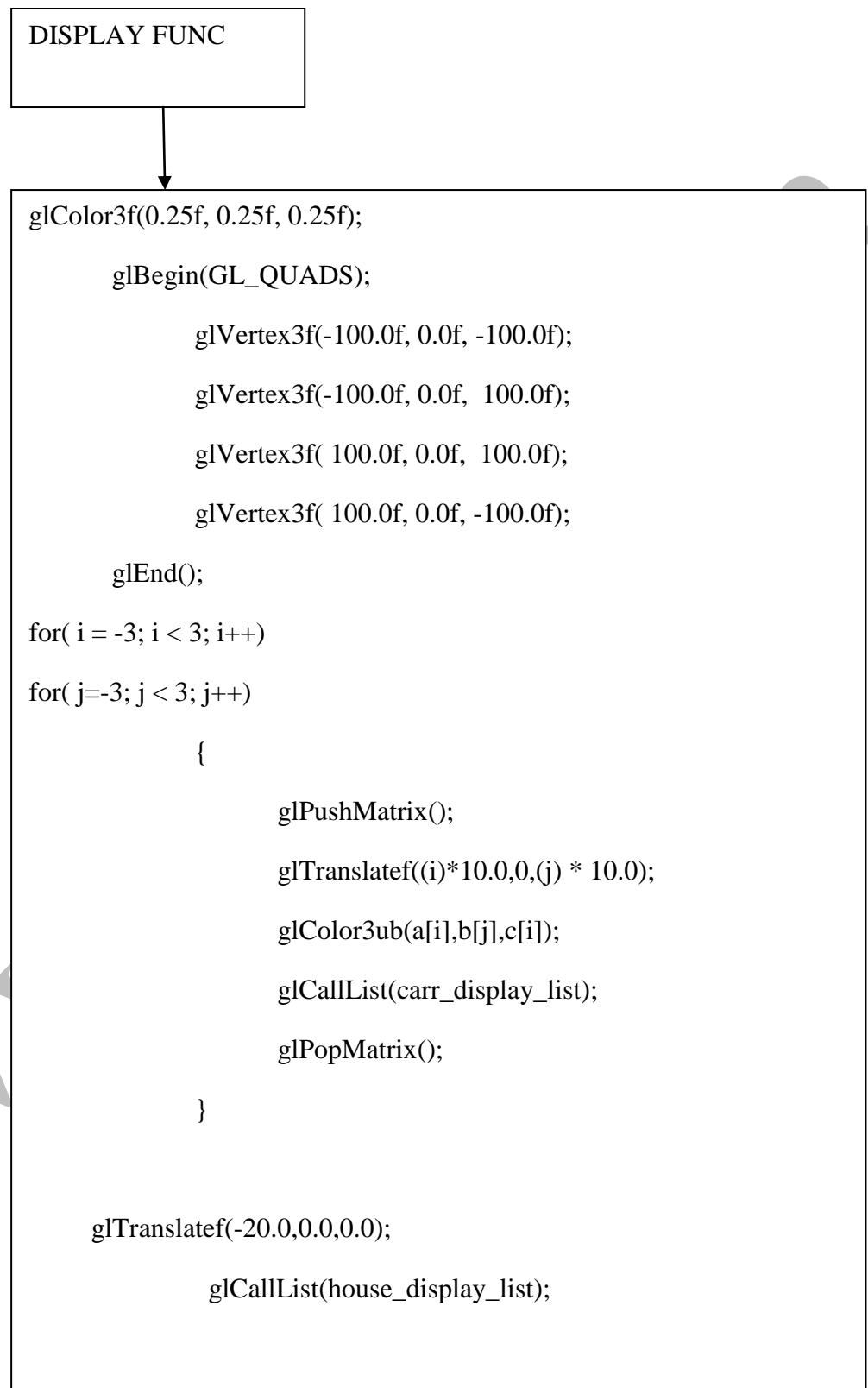
5.3:Reshape function.

5.1:Keyboard function:



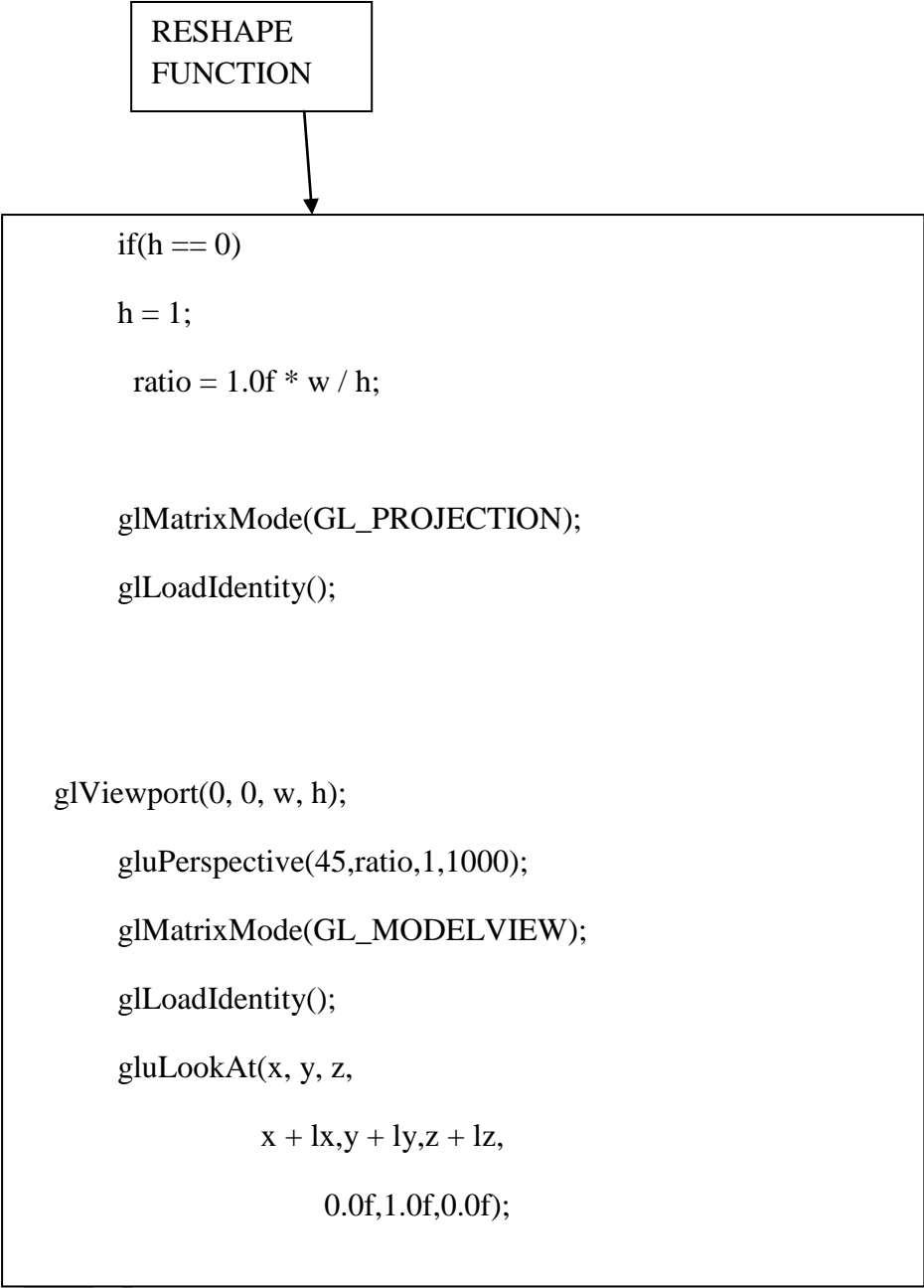


### 5.2:Display function:



### 5.3:Reshape function:

RESHAPE  
FUNCTION



```
if(h == 0)
    h = 1;
    ratio = 1.0f * w / h;

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    glViewport(0, 0, w, h);
    gluPerspective(45,ratio,1,1000);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(x, y, z,
              x + lx,y + ly,z + lz,
              0.0f,1.0f,0.0f);
```

## CHAPTER 6

### SNAPSHOTS.

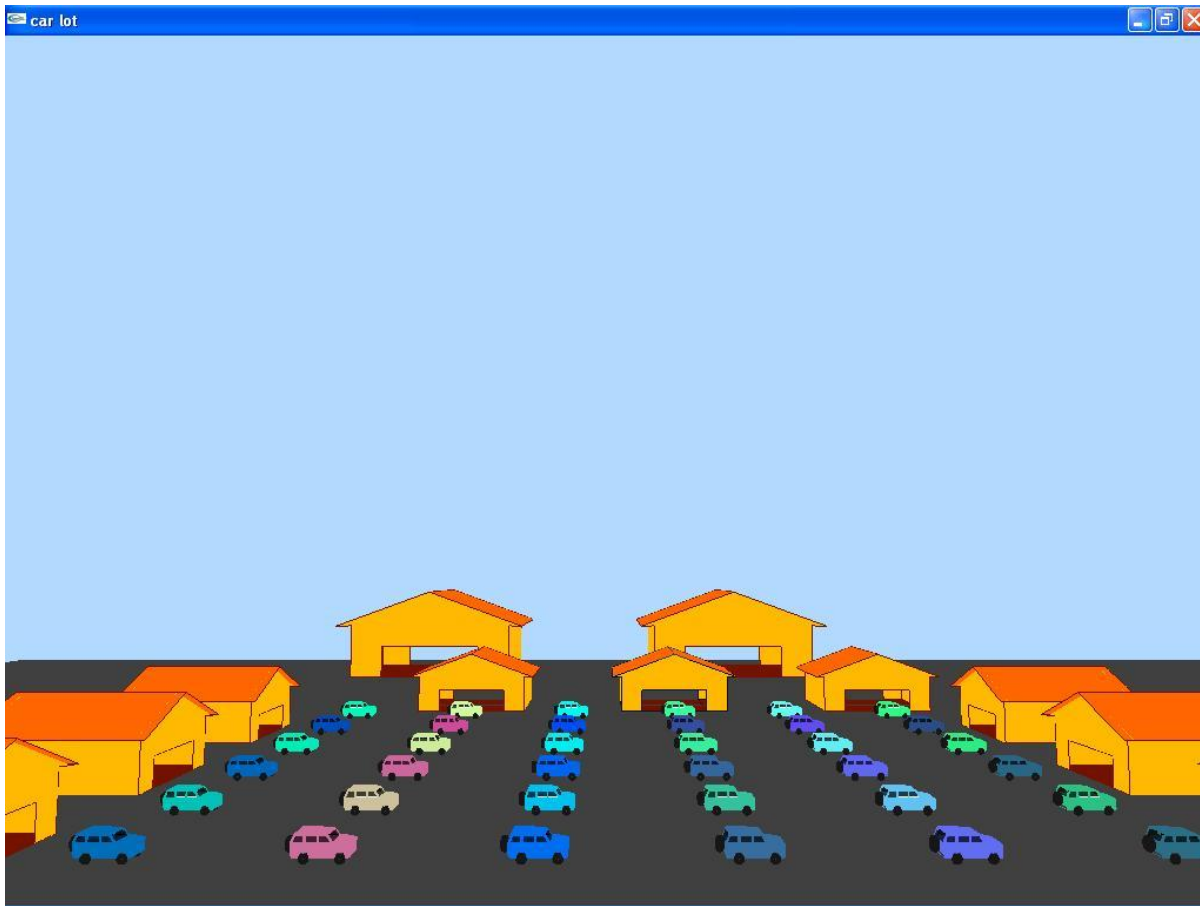
#### 6.1: INITIAL SCENE:

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



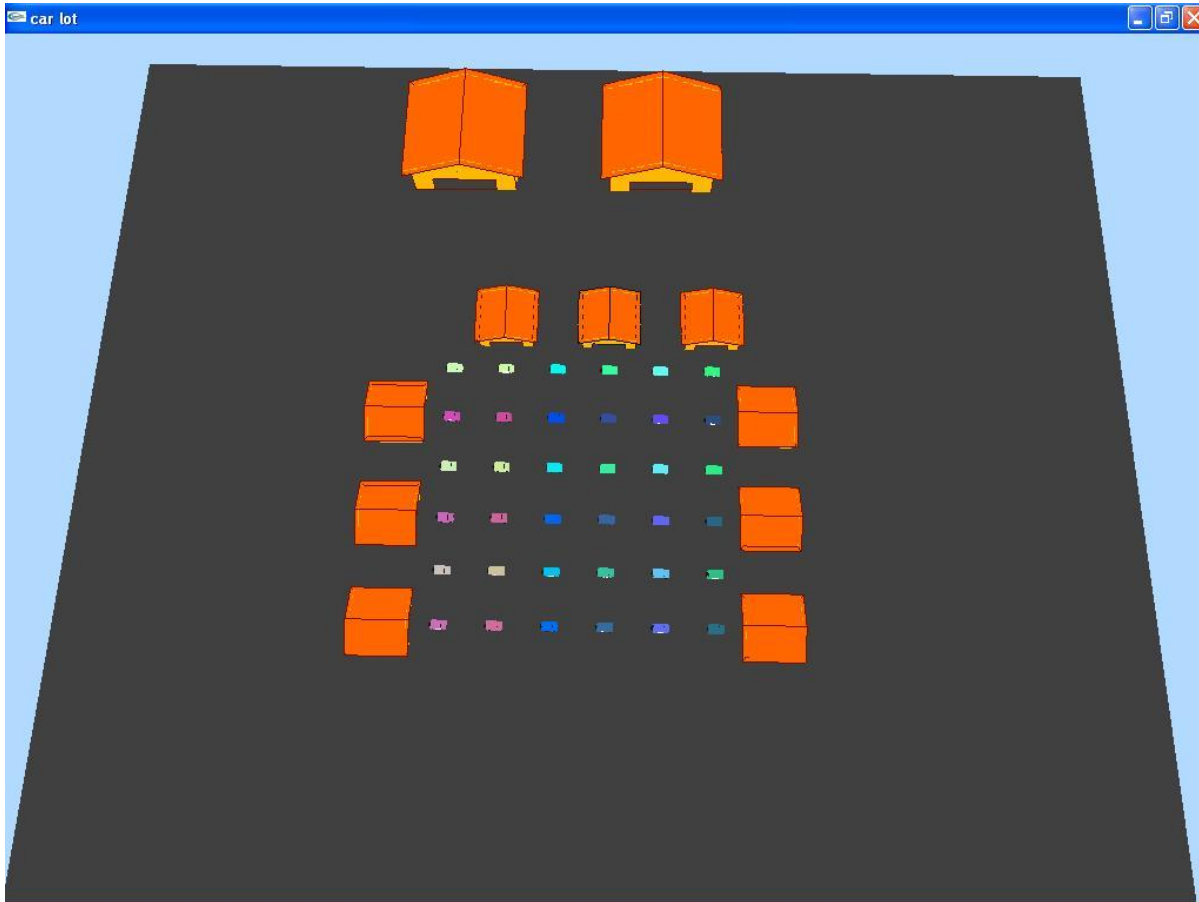
### 6.2:FRONT VIEW :

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



### 6.3:TOP VIEW :

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



## CHAPTER 7

### CONCLUSION and FUTURE SCOPE.

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

#### Future scope:

##### 1. 3-D MAP:

This project can be modified and a lot of other objects can be added for  
example:-

Trees , boundary walls, multiplexes ,roads etc.

THUS A WHOLE CITY CAN BE CONSTRUCTED.

##### 2. GAME:

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

## APPENDIX.

```
#include <GL/glut.h>
#include <math.h>
#include <stdlib.h>

static float angle=0.0, ratio;
static float x=0.0f, y=1.75f, z=5.0f;
static float lx=0.10f, ly=0.10f, lz=-1.0f;
static GLint carr_display_list, house_display_list;
float theta=0.01, fxincr=0.1, fzincr=0, temp, theta1, fx=-10, fz=80;
int xxxx=0, yyyy=0, kk=0, housevisible=0, movecarvar=0;
int
a[36]={55, 97, 44, 152, 55, 171, 108, 86, 168, 99, 147, 207, 238, 55, 233, 167, 105, 80, 134, 29, 253, 130, 32,
240, 110, 199, 224, 121, 93, 199, 180, 61, 110, 251, 77, 237};
int
b[36]={102, 194, 110, 152, 153, 184, 137, 113, 55, 138, 104, 43, 240, 255, 203, 8, 100, 53, 88, 64, 127, 64, 87
, 5, 2, 144, 211, 128, 10, 89, 27, 11, 175, 185, 157, 241};
int
c[36]={159, 243, 133, 253, 233, 228, 141, 18, 46, 195, 75, 52, 253, 204, 169, 30, 78, 94, 68, 117, 4, 2, 33, 12,
2, 25, 195, 76, 26, 54, 98, 103, 205, 173, 65, 242};

void changeSize(int w, int h)
{
if(h == 0) // Prevent a divide by zero, when window is too short
// (you cant make a window of zero width).
h = 1;
ratio = 1.0f * w / h; // Reset the coordinate system before modifying

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glViewport(0, 0, w, h); // Set the viewport to be the entire window
gluPerspective(45, ratio, 1, 1000);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(x, y, z, x + lx, y + ly, z + lz, 0.0f, 1.0f, 0.0f);
}

void drawcarr()
{
glTranslatef(.0, 0.8, 0.0);
glEnable(GL_BLEND); //TRANCPARENCY1
glBlendFunc(GL_ONE, GL_ZERO); //TRANCPARENCY2
glBegin(GL_LINE_LOOP);
glVertex3f(-1.12, -.48, 0.7); //a
glVertex3f(-0.86, -.48, 0.7); //b
glVertex3f(-.74, -.2, 0.7); //c
glVertex3f(-.42, -.2, 0.7); //d
glVertex3f(-0.3, -.48, 0.7); //e
glVertex3f(.81, -.48, 0.7); //f
glVertex3f(.94, -.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_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);//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'
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)//o
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)//o'
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
```

## Car park

---

```
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();
```

## Car park

---

```
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;
    }
}
```

## Car park

---

```
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 controll1;
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);
controll1=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);
}
```

## **BIBLIOGRAPHY.**

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

[www.vtlucS.com](http://www.vtlucS.com)