1. INTRODUCTION

1.1 PROBLEM STATEMENT

3D Home Architect is a property designing program. Harneet's guide to 3D Home Architect comes in three designs for specific purposes: Home and Landscape Design Suite, Home Design Deluxe, and Landscape Design Deluxe. Home Design Deluxe simulates home designs, Landscape Design Deluxe simulates landscape designs, and Home and Landscape Design Suite is used for both. 3D Home Architect was introduced by Broderbund in the 1990s and was a scaled down version of a professional home design application called Chief Architect, made by Advanced Relational Technology (ART) Inc. (now renamed to Chief Architect, Inc.). After version 4.0, the agreement between Broderbund and ART Inc. was terminated, and 3D Home Architect 5.0 and later versions are based on a similar professional application called Cad soft Envisioned.

1.2 OBJECTIVE

- The narrative mode (also known as the mode of narration) is the set of methods the author of a literary, theatrical, cinematic, or musical story uses to convey the plot to the audience.
- Narration, the process of presenting the narrative, occurs because of the narrative mode. It encompasses several overlapping areas of concern, most importantly narrative point-of-view, which determines through whose perspective the story is viewed; narrative voice, which determines the manner through which the story is communicated to the author to be the same person.
- However, the narrator may be a fictive person devised by the author as a stand-alone entity, or even a character.
- The narrator is considered participant if an actual character in the story, and nonparticipant if only an implied character, or a sort of omniscient or semi-omniscient being who does not take part in the story but only relates it to the audience.

1.3 SOLUTION

To accomplish the objective, the following steps will be taken:-

1. Set up your OpenGL environment:
   -Set up any necessary libraries or frameworks.
   -Initialize your window and OpenGL context.

-Define the viewport and projection matrix.

2. Define the geometry of the house:

-Use basic 3D geometric primitives like cubes, rectangles, and triangles to define the walls, roof, and other components of the house.

- Specify the vertices, normals, and texture coordinates for each face of the house.

- Optionally, you can load external 3D models for more complex objects like doors, windows, or furniture.

3. Implement transformations:

- Use transformation matrices to position and orient the house in the 3D space.

- Apply translation, rotation, and scaling operations to achieve the desired placement and size of the house.

4. Set up lighting:

- Define the position and properties of light sources in the scene, such as ambient, diffuse, and specular lighting.

- Set the material properties of the house to interact with the light sources.

5. Implement rendering:

- Write the rendering code to draw the house on the screen.

- Use appropriate OpenGL functions to bind textures, enable depth testing, and handle face culling.

- Iterate through the defined geometry and render each face of the house.

6. Handle user input and interactions:

- Implement controls to allow the user to interact with the 3D house, such as rotating or moving the camera around the scene.

- Handle user input events and update the corresponding transformations or camera parameters accordingly.

2. ABOUT OPENGL

2.1 HISTORY

2.1.1 Overview Of Computer Graphics

Computer graphics are graphics created using computers and, more generally, the representation and manipulation of image data by a computer hardware and software. The development of computer graphics, or simply referred to as CG, has made computers easier to interact with, and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized the animation and video game industry. 2D computer graphics are digital images4mostly from two-dimensional models, such as 2D geometric models, text (vector array), and 2D data. 3D computer graphics in contrast to 2D computer graphics are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering images. The user controls the contents, structure, and appearance of the objects and of their displayed images by using input devices, such as keyboard, mouse, or touch screen. Due to close relationships between the input devices and the display, the handling of such devices is included in the study of computer graphics. The advantages of the interactive graphics are many in number. Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D and 3D patterrecognition abilities allow us to perceive and process data rapidly and efficiently. In many designs, implementation, and construction processes today, the information pictures can give is virtually indispensable. Scientific visualization became an important field in the 1980s when the scientists and engineers realized that they could not interpret the prodigious quantities of data produced in supercomputer runs without summarizing the data and highlighting trends and phenomena in various kinds of graphical representations.

2.1.2 Overview Of OpenGL

OpenGL is the most extensively documented 3D graphics API (Application Program Interface) to date. Information regarding OpenGL is all over the Web and in print. It is impossible to exhaustively list all sources of OpenGL information. OpenGL programs are typically written in C and C++. One can also program OpenGL from Delphi (a Pascal-like language), Basic, Fortran, Ada, and other languages. To compile and link OpenGL programs, one will need OpenGL header files. To run OpenGL programs, one may need shared or dynamically loaded OpenGL libraries, or a vendor-specific OpenGL Installable

Client Driver (ICD). OpenGL is a low-level graphics library specification. It makes available to the programmer a small set of geometric primitives - points, lines, polygons, images, and bitmaps. OpenGL provides a set of commands that allow the specification of geometric objects in two or three dimensions, using the provided primitives, together with commands that control how these objects are rendered (drawn).

2.1.3 Overview Of GLUT

The OpenGL Utility Toolkit (GLUT) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system. Functions performed include window definition, window control, and monitoring of keyboard and mouse input. Routines for drawing a number of geometric primitives (both in solid and wireframe mode) are also provided, including cubes, spheres, and cylinders. GLUT even has some limited support for creating pop-up menus. The two aims of GLUT are to allow the creation of rather portable code between operating systems (GLUT is cross platform) and to make learning OpenGL easier. All GLUT functions start with the glut prefix (for example, glutPostRedisplay marks the current window as needing to be redrawn).

2.2 OpenGL ARCHITECTURE

The OpenGL architecture is structured as a state-based pipeline. Below is a simplified diagram of this pipeline. Commands enter the pipeline from the left.

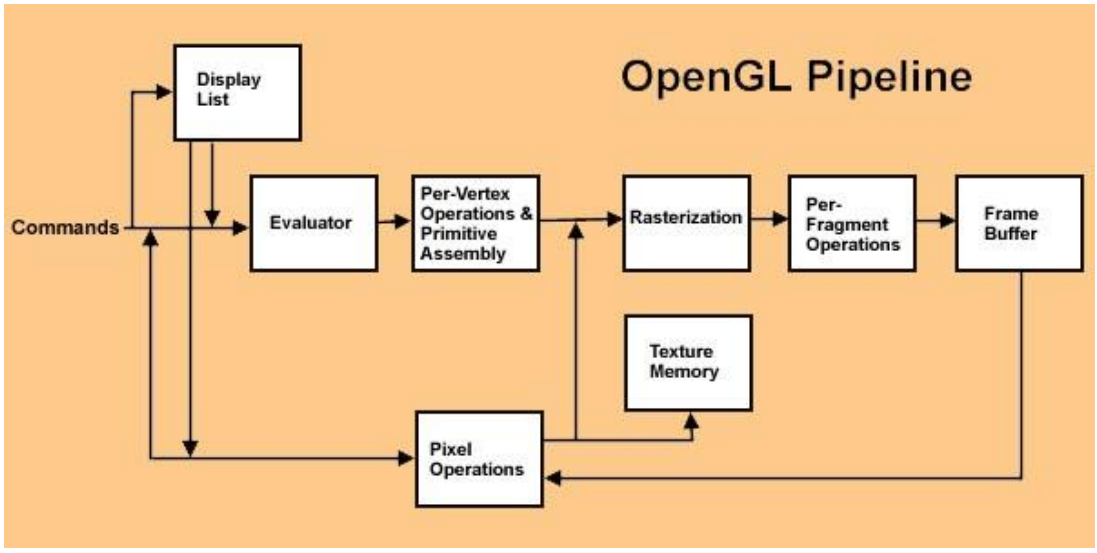

fig.2.1 OpenGL Architecture

- Commands may either be accumulated in display lists, or processed immediately through the pipeline. Display lists allow for greater optimization and command reuse, but not all commands can be put in display lists.

- The first stage in the pipeline is the evaluator. This stage effectively takes any polynomial evaluator commands and evaluates them into their corresponding vertex and attribute commands.
- The second stage is the per-vertex operations, including transformations, lighting, primitive assembly, clipping, projection, and viewport mapping.
- The third stage is rasterization. This stage produces fragments, which are series of framebuffer addresses and values, from the viewport-mapped primitives as well as bitmaps
- The fourth stage is the per-fragment operations. Before fragments go to the framebuffer, they may be subjected to a series of conditional tests and modifications, such as blending or z-buffering.
- Parts of the framebuffer may be fed back into the pipeline as pixel rectangles. Texture memory may be used in the rasterization process when texture mapping is enabled.

2.3 OpenGL Functions

OpenGL functions can be broadly categorized into seven types based on their functionality and purpose. These types are:

1. Initialization Functions: These functions are used to initialize the OpenGL context and set up the rendering environment. Examples include glutInit(), glutInitDisplayMode(), and glClearColor().

2. State Management Functions: These functions allow you to query and modify the state of the OpenGL rendering pipeline. They include functions like glEnable(), glDisable(), glViewport(), and glMatrixMode().

3. Drawing Functions: These functions are used to render geometry and primitives. They include functions such as glBegin(), glEnd(), glVertex(), glColor(), and glDrawArrays().

4. Buffer Functions: These functions are used to manage various buffers used in OpenGL, such as vertex buffers, index buffers, and frame buffers. Examples include glGenBuffers(), glBindBuffer(), glBufferData(), and glFramebufferTexture2D().

5. Shader Functions: OpenGL uses programmable shaders for advanced rendering techniques. Shader functions allow you to create, compile, link, and use shaders. Functions in this category include glCreateShader(), glShaderSource(), glCompileShader(), glCreateProgram(), and glUseProgram().

6. Texture Functions: These functions are used to manage textures, which are used to apply images or patterns to geometry. Examples include glGenTextures(), glBindTexture(), glTexImage2D(), glTexParameteri(), and glTexSubImage2D().

7. Query Functions: These functions allow you to query information about the OpenGL capabilities and state. They include functions like glGetString(), glGetError(), glGetUniformLocation(), and glGetVertexAttribfv().

These categories are not mutually exclusive, and many OpenGL functions can fall into multiple categories depending on their usage and context.

3. DESIGN AND IMPLEMENTATION

3.1 DESIGN AND FUNCTIONALITIES

To achieve three dimensional effects, open GL software is proposed. It is software which provides a graphical interface. It is a interface between application program and graphics hardware. The advantages are:

- Open GL is designed as a streamlined
  .
- It's a hardware independent interface i.e it can be implemented on many different hardware platforms.

- With openGL we can draw a small set of geometric primitives such as points, lines and polygons etc.

- It provides double buffering which is vital in providing transformations.

- It is event driven software.

- It provides call back function.

3.2 IMPLEMENTATION DETAILS

3.2.1 Software Requirements

- **Operating system** : UBUNTU 10.10
- **Tool Used** : Eclipse
- OPENGL Library
- X86
- X64(WOW)
- Mouse Driver
- Graphics Driver
- C Language

3.2.2 Hardware Requirements

- Microprocessor: 1.0 GHz and above CPU based on either AMD or INTEL Microprocessor Architecture
- Main memory : 512 MB RAM
- Hard Disk : 40 GB
- Hard disk speed in RPM:5400 RPM
- Keyboard: QWERTY Keyboard
- Mouse :2 or 3 Button mouse
- Monitor : 1024 x 768 display resolution

3.2.3 OpenGL Functions Used

- The glColor3f (float, float, float) :- This function will set the current drawing color
- gluOrtho2D (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top):- which defines a two dimensional viewing rectangle in the plane z=0.
- glClear( ):-Takes a single argument that is the bitwise OR of several values indicating which buffer is to be cleared.
- glClearColor ():-Specifies the red, green, blue, and alpha values used by glClear to clear the color buffers.
- GlLoadIdentity( ):-the current matrix with the identity matrix.
- glMatrixMode(mode):-Sets the current matrix mode, mode can be GL_MODELVIEW, GL_PROJECTION or GL_TEXTURE.
- Void glutInit (int *argc, char**argv):-Initializes GLUT, the arguments from main are passed in and can be used by the application.
- 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 and buffering. 3D HOUSE Page | 9 City Engineering College Dept. of CSE 2010-2011 Void glutInitWindowSize (int width, int height):- Specifies the initial position of the topleft corner of the window in pixels
- Int glutCreateWindow (char *title):-A window on the display. The string title can be used to label the window. The return value provides references to the window that can be used when there are multiple windows.
- Void glutMouseFunc(void *f(int button, int state, int x, int y):-Register the mouse callback function f. The callback function returns the button, the state of button after the event and the position of the mouse relative to the top-left corner of the window.
- Void glutKeyboardFunc(void(*func) (void)):-This function is called every time when you press enter key to resume the game or when you press ‘b’ or ‘B’ key to go back to the initial screen or when you press esc key to exit from the application.
- Void glutDisplayFunc (void (*func) (void)):-Register the display function func that is executed when the window needs to be redrawn.

- Void glutSpecialFunc(void(*func)(void)):-This function is called when you press the special keys in the keyboard like arrow keys, function keys etc. In our program, the func is invoked when the up arrow or down arrow key is pressed for selecting the options in the main menu and when the left or right arrow key is pressed for moving the object(car) accordingly.
- glut PostReDisplay ( ) :-which requests that the display callback be executed after the current callback returns.
- Void MouseFunc (void (*func) void)):- This function is invoked when mouse keys are pressed. This function is used as an alternative to the previous function i.e., it is used to move the object(car) to right or left in our program by clicking left and right button respectively.
- Void glutMainLoop ():- Cause the program to enter an event-processing loop. It should be the last statement in main function.

3.2.4 Source Code

```
#include<GL/glut.h>
#include <time.h>
double w=1280,h=720;
double view[3]={2,2,12.9};
double look[3]={2,2,2};
int flag=-1;
void steps(void);
void window(void);
void sgate(void);
void gate(void);
double angle=0,speed=5,maino=0,romo=0,tro=0,mgo=0,sgo=0;
//declarating quadric objects
GLUquadricObj *Cylinder;
GLUquadricObj *Disk;
struct tm *newtime;
time_t ltime;
GLfloat angle1;
//initialisation
void myinit(void)
{
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-1.0,1.0,-1*w/h,1*w/h,1,200.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
//defining new quadric object
 Cylinder = gluNewQuadric();
//to set drawing style
 gluQuadricDrawStyle( Cylinder, GLU_FILL);
//to set automatic normals
 gluQuadricNormals( Cylinder,GLU_SMOOTH);
 Disk = gluNewQuadric();
```

```
 gluQuadricDrawStyle( Disk, GLU_FILL);
 gluQuadricNormals( Disk, GLU_SMOOTH);
GLfloat gam[]={0.2,.2,.2,1};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT,gam);
}
//set material property
void matprop(GLfloat amb[],GLfloat dif[],GLfloat spec[],GLfloat shi[])
{
glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT,amb);
glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,dif);
glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,spec);
glMaterialfv(GL_FRONT_AND_BACK,GL_SHININESS,shi);
}
//to create wall
void wall(double thickness)
{
glPushMatrix();
glTranslated(2,.5*thickness,2);
glScaled(4.0,thickness,4.0);
glutSolidCube(1.0);
glPopMatrix();
}
//to create compound wall
void wall2(double thickness)
{
glPushMatrix();
glTranslated(.8,.5*thickness*4,3.5);
glScaled(1.6,thickness*4,7.0);
glutSolidCube(1.0);
glPopMatrix();
}
//to create earth
void earth(void)
```

```
{
GLfloat ambient[]={1,0,0,1};
GLfloat specular[]={0,1,1,1};
GLfloat diffuse[]={.5,.5,.5,1};
GLfloat shininess[]={50};
matprop(ambient,diffuse,specular,shininess);
GLfloat lightIntensity[]={.7,.7,.7,1};
GLfloat light_position[]={2,5,-3,0};
glLightfv(GL_LIGHT0,GL_POSITION,light_position);
glLightfv(GL_LIGHT0,GL_DIFFUSE,lightIntensity);
glPushMatrix();
glTranslated(0,-.25,0);
glScaled(10000,.5,1000000);
glutSolidCube(1.0);
glPopMatrix();
glFlush();
}
void compound(void)
{
GLfloat ambient[]={1,0,0,1};
GLfloat specular[]={0,1,1,1};
GLfloat diffuse[]={.7,1,.7,1};
GLfloat shininess[]={50};
matprop(ambient,diffuse,specular,shininess);
GLfloat lightIntensity[]={.7,.7,.7,1};
GLfloat light_position[]={2,6,1.5,0};
glLightfv(GL_LIGHT0,GL_POSITION,light_position);
glLightfv(GL_LIGHT0,GL_DIFFUSE,lightIntensity);
//left wall of compound
glPushMatrix();
glPushMatrix();
glTranslated(-4,0,-1-.04);
glRotated(90.0,0,0,1);
```

```
wall2(0.08);
glPopMatrix();
//right wall of compound
glPushMatrix();
glTranslated(8,0,-1-.02);
glRotated(90.0,0,0,1);
wall2(0.08);
glPopMatrix();
//back wall of compound
glPushMatrix();
glTranslated(2,.8,-1);
glRotated(-90,1,0,0);
glScaled(12,.02*4,1.6);
glutSolidCube(1.0);
glPopMatrix();
//front left wall of compound
glPushMatrix();
glTranslated(-3,.8,6-.08);
glRotated(-90,1,0,0);
glScaled(2,.02*4,1.6);
glutSolidCube(1.0);
glPopMatrix();
//front middle wall of compound
glPushMatrix();
glTranslated(2.5,.8,6-.08);
glRotated(-90,1,0,0);
glScaled(6,.02*4,1.6);
glutSolidCube(1.0);
glPopMatrix();
//front right wall of compound
glPushMatrix();
glTranslated(7,.8,6-.08);
glRotated(-90,1,0,0);
```

```
glScaled(2,.02*4,1.6);
glutSolidCube(1.0);
glPopMatrix();
glPopMatrix();
GLfloat ambient2[]={0,1,0,1};
GLfloat specular2[]={1,1,1,1};
GLfloat diffuse2[]={.2,.6,0.1,1};
GLfloat shininess2[]={50};
matprop(ambient2,diffuse2,specular2,shininess2);
//floor
glPushMatrix();
glTranslated(-4,-0.05,-1);
glScaled(3,3,1.7);
wall(0.08);
glPopMatrix();
gate();
 sgate();
 glFlush();
}
void room()
{
GLfloat ambient1[]={1,0,1,1};
GLfloat specular1[]={1,1,1,1};
GLfloat diffuse1[]={0.5,0.5,0.5,1};
GLfloat mat_shininess[]={50};
 matprop(ambient1,diffuse1,specular1,mat_shininess);
glPushMatrix();
glTranslated(.5,4,.5);
//roof
glPushMatrix();
glTranslated(-.02*4,.7*3.9,-.02*4);
glScaled(.6+.02,1.5,.5+.02+.1);
wall(0.08);
```

```
glPopMatrix();
GLfloat ambient2[]={1,0,0,1};
GLfloat specular2[]={1,1,1,1};
GLfloat diffuse2[]={1,1,.7,1};
GLfloat shininess1[]={50};
matprop(ambient2,diffuse2,specular2,shininess1);
//left wall
glPushMatrix();
glTranslated(0,0,-.02);
glScaled(1,.7,.5);
glTranslated(-.08,0,0);
glScaled(.62,.7,1);
glRotated(-90.0,1,0,0);
wall(0.08);
glPopMatrix();
//front wall
glPushMatrix();
glTranslated(-0.08,0,2);
glScaled(.5,.7,1);
glRotated(-90.0,1,0,0);
wall(0.08);
GLfloat ambient[]={1,0.5,.5,1};
GLfloat specular[]={1,1,1,1};
GLfloat diffuse[]={1,0.5,0.5,1};
 matprop(ambient,diffuse,specular,mat_shininess);
//door
glPushMatrix();
glTranslated(2.3,0,(2-.05));
glTranslated(1.927,0,2);
glScaled(.09,.525,1);
glRotated(-90.0,1,0,0);
wall(0.02);
glPopMatrix();
```

```
glPushMatrix();
glTranslated(2.3,0,2-.05);
glScaled(.6,.7,.8);
glRotated(-90,1,0,0);
gluCylinder(Cylinder, 0.05, 0.05, 3, 16, 16);
glPopMatrix();
glPopMatrix();
glPopMatrix();
}
void tankwall(float thk)
{
glTranslated(.5,.5*thk,.5);
glScaled(1,thk,1);
glutSolidCube(1);
}
void watertank(void)
{
float thk=.04,hght=1,wdth=1,bdth=1;
GLfloat ambient1[]={.5,0,1,1};
GLfloat specular1[]={1,1,1,1};
GLfloat diffuse1[]={.5,.8,1,1};
GLfloat mat_shininess[]={50};
 matprop(ambient1,diffuse1,specular1,mat_shininess);
glPushMatrix();
glTranslated(1.5,4+4*.7,1.5);
glScaled(.8,.8,.8);
//tank floor
glPushMatrix();
glScaled(wdth,1,bdth);
tankwall(thk);
glPopMatrix();
//tank left wall
glPushMatrix();
```

```
glScaled(1,hght,bdth);
glRotated(90.0,0,0,1);
tankwall(thk);
glPopMatrix();
//tank right wall
glScaled(wdth,hght,1);
glRotated(-90.0,1,0,0);
tankwall(0.04);
glPopMatrix();
//tank roof
glPushMatrix();
glTranslated(-thk,hght,0);
glScaled(wdth*.8,1,bdth);
tankwall(0.04);
glPopMatrix();
glPushMatrix();
glTranslated(wdth*.8-thk,hght,0);
glScaled(wdth*.2+thk,1,bdth*.6);
tankwall(0.04);
glPopMatrix();
glPopMatrix();
}
void menu()
{
int sub_menu1=glutCreateMenu(fan_menu);
glutAddMenuEntry("on/off fan(s)",1);
glutAddMenuEntry("speed up(up arrow)",2);
glutAddMenuEntry("speed down(down arrow)",3);
int sub_menu2=glutCreateMenu(door_menu);
glutAddMenuEntry("main door(q)",1);
glutAddMenuEntry("ground floor room door(o)",2);
int sub_menu4=glutCreateMenu(house_view);
glutAddMenuEntry("front view(j)",3);
```

```
glutAddMenuEntry("top view(t)",2);
glutAddMenuEntry("inside view(i)",1);
glutAddMenuEntry("back view(k)",4);
glutCreateMenu(main_menu);
glutAddMenuEntry("quit",1);
glutAddSubMenu("fan menu",sub_menu1);
glutAddSubMenu("open/close door",sub_menu2);
glutAddSubMenu("open/close gate",sub_menu3);
glutAddSubMenu("house view",sub_menu4);
glutAttachMenu(GLUT_RIGHT_BUTTON);
}
void main(int argc,char**argv)
{
glutInit(&argc,argv);//to initialize the glut library
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(w,h);
glutInitWindowPosition(0,0);
glutCreateWindow("er");
myinit();
glutDisplayFunc(display);
glutKeyboardFunc(Keyboard);
glutSpecialFunc(mySpecialKeyFunc);
menu();
glutFullScreen();//to see o/p in full screen on monitor
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glShadeModel(GL_SMOOTH);//smooth shaded
glEnable(GL_DEPTH_TEST);//to remove hidden surface
glEnable(GL_NORMALIZE);//to make normal vector to unit vector
glClearColor(0,.3,.8,0);
glViewport(0,0,w,h);
glutMainLoop();
return 0;
}
```

4.SNAPSHOTS

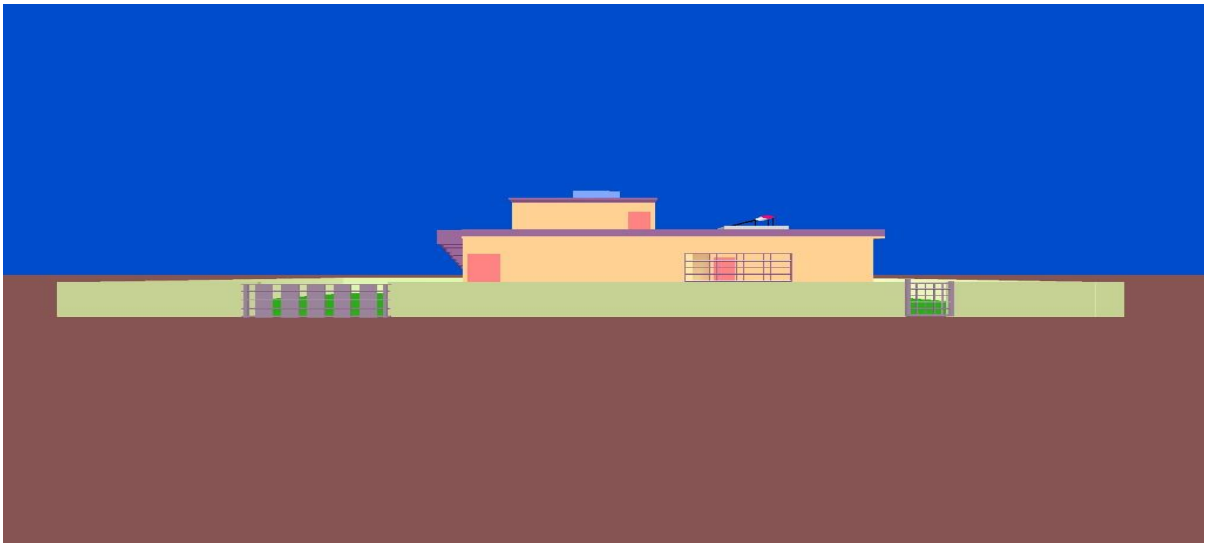

Figure1.After Run the Code

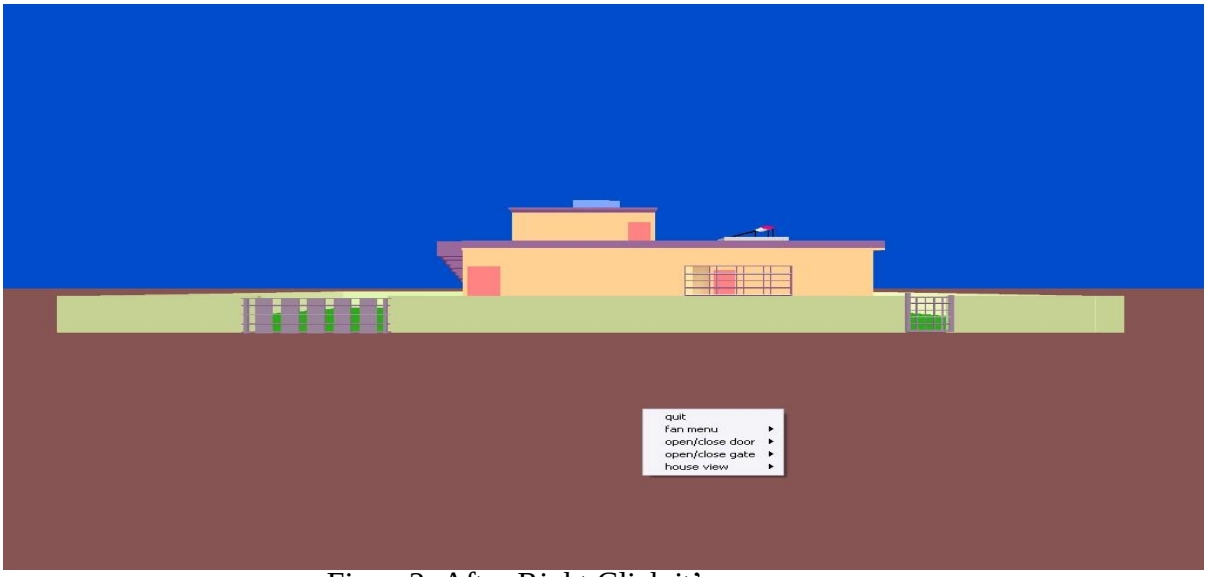

Figure2. After Right Click it's Showing options

Figure3.After Selected inner view of house

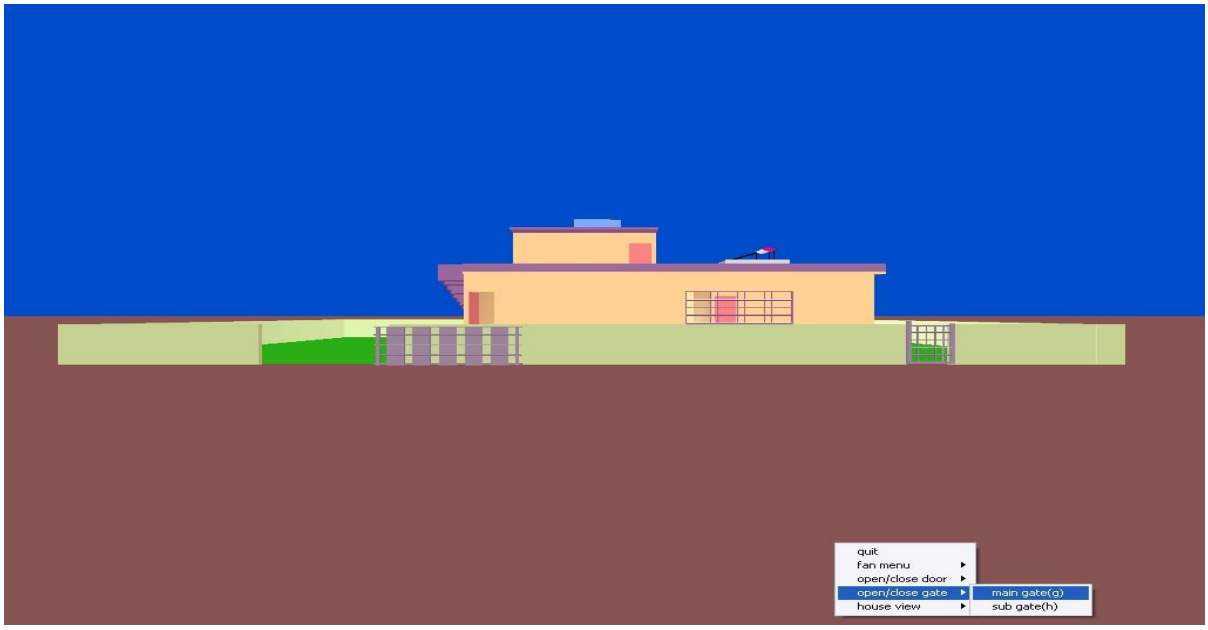

Figure4.After Selecting Main door to open

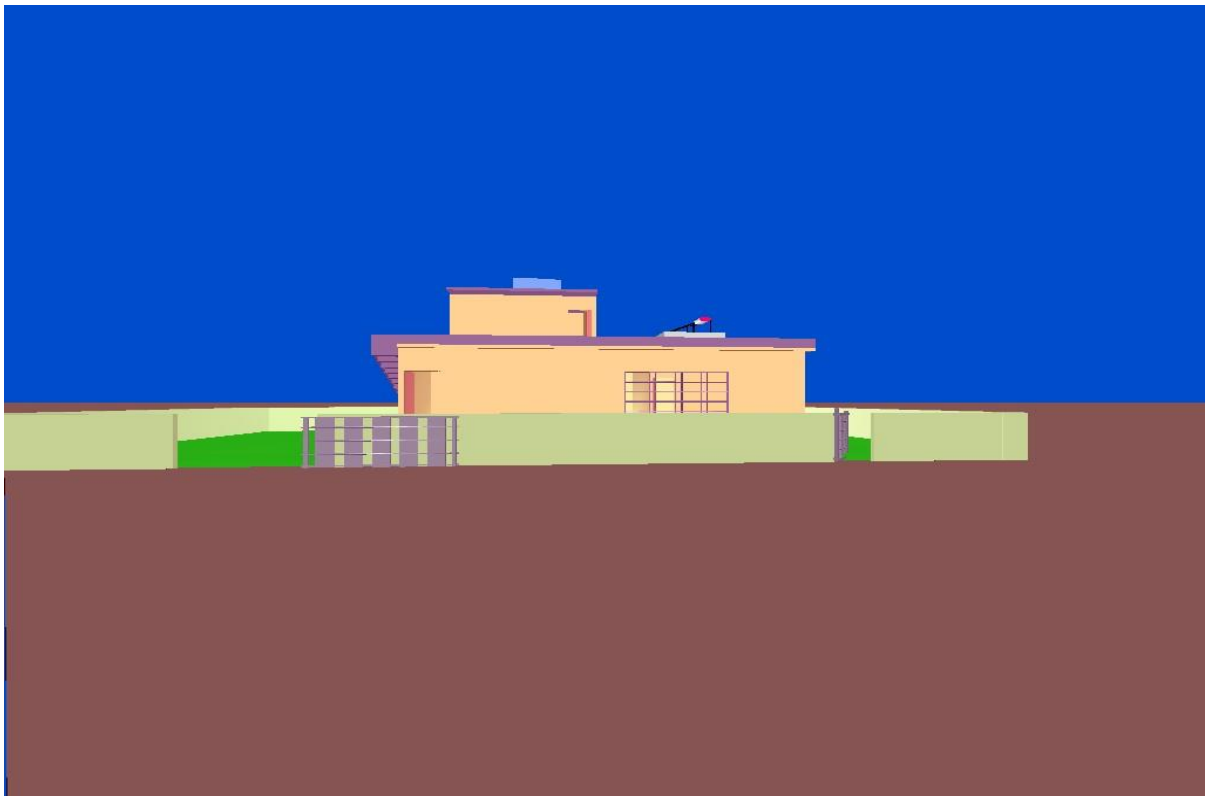

Figure5.After Selected inner view of house

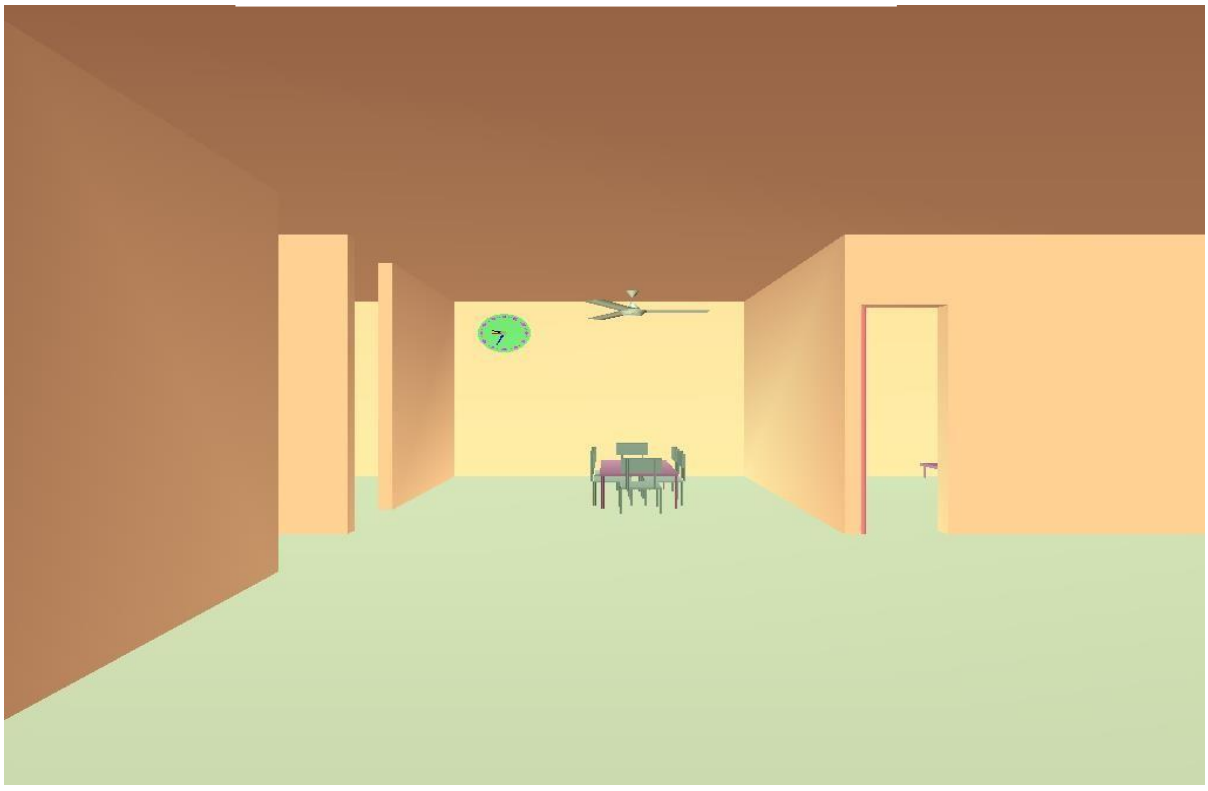

Figure6.After Selected inner view of house

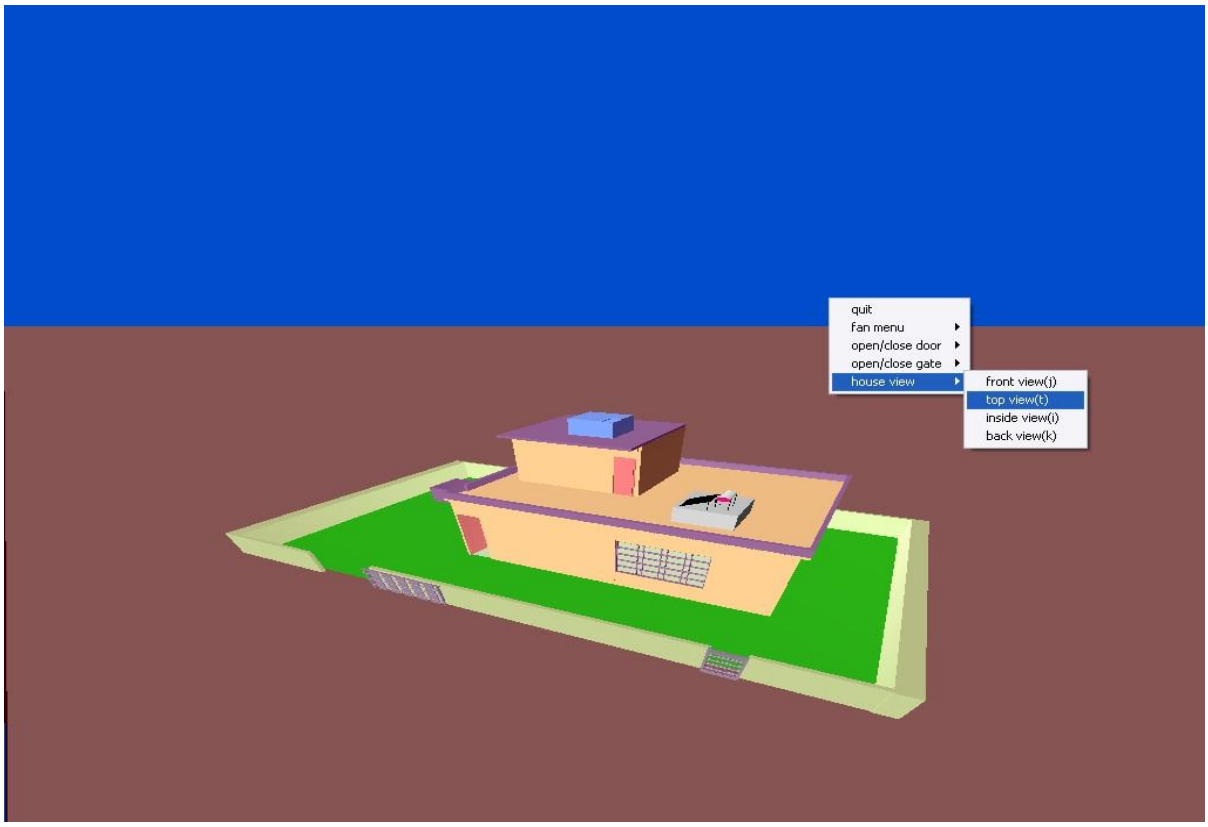

Figure7.House Top view showing

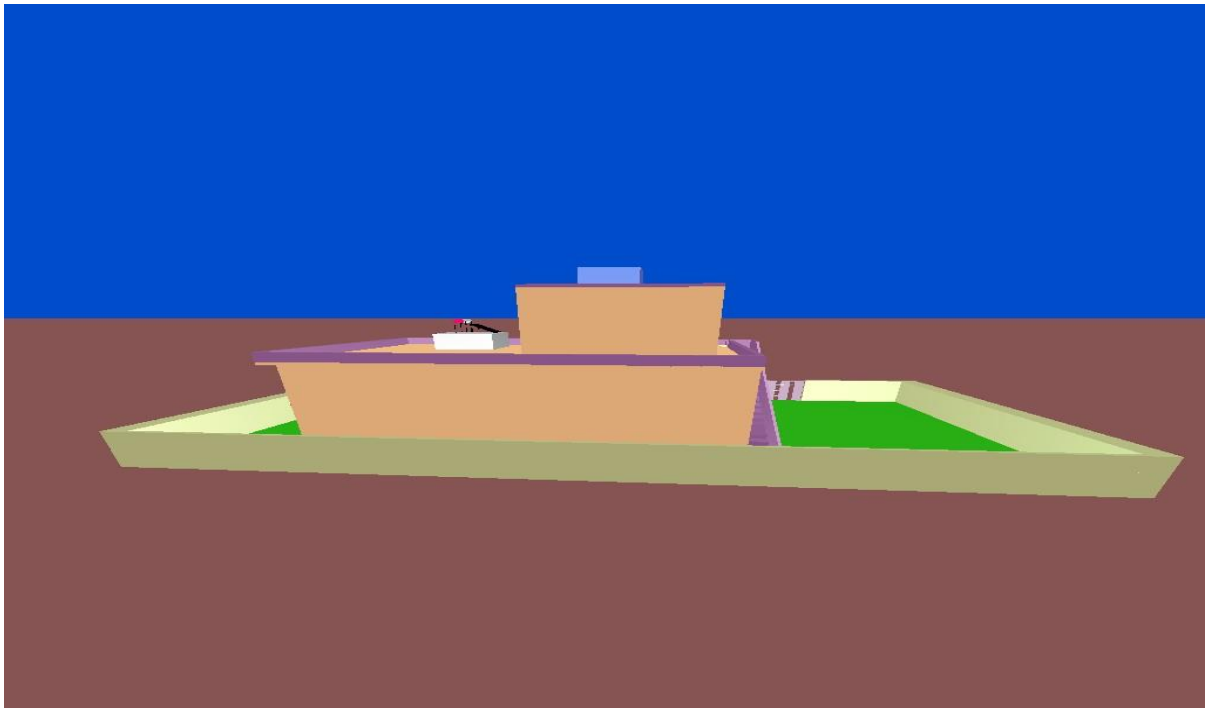

Figure8.Back View of House

5. CONCLUSION

The 3D House has been tested under Windows XP and has been found to provide ease of use and manipulation to the user. The 3D house created for the Windows XP operating system can be used to draw lines, boxes, circles, ellipses, and polygons. It has a very simple and aesthetic user interface.

We found designing and developing this 3D House as a very interesting and learning experience. It helped us to learn about computer graphics, design of Graphical User Interfaces, interface to the user, user interaction handling and screen management. The graphics editor provides all and more than the features that have been detailed in the university syllabus.

BIBLIOGRAPHY

- Edward Angel's Interactive Computer Graphics Pearson Education 5th Editio

- Interactive computer Graphics --A top down approach using open GL—by Edward Angle

- Jackie .L. Neider, Mark Warhol, Tom.R.Davis, "OpenGL Red Book", Second Revised Edition, 2005.

- Website - https://www.vtupulse.com/cgv-mini-projects/computer-graphics-and-visualization-mini-project/