Assignment - 9

CS-352: Computer Graphics & Visualization

KULDEEP SINGH
190001030

02nd April 2022

Ques - 1: 3D-CITY + HOUSE

```
// KULDEEP SINGH

// 190001030


#include <bits/stdc++.h>

#include <GL/gl.h>

#include <GL/glut.h>


using namespace std;


#define PI 3.14


float reference_point = 0.0;


int intensity = 0;


// Dimensions

float house_width = 6.0;

float house_height_ = 4.0;

float length_of_house = -4.0;

float width_of_roof = house_width + 1.5;
float height_of_roof = 8.0;

float extra_depth = 0.01;

float door_width = 0.8;

float door_height = 4.0;

float window_size = 0.8;


// colors

float house_red = 245.0 / 255, house_green = 175.0 / 255, house_blue =
103.0 / 255;

float door_red = 255 / 255, door_green = 255 / 255, door_blue = 0 / 255;

float window_red = 160 / 255.0, window_green = 82 / 255.0, window_blue =
45 / 255.0;

float roof_red = 139 / 255.0, roof_green = 69 / 255.0, roof_blue = 19 /
255.0;


// a starting point for comparison

int start_x = -1;

int start_y = -1;


// amount needed to lower the intensity

float reduce_intensity = 0.0;


// Rotational angles

float Theta = PI / 2;

float Phi = 0;
float Radius = 25;

float front_back = 0;

float left_right = 0;

int flip = 0;

int window1 = 0;

int window2 = 0;


bool mouse_pressed = false;


float unit_change = (2 * PI * Radius) / 1000;


float camera[3] = {0, 0, 25};


void fill_color(float x, float y, float z)

{

    glColor3f(x / 255, y / 255, z / 255);

}


// This function handles changes in window size.

void update_window_size(int width, int height)

{

    if (height == 0)

        height = 1;

    float ratio = (width * 1.0) / height;
    glMatrixMode(GL_PROJECTION);

    glLoadIdentity();

    glViewport(0, 0, width, height);

    gluPerspective(60.0, ratio, 0.1, 1000.0);

    glMatrixMode(GL_MODELVIEW);

}


void wall_Window(int x)

{

    glBegin(GL_POLYGON);

    glVertex3f(x, house_height_, house_width);

    glVertex3f(x, window_size, house_width);

    glVertex3f(x, window_size, -house_width);

    glVertex3f(x, house_height_, -house_width);

    glEnd();

    glBegin(GL_POLYGON);

    glVertex3f(x, -house_height_, house_width);

    glVertex3f(x, -window_size, house_width);

    glVertex3f(x, -window_size, -house_width);

    glVertex3f(x, -house_height_, -house_width);

    glEnd();

    glBegin(GL_POLYGON);

    glVertex3f(x, window_size, house_width);

    glVertex3f(x, window_size, window_size);
    glVertex3f(x, -window_size, window_size);

    glVertex3f(x, -window_size, house_width);

    glEnd();

    glBegin(GL_POLYGON);

    glVertex3f(x, window_size, -house_width);

    glVertex3f(x, window_size, -window_size);

    glVertex3f(x, -window_size, -window_size);

    glVertex3f(x, -window_size, -house_width);

    glEnd();

}


void wall_Door(int z)

{

    glBegin(GL_POLYGON);

    glVertex3f(house_width, house_height_, z);

    glVertex3f(house_width, -house_height_ + door_height, z);

    glVertex3f(-house_width, -house_height_ + door_height, z);

    glVertex3f(-house_width, house_height_, z);

    glEnd();

    glBegin(GL_POLYGON);

    glVertex3f(house_width, -house_height_ + door_height, z);

    glVertex3f(house_width, -house_height_, z);

    glVertex3f(door_width, -house_height_, z);

    glVertex3f(door_width, -house_height_ + door_height, z);
    glEnd();

    glBegin(GL_POLYGON);

    glVertex3f(-house_width, -house_height_ + door_height, z);

    glVertex3f(-house_width, -house_height_, z);

    glVertex3f(-door_width, -house_height_, z);

    glVertex3f(-door_width, -house_height_ + door_height, z);

    glEnd();

}


void cuboid(float x, float y, float z, float X, float Y, float Z)

{

    glBegin(GL_POLYGON);

    glVertex3f(-x + X, Y + y, z + Z);

    glVertex3f(x + X, Y + y, z + Z);

    glVertex3f(x + X, Y - y, z + Z);

    glVertex3f(-x + X, Y - y, z + Z);

    glEnd();

    glBegin(GL_POLYGON);

    glVertex3f(-x + X, Y + y, -z + Z);

    glVertex3f(x + X, Y + y, -z + Z);

    glVertex3f(x + X, Y - y, -z + Z);

    glVertex3f(-x + X, Y - y, -z + Z);

    glEnd();

    glBegin(GL_POLYGON);
    glVertex3f(-x + X, Y + y, z + Z);

    glVertex3f(-x + X, Y + y, -z + Z);

    glVertex3f(-x + X, Y - y, -z + Z);

    glVertex3f(-x + X, Y - y, z + Z);

    glEnd();

    glBegin(GL_POLYGON);

    glVertex3f(x + X, Y + y, z + Z);

    glVertex3f(x + X, Y + y, -z + Z);

    glVertex3f(x + X, Y - y, -z + Z);

    glVertex3f(x + X, Y - y, z + Z);

    glEnd();

    glBegin(GL_POLYGON);

    glVertex3f(-x + X, Y + y, z + Z);

    glVertex3f(-x + X, Y + y, -z + Z);

    glVertex3f(x + X, Y + y, -z + Z);

    glVertex3f(x + X, Y + y, z + Z);

    glEnd();

    glBegin(GL_POLYGON);

    glVertex3f(-x + X, Y - y, z + Z);

    glVertex3f(-x + X, Y - y, -z + Z);

    glVertex3f(x + X, Y - y, -z + Z);

    glVertex3f(x + X, Y - y, z + Z);

    glEnd();

}
```

```
void table(float x, float y, float z)
{
    cuboid(1.8, 0.15, 1.8, x, y + 1.5, z);
    cuboid(0.25, 1.5, 0.25, x - 1.5, y, z - 1.5);
    cuboid(0.25, 1.5, 0.25, x + 1.5, y, z - 1.5);
    cuboid(0.25, 1.5, 0.25, x - 1.5, y, z + 1.5);
    cuboid(0.25, 1.5, 0.25, x + 1.5, y, z + 1.5);
}

void bed(float x, float y, float z)
{
    cuboid(1.5, 0.7, 0.15, x, y + 1, z + 3);
    cuboid(1.5, 0.1, 3.2, x, y + 0.6, z);
    cuboid(0.15, 0.6, 0.15, x - 1.4, y, z - 3.0);
    cuboid(0.15, 0.6, 0.15, x + 1.4, y, z - 3.0);
    cuboid(0.15, 0.6, 0.15, x - 1.4, y, z + 3.0);
    cuboid(0.15, 0.6, 0.15, x + 1.4, y, z + 3.0);
}

void door()
{
    if (flip)
    {
        float x = -door_width;
        float z = house_width;
        glBegin(GL_POLYGON);
        glVertex3f(x, -house_height_ + door_height, z);
        glVertex3f(x, -house_height_ + door_height, z + 2 * door_width);
        glVertex3f(x, -house_height_, z + 2 * door_width);
        glVertex3f(x, -house_height_, z);
        glEnd();
    }
    else
    {
        float z = house_width;
        glBegin(GL_POLYGON);
        glVertex3f(-door_width, -house_height_ + door_height, z);
        glVertex3f(door_width, -house_height_ + door_height, z);
        glVertex3f(door_width, -house_height_, z);
        glVertex3f(-door_width, -house_height_, z);
        glEnd();
    }
}

void window()
{
    if (window1)
```

```
{
    float x = -house_width;
    float z = -window_size;
    glBegin(GL_POLYGON);
    glVertex3f(x, window_size, z);
    glVertex3f(x - window_size * 2, window_size, z);
    glVertex3f(x - window_size * 2, -window_size, z);
    glVertex3f(x, -window_size, z);
    glEnd();
}
else
{
    float x = -house_width;
    glBegin(GL_POLYGON);
    glVertex3f(x, window_size, window_size);
    glVertex3f(x, window_size, -window_size);
    glVertex3f(x, -window_size, -window_size);
    glVertex3f(x, -window_size, window_size);
    glEnd();
}
if (window2)
{
    float x = house_width;
    float z = -window_size;
```

```
        glBegin(GL_POLYGON);
        glVertex3f(x, window_size, z);
        glVertex3f(x + window_size * 2, window_size, z);
        glVertex3f(x + window_size * 2, -window_size, z);
        glVertex3f(x, -window_size, z);
        glEnd();
    }
    else
    {
        float x = house_width;
        glBegin(GL_POLYGON);
        glVertex3f(x, window_size, window_size);
        glVertex3f(x, window_size, -window_size);
        glVertex3f(x, -window_size, -window_size);
        glVertex3f(x, -window_size, window_size);
        glEnd();
    }
}

void tree(float X, float Y, float Z)
{
    fill_color(0, 255, 0);
    glPushMatrix();
    glTranslated(X, Y + 15, Z);
    glRotated(90, -1.0, 0.0, 0.0);

    glutSolidCone(3, 6, 50, 50);

    glPopMatrix();

    glPushMatrix();

    glTranslated(X, Y + 13, Z);

    glRotated(90, -1.0, 0.0, 0.0);

    glutSolidCone(3.5, 5, 50, 50);

    glPopMatrix();

    glPushMatrix();

    glTranslated(X, Y + 11, Z);

    glRotated(90, -1.0, 0.0, 0.0);

    glutSolidCone(4, 4, 50, 50);

    glPopMatrix();

    glPushMatrix();

    fill_color(200, 100, 20);

    glTranslated(X, Y + 7, Z);

    GLUquadricObj *quadratic;

    quadratic = gluNewQuadric();

    glRotatef(90.0f, -1.0f, 0.0f, 0.0f);

    gluCylinder(quadratic, 1.0, 1.0, 6.0, 32, 32);

    glPopMatrix();

}


void ground(float X)
```

```
{
    fill_color(29, 110, 26);
    glBegin(GL_POLYGON);
    glVertex3f(-X, -house_height_, -X);
    glVertex3f(X, -house_height_, -X);
    glVertex3f(X, -house_height_, X);
    glVertex3f(-X, -house_height_, X);
    glEnd();
}

void road(float l)
{
    fill_color(128, 128, 128);
    glBegin(GL_POLYGON);
    glVertex3f(-l, -house_height_ + 0.01, 3 * house_height_);
    glVertex3f(l, -house_height_ + 0.01, 3 * house_height_);
    glVertex3f(l, -house_height_ + 0.01, 6 * house_height_);
    glVertex3f(-l, -house_height_ + 0.01, 6 * house_height_);
    glEnd();
}

// display
void house()
{
    // ground

    ground(1000);


    // back_face

    fill_color(255, 255, 255);

    glBegin(GL_POLYGON);


    glVertex3f(reference_point + house_width, reference_point -
house_height_, reference_point - house_width);

    glVertex3f(reference_point + house_width, reference_point +
house_height_, reference_point - house_width);

    glVertex3f(reference_point - house_width, reference_point +
house_height_, reference_point - house_width);

    glVertex3f(reference_point - house_width, reference_point -
house_height_, reference_point - house_width);

    glEnd();


    glColor3f(house_red, house_green, house_blue);


    // front_face

    wall_Door(house_width);


    // right_face

    wall_Window(house_width);
    // left_face
    wall_Window(-house_width);


    // bottom face
    fill_color(255, 0, 0);


    glBegin(GL_POLYGON);
    glVertex3f(reference_point + house_width, reference_point -
house_height_ + 0.01, reference_point + house_width);
    glVertex3f(reference_point + house_width, reference_point -
house_height_ + 0.01, reference_point - house_width);
    glVertex3f(reference_point - house_width, reference_point -
house_height_ + 0.01, reference_point - house_width);
    glVertex3f(reference_point - house_width, reference_point -
house_height_ + 0.01, reference_point + house_width);
    glEnd();


    // Roof
    glColor3f(roof_red, roof_green, roof_blue);
    glBegin(GL_POLYGON);
    glVertex3f(reference_point + width_of_roof, reference_point +
house_height_, reference_point + width_of_roof);
    glVertex3f(reference_point + width_of_roof, reference_point +
house_height_, reference_point - width_of_roof);
    glVertex3f(reference_point - width_of_roof, reference_point +
house_height_, reference_point - width_of_roof);
```

```
    glVertex3f(reference_point - width_of_roof, reference_point +
house_height_, reference_point + width_of_roof);

    glEnd();

    glBegin(GL_POLYGON);

    glVertex3f(reference_point + width_of_roof, reference_point +
house_height_, reference_point + width_of_roof);

    glVertex3f(reference_point + width_of_roof, reference_point +
house_height_, reference_point - width_of_roof);

    glVertex3f(reference_point, height_of_roof, reference_point);

    glEnd();

    glBegin(GL_POLYGON);

    glVertex3f(reference_point + width_of_roof, reference_point +
house_height_, reference_point - width_of_roof);

    glVertex3f(reference_point - width_of_roof, reference_point +
house_height_, reference_point - width_of_roof);

    glVertex3f(reference_point, height_of_roof, reference_point);

    glEnd();

    glBegin(GL_POLYGON);

    glVertex3f(reference_point, height_of_roof, reference_point);

    glVertex3f(reference_point - width_of_roof, reference_point +
house_height_, reference_point - width_of_roof);

    glVertex3f(reference_point - width_of_roof, reference_point +
house_height_, reference_point + width_of_roof);

    glEnd();

    glBegin(GL_POLYGON);

    glVertex3f(reference_point + width_of_roof, reference_point +
house_height_, reference_point + width_of_roof);

    glVertex3f(reference_point, height_of_roof, reference_point);

    glVertex3f(reference_point - width_of_roof, reference_point +
house_height_, reference_point + width_of_roof);

    glEnd();


    // table

    table(-house_width + 2, -house_height_ + 1.5, -house_width + 2);

    bed(0.5 * house_width, -house_height_ + 0.6, 0);

    door();

    window();

}


void light(float X, float Y, float Z)

{

    GLfloat position0[] = {X, Y, Z, 0.0};


    GLfloat ambient0[] = {0.0, 0.0, 0.0, 1.0};

    GLfloat specular0[] = {1.0, 1.0, 1.0, 1.0};

    GLfloat diffuse0[] = {1.0, 1.0, 1.0, 1.0};


    glEnable(GL_LIGHTING);


    glEnable(GL_COLOR_MATERIAL);
```

```
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

GLfloat specular_material[] = {0, 0, 0, 1};
GLfloat emission_material[] = {0, 0, 0, 1};
glMaterialfv(GL_FRONT, GL_SPECULAR, specular_material);
glMaterialfv(GL_FRONT, GL_EMISSION, emission_material);

if (intensity == 1)
{
    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_POSITION, position0);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient0);
    glLightfv(GL_LIGHT0, GL_SPECULAR, specular0);
}
else
{
    glDisable(GL_LIGHT0);
}
if (intensity == 2)
{
    glEnable(GL_LIGHT1);
    glLightfv(GL_LIGHT1, GL_POSITION, position0);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, diffuse0);

    glLightfv(GL_LIGHT1, GL_AMBIENT, ambient0);

    glLightfv(GL_LIGHT1, GL_SPECULAR, specular0);

}

if (intensity == 3)

{

    glEnable(GL_LIGHT2);

    glLightfv(GL_LIGHT2, GL_POSITION, position0);

    glLightfv(GL_LIGHT2, GL_DIFFUSE, diffuse0);

    glLightfv(GL_LIGHT2, GL_AMBIENT, ambient0);

    glLightfv(GL_LIGHT2, GL_SPECULAR, specular0);

}

if (intensity == 4)

{

    glEnable(GL_LIGHT3);

    glLightfv(GL_LIGHT3, GL_POSITION, position0);

    glLightfv(GL_LIGHT3, GL_DIFFUSE, diffuse0);

    glLightfv(GL_LIGHT3, GL_AMBIENT, ambient0);

    glLightfv(GL_LIGHT3, GL_SPECULAR, specular0);

}

if (intensity == 5)

{

    glEnable(GL_LIGHT4);

    glLightfv(GL_LIGHT4, GL_POSITION, position0);
```

```
        glLightfv(GL_LIGHT4, GL_DIFFUSE, diffuse0);

        glLightfv(GL_LIGHT4, GL_AMBIENT, ambient0);

        glLightfv(GL_LIGHT4, GL_SPECULAR, specular0);

    }

    if (intensity == 6)

    {

        glEnable(GL_LIGHT5);

        glLightfv(GL_LIGHT5, GL_POSITION, position0);

        glLightfv(GL_LIGHT5, GL_DIFFUSE, diffuse0);

        glLightfv(GL_LIGHT5, GL_AMBIENT, ambient0);

        glLightfv(GL_LIGHT5, GL_SPECULAR, specular0);

    }


}


// Function handles the rotation of house

void draw()

{

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);


    glLoadIdentity();


    glTranslatef(0 + left_right, 0 + front_back, -1.0 * Radius);

    camera[0] += front_back * sin(Phi);
    camera[2] -= front_back * cos(Phi);

    front_back = 0;

    float lookat[] = {Radius * sin(Phi) + camera[0], Radius * cos(Theta) +
camera[1], -Radius * cos(Phi) + camera[2]};

    cout << "Radius " << Radius << endl;

    gluLookAt(camera[0], camera[1], camera[2], /* look from camera XYZ */

              lookat[0], lookat[1], lookat[2], /* look at the origin */

              0, 1, 0);                        /* positive Y up vector */

    cout << "see at " << lookat[0] << " " << lookat[1] << " " << lookat[2]
<< endl;

    cout << "Phi " << Phi << endl;


    glPushMatrix();

    light(30, 30, 30);

    ground(1000);

    tree(5 * house_width, -3 * house_height_, house_width);

    house();

    road(300);


    glPopMatrix();


    glutSwapBuffers();

}
```

```
// Activate rotation if the left mouse button is pressed.
void track_mouse_press(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON)
    {
        if (state == GLUT_UP)
        {
            mouse_pressed = false;
            start_x = -1;
            start_y = -1;
        }
        else
        {
            mouse_pressed = true;
        }
    }
    if (state == GLUT_DOWN)
    {
        // for scroll
        switch (button)
        {
        case 3:
            Radius -= 0.5;
            break;
        case 4:
            Radius += 0.5;
            break;
        default:
            break;
        }
    }
}

// adjusting the angle based on the current mouse position
void track_mouse(int x, int y)
{
    if (start_x == -1)
        start_x = x;
    if (start_y == -1)
        start_y = y;
    if (mouse_pressed)
    {

        Theta += (y - start_y) * unit_change * 0.015;

        Phi -= (x - start_x) * unit_change * 0.015;

        if (Phi > 2 * PI)
```

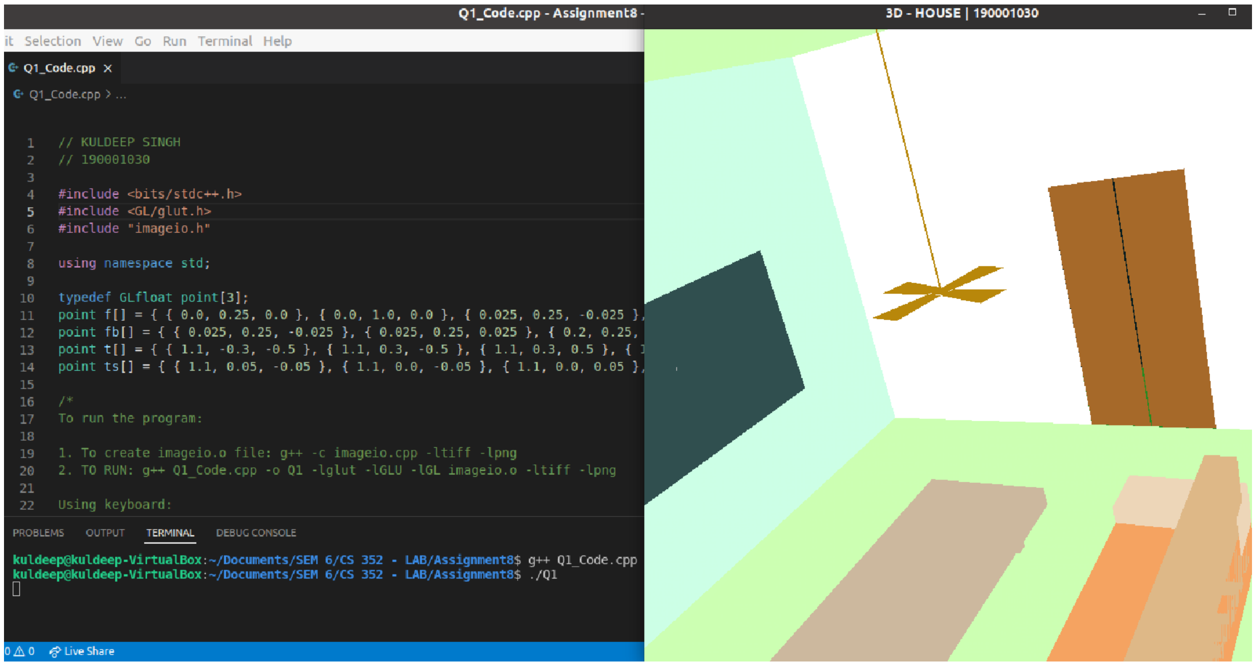
```
            Phi -= 2 * PI;
        if (Phi < 0)
            Phi += 2 * PI;
        if (Theta > 2 * PI)
            Theta -= 2 * PI;
        if (Theta < 0)
            Theta += 2 * PI;
    }
    start_x = x;
    start_y = y;
}

void processSpecialKeys(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 'z':
        intensity += 1;
        break;
    case 'y':
        intensity -= 1;
        break;
    }
    switch (key)
```

```
{
// case 'a':
//     left_right += 1;
//     break;
case 'w':
    front_back += 1;
    break;
case 's':
    front_back -= 1;
    break;
    // case 'd':
    //     left_right -= 1;
    //     break;
}
switch (key)
{
case 'f':
    flip = !flip;
    break;
case 'c':
    window1 = !window1;
    break;
case 'v':
    window2 = !window2;
```

```
        break;
    }

    glutPostRedisplay();
}

void processSpecialKeys(int key, int x, int y)
{
    switch (key)
    {
    case GLUT_KEY_UP:
        Theta -= unit_change * 0.3;
        break;
    case GLUT_KEY_DOWN:
        Theta += unit_change * 0.3;
        break;
    }
    switch (key)
    {
    case GLUT_KEY_RIGHT:
        if (Theta >= (3 * PI / 2) || Theta <= (PI / 2))
            Phi -= unit_change * 0.3;
        else
            Phi += unit_change * 0.3;
        break;

    case GLUT_KEY_LEFT:

        if (Theta >= (3 * PI / 2) || Theta <= (PI / 2))

            Phi += unit_change * 0.3;

        else

            Phi -= unit_change * 0.3;

        break;

    }

    if (Phi > 2 * PI)

        Phi -= 2 * PI;

    if (Phi < 0)

        Phi += 2 * PI;

    if (Theta > 2 * PI)

        Theta -= 2 * PI;

    if (Theta < 0)

        Theta += 2 * PI;

}


int main(int C, char *V[])

{

    glutInit(&C, V);

    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);

    glutInitWindowPosition(300, 300);

    glutInitWindowSize(1000, 1000);
    glutCreateWindow("3D - House | 190001030");

    glutDisplayFunc(draw);
    glutReshapeFunc(update_window_size);
    glutIdleFunc(draw);

    glClearColor(128 / 255.0, 128 / 255.0, 255 / 255.0, 0.0);

    glutMouseFunc(track_mouse_press);
    glutMotionFunc(track_mouse);
    glutKeyboardFunc(processSpecialKeys);
    glutSpecialFunc(processSpecialKeys);

    glEnable(GL_DEPTH_TEST);
    glutMainLoop();

    return 0;
}
```

Output:

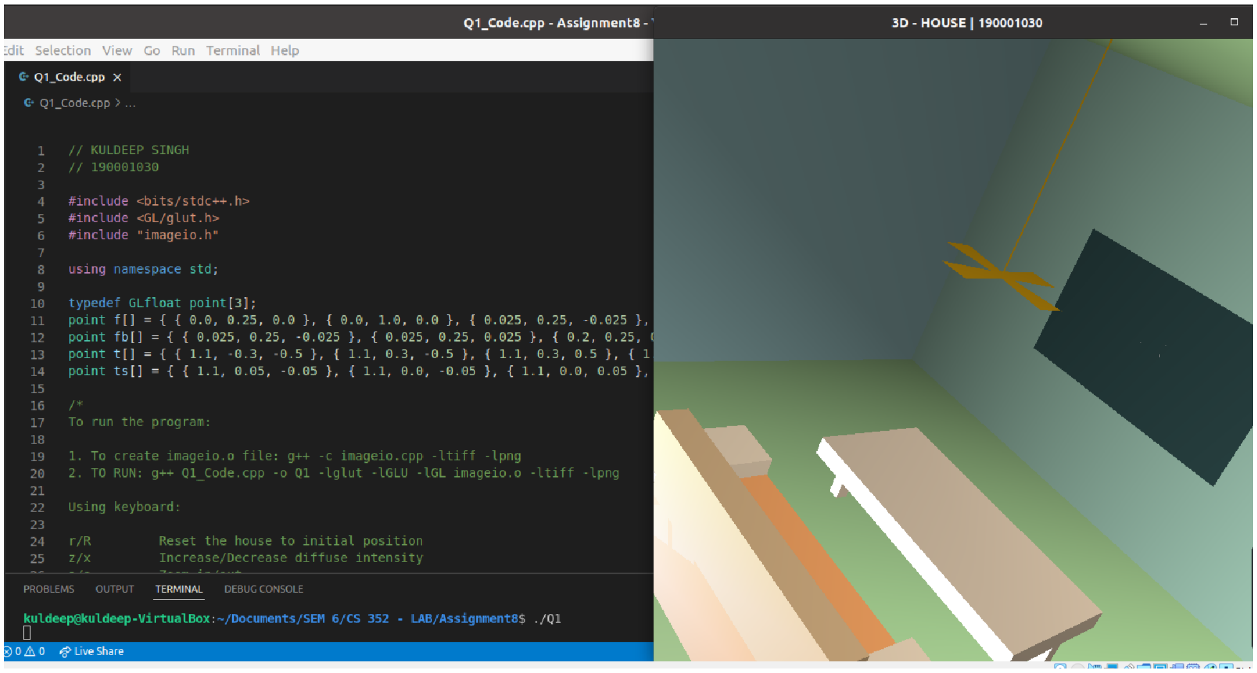
INSIDE VIEW (LIGHTS-ON): FAN, SOFA, TV, TABLE



```cpp
// KULDEEP SINGH
// 190001030

#include <bits/stdc++.h>
#include <GL/glut.h>
#include "imageio.h"

using namespace std;

typedef GLfloat point[3];
point f[] = { { 0.0, 0.25, 0.0 }, { 0.0, 1.0, 0.0 }, { 0.025, 0.25, -0.025 },
point fb[] = { { 0.025, 0.25, -0.025 }, { 0.025, 0.25, 0.025 }, { 0.2, 0.25,
point t[] = { { 1.1, -0.3, -0.5 }, { 1.1, 0.3, -0.5 }, { 1.1, 0.3, 0.5 }, { 1
point ts[] = { { 1.1, 0.05, -0.05 }, { 1.1, 0.0, -0.05 }, { 1.1, 0.0, 0.05 },

/*
To run the program:

1. To create imageio.o file: g++ -c imageio.cpp -ltiff -lpng
2. TO RUN: g++ Q1_Code.cpp -o Q1 -lglut -lGLU -lGL imageio.o -ltiff -lpng

Using keyboard:
```

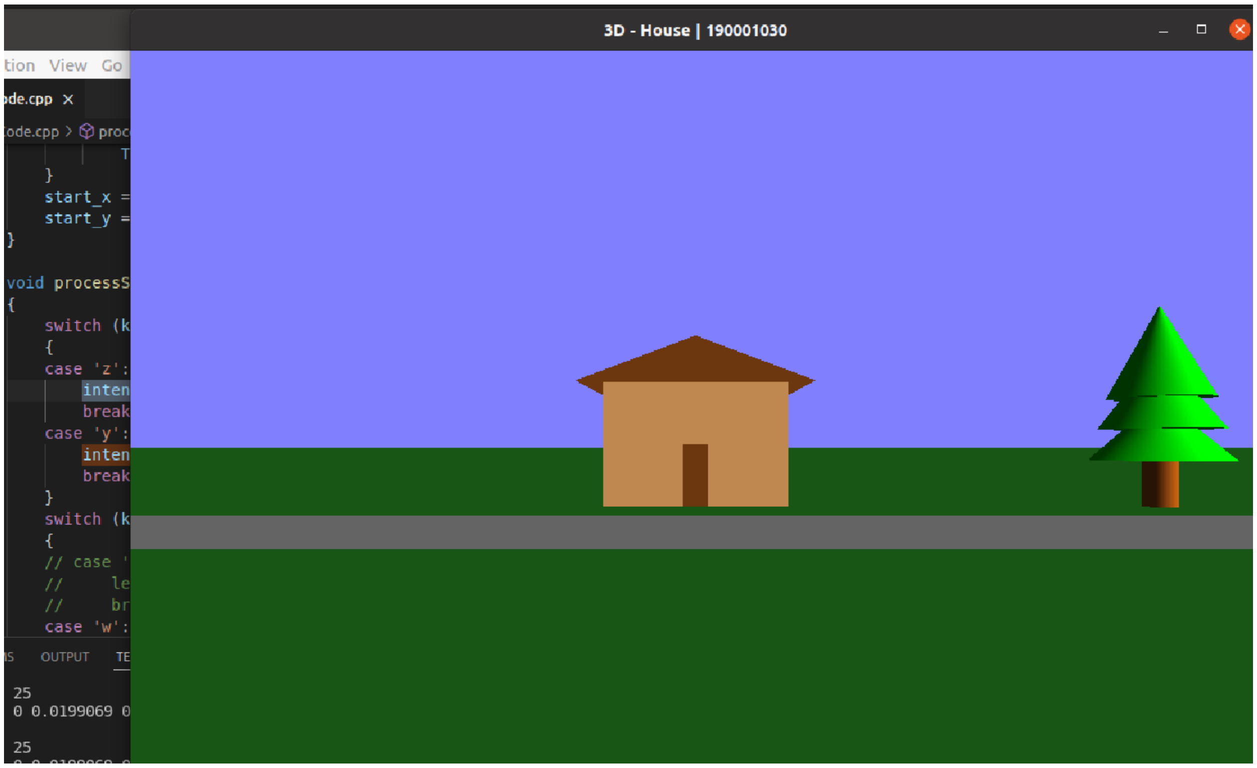
```
kuldeep@kuldeep-VirtualBox:~/Documents/SEM 6/CS 352 - LAB/Assignment8$ g++ Q1_Code.cpp
kuldeep@kuldeep-VirtualBox:~/Documents/SEM 6/CS 352 - LAB/Assignment8$ ./Q1
```

INSIDE VIEW (LIGHTS-OFF): FAN, SOFA, TV, TABLE



```cpp
// KULDEEP SINGH
// 190001030

#include <bits/stdc++.h>
#include <GL/glut.h>
#include "imageio.h"

using namespace std;

typedef GLfloat point[3];
point f[] = { { 0.0, 0.25, 0.0 }, { 0.0, 1.0, 0.0 }, { 0.025, 0.25, -0.025 },
point fb[] = { { 0.025, 0.25, -0.025 }, { 0.025, 0.25, 0.025 }, { 0.2, 0.25,
point t[] = { { 1.1, -0.3, -0.5 }, { 1.1, 0.3, -0.5 }, { 1.1, 0.3, 0.5 }, { 1
point ts[] = { { 1.1, 0.05, -0.05 }, { 1.1, 0.0, -0.05 }, { 1.1, 0.0, 0.05 },

/*
To run the program:

1. To create imageio.o file: g++ -c imageio.cpp -ltiff -lpng
2. TO RUN: g++ Q1_Code.cpp -o Q1 -lglut -lGLU -lGL imageio.o -ltiff -lpng

Using keyboard:

r/R         Reset the house to initial position
z/x         Increase/Decrease diffuse intensity
```

```
kuldeep@kuldeep-VirtualBox:~/Documents/SEM 6/CS 352 - LAB/Assignment8$ ./Q1
```



```
        T
    }
    start_x =
    start_y =
}

void processS
{
    switch (k
    {
    case 'z':
        inten
        break
    case 'y':
        inten
        break
    }
    switch (k
    {
    // case '
    //     le
    //     br
    case 'w':
```

```
25
0 0.0199069 0

25
```

========================== x ================ x ==========================