CS352 Assignment-8

Krishanu Saini
190001029

Write a program using existing libraries to:

1. Draw a 3D house.
2. Provide the facility to view the house from inside using a mouse.
3. Enable the gate and windows to be opened and closed.
4. Draw some furniture inside the house.

Program

We will use GlutMouseFunc to find out angles to rotate and use mainloop to rotate given the angle.
We will use glTexImage2D for texture
We will open windows using keyboard functions

Code

```
#include <stdlib.h>
#include <math.h>
#include "imageio.h"

#include <bits/stdc++.h>
using namespace std;

#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

/*
* Name: Krishanu Saini
* Assn: 8
* Ques: Build interior of house
* Step: run ./Q3
* Uses: arrow keys to move
*       r for door
*       d for windows
*/

// angle of rotation for the camera direction
float angle = 0.0f;
int y_rot = 0.0, y_door = 0.0;
// actual vector representing the camera's direction
float lx = 0.0f, lz = -1.0f;
float lx_delta = 0.0f;
```

```
// XZ position of the camera
float x = 0.0f, z = 5.0f;

GLfloat intensity = 0;
GLubyte *textureImage;
vector<GLfloat> tx(3);
GLuint texture[2];

// the key states. These variables will be zero
// when no key is being presses
float deltaAngle = 0.0f;
float deltaMove = 0;
int xOrigin = -1;

// Positions of lights
GLfloat front_left_light_position[] = {-1.0, 0.0, 1.0, 0.0};
GLfloat front_right_light_position[] = {1.0, 0.0, 1.0, 0.0};
GLfloat back_left_light_position[] = {-1.0, 0.0, -1.0, 0.0};
GLfloat back_right_light_position[] = {1.0, 0.0, -1.0, 0.0};

void changeSize(int w, int h)
{

   // Prevent a divide by zero, when window is too short
   // (you cant make a window of zero width).
   if (h == 0)
       h = 1;

   float ratio = w * 1.0 / h;

   // Use the Projection Matrix
   glMatrixMode(GL_PROJECTION);

   // Reset Matrix
   glLoadIdentity();

   // Set the viewport to be the entire window
   glViewport(0, 0, w, h);

   // Set the correct perspective.
```

```
    gluPerspective(45.0f, ratio, 0.1f, 100.0f);

    // Get Back to the Modelview
    glMatrixMode(GL_MODELVIEW);
}

void drawSnowMan()
{

    glColor3f(1.0f, 1.0f, 1.0f);

    // Draw Body
    glTranslatef(0.0f, 0.75f, 0.0f);
    glutSolidSphere(0.75f, 20, 20);

    // Draw Head
    glTranslatef(0.0f, 1.0f, 0.0f);
    glutSolidSphere(0.25f, 20, 20);

    // Draw Eyes
    glPushMatrix();
    glColor3f(0.0f, 0.0f, 0.0f);
    glTranslatef(0.05f, 0.10f, 0.18f);
    glutSolidSphere(0.05f, 10, 10);
    glTranslatef(-0.1f, 0.0f, 0.0f);
    glutSolidSphere(0.05f, 10, 10);
    glPopMatrix();

    // Draw Nose
    glColor3f(1.0f, 0.5f, 0.5f);
    glRotatef(0.0f, 1.0f, 0.0f, 0.0f);
    glutSolidCone(0.08f, 0.5f, 10, 2);
}

void computePos(float deltaMove)
{

    x += deltaMove * lx * 0.1f;
    z += deltaMove * lz * 0.1f;
}
```

```
void init_texture(void)
{
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glEnable(GL_DEPTH_TEST);
   // The following two lines enable semi transparent
   // glEnable(GL_BLEND);
   // glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

   int width, height;
   int width1, height1;
   bool hasAlpha;
   char filename[] = "cupboard.png";
   char filename1[] = "d1.png";
   // bool success = loadPngImage(filename, width, height, hasAlpha,
&textureImage);
   unsigned char *ibuffer = loadImageRGBA(filename, &width, &height);
   std::cout << "Image loaded " << width << " " << height << " alpha " <<
hasAlpha << std::endl;
   std::cout << "Image loaded " << width1 << " " << height1 << " alpha "
<< hasAlpha << std::endl;
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
   glGenTextures(2, texture);
   cout << texture[0] << " " << texture[1] << endl;
   glBindTexture(GL_TEXTURE_2D, texture[0]);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, ibuffer);

   ibuffer = loadImageRGBA(filename1, &width, &height);
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
   glBindTexture(GL_TEXTURE_2D, texture[1]);
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, ibuffer);

   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

```
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glEnable(GL_TEXTURE_2D);
    glShadeModel(GL_FLAT);
}

void FaceTexture(GLfloat A[], GLfloat B[], GLfloat C[], GLfloat D[], int
text = 1)
{
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture[text]);
    glBegin(GL_POLYGON);
    glTexCoord2f(1.0, 1.0);
    glVertex3fv(A);
    glTexCoord2f(0.0, 1.0);
    glVertex3fv(B);
    glTexCoord2f(0.0, 0.0);
    glVertex3fv(C);
    glTexCoord2f(1.0, 0.0);
    glVertex3fv(D);
    glEnd();
}
/*------------------ house functions --------------------*/

GLfloat rec1[32][3] = {
    {-10, 7, 9},
    {10, 7, 9},
    {10, 0, 9},
    {-10, 0, 9},
    {-10, 7, -9},
    {10, 7, -9},
    {10, 0, -9},
    {-10, 0, -9},
};

GLfloat tri1[32][3] = {
    {0, 3, 10},
    {0, 3, 10},
    {12, 0, 10},
    {-12, 0, 10},
```

```
    {-0, 3, -10},
    {0, 3, -10},
    {12, 0, -10},
    {-12, 0, -10},
};

GLfloat win1[32][3] = {
    {1, 1, 0},
    {4, 1, 0},
    {4, -2, 0},
    {1, -2, 0},
};

GLfloat win2[32][3] = {
    {1, 1, 0},
    {4, 1, 0},
    {4, -2, 0},
    {1, -2, 0},
};

GLfloat door[32][3] = {
    {-2, 2, 0},
    {2, 2, 0},
    {2, -4, 0},
    {-2, -4, 0},
};

GLfloat handle[32][3] = {
    {-0.1, -2.1, 0},
    {0.1, -2.1, 0},
    {0.1, -2, 0},
    {-0.1, -2, 0},
};

GLfloat photo1[32][3] = {
    {2.5, 2, 0},
    {1, 1, 0},
    {4, 1, 0},
    {4, -1, 0},
    {1, -1, 0},
```

```
};

GLfloat photo2[32][3] = {
    {0, 2, 0},
    {0, 1, 1},
    {0, 1, -1},
    {0, -1, -1},
    {0, -1, 1},
};

GLfloat cupboard[32][3] = {
    {-2, 4, 1},
    {2, 4, 1},
    {2, 0, 1},
    {-2, 0, 1},
    {-2, 4, -1},
    {2, 4, -1},
    {2, 0, -1},
    {-2, 0, -1},
};

GLfloat bed[32][3] = {
    {0, 1, 2},
    {6, 1, 2},
    {6, 0, 2},
    {0, 0, 2},
    {0, 1, -2},
    {6, 1, -2},
    {6, 0, -2},
    {0, 0, -2},
};

GLfloat bedpost[32][3] = {
    {0, 3, 2},
    {0.4, 3, 2},
    {0.4, 1, 2},
    {0, 1, 2},
    {0, 3, -2},
    {0.4, 3, -2},
    {0.4, 1, -2},
```

```
    {0, 1, -2},
};

void shapeTranslate(GLfloat V[32][3])
{
    for (int i = 0; i < 32; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            V[i][j] += tx[j];
        }
    }
}

void MyInit()
{

    // shapes and translate
    tx = {0, 0, 0};
    shapeTranslate(rec1);
    tx = {0, 7.2, 0};
    shapeTranslate(tri1);
    tx = {-8, 4, 9.02};
    shapeTranslate(win1);
    tx = {3, 4, 9.02};
    shapeTranslate(win2);
    tx = {0, 4, 9.02};
    shapeTranslate(door);
    tx = {-0.5, 5, 9.03};
    shapeTranslate(handle);
    // furniture
    tx = {2, 2, -8.97};
    shapeTranslate(photo1);
    tx = {-9.7, 2, 5};
    shapeTranslate(photo2);
    tx = {-0.5, 0, -8.97};
    shapeTranslate(cupboard);
    tx = {-8, 0, 0};
    shapeTranslate(bed);
    shapeTranslate(bedpost);
```

```
    // items for
    glClearColor(0, 0, 0, 1);
    glEnable(GL_DEPTH_TEST);
}

void Face(GLfloat A[], GLfloat B[], GLfloat C[], GLfloat D[])
{
    glBegin(GL_POLYGON);
    glVertex3fv(A);
    glVertex3fv(B);
    glVertex3fv(C);
    glVertex3fv(D);
    glEnd();
}

void Cube(GLfloat V0[], GLfloat V1[], GLfloat V2[], GLfloat V3[], GLfloat
V4[], GLfloat V5[], GLfloat V6[], GLfloat V7[])
{
    glColor3f(1, 0, 0);
    /*
    {-10, 7, 9},
    {10, 7, 9},
    {10, 0, 9},
    {-10, 0, 9},
    */
    GLfloat cg[32][3] = {
        {-10, 7, 9},
        {10, 7, 9},
        {10, 5, 9},
        {-10, 5, 9},
        {10, 0, 9},
        {-10, 0, 9},
        {-7, 0, 9},
        {-7, 7, 9},
        {7, 0, 9},
        {7, 7, 9},
        {7, 0, 9},
        {7, 3, 9},
        {4, 3, 9},
```

```
        {4, 0, 9},
        {2, 0, 9},
        {2, 7, 9},
        {4, 7, 9},
        {4, 0, 9},
        {-2, 0, 9},
        {-2, 7, 9},
        {-4, 7, 9},
        {-4, 0, 9},
        {-7, 0, 9},
        {-7, 3, 9},
        {-4, 3, 9},
        {-4, 0, 9},
    };
    Face(cg[0], cg[1], cg[2], cg[3]); // Front
    Face(cg[0], cg[5], cg[6], cg[7]); // Front
    Face(cg[1], cg[4], cg[8], cg[9]); // Front
    Face(cg[10], cg[11], cg[12], cg[13]); // Front
    Face(cg[14], cg[15], cg[16], cg[17]); // Front
    Face(cg[18], cg[19], cg[20], cg[21]); // Front
    Face(cg[22], cg[23], cg[24], cg[25]); // Front

    glColor3f(0, 1, 0);
    Face(V4, V5, V6, V7); // Back
    glColor3f(0, 0, 1);
    Face(V0, V4, V7, V3); // Left
    glColor3f(1, 1, 0);
    Face(V1, V5, V6, V2); // Right
    glColor3f(1, 0, 1);
    Face(V2, V3, V7, V6); // Bot
    glColor3f(0, 1, 1);
    Face(V0, V1, V5, V4); // Top
}

void Triangle(GLfloat V0[], GLfloat V1[], GLfloat V2[], GLfloat V3[],
GLfloat V4[], GLfloat V5[], GLfloat V6[], GLfloat V7[])
{
    glColor3f(0.5, 0, 0);
    Face(V0, V1, V2, V3); // Front
    glColor3f(0, 0.5, 0);
```

```
    Face(V4, V5, V6, V7); // Back
    glColor3f(0, 0, 0.5);
    Face(V0, V4, V7, V3); // Left
    glColor3f(0.5, 0.5, 0);
    Face(V1, V5, V6, V2); // Right
    glColor3f(0.5, 0, 0.5);
    Face(V2, V3, V7, V6); // Bot
    glColor3f(0.5, 1, 1);
    Face(V0, V1, V5, V4); // Top
}

void Windows(GLfloat V0[], GLfloat V1[], GLfloat V2[], GLfloat V3[])
{
    glColor3f(0.5, 0.7, 0.7);
    Face(V0, V1, V2, V3); // Front
}

void Door(GLfloat V0[], GLfloat V1[], GLfloat V2[], GLfloat V3[])
{
    glColor3f(1, 0.7, 0.7);
    Face(V0, V1, V2, V3); // Front
}

float crgb[3] = {0.5, 0.5, 0.5};

void Rope(GLfloat V0[], GLfloat V1[])
{
    glColor3f(crgb[0], crgb[1], crgb[2]);
    glLineWidth(5);
    glBegin(GL_LINE_LOOP);
    glVertex3fv(V0);
    glVertex3fv(V1);
    glLineWidth(2);
    glEnd();
}

void Texture_Box(GLfloat V0[], GLfloat V1[], GLfloat V2[], GLfloat V3[])
{
    FaceTexture(V0, V1, V2, V3); // Front
}
```

```
void copyMatrix(GLfloat V[32][3], GLfloat rV[32][3])
{
    for (int i = 0; i < 32; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            V[i][j] = rV[i][j];
        }
    }
}

void Cupboard1(GLfloat V0[], GLfloat V1[], GLfloat V2[], GLfloat V3[],
GLfloat V4[], GLfloat V5[], GLfloat V6[], GLfloat V7[])
{
    FaceTexture(V0, V1, V2, V3, 0); // Front
    Face(V4, V5, V6, V7);           // Back
    Face(V0, V4, V7, V3);           // Left
    Face(V1, V5, V6, V2);           // Right
    Face(V2, V3, V7, V6);           // Bot
    Face(V0, V1, V5, V4);           // Top
}

void Bed1(GLfloat V0[], GLfloat V1[], GLfloat V2[], GLfloat V3[], GLfloat
V4[], GLfloat V5[],
          GLfloat V6[], GLfloat V7[])
{
    Face(V0, V1, V2, V3); // Front
    Face(V4, V5, V6, V7); // Back
    Face(V0, V4, V7, V3); // Left
    Face(V1, V5, V6, V2); // Right
    Face(V2, V3, V7, V6); // Bot
    Face(V0, V1, V5, V4); // Top
}

void Draw()
{
    Cube(rec1[0], rec1[1], rec1[2], rec1[3], rec1[4], rec1[5], rec1[6],
rec1[7]);
```

```
    Triangle(tri1[0], tri1[1], tri1[2], tri1[3], tri1[4], tri1[5], tri1[6],
tri1[7]);
    glPushMatrix();
    glRotatef(y_door, 0.0, 1.0, 0.0);
    Windows(win1[0], win1[1], win1[2], win1[3]);
    Windows(win2[0], win2[1], win2[2], win2[3]);
    glPopMatrix();

    // open door
    glPushMatrix();
    glRotatef(y_rot, 0.0, 1.0, 0.0);
    Door(door[0], door[1], door[2], door[3]);
    glPopMatrix();

    Windows(handle[0], handle[1], handle[2], handle[3]);
    // photo1
    Rope(photo1[0], photo1[1]);
    Rope(photo1[0], photo1[2]);
    Texture_Box(photo1[1], photo1[2], photo1[3], photo1[4]);

    Rope(photo2[0], photo2[1]);
    Rope(photo2[0], photo2[2]);
    Texture_Box(photo2[1], photo2[2], photo2[3], photo2[4]);

    glColor3f(0.4, 0.28, 0.25);
    Cupboard1(cupboard[0], cupboard[1], cupboard[2], cupboard[3],
cupboard[4], cupboard[5], cupboard[6], cupboard[7]);
    glColor3f(0.4, 0.25, 0.20);
    Bed1(bed[0], bed[1], bed[2], bed[3], bed[4], bed[5], bed[6], bed[7]);
    glColor3f(0.5, 0.4, 0.8);
    Bed1(bedpost[0], bedpost[1], bedpost[2], bedpost[3], bedpost[4],
bedpost[5], bedpost[6], bedpost[7]);

    // chandelier
    glColor3f(1, 0, 0);
    glPushMatrix();
    GLUquadricObj *qobj = gluNewQuadric();
    glTranslated(-5.0, 4, -5.0);
    glRotatef(90, 1.0f, 0.0f, 0.0f);
    gluCylinder(qobj, 0.2, 0.5, 1, 16, 16);
```

```
    glColor3f(0.5, 0.5, 0);
    gluCylinder(qobj, 0.05, 0.05, 4, 16, 16);
    gluDeleteQuadric(qobj);
    glPopMatrix();
}

void renderScene(void)
{

    if (deltaMove)
        computePos(deltaMove);

    // Clear Color and Depth Buffers
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Reset transformations
    glLoadIdentity();
    // Set the camera
    lx += lx_delta;
    gluLookAt(x, 1.0f, z + 40,
              x + lx, 1.0f, z + lz + 40,
              0.0f, 1.0f, 0.0f);

    // Draw ground

    glColor3f(0.9f, 0.9f, 0.9f);
    glBegin(GL_QUADS);
    glVertex3f(-100.0f, -0.1f, -100.0f);
    glVertex3f(-100.0f, -0.1f, 100.0f);
    glVertex3f(100.0f, -0.1f, 100.0f);
    glVertex3f(100.0f, -0.1f, -100.0f);
    glEnd();

    // draw house
    Draw();

    glutSwapBuffers();
}

void processNormalKeys(unsigned char key, int xx, int yy)
```

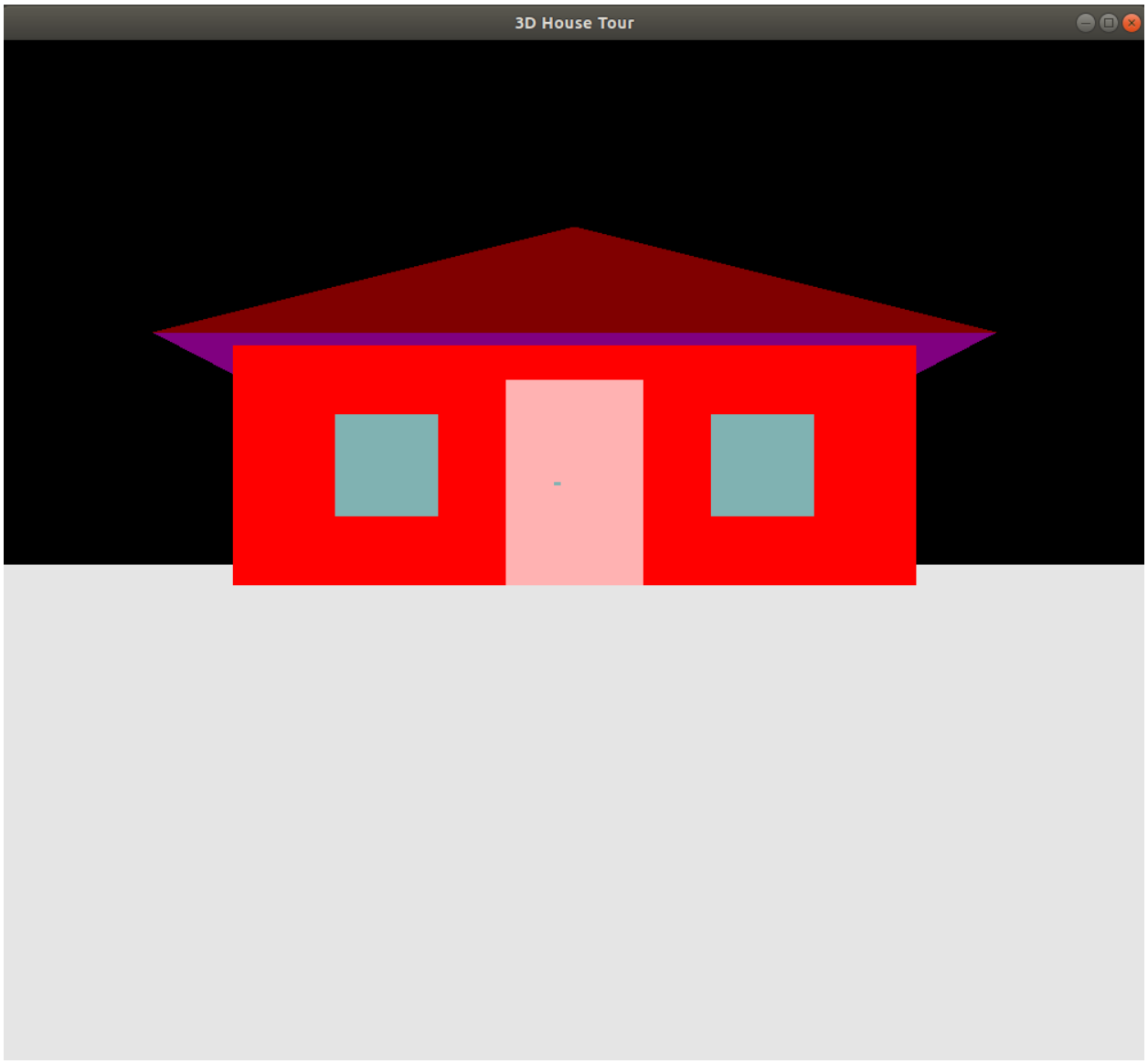
```
{

    if (key == 27)
        exit(0);
}

void pressKey(int key, int xx, int yy)
{

    switch (key)
    {
    case GLUT_KEY_UP:
        deltaMove = 0.1f;
        break;
    case GLUT_KEY_DOWN:
        deltaMove = -0.1f;
        break;
    case GLUT_KEY_LEFT:
        lx_delta = -0.0005;
        break;
    case GLUT_KEY_RIGHT:
        lx_delta = 0.0005;
        break;
    }
}

void releaseKey(int key, int x, int y)
{
    switch (key)
    {
    case GLUT_KEY_UP:
    case GLUT_KEY_DOWN:
        deltaMove = 0;
        break;
    case GLUT_KEY_LEFT:
    case GLUT_KEY_RIGHT:
        lx_delta = 0;
        break;
    }
}
```

```
void mouseMove(int x, int y)
{

    // this will only be true when the left button is down
    if (xOrigin >= 0)
    {

        // update deltaAngle
        deltaAngle = (x - xOrigin) * 0.001f;

        // update camera's direction
        lx = sin(angle + deltaAngle);
        lz = -cos(angle + deltaAngle);
    }
}

void mouseButton(int button, int state, int x, int y)
{

    // only start motion if the left button is pressed
    if (button == GLUT_LEFT_BUTTON)
    {

        // when the button is released
        if (state == GLUT_UP)
        {
            angle += deltaAngle;
            xOrigin = -1;
        }
        else
        { // state = GLUT_DOWN
            xOrigin = x;
        }
    }
}

void SpecialKeys(unsigned char key, int x, int y)
{
    if (key == 'r') // Open Gate
```

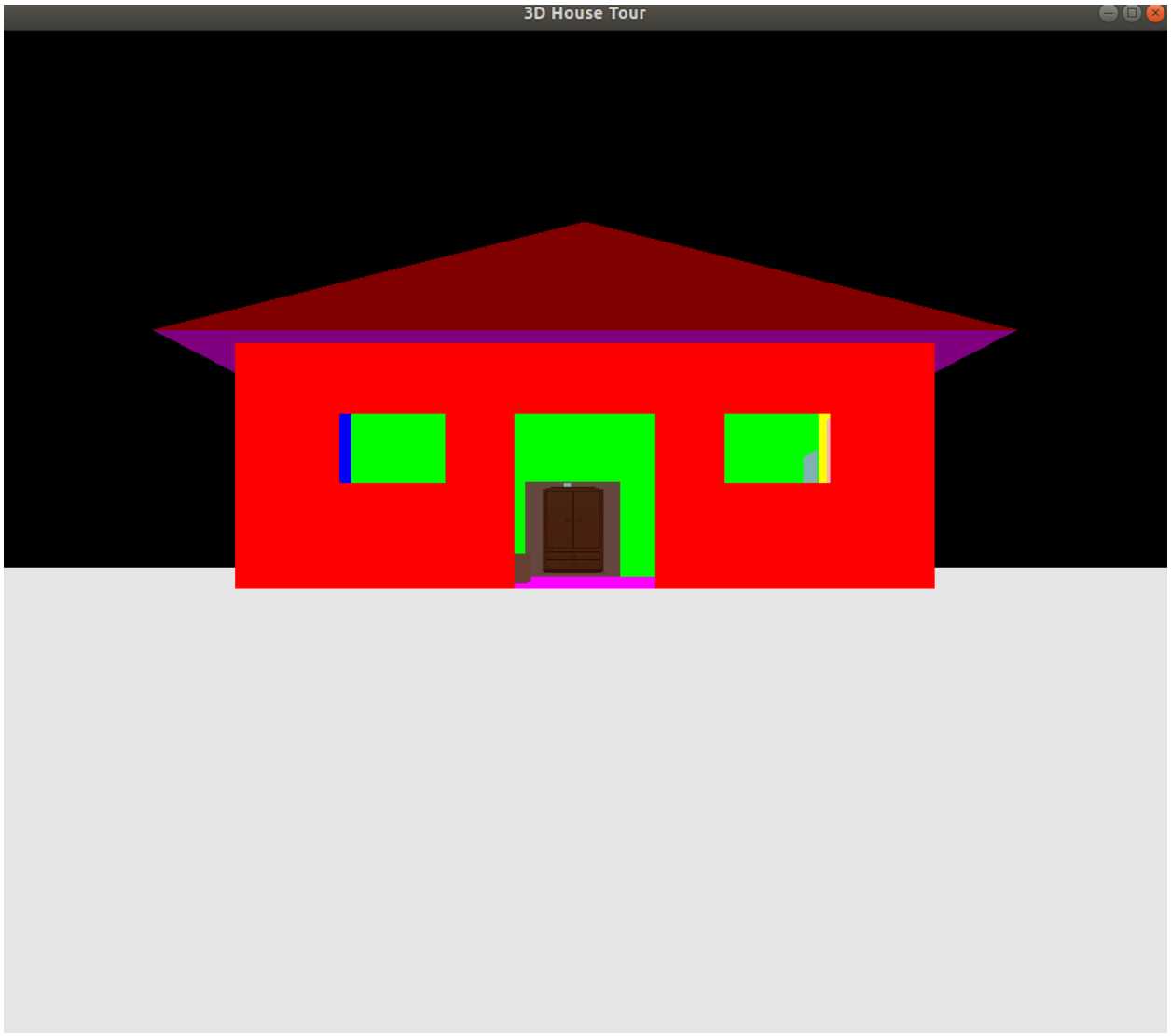
```
    {
        if (y_rot == 90)
        {
            y_rot = 0;
        }
        else
        {
            y_rot = 90;
        }
    }
    if (key == 'd') // Open Door
    {
        if (y_door == 90)
        {
            y_door = 0;
        }
        else
        {
            y_door = 90;
        }
    }
    glutPostRedisplay();
}

int main(int argc, char **argv)
{

    // init GLUT and create window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(1000, 1000); // set window size
    glutCreateWindow("3D House Tour");
    // register callbacks
    MyInit();
    init_texture();

    glutDisplayFunc(renderScene);
    glutReshapeFunc(changeSize);
    glutIdleFunc(renderScene);
```

```
    glutIgnoreKeyRepeat(1);
    glutKeyboardFunc(processNormalKeys);
    glutSpecialFunc(pressKey);
    glutSpecialUpFunc(releaseKey);

    // here are the two new functions
    glutKeyboardFunc(SpecialKeys);
    glutMouseFunc(mouseButton);
    glutMotionFunc(mouseMove);

    // OpenGL init
    glEnable(GL_DEPTH_TEST);

    // enter GLUT event processing cycle
    glutMainLoop();

    return 1;
}
```

Output



Open window and doors



Interior