Group QuantumCat: "Living Room"

https://github.com/Ken19149/3D-Engine

main.cpp

```
#include <GL/glut.h>
#include <iostream>
#include <vector>
#include <string>
#include <cmath>

// LIBRARIES

#define TINYOBJLOADER_IMPLEMENTATION
#include "tiny_obj_loader.h"

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"

// ==========================================
// 1. STRUCTURES
// ==========================================

struct Model {
    std::string name;
    std::vector<float> vertices;    // x, y, z
    std::vector<float> normals;     // nx, ny, nz
    std::vector<float> texcoords;   // u, v
    GLuint textureID = 0;           // OpenGL Texture ID
    bool loaded = false;
};
```

```
struct Object {
    std::string name;
    // Transform
    float x = 0, y = 0, z = 0;
    float rx = 0, ry = 0, rz = 0;
    float sx = 1, sy = 1, sz = 1;

    // Animation
    bool spinAnimation = false;
    float spinSpeed = 0.0f;

    Model* model = nullptr;
};

// ==========================================
// 2. GLOBALS
// ==========================================
std::vector<Object*> sceneObjects;
std::vector<Model*> loadedModels;

Object* selectedObject = nullptr;
int selectionIndex = 0;

// Camera (Orbit)
float cameraAngle = 0.0f;
float cameraHeight = 5.0f;
float cameraDist = 15.0f;

bool isClockAnimating = true;
bool isRoomSpinning = false; // NEW: Controls the 360 view
```

```
// ============================================
// 3. TEXTURE LOADING
// ============================================
GLuint LoadTextureFromFile(const char* filename) {
    // UPDATED: User requested path "models/textures/"
    std::string fullPath = "models/textures/" + std::string(filename);

    int width, height, nrChannels;
    stbi_set_flip_vertically_on_load(true);
    unsigned char *data = stbi_load(fullPath.c_str(), &width, &height, &nrChannels, 0);

    if (!data) {
        std::cout << "Failed to load texture: " << fullPath << " (using white fallback)" <<
std::endl;
        return 0;
    }

    GLuint textureID;
    glGenTextures(1, &textureID);
    glBindTexture(GL_TEXTURE_2D, textureID);

    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);

    int format = (nrChannels == 4) ? GL_RGBA : GL_RGB;
    glTexImage2D(GL_TEXTURE_2D, 0, format, width, height, 0, format,
GL_UNSIGNED_BYTE, data);

    stbi_image_free(data);
    return textureID;
}

// ==========================================
// 4. MODEL LOADING
// ==========================================
Model* GetModel(std::string filename) {
    for (auto* m : loadedModels) {
        if (m->name == filename) return m;
    }

    std::cout << "Loading Model: " << filename << "... ";
    Model* m = new Model();
    m->name = filename;

    tinyobj::attrib_t attrib;
    std::vector<tinyobj::shape_t> shapes;
    std::vector<tinyobj::material_t> materials;
    std::string warn, err;

    // Load OBJ from "models/" folder
    std::string fullPath = "models/" + filename;
    bool ret = tinyobj::LoadObj(&attrib, &shapes, &materials, &warn, &err, fullPath.c_str(),
"models/");

    if (!warn.empty()) std::cout << "WARN: " << warn << std::endl;
    if (!ret) {
        std::cout << "FAILED! " << err << std::endl;
        delete m;
        return nullptr;
    }
    }

    // Load Texture from MTL if available
    if (!materials.empty() && !materials[0].diffuse_texname.empty()) {
        std::string rawName = materials[0].diffuse_texname;
        size_t lastSlash = rawName.find_last_of("/\\");
        std::string fileName = (lastSlash == std::string::npos) ? rawName :
rawName.substr(lastSlash + 1);

        std::cout << "[Texture: " << fileName << "] ";
        m->textureID = LoadTextureFromFile(fileName.c_str());
    }

    for (const auto& shape : shapes) {
        for (const auto& index : shape.mesh.indices) {
            m->vertices.push_back(attrib.vertices[3 * index.vertex_index + 0]);
            m->vertices.push_back(attrib.vertices[3 * index.vertex_index + 1]);
            m->vertices.push_back(attrib.vertices[3 * index.vertex_index + 2]);

            if (index.normal_index >= 0) {
                m->normals.push_back(attrib.normals[3 * index.normal_index + 0]);
                m->normals.push_back(attrib.normals[3 * index.normal_index + 1]);
                m->normals.push_back(attrib.normals[3 * index.normal_index + 2]);
            }

            if (index.texcoord_index >= 0) {
                m->texcoords.push_back(attrib.texcoords[2 * index.texcoord_index + 0]);
                m->texcoords.push_back(attrib.texcoords[2 * index.texcoord_index + 1]);
            }
        }
    }
    m->loaded = true;
    std::cout << "Done. (" << m->vertices.size()/3 << " tris)" << std::endl;
    loadedModels.push_back(m);
    return m;
}

// ============================================
// 5. SCENE SETUP
// ============================================
void AddObj(std::string name, std::string modelName,
            float x, float y, float z,
            float rx, float ry, float rz,
            float sx, float sy, float sz,
            bool isAnimated = false)
{
    Object* obj = new Object();
    obj->name = name;
    obj->model = GetModel(modelName);

    obj->x = x; obj->y = y; obj->z = z;
    obj->rx = rx; obj->ry = ry; obj->rz = rz;
    obj->sx = sx; obj->sy = sy; obj->sz = sz;

    if (isAnimated) {
        obj->spinAnimation = true;
        obj->spinSpeed = 1.0f;
    }

    sceneObjects.push_back(obj);
}
```

```
void LoadScene() {
    // UPDATED DATA: Position kept, Rotation set to 0, Scale set to 1
    AddObj("big_sofa",    "big_sofa.obj",    -1.854, 0.030, 0.198,     0.0, 0.0, 0.0,       1.0,
1.0, 1.0,      false);
    AddObj("bookshelf",   "bookshelf.obj",   -2.053, -1.771, 0.030,    0.0, 0.0, 0.0,       1.0,
1.0, 1.0,      false);
    AddObj("cactus",      "cactus.obj",      -0.155, -0.131, 0.503,    0.0, 0.0, 0.0,       1.0,
1.0, 1.0,      false);
    AddObj("carpet",      "carpet.obj",      -0.039, 0.244, 0.046,     0.0, 0.0, 0.0,       1.0,
1.0, 1.0,      false);
    AddObj("clock",       "clock.obj",       -2.262, -1.811, 2.082,    0.0, 0.0, 0.0,       1.0, 1.0,
1.0,      true);
    AddObj("lamp",        "lamp.obj",        -1.829, 1.863, 0.088,     0.0, 0.0, 0.0,       1.0,
1.0, 1.0,      false);
    AddObj("shelf",       "shelf.obj",       -2.181, 0.072, 1.499,     0.0, 0.0, 0.0,       1.0, 1.0,
1.0,      false);
    AddObj("sofa",        "sofa.obj",        -0.077, 1.839, 0.336,     0.0, 0.0, 0.0,       1.0, 1.0,
1.0,      false);
    AddObj("table",       "table.obj",       -0.285, -0.104, 0.048,    0.0, 0.0, 0.0,       1.0, 1.0,
1.0,      false);
    AddObj("tv",          "tv.obj",          2.026, 0.132, 0.720,      0.0, 0.0, 0.0,       1.0, 1.0,
1.0,      false);
    AddObj("walls",       "walls.obj",       -0.178, 2.213, 1.590,     0.0, 0.0, 0.0,       1.0, 1.0,
1.0,      false);

    if(!sceneObjects.empty()) {
        selectedObject = sceneObjects[0];
    }
}

// ============================================
// 6. GLUT FUNCTIONS
// ============================================
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    // ORBIT CAMERA
    float camX = cameraDist * sin(cameraAngle);
    float camY = cameraDist * cos(cameraAngle);
    gluLookAt(camX, camY, cameraHeight,  0, 0, 0,  0, 0, 1);

    // DYNAMIC LIGHTS
    for (Object* obj : sceneObjects) {
        if (obj->name == "tv") {
            GLfloat blueColor[] = { 0.2f, 0.2f, 1.0f, 1.0f };
            GLfloat lightPos[]  = { obj->x, obj->y, obj->z + 0.5f, 1.0f };
            glLightfv(GL_LIGHT1, GL_DIFFUSE, blueColor);
            glLightfv(GL_LIGHT1, GL_POSITION, lightPos);
            glLightf(GL_LIGHT1, GL_CONSTANT_ATTENUATION, 1.0f);
            glLightf(GL_LIGHT1, GL_LINEAR_ATTENUATION, 0.2f);
            glLightf(GL_LIGHT1, GL_QUADRATIC_ATTENUATION, 0.05f);
        }

        if (obj->name == "lamp") {
            GLfloat orangeColor[] = { 1.0f, 0.7f, 0.2f, 1.0f };
            GLfloat lightPos[]    = { obj->x, obj->y, obj->z + 1.5f, 1.0f };
            glLightfv(GL_LIGHT2, GL_DIFFUSE, orangeColor);
            glLightfv(GL_LIGHT2, GL_POSITION, lightPos);
            glLightf(GL_LIGHT2, GL_CONSTANT_ATTENUATION, 1.0f);
            glLightf(GL_LIGHT2, GL_LINEAR_ATTENUATION, 0.1f);
            glLightf(GL_LIGHT2, GL_QUADRATIC_ATTENUATION, 0.02f);
        }
    }

    glEnable(GL_LIGHTING);

    // Draw Objects
    for (Object* obj : sceneObjects) {
        if (!obj->model || !obj->model->loaded) continue;

        glPushMatrix();

        // Transforms
        glTranslatef(obj->x, obj->y, obj->z);
        glRotatef(obj->rx, 1, 0, 0);
        glRotatef(obj->ry, 0, 1, 0);
        glRotatef(obj->rz, 0, 0, 1);
        glScalef(obj->sx, obj->sy, obj->sz);

        // Selection Highlight
        if (obj == selectedObject) {
            float pulse = (sin(glutGet(GLUT_ELAPSED_TIME) * 0.005f) + 1.0f) * 0.2f + 0.8f;
            glColor3f(pulse, pulse, 0.5f);
        } else {
            glColor3f(1, 1, 1);
        }

        if (obj->model->textureID != 0) {
            glEnable(GL_TEXTURE_2D);
            glBindTexture(GL_TEXTURE_2D, obj->model->textureID);
        } else {
            glDisable(GL_TEXTURE_2D);
        }

        glBegin(GL_TRIANGLES);
        int numVerts = obj->model->vertices.size() / 3;
        for (int i = 0; i < numVerts; i++) {
            if (!obj->model->normals.empty())
                glNormal3f(obj->model->normals[3*i+0], obj->model->normals[3*i+1],
obj->model->normals[3*i+2]);
            if (!obj->model->texcoords.empty())
                glTexCoord2f(obj->model->texcoords[2*i+0], obj->model->texcoords[2*i+1]);
            glVertex3f(obj->model->vertices[3*i+0], obj->model->vertices[3*i+1],
obj->model->vertices[3*i+2]);
        }
        glEnd();

        glPopMatrix();
    }

    glutSwapBuffers();
}

void keyboard(unsigned char key, int x, int y) {
    if (!selectedObject) return;
    float speed = 0.2f;
    float rSpeed = 5.0f;

    switch(key) {
        case 27: exit(0); break; // ESC
        case 9: // TAB
            selectionIndex = (selectionIndex + 1) % sceneObjects.size();
            selectedObject = sceneObjects[selectionIndex];
            std::cout << "Selected: " << selectedObject->name << std::endl;
            break;

        case ' ': isClockAnimating = !isClockAnimating; break; // Space: Pause Clock

        // ENTER KEY (13): Toggle 360 Degree Room View
        case 13:
            isRoomSpinning = !isRoomSpinning;
            std::cout << "360 Spin: " << (isRoomSpinning ? "ON" : "OFF") << std::endl;
            break;

        // Position
        case 'q': selectedObject->x += speed; break;
        case 'a': selectedObject->x -= speed; break;
        case 'w': selectedObject->y += speed; break;
        case 's': selectedObject->y -= speed; break;
        case 'e': selectedObject->z += speed; break;
        case 'd': selectedObject->z -= speed; break;

        // Rotation
        case 'r': selectedObject->rx += rSpeed; break;
        case 'f': selectedObject->rx -= rSpeed; break;
        case 't': selectedObject->ry += rSpeed; break;
        case 'g': selectedObject->ry -= rSpeed; break;
        case 'y': selectedObject->rz += rSpeed; break;
        case 'h': selectedObject->rz -= rSpeed; break;

        // Scale
        case 'u': selectedObject->sx += 0.05; selectedObject->sy += 0.05;
selectedObject->sz += 0.05; break;
```

```
        case 'j': selectedObject->sx -= 0.05; selectedObject->sy -= 0.05;
selectedObject->sz -= 0.05; break;
    }
    glutPostRedisplay();
}

void specialKeys(int key, int x, int y) {
    switch(key) {
        case GLUT_KEY_LEFT:  cameraAngle -= 0.1f; break;
        case GLUT_KEY_RIGHT: cameraAngle += 0.1f; break;
        case GLUT_KEY_UP:    cameraDist -= 0.5f; break;
        case GLUT_KEY_DOWN:  cameraDist += 0.5f; break;
    }
    glutPostRedisplay();
}

void idle() {
    // 1. Clock Animation (Updated: Rotate -X)
    if (isClockAnimating) {
        for (Object* obj : sceneObjects) {
            if (obj->spinAnimation) {
                obj->rx -= obj->spinSpeed;
            }
        }
    }

    // 2. Room 360 Spin Animation (New!)
    if (isRoomSpinning) {
        cameraAngle += 0.005f; // Adjust this number to change spin speed
    }
    glutPostRedisplay();
}

void init() {
    glEnable(GL_DEPTH_TEST);

    // LIGHTING SETUP
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0); // White general
    glEnable(GL_LIGHT1); // TV Blue
    glEnable(GL_LIGHT2); // Lamp Orange

    glEnable(GL_COLOR_MATERIAL);
    glDisable(GL_CULL_FACE);
    glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);

    GLfloat light_pos[] = { 5.0, 5.0, 10.0, 1.0 };
    glLightfv(GL_LIGHT0, GL_POSITION, light_pos);

    glClearColor(0.1f, 0.1f, 0.15f, 1.0f);
}

void reshape(int w, int h) {
    if (h == 0) h = 1;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (float)w / h, 0.1, 100.0);
    glMatrixMode(GL_MODELVIEW);
}
```

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(1024, 768);
    glutCreateWindow("Final Room Project");

    init();
    LoadScene();

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutSpecialFunc(specialKeys);
    glutIdleFunc(idle);

    std::cout << "CONTROLS:\nArrows: Manual Camera\nENTER: Toggle 360
View\nTAB: Select Object\nWASD/QE: Move Object\nRF/TG/YH: Rotate
Object\nSpace: Pause Clock\n";

    glutMainLoop();
    return 0;
}
```

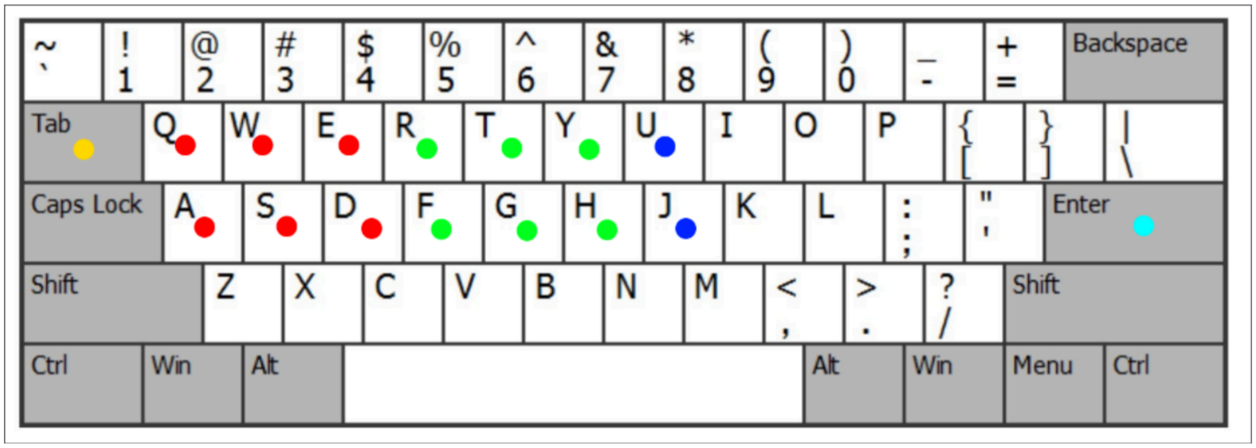
Requirements:

- stb_image.h
- Tiny_obj_loader.h

Compile command:

```
g++ main.cpp -o main -I./dependencies/include -lglut -lGL -lGLU -ldl
```

Instruction:

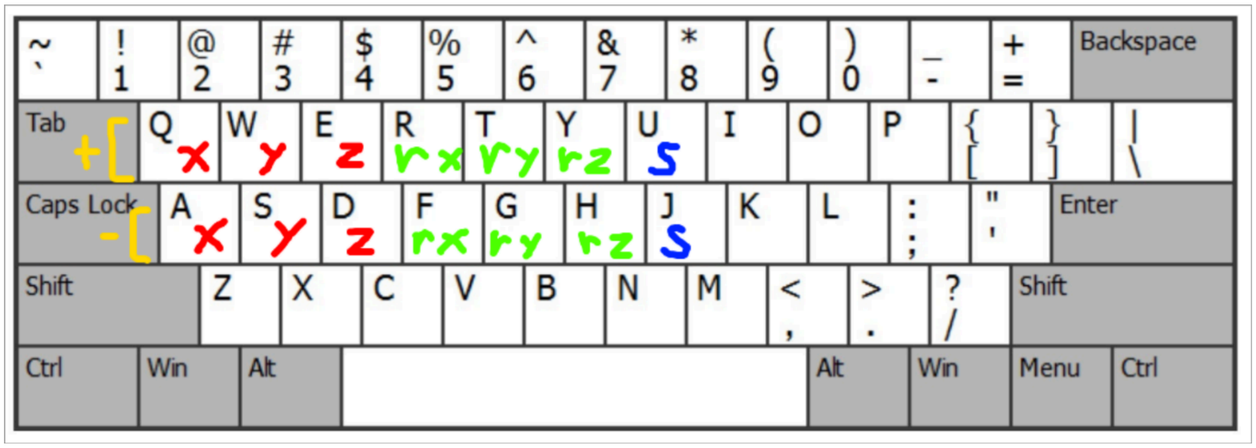


🟥 Translate 🟩 Rotate 🟦 Scale

🟨 Select Next Object 🟦 Toggle Animation

Transformation:

- Upper Row = Increase Value
- Lower Row = Decrease Value

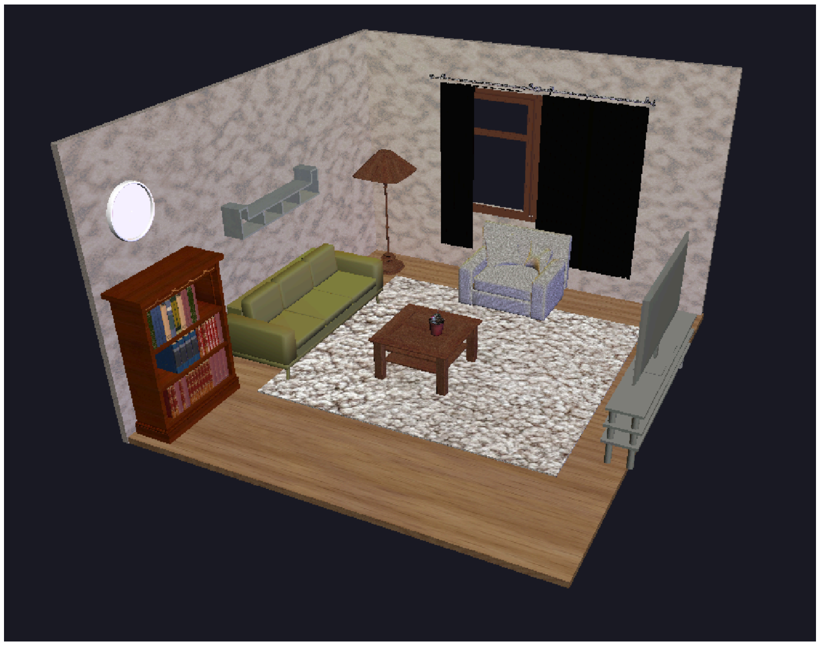


Initial Thoughts

At first we're thinking of going big with structures like models and scenes folders. Then let the main code read the scene setting from scene.json and be able to manipulate, add and remove objects, as well as saving it as a new scene with object having grouping functionality. Furthermore, we also intended to implement multiple modes of control for manipulating objects and controlling camera movement.

The progress went well for a while as we're able to import the models that the code read from scene.json, but the problem was texture missing and parenting made rotation confusing. It was clear that the project of that scale is too big for this timeframe considering other responsibilities we have to take care of as well. So, we later decided to opt for the simpler solution: no grouping or parenting, and use only simple models instead of multipart models like clocks with hands.

In the end, after a few iterations of trials and errors, we finally reached a rather satisfactory outcome and below are the overall process of the final program.



Workflow

The elements of the scene were found on the internet and then imported into blender. Their scales, rotations and positions were then adjusted to suitable values before textures were added. After that the objects were exported into Wavefront file format or .obj and .mtl files. Using TinyObjLoader, we were able to load the files and import the models into the C++ code.

The coordinates of each element of the scene reference the coordinates of the elements in the blender file.

Loading the scene focuses on the overall objects in the scene. It is controlled by the Loadscene() function, which defines the properties of the object instance (position, rotation, and scale). While Loadscene() is running, it triggers TinyObjLoader to open the .obj text file, converts lines of text into floating-point numbers, and then stores information into the std::vector<float> vertices arrays. Once LoadScene() finishes, all models will be loaded and the sceneObjects vector will be full. Finally, the display() function will loop through the vector by reading x, y, and z of the Object Instance, transform it using glTranslatef, follow the pointer to the obj->model and then draw the vertices stored in that model.

However, loading the 3d objects focuses on the physical data of the model. It is the process of translating the .obj file into arrays of numbers inside the GetModel() function, so inside this function it will call tinyobj::LoadObj opens the text file, reads the data, and separates lines starting with "v" (vertices) and "f" (faces), then connects the dots in the correct order to form the shape.

Each object instance includes position x, y, z, when AddObj() is running, it creates the specific instance of the object in the scene and assigns these coordinates. The Positioning 3d model is performed inside the display() loop. Every time the loop starts for a new object, the OpenGL coordinate system resets to the center and the model will be in the center of the room (0,0,0). The command glTranslatef will move this cursor and shift the drawing position to the center of where the object should be.

If an object is named “tv”, it configures GL_LIGHT1 as blue, positional light located slightly in front of the TV, with attenuation so the light fades over distance. If an object is named “lamp”, it configures GL_LIGHT2 as an orange light positioned inside of the lamp, also with distance-based attenuation. This allows the light to move automatically with the objects they belong to, while the command glEnable(GL_LIGHTING) turns on OpenGL’s fixed-function lighting system so these lights affect the rendered scene.

Loading textures starts with the GetModel() function finding which image to load using TinyObjLoader. When TinyObjLoader reads .obj file, it also looks for the linked .mtl file to find the filename of image (texture) to be used. Once triggered, LoadTextureFromFile function performs loading. It uses the “stb_image” library to read the .jpg or .png from your file, convert it into raw pixel data (red, green, blue), and upload it to the graphic card. Finally, LoadTextureFromFile returns a Texture ID number to tell OpenGL exactly which image to use during drawing.