

Crearea unei scene folosind Blender si OpenGL

Student: Mihnea Tîrcă



Contents

1	Prezentarea temei	2
2	Scenariul	2
2.1	Descrierea scenei si a obiectelor	2
2.1.1	Casele	2
2.1.2	Terenul	2
2.1.3	Bancile	2
2.1.4	Fantana	2
2.1.5	Lampile	4
2.1.6	Strada	4
2.1.7	Trotuarele	4
2.1.8	Cosurile de gunoi	4
2.1.9	Copacii	4
2.2	Functionalitati	4
3	Detalii de implementare	6
3.1	Functii si algoritmi	6
3.1.1	Lumina	6
3.1.2	Umbre	6
3.1.3	Ceata	7
3.1.4	Mapare de texturi de opacitate	7
3.1.5	Camera	8
3.1.6	Animatia obloanelor	8
3.1.7	Animatie de prezentare	9
3.1.8	Borderless fullscreen	11
3.1.9	Solid, Wireframe, Punctiforma	11
3.2	Structuri de date	11
4	Manual de utilizare	12
5	Concluzii si dezvoltari ulterioare	12
6	Referinte	13



Figure 1: Scena

1 Prezentarea temei

In acest proiect am creat o scena grafica in Blender si, folosind librariile OpenGL GLM, GLFW si limbajul GLSL, am adaugat efecte in scena pentru a creste realismul scenei.

2 Scenariul

2.1 Descrierea scenei si a obiectelor

Scena este un cartier al unui oras, in stil "centru vechi", intr-o dimineata de primavara. In scena sunt incluse case, banci, cosuri de gunoi, lampa, trotuare, o strada si copaci¹. Efectele implementate includ lumina ambientala, difuza, speculara, harti de umbre, ceata, animatii, skybox, mapare de texturi: ambientale, difuze, speculare si de opacitate.

Majoritatea obiectelor au venit si cu texturi de mapare *normal*, *roughness*, *metallic* etc., insa, desi le-am aplicat in Blender, nu le-am aplicat pe toate in OpenGL, din cauza faptului ca ar fi trebuit sa modific mult clasa Model3D, ceea ce am si facut un pic pentru a reusi sa mapez opacitatea. Texturile pe care le-am aplicat in OpenGL (dupa caz) sunt ambientale, difuze, speculare si de opacitate.

2.1.1 Casele

Casele au mapate texturi difuze si speculare. Deoarece scena este incadrata de case si utilizatorul nu ar trebui sa vada vreodata spatele lor, am tata varfurile din spatele caselor pentru performanta. Insa, din cauza faptului ca soarele bate din spatele anumitor case, a trebuit sa dezactivez *back face culling*: figurile 2 si 3.

2.1.2 Terenul

Terenul este un plan simplu cu o textura difusa aplicata pe el.

2.1.3 Bancile

Bancile au mapate o textura difusa.

2.1.4 Fantana

Fantana⁴ are mapata o textura difusa pe ea. Am adaugat si apa prin maparea unei texturi de apa pe cele 4 planuri. I-am combinat textura fantanii cu textura apei pe care am adaugat-o eu, pentru a crea o singura textura.

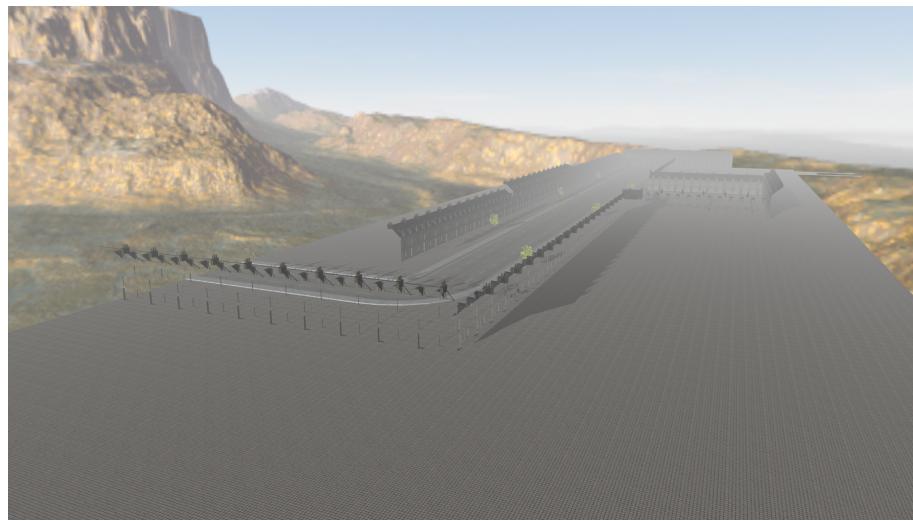


Figure 2: Umbrele cu culling pornit

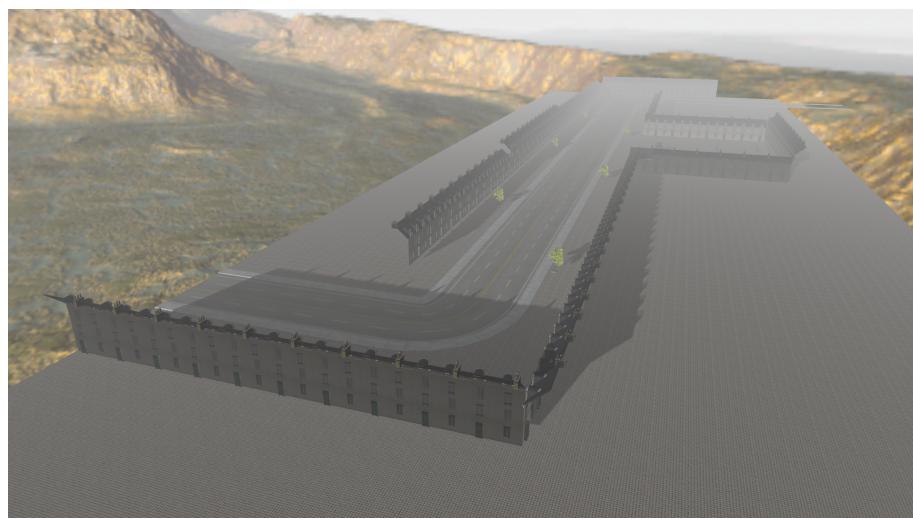


Figure 3: Umbrele cu culling pornit



Figure 4: Fantana



Figure 5: Lampa

2.1.5 Lampile

Lampile⁵ au mapate textura difusa si de opacitate, adica fetele lampii sunt transparente.

2.1.6 Strada

Strada are mapata o textura difusa.

2.1.7 Trotuarele

Trotuarele au mapate o textura difusa.

2.1.8 Cosurile de gunoi

Cosurile de gunoi au mapate o textura difusa.

2.1.9 Copacii

Copacii⁶ au mapate texturi difuze si de opacitate (pentru frunze⁷). Am creat de asemenea un obiect pentru baza copacului si am mapat niste texturi difuze pe ele. Baza copacului am facut-o prin maparea unei texturi de frunzis pe un plan si inconjurarea lui cu un cub cu fetele de sus si jos scoase si scalat pentru turtirea lui. Pe acesta am mapat o textura de piatra.

2.2 Functionalitati

Functionalitatatile implementate sunt:

- Miscarea camerei cu tastatura
- Miscarea camerei cu mouse-ul
- Pornirea unei animatii de prezentare
- Schimbarea intensitatii cetii
- Animarea deschiderii unor obloane de la o casa
- Vederile wireframe, solid, punctiforma
- Vederea hartii de adancime



Figure 6: Un copac



Figure 7: Frunzele

3 Detalii de implementare

3.1 Functii si algoritmi

In acest capitol voi vorbi doar despre algoritmii implementati de mine, nu si de cei pre-existenti in template-ul de la care am pornit.

3.1.1 Lumina

Am implementat lumina directionalala (de la soare) prin miscarea camerei spre soare, destul de departata de scena, si luarea pozitii camerei. Acolo am definit ca fiind sursa luminii. Am miscat camera sa se uite inspre centrul aproximativ al scenei (fantana) si asa am luat target-ul luminii. Am calculat directia luminii prin scaderea celor doua valori. Le-am definit pe toate ca fiind constante pentru evitarea modificarii lor accidentale (pozitia soarelui este statica):

```
const glm::vec3 lightPos = glm::vec3(25.5321f, 80.821f, -233.757f);
const glm::vec3 lightTarget = glm::vec3(-35.0f, 11.0f, 32.0f);
const glm::vec3 lightDir = lightPos - lightTarget;
```

Aceasta este trimisa ca variabila uniform in fragment shader si prelucrata.

3.1.2 Umbre

Umbrele sunt implementate folosind tehnica hartilor de umbra. Pentru calcularea matricei de transformare in spatiul luminii, am folosit functia `glm::lookAt` cuplata cu `glm::ortho` (deoarece lumina este directionalala):

```
glm::mat4 computeLightSpaceTrMatrix() {
    glm::mat4 lightView = glm::lookAt(lightPos, lightTarget, glm::vec3(0.0f,
    1.0f, 0.0f));
    const GLfloat near_plane = 0.01f, far_plane = 700.0f;
    glm::mat4 lightProjection = glm::ortho(-70.0f, 70.0f, -70.0f, 70.0f,
    near_plane, far_plane);
    glm::mat4 lightSpaceTrMatrix = lightProjection * lightView;
    return lightSpaceTrMatrix;
}
```

Aceasta este transmisa ca variabila uniform in vertex shader si prelucrata.

De asemenea, am imbunatatit algoritmul de calculare a umbrelor cu tehnica PCF ([percentage-closer filtering](#)):

```
float computeShadow(){
    ...
    // percentage-closer filtering
    float shadow = 0.0;
    vec2 texelSize = 1.0 / textureSize(shadowMap, 0);
    for(int x = -1; x <= 1; ++x)
    {
        for(int y = -1; y <= 1; ++y)
        {
            float pcfDepth = texture(shadowMap, normalizedCoords.xy +
    vec2(x, y) * texelSize).r;
            shadow += currentDepth - bias > pcfDepth ? 1.0 : 0.0;
        }
    }
    shadow /= 9.0;
    ...
    return shadow;
}
```

3.1.3 Ceata

Ceata este implementata ca in laborator, insa am implementat schimbarea valorii densitatii cetii prin user input, trimitand densitatea cetii ca variabila uniform in fragment shader:

```
// fog
float fogDensity = 0.005f;
void keyboardCallback(GLFWwindow* window, int key, int scanCode, int action,
int mode) {
    ...
    // -density
    if (key == GLFW_KEY_3) {
        if (fogDensity >= 0.001f) {
            fogDensity -= 0.001f;
        }
        myCustomShader.useShaderProgram();
        glUniform1f(glGetUniformLocation(myCustomShader.shaderProgram,
            "fogDensity"), fogDensity);
    }

    // +density
    if (key == GLFW_KEY_4) {
        fogDensity += 0.001f;
        myCustomShader.useShaderProgram();
        glUniform1f(glGetUniformLocation(myCustomShader.shaderProgram,
            "fogDensity"), fogDensity);
    }
    ...
}
```

3.1.4 Mapare de texturi de opacitate

Pe langa maparea de texturi ambientale, difuze si speculare, care erau deja implementate in clasa **Model3D**, am implementat si aplicarea texturilor de opacitate, anume pe frunzele copacilor si pe sticla lampilor. Pentru extragerea liniei de cod din fisierul .mtl care corespunde cu textura de opacitate, a trebuit sa adaug in **Model3D**:

```
...
//alpha mask
std::string alphaTexturePath = materials[materialId].alpha_texname;
if (!alphaTexturePath.empty())
{
    gps::Texture currentTexture;
    currentTexture = LoadTexture(basePath + alphaTexturePath, "alphaMask");
    textures.push_back(currentTexture);
}
...
```

Apoi, trimitem textura de opacitate in fragment shader. Textura de opacitate are un singur canal, adica are doar nuante de alb-negru. Logica este ca, valoarea de la alb la negru din textura de opacitate corespunde unei valori de transparenta. Aceasta valoare este inmultita cu valoarea alpha a culorii, iar daca culoarea este aproape neagra, elimin fragmentul. De asemenea, pentru a aplica textura de opacitate doar pentru modelele care o au, am transmis, printre-o variabila uniform *hasMask*, daca fragmentul respectiv are sau nu textura de opacitate. Pe aceasta o setez la valoarea '1' inainte sa desenez obiectele care au aceasta textura. Astfel, desenarea obiectelor cu textura de opacitate:

```

void drawObjects(gps::Shader shader, bool depthPass) {
    ...
    glUniform1i(glGetUniformLocation(myCustomShader.shaderProgram, "hasMask"), 1);
    leaves.Draw(shader);
    lamps.Draw(shader);
    glUniform1i(glGetUniformLocation(myCustomShader.shaderProgram, "hasMask"), 0);
    ...
}

```

Iar in fragment shader:

```

uniform sampler2D alphaMask;
uniform int hasMask = 0;
void main()
{
    ...
    if(hasMask == 1){
        vec4 mask = texture(alphaMask, fTexCoords);
        //fColor = vec4(fColor.rgb, fColor.a * mask.r);
        if(mask.r < 0.1)
            discard;
    }
}

```

Insa, acest cod imi da un efect neprevazut la lampi: daca ma uit prin sticla lampilor, anumite obiecte nu apar, asa ca am eliminat linia de cod in care inmultesc canelele 'a' si 'r', si ramane doar sa elimin fragmentele unde opacitatatea este sub 0.1.

3.1.5 Camera

Camera a fost implementata asemenea [tutorial-ului learnopengl](#). Important de mentionat este folosirea unei variabile *deltaTime* pentru a tine minte timpul intre frame-uri, astfel incat doua calculatoare cu viteze de procesare diferite sa nu aiba viteza camerei diferita.

3.1.6 Animatia obloanelor

Am animat obloanele de la o casa, astfel incat sa se inchida si sa se deschida la apasarea unei taste. A trebuit in Blender sa despart obloanele de casa de care facea parte, pentru a le anima separat. Pentru rotirea obloanelor in jurul unei axe, a trebuit sa translatez obloanele cu opusul coordonatelor axei, rotirea lor, si translatarea inapoi cu coordonatele axei. Pentru implementare, am nevoie de cateva variabile:

```

// object animation
bool shutters_direction = false, shutters_opened = true, shutters_closed = false;
float angle = 0;
bool animate = false;

```

La apasarea unei taste, se schimba directia animatiei obloanelor si variabilele care tin minte daca obloanele sunt deschise sau inchise devin ambele false.

```

if (key == GLFW_KEY_E && action == GLFW_PRESS) {
    shutters_opened = false;
    shutters_closed = false;
    shutters_direction = !shutters_direction;
}

```

Apoi, daca se schimba directia, se rotesc obloanele cu o viteza constanta (folosind din nou variabila *deltaTime*). In functia *drawObjects*:

```

if (!shutters_closed && !shutters_opened) {
    if (shutters_direction) {
        angle -= deltaTime * 100.0f;
    }
    else {
        angle += deltaTime * 100.0f;
    }
}

if (angle < -179.0f && shutters_direction) {
    shutters_closed = true;
    angle = -180.0f;
}

if (angle > -1.0f && !shutters_direction) {
    shutters_opened = true;
    angle = 0.0f;
}

model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(-37.8676f, 10.6024f, 58.9943f));
model = glm::rotate(model, glm::radians(-angle), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(37.8676f, -10.6024f, -58.9943f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
// do not send the normal matrix if we are rendering in the depth map
if (!depthPass) {
    normalMatrix = glm::mat3(glm::inverseTranspose(view * model));
    glUniformMatrix3fv(normalMatrixLoc, 1, GL_FALSE, glm::value_ptr(normalMatrix));
}
shutter1.Draw(shader);

model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(-38.4758f, 10.611f, 58.9943f));
model = glm::rotate(model, glm::radians(angle), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(38.4758f, -10.611f, -58.9943f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
// do not send the normal matrix if we are rendering in the depth map
if (!depthPass) {
    normalMatrix = glm::mat3(glm::inverseTranspose(view * model));
    glUniformMatrix3fv(normalMatrixLoc, 1, GL_FALSE, glm::value_ptr(normalMatrix));
}
shutter2.Draw(shader);

```

Unde traducerile sunt coordonatele axelor de rotatie.

Dupa aceasta implementare, am avut probleme cu faptul ca, la inchiderea obloanelor, spatele lor era neluminat. Mi-am dat seama ca fetele neiluminate aveau de fapt doar un "strat", si de aceea avea normalele intr-o singura directie, si am reusit sa rezolv problema.

3.1.7 Animatie de prezentare

Am implementat de asemenea o animatie de prezentare. Ea are mai multe stagii. Trecerea de la un stagiul altul se face cu ajutorul unor variabile. Tin minte de asemenea timpul la care a inceput prezentarea cu functia `glfwGetTime()`, care ma ajuta sa stiu cand sa trec de la un stagiul altul. Folosesc de asemenea pozitia camerei, cat si orientarea ei pentru a stii cand sa trec la alt stagiul (ex: merg in fata 5 secunde, merg in stanga pana ajung la o coordonata, rotesc camera pana ajung la o anumita valoare a campului 'Yaw' din *Camera*). De asemenea, dezactivez orice miscare a camerei pe parcursul prezentarii. Ea poate fi oprita oricand apasand aceeasi tasta.

Global, am declarat variabilele:

```
// presentation animation
bool stage[10];
double presentation_start;
bool animate = false;
```

In *keyboardCallback()*:

```
// animatie de prezentare
if (key == GLFW_KEY_P && action == GLFW_PRESS) {
    animate = !animate;
    if (animate) {
        // init animation
        glfwSetCursorPosCallback(glWindow, NULL);
        myCamera.initAnimation();
        for (int i = 1; i < 10; i++) {
            stage[i] = true;
        }
        presentation_start = glfwGetTime();
    }
    else {
        //end animation
        glfwSetCursorPosCallback(glWindow, mouse_callback);
        myCamera.endAnimation();
    }
}
```

Initializarea si terminarea prezentarii, in *Camera.cpp*:

```
void Camera::initAnimation() {
    Position = glm::vec3(-17.1456f, 11.9768f, 133.581f);
    Yaw = -90.0f;
    Pitch = 0.0f;
    MovementSpeed = 10.0f;
    updateCameraVectors();
}

void Camera::endAnimation() {
    MovementSpeed = SPEED;
    MouseSensitivity = SENSITIVITY;
}
```

Animatia, in *renderScene()*:

```
if (animate && glfwGetTime() > presentation_start + 0.01) {
    if (glfwGetTime() < presentation_start + 5 && stage[1]) {
        myCamera.ProcessKeyboard(MOVE_FORWARD, deltaTime);
    }
    else if (glfwGetTime() < presentation_start + 8 && stage[2]) {
        stage[1] = false;
        myCamera.ProcessKeyboard(MOVE_FORWARD, deltaTime);
        myCamera.ProcessMouseMovement(-1.0f, 0.0f);
    }
    else if (glfwGetTime() > presentation_start + 8 && glfwGetTime() <
    presentation_start + 12 && stage[3]) {
        stage[2] = false;
        myCamera.ProcessMouseMovement(-1.0f, 0.0f);
```

```

    }
    else if(myCamera.getYaw() < -90.0f && stage[4]) {
        stage[3] = false;
        myCamera.ProcessMouseMovement(1.0f, 0.0f);
    }
    else if (myCamera.getPosition().x > -54.0f && stage[5]) {
        stage[4] = false;
        myCamera.setMovementSpeed(5.0f);
        myCamera.ProcessKeyboard(MOVE_LEFT, deltaTime);
    }
    else if (myCamera.getYaw() < 0.0f && stage[6]) {
        stage[5] = false;
        myCamera.setMouseSensitivity(0.08f);
        myCamera.ProcessKeyboard(MOVE_FORWARD, deltaTime);
        myCamera.ProcessMouseMovement(1.0f, 0.0f);
    }
    else if (myCamera.getYaw() > -140.0f && stage[7]) {
        stage[6] = false;
        myCamera.ProcessKeyboard(MOVE_FORWARD, deltaTime);
        myCamera.ProcessMouseMovement(-1.5f, 0.0f);
    }
    else if (myCamera.getYaw() < -90.0f && stage[8]) {
        stage[7] = false;
        myCamera.ProcessKeyboard(MOVE_FORWARD, deltaTime);
        myCamera.ProcessMouseMovement(1.0f, 0.0f);
    }
}
}

```

3.1.8 Borderless fullscreen

Am implementat fereastra in modul *borderless fullscreen*, asemenea [documentatiei oficiale](#).

3.1.9 Solid, Wireframe, Punctiforma

Se poate trece intre vizualizarea solida, wireframe si punctiforma la apasarea tastelor:

```

// solid
if (key == GLFW_KEY_1 && action == GLFW_PRESS) {
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
}

// wireframe
if (key == GLFW_KEY_2 && action == GLFW_PRESS) {
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
}
// punctiform
if (key == GLFW_KEY_5 && action == GLFW_PRESS) {
    glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
}

```

3.2 Structuri de date

Structurile de date folosite sunt cele primitive, cat si cele din libraria GLM:

1. vectori: **glm::vecx**
2. matrice: **glm::matx**

3. primitive GLM: **GLint**, **GLuint**, **GLfloat** etc.

Mai folosesc structuri de date din libraria GLFW pentru fereastra:

1. **GLFWwindow**: fereastra
2. **GLFWmonitor**: monitorul, folosit pentru afisarea in *borderless fullscreen*
3. **GLFWvidmode**: modul video al monitorului, folosit pentru afisarea in *borderless fullscreen*

Cat si structuri de date precum clasele:

1. **Camera**: camera de vizionare - miscarea camerei, returnarea de matrice *view*
2. **Mesh**: mesh - date, incarcare de model etc.
3. **Model3D**: model - date, incarcare de model etc.
4. **Shader**: shader - date, incarcare de shader etc.
5. **SkyBox**: skybox - date, incarcare, afisare etc.

Si resurse open-source:

1. **tiny_obj_loader**: pentru incarcare de obiecte .obj
2. **stb_image**: pentru incarcare de texturi

4 Manual de utilizare

- Miscarea camerei cu tastatura:
 - fata: tasta 'W'
 - stanga: tasta 'A'
 - spate: tasta 'S'
 - dreapta: tasta 'D'
 - sus: tasta 'spacebar'
 - jos: tasta 'left control'
- Miscarea camerei cu mouse-ul
- Pornirea unei animatii de prezentare: tasta 'P'
- Schimbarea intensitatii cetii: tastele '3' si '4'
- Animarea deschiderii unor obloane de la o casa: tasta 'E'
- Vederea wireframe, solid si punctiforma: tastele '1', '2' si '5'
- Vederea hartii de adancime: tasta 'M'

5 Concluzii si dezvoltari ulterioare

In concluzie, am invatat nu doar cum sa lucrez in OpenGL, dar si in Blender. Desi nu am aplicat diverse texturi in OpenGL cum ar fi *normal*, *roughness*, *metallic* etc., am invatat ce inseamna si cum sa lucrez cu ele in Blender si am invatat sa implementez aplicarea texturii de opacitate in OpenGL. Imbunatatirile ulterioare ar fi, desigur, sa aplic si alte texturi si imbunatatirea umbrelor (reducerea efectului de *pixel panning*). As putea adauga lumini punctiforme la lampile adaugate in scena. As putea, de asemenea, sa imi termin implementatia de schimbare intre modurile ferestrei de *windowed* si *borderless fullscreen* si sa termin implementatia de utilizare a aplicatiei in *first-person*. As fi vrut de asemenea sa mai adaug o animatie cu o masina care urmeaza curba strazii.

6 Referinte

Modelele care nu au fost create de mine au fost luate de pe site-uri de modele 3D:

[Banci](#)

[Fantana](#) (fara apa)

[Case](#)

[Lampi](#)

[Cosuri de gunoi](#)

[Copaci](#) (fara baza)

Alte referinte: [percentage-closer filtering](#)

[Camera Ferestrel GLFW](#)