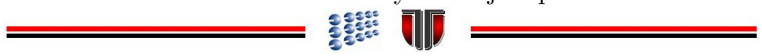
Department of Computer Science
Technical University of Cluj-Napoca



Sisteme de prelucrare grafica

Laboratory activity

Name: Ghiurca Dorin
Group: 30233
Email: dorin.ghiurca@gmail.com

Contents

Chapter 1

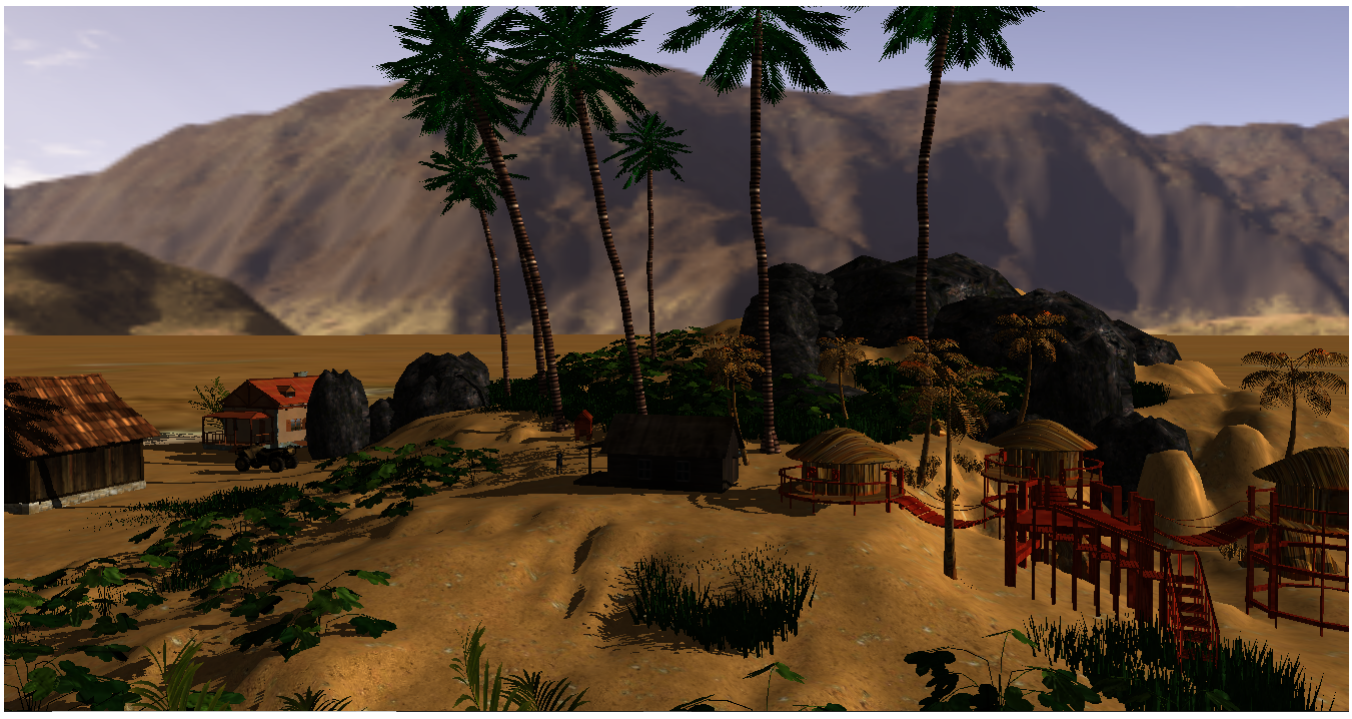
Prezentarea temei

Proiectul consta in realizarea unei prezentari fotorealiste a unor scene de obiecte 3D utilizand librariile prezentate in cadrul laboratorului (OpenGl, GLFW, GLM etc.).

Proiectul trebuie sa dispuna de mai multe functionalitati. In ceea ce priveste modul de vizualizare a scenei, prin intermediul utilizarii tastaturii si mouse-ului sau prin utilizarea animatiilor de prezentare trebuie sa se poata realiza scalarea, translatia, rotatia si miscarea camerei cu scopul de a vizualiza in intregime scena si de a surprinde diferite aspecte ale acesteia.

Scena trebuie sa contina cel putin 2 surse de lumina diferite, sa ofere posibilitatea vizualizarii in modurile solid, wireframe, poligonal si smooth. Un alt aspect se refera la maparea texturilor si definirea materialelor. Maparea trebuie sa se realize intr-un mod corespunzator iar nivelul de detaliu al acestora sa fie unul decent. Umbrele reprezinta o alta parte importanta, ele contribuind intr-o mare masura la fotorealismul scenei, astfel este necesar exemplificarea generarii lor.

Este necesara exemplificarea animarii diferitelor componente ale obiectelor. Asigurarea fotorealismului prin intermediul complexitatii scenei, nivelul de detaliere al modelarii, dezvoltarea diferitilor algoritmi si implementarea acestora (generare dinamic de obiecte, detectia coliziunilor, generarea umbrelor, ceata, ploaie, vant), calitatea animatiilor, utilizarea diferitelor surse de lumina (globala, locala, de tip spot).



Chapter 2

Scenariu

2.0.1 Descrierea scenei si a obiectelor

Scena are la baza o insula. Acest lucru presupune existenta unor forme de relief tipice precum mici formatiuni deluroase/stancoase, prezenta stancilor, a nisipului precum si diferite plante specifice.

In ceea ce priveste plantele, se poate observa prezenta ierbii, a diferitelor plante de dimensiuni mici precum si a palmierilor, specifici unei insule. Casele sunt alte obiecte ce intra in constitutia scenei, acestea presupunand exista unui numar mic de localnici.
Se mai poate observa totodata prezenta unui autovehicul de teren precum si a mediul inconjurator ce consta in munti, cer, nori etc. Acestea au fost realizate prin folosirea cubemap-ului.

2.0.2 Functionalitati

Pentru o mai buna vizualizare a obiectelor, se poate realiza deplasarea prin scena cu ajutorul tastaturii si a mouse-ului, se poate realiza scalarea si rotirea scenei.

Chapter 3

Detalii de implementare

3.0.1 Functii si algoritmi

Solutii posibile

`Functia de procesare a miscarii prin intermediul tastaturii si a mouse-ului:`

```
float speed = 3.0f; // 3 units / second
float mouseSpeed = 0.05f;
float horizontalAngle = 3.14f;
// vertical angle : 0, look at the horizon
float verticalAngle = 0.0f;
double deltaTime = 1.0f;

void mouseCallback(GLFWwindow* window, double xpos, double ypos)
{
        glfwSetCursorPos(window, glWindowWidth / 2.0f, glWindowHeight / 2.0f);
        horizontalAngle += mouseSpeed * deltaTime * float(glWindowWidth / 2 - xpos);
        verticalAngle += mouseSpeed * deltaTime * float(glWindowHeight / 2 - ypos);
        myCamera.rotate(glm::radians(horizontalAngle), glm::radians(verticalAngle));
}

void processMovement()
{

        if (pressedKeys[GLFW_KEY_Q]) {
                angle += 0.1f;
                if (angle > 360.0f)
                        angle -= 360.0f;
        }

        if (pressedKeys[GLFW_KEY_E]) {
                angle -= 0.1f;
                if (angle < 0.0f)
                        angle += 360.0f;
        }

        if (pressedKeys[GLFW_KEY_W]) {
                myCamera.move(gps::MOVE_FORWARD, cameraSpeed);
        }

        if (pressedKeys[GLFW_KEY_S]) {
                myCamera.move(gps::MOVE_BACKWARD, cameraSpeed);
        }

        if (pressedKeys[GLFW_KEY_A]) {
                myCamera.move(gps::MOVE_LEFT, cameraSpeed);
```

```
        }

        if (pressedKeys[GLFW_KEY_D]) {
                myCamera.move(gps::MOVE_RIGHT, cameraSpeed);
        }

        if (pressedKeys[GLFW_KEY_J]) {

                lightAngle += 0.3f;
                if (lightAngle > 360.0f)
                        lightAngle -= 360.0f;
                glm::vec3 lightDirTr = glm::vec3(glm::rotate(glm::mat4(1.0f),
                glm::radians(lightAngle), glm::vec3(0.0f, 1.0f, 0.0f)) *
                glm::vec4(lightDir, 1.0f));
                myCustomShader.useShaderProgram();
                glUniform3fv(lightDirLoc, 1, glm::value_ptr(lightDirTr));
        }

        if (pressedKeys[GLFW_KEY_L]) {
                lightAngle -= 0.3f;
                if (lightAngle < 0.0f)
                        lightAngle += 360.0f;
                glm::vec3 lightDirTr = glm::vec3(glm::rotate(glm::mat4(1.0f),
                glm::radians(lightAngle), glm::vec3(0.0f, 1.0f, 0.0f)) *
                glm::vec4(lightDir, 1.0f));
                myCustomShader.useShaderProgram();
                glUniform3fv(lightDirLoc, 1, glm::value_ptr(lightDirTr));
        }
}
```

Functia de initializare a FBOs:

```
void initFBOs()
{

        //generate FBO ID
        glGenFramebuffers(1, &shadowMapFBO);

        //create depth texture for FBO
        glGenTextures(1, &depthMapTexture);
        glBindTexture(GL_TEXTURE_2D, depthMapTexture);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT,
        SHADOW_WIDTH, SHADOW_HEIGHT, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

        //attach texture to FBO
        glBindFramebuffer(GL_FRAMEBUFFER, shadowMapFBO);
        glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
        GL_TEXTURE_2D, depthMapTexture, 0);
        glDrawBuffer(GL_NONE);
        glReadBuffer(GL_NONE);
        glBindFramebuffer(GL_FRAMEBUFFER, 0);
}
```

Functia de initializare a modelelor:

```
void initModels()
{
        //myModel = gps::Model3D("objects/nanosuit/nanosuit.obj",
        "objects/nanosuit/");
        myModel2 = gps::Model3D("objects/island/untitled.obj",
        "objects/island/");
        ground = gps::Model3D("objects/ground/ground.obj",
         "objects/ground/");
        lightCube = gps::Model3D("objects/cube/cube.obj",
        "objects/cube/");
}
```

Functia de initializare a shader-elor:

```
void initShaders()
{
        myCustomShader.loadShader("shaders/shaderStart.vert",
        "shaders/shaderStart.frag");
        lightShader.loadShader("shaders/lightCube.vert",
        "shaders/lightCube.frag");
        depthMapShader.loadShader("shaders/simpleDepthMap.vert",
        "shaders/simpleDepthMap.frag");
}
```

Functia de initializare a unform-urilor:

```
void initUniforms()
{

        myCustomShader.useShaderProgram();

        modelLoc = glGetUniformLocation(myCustomShader.shaderProgram,
        "model");

        viewLoc = glGetUniformLocation(myCustomShader.shaderProgram,
        "view");

        normalMatrixLoc = glGetUniformLocation(myCustomShader.shaderProgram,
        "normalMatrix");

        lightDirMatrixLoc = glGetUniformLocation(myCustomShader.shaderProgram,
        "lightDirMatrix");

        projection = glm::perspective(glm::radians(45.0f), (float)retina_width /
        (float)retina_height, 0.1f, 1000.0f);
        projectionLoc = glGetUniformLocation(myCustomShader.shaderProgram,
        "projection");
        glUniformMatrix4fv(projectionLoc, 1, GL_FALSE,
        glm::value_ptr(projection));

        //set the light direction (direction towards the light)
        lightDir = glm::vec3(0.0f, 1.0f, 2.0f);
        lightDirLoc = glGetUniformLocation(myCustomShader.shaderProgram,
        "lightDir");
        glUniform3fv(lightDirLoc, 1, glm::value_ptr(lightDir));

        //set light color
        lightColor = glm::vec3(1.0f, 1.0f, 1.0f); //white light
```

```
        lightColorLoc = glGetUniformLocation(myCustomShader.shaderProgram,
        "lightColor");
        glUniform3fv(lightColorLoc, 1, glm::value_ptr(lightColor));

        lightShader.useShaderProgram();
        glUniformMatrix4fv(glGetUniformLocation(lightShader.shaderProgram,
        "projection"), 1, GL_FALSE, glm::value_ptr(projection));
}
```

Functia de randare a scenei:

```
void renderScene()
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        processMovement();

        //render the scene to the depth buffer (first pass)

        depthMapShader.useShaderProgram();

        glUniformMatrix4fv(glGetUniformLocation(depthMapShader.shaderProgram,
        "lightSpaceTrMatrix"),
                1,
                GL_FALSE,
                glm::value_ptr(computeLightSpaceTrMatrix()));

        glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
        glBindFramebuffer(GL_FRAMEBUFFER, shadowMapFBO);
        glClear(GL_DEPTH_BUFFER_BIT);

        //create model matrix for nanosuit
        model = glm::rotate(glm::mat4(1.0f), glm::radians(angle),
        glm::vec3(0, 1, 0));
        //send model matrix to shader
        glUniformMatrix4fv(glGetUniformLocation(depthMapShader.shaderProgram,
        "model"),
                1,
                GL_FALSE,
                glm::value_ptr(model));

        //myModel.Draw(depthMapShader);
        myModel2.Draw(depthMapShader);
        //create model matrix for ground
        model = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, -1.0f, 0.0f));
        //send model matrix to shader
        glUniformMatrix4fv(glGetUniformLocation(depthMapShader.shaderProgram,
        "model"),
                1,
                GL_FALSE,
                glm::value_ptr(model));

        ground.Draw(depthMapShader);

        glBindFramebuffer(GL_FRAMEBUFFER, 0);

        //render the scene (second pass)

        myCustomShader.useShaderProgram();
```

```
//send lightSpace matrix to shader
glUniformMatrix4fv(glGetUniformLocation(myCustomShader.shaderProgram,
"lightSpaceTrMatrix"),
        1,
        GL_FALSE,
        glm::value_ptr(computeLightSpaceTrMatrix()));

//send view matrix to shader
view = myCamera.getViewMatrix();
glUniformMatrix4fv(glGetUniformLocation(myCustomShader.shaderProgram,
"view"),
        1,
        GL_FALSE,
        glm::value_ptr(view));

//compute light direction transformation matrix
lightDirMatrix = glm::mat3(glm::inverseTranspose(view));
//send lightDir matrix data to shader
glUniformMatrix3fv(lightDirMatrixLoc, 1, GL_FALSE,
glm::value_ptr(lightDirMatrix));

glViewport(0, 0, retina_width, retina_height);
myCustomShader.useShaderProgram();

//bind the depth map
glActiveTexture(GL_TEXTURE3);
glBindTexture(GL_TEXTURE_2D, depthMapTexture);
glUniform1i(glGetUniformLocation(myCustomShader.shaderProgram,
"shadowMap"), 3);

//create model matrix for nanosuit
model = glm::rotate(glm::mat4(1.0f), glm::radians(angle),
glm::vec3(0, 1, 0));
//send model matrix data to shader
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));

//compute normal matrix
normalMatrix = glm::mat3(glm::inverseTranspose(view*model));
//send normal matrix data to shader
glUniformMatrix3fv(normalMatrixLoc, 1, GL_FALSE, glm::value_ptr(normalMatrix));

//myModel.Draw(myCustomShader);
myModel2.Draw(myCustomShader);

//create model matrix for ground
model = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, -1.0f, 0.0f));
//send model matrix data to shader
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));

//create normal matrix
normalMatrix = glm::mat3(glm::inverseTranspose(view*model));
//send normal matrix data to shader
glUniformMatrix3fv(normalMatrixLoc, 1, GL_FALSE,
glm::value_ptr(normalMatrix));

//ground.Draw(myCustomShader);

//draw a white cube around the light

lightShader.useShaderProgram();
```

```
        glUniformMatrix4fv(glGetUniformLocation(lightShader.shaderProgram,
        "view"), 1, GL_FALSE, glm::value_ptr(view));

        model = glm::rotate(glm::mat4(1.0f), glm::radians(lightAngle),
        glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::translate(model, lightDir);
        model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
        glUniformMatrix4fv(glGetUniformLocation(lightShader.shaderProgram,
        "model"), 1, GL_FALSE, glm::value_ptr(model));

        mySkyBox.Draw(skyboxShader, view, projection);
        lightCube.Draw(lightShader);
}
```

Functia de realizare a cubemap-ului:

```
void initSkybox(){

        std::vector<const GLchar*> faces;
        faces.push_back("skybox/right.tga");
        faces.push_back("skybox/left.tga");
        faces.push_back("skybox/top.tga");
        faces.push_back("skybox/bottom.tga");
        faces.push_back("skybox/back.tga");
        faces.push_back("skybox/front.tga");

        mySkyBox.Load(faces);
        skyboxShader.loadShader("shaders/skyboxShader.vert",
        "shaders/skyboxShader.frag");
        skyboxShader.useShaderProgram();

        view = myCamera.getViewMatrix();
        glUniformMatrix4fv(glGetUniformLocation(skyboxShader.shaderProgram,
        "view"), 1, GL_FALSE, glm::value_ptr(view));

        projection = glm::perspective(glm::radians(45.0f), (float)retina_width /
        (float)retina_height, 0.1f, 1000.0f);
        glUniformMatrix4fv(glGetUniformLocation(skyboxShader.shaderProgram,
        "projection"), 1, GL_FALSE, glm::value_ptr(projection));
}
```

Motivarea abordarii alese

S-au folosit functiile de procesare a miscarii prin intermediul mouse-ului si a tastaturii pentru a satisface nevoia de deplasare prin scena. Functia de initializare a FBO's se foloseste pentru crearea FBO-urilor(Framebuffer objects) cu scopul redarii flexibile precum si rederea texturilor. In functia de initializare modele se ofera caile spre locatia unde se gasesc obiectele necesare implementarii scenei.

Prin intermediul functiei de procesare a shader-elor se realizeaza procesarea datelor in pipeline-ul grafic. Functia de initializare a uniform-urilor permite crearea unor uniform-uri necesare pentru transmiterea date din interiorul aplicatiei. Functia de randare a scene se ocupa cu proiectarea propriu-zisa a obiectelor in care se se realizeaza toate operatiile necesare precum translatii, rotatii, scalari precum si realizarea matricelor de transformari. Functia initSkybox ne permita sa oferim un anumit "fundal" scenei, contribuind astfel la fotorealismul obiectelor, simuland mediul inconjurator.

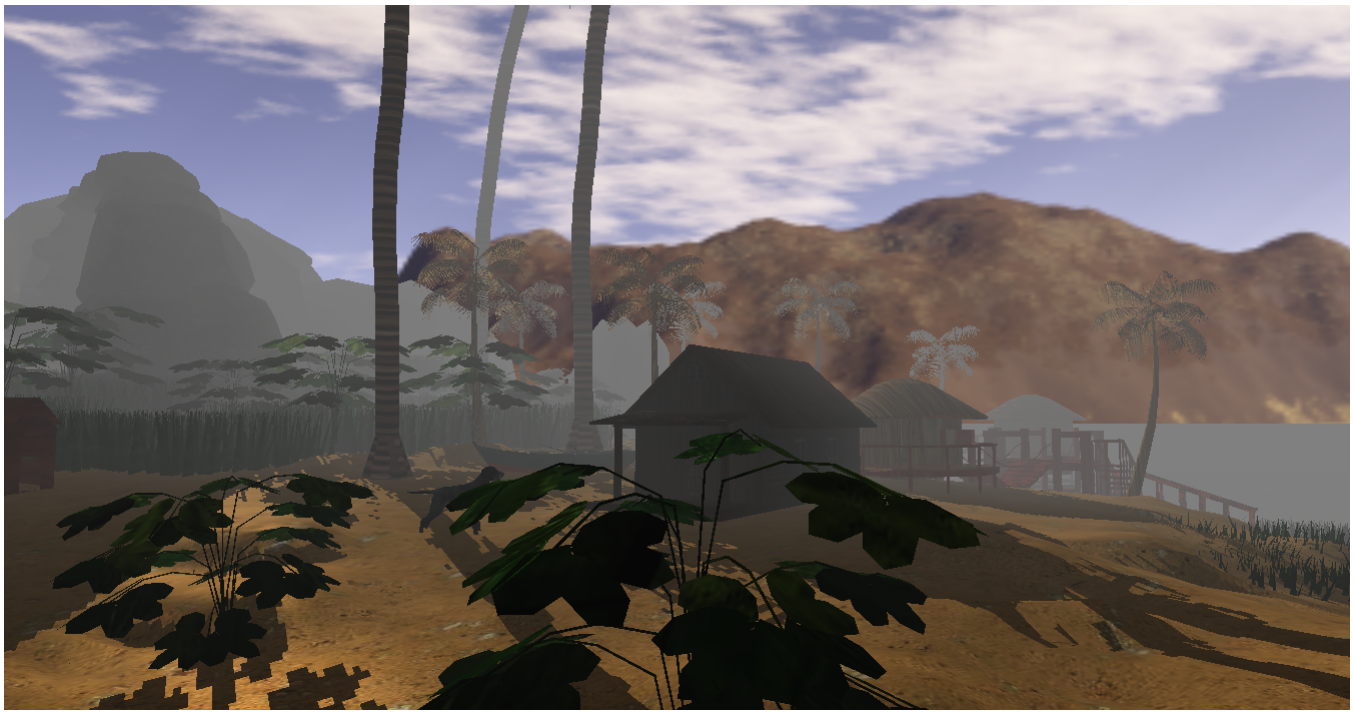
3.0.2 Structuri de date. Ierarhia de clase

S-au folosit structuri de date pentru reprezentarea vectorilor de 3 sau 4 elemente si a matricelor necesare pentru realizarea transformarilor.

Chapter 4

Prezentarea interfetei grafice utilizator. Manual de utilizare

Pentru a putea vizualiza scena, trebuie in primul rand rulata aplicatia. Acest lucru se realizeaza prin combinatia de taste CTRL + F5 in mediul de dezvoltare Visual Studio. Dupa realizarea acestui pas, se va afisa scena propriu-zisa cu toate obiectele. Pentru vizualizarea acestora precum si pentru miscarea prin interiorul scenei, se utilizeaza tastatura si mouseului. Pentru deplasarea inainte (in adancime) se utilizeaza tasta "W", pentru deplasare inapoi (departare de scena) se utilizeaza tasta "S", deplasarea stanga si dreapta este asigurata de tastele "A", respectiv "D". Pentru a vedea efectele umbrelor, se utilizeaza tastele "J" si "L" care schimba pozitia sursei de iluminare, astfel umbrele schimbadu-si si ele la randul lor pozitia.



Chapter 5

Concluzii si dezvoltari ulterioare

Posibilitatea programarii anumitor parti din pipeline-ul grafic a dus la o multitudine de progrese in ceea ce priveste proiectarea imaginilor 2D. In prezent, se realizeaza nenumarate proiecte ce tin de domeniul graficii pe calculator, contribuind astfel in multe domenii ale vietii noastre. Apropierea de realitate este imbunatatita pe zi ce trece, cu greu realizandu-se in prezent distinctia intre imaginile proiectate pe ecran si diferentele fata de realitate.

Proiectului de fata i se pot aduce o multitudine de imbunatatiri. Se pot implementa diferite surse de iluminare distincte, fiecare imprimand propriul efect asupra obiectelor. Se pot realiza totodata diferite animatii care ofera dinamism scenei.

Chapter 6

Bibliography

https://learnopengl.com/