PROIECT

Titlu: ***Sat părăsit la ocean***

Nume studentă: ***Ciochină Cătălina-Andreea***

Disciplina: ***Prelucrare Grafică***

Universitate: ***Universitatea Tehnică din Cluj-Napoca, Facultatea de Automatică și Calculatoare***

Număr grupa: ***30238***

Nume îndrumător de proiect: ***Gorgan Raul Alexandru***

Data: ***18.01.2024***

Cuprins

1. Prezentarea temei

Din dorința de a reproduce fidel aspectul unui sat părăsit la ocean, am construit o scenă detaliată care combină elemente de apă, terenuri variate și obiecte specifice pentru a transmite atmosfera corespunzătoare.

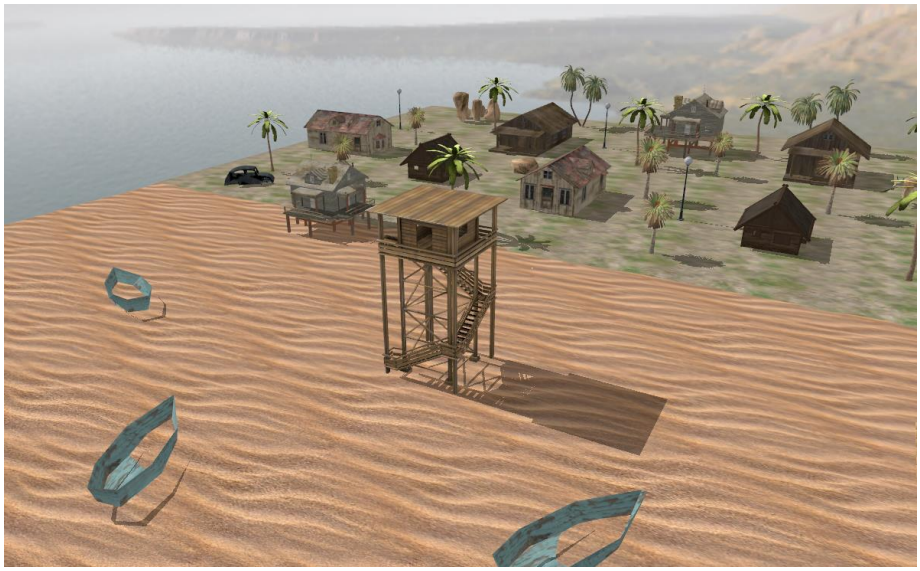
2. Scenariul

2.1. Descrierea scenei și a obiectelor

Tema proiectului a luat naștere în jurul ideii pe care am avut-o după construirea în Blender a terenului cu textură de apă, pe care l-am modelat cu ajutorul tool-urilor de modelare – Grab, Elastic Deform, Snake Hook, astfel încât să imite aspectul valurilor de suprafață ale unui ocean. Apoi am modelat un teren de deșert și am aplicat o textură de nisip, pe care am adăugat obiecte specifice unui sat părăsit (turn de supraveghere al salvamarilor, șezlonguri vechi și bărcuțe). Pe terenul cu iarbă am adăugat bungalow-uri părăsite cu aspect învechit, palmieri, animale sălbatice, roci, o mașină stricată și felinare ruginite, conturând astfel atmosfera unui sat părăsit la ocean/mare. Obiectele care se mișcă sunt vaporul și broasca care conferă un aspect realist aplicației. Scena a fost construită cu ajutorul unor modele 3D de tip .obj care au asociate fisiere .mtl pentru textură.

2.2. Funcționalități

Obiectele au fost aranjate prin intemediul diferitelor transformări de scalare, translație și rotație și pot fi vizualizate prin intemediul mișcărilor camerei pe cele 4 direcții cu ajutorul tastelor: înainte-W, inapoi-S, stânga-A și dreapta-D. De asemenea, mișcarea user-ului se poate realiza și prin transformări de rotație, tastele Q și E, dar și prin ajustarea perspectivei și a câmpului de vizualizare, navigând cu mouse-ul.



În cadrul aplicației OpenGl, am mai implementat funcționalități cheie pentru sporirea fotorealismului scenei, precum lumină direcțională, pe care am considerat-o venind din imaginea soarelui din skybox, unghiul poate fi schimbat cu ajutorul tastelor J si L și lumini poziționale, care sunt activate/dezactivate prin intermediul comutării între tastele N și Z. Luminile poziționale sunt amplasate în bulbii celor 4 felinare ale scenei.

Vizualizarea scenei se poate realiza în 4 moduri: solid (afișează obiectele cu suprafețe pline, fără a evidenția detaliile structurale interioare), wireframe (afișează muchiile, doar contururile obiectelor, fără a umple spațiul dintre ele), poligonal (afișează vârfurile poligoanelor din care este construit obiectul) și smooth (afișează obiectele cu o suprafață netedă, eliminând vizual poligoanele evidente).

Toate obiectele din scenă au fost texturate folosind fișiere de obiecte Wavefront .obj și fișierele de materiale .mtl corespunzătoare. Această metodă a permis asocierea detaliilor texturilor și proprietăților materialelor într-un mod eficient, contribuind astfel la realismul și complexitatea generală a scenei create.

De asemenea, am creat și hărți de adâncime, care au susținut creerea umbrelor din scenă, prezente numai în peisajul de zi. Skybox-ul de zi este preluat din resursele laboratorului, iar cel de noapte din resursele prezente pe moodle. Prin intermediul tastelor de F și G se realizează incrementarea, respectiv decrementarea factorului de ceață, conferind un aspect fotorealist scenei.

Ca animații ale scenei am utilizat transformări de cameră pentru animația de deschidere a aplicației și mișcări ale navei și broaștei țestoase, care se realizează prin transformări de model.

Am mai implementat un simplu algoritm de evitare a coliziunii user-ului cu apa, astfel încât nu îi este permis să intre în apă.

3. Detalii de implementare

3.1.Funcții și algoritmi

3.1.1. Soluții posibile

3.1.2. Motivarea abordării alese

- ***Lumina globală și luminile punctiforme***

Lumina globală am implementat-o cu ajutorul shaderelor, utilizându-mă de iluminarea OpenGL cu pipeline fix care însumează un set de componente independente pentru a obține efectul de iluminare general al unui punct de pe suprafața unui obiect. Din laboratorul 7: „lumina ambientală nu vine dintr-o anumită direcție, fiind o aproximare decentă a luminii care există împrăștiată în jurul unei scene. Calculul său nu depinde de poziția spectatorului sau de direcția luminii. Valoarea iluminării ambientale poate fi precalculată ca efect global sau adăugată independent pentru fiecare sursă de lumină.

Lumina difuză este împrăștiată în mod egal în toate direcțiile pentru o sursă de lumină. Cantitatea de lumină difuză care există pe un punct nu depinde de poziția spectatorului, dar depinde de direcția luminii. Intensitatea luminoasa este mai puternică pe suprafețele care sunt orientate spre sursa de lumină și mai slabă pe cele orientate în direcția opusă. Calculul luminii difuze depinde de normala suprafeței și de direcția sursei de lumină, dar nu și de direcția de vizualizare.

Lumina speculară este lumina reflectată direct de către suprafață și se referă la cât de asemănătoare este suprafata (materialul) obiectului cu o oglindă. Intensitatea acestui efect este denumită Shininess (stralucire). Calculul său necesită să se știe cât de apropiată este orientarea suprafeței de reflecția dintre direcția luminii și ochi. Astfel, calculul luminii speculare depinde de normala suprafaței, de direcția luminii și de direcția de vizionare.”

```
void calculCuloare()
{
    vec3 normSpCamera = normalize(normalMatrix * fNormal);
    vec3 dirLuminaN = vec3(normalize(view * vec4(dirLumina, 0.0f)));
    vec3 dirVizualizare = normalize(vec3(0.0f) - fPosEye.xyz);

    ambient = putereAmbient * culoareLumina;

    diffuse = max(dot(normSpCamera, dirLuminaN), 0.0f) * culoareLumina;

    vec3 dirReflectata = normalize(reflect(-dirLuminaN, normSpCamera));
    float coefSpecular = pow(max(dot(dirVizualizare, dirReflectata), 0.0f), stralucireaSpeculara);
    specular = putereSpecular * coefSpecular * culoareLumina;
}
```

Pentru realizarea luminii punctiforme am folosit algoritmul Phong, calculând pentru fiecare poziție a felinarelor distanța până la fragmentul procesat și atenuarea și adunându-le pe fiecare la componentele ambiental, difuză și speculară. Am folosit pentru implementare formula:

att = 1/ (constant + linear * distance + quadratic * distance^2)

```
void calculCuloareCuFelinare()
{
    ambientFelinare = vec3(0.0f);
    diffuseFelinare = vec3(0.0f);
    specularFelinare = vec3(0.0f);

    for(int i=0; i<=3; i++){

        vec3 normalEye = normalize(fNormal);

        vec4 lightPosEye1 = view * model * vec4(coordonateFelinare[i], 1.0f);
        vec3 lightPosEye = lightPosEye1.xyz;

        vec3 dirLuminaN = normalize(fPosEye.xyz - lightPosEye);
        vec3 viewDirN = normalize(-fPosEye.xyz);
        vec3 reflection = normalize(reflect(dirLuminaN, normalEye));
        float specCoeff = pow(max(dot(viewDirN, reflection), 0.0f), stralucireaSpeculara);

        float distantaL = length(lightPosEye - fPosEye.xyz);
        float atenuareDist = 1.0 / (constant + linear * distantaL + quadratic * (distantaL * distantaL));

        vec3 ambientp = atenuareDist * putereAmbientFelinare * culoareLuminaFelinare;
        vec3 diffusep = atenuareDist * max(dot(normalEye, dirLuminaN), 0.0f) * culoareLuminaFelinare;
        vec3 specularp = atenuareDist * putereSpecularFelinare * specCoeff * culoareLuminaFelinare;

        ambientp *= texture(texturaDifuza, fTexCoords);
        diffusep *= texture(texturaDifuza, fTexCoords);
        specularp *= texture(texturaSpeculara, fTexCoords);

        ambientFelinare += ambientp;
        diffuseFelinare += diffusep;
        specularFelinare += specularp;
    }
}
```

Am folosit aceste metode, deoarece au fost parcurse de-a lungul laboratoarelor.

- ***Animații***

În total am 3 animații care conferă scenei un aspect fotorealist. Animația de cameră deschide aplicația și este formată din 2 alte animații. Prima animație se realizează în primele 19 de secunde de la începerea programului și presupune rotirea camerei deasupra scenei cu un unghi de rotație care este incrementat constant. De asemenea, în prima animație are loc și comutarea la peisajul nocturn, intre secundele 15 si 19.

```
void animatieInceputCamera()
{
    if (animatieComutare)
    {
        if (numarAnimatie == 1)
        {
            unghiRotatieAnimatie += 1.0f;
            cameraUtilizator.animatie1Scena(unghiRotatieAnimatie, glm::vec3(-100.0, 50.0, 40.0));
            if ((glfwGetTime() - timpInceputAnimatie) > 14 && (glfwGetTime() - timpInceputAnimatie) < 20.0)
            {
                ziNoapte = 0;
                initializareSkybox("noapte");
            }
            else if ((glfwGetTime() - timpInceputAnimatie) > 21.0)
            {
                unghiRotatieAnimatie = 0.0f;
                ziNoapte = 1;
                initializareSkybox("zi");
                numarAnimatie = 2;
                timpInceputAnimatie = glfwGetTime();
            }
        }
```

De la secunda 20 se desfășoară animația 2 care presupune aducerea camerei în poziția (0,5,0), ca mai apoi să comute în noile valori ale camerei: poziție (0,2,5), target (0,1.5,0) și sus (0,1,0).

```
else
{
    glm::vec3 destinatie(0.0f, 5.0f, 5.0f);
    float viteza = 0.015f;
    cameraUtilizator.animatie2Scena(destinatie, viteza);
    if ((glfwGetTime() - timpInceputAnimatie) >= 8) {
        animatieComutare = false;
        gps::Camera cameraDupaAnimatii(glm::vec3(0.0f, 2.0f, 5.0f), glm::vec3(0.0f, 1.5f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
        cameraUtilizator = cameraDupaAnimatii;
    }
}
```

Se actualizează poziția cu o interpolare între poziția curentă și poziția destinație, utilizând viteza animației. Se recalculează direcția către înainte bazată pe noul target și poziția, iar apoi se recalculează direcția către dreapta bazată pe direcția înainte și direcția sus.

```
void Camera::animatie1Scena(float unghiRotatie, const glm::vec3 pozitieCamera)
{
    this->pozitie = pozitieCamera;
    glm::mat4 rotatie = glm::rotate(glm::mat4(1.0f), glm::radians(unghiRotatie), glm::vec3(0, 1, 0));
    this->pozitie = glm::vec4(rotatie * glm::vec4(this->pozitie, 1));
    this->directieInainte = glm::normalize(this->target - this->pozitie);
    this->directieDreapta = glm::normalize(glm::cross(this->directieInainte, this->directieSus));
}

void Camera::animatie2Scena(const glm::vec3& pozitieDestinatie, float vitezaAnimatie) {

    vitezaAnimatie = glm::clamp(vitezaAnimatie, 0.0f, 1.0f);
    this->pozitie = glm::mix(this->pozitie, pozitieDestinatie, vitezaAnimatie);
    this->directieInainte = glm::normalize(this->target - this->pozitie);
    this->directieDreapta = glm::normalize(glm::cross(this->directieInainte, this->directieSus));

}
```

Am implementat în acest mod, deoarece așa am considerat că ar fi cel mai potrivit pentru scenă să fie văzută de sus și să ducă utilizatorul în centru.

De asemenea, am mai implementat 2 animații ale obiectelor navă și broască, care se deplasează continuu pe axele z și respectiv x, între coordonate fixate specifice dimensiunilor planurilor de apă și respectiv nisip. Atunci când obiectul ajunge la o limită are loc o transformare de rotație care are rolul de schimbare a direcției sale de mers. Mișcarea broaștei este sistată în timpul nopții și a amândurora atunci când user-ul apasă tastele de rotație a scenei Q și E.

```
void deseneazaNava(gps::Shader shader, bool depthPass)
{
    if (navaPozitie.z >= 127.0f || navaPozitie.z <= -127.0f) {
        directieNava = !directieNava;
        unghiNava = unghiNava + 180.0f;
    }

    if (directieNava) {
        navaPozitie.z = navaPozitie.z + 0.1f;
    }
    else {
        navaPozitie.z = navaPozitie.z - 0.1f;
    }

    model = glm::rotate(glm::mat4(1.0f), glm::radians(unghiRotatieScena), glm::vec3(0.0f, 1.0f, 0.0f));
    model = glm::translate(model, navaPozitie);
    model = glm::rotate(model, glm::radians(unghiNava), glm::vec3(0.0f, 1.0f, 0.0f));
    model = glm::translate(model, -navaPozitie);
    model = glm::translate(model, navaPozitie);

    if (!depthPass) {
        normalMatrix = glm::mat3(glm::inverseTranspose(view * model));
        glUniformMatrix3fv(normalMatrixLoc, 1, GL_FALSE, glm::value_ptr(normalMatrix));
    }

    glUniformMatrix4fv(glGetUniformLocation(shader.shaderProgram, "model"), 1, GL_FALSE, glm::value_ptr(model));
    nava.Draw(shader);
}
```

Am ales această soluție deoarece am dorit să animez cumva scena pe care am construit-o și cele două obiecte au fost cele mai potrivite pentru animare.

- ***Ceața***

Pentru stabilirea coeficientului de ceață am implementat formula cetii exponențiale pătratice:

coef = e ^-(distFragment * intensitateCeata) ^2

Are loc o incrementare/decremetare a intensității cu ajutorul tastelor F și G. În shadere am calculat poziția obiectului și distanța relativă la camera de vizualizare și înlocuind în formulă am obținut coef pe care l-am utilizat în stabilirea culorii fragmentului prin interpolare.

```
float calculCoeficientCeata()
{
    float distantaFrag = length(fPosEye);
    float ceataCoef = exp(-pow(distantaFrag * intensitateCeata, 2));
    return clamp(ceataCoef, 0.0f, 1.0f);
}
```

- ***Mișcare cameră***

Schimbarea unghiului de vizualizare se face cu ajutorul mișcărilor mouse-ului, iar mișcarea pe direcții cu ajutorul tastelor. Pentru rotația camerei se preiau coordonatele inițiale ale mouse-ului, iar apoi are loc rotația în funcțiile mișcării. Mișcarea pe direcții se realizează relativ la poziție pe o anumită direcție și cu o anumită viteză stabilită în main.

```
void Camera::miscaCamera(DIRECTIE directie, float viteza)
{
    switch (directie)
    {
        case INAINTE:
            this->pozitie = this->pozitie + (viteza * this->directieInainte);
            break;
        case INAPOI:
            this->pozitie = this->pozitie -  (viteza * this->directieInainte);
            break;
        case STANGA:
            this->pozitie = this->pozitie - (viteza * this->directieDreapta);
            break;
        case DREAPTA:
            this->pozitie = this->pozitie + (viteza * this->directieDreapta);
            break;
    }
}
```

- ***Umbre***

Pentru construirea umbrelor am folosit shadow mapping, care este o tehnică multi-trecere care utilizează texturi de adâncime pentru a decide dacă

un punct se află în umbră sau nu. Cheia este aceea de a observa scena din punctul de vedere al sursei de lumină în loc de locația finală de vizionare (locația camerei). Orice parte a scenei care nu este direct observabilă din perspectiva luminii va fi în umbră.

```
float calculCoeficientUmbre()
{
    float umbraCoeficient = 0.0f;

    vec3 coordNormalizate = fragPosLightSpace.xyz / fragPosLightSpace.w;
    coordNormalizate = coordNormalizate * 0.5 + 0.5;
    if (coordNormalizate.z > 1.0f)
        return 0.0f;

    float adanAprop = texture(shadowMap, coordNormalizate.xy).r;
    float adanCur = coordNormalizate.z;
    float bias = max(0.05f * (1.0f - dot(fNormal, dirLumina)), 0.005f);
    if (adanCur - bias > adanAprop)
        umbraCoeficient = 1.0f;

    return umbraCoeficient;
}
```

În fragment shader, am transformat coordonatele din spațiul obiectului în spațiul luminii, am verificat dacă punctul este în afara frustului camerei și am calculat adâncimea apropierii si cea curentă. Apoi am tratat cazul de shadow acne.

3.2. Modelul grafic

Am folosit pipeline-ul de rasterizare implementat în OpenGL din figura din laborator, programând doar Vertex și Fragment Shaders.

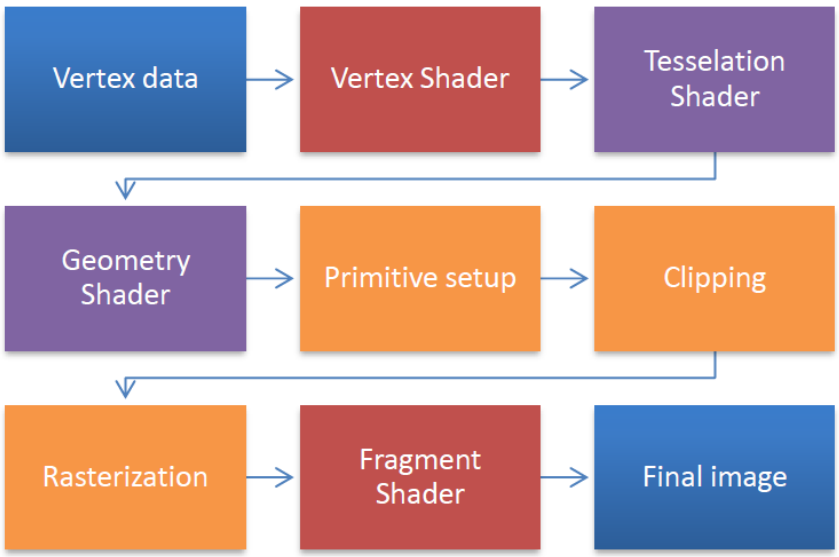
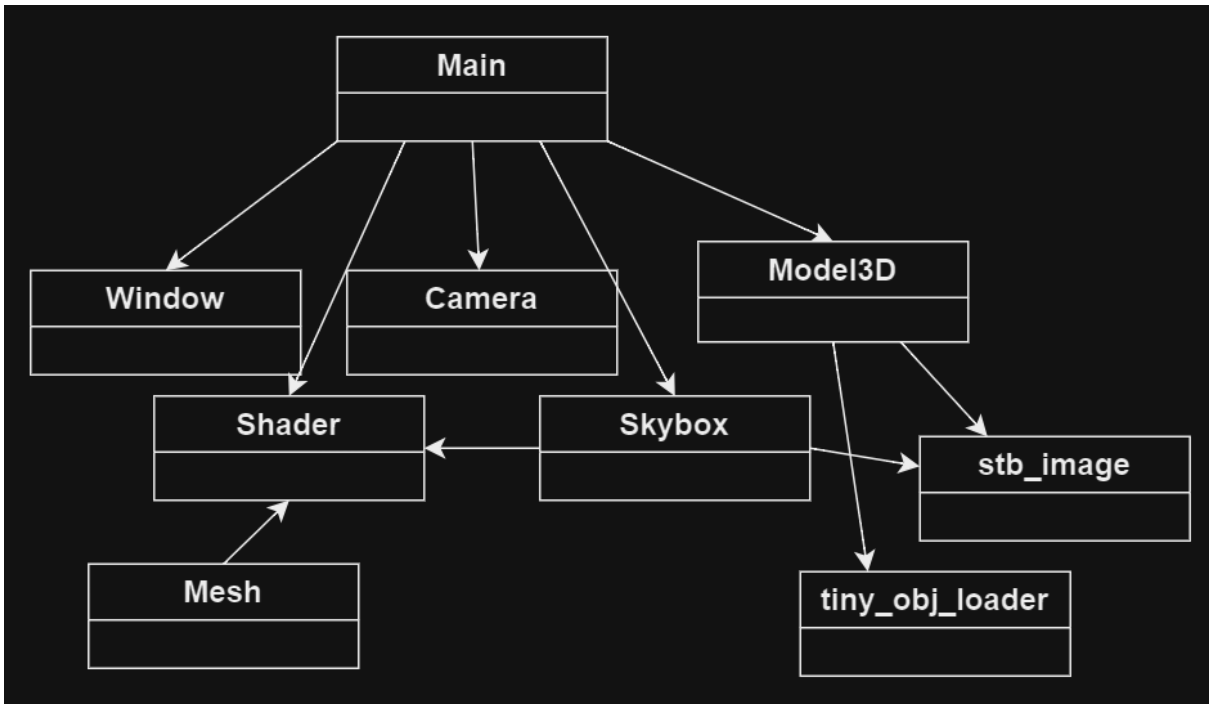


Figure 1 – Pipeline-ul programabil OpenGL

3.3. Structuri de date

Nu am folosit structuri de date adiționale, doar pe cele oferite drept suport pe parcursul laboratoarelor și cele oferite de glm.

3.4. Ierarhia de clase



4. Prezentarea interfeței grafice utilizator / manual de utilizare

Taste

W – mișcare înainte

S – mișcare înapoi

A – mișcare stânga

D – mișcare dreapta

N – comutare la peisaj noapte

Z – comutare la peisaj zi

F – incrementare factor de ceață

G – decrementare factor de ceață

L – rotire lumină dreapta

J – rotire lumină stânga

9 – vizualizare modul solid

8 – vizualizare modul wireframe

7 – vizualizare modul poligonal

6 – vizualizare modul smooth

Q, E – rotire scenă stânga/dreapta

M – vizualizare hartă adâncime

Esc – închidere fereastră aplicație

5. Concluzii și dezvoltări ulterioare

Proiectul m-a ajutat cu familiarizarea cu OpenGl, oferindu-mi o bază solidă în dezvoltarea mea viitoare în acest domeniu. Ceea ce am construit poate să fie puternic dezvoltat prin îmbunătățirea algoritmilor implementați, precum și prin adăugarea de noi funcționalități: adăugarea unui avatar care să întreprindă anumite task-uri, animarea valurilor sau construirea peisajului în mai multe anotimpuri.

6. Referințe

https://www.turbosquid.com/

https://free3d.com/

https://www.youtube.com/playlist?list=PLrgcDEgRZ_kndoWmRkAK4Y7ToJdOf-OSM

https://www.ambiera.com/forum.php?t=7873