Universitatea Tehnică din Cluj-Napoca
Catedra de Calculatoare

Prelucrare Grafică

-proiect-

Bază militară

Student: Pop Ruxandra Maria
Grupa : 30236

Profesor coordonator:
Bacu Victor Ioan

Data 11.01.2022

Table of Contents

1.Prezentarea temei

Proiectul de față are ca scop crearea unei aplicații care redă o scenă 3D realistă a obiectelor folosind API-ul OpenGl . Această scena trebuie să conțină mai multe obiecte și să exemplifiice tehnici precum iluminarea,umbre,animații pentru cameră sau pentru diferite obiecte , înteracțiunea utilizatorului (cu mouse-ul si cu tastatura). După cum se poate observa din titlu , am ales ca si scenă bază militară situată în desert. Librările pe care le-am utilizat pentru realizarea proiectului sunt :GLFW,GLM descrise în [1].

2.Scenariul

2.1 Descrierea scenei și a obiectelor

Scena începe cu un deșert mare care are mai multe elemente plasate pe el (copaci fara frunze , iarba,masina,cutiile).După care se poate oberva baza militară pe care am creat-o în openGl prin adaugarea de obiecte de tip .obj și texturi. Baza militară conține multiple obiecte cum ar fi :tancuri ,mașini militare, grenade, cutii cu arme, o baza pentru un avion, precum și avionul. Deasupra scenei se poate observa un elicopter care zboară în jurul scenei.Toate obiectele din scenă , înafară de elicopterul care zboară deasupra sunt statice , de asemenea obiectele sunt de tip low-poly pentru a putea fi mai ușor de exportat și sunt grupate într-un singur model de tip .obj pe care l-am împortat în proiect.

Figura 1. Scene construită în blender

2.2 Functionalități

Utilizatorul poate interacționa cu aplicația în mai multe moduri :

- **Miscarea camerei** - este cea mai importantă funcționalitate deoarece prezintă controlului utilizatorului asupra aplicației , privind scena într-o manieră la persoana întâi (controlează direct camera) . Camera poate fi controlată prin mouse (pentru a roti vizualizarea în orice direcție dorită) și prin tastatură (tastele A,S,D,W,Shift,Space pentru a muta camera în scena)
- **Manipularea scenei** -utilizatorul are mai multe opțiuni prin care să manipuleze scena și anume poate sa hotarească dacă e zi sau noapte, poate să vizioneze scena în cele 3 moduri : wireframe,solid,poligonal ,poate să pornească o aplicație de prezentare a scenei.

3. Detalii de implementare

3.1 Funcții și algoritmi

Pentru a realiza o împlementare cât mai fotorealistă a proiectului am implementat diferiți algoritmi .

3.1.1 Surse de lumină

În acest proiect există 3 tipuri de surse de lumină .Una dintre ele este o sursă de lumină globală ,direcțională , punctiformă prezentă în garaj , și surse de lumină de tip spotlight folosite pentru a implementa funcționalitatea felinarelor de pe stradă. Lumină punctiformă se impraștie în toate părțiile .
Spre deosebire de lumina punctiformă,care se împraștie în toate părțiile , lumina de tip spotlight nu se împrăstie în toate părțiile și are un unghi de cutOff unde se oprește , iluminând doar o zonă circulară.Toate sursele de lumină au fost realizate prin modelul Blinn-Phong ,care ilumineză obiectul printr-o compunere a componentei abientale,difuze și speculare.Eficientizează calculul componentei speculare prin prezența Half-Vectorului

```
vec3 computeDirLight()
{
    //compute eye space coordinates
    vec4 fPosEye = view * model * vec4(fPosition, 1.0f);
    vec3 normalEye = normalize(normalMatrix * fNormal);

    //normalize light direction
   // vec3 lightDirN = vec3(normalize(view * vec4(lightDir, 0.0f)));
   vec3 lightDirN = normalize(lightDir);
    //compute view direction (in eye coordinates, the viewer is situated at the origin
    vec3 viewDir = normalize(- fPosEye.xyz);

    //compute ambient light
    ambient = ambientStrength * lightColor;

    //compute diffuse light
    diffuse = max(dot(normalEye, lightDirN), 0.0f) * lightColor;

    //compute specular light
    vec3 reflectDir = reflect(-lightDirN, normalEye);
    float specCoeff = pow(max(dot(viewDir, reflectDir), 0.0f), 32);
    specular = specularStrength * specCoeff * lightColor;

    float shadow = computeShadow();
 return   min((ambient + (1.0f - shadow) * diffuse) * texture(diffuseTexture,
     fTexCoords).rgb + (1.0f - shadow) * specular * texture(specularTexture,
     fTexCoords).rgb, 1.0f);

}
```

Figura 2 . Funcția pentru compunerea luminii direcționale

```
vec3 computePointLight(){

    //compute eye space coordinates
    vec4 fPosEye = vec4(fPosition, 1.0f);

    vec3 normalEye = normalize(fNormal);

    //compute light direction
    vec3 lightDirN = normalize(lightPosEye - fPosEye.xyz);

    //compute view direction (in eye coordinates, the viewer is situated at the origin
    vec3 viewDirN = normalize(- fPosEye.xyz);

   //compute half vector
    vec3 halfVector = normalize(lightDirN + viewDirN);

    //compute distance to light
    float dist = length(lightPosEye - fPosEye.xyz);

    //compute attenuation
    float att = 1.0f / (constant + linear * dist + quadratic * (dist * dist));

    //compute ambient light
    vec3 ambient = att * ambientStrength * lightColorPos;

    //compute diffuse light
    vec3 diffuse = att * max(dot(normalEye, lightDirN), 0.0f) * lightColorPos;

    //compute specular light
    float specCoeff = pow(max(dot(normalEye, halfVector), 0.0f), 32.0f);
    vec3 specular = att * specularStrength * specCoeff * lightColorPos;

    return min((ambient + diffuse) * texture(diffuseTexture, fTexCoords).rgb + specular *
        texture(specularTexture, fTexCoords).rgb, 1.0f);

}
```

Figura 3 . Funcția pentru compunerea luminii punctiforme

```
}

vec3 computeSpotLight(Spotlight light)
{
    vec4 fPosEye = light.view * model * vec4(fPosition, 1.0f);
    vec3 normalEye = normalize(light.normalM * fNormal);
    vec3 lightPosEye = vec3(light.view * vec4(light.position, 1.0f));
    vec3 viewDir = normalize(-fPosEye.xyz);

    vec3 lightDirN = normalize(lightPosEye - vec3(fPosEye));

    // diffuse shading
    float diff = max(dot(normalEye, lightDirN), 0.0);
    // specular shading
    vec3 reflectDir = reflect(-lightDirN, normalEye);
    float specCoeff = pow(max(dot(viewDir, reflectDir), 0.0f), 32);
    // attenuation
    float distToLight = length(lightPosEye - vec3(fPosEye));
    float attenuation = 1.0f / (light.constant + light.linear * distToLight +
        light.quadratic * (distToLight * distToLight));
    // spotlight intensity
    float theta = dot(lightDirN, normalize(-vec3(light.view * vec4(light.direction, 1.0f))));
    float epsilon = light.cutOff - light.outerCutOff;
    float intensity = clamp((theta - light.outerCutOff) / epsilon, 0.0, 1.0);
    // combine results
    vec3 ambient = light.color * texture(diffuseTexture, fTexCoords).rgb;
    vec3 diffuse = diff * light.color * texture(diffuseTexture, fTexCoords).rgb;
    vec3 specular = specCoeff * light.color * texture(specularTexture, fTexCoords).rgb;

    ambient *= attenuation * intensity;;
    diffuse *= attenuation * intensity;;
    specular *= attenuation * intensity;;

    return clamp((ambient + diffuse + specular), 0.0f, 1.0f);
}

float computeFog()
```

Figura 4 .Funcția pentru compunerea luminii de tip spotlight

Motivare. Modelul Blinn-Phong a fost folosit deoarece pare mai real decât algoritmul Gourard ,chiar dacă Gourard este mai rapid .

3.1.2 Umbre

Pentru a putea calcula umbrele ,trebuie mai întâi să generăm o textură a hărții de adâncime a umbrei care va conține înformațiile despre umbre asa cum sunt vazute din perspectiva luminii directionale Scena fiind redată de două ori.Mai întâi redă scena din punct de vedere al sursei de lumină ,nu pe ecran ,ci intr-o textură ,ținând evidența celui mai apropiat obiect de lumina direcțională .Textura respectiva va fi apoi utilizată în a doura randare ,prentru a calcula ce obiect este mai aproape de sursa de lumină.

```
float computeShadow(){

    vec3 normalizedCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;

    normalizedCoords = normalizedCoords * 0.5 + 0.5;

    // Get closest depth value from light's perspective
    float closestDepth = texture(shadowMap, normalizedCoords.xy).r;
    // Get depth of current fragment from light's perspective
    float currentDepth = normalizedCoords.z;

    vec3 normal = normalize(normalMatrix * fNormal);
    // Check whether current frag pos is in shadow
    float bias = max(0.05f * (1.0f - dot(normal, lightDir)), 0.005f);
    float shadow = currentDepth - bias > closestDepth ? 1.0f : 0.0f;
    return shadow;
}
```

Figura 4 . Funcția pentru compunerea umbrelor

Motivare. Este algoritmul prezentat la laborator, am ales să îl implementez deoarece a fost usor de înțeles si oferă mai mult realism scenei .

3.1.3 Animația camerei

Pentru a realiza animația camerei am mai adăugat un obiect de tip cameră , pe care am deplasat-o la stanga .

```
bool isCameraAn = false;
gps::Camera anCamera(
                    glm::vec3(-10.0f, 40.0f, -90.0f),
                    glm::vec3(0.0f, 0.0f, 0.0f),
                    glm::vec3(0.0f, 1.0f, 0.0f));
```

```
if (isCameraAn) {
    anCamera.move(gps::MOVE_RIGHT, cameraSpeed);

    anCamera.setCameraTarget(glm::vec3(0.0f, 4.0f, 0.0f));

    view = anCamera.getViewMatrix();


    glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));

    normalMatrix = glm::mat3(glm::inverseTranspose(view * model));

}
```

Figura 5. Noua cameră folosită pentru aplicație

3.1.4 Generarea ceață

Am introdus și efectul de ceață pentru a reda mai mult realism scenei. Pentru a genera ceața m-am folosit de mecanismul simplu care este încorporat în OpenGl.Când ceața este activată ,obiectele încep să se estompeze . Densitatea ceții poate fi controlotă prin tastele 8 respectiv 9. Am folosit ceață exponențială pătratică.

$$fogFactor = e^{-(fragmentDistance * fogDensity)^2}$$

```
float computeFog()
{
    float fragmentDistance = length(fPosition);
    float fogFactor = exp(-pow(fragmentDistance * fogDensity, 2));
    return clamp(fogFactor, 0.0f, 1.0f);

}
```

Figura 6. Compunerea ceții

3.1.5 Generarea animație pentru avion

Aplicația funcționează îndependent de capacitățile de refresh ale scenei.Am crescut delta pe baza unei unități pe secunda și am măsurat timpul dintre actualizările succesive și am calculat dinstanța pe care să se deplaseze obiectul.
Elicopterul este compus din două obiecte , corpul propriu zis și elicea.Astfel avem două matrici de model pentru fiecare obiect,miscarea lor fiind astfel independentă unul de cealaltălt ,dând senzația ca elicopterul se deplasează corect.

3.1.6 Eliminarea fragmentelor

```
vec4 texColorDifusse=texture(diffuseTexture,fTexCoords);
if(texColorDifusse.a <0.0005){
    discard;
}
```

Figura 7. Eliminarea fragmentelor

3.2 Modelul grafic

Pentru a realiză scena ,am folosit mai multe modele 3D pe care le-am descarcat gratuit de pe internet de pe site-urile [2] și [3]. În legătură cu texturile ,le-am luat pe cele care erau deja la obiecte .Software-ul folosit pentru a modela obiectele ,pentru a creea scena și a exporta-o în format obj a fost Blender (mai multe detalii despre acest soft se găsesc în [4] .
Primul pas la reprezentat , construcția modelului terenului care s-a realizat cu modulul Sculpt din blender pornind de la niste plane.După care am adăugat pe rând modele descarcate de pe internet.

3.3 Structuri de date

Structurile de date utilizate sunt cele prezentate și discutate în laborator.

```
0  //spotlight
1  struct Spotlight {
2      mat4 view;
3      mat3 normalM;
4
5      vec3 position;
6      vec3 direction;
7      float cutOff;
8      float outerCutOff;
9
0      float constant;
1      float linear;
2      float quadratic;
3
4      vec3 color;
5  };
```

```
struct Vertex
{
    glm::vec3 Position;
    glm::vec3 Normal;
    glm::vec2 TexCoords;
};

struct Texture
{
    GLuint id;
    //ambientTexture, diffuseTexture, specularTexture
    std::string type;
    std::string path;
};

struct Material
    {
        glm::vec3 ambient;
        glm::vec3 diffuse;
        glm::vec3 specular;
    };

struct Buffers {
    GLuint VAO;
    GLuint VBO;
    GLuint EBO;
};
```

- Vertex -menține coordonatele de poziție,normală și texturp
- Texture -deține id-ul,tipul (ambient,difuz sau specular) și calea
- Material- are trei vectori care descriu modul în care se comportă cu lumina ambientală ,difuză și specular
- Spotligh – conține atributele pe care trebuie să le aibe o lumină de tip spotlight

3.4 Ierarhia de clase

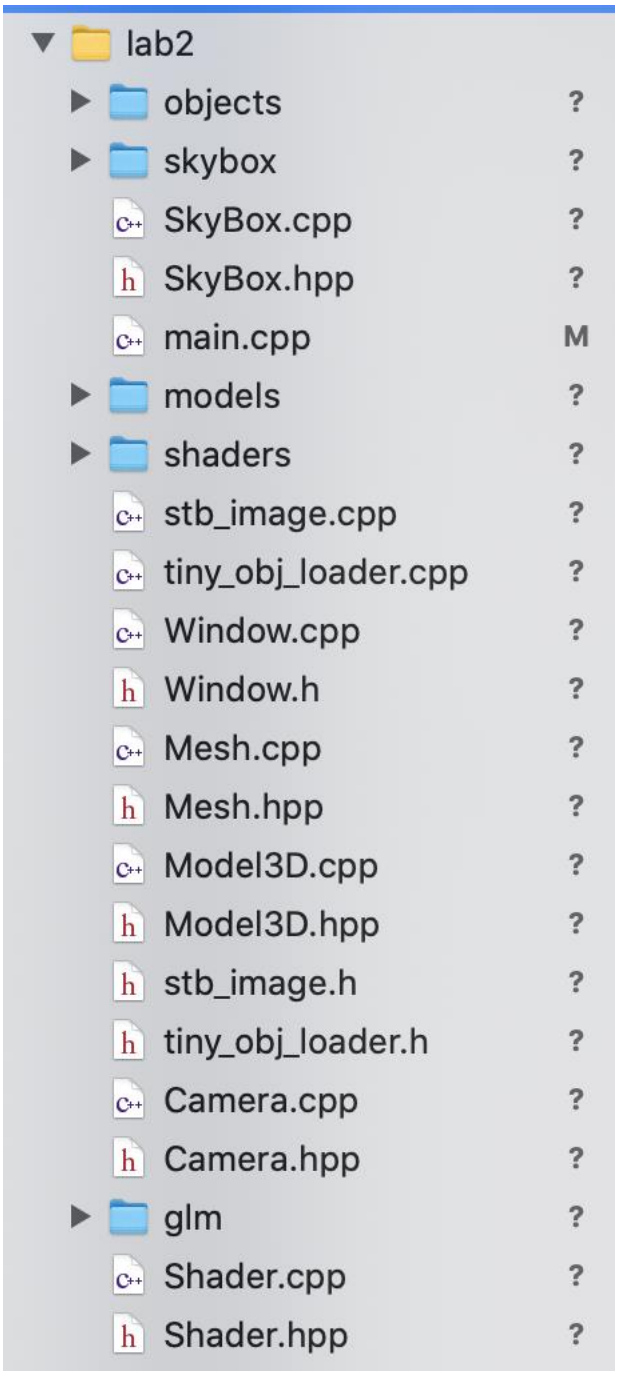

Figura 7. Ierarhia de clase

Fiecare clasă are următorul rol:

- SkyBox – folosit pentru citirea și desenarea sky-boxului de noapte si de zi
- Main – este clasa principală a proiectului ,aici sunt implementate toate metodele importante ale aplicației
- Mesh – deține informații despre o plasă (poligon) și structurile care o alcătuiesc
- Model3D- este folosită pentru citirea obiectului dintr-un fisier ,citirea texturi aferente și desenarea obiectului folosind un shader.Pe scurt deține informații despre obiecte
- Camera -pastrează poziția ,direcția și target-ul camerei , este responsabilă pentru miscarea camerei și rotirea acesteia , are si metode de mutare a camerei pentru prezentarea scenei. Aici am implementat metodele de miscarea ale camerei.

4. Manual de utilizare

Figura 8 . Scena vazută pe timp de zi

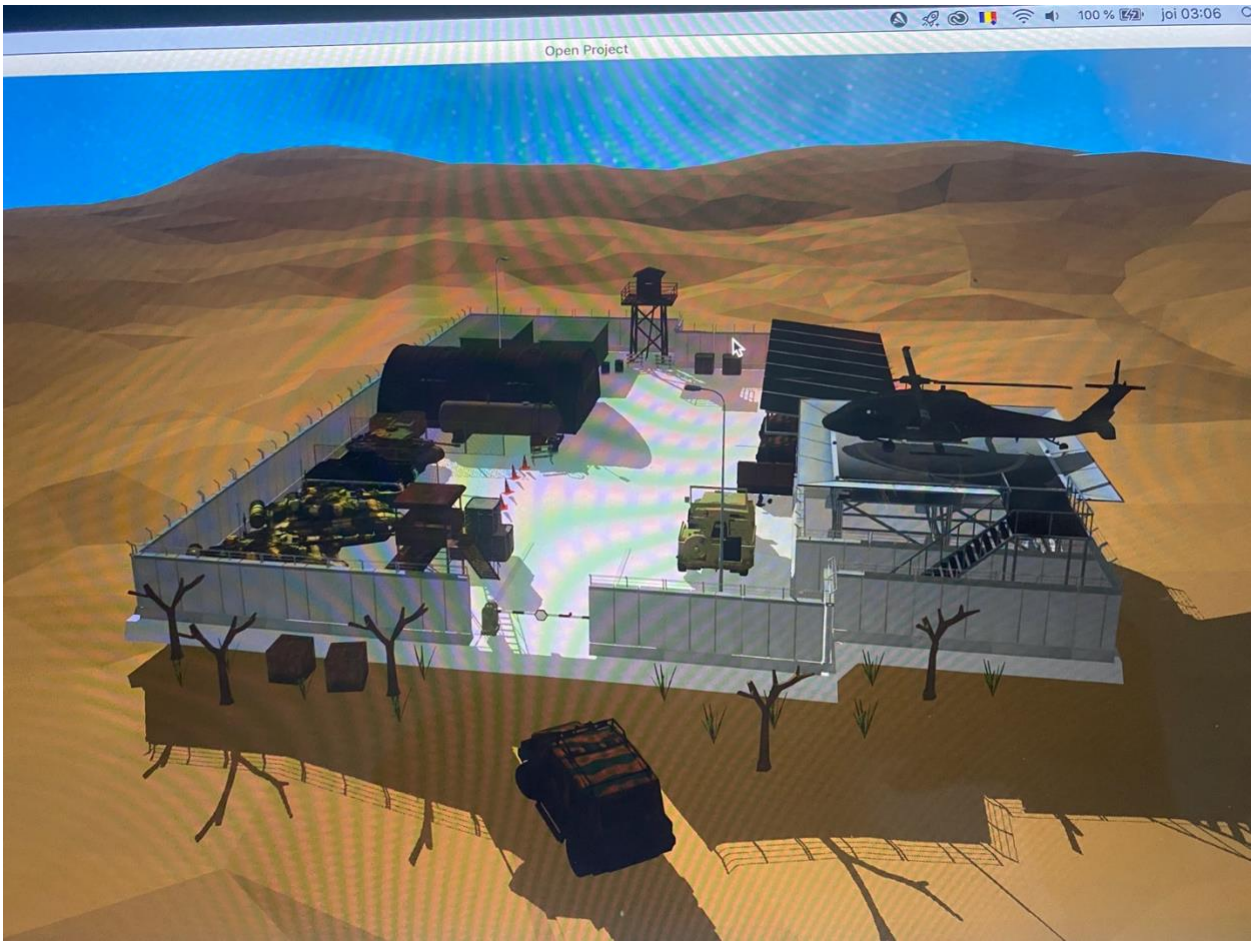

Figura 9 . Scena vazută pe timp de noapte

Utilizatorul poate efectua următoarele funcții pe scenă folosind următoarele taste de la tastatură:

- W – merge camera în față
- S -merge camera în spate
- A – merge camera în stânga
- D – merge camera în dreapta
- Shift – merge camera în jos
- Space -merge camera în sus
- Q – se rotește camera la stânga
- E -se roteste camera la dreapta
- Esc – ieșire din aplicație
- M- afisare harți de adâncime pentru umbră
- 1- afisare mod wireframe
- 2 – afisare mod solid

- 3 – afisare mod poligonal
- 8 – scade densitatea ceți
- 9 – creste densitatea ceți
- N – schimba din zi/noapte
- V – porneste animația de cameră

5. Concluzie

Acest proiect a reprezentat pentru mine o introducere în dezolvatrea OpenGL, ajutându-ma să mă familarizez cu pașii implementării unei astfel de aplicați , precum și șă mă obisnuiesc cu pipeline-ul grafic și cu shaders. Totodată am reusit sa experimentez bibliotecile :GLM,GLFW. De asemnea am reusit să învăț să folosesc Blender, să modelez diferite obiecte și să adaug texturi.

În ceea ce priveste aplicația pe care am implementat-o ,există suficient loc de îmbunătățire . In primul rând ar trebui adaugată coleziunea obiectelor, după aia s-ar putea adauga mai multe efecte cum ar fi ploaie , sau vânt . Adăugarea de umbre pentru luminile punctiforme .,mai multe animați Și nu în ultimul rând ,scena ar putea fi extinsă , cu mai multe obiecte.

6. Referințe

- [1] https://cis.gvsu.edu/~dulimarh/Okto/cis367/
- [2] https://www.google.com/search?client=safari&rls=en&q=model+3+d&ie=UTF-8&oe=UTF-8
- [3] https://www.cgtrader.com/3d-models
- [4] https://en.wikipedia.org/wiki/Blender_(software)