MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Simulator - Fermă la munte

-Aplicație OpenGL-

-Universitatea Tehnică din Cluj-Napoca-

2023-2024

Student: Buda Andreea Rodica

Grupa: 30231

1. Cuprins

2. Prezentarea temei

Proiectul are ca obiectiv crearea unei prezentări fotorealiste a unor scene de obiecte 3D, în C++ folosind OpenGL. Se va folosi Blender pentru poziționarea, modelarea și texturarea obiectelor, care vor fi ulterior importate în proiectul principal. Utilizatorul are, de asemenea, posibilitatea să interacționeze cu scena, să se deplaseze în jurul ei și să declanșeze animații, având posibilitatea de a controla scena prin intermediul mouse-ului și tastaturii.

3. Scenariul

3.1. Descrierea scenei și a obiectelor

Scena reprezintă o fermă situată într-o zonă montană, fiind modelată prin plasarea atentă a mai multor obiecte pe un plan personalizat în Blender, utilizând modul *Sculpt* pentru a crea dealuri și adâncituri, inclusiv un lac generat cu ajutorul unui alt plan. Mai multe obiecte au fost preluate de pe site-urile "free3d.com" și "models-resource.com", în timp ce câteva dintre ele au fost create manual în Blender.

Obiectele plasate în scenă includ:

Animale:

- 2 căprioare
- 1 cerb
- 1 miel
- 8 găini

Clădiri:

- cabană mobilată cu diverse obiecte
- hambar
- turn de supraveghere
- moară de vânt

Arbori:

- diverse tipuri de arbori în 47 de exemplare, majoritatea creați în Blender
- 5 trunchiuri de copac

Elemente Peisagistice:

- stânci mari, menite să creeze impresia de munți, modelate în Blender

Alte Elemente:

- gard
- trăsură
- barcă
- pod
- secure

Majoritatea obiectelor au fost texturate manual în Blender, având în vedere că au fost preluate într-o formă netexturată. Scena a fost organizată în Blender și exportată astfel încât coordonatele relative ale obiectelor să nu necesite modificări din cod, asigurând astfel o integrare fluentă în proiectul final.

3.2. Funcționalități

Utilizatorul nu doar va observa scena, ci va interacționa activ cu aceasta. Funcționalitățile includ posibilitatea de a se deplasa în jurul scenei folosind atât mouse-ul, cât și tastatura. De asemenea, utilizatorul poate declanșa animația morii de vânt pentru a învârti morișca sau poate experimenta cu o cameră cinematografică, utilizând animații de prezentare. Camera fiind poziționată aproape de centrul scenei, apăsând un buton specific, se poate realiza o vedere panoramică a întregii scene prin rotirea camerei.

Este, de asemenea, posibil să vizualizeze scena cu obiecte într-un mod solid, wireframe sau ca puncte. De asemenea, poate afișa harta de adâncime, care este folosită pentru calculul umbrelor.

4. Detalii de implementare

4.1. Funcții și algoritmi

4.1.1. Soluții posibile

4.1.1.1. Mișcarea camerei

Clasa “Camera” este implementată pentru gestionarea parametrilor și mișcărilor camerei într-un spațiu tridimensional. Constructorul primește poziția camerei, punctul de vizare și direcția de sus, calculând ulterior direcția frontală și laterală ale camerei.

Funcția *getViewMatrix*, prin utilizarea funcției glm::lookAt(), aceasta returnează matricea necesară pentru a plasa camera în spațiu, având în vedere poziția, punctul de vizare și direcția de sus ale camerei. Mișcarea camerei este gestionată prin funcția move, care permite deplasarea acesteia în diverse direcții (MOVE_FORWARD, MOVE_BACKWARD, MOVE_LEFT, MOVE_RIGHT, MOVE_UP, MOVE_DOWN) cu o viteză specificată. Această funcționalitate facilitează explorarea spațiului tridimensional al aplicației.

Funcția rotate permite rotația camerei în jurul axelor Y și X (denumite și yaw și pitch). Această rotație se traduce într-o schimbare a direcției frontale a camerei și a direcției laterale, oferind astfel un control precis asupra orientării camerei. Pentru flexibilitate suplimentară, clasa dispune de funcția *setCameraPosition*, care permite setarea manuală a poziției camerei în spațiu.

În cadrul fișierului Main.cpp, se inițializează o instanță a clasei Camera numită *myCamera*, oferind parametri precum poziția inițială, punctul de vizare și direcția de sus. Interacțiunile utilizatorului sunt gestionate prin verificarea tastelor apăsate, permițând controlul mișcării camerei, activarea/dezactivarea anumitor funcționalități și ajustarea parametrilor cum ar fi densitatea ceții (*fogDensity*).

Implementarea mișcării mouse-ului (*mouseCallback*) calculează offset-ul mouse-ului pentru a regla unghiurile yaw și pitch ale camerei. Sunt aplicate limite pentru a preveni comportamente de vizualizare neașteptată. Funcția *processMovement* reacționează la tastele apăsate pentru a ajusta parametrii precum unghiurile de rotație (angleY), unghiul de iluminare (lightAngle), și pentru a efectua mișcarea în spațiu utilizând taste precum W, A, S, D, Space, și Left Control.

Funcția *prezentareScena* este utilizată pentru a gestiona prezentarea scenei, inclusiv animația. Dacă animația este activată (animate este true), atunci se efectuează o rotație continuă a camerei în jurul axelor Y și X (specificată de variabilele pitch și yaw). De asemenea, se configurează poziția și orientarea camerei pentru a iniția animația. Rotația cinematică este realizată prin actualizarea continuă a unghiurilor de pitch și yaw, ceea ce contribuie la efectul de rotație în timpul afișării scenei.

Această implementare facilitează controlul asupra camerei în cadrul unei aplicații 3D, oferind utilizatorului posibilitatea de a naviga și explora mediul virtual.

4.1.1.2. Iluminarea

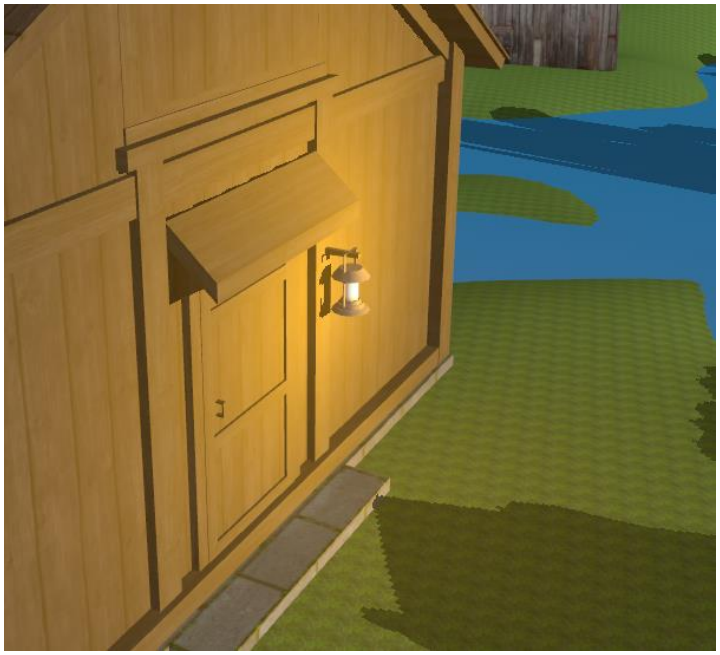
Direcțională:

Aplicația utilizează iluminare direcțională, ceea ce înseamnă că sursa de lumină este considerată a fi la o distanță infinită. Acesta este un abordaj comun și eficient deoarece simplifică calculațiile de iluminare.

Pentru a implementa iluminarea direcțională, aplicația creează inițial o hartă de adâncime, care este o textură ce stochează informații despre adâncimea fiecărui pixel din scenă. Acest lucru se realizează prin randarea scenei din perspectiva luminii și scrierea doar a informațiilor despre adâncime în buffer-ul de adâncime. Pașii specifici pe care aplicația îi urmează sunt urmatorii:

- Creează o hartă de adâncime: În funcția renderScene(), shader-ul depthMapShader este folosit pentru a randare scena din perspectiva luminii. Informațiile despre adâncime sunt scrise într-o textură de adâncime numită depthMapTexture.
- Folosește harta de adâncime pentru umbre: În pasul principal de randare al funcției renderScene(), shader-ul myCustomShader este folosit pentru a randare scena din perspectiva camerei. Harta de adâncime este legată ca o textură și este folosită pentru a calcula umbrele pentru fiecare pixel.
- Desenează un cub alb în jurul luminii: Shader-ul lightShader este utilizat pentru a desena un cub mic, alb, în jurul sursei de lumină. Acest cub servește ca o reprezentare vizuală a poziției și direcției luminii.
- Desenează skybox-ul: Shader-ul skyboxShader este utilizat pentru a desena skybox-ul. Skybox-ul este un cubemap folosit pentru a crea iluzia unui mediu îndepărtat.

Pozițională:



```
void punctLumina()
{
    // Calcul transformare normală
    vec3 normalEye = normalize(fNormal);

    // Calcul distanță și atenuare
    float distanta = length(pozitieLumina - fPosition.xyz);
    float atenuare = 1.0f / (constanta + liniar * distanta + cuadratic * (distanta * distanta));

    // Direcția luminii și direcția privitorului
    vec3 lightDir = normalize(pozitieLumina - fPosEye.xyz);
    vec3 viewDirN = normalize(-fPosEye.xyz);

    // Calcul componente diffuse, ambient și specular
    diffuseP = atenuare * max(dot(normalEye, lightDir), 0.0f) * vec3(1.0f, 0.478f, 0.0f);
    ambientP = atenuare * ambientStrength * vec3(1.0f, 0.478f, 0.0f);

    // Calcul reflexie specular
    vec3 reflection = reflect(-lightDir, normalEye);
    float specCoeff = pow(max(dot(viewDirN, reflection), 0.0f), shininess);
    specularP = atenuare * specularStrength * specCoeff * vec3(1.0f, 0.478f, 0.0f);
}
```

Acest fragment shader implementează iluminarea poziţională pentru a simula efectul de iluminare provenind de la o sursă de lumină poziţională în scenă. Prin intermediul variabilelor şi funcţiilor, shader-ul calculează componente precum ambientală, difuză şi specular, având în vedere atenuarea în funcţie de distanţa faţă de sursa de lumină.

Variabile pentru Iluminare și Textură:

- lightDir: Direcția luminii în spațiul obiectelor.
- lightColor: Culoarea luminii.
- ambientStrength, specularStrength: Factori de putere pentru componentele ambientale și specular.
- shininess: Factor de strălucire specular.
- fogDensity: Densitatea pentru calculul efectului de ceață.
- liniar, constanta, cuadratic: Coeficienți pentru atenuarea luminii în funcție de distanță.

Texturi și Unelte

- diffuseTexture, specularTexture: Texturi pentru harta de difuzie și specularitate.
- shadowMap: Harta umbrelor pentru efectul de umbră.

4.1.1.3. Exemplificarea umbrelor

Împărțirea perspectivă:

- normalizedCoords reprezintă coordonatele normalizate ale fragmentului în spațiul de adâncime al camerei de lumină. Aceasta este o etapă standard în calculul umbrelor.
- normalizedCoords este transformat pentru a se afla în intervalul [0,1]. Această etapă este necesară pentru a folosi coordonatele în cadrul hărții de adâncime.

Verificarea dacă este în afara frustrumului:

- se verifică dacă fragmentul este în afara frustrumului camerei de lumină. Dacă este în afara, nu contribuie la umbre.

Obținerea celei mai apropiate adâncimi:

- closestDepth reprezintă valoarea adâncimii celei mai apropiate din perspectiva camerei de lumină, obținută din harta de adâncime.

Obținerea adâncimii fragmentului curent:

- currentDepth reprezintă adâncimea fragmentului curent din perspectiva camerei de lumină.

Verificarea dacă Poziția Fragmentului Curent este în Umbră:

Se aplică un mic bias pentru a evita artefactele. Dacă adâncimea fragmentului curent, ajustată cu bias, este mai mare decât cea mai apropiată adâncime, atunci fragmentul este în umbră și shadow primește valoarea 1.0, altfel primește valoarea 0.0.

```
float computeShadow()
{
        // Realizăm împărțirea perspectivă
        vec3 normalizedCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;

        // Transformăm în intervalul [0,1]
        normalizedCoords = normalizedCoords * 0.5 + 0.5;

        // Verificăm dacă fragmentul este în afara frustrumului camerei de lumină
        if (normalizedCoords.z > 1.0f)
                return 0.0f;

        // Obținem valoarea adâncimii celei mai apropiate din perspectiva camerei de lumină
        float closestDepth = texture(shadowMap, normalizedCoords.xy).r;

        // Obținem adâncimea fragmentului curent din perspectiva camerei de lumină
        float currentDepth = normalizedCoords.z;

        // Verificăm dacă poziția fragmentului curent este în umbră
        float bias = 0.0005f;
        float shadow = currentDepth - bias > closestDepth ? 1.0f : 0.0f;

        return shadow;
}
```

4.1.1.4. Texturarea

În cadrul proiectului, am adăugat texturi la majoritatea obiectelor pentru a le conferi realism și detaliu. Atunci când am descărcat modelele inițiale în Blender, acestea nu aveau texturi, iar texturarea a devenit necesară pentru a îmbunătăți aspectul final al scenei.

Procesul de texturare cu Blender l-am început prin a deschide Editorul de Imagine în Blender, unde am gestionat și adaptat texturile pentru modele. Am utilizat opțiuni precum "Unwrap" pentru a asocia porțiuni ale modelelor cu texturi 2D, asigurându-ne că acestea se potrivesc în mod corespunzător și oferă aspectul dorit. Am ajustat detaliile texturii, inclusiv repetarea, scalarea și rotația, iar Blender a facilitat adăugarea de hărți normale sau speculare pentru a îmbunătăți calitatea vizuală.

Integrarea modelelor texturate în OpenGL s-a realizat astfel: am exportat modelele texturate din Blender în formate suportate de OpenGL, iar apoi, în OpenGL, am încărcat texturile și am implementat shader-urile pentru manipularea acestora în timpul procesului de rendering. Coordonatele de textură stabilite în Blender au fost utilizate în OpenGL pentru a mapează texturile pe obiecte, contribuind la realismul general al scenei 3D.

4.1.1.5. Efectul de ceață

Am implementat efectul de ceață prin adăugarea unui parametru numit "fogDensity" în shaderul de fragmente. Acest parametru controlează cât de densă este ceața și poate fi ajustat dinamic. Atunci când apăs tasta "F," incrementez densitatea ceții, iar când apăs tasta "G," o decrementez. Aceste modificări ale densității sunt propagate în shaderul principal folosit pentru randarea scenei.

În shaderul principal, am introdus variabila "fogDensity" și am calculat culoarea finală a pixelului în funcție de aceasta. Cu cât un obiect este mai îndepărtat în scenă, cu atât este afectat mai mult de ceață, ceea ce contribuie la un aspect mai realist al adâncimii și distanței în peisaj.

4.1.1.6. Animații

```
456  void rotireElice(gps::Shader shader) {
457      if (animatieElice) {
458          model = glm::translate(glm::mat4(1.0f), 1.0f * glm::vec3(206.087f, 29.0071f, -17.413f));
459          model = glm::rotate(model, glm::radians(unghiRotireElice), glm::vec3(0.0f, 0.0f, 1.0f));
460          model = glm::translate(model, 1.0f * glm::vec3(-206.087f, -29.0071f, 17.413f));
461          glUniformMatrix4fv(glGetUniformLocation(shader.shaderProgram, "model"), 1, GL_FALSE, glm::value_ptr(model));
462
463          unghiRotireElice = unghiRotireElice + 0.6f;
464      }
465  }
466
```

Această funcție este apelată în fiecare cadru și este responsabilă de rotirea modelului elicei. Ea primește ca parametru un obiect gps::Shader pentru a seta și utiliza shader-ul corespunzător.

Această funcție folosește funcțiile din biblioteca GLM pentru a manipula matricile de transformare ale modelului. Ea aplică o translație pentru a plasa centrul rotației în punctul dorit, efectuează rotația și apoi translatează înapoi la poziția inițială. Această secvență de transformări creează o rotație în jurul punctului specificat.

Animarea elicei este activată și dezactivată prin variabila booleană animatieElice. Dacă aceasta este setată pe true, atunci funcția rotireElice va fi apelată în fiecare cadru.

```
if (key == GLFW_KEY_Y && action == GLFW_PRESS) {
    animatieElice = !animatieElice;
}
```

Astfel, apăsarea tastei Y inversează starea variabilei animatieElice, pornind sau oprind astfel animația elicei.

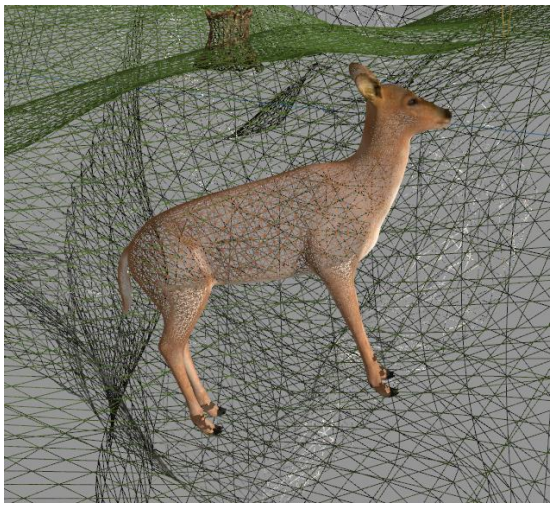
4.1.1.7. Vizualizarea obiectelor în diferite moduri

```
if (key == GLFW_KEY_H) {
    // Modul de vizualizare Wireframe (contur)
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
}

if (key == GLFW_KEY_J) {
    // Modul de vizualizare Solid (umplere)
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
}

if (key == GLFW_KEY_G) {
    // Modul de vizualizare Puncte
    glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
}
```

Acestea sunt comenzi care schimbă modul de desenare al poligoanelor pentru a le reprezenta fie ca wireframe (contur), fie ca obiecte solide (umplute), fie ca puncte.



4.1.2. Motivarea abordării alese

Motivul pentru adoptarea strategiei prezentate în secțiunea anterioară a fost dorința de a concepe o scenă de fermă cât mai realistă și autentică, integrând o ambianță corespunzătoare și detalii de decor. Animația morii de vânt a fost implementată cu scopul de a adăuga dinamism scenei, iar utilizarea camerei cinematice are ca obiectiv oferirea utilizatorului unei perspective rapide asupra întregii scene, o vedere panoramică.

4.2. Modelul grafic

În aplicația mea, modelul grafic este construit folosind OpenGL, o bibliotecă grafică 3D, pentru a crea o scenă tridimensională interactivă. Iată câteva aspecte relevante despre modelul grafic în contextul aplicației tale

- Folosesc GLFW pentru gestionarea ferestrei OpenGL și a evenimentelor de input și utilizez GLEW pentru gestionarea extensiilor OpenGL pe platforme non-Apple.
- Folosesc un sistem de coordonate tridimensional pentru poziționarea și orientarea obiectelor.
- Există un model al terenului ("Peisaj3"), al unei elice și al unui cub de lumină.
- Există mai multe șabloane de shader-uri, inclusiv pentru scene, lumină, adâncime și skybox.
- Folosesc o cameră 3D (gps::Camera) pentru a controla perspectiva și poziția privitorului în
- scenă.
- Am implementat o hartă de adâncime pentru a simula umbre realiste în scenă.
- Există un skybox care furnizează un fundal coerent pentru scenă.

4.3. Structuri de date

Structurile principale și funcțiile din cod includ:

- GLFW și OpenGL Setup:
  - initOpenGLWindow: Inițializează fereastra OpenGL utilizând GLFW.
  - initOpenGLState: Setează starea OpenGL precum culoarea de fundal și funcțiile pentru teste de adâncime și cull-face.
  - initFBO: Inițializează un Frame Buffer Object pentru generarea unei hărți de adâncime (shadow map).

- Shader și obiecte 3D:

  - Shader și Model3D: Acestea sunt clase pentru gestionarea shaderelor și modelelor 3D.
  - initObjects: Încarcă modelele 3D ale scenei (peisaj, elice, etc.) și le pregătește pentru afișare.
  - initShaders: Inițializează și încarcă shader-urile necesare pentru iluminare și umbre.
  - initUniforms: Inițializează variabilele uniforme din shader pentru iluminare, proiecție, vizualizare, etc.

- Controlul camerei:

  - processMovement și keyboardCallback: Funcții pentru controlul camerei și a altor acțiuni la apăsarea/dezactivarea unor taste.

- Rendering și animare:

  - renderScene: Funcția principală pentru randarea scenei, inclusiv adâncimea, iluminarea și obiectele 3D.
  - prezentareScena: Funcție pentru prezentarea inițială a scenei și animarea ulterioară.
  - rotireElice: Funcție pentru animarea rotației elicei.

- Skybox și Shader pentru Skybox:

  - SkyBox și skyboxShader: Clase pentru gestionarea unui skybox și shader-ul asociat.

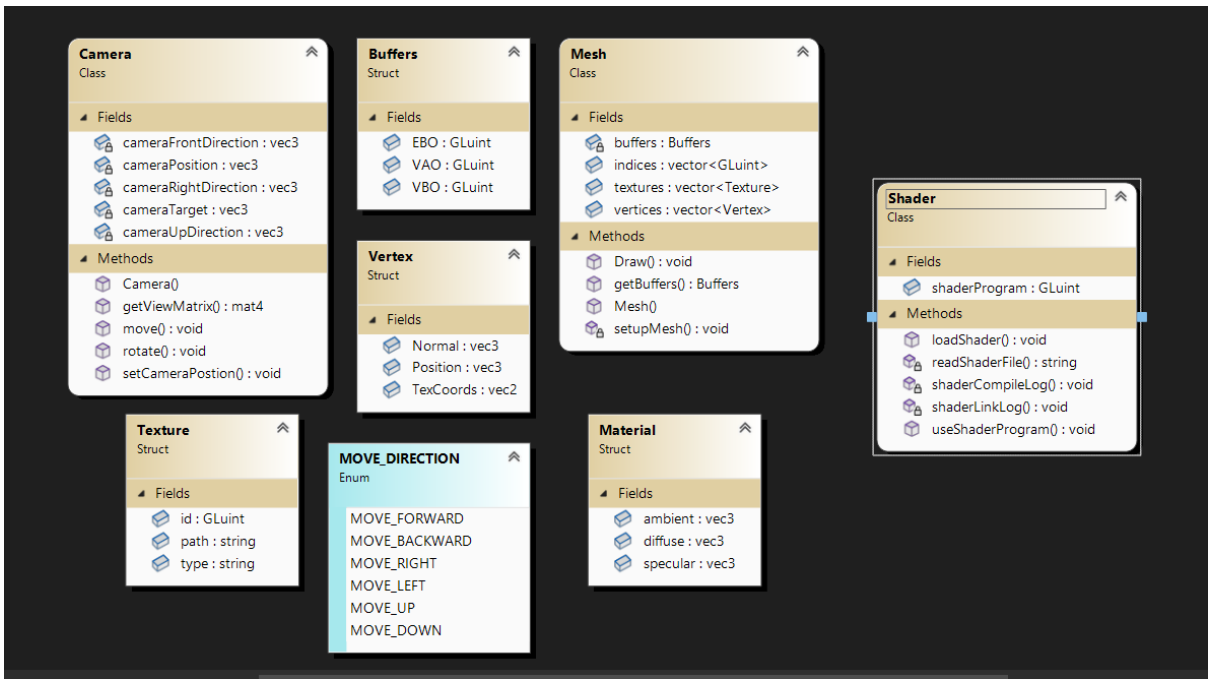
- Input de la Utilizator:

  - mouseCallback: Funcție pentru gestionarea mișcărilor mouse-ului și rotirea camerei.

- Cleanup:

  - cleanup: Funcție pentru eliberarea resurselor OpenGL la închiderea aplicației.

4.4. Ierarhia de clase



5. Prezentarea interfeței grafice utilizator / manual de utilizare

În codul meu, am folosit mai multe taste pentru a controla diferite aspecte ale scenei. Iată o listă a tastelor și funcțiilor corespunzătoare pe care le-am implementat:

Esc: Închide fereastra aplicației.

M: Activează/dezactivează vizualizarea hărții de adâncime (depth map).

K: Pornire animație scenă.

L: Oprire animație scenă.

H: Comutare la modul de vizualizare Wireframe (contur).

J: Comutare la modul de vizualizare Solid (umplere).

B: Comutare la modul de vizualizare Puncte.

Y: Activează/dezactivează animația elicei.

W/A/S/D: Mișcare înainte/stânga/spate/dreapta.

Space/Control: Mișcare sus/jos.

Q/E: Rotire la stânga/dreapta.

F/G: Creștere/scădere densitate ceții.

6. Concluzii și dezvoltări ulterioare

Scopul proiectului de a construi o scenă interactivă în care utilizatorul poate explora o fermă a fost atins, iar rezultatul este o aplicație funcțională. Cu toate acestea, există oportunități ample pentru dezvoltări ulterioare și îmbunătățiri. Printre acestea se numără adăugarea de animații suplimentare pentru a face scena mai vie și captivantă.

De asemenea, s-ar putea explora implementarea unor funcționalități precum oprirea personajului când acesta intră în contact cu obiecte sau adăugarea unor efecte de mediu, cum ar fi schimbările de vreme (ploaie, zăpadă).

Alte îmbunătățiri pot include extinderea interactivității cu obiectele din scenă și integrarea unor personaje non-jucabile pentru a adăuga complexitate și diversitate experienței utilizatorului. Prin continuarea acestor dezvoltări, proiectul poate evolua într-o experiență captivantă și inovatoare, aducând beneficii semnificative în ceea ce privește învățarea și dezvoltarea abilităților de programare și grafică.

7. Referințe

De unde am descărcat obiectele:

https://www.models-resource.com/

https://free3d.com/3d-models/windmill

Tutoriale Blender:

https://www.youtube.com/watch?v=ckGg7aUGoPQ&list=PLrgcDEgRZ_kndoWmRkAK4Y7ToJdOf-OSM&index=1

https://docs.google.com/document/d/1njtWPMmOQNIaD_z9ve8iPRUqQTWdIV_PO-NvPD0nOuM/edit#heading=h.7ck7keotfet

https://www.youtube.com/watch?v=4EqLyGsu3AA

https://www.youtube.com/watch?v=y7PdiGXbrD0&t=1s

Lumina:

https://learnopengl.com/PBR/Lighting

https://www.youtube.com/watch?v=tmCOMzAA4rc

https://learnopengl.com/Lighting/Multiple-lights