UNIVERSTATEA TEHNICĂ DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

PROIECT PRELUCRAREA GRAFICĂ

STUDENT: TODOR ANDREI-PETRU
GRUPA: 30237
PROFESOR ÎNDRUMĂTOR: CONSTANTIN IOAN NANDRA

CUPRINS

1. **Prezentare temei**

Pentru acest proiect am avut următoarele cerințe:

- (2p) vizualizarea scenei: scalare, translație, rotație, mișcarea camerei
    - utilizând tastatura sau mausul
    - utilizând animații de prezentare
- (1p) specificarea surselor de lumină (cel puțin două surse de lumină diferite)
- (0.5p) vizualizare scenei în modurile solid, wireframe, poligonal și smooth
- (1p) maparea texturilor și definirea materialelor
    - calitatea texturilor și nivelul de detaliu al acestora
    - maparea texturilor pe obiecte
- (1p) exemplificarea generării umbrelor
- (0.5p) exemplificarea animării diferitelor componente ale obiectelor
- (3p) fotorealism, complexitatea scenei, nivelul de detaliere al modelării, dezvoltarea diferiților algoritmi și implementarea acestora (generare dinamică de obiecte, detecția coliziunilor, generarea umbrelor, ceață, ploaie, vânt), calitatea animațiilor, utilizarea diferitelor surse de lumină (globală, locală, de tip spot)
- (1p) documentația (obligatorie)

Pentru realizarea acestor cerințe am ales folosirea unei scene care reprezintă un mic aeroport.

2. **Scenariul**

a. Descrierea scenei și a obiectelor

Scena aleasă de mine reprezintă un aeroport unde pot ateriza/decola avioane mari și mici, dar și elicoptere.

Obiectele au fost editate în programul 3D Blender, apoi integrate în peisaj. Tot în Blender a fost realizată și texturarea obiectelor, texturile reprezentând imagini în format .png sau .jpg.

Obiectele individuale sunt:

- Un turn de control aerian
- Un centru de unde se cumpără bilete și se verifică bagajele clienților
- Un acoperiș pentru adăpostirea avioanelor mici
- Un elicopter SMURD
- Un tir cu combustibil
- Două avioane mici

- Două avioane mari
- Două mașini cu platformă
- Două mașini cu cârlig pentru tractare
- Cinci cărucioare de transport
- Opt bariere la capătul pistei
- Mai multe bucăți de drum unite alcătuind pista
- Terenul acoperit de nisip dimprejurul pistei

În proiect am încărcat scena finală, reprezentând un singur obiect.

b. Funcționalități

Proiectul are următoarele funcționalități:

- Navigarea prin scenă
- Pornirea unui sunet de decolare
- Activarea/dezactivarea efectului de ceață
- Vizualizarea scenei în modurile solid/wireframe/punctiform
- Pornirea/oprirea celor 2 surse de lumină din scenă

3. Detalii despre implementare

a. Funcții și algoritmi

i. Soluții posibile

Inițial am pornit de la Project Core (schița proiectului care ne-a fost pusă la dispoziție). Pentru a implementa funcționalitățile din acest proiect, am folosit funcții din biblioteca GLM, algoritmi de iluminare(Phong), precum și alte funcții auxiliare pentru inițializarea variabilelor uniform și legarea acestora cu programul, din shader, funcții de inițializare a ferestrei etc.

În fișierul main.cpp se regăsesc metodele:

- glCheckError(...)
- windowResizeCallback(...)
- keyboardCallback(...)
- mouseCallback(...)
- processMovement() – apăsarea tastelor
- initOpenGLWindow() – crearea ferestrei de vizualizare
- setWindowCallbacks()
- initOpenGLState()
- initModels() – inițializarea modelelor
- initShaders() – inițializarea shaderelor
- initUniforms()
- renderAirport(…)
- renderScene() – afișarea scenei
- cleanup()
- main(…) – metoda principală unde se apelează metodele anterioare

ii. Realizarea funcționalităților

- **Mișcarea** camere pe axele X, Y și Z
- Camera are 3 componente: cameraPosition, cameraTarget și cameraUpDirection
- cameraDirection se calculează ca și diferență normalizată dintre cameraTarget și cameraPositon
- cameraRightDirection se calculează ca produs vectorial între cameraDirection și cameraUpDirection

- **Rotația** camerei pe axele X și Y
- cameraTarget va fi recalculată astfel:

```
void Camera::rotate(float pitch, float yaw) {
    this->cameraTarget.x = this->cameraPosition.x + sin(yaw) * cos(pitch);
    this->cameraTarget.y = this->cameraPosition.y + sin(pitch);
    this->cameraTarget.z = this->cameraPosition.z - cos(yaw) * cos(pitch);
```

- de asemenea, actualizăm cameraDirection și cameraRightDirection

- **Efectul de ceață** în fragment shader (basic.frag), ceață exponențial-pătratică, fogDensity declarat global ca float, valoare 0.03f

```
88  float computeFog()
89  {
90      float fragmentDistance = length(fPosEye);
91      float fogFactor = exp(-pow(fragmentDistance * fogDensity, 2));
92
93      return clamp(fogFactor, 0.0f, 1.0f);
94  }
```

- **Luminile direcționale** în număr de 2, create în shader
- Pentru oprire, setez valorile RGB pe 0
- Fiecare lumină are componentă ambientală, difuză și specular proprie

```
40  void computeDirLight()
41  {
42      //compute eye space coordinates
43      fPosEye = view * model * vec4(fPosition, 1.0f);
44      vec3 normalEye = normalize(normalMatrix * fNormal);
45
46      //normalize light direction
47      vec3 lightDirN = vec3(normalize(view * vec4(lightDir, 0.0f)));
48
49      //compute view direction (in eye coordinates, the viewer is situated at the origin
50      vec3 viewDir = normalize(- fPosEye.xyz);
51
52      //compute ambient light
53      ambient = ambientStrength * lightColor;
54
55      //compute diffuse light
56      diffuse = max(dot(normalEye, lightDirN), 0.0f) * lightColor;
57
58      //compute specular light
59      vec3 reflectDir = reflect(-lightDirN, normalEye);
60      float specCoeff = pow(max(dot(viewDir, reflectDir), 0.0f), 32);
61      specular = specularStrength * specCoeff * lightColor;
62  }
```

```
64  void computeDirLight2()
65  {
66      //compute eye space coordinates
67      fPosEye = view * model * vec4(fPosition, 1.0f);
68      vec3 normalEye = normalize(normalMatrix * fNormal);
69
70      //normalize light direction
71      vec3 lightDirN = vec3(normalize(view * vec4(lightDir2, 0.0f)));
72
73      //compute view direction (in eye coordinates, the viewer is situated at the origin
74      vec3 viewDir = normalize(- fPosEye.xyz);
75
76      //compute ambient light
77      ambient2 = ambientStrength * lightColor2;
78
79      //compute diffuse light
80      diffuse2 = max(dot(normalEye, lightDirN), 0.0f) * lightColor2;
81
82      //compute specular light
83      vec3 reflectDir = reflect(-lightDirN, normalEye);
84      float specCoeff = pow(max(dot(viewDir, reflectDir), 0.0f), 32);
85      specular2 = specularStrength * specCoeff * lightColor2;
86  }
```

- Pentru **sunet** am folosit bibliotecile Windows.h și MMSystem.h și am apelat funcția PlaySound la apăsarea tastei 1; argumentul funcției este fișierul cu extensia .wav care conține sunetul pe care am dorit să îl redau.
- Vizualizare **solid, wireframe și punctiform** cu funcția glPolygonMode

```
243     if (pressedKeys[GLFW_KEY_Z]) {
244         glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
245     }
246     if (pressedKeys[GLFW_KEY_X]) {
247         glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
248     }
249     if (pressedKeys[GLFW_KEY_C]) {
250         glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
251     }
```

iii. Motivarea abordării alese

Întrucât la laborator am acumulat cunoștințe despre majoritatea elementelor de bază din acest proiect, am ales o abordare care folosește cunoștințele deja însușite și care îmbină toate elementele învățate pe parcursul laboratoarelor.

b. Modelul grafic

Am optat pentru modelele prezente în laboratoare.

Astfel, pentru iluminarea scenei am folosit două surse de lumină care au la bază modelul de iluminare Phong.

Pentru ceață am folosit metoda exponențial-pătratică, ca să fie cât mai realistă.

c. Structuri de date

Pentru realizarea operațiilor necesare am folosit structurile de date disponibile în bilbioteca GLM, cum ar fi vec<n>, sau mat<n>, precum și structurile de date specifice OpenGL cum ar fi GLuint, GLFWwindow, etc.

d. Ierarhia de clase

Pentru funcționalități am inclus în proiect fișierele:

- Camera.hpp și Camera.cpp pentru mișcarea camerei
- Mesh.hpp și Mesh.cpp pentru definirea vârfurilor unui obiect
- Model3D.hpp și Model3D.cpp pentru crearea unui model 3D
- Shader.hpp și Shader.cpp pentru încărcarea shaderelor
- Skybox.hpp și Skybox.cpp pentru încărcarea Skybox-ului

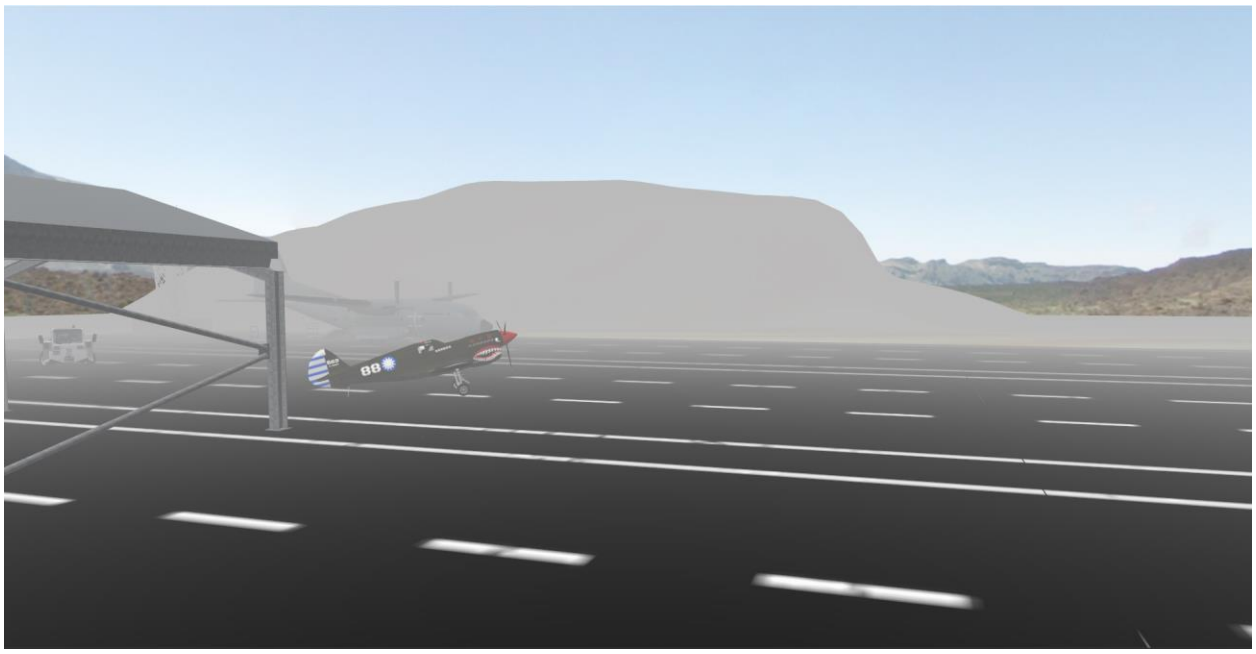
4. Prezentarea interfeței grafice (manual de utilizare)

Pentru funcționalități am asignat următoarele taste:

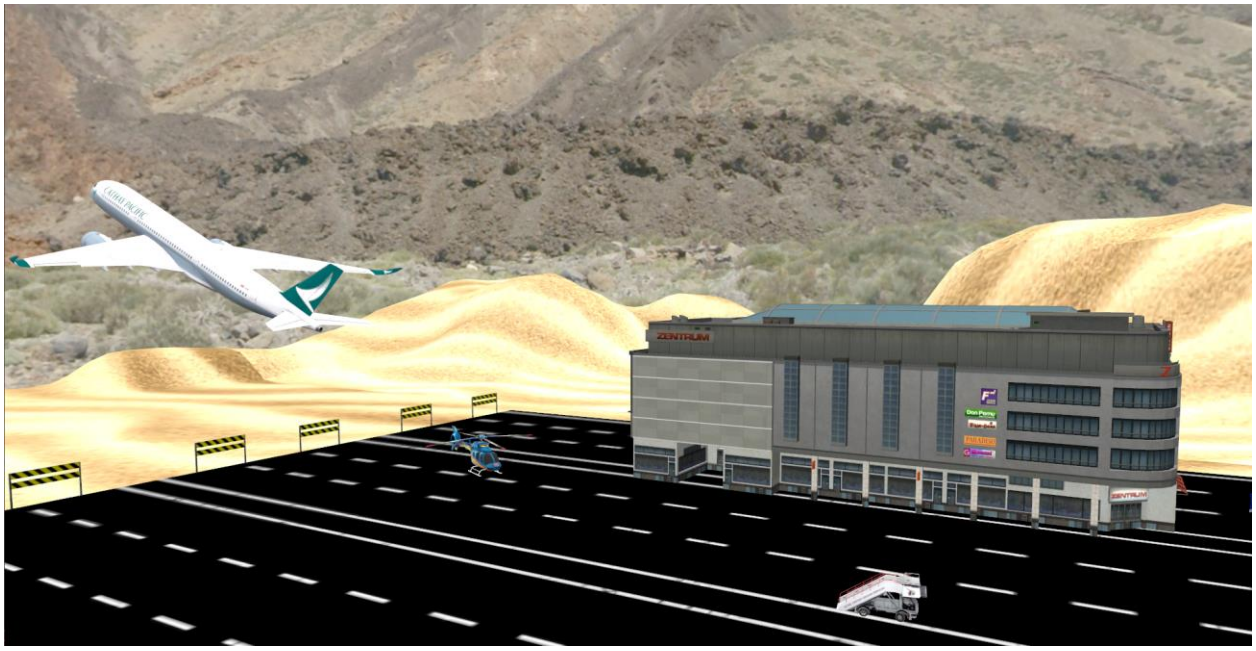
- **W** pentru mișcare înainte
- **S** pentru mișcare înapo
- **A** pentru mișcare stânga
- **D** pentru mișcare dreapta

- **E** pentru mișcare în sus
- **Q** pentru mișcare în jos
- **Săgeată sus** pentru rotirea camerei în sus
- **Săgeată jos** pentru rotirea camerei în jos
- **Săgeată dreapta** pentru rotirea camera la dreapta
- **Săgeată stânga** pentru rotirea camera la stânga
- **1**, **2** pentru pornirea, respectiv oprirea sunetului
- **3**, **4** pentru aprinderea, respectiv oprirea primei surse de lumină
- **5**, **6** pentru aprinderea, respectiv oprirea celei de-a doua surse de lumina
- **F**, **G** pentru activarea, respectiv dezactivarea efectului de ceață
- **Z, X** și **C** pentru modurile solid, wireframe respective punctiform

Scena inițială



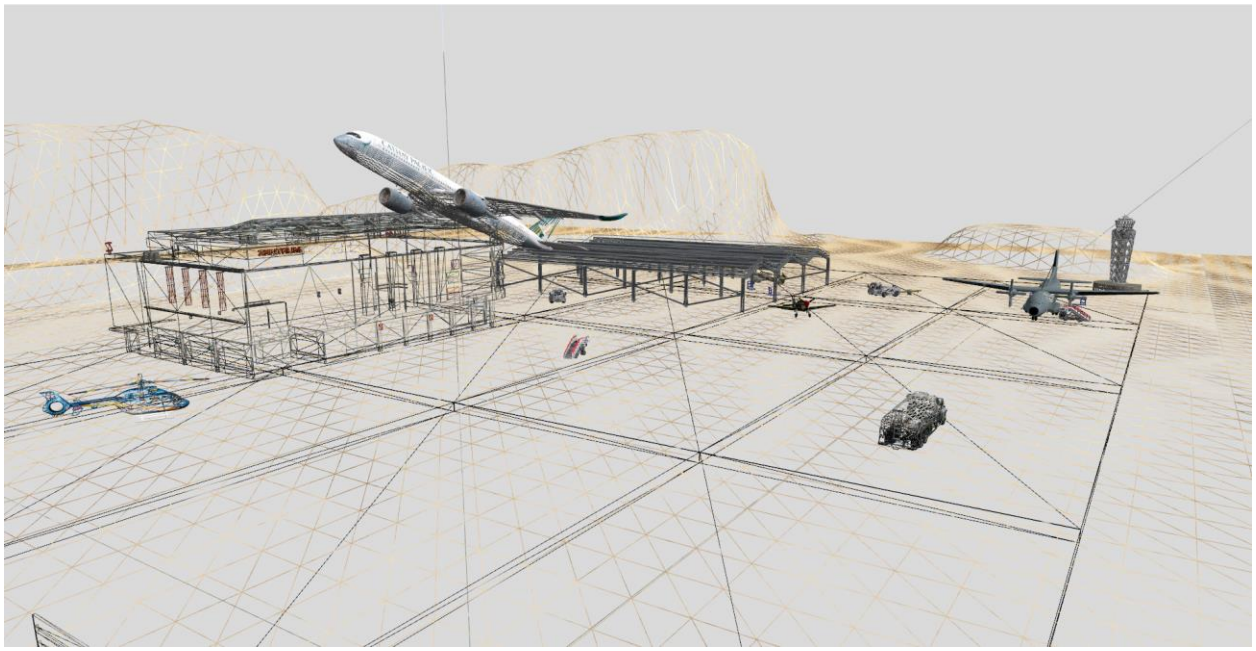
Ceață



Ambele lumini

Doar lumina 1

Doar lumina 2



Wireframe

Punctiform

5. Concluzii și dezvoltări ulterioare

Aplicația a fost creată pentru a oferi utilizatorului sentimentul de observator atotputernic, putând naviga prin scenă în orice punct dorește, schimbând condițiile meteorologice.

Dezvoltarea acestui proiect mi-a îmbunătățit abilitățile de a lucra în Blender și cunoștințele despre OpenGL, dar mi-a dezvoltat și spiritul de creație.

Ca și dezvoltări ulterioare, ar putea fi adăugate obiecte, animații, condiții meteo, sunete multiple, umbre și s-ar putea extinde scena.

6. Referințe

https://sketchfab.com/feed

https://free3d.com/

https://rigmodels.com/

https://stackoverflow.com/questions/8804880/use-playsound-in-c-opengl-to-play-sound-in-background

https://docs.google.com/document/d/1njtWPMmOQNIaD_z9ve8iPRUqQTWdIV_PO-NvPD0nOuM/edit

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