UNIVERSITATEA TEHNICĂ „GHEORGHE ASACHI" IAŞI
FACULTATEA AUTOMATICĂ ŞI CALCULATOARE
SPECIALIZAREA CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI
DISCIPLINA: Sisteme de prelucrare grafică

APLICAȚIE PENTRU RANDARE 3D

Student,
Iftime Adrian Dumitru, 1306B

IAŞI, 2021

1. Introducere

1.1. Scopul

Acest proiect are ca scop descrierea funcționalității aplicației pentru crearea și randarea de scene 3D. Funcționalitatea aplicației constă în alegerea obiectelor 3D, aplicarea de diferite operații asupra lor (precum scalarea și rotirea) și pozitionarea lor intr-o anumita locație din scena creata.

Motivul din spatele proiectului este de a crea o aplicație care ușurează munca persoanelor care doresc să construiasca scene atragatoare folosind obiecte 3D și puține cunoștiinte în domeniu pentru a putea face obiectele să aiba caracteristicile dorite și pentru a putea surprinde diferite cadre ale acestei scene.

1.2. Referințe

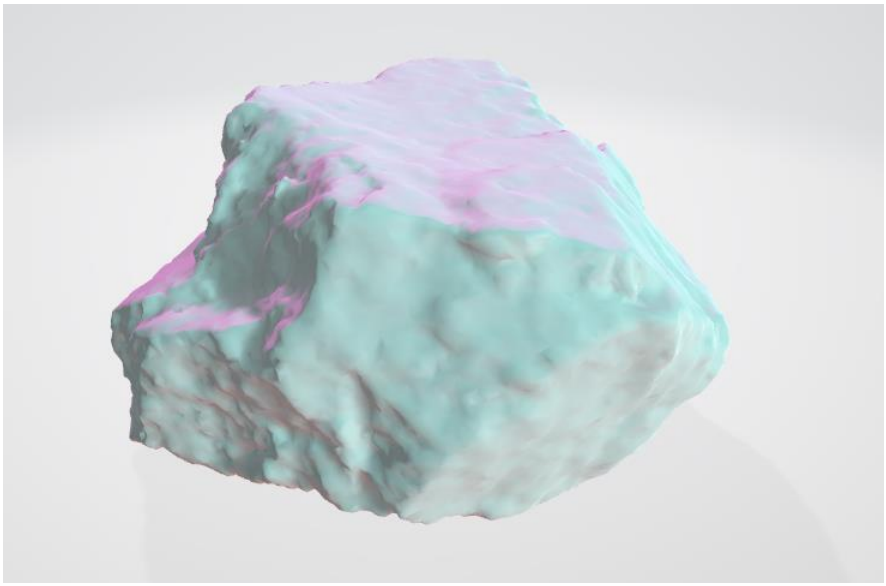
Pentru a realiza aplicația aceasta, am folosit urmatoarele surse de unde am procurat obiectele și pentru a mă documenta despre diferitele funcționalități ale librariei OpenGL

- https://sketchfab.com/
- Cursurile și laboratoarele predate în cadrul materiei Sisteme de prelucrare grafică
- https://edu.tuiasi.ro/course/view.php?id=454

2. Descriere

Această aplicație se foloseste de bibliotecile OpenGL, de fișierele objloader și de fisiere de tipul .obj pentru a pune cap la cap diferite scene dinamice, cu obiecte care pot fi modificate după plac și cu care utilizatorul poate să interactioneze având posibilitatea de a schimba originea sistemului de observare, pozitia sursei de lumină, și să rotească obiectele.

Totul incepe cu alegerea obiectelor care vor apărea in scenă.



Odată ce aceste obiecte au fost selectate, caracteristile acestora vor fi introduse in tablouri unidimensionale (vertices, uvs, normals) pentru a putea fi incărcate in bufferul grafic.

```
res[0] = loadOBJ("obj/caprioara.obj", vertices[0], uvs[0], normals[0]);
res[1] = loadOBJ("obj/padure.obj", vertices[1], uvs[1], normals[1]);
res[2]=loadOBJ("obj/terrain.obj", vertices[2], uvs[2], normals[2]);
res[3]= loadOBJ("obj/lup.obj", vertices[3], uvs[3], normals[3]);
res[4]= loadOBJ("obj/cerb.obj", vertices[4], uvs[4], normals[4]);
res[5] = loadOBJ("obj/pestera.obj", vertices[5], uvs[5], normals[5]);
res[6]= loadOBJ("obj/buturuga.obj", vertices[6], uvs[6], normals[6]);
```

```
for (int i = 0; i <= NrTexturi; i++)
{
    verticesNormals[i] = vertices[i];

    verticesNormals[i].insert(verticesNormals[i].end(), normals[i].begin(), normals[i].end());

    glGenBuffers(1, &vboObj[i]);
    glBindBuffer(GL_ARRAY_BUFFER, vboObj[i]);
    glBufferData(GL_ARRAY_BUFFER, verticesNormals[i].size() * sizeof(glm::vec3), verticesNormals[i].data(), GL_STATIC_DRAW);

    glGenVertexArrays(1, &vaoObj[i]);
    glBindVertexArray(vaoObj[i]);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), NULL);

    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)(3 * vertices[i].size() * sizeof(float)));
}
```

Pentru a face codul mai usor de urmărit și înteles, s-au folosit aceste tablouri iar caracteristicile au fost introduse prin intermediul pozitiei lor in acest tablou.

Odată ce caracteristicile obiectului au fost obținute, programul va incepe afișarea scenei creând obiectul după specificațiile introduse de către utilizator prin intermediul tablourilor precum:

- tabloul axisRotAngle care ajută la specificarea direcție inițiale spre care este prezentat obiectul;
- tabloul translate, in care sunt introduse coordonatele poziției inițiale ale obiectului;
- tabloul rotate, care dictează pe ce direcție se va roti obiectul odată ce utilizatorul apasă tastele specifice;
- tabloul scale, care este utilizat pentru a modifica dimensiunile obiectului;
- tabloul culoare, in care sunt introduse valorile RGB prin intermediul cărora vor fi afișate culorile obiectului
- tabloul replici, in care poate fi observat numărul de obiecte create după aceeasi formă.

```
float axisRotAngle[100] = {
                          PI/2.0,
                          PI/2.0,
                          PI/16.0,
                          PI / 16.0,
                          PI / 16.0,
                          PI / 16.0,
                          PI/16.0,
                          PI/16.0,
                          PI*1.5,
                          PI * 1.5,
                          PI * 1.5,
                          PI/2.0,
                          PI / 2,
                          PI / 2.0
                          };
float radius = 2;
float scaleFactor = 0.01;

glm::vec3 translate[100] = {
                          glm::vec3(-450, 0, 0),
                          glm::vec3(-300, 0, 200),
                          glm::vec3(-500, 0, -1300),
                          glm::vec3(500, 0, -1000),
                          glm::vec3(-1700 ,0, 150),
                          glm::vec3(-2100 ,0, 400),
                          glm::vec3(-1000 ,0, -1500),
                          glm::vec3(6000, 0, -6000),
                          glm::vec3(600, 0, 0),
                          glm::vec3(550, 0, 50),
                          glm::vec3(550, 0, -100),
                          glm::vec3(-270, 0, 50),
                          glm::vec3(2100 ,0, -400),
                          glm::vec3(0, 0, -500)
                          };
```

```
glm::vec3 rotate[100] = {
                          glm::vec3(0, 1, 0),
                          glm::vec3(0, 1, 0),
                          glm::vec3(0, 1, 0),
                          glm::vec3(0, 1, 0),
                          glm::vec3(0, 1, 0),
                          glm::vec3(0, 1, 0),
                          glm::vec3(0, 1, 0),
                          glm::vec3(0, 1, 0),
                          glm::vec3(0, 1, 0),
                          glm::vec3(0, 1, 0),
                          glm::vec3(0, 1, 0),
                          glm::vec3(0, 1, 0),
                          glm::vec3(0, 1, 0),
                          glm::vec3(0, 1, 0)
                          };
glm::vec3 scale[100] = {
                          glm::vec3(1.5, 1.5, 1.5) ,
                          glm::vec3(2, 2, 2),
                          glm::vec3(3,  3,  3),
                          glm::vec3(3,  3,  3),
                          glm::vec3(3,  3,  3),
                          glm::vec3(3,  3,  3),
                          glm::vec3(3,  3,  3),
                          glm::vec3(10, 1, 10),
                          glm::vec3(2, 2, 2),
                          glm::vec3(1.5, 1.5, 1.5),
                          glm::vec3(1.5, 1.5, 1.5),
                          glm::vec3(2, 2, 2),
                          glm::vec3(3,  3,  3),
                          glm::vec3(3,  3,  3)
                          };
```

```
glm::vec3 culoare[100] = {
                          glm::vec3(0.8470, 0.3647, 0.3657) ,
                          glm::vec3(0.8470, 0.3647, 0.3657) ,
                          glm::vec3(0.6470, 0.1647, 0.1657) ,
                          glm::vec3(0.6470, 0.1647, 0.1657) ,
                          glm::vec3(0.6470, 0.1647, 0.1657) ,
                          glm::vec3(0.13, 0.54, 0.13) ,
                          glm::vec3(0.13, 0.54, 0.13) ,
                          glm::vec3(0.25, 0.32, 0.14),
                          glm::vec3(0.8, 0.8, 0.8),
                          glm::vec3(0, 0, 0),
                          glm::vec3(0, 0, 0),
                          glm::vec3(0.6470, 0.1647, 0.1657),
                          glm::vec3(0.8, 0.8, 0.8),
                          glm::vec3(0.6470, 0.1647, 0.1657)
                          };
float replici[100] = {
                        2,
                        5,
                        1,
                        3,
                        1,
                        1,
                        1
                        };
```

Scopul funcției display este de a afișa obiectele. În interiorul acesteia se află o structură de tipul while care precum și la introducerea obiectelor, ajută la scrierea codului într-un mod lizibil. Aici sunt trimise caracterisiticile fiecarui obiect catre fragment.frag pentru a putea fi modificat obiectul curent pixel cu pixel. De asemenea, a fost folosit tabloul replici pentru a putea economisi spațiu. Astfel în loc să reintroducem de mai multe ori aceeași textură, în cazul obiectelor care apar de mai multe ori vom folosi valorile din tabloul replici pentru a putea ști de câte ori apare fiecare obiect.

```
int indexTextura = 0;
int indexObiect = 0;
int nrReplici = 0;
int NrObiecte = 0;
for (int i = 0; i < NrTexturi; i++)
{
    NrObiecte += replici[i];
}
while(indexTextura<NrTexturi)
{
    nrReplici = replici[indexTextura];
    while (nrReplici != 0)
    {
        nrReplici--;
        glBindVertexArray(vaoObj[indexTextura]);

        GLuint lightPosLoc = glGetUniformLocation(shader_programme, "lightPos");
        glUniform3fv(lightPosLoc, 1, glm::value_ptr(lightPos));

        GLuint viewPosLoc = glGetUniformLocation(shader_programme, "viewPos");
        glUniform3fv(viewPosLoc, 1, glm::value_ptr(viewPos));

        GLuint personajCuloare = glGetUniformLocation(shader_programme, "culoare");
        glUniform3fv(personajCuloare, 1, glm::value_ptr(culoare[indexObiect]));

        modelMatrix = modelStack.top();
        modelMatrix *= glm::translate(translate[indexObiect] * glm::vec3(scaleFactor, scaleFactor, scaleFactor));
        modelMatrix *= glm::rotate(axisRotAngle[indexObiect], rotate[indexObiect]);
        modelMatrix *= glm::scale(scale[indexObiect] * glm::vec3(scaleFactor, scaleFactor, scaleFactor));
        GLuint modelMatrixLoc = glGetUniformLocation(shader_programme, "modelViewProjectionMatrix");
        glUniformMatrix4fv(modelMatrixLoc, 1, GL_FALSE, glm::value_ptr(projectionMatrix * viewMatrix * modelMatrix));

        glm::mat4 normalMatrix = glm::transpose(glm::inverse(modelMatrix));
        GLuint normalMatrixLoc = glGetUniformLocation(shader programme, "normalMatrix");

         glDrawArrays(GL_TRIANGLES, 0, vertices[indexTextura].size());
         indexObiect++;
    }
    indexTextura++;
}

glutSwapBuffers();
```

Pentru a putea construi imagini cât mai realiste, s-a folosit funcția de iluminare lighting care modifică aspectul obiectelor dând impresia existentei unei surse de lumină adevărate

```
vec3 lighting(vec3 objectColor, vec3 pos, vec3 normal, vec3 lightPos, vec3 viewPos,
                vec3 ambient, vec3 lightColor, vec3 specular, float specPower)
{
    vec3 L=normalize(lightPos-pos);
    vec3 N=normalize(normal);
    vec3 V=normalize(viewPos-pos);
    vec3 R=reflect(-L,N);


    float diffCoef = max(0,dot(L,N));
    float specCoef = pow(max(0,dot(R,V)),specPower);



    vec3 ambientColor = ambient * lightColor;
    vec3 diffuseColor = diffCoef * lightColor;
    vec3 specularColor = specCoef * specular * lightColor;
    vec3 col = ( ambientColor + diffuseColor + specularColor ) * objectColor;

    return clamp(col, 0, 1);
}
```

Pentru a da posibilitatea utilizatorului de a alege unghiul perfect al scenei, s-a folosit un keyboard listener. Astfel, prin intermediul tastelor a s d f, utilizatorul poate roti obiectele iar prin intermediul tastelor q w e r t y să modifice pozitia de observare cu totul. De asemenea, la apăsarea tastelor – și + se va produce o mărire/micșorare a obiectelor care din perspectiva utilizatorul va arata precum o apropiere/ indepărtare de acestea. Pentru a modifica poziția (coordonaatele x respectiv z) ale unor obiecte, utilizatorul poate apăsa tastele g h j k.