Echipa formată din : Andronic Smaranda, Coriciuc Tudor, Dinu Andreea, Tănase Daniel

Grupa 461

PROIECT 3D – SISTEM SOLAR

Proiectul ales de echipa noastră vizează implementarea în 3D a unui sistem solar personalizat, care conține ca element de unicitate o pisică în locul unei planete, realizată cu ajutorul programului Blender.

Pentru a face peisajul nostru să semene mai mult cu unul galactic, am adăugat stele pe cer, un număr de 500 distribuite și culorate aleator cu ajutorul unui VAO, ele fiind desenate cu GL_POINTS.

Implementare:

```
void CreateVAOStars() {

    const int numStars = 500;

    glm::vec3 starPositions[numStars];
    glm::vec3 starColors[numStars];
    GLushort starIndices[numStars];

    for (int i = 0; i < numStars; ++i) {
        // Assign random positions
        starPositions[i] = glm::vec3(
            (rand() % 1200 - 600), // X: [-600, 600]
            (rand() % 1000 - 500),  // Z: [-500, 500]
            (rand() % 500 - 250)  // Z: [-250, 250]
        );

        //Assign random colors
        starColors[i] = glm::vec3(
            static_cast<float>(rand()) / RAND_MAX,  // R
            static_cast<float>(rand()) / RAND_MAX,  // G
            static_cast<float>(rand()) / RAND_MAX   // B
        );
```

```
//for stars
codCol = 0;
glUniform1i(codColLocation, codCol);
glBindVertexArray(starVAO);
glUniformMatrix4fv(viewModelLocation, 1, GL_FALSE, glm::value_ptr(view)); // Matrice de vizualizare
glPointSize(5.0f); // Dimensiunea punctelor
glDrawArrays(GL_POINTS, 0, numStars);
glBindVertexArray(0);
```

Sistemul solar este realizat folosind transformări 3D și o stivă de matrice pentru a organiza și gestiona transformările pentru diferite componente ale scenei (Soare, planete, etc.). Mai întâi definim matricea de vizualizare (view) și proiecția.

view = glm::lookAt(obs, pctRef, vert); //matricea de vizuzlizare

projection = glm::infinitePerspective(fov, width / height, dNear);

glUniformMatrix4fv(projLocation, 1, GL_FALSE, &projection[0][0]);

După care, cu ajutorul matricei translateSystem controlăm poziția întregului sistem solar în spațiu. În acest caz, sistemul nu este translatat (centrat la origine).

translateSystem = glm::translate(glm::mat4(1.0f), glm::vec3(0.0, 0.0, 0.0));

Întrucât soarele este centrul sistemului soalar, el se rotește în jurul propriei axe.

rotateSun = glm::rotate(glm::mat4(1.0f), -(float)0.001 * timeElapsed, glm::vec3(0.0, 1.0, 0.0));

Pe de altă parte, restul planetelor suferă modificări ce vizează scalare, rotație și translație în jurul soarelui și rotație în jurul propriei axe. După ce matricea de transformare este calculată pentru fiecare obiect, se transmite la shader.

```
//4) Pentru planeta

mvStack.top() *= rotatePlanet;      // In varful stivei:  view * translateSystem * rPl
mvStack.top() *= translatePlanet;   // In varful stivei:  view * translateSystem * rPl * tPl
mvStack.top() *= rotatePlanetAxis;  // In varful stivei:  view * translateSystem * rPl * tPl * rPlAx
mvStack.top() *= scalePlanet;       // In varful stivei:  view * translateSystem * rPl * tPl * rPlAx * scPl

// Transmitere matrice de deplasare a planetei catre shader, apoi eliminare din varful stivei
glUniformMatrix4fv(viewModelLocation, 1, GL_FALSE, glm::value_ptr(mvStack.top()));

//  Desenarea propriu-zisa a obiectului 3D
codCol = 1;  // Regula de colorare este diferita;
glUniform1i(codColLocation, codCol);
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate fetele in care in stanga jos este Polul Nord
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr) * sizeof(GLushort)));
}
```

Totodată, pentru a realiza centura de asteroizi din jurul planetei Saturn, matricea obținută (asteroidRingTransform) include toate transformările aplicate până la poziția planetei, astfel încât asteroizii să urmeze planeta aferentă în rotația și translația sa.

```
//asteroids
glm::mat4 asteroidRingTransform = mvStack.top();
glUniformMatrix4fv(viewModelLocation, 1, GL_FALSE, glm::value_ptr(asteroidRingTransform));
codCol = 0;
glUniform1i(codColLocation, codCol);
glBindVertexArray(asteroidsVAO);
glPointSize(5.0f);
glDrawArrays(GL_POINTS, 0, numAsteroids);
glBindVertexArray(0);
```

```
void CreateVAOAsteroidRing() {
    const float radius = 100.0f;    // Radius of the ring
    const float offset = 5.0f;     // Random offset for natural look

    glm::vec3 asteroidPositions[numAsteroids];
    glm::vec3 asteroidColors[numAsteroids];
    float asteroidScales[numAsteroids]; // Array for random scales
    GLushort asteroidIndices[numAsteroids];

    // Generate circular positions and scales for the asteroid ring
    for (int i = 0; i < numAsteroids; ++i) {
        float angle = (float)i / (float)numAsteroids * 360.0f; // Spread evenly around the circle
        float displacement = (rand() % (int)(2 * offset * 100)) / 100.0f - offset; // Random offset
        float x = sin(glm::radians(angle)) * radius + displacement;
        displacement = (rand() % (int)(2 * offset * 100)) / 100.0f - offset;
        float y = displacement * 0.4f; // Smaller variation in height
        displacement = (rand() % (int)(2 * offset * 100)) / 100.0f - offset;
        float z = cos(glm::radians(angle)) * radius + displacement;

        asteroidPositions[i] = glm::vec3(x, y, z);

        // Assign random sizes (scale between 0.5 and 2.0)
        asteroidScales[i] = 0.5f + static_cast<float>(rand()) / (static_cast<float>(RAND_MAX / 1.5f));

        asteroidColors[i] = glm::vec3(1.0f, 1.0f, 1.0f); // Default white
        asteroidIndices[i] = static_cast<GLushort>(i);
    }
```

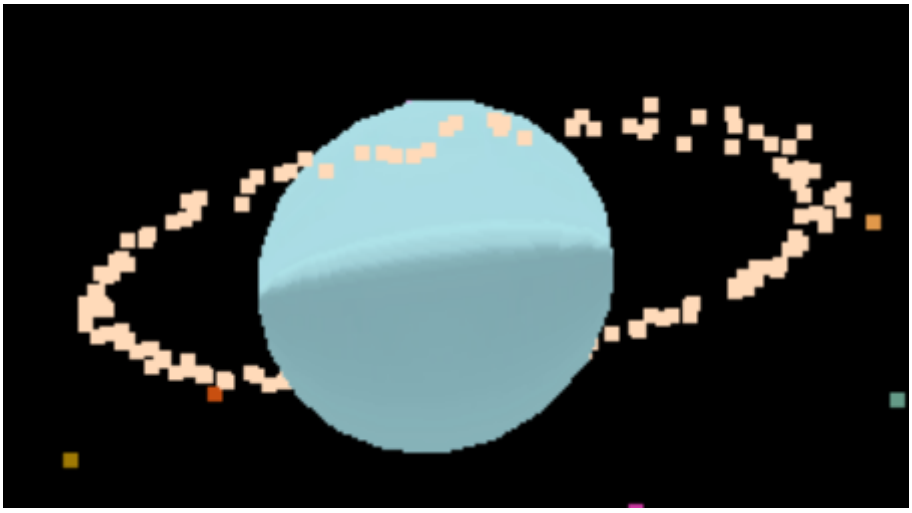
Astfel, pentru a genera locația celor cateva sute de asteroizi, se calculează unghiul pentru fiecare asteroid, împărțind uniform cei numAsteroids în jurul unui cerc complet (360 de grade).Fiecare asteroid primește un unghi unic (angle), astfel încât să fie distribuit uniform pe inel. Apoi, prin variabila displacement, adaugăm o variație aleatorie la pozițiile asteroizilor pentru un aspect mai natural și poziționăm asteroidul pe un cerc de rază radius. În plus, variația mică de pe axa Y are rolul de a crea o distribuție tridimensională (înălțime).

În funcția de Initialize() încărcăm fișierul .obj, care conține modelul cu pisica astronaut. Totodată, ne folosim și de VAO-ul corespunzător modelului.

```
void Initialize(void)
{
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);       //  Culoarea de fond a ecranului;
    bool model = loadOBJ("bigCat.obj", vertices, uvs, normals);

    nrVertices = vertices.size();

    CreateVAOCat();
    CreateVBOSfera();                           //  Trecerea datelor de randare spre bufferul folosit de shadere;
    CreateVAOStars();
    CreateVAOAsteroidRing();
    CreateShaders();                            //  Initilizarea shaderelor;
```

```
void CreateVAOCat() {
    // Crearea și legarea unui VAO
    glGenVertexArrays(1, &VaoIdCat);
    glBindVertexArray(VaoIdCat);

    // Crearea și umplerea unui VBO
    glGenBuffers(1, &VboIdCat);
    glBindBuffer(GL_ARRAY_BUFFER, VboIdCat);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(glm::vec3) + normals.size() * sizeof(glm::vec3), NULL, GL_STATIC_DRAW);
    glBufferSubData(GL_ARRAY_BUFFER, 0, vertices.size() * sizeof(glm::vec3), &vertices[0]);
    glBufferSubData(GL_ARRAY_BUFFER, vertices.size() * sizeof(glm::vec3), normals.size() * sizeof(glm::vec3), &normals[0]);

    // Configurarea atributelor de poziție și normale
    glEnableVertexAttribArray(0); // Poziție
    glVertexAttribPointer(0,  3, GL_FLOAT, GL_FALSE, 0, (GLvoid*)0);

    glEnableVertexAttribArray(1); // Normale
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, (GLvoid*)(vertices.size() * sizeof(glm::vec3)));

    glEnableVertexAttribArray(2); // culoare
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 0, (GLvoid*)(vertices.size() * sizeof(glm::vec3) + normals.size() * sizeof(glm::vec3)));

    // Debind VAO pentru a evita modificări neintenționate
    glBindVertexArray(0);
}
```

Rezultatul adaugării asteroizilor:



În ceea ce privește texturile aplicate planetelor, acestea sunt încărcate folosind biblioteca **stb_image.h** și sunt utilizate în program astfel:

- Funcția LoadTexture încarcă imaginea, creează un obiect de textură OpenGL (GLuint textureID) și setează parametrii necesari pentru texturare (GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T, GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER).
- Texturile sunt legate la unități de textură (ex. GL_TEXTURE0, GL_TEXTURE1, etc.) și transmise shaderului utilizând variabile uniforme (ex. moonTexture, jupiterTexture).

În RenderFunction:

- Texturile sunt încărcate (LoadTexture este apelată pentru fiecare textură: moon.jpg, galben.jpg, uranus.jpg, earth_clouds.jpg).
- Fiecare textură este activată cu glActiveTexture și legată cu glBindTexture.
- Variabilele uniforme din shader sunt setate cu glUniformli pentru a asocia unitățile de textură cu texturile specifice.

Implementarea celor menționate:

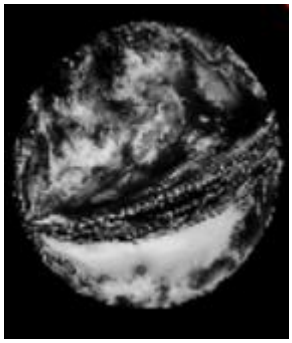
```
GLuint LoadTexture(const char* filePath) {
    int width, height, nrChannels;
    unsigned char* data = stbi_load(filePath, &width, &height, &nrChannels, 0);
    GLuint textureID;
    glGenTextures(1, &textureID);
    glBindTexture(GL_TEXTURE_2D, textureID);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
    glGenerateMipmap(GL_TEXTURE_2D);
    stbi_image_free(data);
    return textureID;
```

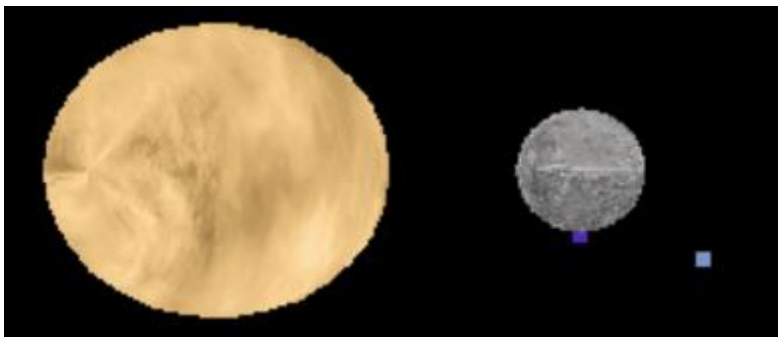
```
//texturize
GLuint textureMoon = LoadTexture("moon.jpg");
//GLuint textureJupiter = LoadTexture("jupiter.jpg");
GLuint textureJupiter = LoadTexture("galben.jpg");
GLuint textureUranus = LoadTexture("uranus.jpg");
GLuint textureEarth = LoadTexture("earth_clouds.jpg");
```

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, textureMoon);
// Set parameters for the second texture
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glUniform1i(glGetUniformLocation(ProgramId, "moonTexture"), 0);
```

```
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, textureJupiter);
// Set parameters for the second texture
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glUniform1i(glGetUniformLocation(ProgramId, "jupiterTexture"), 1);

glActiveTexture(GL_TEXTURE2);
glBindTexture(GL_TEXTURE_2D, textureUranus);
glUniform1i(glGetUniformLocation(ProgramId, "uranusTexture"), 2);

glActiveTexture(GL_TEXTURE3);
glBindTexture(GL_TEXTURE_2D, textureEarth);
// Set parameters for the second texture
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glUniform1i(glGetUniformLocation(ProgramId, "earthTexture"), 3);
```

Rezultatul în urma aplicării texturilor:





Resurse folosite:
Laboratoare

https://github.com/nothings/stb

https://www.programmingcreatively.com/opengl-tutorial-4-qs.php