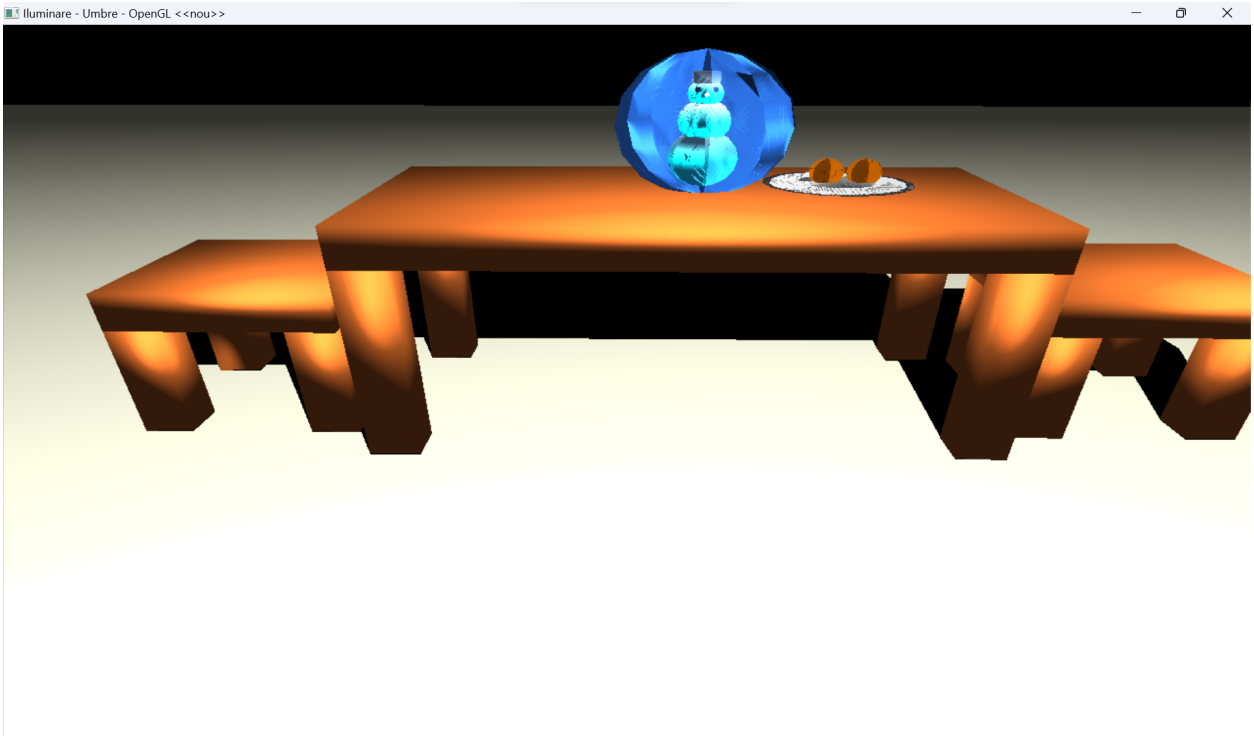
Proiect 3D

- Conceptul Proiectului
  - *Ideea Inițială*
  - *Procesul de Dezvoltare*
- Elementele Incluse
- Implementare
  - Masa
  - Scaunele
  - Omul de Zapada
    - Corpul
    - Ochii
    - Nasul
    - Tichie
  - Globul de Zapada
  - Farfuria
  - Portocalele
- Alcătuirea echipei
  - Contributii individuale
- Cod sursa



Conceptul Proiectului

Proiectul nostru are ca obiectiv principal crearea unei scene 3D, având în centrul atenției un glob de zăpadă.

Ideea Inițială

- **Globul de Zăpadă:** Totul a pornit de la încercarea de a integra un om de zăpadă în scena noastră 3D, iar de aici a luat naștere ideea unui glob de zăpadă, care va deveni punctul focal al scenei.

Procesul de Dezvoltare

- **Extinderea Scenei:** Odată stabilit globul de zăpadă drept punct focal, am continuat să extindem scena prin adăugarea altor obiecte. Astfel, masa, scaunele și farfuria cu portocale au fost integrate pentru a completa scena și a oferi atmosfera festivă.

Elementele Incluse

- **Globul de Zăpadă:** Globul în centrul căruia se află un om de zăpadă reprezintă elementul central al scenei.

- **Masa și Scaunele:** Pentru a completa atmosfera și pentru a oferi un context mai amplu, am inclus masa și scaunele în scena 3D.
- **Farfuria cu Portocale:** Pentru a adăuga detalii și a contribui la atmosfera festivă, am introdus o farfurie pe care se află două portocale.

Toate aceste elemente contribuie la ilustrarea tehnicilor învățate la laborator: prezentarea obiectelor 3D, iluminare, umbre, amestecare.

Implementare

În implementarea proiectului, am pornit de la codul sursă din laboratorul 11, 11_01_umbra.cpp, și aici am adăugat toate elementele necesare pentru a aduce scena 3D la viață.

Masa

Definirea geometriei: Masa este formată din paralelipipede, prin urmare, în `CreateVBO()`, în `Vertices`, am definit coordonatele punctelor pentru masă și picioarele acesteia.

```
// varfuri cub
    -50.0f,  -100.0f, 100.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,
     80.0f,  -100.0f,  100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
     80.0f,  150.0f,  100.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0f,
    -50.0f,  150.0f, 100.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0f,
    -50.0f,  -100.0f, 120.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0f,
     80.0f,  -100.0f,  120.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0f,
     80.0f,  150.0f,  120.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
    -50.0f,  150.0f, 120.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,

    //picior masa stang spate
    -50.0f,  -100.0f, 0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,
     -30.0f,  -100.0f,  0.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
     -30.0f,  -80.0f,  0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0f,
    -50.0f,  -80.0f, 0.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0f,
    -50.0f,  -100.0f, 100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0f,
     -30.0f,  -100.0f,  100.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0f,
```

```
 -30.0f,  -80.0f,  100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f, 1
-50.0f,  -80.0f, 100.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f, 1

//picior masa stang fata
60.0f,  -100.0f, 0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f, -1
 80.0f,  -100.0f,  0.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f, -1
 80.0f,  -80.0f,  0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0f, 1.0
60.0f,  -80.0f, 0.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0f, 1.0
60.0f,  -100.0f, 100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0f, -1
 80.0f,  -100.0f,  100.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0f, -1
 80.0f,  -80.0f,  100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f, 1
60.0f,  -80.0f, 100.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f, 1

//picior masa dreapta fata
60.0f,  130.0f, 0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f, -1.0
 80.0f,  130.0f,  0.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f, -1.0
 80.0f,  150.0f,  0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0f, 1.0
60.0f,  150.0f, 0.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0f, 1.0
60.0f,  130.0f, 100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0f, -1
 80.0f,  130.0f,  100.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0f, -1
 80.0f,  150.0f,  100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f, 1
60.0f,  150.0f, 100.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f, 1

//picior masa dreapta spate
-50.0f,  130.0f, 0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f, -1
 -30.0f,  130.0f,  0.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f, -1
 -30.0f,  150.0f,  0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0f, 1
-50.0f,  150.0f, 0.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0f, 1
-50.0f,  130.0f, 100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0f, -1
 -30.0f,  130.0f,  100.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0f, -1
 -30.0f,  150.0f,  100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f, 1
-50.0f,  150.0f, 100.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f, 1
```

Indicii pentru Vârfuri: În `Indices`, am specificat modul în care vârfurile sunt conectate pentru a forma fețele.

```
// fetele cubului
     5, 6, 4,   6, 4, 7,
     6, 7, 10, 10, 7, 11,
     11, 7, 8,   8, 7, 4,
     8, 4, 9,   9, 4, 5,
     5, 6, 9,   9, 6, 10,
     9, 10, 8,  8, 10, 11,
     //picior stang spate
     13, 14, 12,   14, 12, 15,
     14, 15, 18, 18, 15, 19,
     19, 15, 16,   16, 15, 12,
     16, 12, 17,   17, 12, 13,
     13, 14, 17,   17, 14, 18,
     17, 18, 16,  16, 18, 19,
     //picior stang fata
     21, 22, 20,   22, 20, 23,
     22, 23, 26, 26, 23, 27,
     27, 23, 24,   24, 23, 20,
     24, 20, 25,   25, 20, 21,
     21, 22, 25,   25, 22, 26,
     25, 26, 24,  24, 26, 27,
     //picior dreapta fata
    29, 30, 28,   30, 28, 31,
    30, 31, 34,   34, 31, 35,
    35, 31, 32,   32, 31, 28,
    32, 28, 33,   33, 28, 29,
    29, 30, 33,   33, 30, 34,
    33, 34, 32,   32, 34, 35,

    //picior dreapta fata
37, 38, 36,   38, 36, 39,
38, 39, 42,   42, 39, 43,
43, 39, 40,   40, 39, 36,
40, 36, 41,   41, 36, 37,
```

```
37, 38, 41,   41, 38, 42,
41, 42, 40,   40, 42, 43,
```

Renderizarea Mesei: În funcția de desenare `RenderFunction()`, desenăm fiecare parte a mesei prin apeluri separate la `glDrawElements`.

```
// desenare cub
glBindVertexArray(VaoId);
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, 0);
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(6));

//desenare picior stang spate
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(42))

//desenare picior stang fata
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(78))

//desenare picior drept fata
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(114)
```

```
//desenare picior drept fata
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(150))
```

Iar apoi, pentru fiecare formă geometrică construită, am adăugat umbre:

```
// desenare umbra cub
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(6));


// desenare umbra picior stang spate
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(42));

// desenare umbra picior stang fata
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(78));

// desenare umbra picior drept fata
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
```

```
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(114)

// desenare umbra picior drept spate
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(150)
```

Scaunele

Pentru scaune procesul a fost similar.

Definirea geometriei: Mai întâi am definit în `CreateVBO()`, în `Vertices`, coordonatele fiecărei forme geometrice care va alcătui scaunul.

```
// varfuri scaun 1
       -30.0f,  -200.0f, 60.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f
        60.0f,  -200.0f,  60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
        60.0f,  -110.0f,  60.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0f
       -30.0f,  -110.0f, 60.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0
       -30.0f,  -200.0f, 80.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0f,
        60.0f,  -200.0f,  80.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0f,
        60.0f,  -110.0f,  80.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
       -30.0f,  -110.0f, 80.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f

       //picior scaun stang spate
       -10.0f,  -200.0f, 0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,
        -30.0f,  -200.0f,  0.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
        -30.0f,  -180.0f,  0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0f
       -10.0f,  -180.0f, 0.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0f
       -10.0f,  -200.0f, 60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0f,
        -30.0f,  -200.0f,  60.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0f,
        -30.0f,  -180.0f,  60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f
       -10.0f,  -180.0f, 60.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f
```

```
//picior masa stang fata
40.0f,  -130.0f, 0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,
 60.0f,  -130.0f,  0.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
 60.0f,  -110.0f,  0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0f,
40.0f,  -110.0f, 0.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0f,
40.0f,  -130.0f, 60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0f,
 60.0f,  -130.0f,  60.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0f,
 60.0f,  -110.0f,  60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
40.0f,  -110.0f, 60.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,

//picior masa dreapta fata
-30.0f,  -130.0f, 0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,
 -10.0f,  -130.0f,  0.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
 -10.0f,  -110.0f,  0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0
-30.0f,  -110.0f, 0.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0
-30.0f,  -130.0f, 60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0f,
 -10.0f,  -130.0f,  60.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0f,
 -10.0f,  -110.0f,  60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0
-30.0f,  -110.0f, 60.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0

//picior masa dreapta spate
40.0f,  -200.0f, 0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,
 60.0f,  -200.0f,  0.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
 60.0f,  -180.0f,  0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0f,
40.0f,  -180.0f, 0.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0f,
40.0f,  -200.0f, 60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0f,
 60.0f,  -200.0f,  60.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0f,
 60.0f,  -180.0f,  60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
40.0f,  -180.0f, 60.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,

// SCAUN 2
     -30.0f, 250.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0
     60.0f, 250.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
     60.0f, 160.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
     -30.0f, 160.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0
```

```
            -30.0f, 250.0f, 80.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0
            60.0f, 250.0f, 80.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            60.0f, 160.0f, 80.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            -30.0f, 160.0f, 80.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0

            //picior scaun stang spate
            -10.0f, 160.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
            -30.0f, 160.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            -30.0f, 180.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            -10.0f, 180.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
            -10.0f, 160.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0
            -30.0f, 160.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            -30.0f, 180.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            -10.0f, 180.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0

            //picior masa stang fata
            40.0f, 230.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
            60.0f, 230.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            60.0f, 250.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            40.0f, 250.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
            40.0f, 230.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
            60.0f, 230.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            60.0f, 250.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            40.0f, 250.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,

            //picior masa dreapta fata
            -30.0f, 230.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
            -10.0f, 230.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            -10.0f, 250.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            -30.0f, 250.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
            -30.0f, 230.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0
            -10.0f, 230.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            -10.0f, 250.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            -30.0f, 250.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0

            //picior masa dreapta spate
```

```
            40.0f, 160.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
            60.0f, 160.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            60.0f, 180.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            40.0f, 180.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
            40.0f, 160.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
            60.0f, 160.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            60.0f, 180.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            40.0f, 180.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
```

Indicii pentru Varfuri: In `Indices` am specificat modul în care vârfurile sunt conectate pentru a forma fețele.

```
//fata scaun 1
        53, 54, 52,   54, 52, 55,
        54, 55, 58,   58, 55, 59,
        59, 55, 56,   56, 55, 52,
        56, 52, 57,   57, 52, 53,
        53, 54, 57,   57, 54, 58,
        57, 58, 56,   56, 58, 59,
        //picior scaun 1
61, 62, 60,   62, 60, 63,
62, 63, 66,   66, 63, 67,
67, 63, 64,   64, 63, 60,
64, 60, 65,   65, 60, 61,
61, 62, 65,   65, 62, 66,
65, 66, 64,   64, 66, 67,
//picior scaun 1
        69, 70, 68,  70, 68, 71,
        70, 71, 74,  74, 71, 75,
        75, 71, 72,  72, 71, 68,
        72, 68, 73,  73, 68, 69,
        69, 70, 73,  73, 70, 74,
        73, 74, 72,  72, 74, 75,
        //picior scaun 1
        77, 78, 76,   78, 76, 79,
        78, 79, 82,   82, 79, 83,
```

```
        83, 79, 80,   80, 79, 76,
        80, 76, 81,   81, 76, 77,
        77, 78, 81,   81, 78, 82,
        81, 82, 80,   80, 82, 83,
        //picior scaun 1
        85, 86, 84,   86, 84, 87,
        86, 87, 90,   90, 87, 91,
        91, 87, 88,   88, 87, 84,
        88, 84, 89,   89, 84, 85,
        85, 86, 89,   89, 86, 90,
        89, 90, 88,   88, 90, 91,

        //SCAUN 2
        //fata scaun
        93, 94, 92,   94, 92, 95,
        94, 95, 98,   98, 95, 99,
        99, 95, 96,   96, 95, 92,
        96, 92, 97,   97, 92, 93,
        93, 94, 97,   97, 94, 98,
        97, 98, 96,   96, 98, 99,
        //picior scaun
        101, 102, 100,   102, 100, 103,
        102, 103, 106,   106, 103, 107,
        107, 103, 104,   104, 103, 100,
        104, 100, 105,   105, 100, 101,
        101, 102, 105,   105, 102, 106,
        105, 106, 104,   104, 106, 107,
        //picior scaun
        109, 110, 108,   110, 108, 111,
        110, 111, 114,   114, 111, 115,
        115, 111, 112,   112, 111, 108,
  112, 108, 113,   113, 108, 109,
        109, 110, 113,   113, 110, 114,
        113, 114, 112,   112, 114, 115,
        //picior scaun
        117, 118, 116,   118, 116, 119,
```

```
        118, 119, 122,   122, 119, 123,
        123, 119, 120,   120, 119, 116,
        120, 116, 121,   121, 116, 117,
        117, 118, 121,   121, 118, 122,
        121, 122, 120,   120, 122, 123,
        //picior scaun
        125, 126, 124,   126, 124, 127,
        126, 127, 130,   130, 127, 131,
        131, 127, 128,   128, 127, 124,
        128, 124, 129,   129, 124, 125,
        125, 126, 129,   129, 126, 130,
        129, 130, 128, 128, 130, 131
```

Renderizarea Scaunelor si Adaugarea Umbrelor: Infunctia de desenare `RenderFunction()` desenam fiecare parte a fiecarui scaun prin apeluri separte la `glDrawElements` .

```
// desenare fata fata scaun1
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(222)

// desenare umbra fata scaun1
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(222)

// desenare picior scaun1
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
```

```
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(258)

// desenare umbra picior scaun1
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(258)

// desenare picior scaun1
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(294)

// desenare umbra picior scaun1
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(294)

// desenare picior scaun1
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(330)

// desenare umbra picior scaun1
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(330)
```

```
// desenare picior scaun1
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(366)

// desenare umbra picior scaun1
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(366)


//SCAUN2
// desenare fata fata scaun1
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(402)

// desenare umbra fata scaun1
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(402)

// desenare picior scaun1
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
```

```
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(438)

// desenare umbra picior scaun1
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(438)

// desenare picior scaun1
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(474)

// desenare umbra picior scaun1
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(474)

// desenare picior scaun1
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(510)

// desenare umbra picior scaun1
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(510)
```

```
// desenare picior scaun1
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(546)

// desenare umbra picior scaun1
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(546)
```

Omul de Zapada

Omul de zapda este alcatuit din sfere. Fiecare sfera este definita de un set de vertecsi, culori si indici.

Corpul

Pentru inceput am definit functia `CreareaVAO1` care se ocupa de crearea VAO-ului pentru una dintre sferele care vor alcatui omul de zapada.

- Aici se genereaza si se completeaza datele necesare pentru omul de zapada in matricele `Vertices1`, `Colors1`, și `Indices1`.

- Apoi se creeaza si se leaga VAO-ul (`VaoId1`).
- Se generează și leagă bufferul de atribute (`VboId1`) și bufferul de indici (`EboId1`).
- Se definesc atributele si pointerii pentru acestea (pozitia, culoarea si normalele).
- Se incarca datele in buffere

```
//creare om-de-zapada

void CreateVAO1(void)
{
    // SFERA
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices1[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors1[(NR_PARR + 1) * NR_MERID];
    GLushort Indices1[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PAR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * step_u; // valori pentru u
            float v = V_MIN + merid * step_v;
            float x_vf = radius * cosf(u) * cosf(v); // coordona
            float y_vf = radius * cosf(u) * sinf(v);
            float z_vf = radius * sinf(u);

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices1[index] = glm::vec4(x_vf * 2 , y_vf*2 +30.0
            Colors1[index] = glm::vec3(0.0f + sinf(u), 1.0f, 1.0
            Indices1[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelelo
            index_aux = parr * (NR_MERID)+merid;
```

```
            Indices1[(NR_PARR + 1) * NR_MERID + index_aux] = inc

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta j
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices1[AUX + 4 * index] = index1;  // unele va
                Indices1[AUX + 4 * index + 1] = index2;
                Indices1[AUX + 4 * index + 2] = index3;
                Indices1[AUX + 4 * index + 3] = index4;
            }
        }
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId1);
    glBindVertexArray(VaoId1);
    glGenBuffers(1, &VboId1); // atribute
    glGenBuffers(1, &EboId1); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId1);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices1) + sizeof(Col
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices1), Verti
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices1), sizeof(C
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId1);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices1), Indi
```

```
    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 12 * sizeof(
}
```

Desenarea omului de zăpadă presupune desenarea sferelor ce alcătuiesc omul de zăpadă folosind apeluri la `glDrawElements` și adăugarea umbrelor acestora. Astfel, am luat pe rând fiecare sferă începând cu prima:

```
// SFERA
glBindVertexArray(VaoId1);
codCol = 0;
glUniform1i(codColLocation, codCol);
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}

//desenare umbra
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
```

```
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}
```

Am urmat pașii descriși mai sus pentru următoarele sfere care intră în alcătuirea omului de zăpadă.

Am creat `CreateVAO3` :

```
void CreateVAO3(void)
{
    // SFERA
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices1[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors1[(NR_PARR + 1) * NR_MERID];
    GLushort Indices1[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PAR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * step_u; // valori pentru u
            float v = V_MIN + merid * step_v;
            float x_vf = radius * cosf(u) * cosf(v); // coordona
            float y_vf = radius * cosf(u) * sinf(v);
            float z_vf = radius * sinf(u);

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices1[index] = glm::vec4((x_vf * 3) / 2, y_vf *
            Colors1[index] = glm::vec3(0.0f + sinf(u), 1.0f, 1.0
            Indices1[index] = index;
```

```
            // indice ptr acelasi varf la parcurgerea paralelelo
            index_aux = parr * (NR_MERID)+merid;
            Indices1[(NR_PARR + 1) * NR_MERID + index_aux] = ind

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices1[AUX + 4 * index] = index1;  // unele va
                Indices1[AUX + 4 * index + 1] = index2;
                Indices1[AUX + 4 * index + 2] = index3;
                Indices1[AUX + 4 * index + 3] = index4;
            }
        }
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId3);
    glBindVertexArray(VaoId3);
    glGenBuffers(1, &VboId3); // atribute
    glGenBuffers(1, &EboId3); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId3);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices1) + sizeof(Col
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices1), Verti
```

```
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices1), sizeof(C
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId3);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices1), Ind

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(C
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(

}
```

Am desenat și această sferă apelând `glDrawElements` și i-am adăugat și umbra:

```
// SFERA 2
glBindVertexArray(VaoId3);
codCol = 0;
glUniform1i(codColLocation, codCol);
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}
//desenare umbra
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
```

```
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}
```

Similar, pentru a treia sfera am creat `CreateVAO4` :

```
void CreateVAO4(void)
{
    // SFERA
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices1[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors1[(NR_PARR + 1) * NR_MERID];
    GLushort Indices1[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PAR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * step_u; // valori pentru u
            float v = V_MIN + merid * step_v;
            float x_vf = radius * cosf(u) * cosf(v); // coordona
            float y_vf = radius * cosf(u) * sinf(v);
            float z_vf = radius * sinf(u);

            // identificator ptr varf; coordonate + culoar  vedi
            index = merid * (NR_PARR + 1) + parr;
            Vertices1[index] = glm::vec4(x_vf, y_vf + 30.0f, z_v
            Colors1[index] = glm::vec3(0.0f + sinf(u), 1.0f, 1.0
            Indices1[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelel
```

```
            index_aux = parr * (NR_MERID)+merid;
            Indices1[(NR_PARR + 1) * NR_MERID + index_aux] = ind

            // indicii pentru desenarea fetelor, pentru varfurem
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices1[AUX + 4 * index] = index1;  // unele va
                Indices1[AUX + 4 * index + 1] = index2;
                Indices1[AUX + 4 * index + 2] = index3;
                Indices1[AUX + 4 * index + 3] = index4;
            }
        }
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId4);
    glBindVertexArray(VaoId4);
    glGenBuffers(1, &VboId4); // atribute
    glGenBuffers(1, &EboId4); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId4);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices1) + sizeof(Col
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices1), Verti
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices1), sizeof(C
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId4);
```

```
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices1), Ind

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(
}
```

Și am desenat și această sferă apelând `glDrawElements`, iar apoi i-am adăugat umbră:

```
// SFERA
glBindVertexArray(VaoId4);
codCol = 0;
glUniform1i(codColLocation, codCol);
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}

///desenare umbra
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
```

```
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}
```

Ochii

Procesul de a crea ochii nu este cu mult diferit, întrucât și aceștia reprezintă două sfere, fiecare cu VAO-ul propriu `CreateVAO6`, respectiv `CreateVAO7`.

```
void CreateVAO6(void)
{
    // SFERA
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices1[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors1[(NR_PARR + 1) * NR_MERID];
    GLushort Indices1[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PAR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * step_u; // valori pentru u
            float v = V_MIN + merid * step_v;
            float x_vf = radius * cosf(u) * cosf(v); // coordona
            float y_vf = radius * cosf(u) * sinf(v);
            float z_vf = radius * sinf(u);

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices1[index] = glm::vec4(x_vf / 5 + 5.0f, y_vf /
            Colors1[index] = glm::vec3(0.0f, 0.0f, 0.0f);
            Indices1[index] = index;
```

```
            // indice ptr acelasi varf la parcurgerea paralelelo
            index_aux = parr * (NR_MERID)+merid;
            Indices1[(NR_PARR + 1) * NR_MERID + index_aux] = ind

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices1[AUX + 4 * index] = index1;  // unele va
                Indices1[AUX + 4 * index + 1] = index2;
                Indices1[AUX + 4 * index + 2] = index3;
                Indices1[AUX + 4 * index + 3] = index4;
            }
        }
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId6);
    glBindVertexArray(VaoId6);
    glGenBuffers(1, &VboId6); // atribute
    glGenBuffers(1, &EboId6); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId6);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices1) + sizeof(Col
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices1), Verti
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices1), sizeof(C
```

```
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId6);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices1), Ind

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(

}
```

```
//ochi
void CreateVAO7(void)
{
    // SFERA
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices1[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors1[(NR_PARR + 1) * NR_MERID];
    GLushort Indices1[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PAR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * step_u; // valori pentru u
            float v = V_MIN + merid * step_v;
            float x_vf = radius * cosf(u) * cosf(v); // coordona
            float y_vf = radius * cosf(u) * sinf(v);
            float z_vf = radius * sinf(u);

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
```

```
            Vertices1[index] = glm::vec4(x_vf/5 + 5.0f, y_vf/5+3
            Colors1[index] = glm::vec3(0.0f, 0.0f, 0.0f);
            Indices1[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelel
            index_aux = parr * (NR_MERID)+merid;
            Indices1[(NR_PARR + 1) * NR_MERID + index_aux] = ind

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices1[AUX + 4 * index] = index1;  // unele va
                Indices1[AUX + 4 * index + 1] = index2;
                Indices1[AUX + 4 * index + 2] = index3;
                Indices1[AUX + 4 * index + 3] = index4;
            }
        }
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId7);
    glBindVertexArray(VaoId7);
    glGenBuffers(1, &VboId7); // atribute
    glGenBuffers(1, &EboId7); // indici

    // legare+"incarcare" buffer
```

```
    glBindBuffer(GL_ARRAY_BUFFER, VboId7);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices1) + sizeof(Col
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices1), Verti
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices1), sizeof(C
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId7);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices1), Indi

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(C
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(
}
```

Apoi, în funcția `RenderFunction`, ca și până acum, am desenat cele două sfere și am adăugat umbrele corespunzătoare.

```
// SFERA -ochi stang
glBindVertexArray(VaoId6);
codCol = 0;
glUniform1i(codColLocation, codCol);
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}

// SFERA -ochi drept
glBindVertexArray(VaoId7);
```

```
codCol = 0;
glUniform1i(codColLocation, codCol);
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}
```

Nasul

Orice om de zăpadă are nevoie de un nas. În cazul nostru, nasul este reprezentat de un con. Procesul de creare al conului nu este cu mult diferit de pașii prezentati până acum.

Așadar, am creat funcția `CreateVAO5`:

```
void CreateVAO5(void)
{
    // CONUL
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices5[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors5[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Normals5[(NR_PARR + 1) * NR_MERID];
    GLushort Indices5[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PAR

    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * 2 * step_u; // valori pentr
            float v = V_MIN + merid * step_v;
            float x_vf = -4.0f * v; // coordonatele varfului cor
```

```
            float y_vf = v * sin(u);
            float z_vf = v * cos(u);

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices5[index] = glm::vec4(x_vf/4 +11.0f, y_vf/4+
            Colors5[index] = glm::vec3(1.0f, 0.5f, 0.0f);
            Normals5[index] = glm::vec3(x_vf / 4 + 11.0f, y_vf /
            Indices5[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelelo
            index_aux = parr * (NR_MERID)+merid;
            Indices5[(NR_PARR + 1) * NR_MERID + index_aux] = ind

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices5[AUX + 4 * index] = index1;  // unele va
                Indices5[AUX + 4 * index + 1] = index2;
                Indices5[AUX + 4 * index + 2] = index3;
                Indices5[AUX + 4 * index + 3] = index4;
            }
        }
    };
```

```
    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId5);
    glBindVertexArray(VaoId5);
    glGenBuffers(1, &VboId5); // atribute
    glGenBuffers(1, &EboId5); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId5);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices5) + sizeof(Col
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices5), Verti
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices5), sizeof(C
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId5);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices5), Indi

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(C

    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(

}
```

Iar apoi in `RenderFunction` am desenat conul si i-am adaugat umbra:

Iar apoi, în `RenderFunction`, am desenat conul și i-am adăugat umbra:

```
// CONUL2
glBindVertexArray(VaoId5);
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0
```

```
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}
//desenare umbra
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(

for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}
```

Tichie

Tichia omului de zăpadă este un cub, așadar, pentru a-l crea, am scris funcția `CreateVAO2`, în care am definit coordonatele punctelor care alcătuiesc cubul (`Vertices2`), culorile fiecărui punct (`Colors2`) și ordinea în care acestea sunt parcurse (`Indices2`).

```
void CreateVAO2(void)
{
    // CUBUL
    //
    GLfloat Vertices2[] =
```

```
    {
        -5.0f, 25.0f, 170.0f, 1.0f,
         5.0f, 25.0f, 170.0f, 1.0f,
         5.0f,  35.0f, 170.0f, 1.0f,
        -5.0f,  35.0f, 170.0f, 1.0f,
        -5.0f, 25.0f, 175.0f, 1.0f,
         5.0f, 25.0f, 175.0f, 1.0f,
         5.0f,  35.0f, 175.0f, 1.0f,
        -5.0f,  35.0f, 175.0f, 1.0f
    };

    GLfloat Colors2[] =
    {
        0.8f, 0.8f, 0.8f,
        0.7f, 0.7f, 0.7f,
        0.6f, 0.6f, 0.6f,
        0.5f, 0.5f, 0.5f,
        0.4f, 0.4f, 0.4f,
        0.3f, 0.3f, 0.3f,
        0.2f, 0.2f, 0.2f,
        0.1f, 0.1f, 0.1f
    };

    GLushort Indices2[] =
    {
      1, 2, 0,   2, 0, 3,
      2, 3, 6,   6, 3, 7,
      7, 3, 4,   4, 3, 0,
      4, 0, 5,   5, 0, 1,
      1, 2, 5,   5, 2, 6,
      5, 6, 4,   4, 6, 7,
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId2);
    glBindVertexArray(VaoId2);
```

```
    glGenBuffers(1, &VboId2); // atribute
    glGenBuffers(1, &EboId2); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId2);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices2) + sizeof(Co
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices2), Vert
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices2), sizeof(
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId2);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices2), Ind

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(
}
```

În continuare, am desenat cubul apelând funcția `glDrawElements` în `RenderFunction` și umbra acestuia:

```
// cub-tichie om-de-zapada
glBindVertexArray(VaoId2);
codCol = 0;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][0
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_SHORT, (GLvoid*)(0)

codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
```

```
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_SHORT, (GLvoid*)(0)
```

Globul de Zapada

Pentru a reprezenta globul transparent în care stă omul de zăpadă, am implementat funcția `CreateVAO8`.

```
// glob transparent
void CreateVAO8(void)
{
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices8[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors8[(NR_PARR + 1) * NR_MERID];
    GLushort Indices8[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PAR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * step_u; // valori pentru u
            float v = V_MIN + merid * step_v;
            float x_vf = radius * cosf(u) * cosf(v); // coordona
            float y_vf = radius * cosf(u) * sinf(v);
            float z_vf = radius * sinf(u);

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices8[index] = glm::vec4(x_vf * 5, y_vf * 5 + 30
            Colors8[index] = glm::vec3(0.2f, 0.5f, 1.0f);
            Indices8[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelel
            index_aux = parr * (NR_MERID)+merid;
            Indices8[(NR_PARR + 1) * NR_MERID + index_aux] = ind
```

```
            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices8[AUX + 4 * index] = index1;  // unele va
                Indices8[AUX + 4 * index + 1] = index2;
                Indices8[AUX + 4 * index + 2] = index3;
                Indices8[AUX + 4 * index + 3] = index4;
            }
        }
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId8);
    glBindVertexArray(VaoId8);
    glGenBuffers(1, &VboId8); // atribute
    glGenBuffers(1, &EboId8); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId8);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices8) + sizeof(Col
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices8), Verti
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices8), sizeof(C
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId8);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices8), Indi
```

```
    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE,  40 * sizeof
}
```

Apoi, în cadrul funcției `RenderFunction`, se combină globul transparent cu scenele existente, lăsând omul de zăpadă și umbra să fie vizibile sub glob, datorită efectului de transparență aplicat globului.

Așadar, aici folosim funcția de amestecare `glBlendFunc(GL_SRC_ALPHA, GL_SRC_ALPHA);` pentru a oferi iluzia de transparență a globului.

```
// glob
glEnable(GL_BLEND);
//glDepthMask(GL_FALSE);
glBlendFunc(GL_SRC_ALPHA, GL_SRC_ALPHA);
glBindVertexArray(VaoId8);
codCol = 0;
glUniform1i(codColLocation, codCol);
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}
glDepthMask(GL_TRUE);
glDisable(GL_BLEND);
```

```
//desenare umbra
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}
```

Farfuria

In ceea ce priveste farfuria, geometria farfuriei este definită ca un con foarte turnat cu vârful în jos, iar vârful este acoperit de masă. Pentru a crea acest con am definit functia `CreateVAO12` :

```
// farfurie
void CreateVAO12(void)
{
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors[(NR_PARR + 1) * NR_MERID];
    GLushort Indices[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PARR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN +  parr * step_u; // valori pentru u
            float v = V_MIN +  merid * step_v;
```

```
            float x_vf = v * cosf(u); // coordonatele varfului c
            float y_vf = v * sinf(u);
            float z_vf = - 0.05 * v;

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices[index] = glm::vec4(x_vf * 5, y_vf * 5 + 85
            Colors[index] = glm::vec3(1.0f, 1.0f, 1.0f);
            Indices[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelel
            index_aux = parr * (NR_MERID)+merid;
            Indices[(NR_PARR + 1) * NR_MERID + index_aux] = inde

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices[AUX + 4 * index] = index1;  // unele val
                Indices[AUX + 4 * index + 1] = index2;
                Indices[AUX + 4 * index + 2] = index3;
                Indices[AUX + 4 * index + 3] = index4;
            }
        }
    };

    // generare VAO/buffere
```

```
    glGenVertexArrays(1, &VaoId12);
    glBindVertexArray(VaoId12);
    glGenBuffers(1, &VboId12); // atribute
    glGenBuffers(1, &EboId12); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId12);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices) + sizeof(Colo
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices), Vertic
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices), sizeof(Co
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId12);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indic

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(G
}
```

In `RenderFunction` este desenată farfuria și umbra acesteia:

```
glBindVertexArray(VaoId12);
codCol = 0;
glUniform1i(codColLocation, codCol);
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
```

```
}
glDepthMask(GL_TRUE);
glDisable(GL_BLEND);

//desenare umbra
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}
```

Portocalele

Prortocalele sunt reprezentate de două sfere portocalii. Pentru a le reprezenta am definit `CreateVAO10` și `CreateVAO11`.

```
//portocala 1
void CreateVAO10(void)
{
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors[(NR_PARR + 1) * NR_MERID];
    GLushort Indices[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PARR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
```

```
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * step_u; // valori pentru u
            float v = V_MIN + merid * step_v;
            float x_vf = radius * cosf(u) * cosf(v); // coordona
            float y_vf = radius * cosf(u) * sinf(v);
            float z_vf = radius * sinf(u);

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices[index] = glm::vec4(x_vf, y_vf + 80.0f, z_vf
            Colors[index] = glm::vec3(1.0f, 0.5f, 0.0f);
            Indices[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelelo
            index_aux = parr * (NR_MERID)+merid;
            Indices[(NR_PARR + 1) * NR_MERID + index_aux] = inde

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices[AUX + 4 * index] = index1;  // unele val
                Indices[AUX + 4 * index + 1] = index2;
                Indices[AUX + 4 * index + 2] = index3;
                Indices[AUX + 4 * index + 3] = index4;
            }
        }
```

```
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId10);
    glBindVertexArray(VaoId10);
    glGenBuffers(1, &VboId10); // atribute
    glGenBuffers(1, &EboId10); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId10);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices) + sizeof(Colo
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices), Vertic
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices), sizeof(Co
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId10);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indic

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 12 * sizeof(
}
```

```
// portocala 2
void CreateVAO11(void)
{
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors[(NR_PARR + 1) * NR_MERID];
    GLushort Indices[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PARR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
```

```
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * step_u; // valori pentru u
            float v = V_MIN + merid * step_v;
            float x_vf = radius * cosf(u) * cosf(v); // coordona
            float y_vf = radius * cosf(u) * sinf(v);
            float z_vf = radius * sinf(u);

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices[index] = glm::vec4(x_vf, y_vf + 95.0f, z_vf
            Colors[index] = glm::vec3(1.0f, 0.5f, 0.0f);
            Indices[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelelo
            index_aux = parr * (NR_MERID)+merid;
            Indices[(NR_PARR + 1) * NR_MERID + index_aux] = inde

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices[AUX + 4 * index] = index1;  // unele val
                Indices[AUX + 4 * index + 1] = index2;
                Indices[AUX + 4 * index + 2] = index3;
                Indices[AUX + 4 * index + 3] = index4;
```

```
            }
        }
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId11);
    glBindVertexArray(VaoId11);
    glGenBuffers(1, &VboId11); // atribute
    glGenBuffers(1, &EboId11); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId11);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices) + sizeof(Colo
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices), Vertic
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices), sizeof(Co
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId11);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indic

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 12 * sizeof(
}
```

Iar apoi, în `RenderFunction`, am desenat cele două portocale și umbrele lor:

```
// portocala 1

glBindVertexArray(VaoId10);
codCol = 0;
glUniform1i(codColLocation, codCol);
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
```

```
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}
glDepthMask(GL_TRUE);
glDisable(GL_BLEND);

//desenare umbra
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}

// portocala 2

glBindVertexArray(VaoId11);
codCol = 0;
glUniform1i(codColLocation, codCol);
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
```

```
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}
glDepthMask(GL_TRUE);
glDisable(GL_BLEND);

//desenare umbra
codCol = 1;
glUniform1i(codColLocation, codCol);
myMatrix = glm::mat4(1.0f);
glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix[0][(
for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
{
    if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt considerate
        glDrawElements(
            GL_QUADS,
            4,
            GL_UNSIGNED_SHORT,
            (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * patr)
}
```

Alcătuirea echipei

- Băicoianu Bianca, grupa 351
- Buruiană Cosmina, grupa 332
- Georgescu Miruna, grupa 332
- Neaga Maria, grupa 332

Contributii individuale

1. **Băicoianu Bianca (Grupa 351):**
    - Contribuții la structura generală a proiectului.

- Implementarea și documentarea omului de zăpadă.
- Descrierea și documentarea procesului de amestecare pentru obiectele transparente.

2. **Buruiană Cosmina (Grupa 332):**
   - Implementarea și documentarea funcționalității farfuriei.
   - Descrierea și documentarea procesului de creare și desenare a scaunelor.
   - Contribuții la structura generală a documentației.
3. **Georgescu Miruna (Grupa 332):**
   - Implementarea și documentarea funcționalității mesei.
   - Descrierea și documentarea procesului de creare și desenare a portocalelor.
   - Contribuții la stilul și formatul documentației.
4. **Neaga Maria (Grupa 332):**
   - Implementarea și documentarea funcționalității obiectelor de tip sferă (ochi, nas).
   - Descrierea și documentarea procesului de creare și desenare a tichiei și a globului.
   - Contribuții la partea de introducere.

Cod sursa

```
//
// ================================================
// | Grafica pe calculator                        |
// ================================================
// | Laboratorul XI - 11_01_umbra.cpp |
// ====================================
//
//  Program ce deseneaza o casa si care suprinde efectele de umb
//  Elemente de NOUTATE (modelul de iluminare):
//   - pentru a genera umbra unei surse este utilizata o matrice
```

```
//   - in shaderul de varfuri este inclusa si matricea umbrei;
//   - in shaderul de fragment umbra este colorata separat;
//   - sursa de lumina este punctuala(varianta de sursa directio
//
//
//
//  Biblioteci
#include <windows.h>        //  Utilizarea functiilor de sistem
#include <stdlib.h>         //  Biblioteci necesare pentru citir
#include <stdio.h>
#include <math.h>           //  Biblioteca pentru calcule matema
#include <GL/glew.h>        //  Defineşte prototipurile functiil
#include <GL/freeglut.h>    //  Include functii pentru:
                            //  - gestionarea ferestrelor si eve
                            //  - desenarea de primitive grafice
                            //  - crearea de meniuri si submeniu
#include "loadShaders.h"    //  Fisierul care face legatura intr
#include "glm/glm.hpp"      //  Bibloteci utilizate pentru trans
#include "glm/gtc/matrix_transform.hpp"
#include "glm/gtx/transform.hpp"
#include "glm/gtc/type_ptr.hpp"
#include "SOIL.h"


//  Identificatorii obiectelor de tip OpenGL;
GLuint
VaoId1, VaoId2, VaoId3, VaoId4, VaoId5, VaoId6, VaoId7, VaoId8,
VboId1, VboId2, VboId3, VboId4, VboId5, VboId6, VboId7, VboId8,
EboId1, EboId2, EboId3, EboId4, EboId5, EboId6, EboId7, EboId8,
    VaoId,
    VboId,
    EboId,
    ColorBufferId,
    ProgramId,
    myMatrixLocation,
    matrUmbraLocation,
```

```
    viewLocation,
    projLocation,
    matrRotlLocation,
    lightColorLocation,
    lightPosLocation,
    viewPosLocation,
    codColLocation;

GLuint texture;

int codCol;
float PI = 3.141592;

// matrice utilizate
glm::mat4 myMatrix, matrRot;

// elemente pentru matricea de vizualizare
float Refx = 0.0f, Refy = 0.0f, Refz = 0.0f;
float alpha = PI / 8, beta = 0.0f, dist = 400.0f;
float Obsx, Obsy, Obsz;
float Vx = 0.0, Vy = 0.0, Vz = 1.0;
glm::mat4 view;

// elemente pentru matricea de proiectie
float width = 800, height = 600, xwmin = -800.f, xwmax = 800, yw
glm::mat4 projection;

// sursa de lumina
float xL = 500.f, yL = 100.f, zL = 400.f;

// matricea umbrei
float matrUmbra[4][4];

float const U_MIN = -PI / 2, U_MAX = 2 * PI, V_MIN = 0, V_MAX =
// (2) numarul de paralele/meridiane, de fapt numarul de valori
int const NR_PARR = 20, NR_MERID = 30;
```

```
// pasul cu care vom incrementa u, respectiv v
float step_u = (U_MAX - U_MIN) / NR_PARR, step_v = (V_MAX - V_MI

// alte variabile
float radius = 7;
int index, index_aux;

void processNormalKeys(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 'l':
        Vx -= 0.1;
        break;
    case 'r':
        Vx += 0.1;
        break;
    case '+':
        dist += 5;
        break;
    case '-':
        dist -= 5;
        break;
    }
    if (key == 27)
        exit(0);
}
void processSpecialKeys(int key, int xx, int yy)
{
    switch (key)
    {
    case GLUT_KEY_LEFT:
        beta -= 0.01;
        break;
    case GLUT_KEY_RIGHT:
        beta += 0.01;
```

```
        break;
    case GLUT_KEY_UP:
        alpha += 0.01;
        break;
    case GLUT_KEY_DOWN:
        alpha -= 0.01;
        break;
    }
}

//  Functia de incarcare a texturilor in program;
void LoadTexture(const char* photoPath)
{
    glGenTextures(1, &texture);
    glBindTexture(GL_TEXTURE_2D, texture);
    //  Desfasurarea imaginii pe orizonatala/verticala in functi
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEA
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEA

    int width, height;
    unsigned char* image = SOIL_load_image(photoPath, &width, &h
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_
    glGenerateMipmap(GL_TEXTURE_2D);

    SOIL_free_image_data(image);
    glBindTexture(GL_TEXTURE_2D, 0);
}

//creare om-de-zapada

void CreateVAO1(void)
{
    // SFERA
```

```
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices1[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors1[(NR_PARR + 1) * NR_MERID];
    GLushort Indices1[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PAR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * step_u; // valori pentru u
            float v = V_MIN + merid * step_v;
            float x_vf = radius * cosf(u) * cosf(v); // coordona
            float y_vf = radius * cosf(u) * sinf(v);
            float z_vf = radius * sinf(u);

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices1[index] = glm::vec4(x_vf * 2 , y_vf*2 +30.0
            Colors1[index] = glm::vec3(0.0f + sinf(u), 1.0f, 1.0
            Indices1[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelelo
            index_aux = parr * (NR_MERID)+merid;
            Indices1[(NR_PARR + 1) * NR_MERID + index_aux] = ind

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
```

```
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices1[AUX + 4 * index] = index1;  // unele va
                Indices1[AUX + 4 * index + 1] = index2;
                Indices1[AUX + 4 * index + 2] = index3;
                Indices1[AUX + 4 * index + 3] = index4;
            }
        }
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId1);
    glBindVertexArray(VaoId1);
    glGenBuffers(1, &VboId1); // atribute
    glGenBuffers(1, &EboId1); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId1);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices1) + sizeof(Co
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices1), Verti
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices1), sizeof(C
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId1);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices1), Indi

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 12 * sizeof(
}

void CreateVAO2(void)
{
```

```
    // CUBUL
    //
    GLfloat Vertices2[] =
    {
        -5.0f, 25.0f, 170.0f, 1.0f,
         5.0f, 25.0f, 170.0f, 1.0f,
         5.0f,  35.0f, 170.0f, 1.0f,
        -5.0f,  35.0f, 170.0f, 1.0f,
        -5.0f, 25.0f, 175.0f, 1.0f,
         5.0f, 25.0f, 175.0f, 1.0f,
         5.0f,  35.0f, 175.0f, 1.0f,
        -5.0f,  35.0f, 175.0f, 1.0f
    };

    GLfloat Colors2[] =
    {
        0.8f, 0.8f, 0.8f,
        0.7f, 0.7f, 0.7f,
        0.6f, 0.6f, 0.6f,
        0.5f, 0.5f, 0.5f,
        0.4f, 0.4f, 0.4f,
        0.3f, 0.3f, 0.3f,
        0.2f, 0.2f, 0.2f,
        0.1f, 0.1f, 0.1f
    };

    GLushort Indices2[] =
    {
      1, 2, 0,   2, 0, 3,
      2, 3, 6,   6, 3, 7,
      7, 3, 4,   4, 3, 0,
      4, 0, 5,   5, 0, 1,
      1, 2, 5,   5, 2, 6,
      5, 6, 4,   4, 6, 7,
    };
```

```
    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId2);
    glBindVertexArray(VaoId2);
    glGenBuffers(1, &VboId2); // atribute
    glGenBuffers(1, &EboId2); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId2);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices2) + sizeof(Co
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices2), Vert
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices2), sizeof(C
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId2);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices2), Ind

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(C
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(
}


void CreateVAO3(void)
{
    // SFERA
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices1[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors1[(NR_PARR + 1) * NR_MERID];
    GLushort Indices1[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PAR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
```

```
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * step_u; // valori pentru u
            float v = V_MIN + merid * step_v;
            float x_vf = radius * cosf(u) * cosf(v); // coordona
            float y_vf = radius * cosf(u) * sinf(v);
            float z_vf = radius * sinf(u);

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices1[index] = glm::vec4((x_vf * 3) / 2, y_vf *
            Colors1[index] = glm::vec3(0.0f + sinf(u), 1.0f, 1.0
            Indices1[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelelo
            index_aux = parr * (NR_MERID)+merid;
            Indices1[(NR_PARR + 1) * NR_MERID + index_aux] = ind

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices1[AUX + 4 * index] = index1;  // unele va
                Indices1[AUX + 4 * index + 1] = index2;
                Indices1[AUX + 4 * index + 2] = index3;
                Indices1[AUX + 4 * index + 3] = index4;
            }
        }
```

```
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId3);
    glBindVertexArray(VaoId3);
    glGenBuffers(1, &VboId3); // atribute
    glGenBuffers(1, &EboId3); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId3);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices1) + sizeof(Col
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices1), Verti
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices1), sizeof(C
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId3);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices1), Indi

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(

}


void CreateVAO4(void)
{
    // SFERA
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices1[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors1[(NR_PARR + 1) * NR_MERID];
    GLushort Indices1[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PAR
    for (int merid = 0; merid < NR_MERID; merid++)
```

```
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * step_u; // valori pentru u
            float v = V_MIN + merid * step_v;
            float x_vf = radius * cosf(u) * cosf(v); // coordona
            float y_vf = radius * cosf(u) * sinf(v);
            float z_vf = radius * sinf(u);

            // identificator ptr varf; coordonate + culoar  vedi
            index = merid * (NR_PARR + 1) + parr;
            Vertices1[index] = glm::vec4(x_vf, y_vf + 30.0f, z_v
            Colors1[index] = glm::vec3(0.0f + sinf(u), 1.0f, 1.0
            Indices1[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelelo
            index_aux = parr * (NR_MERID)+merid;
            Indices1[(NR_PARR + 1) * NR_MERID + index_aux] = ind

            // indicii pentru desenarea fetelor, pentru varfurem
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices1[AUX + 4 * index] = index1;  // unele va
                Indices1[AUX + 4 * index + 1] = index2;
                Indices1[AUX + 4 * index + 2] = index3;
```

```
                Indices1[AUX + 4 * index + 3] = index4;
            }
        }
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId4);
    glBindVertexArray(VaoId4);
    glGenBuffers(1, &VboId4); // atribute
    glGenBuffers(1, &EboId4); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId4);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices1) + sizeof(Co
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices1), Vert
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices1), sizeof(
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId4);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices1), Ind

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(
}




void CreateVAO6(void)
{
    // SFERA
    // Matricele pentru varfuri, culori, indici
```

```
    glm::vec4 Vertices1[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors1[(NR_PARR + 1) * NR_MERID];
    GLushort Indices1[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PAR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * step_u; // valori pentru u
            float v = V_MIN + merid * step_v;
            float x_vf = radius * cosf(u) * cosf(v); // coordona
            float y_vf = radius * cosf(u) * sinf(v);
            float z_vf = radius * sinf(u);

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices1[index] = glm::vec4(x_vf / 5 + 5.0f, y_vf /
            Colors1[index] = glm::vec3(0.0f, 0.0f, 0.0f);
            Indices1[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelel
            index_aux = parr * (NR_MERID)+merid;
            Indices1[(NR_PARR + 1) * NR_MERID + index_aux] = ind

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
```

```
                }
                Indices1[AUX + 4 * index] = index1;  // unele va
                Indices1[AUX + 4 * index + 1] = index2;
                Indices1[AUX + 4 * index + 2] = index3;
                Indices1[AUX + 4 * index + 3] = index4;
            }
        }
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId6);
    glBindVertexArray(VaoId6);
    glGenBuffers(1, &VboId6); // atribute
    glGenBuffers(1, &EboId6); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId6);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices1) + sizeof(Col
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices1), Verti
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices1), sizeof(C
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId6);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices1), Indi

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(

}

//ochi
void CreateVAO7(void)
```

```
{
    // SFERA
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices1[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors1[(NR_PARR + 1) * NR_MERID];
    GLushort Indices1[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PAR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * step_u; // valori pentru u
            float v = V_MIN + merid * step_v;
            float x_vf = radius * cosf(u) * cosf(v); // coordona
            float y_vf = radius * cosf(u) * sinf(v);
            float z_vf = radius * sinf(u);

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices1[index] = glm::vec4(x_vf/5 + 5.0f, y_vf/5+3
            Colors1[index] = glm::vec3(0.0f, 0.0f, 0.0f);
            Indices1[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelel
            index_aux = parr * (NR_MERID)+merid;
            Indices1[(NR_PARR + 1) * NR_MERID + index_aux] = ind

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
```

```
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices1[AUX + 4 * index] = index1;  // unele va
                Indices1[AUX + 4 * index + 1] = index2;
                Indices1[AUX + 4 * index + 2] = index3;
                Indices1[AUX + 4 * index + 3] = index4;
            }
        }
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId7);
    glBindVertexArray(VaoId7);
    glGenBuffers(1, &VboId7); // atribute
    glGenBuffers(1, &EboId7); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId7);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices1) + sizeof(Col
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices1), Verti
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices1), sizeof(C
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId7);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices1), Indi

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(
}
```

```
void CreateVAO5(void)
{
    // CONUL
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices5[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors5[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Normals5[(NR_PARR + 1) * NR_MERID];
    GLushort Indices5[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PAR

    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * 2 * step_u; // valori pentr
            float v = V_MIN + merid * step_v;
            float x_vf = -4.0f * v; // coordonatele varfului cor
            float y_vf = v * sin(u);
            float z_vf = v * cos(u);

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices5[index] = glm::vec4(x_vf/4 +11.0f, y_vf/4+
            Colors5[index] = glm::vec3(1.0f, 0.5f, 0.0f);
            Normals5[index] = glm::vec3(x_vf / 4 + 11.0f, y_vf /
            Indices5[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelelo
            index_aux = parr * (NR_MERID)+merid;
            Indices5[(NR_PARR + 1) * NR_MERID + index_aux] = ind

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
```

```
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices5[AUX + 4 * index] = index1;  // unele va
                Indices5[AUX + 4 * index + 1] = index2;
                Indices5[AUX + 4 * index + 2] = index3;
                Indices5[AUX + 4 * index + 3] = index4;
            }
        }
    };


    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId5);
    glBindVertexArray(VaoId5);
    glGenBuffers(1, &VboId5); // atribute
    glGenBuffers(1, &EboId5); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId5);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices5) + sizeof(Col
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices5), Verti
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices5), sizeof(C
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId5);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices5), Indi

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
```

```
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G

    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(

}


// glob transparent
void CreateVAO8(void)
{
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices8[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors8[(NR_PARR + 1) * NR_MERID];
    GLushort Indices8[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PAR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * step_u; // valori pentru u
            float v = V_MIN + merid * step_v;
            float x_vf = radius * cosf(u) * cosf(v); // coordona
            float y_vf = radius * cosf(u) * sinf(v);
            float z_vf = radius * sinf(u);

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices8[index] = glm::vec4(x_vf * 5, y_vf * 5 + 30
            Colors8[index] = glm::vec3(0.2f, 0.5f, 1.0f);
            Indices8[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelel
            index_aux = parr * (NR_MERID)+merid;
```

```
            Indices8[(NR_PARR + 1) * NR_MERID + index_aux] = ind

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta j
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices8[AUX + 4 * index] = index1;  // unele va
                Indices8[AUX + 4 * index + 1] = index2;
                Indices8[AUX + 4 * index + 2] = index3;
                Indices8[AUX + 4 * index + 3] = index4;
            }
        }
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId8);
    glBindVertexArray(VaoId8);
    glGenBuffers(1, &VboId8); // atribute
    glGenBuffers(1, &EboId8); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId8);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices8) + sizeof(Col
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices8), Verti
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices8), sizeof(C
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId8);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices8), Indi
```

```
    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE,  40 * sizeof
}

// farfurie
void CreateVAO12(void)
{
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors[(NR_PARR + 1) * NR_MERID];
    GLushort Indices[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PARR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN +  parr * step_u; // valori pentru u
            float v = V_MIN +  merid * step_v;
            float x_vf = v * cosf(u); // coordonatele varfului c
            float y_vf = v * sinf(u);
            float z_vf = - 0.05 * v;

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices[index] = glm::vec4(x_vf * 5, y_vf * 5 + 85
            Colors[index] = glm::vec3(1.0f, 1.0f, 1.0f);
            Indices[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelelo
```

```
            index_aux = parr * (NR_MERID)+merid;
            Indices[(NR_PARR + 1) * NR_MERID + index_aux] = inde

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices[AUX + 4 * index] = index1;  // unele val
                Indices[AUX + 4 * index + 1] = index2;
                Indices[AUX + 4 * index + 2] = index3;
                Indices[AUX + 4 * index + 3] = index4;
            }
        }
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId12);
    glBindVertexArray(VaoId12);
    glGenBuffers(1, &VboId12); // atribute
    glGenBuffers(1, &EboId12); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId12);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices) + sizeof(Colo
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices), Vertic
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices), sizeof(Co
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId12);
```

```
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indic

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(G
}


// portocala 1
void CreateVAO10(void)
{
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors[(NR_PARR + 1) * NR_MERID];
    GLushort Indices[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PARR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * step_u; // valori pentru u
            float v = V_MIN + merid * step_v;
            float x_vf = radius * cosf(u) * cosf(v); // coordona
            float y_vf = radius * cosf(u) * sinf(v);
            float z_vf = radius * sinf(u);

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices[index] = glm::vec4(x_vf, y_vf + 80.0f, z_vf
            Colors[index] = glm::vec3(1.0f, 0.5f, 0.0f);
            Indices[index] = index;
```

```
            // indice ptr acelasi varf la parcurgerea paralelel
            index_aux = parr * (NR_MERID)+merid;
            Indices[(NR_PARR + 1) * NR_MERID + index_aux] = inde

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices[AUX + 4 * index] = index1;  // unele val
                Indices[AUX + 4 * index + 1] = index2;
                Indices[AUX + 4 * index + 2] = index3;
                Indices[AUX + 4 * index + 3] = index4;
            }
        }
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId10);
    glBindVertexArray(VaoId10);
    glGenBuffers(1, &VboId10); // atribute
    glGenBuffers(1, &EboId10); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId10);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices) + sizeof(Colo
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices), Vertic
```

```
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices), sizeof(C
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId10);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indi

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 12 * sizeof(
}

// portocala 2
void CreateVAO11(void)
{
    // Matricele pentru varfuri, culori, indici
    glm::vec4 Vertices[(NR_PARR + 1) * NR_MERID];
    glm::vec3 Colors[(NR_PARR + 1) * NR_MERID];
    GLushort Indices[2 * (NR_PARR + 1) * NR_MERID + 4 * (NR_PARR
    for (int merid = 0; merid < NR_MERID; merid++)
    {
        for (int parr = 0; parr < NR_PARR + 1; parr++)
        {
            // implementarea reprezentarii parametrice
            float u = U_MIN + parr * step_u; // valori pentru u
            float v = V_MIN + merid * step_v;
            float x_vf = radius * cosf(u) * cosf(v); // coordona
            float y_vf = radius * cosf(u) * sinf(v);
            float z_vf = radius * sinf(u);

            // identificator ptr varf; coordonate + culoare + in
            index = merid * (NR_PARR + 1) + parr;
            Vertices[index] = glm::vec4(x_vf, y_vf + 95.0f, z_vf
            Colors[index] = glm::vec3(1.0f, 0.5f, 0.0f);
```

```
            Indices[index] = index;

            // indice ptr acelasi varf la parcurgerea paralelel
            index_aux = parr * (NR_MERID)+merid;
            Indices[(NR_PARR + 1) * NR_MERID + index_aux] = inde

            // indicii pentru desenarea fetelor, pentru varful c
            if ((parr + 1) % (NR_PARR + 1) != 0) // varful consi
            {
                int AUX = 2 * (NR_PARR + 1) * NR_MERID;
                int index1 = index; // varful v considerat
                int index2 = index + (NR_PARR + 1); // dreapta l
                int index3 = index2 + 1;  // dreapta sus fata de
                int index4 = index + 1;  // deasupra lui v, pe a
                if (merid == NR_MERID - 1)  // la ultimul meridi
                {
                    index2 = index2 % (NR_PARR + 1);
                    index3 = index3 % (NR_PARR + 1);
                }
                Indices[AUX + 4 * index] = index1;  // unele val
                Indices[AUX + 4 * index + 1] = index2;
                Indices[AUX + 4 * index + 2] = index3;
                Indices[AUX + 4 * index + 3] = index4;
            }
        }
    };

    // generare VAO/buffere
    glGenVertexArrays(1, &VaoId11);
    glBindVertexArray(VaoId11);
    glGenBuffers(1, &VboId11); // atribute
    glGenBuffers(1, &EboId11); // indici

    // legare+"incarcare" buffer
    glBindBuffer(GL_ARRAY_BUFFER, VboId11);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices) + sizeof(Colo
```

```
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(Vertices), Vertic
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(Vertices), sizeof(Co
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId11);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indic

    // atributele;
    glEnableVertexAttribArray(0); // atributul 0 = pozitie
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, (GLvoid*)
    glEnableVertexAttribArray(1); // atributul 1 = culoare
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(G
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 12 * sizeof(
}


void CreateVBO1(void)
{
    GLfloat Vertices[] = {
        // coordonate                    // culori           // n
            // varfuri "ground"
        -1500.0f, -1500.0f, 0.0f, 1.0f, 1.0f, 1.0f, 0.9f, 0.0f,
        1500.0f, -1500.0f, 0.0f, 1.0f, 1.0f, 1.0f, 0.9f, 0.0f, 0
        1500.0f, 1500.0f, 0.0f, 1.0f, 1.0f, 1.0f, 0.9f, 0.0f, 0
         -1500.0f, 1500.0f, 0.0f, 1.0f, 1.0f, 1.0f, 0.9f, 0.0f,
    };

    GLfloat Indices[] = {

        1, 2, 0,   2, 0, 3,
    };

    //  Transmiterea datelor prin buffere;

    //  Se creeaza / se leaga un VAO (Vertex Array Object) - uti
    glGenVertexArrays(1, &VaoId9);
```

```
    glBindVertexArray(VaoId9);

    //  Se creeaza un buffer pentru VARFURI - COORDONATE, CULORI
    glGenBuffers(1, &VboId9);
    glBindBuffer(GL_ARRAY_BUFFER, VboId9);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL

    //  Se creeaza un buffer pentru INDICI;
    glGenBuffers(1, &EboId9);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId9);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indic

    //  Se activeaza lucrul cu atribute;
    //  Se asociaza atributul (0 = coordonate) pentru shader;
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 10 * sizeof(
    //  Se asociaza atributul (1 =  culoare) pentru shader;
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(
    //  Se asociaza atributul (2 =  texturare) pentru shader;
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 10 * sizeof(
    //  Se asociaza atributul (2 =  texturare) pentru shader;
    glEnableVertexAttribArray(3);
    glVertexAttribPointer(3, 2, GL_FLOAT, GL_FALSE, 10 * sizeof(
}

void CreateVBO(void)
{
    // varfurile
    GLfloat Vertices[] =
    {
        // coordonate                    // culori           // n
        // varfuri "ground"
       -1500.0f,  -1500.0f, 0.0f, 1.0f,  1.0f, 1.0f, 0.9f,  0.0f
        1500.0f,  -1500.0f, 0.0f, 1.0f,  1.0f, 1.0f, 0.9f,  0.0f
```

```
         1500.0f,  1500.0f,  0.0f, 1.0f,  1.0f, 1.0f, 0.9f,  0.0
        -1500.0f,  1500.0f,  0.0f, 1.0f,  1.0f, 1.0f, 0.9f,  0.0
        // varfuri cub
         -50.0f,  -100.0f, 100.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1
          80.0f,  -100.0f,  100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0
          80.0f,  150.0f,  100.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0
         -50.0f,  150.0f, 100.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1
         -50.0f,  -100.0f, 120.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0
          80.0f,  -100.0f,  120.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0
          80.0f,  150.0f,  120.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0
         -50.0f,  150.0f, 120.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0

         //picior masa stang spate
         -50.0f,  -100.0f, 0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0
          -30.0f,  -100.0f,  0.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0
          -30.0f,  -80.0f,  0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0
         -50.0f,  -80.0f, 0.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0
         -50.0f,  -100.0f, 100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0
          -30.0f,  -100.0f,  100.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0
          -30.0f,  -80.0f,  100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0
         -50.0f,  -80.0f, 100.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0

         //picior masa stang fata
         60.0f,  -100.0f, 0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,
          80.0f,  -100.0f,  0.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
          80.0f,  -80.0f,  0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0f,
         60.0f,  -80.0f, 0.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0f,
         60.0f,  -100.0f, 100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0
          80.0f,  -100.0f,  100.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0
          80.0f,  -80.0f,  100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0
         60.0f,  -80.0f, 100.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0

         //picior masa dreapta fata
         60.0f,  130.0f, 0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,
          80.0f,  130.0f,  0.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
          80.0f,  150.0f,  0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0f,
```

```
        60.0f,  150.0f, 0.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0f,
        60.0f,  130.0f, 100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0f,
         80.0f,  130.0f,  100.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0f,
         80.0f,  150.0f,  100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0
        60.0f,  150.0f, 100.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0

        //picior masa dreapta spate
        -50.0f,  130.0f, 0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,
         -30.0f,  130.0f,  0.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
         -30.0f,  150.0f,  0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0
        -50.0f,  150.0f, 0.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0
        -50.0f,  130.0f, 100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0
         -30.0f,  130.0f,  100.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0
         -30.0f,  150.0f,  100.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0
        -50.0f,  150.0f, 100.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0

        //cub transparent
        // coordonate                    // culori
        -15.0f,  15.0f, 120.0f, 1.0f,  0.0f, 0.5f, 0.9f,  -1.0f,
        15.0f,  15.0f,  120.0f, 1.0f,  0.0f, 0.5f, 0.9f, 1.0f,
        15.0f,  45.0f,  120.0f, 1.0f,   0.0f, 0.5f, 0.9f,  1.0f,
        -15.0f,  45.0f, 120.0f, 1.0f,   0.0f, 0.5f, 0.9f,  -1.0
        -15.0f,  15.0f, 180.0f, 1.0f,  0.0f, 0.5f, 0.9f,  -1.0f,
        15.0f,  15.0f,  180.0f, 1.0f,  0.0f, 0.5f, 0.9f,  1.0f,
        15.0f,  45.0f,  180.0f, 1.0f,   0.0f, 0.5f, 0.9f,  1.0f,
        -15.0f,  45.0f, 180.0f, 1.0f,   0.0f, 0.5f, 0.9f,  -1.0

        // varfuri scaun 1
       -30.0f,  -200.0f, 60.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0
        60.0f,  -200.0f,  60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
        60.0f,  -110.0f,  60.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0
       -30.0f,  -110.0f, 60.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0
       -30.0f,  -200.0f, 80.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0f,
        60.0f,  -200.0f,  80.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0f,
        60.0f,  -110.0f,  80.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
       -30.0f,  -110.0f, 80.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0
```

```
//picior scaun stang spate
-10.0f,  -200.0f, 0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,
 -30.0f,  -200.0f,  0.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
 -30.0f,  -180.0f,  0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0f
-10.0f,  -180.0f, 0.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0f
-10.0f,  -200.0f, 60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0f,
 -30.0f,  -200.0f,  60.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0f,
 -30.0f,  -180.0f,  60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f
-10.0f,  -180.0f, 60.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f

//picior masa stang fata
40.0f,  -130.0f, 0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,
 60.0f,  -130.0f,  0.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
 60.0f,  -110.0f,  0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0f,
40.0f,  -110.0f, 0.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0f,
40.0f,  -130.0f, 60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0f,
 60.0f,  -130.0f,  60.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0f,
 60.0f,  -110.0f,  60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
40.0f,  -110.0f, 60.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,

//picior masa dreapta fata
-30.0f,  -130.0f, 0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,
 -10.0f,  -130.0f,  0.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
 -10.0f,  -110.0f,  0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0f
-30.0f,  -110.0f, 0.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0f
-30.0f,  -130.0f, 60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0f,
 -10.0f,  -130.0f,  60.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0f,
 -10.0f,  -110.0f,  60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f
-30.0f,  -110.0f, 60.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f

//picior masa dreapta spate
40.0f,  -200.0f, 0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,
 60.0f,  -200.0f,  0.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
 60.0f,  -180.0f,  0.0f, 1.0f,   1.0f, 0.5f, 0.2f,  1.0f,
40.0f,  -180.0f, 0.0f, 1.0f,    1.0f, 0.5f, 0.2f,  -1.0f,
```

```
        40.0f,  -200.0f, 60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  -1.0f,
         60.0f,  -200.0f,  60.0f, 1.0f, 1.0f, 0.5f, 0.2f,  1.0f,
         60.0f,  -180.0f,  60.0f, 1.0f,  1.0f, 0.5f, 0.2f,  1.0f,
        40.0f,  -180.0f, 60.0f, 1.0f,   1.0f, 0.5f, 0.2f,  -1.0f,

        // SCAUN 2
             -30.0f, 250.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f
             60.0f, 250.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
             60.0f, 160.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
             -30.0f, 160.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f
             -30.0f, 250.0f, 80.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f
             60.0f, 250.0f, 80.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
             60.0f, 160.0f, 80.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
             -30.0f, 160.0f, 80.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f

             //picior scaun stang spate
             -10.0f, 160.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
             -30.0f, 160.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
             -30.0f, 180.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
             -10.0f, 180.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
             -10.0f, 160.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f
             -30.0f, 160.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
             -30.0f, 180.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
             -10.0f, 180.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f

             //picior masa stang fata
             40.0f, 230.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
             60.0f, 230.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
             60.0f, 250.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f, 1
             40.0f, 250.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
             40.0f, 230.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
             60.0f, 230.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
             60.0f, 250.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
             40.0f, 250.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,

             //picior masa dreapta fata
```

```
            -30.0f, 230.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
            -10.0f, 230.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            -10.0f, 250.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            -30.0f, 250.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
            -30.0f, 230.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f
            -10.0f, 230.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            -10.0f, 250.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            -30.0f, 250.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f

            //picior masa dreapta spate
            40.0f, 160.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
            60.0f, 160.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            60.0f, 180.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f, 1
            40.0f, 180.0f, 0.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
            40.0f, 160.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,
            60.0f, 160.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            60.0f, 180.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, 1.0f,
            40.0f, 180.0f, 60.0f, 1.0f, 1.0f, 0.5f, 0.2f, -1.0f,


    };

    // indicii pentru varfuri
    GLubyte Indices[] =
    {
        // fetele "ground"
         1, 2, 0,   2, 0, 3,
         // fetele cubului
         5, 6, 4,   6, 4, 7,
         6, 7, 10, 10, 7, 11,
         11, 7, 8,   8, 7, 4,
         8, 4, 9,   9, 4, 5,
         5, 6, 9,   9, 6, 10,
         9, 10, 8,  8, 10, 11,
         //picior stang spate
         13, 14, 12,   14, 12, 15,
```

```
          14, 15, 18, 18, 15, 19,
          19, 15, 16,   16, 15, 12,
          16, 12, 17,   17, 12, 13,
          13, 14, 17,   17, 14, 18,
          17, 18, 16,  16, 18, 19,
          //picior stang fata
          21, 22, 20,   22, 20, 23,
          22, 23, 26, 26, 23, 27,
          27, 23, 24,   24, 23, 20,
          24, 20, 25,   25, 20, 21,
          21, 22, 25,   25, 22, 26,
          25, 26, 24,  24, 26, 27,
          //picior dreapta fata
         29, 30, 28,   30, 28, 31,
         30, 31, 34,   34, 31, 35,
         35, 31, 32,   32, 31, 28,
         32, 28, 33,   33, 28, 29,
         29, 30, 33,   33, 30, 34,
         33, 34, 32,   32, 34, 35,

         //picior dreapta fata
        37, 38, 36,   38, 36, 39,
        38, 39, 42,   42, 39, 43,
        43, 39, 40,   40, 39, 36,
        40, 36, 41,   41, 36, 37,
        37, 38, 41,   41, 38, 42,
        41, 42, 40,   40, 42, 43,

        //cub transparent
        45, 46, 44,   46, 44, 47,
        46, 47, 50,   50, 47, 51,
        51, 47, 48,   48, 47, 44,
        48, 44, 49,   49, 44, 45,
        45, 46, 49,   49, 46, 50,
        49, 50, 48,   48, 50, 51,
```

```
        //fata scaun 1
         53, 54, 52,   54, 52, 55,
         54, 55, 58,   58, 55, 59,
         59, 55, 56,   56, 55, 52,
         56, 52, 57,   57, 52, 53,
         53, 54, 57,   57, 54, 58,
         57, 58, 56,   56, 58, 59,
         //picior scaun 1
       61, 62, 60,   62, 60, 63,
       62, 63, 66,   66, 63, 67,
       67, 63, 64,   64, 63, 60,
       64, 60, 65,   65, 60, 61,
       61, 62, 65,   65, 62, 66,
       65, 66, 64,   64, 66, 67,
       //picior scaun 1
         69, 70, 68,  70, 68, 71,
         70, 71, 74,  74, 71, 75,
         75, 71, 72,  72, 71, 68,
         72, 68, 73,  73, 68, 69,
         69, 70, 73,  73, 70, 74,
         73, 74, 72,  72, 74, 75,
         //picior scaun 1
         77, 78, 76,   78, 76, 79,
         78, 79, 82,   82, 79, 83,
         83, 79, 80,   80, 79, 76,
         80, 76, 81,   81, 76, 77,
         77, 78, 81,   81, 78, 82,
         81, 82, 80,   80, 82, 83,
         //picior scaun 1
         85, 86, 84,   86, 84, 87,
         86, 87, 90,   90, 87, 91,
         91, 87, 88,   88, 87, 84,
         88, 84, 89,   89, 84, 85,
         85, 86, 89,   89, 86, 90,
         89, 90, 88,   88, 90, 91,
```

```
        //SCAUN 2
        //fata scaun
        93, 94, 92,   94, 92, 95,
        94, 95, 98,   98, 95, 99,
        99, 95, 96,   96, 95, 92,
        96, 92, 97,   97, 92, 93,
        93, 94, 97,   97, 94, 98,
        97, 98, 96,   96, 98, 99,
        //picior scaun
        101, 102, 100,   102, 100, 103,
        102, 103, 106,   106, 103, 107,
        107, 103, 104,   104, 103, 100,
        104, 100, 105,   105, 100, 101,
        101, 102, 105,   105, 102, 106,
        105, 106, 104,   104, 106, 107,
        //picior scaun
        109, 110, 108,   110, 108, 111,
        110, 111, 114,   114, 111, 115,
        115, 111, 112,   112, 111, 108,
        112, 108, 113,   113, 108, 109,
        109, 110, 113,   113, 110, 114,
        113, 114, 112,   112, 114, 115,
        //picior scaun
        117, 118, 116,   118, 116, 119,
        118, 119, 122,   122, 119, 123,
        123, 119, 120,   120, 119, 116,
        120, 116, 121,   121, 116, 117,
        117, 118, 121,   121, 118, 122,
        121, 122, 120,   120, 122, 123,
        //picior scaun
        125, 126, 124,   126, 124, 127,
        126, 127, 130,   130, 127, 131,
        131, 127, 128,   128, 127, 124,
        128, 124, 129,   129, 124, 125,
        125, 126, 129,   129, 126, 130,
        129, 130, 128, 128, 130, 131,
```

```
    };

    glGenVertexArrays(1, &VaoId);
    glGenBuffers(1, &VboId);
    glGenBuffers(1, &EboId);
    glBindVertexArray(VaoId);

    glBindBuffer(GL_ARRAY_BUFFER, VboId);
    glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EboId);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(Indices), Indic

    // atributul 0 = pozitie
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 10 * sizeof(
    // atributul 1 = culoare
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(
    // atributul 2 = normale
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 10 * sizeof(
}
void DestroyVBO(void)
{
    glDisableVertexAttribArray(2);
    glDisableVertexAttribArray(1);
    glDisableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glDeleteBuffers(1, &VboId);
    glDeleteBuffers(1, &EboId);
    glBindVertexArray(0);
    glDeleteVertexArrays(1, &VaoId);

    //  Stergerea bufferelor pentru VARFURI (Coordonate, Culori)
```

```
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glDeleteBuffers(1, &VboId1);
    glDeleteBuffers(1, &EboId1);
    glDeleteBuffers(1, &VboId2);
    glDeleteBuffers(1, &EboId2);
    glDeleteBuffers(1, &VboId3);
    glDeleteBuffers(1, &EboId3);
    glDeleteBuffers(1, &VboId4);
    glDeleteBuffers(1, &EboId4);
    glDeleteBuffers(1, &VboId5);
    glDeleteBuffers(1, &EboId5);
    glDeleteBuffers(1, &VboId6);
    glDeleteBuffers(1, &EboId6);
    glDeleteBuffers(1, &VboId7);
    glDeleteBuffers(1, &EboId7);
    glDeleteBuffers(1, &VboId8);
    glDeleteBuffers(1, &EboId8);
    glDeleteBuffers(1, &VboId9);
    glDeleteBuffers(1, &EboId9);
    glDeleteBuffers(1, &VboId10);
    glDeleteBuffers(1, &EboId10);
    glDeleteBuffers(1, &VboId11);
    glDeleteBuffers(1, &EboId11);
    glDeleteBuffers(1, &VboId12);
    glDeleteBuffers(1, &EboId12);
}

void CreateShaders(void)
{
    ProgramId = LoadShaders("11_01_Shader.vert", "11_01_Shader.f
    glUseProgram(ProgramId);
}

void DestroyShaders(void)
{
    glDeleteProgram(ProgramId);
```

```
}

void Initialize(void)
{
    myMatrix = glm::mat4(1.0f);
    matrRot = glm::rotate(glm::mat4(1.0f), PI / 8, glm::vec3(0.0
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    CreateVBO();
    CreateVAO1();
    CreateVAO2();
    CreateVAO3();
    CreateVAO4();
    CreateVAO5();
    CreateVAO6();
    CreateVAO7();
    CreateVAO8();
    CreateVAO10();
    CreateVAO11();
    CreateVAO12();
    CreateShaders();
    // locatii pentru shader-e
    myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix
    matrUmbraLocation = glGetUniformLocation(ProgramId, "matrUmb
    viewLocation = glGetUniformLocation(ProgramId, "view");
    projLocation = glGetUniformLocation(ProgramId, "projection");
    lightColorLocation = glGetUniformLocation(ProgramId, "lightC
    lightPosLocation = glGetUniformLocation(ProgramId, "lightPos
    viewPosLocation = glGetUniformLocation(ProgramId, "viewPos");
    codColLocation = glGetUniformLocation(ProgramId, "codCol");
}
void RenderFunction(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);

    //pozitia observatorului
```

```
    Obsx = Refx + dist * cos(alpha) * cos(beta);
    Obsy = Refy + dist * cos(alpha) * sin(beta);
    Obsz = Refz + dist * sin(alpha);

    // matrice de vizualizare + proiectie
    glm::vec3 Obs = glm::vec3(Obsx, Obsy, Obsz);   // se schimba
    glm::vec3 PctRef = glm::vec3(Refx, Refy, Refz); // pozitia p
    glm::vec3 Vert = glm::vec3(Vx, Vy, Vz); // verticala din pla
    view = glm::lookAt(Obs, PctRef, Vert);
    glUniformMatrix4fv(viewLocation, 1, GL_FALSE, &view[0][0]);
    projection = glm::infinitePerspective(fov, GLfloat(width) /
    glUniformMatrix4fv(projLocation, 1, GL_FALSE, &projection[0]

    // matricea pentru umbra
    float D = -0.5f;
    matrUmbra[0][0] = zL + D; matrUmbra[0][1] = 0; matrUmbra[0]
    matrUmbra[1][0] = 0; matrUmbra[1][1] = zL + D; matrUmbra[1]
    matrUmbra[2][0] = -xL; matrUmbra[2][1] = -yL; matrUmbra[2][2
    matrUmbra[3][0] = -D * xL; matrUmbra[3][1] = -D * yL; matrUm
    glUniformMatrix4fv(matrUmbraLocation, 1, GL_FALSE, &matrUmbr

    // Variabile uniforme pentru iluminare
    glUniform3f(lightColorLocation, 1.0f, 1.0f, 1.0f);
    glUniform3f(lightPosLocation, xL - 30.0f, yL - 30.0f, zL + 2
    glUniform3f(viewPosLocation, Obsx, Obsy, Obsz);

    // SFERA
    glBindVertexArray(VaoId1);
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt consider
            glDrawElements(
                GL_QUADS,
                4,
```

```
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }

    //desenare umbra
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt consider
            glDrawElements(
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }
    // SFERA 2
    glBindVertexArray(VaoId3);
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt consider
            glDrawElements(
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }
    //desenare umbra
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
```

```
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt consider
            glDrawElements(
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }

    // SFERA
    glBindVertexArray(VaoId4);
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt consider
            glDrawElements(
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }

    //desenare umbra
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt consider
            glDrawElements(
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
```

```
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }

    // SFERA -ochi stang
    glBindVertexArray(VaoId6);
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt consider
            glDrawElements(
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }

    // SFERA -ochi drept
    glBindVertexArray(VaoId7);
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt consider
            glDrawElements(
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }
    // CONUL2
    glBindVertexArray(VaoId5);
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
```

```
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt consider
            glDrawElements(
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }
    //desenare umbra
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix

    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt consider
            glDrawElements(
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }

    // cub-tichie om-de-zapada
    glBindVertexArray(VaoId2);
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_SHORT, (GLvoid*

    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
```

```
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_SHORT, (GLvoid*

    glBindVertexArray(VaoId9);
    codCol = 0;

    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, 0);

    LoadTexture("parchet.png");
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);



    // glob
    glEnable(GL_BLEND);
    //glDepthMask(GL_FALSE);
    glBlendFunc(GL_SRC_ALPHA, GL_SRC_ALPHA);
    glBindVertexArray(VaoId8);
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt consider
            glDrawElements(
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }
    glDepthMask(GL_TRUE);
    glDisable(GL_BLEND);

    //desenare umbra
```

```
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt conside
            glDrawElements(
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }

    // portocala 1

    glBindVertexArray(VaoId10);
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt conside
            glDrawElements(
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }
    glDepthMask(GL_TRUE);
    glDisable(GL_BLEND);

    //desenare umbra
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
```

```
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt consider
            glDrawElements(
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }

    // portocala 2

    glBindVertexArray(VaoId11);
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt consider
            glDrawElements(
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }
    glDepthMask(GL_TRUE);
    glDisable(GL_BLEND);

    //desenare umbra
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt consider
            glDrawElements(
```

```
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }


    // farfurie
    glBindVertexArray(VaoId12);
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt consider
            glDrawElements(
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
    }
    glDepthMask(GL_TRUE);
    glDisable(GL_BLEND);

    //desenare umbra
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    for (int patr = 0; patr < (NR_PARR + 1) * NR_MERID; patr++)
    {
        if ((patr + 1) % (NR_PARR + 1) != 0) // nu sunt consider
            glDrawElements(
                GL_QUADS,
                4,
                GL_UNSIGNED_SHORT,
                (GLvoid*)((2 * (NR_PARR + 1) * (NR_MERID)+4 * pa
```

```
    }


    // desenare cub
    glBindVertexArray(VaoId);
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, 0);
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(6

    //desenare picior stang spate
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(4

    //desenare picior stang fata
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(7

    //desenare picior drept fata
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(1


    //desenare picior drept fata
    codCol = 0;
```

```
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(1



    ////desenare cub transparent
    //glEnable(GL_BLEND);
    ////glDepthMask(GL_FALSE);
    //glBlendFunc(GL_SRC_ALPHA, GL_SRC_ALPHA);
    //codCol = 0;
    //glUniform1i(codColLocation, codCol);
    //myMatrix = glm::mat4(1.0f);
    //glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatri
    //glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)
    //glDepthMask(GL_TRUE);
    //glDisable(GL_BLEND);




    // desenare umbra cub
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(6



    // desenare umbra picior stang spate
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(4
```

```
    // desenare umbra picior stang fata
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(

    // desenare umbra picior drept fata
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(1

    // desenare umbra picior drept spate
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(1

    // desenare umbra cub transparent
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(1

    // desenare fata fata scaun1
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(2

    // desenare umbra fata scaun1
```

```
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(2

    // desenare picior scaun1
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(2

    // desenare umbra picior scaun1
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(2

    // desenare picior scaun1
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(2

    // desenare umbra picior scaun1
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(2

    // desenare picior scaun1
    codCol = 0;
```

```
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(3

    // desenare umbra picior scaun1
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(3

    // desenare picior scaun1
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(3

    // desenare umbra picior scaun1
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(3



    //SCAUN2
    // desenare fata fata scaun1
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(4

    // desenare umbra fata scaun1
```

```
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(4

    // desenare picior scaun1
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(4

    // desenare umbra picior scaun1
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(4

    // desenare picior scaun1
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(4

    // desenare umbra picior scaun1
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(4

    // desenare picior scaun1
    codCol = 0;
```

```
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(5

    // desenare umbra picior scaun1
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(5

    // desenare picior scaun1
    codCol = 0;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(5

    // desenare umbra picior scaun1
    codCol = 1;
    glUniform1i(codColLocation, codCol);
    myMatrix = glm::mat4(1.0f);
    glUniformMatrix4fv(myMatrixLocation, 1, GL_FALSE, &myMatrix
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, (void*)(5




    glutSwapBuffers();
    glFlush();
}
void Cleanup(void)
{
    DestroyShaders();
    DestroyVBO();
}
```

```
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(1200, 900);
    glutCreateWindow("Iluminare - Umbre - OpenGL <<nou>>");
    glewInit();
    Initialize();
    glutIdleFunc(RenderFunction);
    glutDisplayFunc(RenderFunction);
    glutKeyboardFunc(processNormalKeys);
    glutSpecialFunc(processSpecialKeys);
    glutCloseFunc(Cleanup);
    glutMainLoop();
}
```

```
//
// ================================================
// | Grafica pe calculator                        |
// ================================================
// | Laboratorul XI - 11_01_Shader.frag |
// ======================================
//
//  Shaderul de fragment / Fragment shader - afecteaza culoarea
//

#version 330 core

in vec3 FragPos;
in vec3 Normal;
in vec3 inLightPos;
in vec3 inViewPos;
```

```
in vec3 dir;
in vec3 ex_Color;

out vec4 out_Color;

uniform vec3 lightColor;
uniform int codCol;

// definirea culoorii si a densitatii cetii
uniform vec3 fogColor;
uniform float fogDensity;

void main(void)
  {
    if (codCol==0) // pentru codCol==0 este aplicata iluminarea
    {
    // Ambient
    float ambientStrength = 0.2f;
    vec3 ambient = ambientStrength * lightColor;

    // Diffuse
    vec3 normala = normalize(Normal);
    vec3 lightDir = normalize(inLightPos - FragPos);
    //vec3 lightDir = normalize(dir); // cazul unei surse direct
    float diff = max(dot(normala, lightDir), 0.0);
    vec3 diffuse = diff * lightColor;

    // Specular
    float specularStrength = 0.5f;
    vec3 viewDir = normalize(inViewPos - FragPos);//vector catre
    vec3 reflectDir = reflect(-lightDir, normala); // reflexia r
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), 1);
    vec3 specular = specularStrength * spec * lightColor;
    vec3 emission=vec3(0.0, 0.0, 0.0);
    vec3 result = emission+(ambient + diffuse + specular) * ex_C
    out_Color = vec4(result, 1.0f);
```

```
    // definirea distantei ca diferenta dintre inViewPos si Frag
    float distance = length(inViewPos - FragPos);

    // calculul factorului de ceata
    float fogFactor = pow(2.7182, (-1) * fogDensity * distance),

    // implementarea efectului de ceata
    vec3 foggedColor = mix(fogColor, result, fogFactor);
    out_Color = vec4(foggedColor, 1.0f);
    }

    if (codCol==1) // pentru codCol==1 este desenata umbra
    {
        vec3 black = vec3 (0.0, 0.0, 0.0);
        out_Color = vec4 (black, 1.0);
     }
  }
```

```
//
// ================================================
// | Grafica pe calculator                        |
// ================================================
// | Laboratorul XI - 11_01_Shader.vert |
// ======================================
//
//  Shaderul de varfuri / Vertex shader - afecteaza geometria sc

#version 330 core

layout(location=0) in vec4 in_Position;
layout(location=1) in vec3 in_Color;
layout(location=2) in vec3 in_Normal;

out vec3 FragPos;
```

```
out vec3 Normal;
out vec3 inLightPos;
out vec3 inViewPos;
out vec3 ex_Color;
out vec3 dir;

uniform mat4 matrUmbra;
uniform mat4 myMatrix;
uniform mat4 view;
uniform mat4 projection;
uniform vec3 lightPos;
uniform vec3 viewPos;
uniform vec3 lightColor;
uniform int codCol;



void main(void)
  {
    ex_Color=in_Color;
    if (codCol==0)
    {
        gl_Position = projection*view*myMatrix*in_Position;
        Normal =mat3(projection*view*myMatrix)*in_Normal;
        inLightPos = vec3(projection*view*myMatrix* vec4(lightPo
        inViewPos =vec3(projection*view*myMatrix*vec4(viewPos, 1
        dir = mat3(projection*view*myMatrix) * vec3(0.0,100.0,20
        FragPos = vec3(gl_Position);


    }
    if (codCol==1)
        gl_Position = projection*view*matrUmbra*myMatrix*in_Posi
        FragPos = vec3(gl_Position);
   }
```