



---

## Prelucrare Grafică - Proiect

*Semestrul 1 - 2023 - 2024*

---

AUTOR: Dîrnu Talida

GRUPA: 30236

FACULTATEA DE AUTOMATICĂ  
ŞI CALCULATOARE

30 Decembrie 2023

## Cuprins

<b>1</b>	<b>Prezentarea temei . . . . .</b>	<b>2</b>
<b>2</b>	<b>Scenariul . . . . .</b>	<b>2</b>
2.1	Descrierea scenei și a obiectelor . . . . .	2
2.2	Funcționalități . . . . .	3
<b>3</b>	<b>Detalii de implementare . . . . .</b>	<b>5</b>
3.1	Funcții și algoritmi . . . . .	5
3.1.1	Soluții posibile . . . . .	12
3.1.2	Motivarea abordării alese . . . . .	13
3.2	Modelul grafic . . . . .	13
3.3	Structuri de date . . . . .	13
3.4	Ierarhia de clase . . . . .	14
<b>4</b>	<b>Prezentarea interfeței grafice utilizator / manual de utilizare . . . . .</b>	<b>14</b>
<b>5</b>	<b>Concluzii și dezvoltări ulterioare . . . . .</b>	<b>15</b>
<b>6</b>	<b>Referințe . . . . .</b>	<b>15</b>

# 1 Prezentarea temei

Tema acestui proiect se bazează pe simularea unei scene 3D la alegere cu detalii cât mai realiste. Eu am ales să modelez o scenă de iarnă, cu obiecte specifice, care să permită introducerea de animații, să fie texturate corespunzător și fiecare din acestea să ofere un plus peisajului. Proiectul sugerează implicit o familiarizare în primul rând cu mediul de proiectare Visual Studio, pentru ca apoi să permită aplicarea principiilor OpenGL într-un mod plăcut și practic, în urma prezentării acestora în cadrul orelor de curs și de laborator. Rezultatul permite o interacțiune facilă a utilizatorului cu scena creată, prin intermediul mouse-ului și diferitelor taste care permit vizualizarea unor funcționalități pe care le voi prezenta ulterior.



Figura 1: Prezentarea scenei

## 2 Scenariul

### 2.1 Descrierea scenei și a obiectelor

Am construit scena în software-ul grafic Blender, având ca resurse tutorialele furnizate de către profesorul îndrumător de laborator. Am selectat diverse obiecte 3D sugestive temei alese, le-am texturat după preferințe și le-am introdus pe un plan texturat și modelat corespunzător pentru a reprezenta planul acoperit de zăpadă, cu mici denivelări. Am introdus apoi un skydome prin utilizarea unei sfere secționate pe care am aplicat o textură în interiorul și exteriorul acesteia, care să simuleze un cer îinstelat. Am adăugat lună, case, diverse animăluțe, arbori, un pătionar, un om de zăpadă, un munte, un leagăn acoperit de zăpadă și un foișor. Smania a fost implicată ca obiect separat de scenă pentru a permite ulterior animarea acesteia. Fulgii sunt reprezentăți de un obiect 3D multiplicat, iar pentru artificii am folosit 6 sfere adăugate în blender și texturate cu 6 texturi de culori diferite.

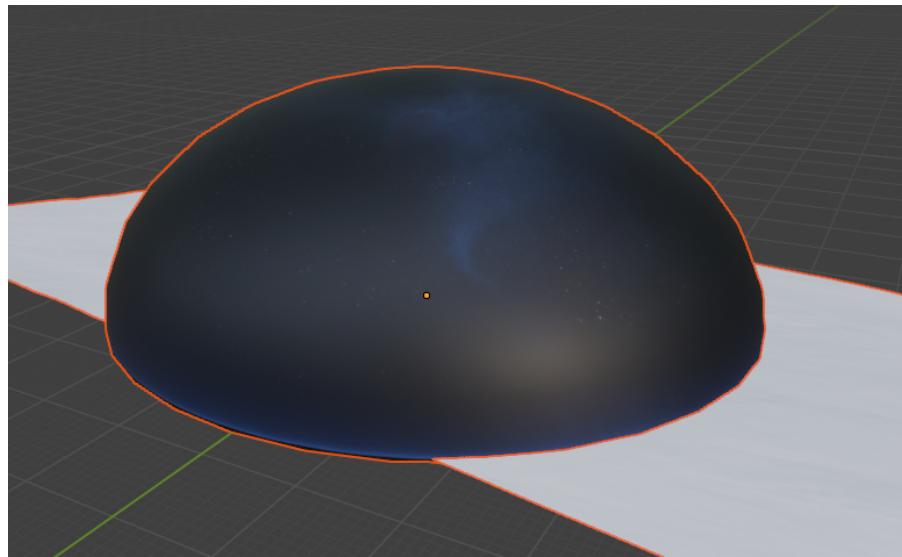


Figura 2: Modelarea scenei în Blender



Figura 3: Modelarea scenei în Blender

## 2.2 Funcționalități

Am reușit să introduc în proiectul meu următoarele funcționalități:

- Posibilități de navigare: prin intermediul mouse-ului și al tastaturii
- Posibilitatea redimensionării ferestrei de vizualizare
- Posibilitatea vizualizării scenei în mod wireframe, solid și punctiform

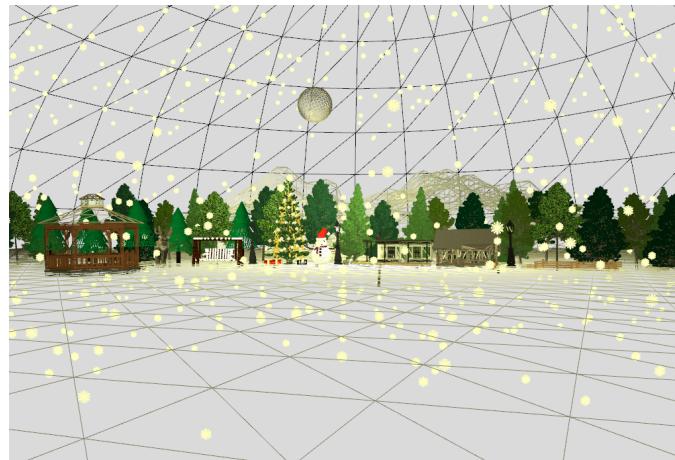


Figura 4: Vizualizare în mod wireframe



Figura 5: Vizualizare în mod solid



Figura 6: Vizualizare în mod punctiform

- Implementarea efectului de ceată
- Implementarea efectului de ninsoare
- Implementarea unei animații (mișcarea unui obiect în scenă)
- Implementarea efectului de lumini punctiforme prin intermediul felinarelor

- Implementarea efectului de lumină globală
- Implementarea animației unor artificii
- Implementarea umbrelor prin intermediul depthMapShader
- Posibilitatea vizualizării scenei printr-un tur automat al acesteia

## 3 Detalii de implementare

### 3.1 Funcții și algoritmi

Au fost implementate funcții din cadrul laboratorului, precum keyboardCallback(), mouseCallback(), processMovement(), initOpenGLWindow(), windowResizeCallback(), initOpenGLState(), initObjects(), initShaders(), initUniforms(), initFBO(), computeLightSpaceTrMatrix() și drawObjects(), altele pentru a introduce efectele dorite, precum ninsoarea și animarea scenei. Implementarea efectului de ceată a avut ca resurse informațiile furnizate în cadrul materialului de laborator.

Pentru implementarea efectului de ninsoare, am utilizat 1000 de fulgi care utilizează același model 3D, situați la distanțe egale pe axe X și Z. Pozițiile pe axe X și Z au fost determinate prin forma matriceală a vectorului de fulgi folosind funcția de mapare prezentată în codul de mai jos. Fulgul este reprezentat în cod sub forma unui struct având ca și componente poziția acestuia în scenă și velocity (practic viteza deplasării acestuia în jos, pe axa Y). În bucla corespunzătoare funcției updateSnowfall(), la componenta y a acestuia se adaugă componenta velocity pentru a simula deplasarea în jos.

Pentru a genera continuitate, la atingerea unui anumit prag negativ pe axa y, componenta y se modifică cu una pozitivă pentru a-și relua deplasarea.

```

1 //SNOWING EFFECT
2 void updateSnowfall() {
3     for (int i = 0; i < numSnowflakes; i++) {
4         snowflakes[i].position += snowflakes[i].velocity;
5         if (snowflakes[i].position.y < -1.0f) {
6             snowflakes[i].position.y = 4.0f;
7         }
8     }
9 }
10
11 void initSnow() {
12     float next_x, next_z;
13     for (int i = 0; i < numSnowflakes; i++) {
14         next_x = -8.0f + (i / 20) * 0.32f; // -8 -> 8 pt x, 50 vals 16 / 50 = 0.32
15         next_z = -5.0f + (i % 20) * 0.5f; // -5 -> 5 pt z, 20 vals, 10 / 20 = 0.5
16         float randomValue = static_cast<float>(std::rand()) / RAND_MAX;
17         float randomInRange = 4.0f + randomValue * 4.0f; //random y value between 4 and 8
18         snowflakes[i].position = glm::vec3(next_x, randomInRange, next_z);
19         snowflakes[i].velocity = glm::vec3(0.0f, -0.05f, 0.0f);
20     }
21 }
```

Pentru efectul de ceată, am ales din cele 3 metode prezentate în cadrul laboratorului pe cea exponențială pătratică, cu următoarea formulă:

$$fogFactor = e^{-(fragmentDistance \times fogDensity)^2}$$

Pentru aceasta, am inclus funcția `computeFog()` în cadrul fragment shader:

```

1 float computeFog()
2 {
3     float fogDensity;
4     if (showFog)
5         fogDensity = 0.05f;
6     else
7         fogDensity = 0.0f;
8     float fragmentDistance = length(fragmentPosEyeSpace);
9     float fogFactor = exp(-pow(fragmentDistance * fogDensity, 2));
10
11    return clamp(fogFactor, 0.0f, 1.0f);
12 }
```



Figura 7: Implementarea căii și a ninsorii

Pentru lumina globală, am combinat cele 3 concepte de lumină ambientală, difuză și speculară, prezentate în cadrul laboratorului. Aceasta a fost calculată în funcția `computeGlobalLight()` din fragment shader:

```

1 void computeGlobalLight()
2 {
3     vec3 normalEye = normalize(normalMatrix * normal);
4     //normalize light direction
5     vec3 lightDirN = vec3(normalize(view * vec4(lightDir, 0.0f)));
6     //compute view direction
7     vec3 viewDir = normalize(- fragmentPosEyeSpace.xyz);
```

```

8   //compute ambient light
9   ambient = ambientStrength * lightColor;
10  //compute diffuse light
11  diffuse = max(dot(normalEye, lightDirN), 0.0f) * lightColor;
12  //compute specular light
13  vec3 reflectDir = reflect(-lightDirN, normalEye);
14  float specCoeff = pow(max(dot(viewDir, reflectDir), 0.0f), shininess);
15  specular = specularStrength * specCoeff * lightColor;
16 }

```



Figura 8: Lumină globală

Am implementat apoi și lumina punctiformă, având ca și surse cele 4 felinare din scenă. Coordonatele din scenă ale acestora sunt definite în vectorul lamp\_positions și sunt transmise către fragment shader. Spre deosebire de luminile direcționale, razele generate de luminile punctiforme se estompează în funcție de distanță, astfel că obiectele mai apropiate de cele 4 felinare par mai iluminate decât obiectele mai îndepărtate. Cei 3 parametri (constant, linear și quadratic) sunt definiți în main și transmiși către fragment shader, unde sunt folosiți pentru a calcula atenuarea după formula prezentată în laborator:  $\text{float att} = 1.0f / (\text{constant} + \text{linear} * \text{dist} + \text{quadratic} * (\text{dist} * \text{dist}))$ . Coeficientul de atenuare este aplicat pentru a reduce intensitatea luminii în legătură cu distanța, iar realismul devine considerabil crescut prin faptul că se încorporează o atenuare pătratică în funcție de distanță. În cele din urmă, se aplică atenuarea componentelor ambientale, difuze și speculare:

```

1 void computeLampLights()
2 {
3     for (int i = 0; i < 4; i++)
4     {
5         vec3 normalEye = normalize(normal);
6         vec4 lpe = view * model * vec4(positions[i], 1.0f);
7         vec3 lightPosEye = lpe.xyz;
8         //compute light direction
9         vec3 lightDirN = normalize(lightPosEye - fragmentPosEyeSpace.xyz);
10        //compute view direction

```

```

11     vec3 viewDirN = normalize(-fragmentPosEyeSpace.xyz);
12     vec3 reflection = normalize(reflect(lightDirN, normalEye));
13     float specCoeff = pow(max(dot(viewDirN, reflection), 0.0f), shininess);
14     //compute distance to light
15     float dist = length(lightPosEye - fragmentPosEyeSpace.xyz);
16     //compute attenuation
17     float att = 1.0f / (constant + linear * dist + quadratic * (dist*dist));
18     //compute ambient light
19     vec3 ambient1 = att * ambientStrength * lightColor;
20     ambient1 *= texture(diffuseTexture, fTexCoords);
21     vec3 diffuse1 = att * max(dot(normalEye, lightDirN), 0.0f) * lightColor;
22     diffuse1 *= texture(diffuseTexture, fTexCoords);
23     vec3 specular1 = att * specularStrength * specCoeff * lightColor;
24     specular1 *= texture(specularTexture, fTexCoords);
25     vec3 newColorPoint = min((ambient + diffuse) + specular, 1.0f);
26     colorPoint += newColorPoint;
27     ambient_computed += ambient1;
28     diffuse_computed += diffuse1;
29     specular_computed += specular1;
30 }
31 }
```



Figura 9: Lumină punctiformă

Pentru efectul de animație, am pornit de la o idee similară cu implementarea ninsorii: sania este un obiect 3D separat de restul scenei, care este introdus în cadru în momentul apăsării tastei Y. Îi este atribuită o poziție inițială și o componentă velocity care repezintă viteza acesteia de deplasare pe axa X. Odată cu atingerea unui prag maxim pe axa X, care indică lipsa vizibilității acesteia în scenă, poziția saniei este resetată la valoarea inițială pentru continuitate.

```

1 //SLEIGH ANIMATION
2 void initSleigh() {
3     mySleigh.position = glm::vec3(-5.0f, 0.12f, 1.4f); //initial position
4     mySleigh.velocity = glm::vec3(0.1f, 0.0f, 0.0f); // "speed" on x axis
```

```

5  }
6
7 void updateSleighAnimation() {
8     if (mySleigh.position.x < 8.0f) {
9         mySleigh.position += mySleigh.velocity;
10    }
11   if (mySleigh.position.x >= 6.0f) {
12       mySleigh.position = glm::vec3(-5.0f, 0.12f, 1.4f);
13   }
14 }
```

Pentru efectul de artificii, am pornit de la texturarea a 6 sfere cu texturi de culori diferite; le-am vizualizat ca mici puncte care pleacă dintr-un centru al cercului și se deplasează pe direcții diferite, până ating conturul cercului. Am ales 18 poziții pentru artificii (18 centre diferite de cerc ca diferite coordonate din scenă); pentru fiecare sfert de cerc există între 4 și 15 direcții de deplasare diferite, împărțite chitabil. Există o rază aleasă random, între 1.5 - 2.5 pentru fiecare cerc și un speed (o viteză de deplasare) între 0.15 și 0.25.

```

1 Firework initFirework(glm::vec3 initPosition, int quarterSplit, float radius, float speed)
2 {
3     Firework firework;
4     firework.initPosition = initPosition;
5     glm::vec3 anchors[] = { glm::vec3(1, 0, 0),
6                             glm::vec3(0, 1, 0), glm::vec3(-1, 0, 0),
7                             glm::vec3(0, -1, 0) };
8     float floatSplit = quarterSplit * 1.0;
9     for (int i = 0; i < 4; i++) {
10         for (int j = 0; j < quarterSplit; j++) {
11             firework.currentPositions[i * quarterSplit + j] = initPosition;
12             float floatJ = j * 1.0;
13             glm::vec3 dir = (anchors[i] * floatJ + anchors[(i + 1) % 4] *
14                               (floatSplit - floatJ)) / floatSplit;
15             // normalize
16             dir /= sqrtf(glm::dot(dir, dir));
17             firework.directions[i * quarterSplit + j] = dir;
18         }
19     }
20     firework.active = false;
21     firework.radius = radius;
22     firework.speed = speed;
23     firework.directionsCount = 4 * quarterSplit;
24     return firework;
25 }
26
27 Firework fireworks[FIREWORKS_COUNT];
28 int fireworksCursor = 0;
29
30 void initFireworks()
31 {
```

```

32     float fireworksY = 3.0f;
33     float values[] = { 1.0f, 1.5f, 3.0f };
34     glm::vec3 fireworksPositions[FIREWORKS_COUNT] = {
35         glm::vec3(values[0], fireworksY, values[0]),
36             glm::vec3(-values[0], fireworksY, -values[0]),
37                 glm::vec3(values[1], fireworksY, values[1]),
38                     glm::vec3(-values[1], fireworksY, -values[1]),
39                         glm::vec3(values[1], fireworksY, values[0]),
40                             glm::vec3(-values[1], fireworksY, values[0]),
41                                 glm::vec3(values[1], fireworksY, -values[0]),
42                                     glm::vec3(-values[1], fireworksY, -values[0]),
43                                         glm::vec3(values[2], fireworksY, values[2]),
44                                             glm::vec3(-values[2], fireworksY, -values[2]),
45                                                 glm::vec3(values[2], fireworksY, values[1]),
46                                                     glm::vec3(values[2], fireworksY, -values[1]),
47                                                         glm::vec3(-values[2], fireworksY, values[1]),
48                                                             glm::vec3(-values[2], fireworksY, -values[1]),
49                                                               glm::vec3(values[2], fireworksY, values[0]),
50                     glm::vec3(values[2], fireworksY, -values[0]),
51                         glm::vec3(-values[2], fireworksY, values[0]),
52                             glm::vec3(-values[2], fireworksY, -values[0]),
53     };
54     for (int i = 0; i < FIREWORKS_COUNT; i++) {
55         int quarterSplit = 4 + rand() % (MAX_FIREWORK_QUARTER_SPLIT - 6 + 1);
56         // 4 - MAX_FIREWORK_QUARTER_SPLIT
57         float radius = (15 + rand() % 11) / 10.0; // 1.5 - 2.5
58         float speed = (15 + rand() % 11) / 100.0; // 0.15 - 0.25
59         fireworks[i] = initFirework(fireworksPositions[i], quarterSplit, radius, speed);
60     }
61 }
62
63
64
65 //FIREWORKS ANIMATION
66 void updateFireworksAnimation() {
67     for (int i = 0; i < FIREWORKS_COUNT; i++) {
68         if (!fireworks[i].active) {
69             continue;
70         }
71         int directions = fireworks[i].directionsCount;
72         for (int j = 0; j < directions; j++) {
73             fireworks[i].currentPositions[j] +=
74                 fireworks[i].directions[j] * fireworks[i].speed;
75         }
76         glm::vec3 radiusVec =
77             fireworks[i].currentPositions[0] - fireworks[i].initPosition;
78             //vectorul de deplasare; cand atinge lungimea razei cercului,
79             //se dezactiveaza si se reseteaza la pozitiile initiale

```

```

80     float radiusVecLength = sqrtf(glm::dot(radiusVec, radiusVec));
81     if (radiusVecLength > fireworks[i].radius) {
82         fireworks[i].active = false;
83         for (int j = 0; j < directions; j++) {
84             fireworks[i].currentPositions[j] =
85                 fireworks[i].initPosition;
86         }
87     }
88 }
89 }
```

Atunci când sferă (artificia) atinge pe vectorul de deplasare lungimea maximă (raza cercului), se dezactivează, iar artificia este setată la coordonatele inițiale, de pornire.

Pentru animația de prezentare, am folosit un fișier ("tour-directions.txt") pentru salvarea pozițiilor camerei (cameraPosition, cameraFrontDirection, cameraRightDirection), iar acest lucru se va realiza din 200 în 200 ms (partea de recording). Pentru vizualizarea turului automat, se vor citi din fișier coordonatele salvate și se vor folosi pentru mișcarea camerei în scenă (se asemenea, din 200 în 200 ms).

```

1 void recordRelatedStuff() {
2     LPFILETIME currentTime = new FILETIME();
3     GetSystemTimeAsFileTime(currentTime);
4     if (currentTime->dwLowDateTime - lastTime < 200) {
5         return;
6     }
7     if (isRecording) {
8         fprintf(
9             recording,
10            "%f %f %f %f %f %f %f %f\n",
11            myCamera.cameraPosition.x, myCamera.cameraPosition.y,
12            myCamera.cameraPosition.z,
13            myCamera.cameraFrontDirection.x, myCamera.cameraFrontDirection.y,
14            myCamera.cameraFrontDirection.z,
15            myCamera.cameraRightDirection.x, myCamera.cameraRightDirection.y,
16            myCamera.cameraRightDirection.z
17        );
18     lastTime = currentTime->dwLowDateTime;
19 }
20 else if (isPlaying) {
21     if (!feof(recording)) {
22         fscanf(
23             recording,
24            "%f%f%f%f%f%f%f%f",
25            &myCamera.cameraPosition.x, &myCamera.cameraPosition.y,
26            &myCamera.cameraPosition.z,
27            &myCamera.cameraFrontDirection.x, &myCamera.cameraFrontDirection.y,
28            &myCamera.cameraFrontDirection.z,
29            &myCamera.cameraRightDirection.x, &myCamera.cameraRightDirection.y,
30            &myCamera.cameraRightDirection.z
31 }
```

```

31     );
32     lastTime = currentTime->dwLowDateTime;
33 }
34 else {
35     printf("Finished tour!\n");
36     fclose(recording);
37     recording = NULL;
38     isPlaying = false;
39 }
40 }
41 delete currentTime;
42 }

```

Pentru generarea umbrelor s-a folosit algoritmul de Shadow Mapping prezentat în laborator. Shadow mapping presupune două etape, rasterizarea scenei din punctul de vedere al luminii și rasterizarea scenei din punctul de vedere al observatorului. În prima etapă, valorile de adâncime din scenă sunt stocate într-o hartă de umbră (sau adâncime). Se creează o textură de adâncime pe care o atașăm la un obiect framebuffer. Toată scena va fi rasterizată în acel obiect. Pentru a doua etapă, se compară adâncimea fiecărui fragment vizibil din punctul de vedere al luminii cu valorile de adâncime din harta umbrelor. Fragmentele cu o adâncime mai mare decât cea stocată anterior în harta de adâncime nu sunt direct vizibile din punctul de vedere al luminii și sunt deci în umbră.



Figura 10: Exemplificare umbre

### 3.1.1 Soluții posibile

Eu am implementat modelul de iluminare Phong; s-ar fi putut implementa modelul Blinn-Phong, versiunea sa modificată, cu scopul de a se îmbunătăți reflexiile speculare în anumite condiții, precum un coeficient foarte scăzut de strălucire. Astfel s-ar fi îmbunătățit performanța, evitându-se costul costisitor al vectorului de reflexie. S-ar fi putut implementa de asemeneea și o animație a unei componente separate a unui obiect, precum roțile unei mașini de zăpadă, care se rotesc în timpul translației acesteia. În plus, s-ar fi putut adăuga un sky-box în locul sky-dome-ului.

### 3.1.2 Motivarea abordării alese

Pentru animații, am ales acest mod de abordare pentru simplitate și eficiență (un simplu struct și o buclă pentru actualizare) și pentru faptul că funcționează atât în cazul mișcării saniei, cât și în cazul simulării efectului de ninsoare. Efectul de ceată a fost implementat și el pentru simplitate, fiind de asemenea și o resursă prezentată în laboratorul meu. Scena a fost modelată de la început prin includerea felinarelor, care au fost inclus în proiectul meu. Deoarece nu am găsit soluții pentru o animație mai complexă (un obiect care se mișcă în raport cu obiectul său constitutiv), am implementat efectul unor artificii prin un algoritm puțin mai complex decât animarea saniei, ca și funcționalitate suplimentară.

## 3.2 Modelul grafic

În ceea ce privește modelul grafic, am selectat diverse obiecte 3D sugestive de pe site-uri și le-am personalizat individual, căutând texturi potrivite și aplicându-le obiectelor în cadrul Blender-ului. Așa cum am precizat anterior, ground-ul a fost creat de mine, precum și skydome-ul, având ca resurse tutorialele furnizate de profesorul îndrumător.

## 3.3 Structuri de date

În afară de structurile de date furnizate de bibliotecile folosite, ca și structuri de date adiționale, am folosit un struct pentru sanie și pentru fulgul de zăpadă, amândouă având ca și componente position (poziția lor în scenă) și velocity (viteza deplasării pe x, respectiv pe y), ajutând la simularea mișcării lor. Pentru fulgii de zăpadă am folosit un vector de structuri. De asemenea, pentru artificii am folosit un struct Firework; pentru fiecare artificie avem o poziție inițială (initPosition), adică centrul cercului de unde își începe deplasarea; radius este raza cercului, active este flag-ul care indică dacă s-a activat artificia prin apăsarea tastei aferente, iar speed este direcția deplasării acesteia pe raza cercului.

```
1  struct Snowflake {
2      glm::vec3 position;
3      glm::vec3 velocity;
4  };
5
6  struct Sleigh {
7      glm::vec3 position;
8      glm::vec3 velocity;
9  };
10
11 const int numSnowflakes = 1000;
12 Snowflake snowflakes[numSnowflakes];
13 Sleigh mySleigh;
14
15
16 #define FIREWORKS_COUNT 18 // vor fi 18 pozitii pentru artificii
17 #define FIREWORK_QUARTER_SPLIT 6 // per sfert de cerc se vor imparti in 6 directii
18 #define FIREWORK_DIRECTIONS FIREWORK_QUARTER_SPLIT * 4
19
```

```

20 struct Firework {
21     glm::vec3 initPosition;
22     glm::vec3 currentPositions[FIREWORK_DIRECTIONS];
23     glm::vec3 directions[FIREWORK_DIRECTIONS];
24     float radius;
25     bool active;
26     float speed;
27 };

```

### 3.4 Ierarhia de clase

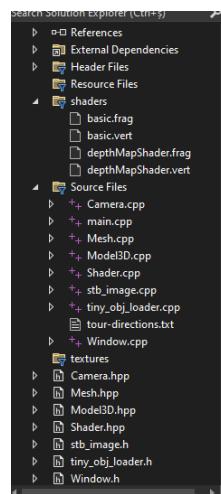


Figura 11: Ierarhia de clase

## 4 Prezentarea interfeței grafice utilizator / manual de utilizare

Pentru deplasarea în scenă cu ajutorul tastaturii se utilizează tastele:

- A - pentru deplasare la stânga
- D - pentru deplasare la dreapta
- W - pentru deplasare înainte
- S - pentru deplasare înapoi
- Q - rotește camera spre stânga
- E - rotește camera spre dreapta

Pentru diverse moduri de vizualizare:

- 1 - pentru modul de vizualizare wireframe
- 2 - pentru modul de vizualizare solid
- 3 - pentru modul de vizualizare punctiform

Alte funcționalități:

- J - pentru vizualizarea ninsorii
- Y - pentru vizualizarea animației (mișcarea saniei)
- F - pentru vizualizarea efectului de ceată
- M - pentru mișcarea camerei în scenă cu ajutorul mouse-ului (sus, jos, stânga, dreapta)
- L - pentru vizualizarea efectului de lumină globală

- P - pentru vizualizarea artificiilor
  - H - pentru înregistrarea pozițiilor în timpul deplasării în scenă (recording)
  - K - pentru vizualizarea animației de prezentare (turul automat al scenei)
- Funcționalitățile se dezactivează prin aceeași tastă.

## 5 Concluzii și dezvoltări ulterioare

Pentru mine, proiectul a fost un mod creativ de a mă familiariza cu multe concepte predate la curs și la laborator, motivându-mă să mă documentez mai mult pentru a-mi crea un proiect cât mai reușit și plăcut vizual. În plus, mi-a plăcut să analizez ideile pentru simularea animației componentelor precum ninsoarea și sania astfel încât să arate cât mai realist. Ca și dezvoltări ulterioare, mi-ar plăcea să reușesc să creez animații mai creative, precum alergarea unei pisici după sanie sau a omului de zăpadă, care să simuleze un mic salut. De altfel, mi-ar plăcea să introduc mai multe obiecte în scenă pentru un peisaj mult mai complex decât cel actual. Mi-ar plăcea să includ mai multe case, un parculeț și mai multe animăluțe. În plus, mi-ar plăcea să dezvolt mai mult animația artificiilor; aş putea să le scalez în funcție de distanța parcursă.

## 6 Referințe

1. Tutoriale Blender
2. Obiecte 3D