UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

# DOCUMENTAȚIE PROIECT
# PRELUCRARE GRAFICĂ

**STUDENT : SZEKELY KRIZTINA**
**GRUPA 30231**
**Profesor Îndrumător : Ing. Constantin Ioan Nandra**

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

# 1.Cuprins

# 2.Prezentarea Temei

Tema acestui proiect este reprezentarea fotorealistică a unei scene care conține diferite obiecte 3D, folosind biblioteca OpenGL. Utilizatorul va putea interacționa cu scena cu ajutorul mouse-ului (rotirea camerei) sau a tastaturii (deplasarea prin scena precum si manipularea diferitelor obiecte). In acest proiect, am decis să reprezint o scena dintr-un serial celebru de animație din Japonia.

# 3.Scenariul

## 3.1 Descrierea scenei și a obiectelor

Scena prezintă bătălia dintre protagonistul 'anime-ului', Naruto, și inamicul său, Pain sau Yamato. În prim plan se află Naruto, care este asistat de către o creatură mitică cu înfățișare de broască și Yamato, care îi are ca aliați pe cei șase indivizi ce reprezintă cele șase căi ale lui Pain, o tehnică prin care acesta poate controla șase corpuri diferite în același timp. Toată scena se petrece in satul ascuns între Frunze, acum distrus de către antagonist, de aceea se poate observa un aspect de crater, pe alocuri aflându-se ruine ale fostului sat. Mai spre marginea scenei, pe stânci se află alte două personaje importante, în spatele antagoniștilor putem vedea unul din personajele principale ale serialului, profesorul Kakashi care controlează o bilă de energie, iar în spatele lui Naruto, căpetenia satului, Tsunade, ranită în urma atacului personajelor negative alături de animalul ei de companie porcușorul Tonton. În partea din dreapta a scenei se află doi ninja cu care utilizatorul poate interacționa care în momentul în care se lovesc scot un sunet de săbii.

## 3.2 Funcționalități

Scena este iluminată de soare ( o iluminare direcțională care poate fi plasată în jurul scenei cu ajutorul tastelor "J" si "L"). Unul din cei doi ninja poate fi deplasat prin scena cu ajutorul sageților si al tastelor "< "și "> " iar celălalt se poate deplasa prin intermediul săgeților și tastelor "5" și "0"de pe tastatura numerica (numpad). Odată cu lansarea aplicației putem observa cum mingea pe care o controlează Pain, precum și cea pe care o controlează Kakashi sunt animate. Utilizatorul se poate deplasa prin scenă cu ajutorul tastelor (W, A, S, D) precum și mișcarea mouse-ului, mișscare pe care o putem vedea și în jocurile video. Prin simpla apăsare a tastei F, utilizatorul poate să controleze apariția ceței și prin apasarea simultană a tastei F și a butonului + sau – poate să schimbe intensitatea ceței. De asemenea, de pe tasta 9 putem porni lumina de tip local. De pe tastele 1,2,3 se poate vizualiza scena în cele trei moduri: wireframe,

polygonal respectiv solid. Pentru ca scena să fie fotorealistă, toate obiectele illuminate, au in spatele lor umbre.

# 4.Detalii de implementare

## 4.1 Funcții și algoritmi

### 4.1.1 Soluții posibile

Librăria OpenGL are o largă varietate de funcții cum ar fi glfwCreateWindow() cu ajutorul căreia putem să cream o fereastră pe care ne vom proiecta scena, glGenTexture(), glBindTexture(), glTextImage2D() cu ajutorul cărora putem să facem legătura dintre texturi și obiecte, precum și alte multe funcții.

Una din funcțiile esențiale ale acestui proiect este renderScene(), care este folosită pentru a trimite toate datele la shadere, dar în acelasi timp apelează funcția drawObjects() care proiecteaza obiectele în scena astfel incât utilizatorul să poată viziona scena. În funcția drawObjects() sunt apelate diferite funcții pentru a creea toate modelele, funcții care să animeze obiectele(translație, rotație), precum și simpla poziționare a acestora în scena. O altă funcție important este initUniforms() unde sunt calculate valori pentru reprezentarea iluminării (direcția luminii, culoarea etc). Funcția initShaders() este o funcție în care sunt instantiate toate shaderele, iar funcția initObjects() în care sunt instantiate toate modelele 3D(obiectele din scenă). Cu ajutorul funcțiilor processMovement(), mouseCallback() și keyboardCallback() s-a putut realiza conecțiunea dintre utilizator și aplicație, funcția făcând legătura dintre tastatură și diferitele componente ale scenei sau mouse și cameră. Funcția mouseCallback() a fost implementată utilizând unghiurile lui Euler. (Camera. (1)) Pentru fundal a fost folosit un skybox.

### 4.1.2 Motivarea abordării alese

Pentru implementarea funcționalităților prezentate și în lucrările de laborator am folosit îndrumătorul de laborator, însă au fost adaugate și cateva acțiuni care nu au fost studiate in laborator. De exemplu, am încercat sa implementez o funcție collisionDetect() pentru detectarea coliziunilor dintre cei doi ninja care atunci cand se apropie la mai mult de o unitate vor scoate un sunet de sabie si se vor depărta cu 3.7 unități. Altă operație pe care am implementat-o este cea de mișcare a celor doi ninja gandită în așa fel încât atunci când este apasat unul dintre butoanele care controleaza caracterele poziția acestuia sa fie schimbată cu 0.2 dar caracterul să nu poată ieși din scenă.

Pentru previzualizarea scenei am ales sa fac o funcție previewFunction() care atunci când apăsăm tasta Z să pornească rotația camerei în jurul scenei în care să se poate vizualiza toate personajele, pe fundal va începe  o melodie din serial și cei doi ninja se vor mișca pentru a face o demonstrație a funcționalității lor. O altă implementare poate fi considerate animarea bilelor de energie și foc. Rotația bilelor de foc sunt generate random folosind funcția rand().

Am ales să folosesc un program software adițional numit Blender pentru a pune obiectele care nu sunt animate în scenă, deoarece am considerat că dacă fac un fișier de tip .obj cu întreaga

scenă scad din inutilitatea unui cod scris în plus. De asemenea, am făcut schimbări la obiectele descărcate de pe internet pentru a face scena mai realistă și mai dinamică.

## 4.2 Modelul grafic

Obiectele din scena sunt modele 3D, cu extensia .obj care au fost preluate de pe diferite site-uri cum ar fi : ( (2)free3d, n.d.) ( (3)turbosquid, n.d.) ( (4)sketchfab, n.d.). Scena a fost construită in Visual Studio 2019. Pământul a fost modelat de către mine, la fel și schimbarea poziției obiectelor 3D luate de pe internet pentru a putea exemplifica scena pe care am dorit să o prezint.

## 4.3 Structuri de date

În fragment shader au fost implementați algoritmi de calculare a luminii, respectiv a umbrei, folosindu-se structuri de date specifice.

Umbrele adaugă un grad de realism scenelor noastre OpenGL, permițând, de asemenea, o percepție mai profundă a adâncimii. Există o mulțime de tehnici moderne care pot fi folosite pentru a produce umbre (calculul umbrelor) și toate acestea pot fi implementate în OpenGL. Cu toate acestea, nu există algoritmi perfecți în timp real pentru acest lucru, fiecare având propriile sale avantaje și dezavantaje. Pentru acest laborator, vom explora o tehnică ce oferă rezultate decente și este ușor de implementat, și anume shadow mapping (“hărți de umbre”). Shadow mapping este o tehnică multi-trecere care utilizează texturi de adâncime pentru a decide dacă un punct se află în umbră sau nu. Cheia este aceea de a observa scena din punctul de vedere al sursei de lumină în loc de locația finală de vizionare (locația camerei). Orice parte a scenei care nu este direct observabilă din perspectiva luminii va fi în umbră.

Luminile punctiforme sunt surse de lumină cu o poziție dată care iluminează radial și uniform în toate direcțiile. Spre deosebire de luminile direcționale, razele generate de luminile punctiforme se estompează în funcție de distanță, făcând astfel obiectele mai apropiate de sursă să pară mai iluminate decât obiectele mai îndepartate.

Atunci când se utilizează luminile punctiforme, direcția luminii nu este constantă între fragmente și trebuie calculată pentru fiecare fragment diferit:

```
//compute light direction

vec3 lightDirN = normalize(lightPosEye - fragPosEye.xyz);
```

Pentru a reduce intensitatea luminii punctiforme cu distanța, trebuie să aplicăm un coeficient de atenuare. Atenuarea poate fi calculată ca o funcție liniară sau patratică dependentă de distanță. Atenuarea liniară necesită un calcul mai puțin complex, dar oferă mai puțin realism,

deoarece, în lumea reală, sursele de lumină sunt strălucitoare atunci când stau aproape de obiecte, dar își diminuează rapid strălucirea, cu cât ne aflăm mai departe de ele. Realismul este considerabil crescut atunci când se încorporează atenuarea patratică în funcție de distanță. Astfel, atenuarea poate fi calculată folosind:

$$Att = 1.0 / (Kc + Kl * d + Kq * d2)$$

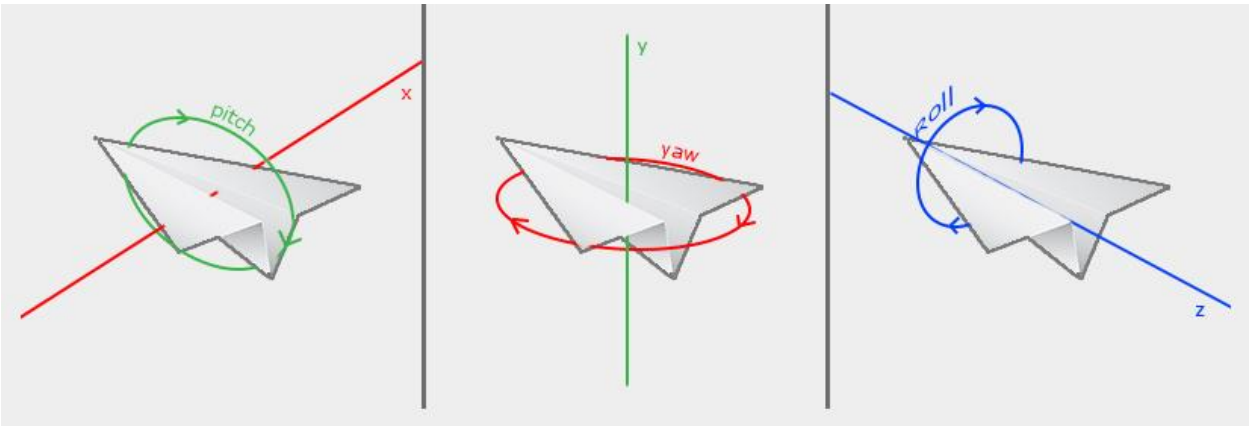
unde Kc este un termen constant, de obicei menținut la 1.0, Kl este un termen liniar, Kq este un termen pătratic și d este distanța.

Pentru miscarea camerei in scena am folosit formulele legate de pozitie si directive. Am create o matrice LookAt de urmatorul tip :

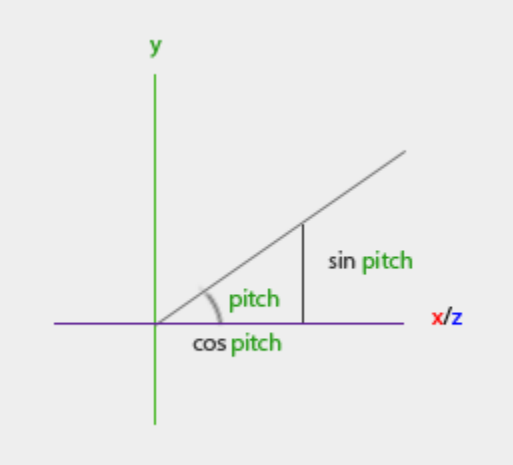
$$LookAt = \begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Unde R este vectorul Dreapta, U este vectorul sus, D este directia vectorului si P este pozitia camerei. Pentru miscarea camerei schimbam coordonatele vectorului LookAt.

Pentru rotirea camerei am folosit unghiurile Euler : pitch si yaw.




Pitch este unghiul care detecteaza cat de mult ne uitam in sus sau in jos. Valoarea lui yaw reprezinta magnitudinea a cum ne uitam inspre stanga sau dreapta.




Putem să creștem realismul unei scene 3D prin adăugarea unui efect de ceață care va crea o atmosferă specială. Prin manipularea atributelor asociate efectului de ceață putem să personalizăm atmosfera și, de asemenea, să ameliorăm percepția de adâncime (axa Z). Culoarea de fundal trebuie aleasă în funcție de culoarea efectului de ceață.

*glClearColor(0.5, 0.5, 0.5, 1.0);*

Prima metodă de calculare a ceții este prin folosirea unei funcții de interpolare liniară care va amesteca culoarea ceții cu aceea a fragmentului, pe baza distanței fragmentului (calculată în spațiul de vizualizare).

$$fogFactor = (fogEnd - fragmentDistance) / (fogEnd - fogStart)$$

Valorile factorului de ceață trebuie să fie între 0.0 and 1.0.

Un rezultat mai bun poate fi obținut luând în considerare reducerea intensității luminii în funcție de distanță. Factorul de atenuare care va fi folosit reprezintă densitatea de ceață, această densitate fiind constantă în toată scena. Rezultatul este o scădere rapidă a factorului de ceață în comparație cu abordarea liniară.

$$fogFactor = e^{-fragmentDistance * fogDensity}$$

Formula este practic aceeași cu cea precedentă, singura diferență provinind din faptul că am ridicat la pătrat distanța fragmentului și densitatea de ceață.

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

## 4.4 Ierarhia de clase

Ierarhia de clase a proiectului este definită prin diferitele librării. De exemplu, Mesh.cpp și Models3D.cpp sunt folosite pentru definirea obiectelor precum și pentru aplicarea de texturi asupra acestora. Cu ajutorul Shader.cpp se vor putea compila și lega shaderele existente de

program. Camera.cpp este locul în care sunt definite funcțiile ce se aplică asupra camerei ( mișcarea acesteia și funcția preview). SkyBox.cpp se folosește la proiectarea skybox-ului.

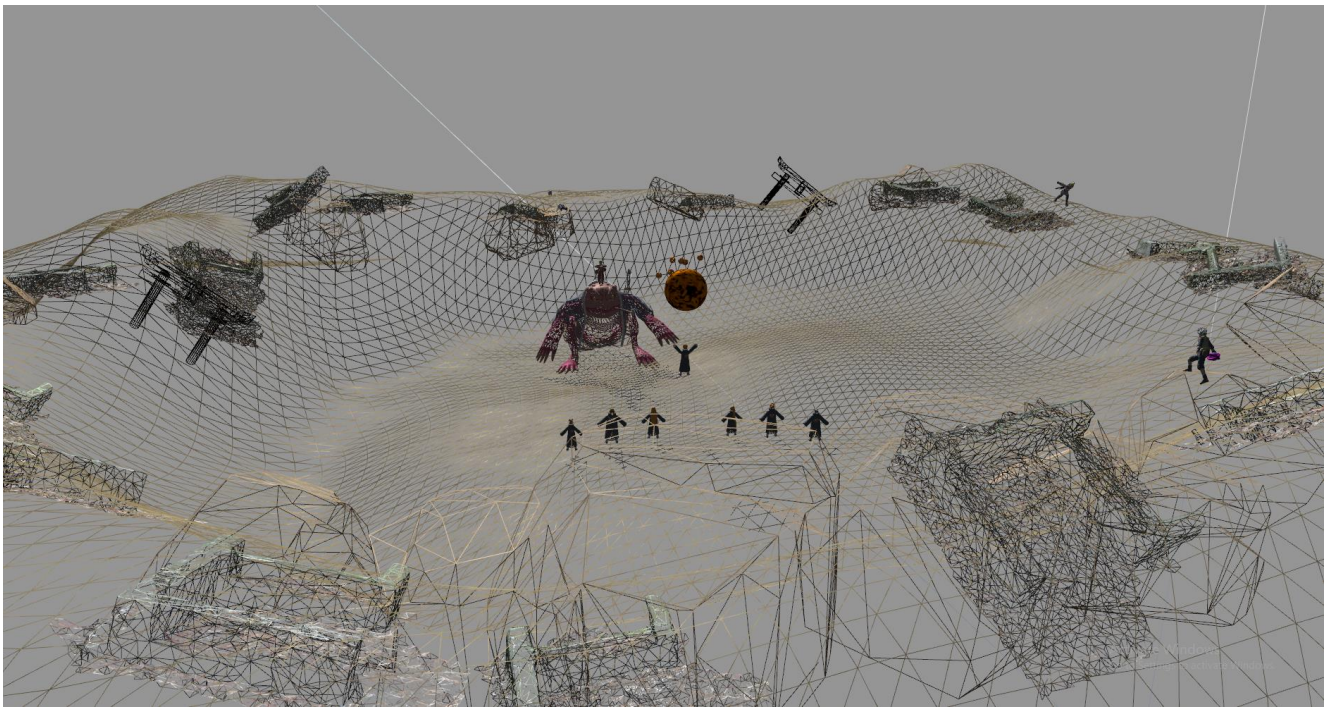
# 5. Prezentarea interfeței utilizator / manual de utilizare

În următoarea secțiune voi atașa imagini din diferite unghiuri ale scenei care să facă mai ușoară înțelegerea tuturor datelor prezentate mai sus.


Activate Windows
Go to Settings to activate Windows.


Activate Windows
Go to Settings to activate Windows.


Activate Windows



Utilizatorul poate efectua următoarele funcții în scena, sau pe obiectele din scena folosind următoarele taste de la tastatură:

- W – deplasarea camerei in fata
- A – deplasarea camerei la stanga
- S – deplasarea camerei in spate
- D – deplasarea camerei la dreapta
- Sageata sus – deplasarea ninja 1 in fata
- Sageata jos – deplasarea ninja 1 in spate
- Sageata stanga – deplasarea ninja 1 la stanga
- Sageata dreapta – deplasarea ninja 1 la dreapta
- < - deplasare ninja 1 in sus pe axa y
- > - deplasare ninja 1 in jos pe axa y
- Sageata sus (8 pe numpad) – deplasare ninja 2 in fata
- Sageata jos (2 pe numpad) – deplasare ninja 2 in spate
- Sageata dreapta (6 pe numpad) – deplasare ninja 2 la dreapta
- Sageata stanga (4 pe numpad) – deplasare ninja 2 la stanga
- Tasta 5 pe numpad – deplasare ninja 2 in sus pe axa y
- Tasta 0 pe numpad – deplasare ninja 2 in jos pe axa y
- F – pornirea efectului de ceata
- G – oprire effect de ceata
- F + “+” sau “-“ – intetirea sau raritatea cetei
- 1 – Start mod wireframe
- 2 – Start mod polygonal
- 3 – Start mod solid
- J si L – rotirea sursei de lumina principale
- Z – start preview
- X – stop preview
- Q – rotirea camerei la dreapta
- E – rotirea camerei la stanga
- 9 – start lumina punctiforma
- 0 – stop lumina punctiforma
- I – afisare coordonate camera

# 6. Concluzii și dezvoltări anterioare

OpenGL este o librarie foarte vastă care poate fi folosită în nenumărate moduri, însă datorită evoluției rapide a tehnologiei este mult mai rar folosită ca acum 10 ani. Acum se

folosesc în industria jocurilor video anumite programe software denumite engine-uri care sunt mult mai dezvoltate din toate punctele de vedere și fac interacțiunea utilizatorului cu produsul sa fie mult mai aproape de realitate (Vezi Unreal Engine 5). De asemenea, se ușurează activitatea programatorului foarte tare datorită interfeței prietenoase. Deși OpenGL stă la baza acestor programe sigur va deveni din ce în ce mai nefolosită, însă este o modalitate bună de a învăța bazele programării graficii 3D.

În cazul de față, scena poate fi dezvoltată în multe alte moduri precum implementarea ploii, a focului, animarea diferitelor personaje sau chiar implementarea unui mini joculeț în care personajul principal poți fi chiar tu!

# 7.Referinte

1. (n.d.). Retrieved from https://learnopengl.com/Getting-started/Camera

2. (n.d.). Retrieved from http://turbosquid.com/

3. (n.d.). Retrieved from http://sketchfab.com/

4. (n.d.). Retrieved from https://free3d.com/