



Scena 3D interactivă

Prelucrare grafică

Autor: Volcov Sabina

Grupa: 30231

FACULTATEA DE AUTOMATICĂ
și CALCULATOARE

15 ianuarie 2024

Cuprins

1	Prezentarea temei	2
2	Scenariul	2
2.1	Scena de obiecte	2
2.2	Functionalitati	3
3	Detalii de implementare	4
3.1	Functii si algoritmi	4
3.2	Modelul grafic	6
3.3	Structuri de date si ierarhia de clase	7
4	Manual de utilizare	9
5	Concluzii si dezvoltari ulterioare	10

1 Prezentarea temei

Proiectul presupune la crearea si incarcarea unei scene de obiecte 3D in Visual Studio cu ajutorul librariilor OpenGL care interactioneaza direct cu placa grafica, dupa care utilizatorul o poate explora dinamic folosind mouse-ul si tastatura. Fiecare obiect este texturat, iar lumina provine de la mai multe surse de lumina pozitionale. Scena este plasata intr-un cadru medieval in aer liber, cu cateva elemente antice.

2 Scenariul

Initial, utilizatorul se afla chiar in centrul scenei, dupa care se poate deplasa folosind tastele A, W, S, D si privi in jur apasand butonul stang al mouse-ului si rotind camera. Daca se apasa tasta "O" de oriunde din scena, va incepe un tur automat al scenei, incepand din centrul acestia si deplasandu-se cativa pasi in stanga si inainte pana se ajunge intr-o alta zona. Daca se apasa repetat tasta "O", utilizatorul va reveni in centrul scenei si turul automat va merge de data aceasta cativa pasi in dreapta, pana se ajunge intr-o alta zona a aplicatiei.

2.1 Scena de obiecte

Scena reprezinta un cadru medieval, combinat cu elemente antice, fara a determina acuratete geografica. Centrul scenei este dominat de o piata medievala cu mai multe tarabe pe care se gasesc lazi cu cartofi, portocale, rosii si morcovi, cosuri cu grane, o masa cu carne si o alta cu cascavaluri. Putin mai in spate se gasesc niste carute incarcate cu fan sau butoaie, o fontana si niste lemn. In fata pietii este plasat un caine.

In partea stanga a scenei, in fata utilizatorului se pot observa ruinele unui templu grecesc, iar si mai departe, un orasel medieval. Casele sunt organizate in cerc, iar in mijlocul acestui cerc este plasata o fontana si o caruta cu fan, care poate fi deplasata de la tastatura.

Privind in spate din orasel, utilizatorul poate vedea niste stanci, dupa care urmeaza un ocean pe care se pot vedea plutind 3 barci.

In partea dreapta a scenei, din pozitia centrala a scenei, in fata se vede un templu grecesc intact si o statuie a zeitatii de care tine (zeita Atena). In spate, se gaseste o alta fontana si o padure, unde ca easter egg utilizatorul poate intalni o faptura mitologica, si anume Minotaurel.

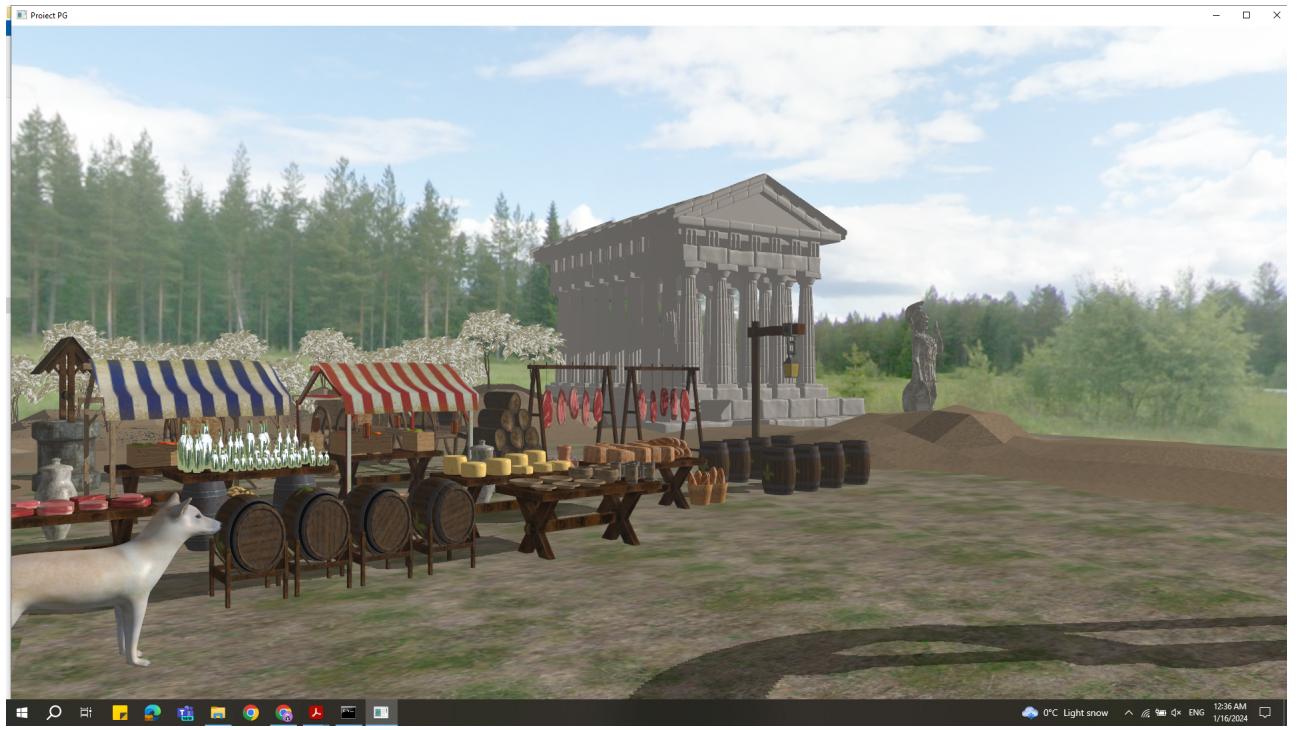


Fig. 1: Captura de ecran la deschiderea aplicatiei

2.2 Functionalitati

Aplicatia este interactiva, permitand folosirea tastaturii si mouse-ului pentru navigarea in scena. Utilizatorul se poate deplasa inainte, inapoi, la dreapta, la stanga si privi in jur.

De asemenea, exista optiunea de tur automat al scenei de la tasta "O", care duce pe rand utilizatorul in 2 zone diferite, dar cheie, ale scenei. Daca se ajunge in mijlocul oraselului, apasand repetat tasta "U", se va deplasa caruta cu fan inainte.

Scena de obiecte este implementata cu un efect de ceata de densitate 0.05, astfel incat lucrurile indepartate de pozitia de vizualizare apar mai neclare decat cele apropiate. Acest lucru se poate verifica la deplasarea in spate sau in sus din centrul scenei.

Lumina provine de la mai multe surse punctiforme, printre care una care functioneaza ca si soarele care lumineaza scena, iar restul reprezentand niste lampi. Pentru a vizualiza efectul acestor lumini, se poate apasa tasta "N", care ignora temporar lumina generata de soare.

Exista si posibilitatea de a activa si dezactiva ploaia, folosind tasta "R", in care caz 2000 de obiecte de forma unor picaturi de ploaie vor incepe sa cada pe pozitii aleatoare din scena, iar in momentul atingerii pamantului, vor reveni sus.

Utilizatorul are optiunea de a vizualiza scena in 4 moduri: solid (default), wireframe, poligonal si smooth. Pe de alta parte, nu se permite iesirea din scena, in momentul in care se atinge valoarea -5.5 sau 5.5 pentru coordonata x sau z a camerei, acestea nu se mai actualizeaza.

3 Detalii de implementare

3.1 Functii si algoritmi

Unul dintre algoritmii folositi in proiect este Shadow Mapping, pentru generarea dinamica a umbrelor. Acest algoritm presupune trasarea scenei de 2 ori, prima data efectuandu-se calculul umbrelor, iar la a doua iterare, randarea completa a scenei, cu tot cu umbre.

Ca prim pas, se seteaza ca punct de vizualizare pozitia luminii, dupa care se salveaza o harta de umbre pe baza informatiilor de adancime ale fiecarui fragment din scena. Aceasta harta este transformata si stocata intr-o textura.

In continuare, se plaseaza camera in pozitia originala a observatorului pentru rasterizarea finala a scenei. Pentru a decide daca un fragment e umbrit sau nu se compara adancimea fiecarui fragment cu cea stocata in harta de adancime. O valoare mai mare inseamna umbra, iar una mai mica - lumina directa.

Un alt algoritm este cel utilizat la calculul cettii exponentiale. Acesta presupune reducerea intensitatii luminii in functie de distanta dintre fragment si lumina. Atenuarea este reprezentata de densitatea de ceata, constanta in toata scena.

Dupa calculul factorului de ceata, se calculeaza culoarea fragmentului amestecand culoarea cettii cu culoarea fragmentului, calculata anterior conform cu sursele de lumina specificate in scena.

```
1 fColor = fogColor * (1 - fogFactor) + color * fogFactor;
```

Picaturile de ploaie sunt niste obiecte animate, generate aleator in numar de 2000, la initializarea aplicatiei. Cand se activeaza de la tastatura ploaia, picaturile incep sa cada cu viteza constanta, iar cand ating planul de pamant sunt mutate inapoi in sus, pe o pozitie aleatorie din scena.



Fig. 2: Efect de ploaie

Importante in realizarea proiectului sunt si rasterizarea obiectelor transparente si partea de fragment discarding, necesara la rasterizarea frunzelor pe copaci. Prima parte a fost implementata in interiorul fragment shader-ului, tinand cont de reflectiile de la skybox, iar a doua prin verificarea coordonatei alpha a texturii. Daca alpha e mai mic decat 1 pentru un anumit fragment, acesta este eliminat, altfel va fi rasterizat in mod obisnuit.

Diferenta se poate observa in urmatoarele imagini:



Fig. 3: Coroana de frunze fara fragment discarding

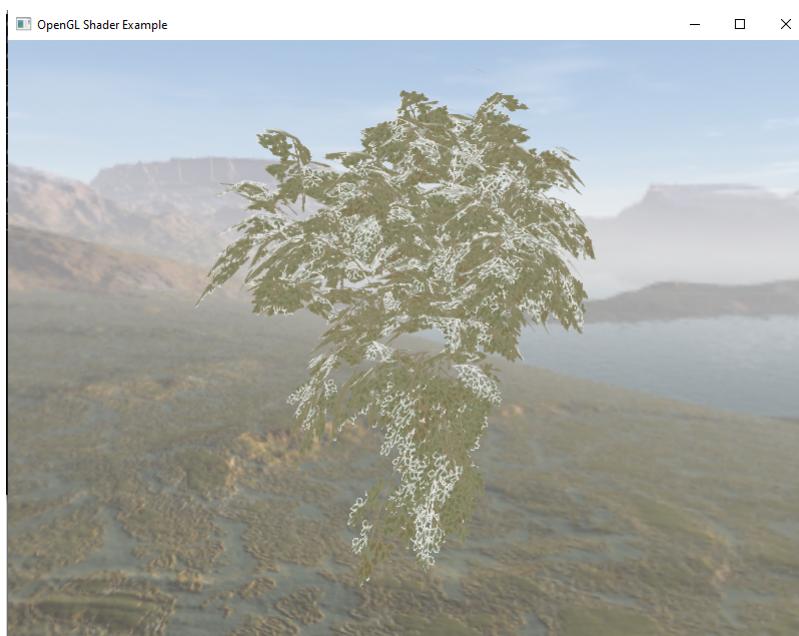


Fig. 4: Coroana de frunze cu fragment discarding

Obiectele transparente (sticlele) au fost modelate cu reflectii, dar si profitand de optiunea de rasterizare cu alpha blending in shader. Am folosit un factor de 0.5 pentru a reprezenta transparenta.



Fig. 5: Sticle transparente si reflectante

Miscarea camerei se face actualizand cei 3 vectori care formeaza sistemul de coordonate camera: cameraFront, cameraRight si cameraUp. In calcul se foloseste parametrul cameraSpeed, care determina viteza de miscare a vizualizatorului in scena. Am ales valoarea de 0.05f pentru viteza, pentru a nu merge nici prea rapid, dar nici prea lent.

Pentru a actualiza pozitia camerei, se calculeaza doi vectori noi, si anume cameraDirection (cameraTarget - cameraPosition) si cameraRight (cross(cameraFront, cameraUp)). Daca observatorul se deplaseaza inainte, la pozitie si target se aduna cameraSpeed * cameraPosition, aceeasi valoare se scade la deplasarea inapoi, iar daca se merge la dreapta se aduna cameraSpeed * cameraRight, sau scade pentru a merge la stanga.

Rotatia camerei are loc folosind functiile trigonometrice sin si cos si 2 unghiuri, yaw si pitch, care specifica rotatie pe axa Ox, respectiv Oy.

```

1 void Camera::rotate(float pitch, float yaw) {
2     glm::vec3 direction;
3     direction.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
4     direction.y = sin(glm::radians(pitch));
5     direction.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));
6     this->cameraTarget = glm::normalize(direction);
7 }
```

3.2 Modelul grafic

Proiectul are drept scop realizarea unei prezentari fotorealiste a unor scene de obiecte 3D utilizand librariile OpenGl, GLFW si GLM, astfel incat aplicatia respecta pipeline-ul grafic clasic pentru a rasteriza scena.

Obiectele definite in coordonate globale trec prin vertex shader, apoi dupa operatii intermediere de transformare (de vizualizare si proiecție) prin fragment shader, ca în final să poată fi rasterizati pixelii de pe ecran cu culorile calculate pentru fiecare fragment.

Proiectul foloseste modelul de iluminare Blinn-Phong pentru simularea surselor de lumina, model care se refera la calculul componentelor ambientala, difusa si speculara pentru un anumit fragment care urmeaza a fi rasterizat. Blinn-Phong functioneaza pentru lumini pozitionale, care au o anumita culoare.

```

1 void computeLightComponents()
2 {
3     vec3 cameraPosEye = vec3(0.0f); //in eye coordinates, the viewer
4     //is situated at the origin
5
6     //transform normal
7     vec3 normalEye = normalize(fNormal);
8
9     //compute light direction
10    vec3 lightDirN = normalize(lightDir);
11
12    //compute view direction
13    vec3 viewDirN = normalize(cameraPosEye - fPosEye.xyz);
14
15    //compute ambient light
16    ambient = ambientStrength * lightColor;
17
18    //compute diffuse light
19    diffuse = max(dot(normalEye, lightDirN), 0.0f) * lightColor;
20
21    //compute specular light
22    vec3 reflection = reflect(-lightDirN, normalEye);
23    float specCoeff = pow(max(dot(viewDirN, reflection), 0.0f), shininess);
24    specular = specularStrength * specCoeff * lightColor;
25 }
```

3.3 Structuri de date si ierarhia de clase

Proiectul este structurat pe mai multe foldere, cel principal cu codul sursa C++ al proiectului, iar folderele objects, shaders si skybox necesare pentru stocarea resurselor necesare rularii proiectului.

Folderul glm contine biblioteca glm, necesara pentru anumite functii care se apeleaza in codul principal al aplicatiei.

Resursele din folder-ul skybox sunt imaginile care se mapeaza in jurul scenei de obiecte ca fundal, iar cele din folder-ul shaders sunt fisierele .vert si .frag care determina operatiile prin care trec toate varfurile fiecarui obiect de rasterizat din scena.

glm	12/20/2023 6:18 PM	File folder	
objects	1/15/2024 10:01 PM	File folder	
shaders	1/15/2024 11:25 PM	File folder	
skybox	1/15/2024 11:22 PM	File folder	
x64	1/15/2024 1:22 PM	File folder	
++ Camera.cpp	1/15/2024 11:18 PM	C++ Source	8 KB
Camera.hpp	1/4/2024 1:43 PM	C/C++ Header	2 KB
++ main.cpp	1/15/2024 11:25 PM	C++ Source	32 KB
++ Mesh.cpp	11/4/2023 11:03 AM	C++ Source	3 KB
Mesh.hpp	12/5/2023 12:28 PM	C/C++ Header	2 KB
++ Model3D.cpp	1/15/2024 12:38 PM	C++ Source	8 KB
Model3D.hpp	1/15/2024 12:38 PM	C/C++ Header	2 KB
proiect_PG	1/15/2024 11:26 PM	Application	590 KB
proiect_PG.sln	12/20/2023 6:17 PM	Visual Studio Solu...	2 KB
proiect_PG.vcxproj	12/20/2023 6:19 PM	VC++ Project	9 KB
proiect_PG.vcxproj.filters	12/20/2023 6:19 PM	VC++ Project Filte...	3 KB
proiect_PG.vcxproj.user	12/20/2023 6:17 PM	Per-User Project O...	1 KB
++ Shader.cpp	10/25/2023 5:21 PM	C++ Source	3 KB
Shader.hpp	11/18/2023 11:26 AM	C/C++ Header	1 KB
++ SkyBox.cpp	12/12/2023 12:34 PM	C++ Source	5 KB
SkyBox.hpp	12/12/2023 12:34 PM	C/C++ Header	1 KB
++ stb_image.cpp	11/3/2021 3:28 PM	C++ Source	1 KB
stb_image.h	11/3/2021 3:28 PM	C/C++ Header	267 KB
++ tiny_obj_loader.cpp	10/25/2020 1:31 PM	C++ Source	1 KB
tiny_obj_loader.h	10/29/2016 2:01 PM	C/C++ Header	59 KB

Fig. 6: Ierarhia de fisiere a proiectului

Folder-ul objects contine toate obiectele care apar in scena. Fiecare astfel de obiect are format .obj si este legat de un fisier .mtl care specifica materialele (texturile) necesare incarcarii acestuia. Obiectele cube si quad, desi nu sunt obligatorii pentru afisarea scenei, au un scop functional. Cube este cubul de lumina care apare la apasarea tastei "C" pe pozitia fiecarei lumini punctiforme, iar quad devine necesar la calcularea umbrelor in scena de obiecte.

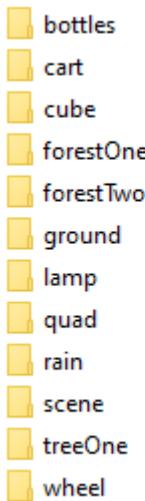


Fig. 7: Structura folder-ului de obiecte

Am declarat ca structura de date noua Raindrop, care include o pozitie si viteza de cadere a picaturilor de ploaie. La initializarea aplicatiei, se genereaza 2000 de astfel de obiecte cu pozitii aleatoare, dupa care atat timp cat ploaia este activata, obiectele cad (y-ul pozitiei se decrementeaza conform cu timpul scurs de la decrementarea precedenta) si cand ating pamantul revin in pozitii de sus aleatoare.

4 Manual de utilizare

Dupa pornirea aplicatiei, utilizatorul se afla in mijlocul scenei, de unde poate explora obiectele din jur.

Pentru a roti camera, se tine apasat butonul stang al mouse-ului si se trage scena in directia de rotatie dorita. Rotatia poate fi si completa, pozitia putand fi observata 360*.

Pentru a se deplasa inainte, se apasa tasta "W", iar inapoi, tasta "S". Pentru miscare la stanga se apasa tasta "A", iar la dreapta, tasta "D". Exista si posibilitatea de a roti scena cu orice unghi, apasand tasta "Q" (rotatie spre stanga) sau "E" (rotatie spre dreapta).

Lumina globala (soarele) poate fi miscata de la tastele "L" si "J" si se poate observa efectul acestei miscari prin deplasarea umbrelor in scena.

Pentru a comuta intre vizualizarea in mod solid si in mod wireframe, se poate folosi tasta "T".

Tasta "P" comuta intre vizualizarea in mod solid si mod poligonal, iar tasta "H" intre vizualizarea in mod solid si mod smooth (identic cu modul solid).

Daca se apasa tasta "O", se activeaza animatia de prezentare a scenei.

Pentru a vizualiza harta de umbre care se va aplica pe obiecte, se foloseste tasta "M".

Tasta "U" permite vizualizarea animatiei pe un singur obiect, si anume deplasarea carutei din mijlocul oraselului inainte, inclusiv cu rotirea rotilor.

Activarea si dezactivarea ploii se face cu tasta "R", iar activarea si dezactivarea luminilor punctiforme - cu tasta "N". Pentru vizualizarea mai clara a acestora din urma, apasand tasta "C" in scena se vor randa cuburi albe in jurul pozitiilor de unde provine lumina.

5 Concluzii si dezvoltari ulterioare

Asadar, in urma realizarii acestui proiect se poate observa ca procesorul GPU este mult mai potrivit pentru rasterizarea unor imagini fotorealistice decat procesorul CPU. Combinatia bibliotecilor OpenGL impreuna cu mecanismul de hardware acceleration demonstreaza avantaje incomparabile ale procesorului grafic fata de rasterizarea traditionala. GPU permite procesarea paralela a fragmentelor, astfel incat un volum mai mare de date poate fi analizat mai rapid si mai eficient.

Pe de alta parte, calculele matematice si executia pipeline-ului grafic sunt simplificate de folosirea functiilor deja implementate, cum ar fi glCullFace, dot, cross sau structuri de date precum vec3, vec4 si mat4.

Dezvoltari ulterioare ale proiectului ar include imbunatatirea luminilor punctiforme (la moment nu e foarte clar de unde provin), corectarea unor texturi care nu se mapeaza perfect pe obiecte, introducerea mai multor animatii sau adaugarea unor umbre pentru obiectele transparente.

Referințe

- [1] *Laboratorul 12/PG - Fog and fragment discarding.*
- [2] *Laboratorul 6/PG - Texturi și animatii simple.*
- [3] *Laboratorul 8/PG - Modelul de iluminare Blinn-Phong.*
- [4] *Laboratorul 9/PG - Shadow Mapping.*
- [5] *Multiple lights in OpenGL*, URL: <https://opentk.net/learn/chapter2/6-multiple-lights.html>.