Universitatea Nationala de Stiinta si Tehnologie Politehnica Bucuresti
Facultate: Automatica si Calculatoare
Program de masterat: Grafica, Multimedia si Realitate Virtuala

Raport Proiect "Motoare de Grafica 3D in Timp Real"

Marian-Paul Lupuleasa

Sl. dr. ing. Victor-Iosif Asavei

Bucharest
2025

CONTENTS

1 INTRODUCERE

1.1 Descrierea Soluției

S-a dorit implementarea unui motor grafic cu funcționalități de bază, ce permite utilizatorului:

- Să populeze scena cu diferite obiecte cu ajutorul unor Mesh-uri

- Să le aplice transformări 3D pentru a le modifica poziția și dimensiunea

- Să le aplice diferite texturi prin importul unor fișiere .png.

- Să le aplice diferite materiale pentru a observa interacțiunea cu o sursă de lumină prezenta în scena

- Să se plimbe prin scenă cu ajutorul unei camere ce permite mișcarea pe cele 3 axe de coordonate și care poate fi rotită în orice direcție

1.2 Tehnologii folosite

Aplicația a fost implementată folosind limbajul de programare **C++** și biblioteca grafică **OpenGL**, utilizându-se și bibliotecile **GLFW** și **GLAD** pentru anumite funcționalități standard.

2 FUNCȚIONALITĂȚI DE BAZĂ

2.1 Clase

2.1.1 Vec2

S-a implementat clasa **Vec2**, în fișierul **vec2.h**, aceasta având următoarele funcționalități:

- Constructorul default **(0, 0)**

- Constructor cu valori date **(x, y)**

- Operații pe vector **(+, -, *, /)**

- **Magnitudinea** și **Normalizarea**

- **Clamping** intre 2 valori date

- **Distanța** intre 2 vectori

2.1.2 Vec3

- S-a implementat clasa **Vec3**, în fișierul **vec3.h**, aceasta având următoarele funcționalități:

- Constructorul default **(0, 0, 0)**

- Constructor cu valori date **(x, y, z)**

- Operații pe vector **(+, -, *, /)**

- **Produsul scalar (Dot product)** a 2 vectori

- **Produsul vectorial (Cross product)** a 2 vectori

- **Magnitudinea** și **Normalizarea**

- **Clamping** intre 2 valori date

- **Distanța** intre 2 vectori

- **Pointer** la primul element al vectorului

2.1.3 Mat4

- S-a implementat clasa **Mat4**, în fișierul **mat4.h**, aceasta având următoarele funcționalități:

- Constructor pentru matricea identitate (**1 pe diagonală**)

- Constructor cu valori date (de la x00 la x33)

- **Înmulțirea** a 2 matrici (*)

- Matricea **transpusă**

- Matricea **translatată** cu anumite coordonate

- Matricea **scalată** cu anumiți scalari

- Matricea **rotită** pe axa **Ox, Oy sau Oz**

- Matricea **perspectivă** (**Projection**)

- Matricea de **vizualizare** (**View**)

- **Pointer** la primul element al matricii

2.1.4 Mesh

S-a implementat clasa **Mesh**, în fișierul **mesh.h**, aceasta având următoarele funcționalități:

- Constructorul default ce conține vertecșii și indicii obiectului, materialul și textura. Acesta este utilizat pentru stocarea informațiilor legate de vertecși și indici, pentru a putea construi Mesh-ul, cu ajutorul obiectelor **VAO, VBO și EBO**.

- Destructorul în care se șterg **VAO, VBO și EBO**.

- Funcția **Draw** în care se trimit la shader toate informațiile necesare desenării Mesh-ului și apoi acesta este randat în scenă.

De asemenea, au fost implementate și 3 structuri pentru gestionarea componentelor Mesh-ului:

- O structură pentru **Vertex** (poziție, normală, coordonată de textură)

- O structură pentru **Material** (textura, componenta ambientală, componenta difuză, componenta specular și strălucirea)

- O structură pentru **Light** (poziția sursei de lumină, componenta ambientală, componenta difuză, componenta specular)

2.2 Import fișier .obj - Obiect 3D

Motorul grafic implementat poate importa fișiere în format .obj, din care se pot extrage informații despre poziția vertecșilor, normalele vertecșilor, coordonata de textura și cum trebuie sa se formeze fețele obiectului:

```
v  -0.5000 -0.5000 0.5000
v  -0.5000 -0.5000 -0.5000
v  0.5000 -0.5000 -0.5000
v  0.5000 -0.5000 0.5000
v  -0.5000 0.5000 0.5000
v  0.5000 0.5000 0.5000
v  0.5000 0.5000 -0.5000
v  -0.5000 0.5000 -0.5000

vn 0.0000 -1.0000 -0.0000
vn 0.0000 1.0000 -0.0000
vn 0.0000 0.0000 1.0000
vn 1.0000 0.0000 -0.0000
vn 0.0000 0.0000 -1.0000
vn -1.0000 0.0000 -0.0000

vt 1.0000 0.0000 0.0000
vt 1.0000 1.0000 0.0000
vt 0.0000 1.0000 0.0000
vt 0.0000 0.0000 0.0000

f 1/1/1 2/2/1 3/3/1 4/4/1
f 5/4/2 6/1/2 7/2/2 8/3/2
f 1/4/3 4/1/3 6/2/3 5/3/3
f 4/4/4 3/1/4 7/2/4 6/3/4
f 3/4/5 2/1/5 8/2/5 7/3/5
f 2/4/6 1/1/6 5/2/6 8/3/6
```

De exemplu, pentru un obiect de tip cub, vom avea fișierul de mai sus:

- v = poziția vertecșilor (8 vertecși)

- vn = normalele vertecșilor (6 normale, 2 pe fiecare axa)

- vt = coordonatele de textură

- f = fețele (în acest caz de tip quad)

Se parcurge fișierul și se salvează informațiile specifice fiecărei componente într-un vector.

Când se ajunge la fețe:

- dacă fața are 3 coordonate, este vorba de un triunghi

- dacă fața are 4 coordonate, este vorba de un quad și se tratează ca 2 triunghiuri

Se creează vertecșii cu ajutorul poziției, normalelor și a coordonatelor de textură, și apoi se populează vectorii de vertecși și indici.

La finalul procesului cei 2 vectori vor conține toate informațiile necesare creării Mesh-ului.

Implementarea este funcțională doar dacă fețele au 3 sau 4 coordonate. În cazul cilindrului prezent în scena, capetele acestuia au 32 de coordonate și, ca urmare, nu vor fi create.

2.3 Import imagini - Texturare

Motorul grafic poate importa imagini în diferite formate (.png, .jpg), care pot fi aplicate asupra Mesh-urilor, sub forma de texturi.

Utilizând imaginea dată ca parametru, se creează un textureID care este trimis la shader pentru a aplica textura asupra Mesh-ului.

2.4 Camera

S-a implementat o cameră ce poate fi mișcată pe toate cele 3 axe, apăsând pe butonul drept al mouse-ului și pe una dintre tastele următoare:

- A/D, pentru mișcarea Left/Right, pe axa Ox

- E/Q, pentru mișcarea Up/Down, pe axa Oy

- W/S, pentru mișcarea Forward/Backward, pe axa Oz

De asemenea, tot prin apăsarea butonului drept al mouse-ului, se poate roti camera în orice direcție.

În fișierul **camera.h**, s-a implementat clasa **Camera** care conține toate informațiile necesare pentru a poziționa, mișca și roti camera în funcție de inputul utilizatorului.

Pentru modificarea poziției, s-au folosit variabilele front, right, worldUp și up, pentru calcularea poziției pe toate cele 3 axe.

Pentru modificarea rotației, s-au folosit variabilele yaw, pentru rotația pe axa Ox, și pitch, pentru rotația pe axa Oy. Pentru a se evita răsturnarea camerei, pe axa Oy camera se poate roti doar în intervalul (-90,90).

2.5 Iluminare

S-a implementat iluminarea în fragment shader, folosindu-se o sursă de lumină și materialul aplicat asupra obiectului. S-au calculat cele 3 componente ale luminii: lumina ambientală, lumina difuză și lumina specular, prin combinarea componentelor sursei de lumină și ale materialului. Apoi, cele 3 s-au combinat în într-o valoare finală care, la rândul ei, a fost combinată cu textura aplicată obiectului pentru a se simula efectul luminii asupra obiectului.

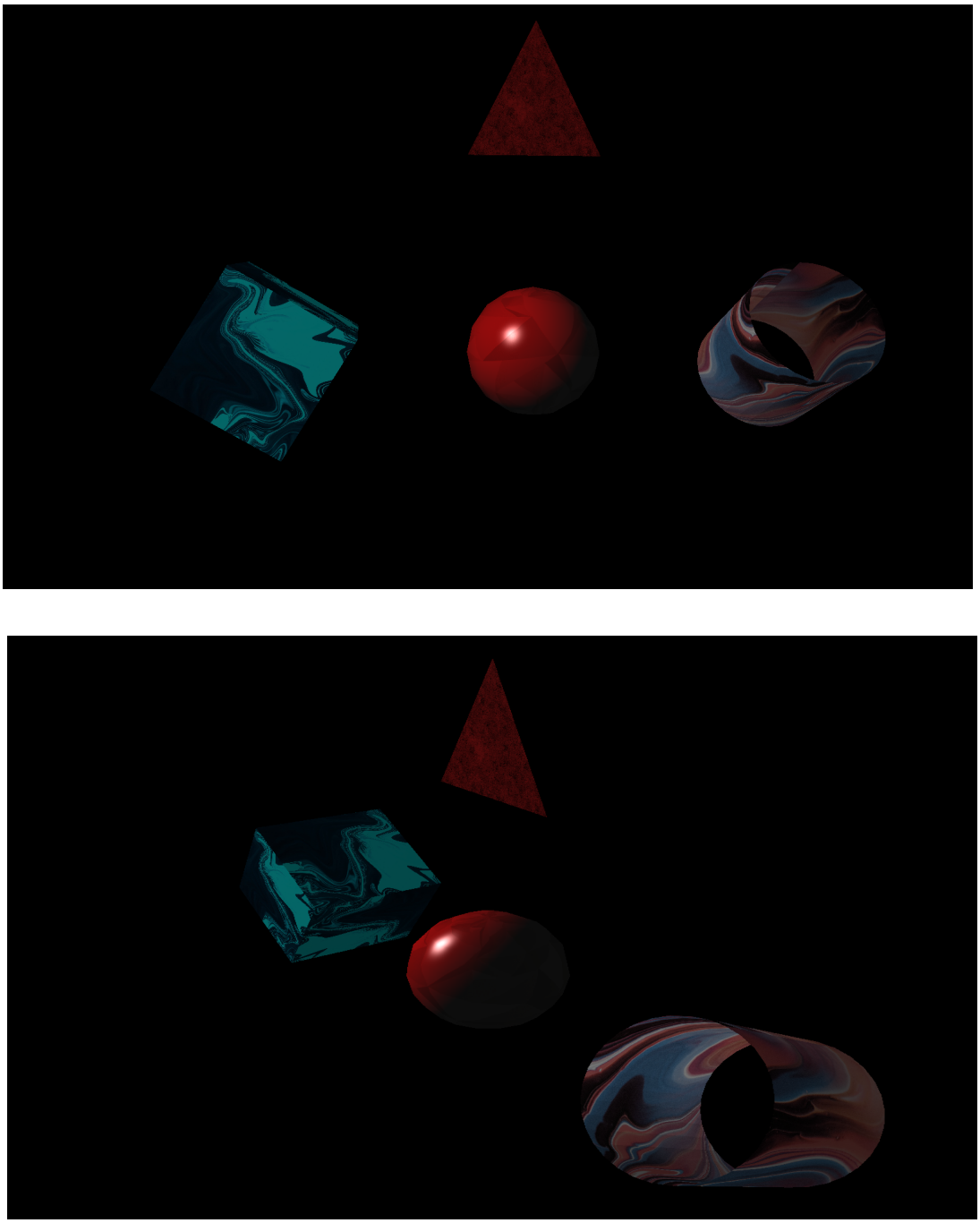

Figure 1: Obiecte în scenă