Laborator 7

1 Obiective

Obiectivul acestui laborator este de a descrie pe scurt modul de adăugare a iluminării în scena dvs. OpenGL 4.0. Vom prezenta noțiunile matematice de bază din spatele iluminatului OpenGL și structura programelor Vertex și Fragmet Shader pentru iluminarea direcțională per-vârf și per-pixel.

2 Fundament teoretic

Modelul de iluminare din OpenGL cu pipeline fix (modelul Gouraud) reprezintă modelul de bază pentru aplicații grafice și este o încercare decentă de a aproxima modul în care funcționează iluminarea din lumea reală. Cu toate acestea, funcționalitatea din pipeline-ul fix vine cu anumite constrângeri, fiind lipsită de realism și de opțiuni atunci când vine vorba de compromisuri calitate-performanță. Shaderele programabile pot să ofere rezultate mult mai bune, mai ales d.p.d.v. al realismului. Cu toate acestea, este destul de important să înțelegem foarte bine modelul de bază OpenGL, deoarece acest model de iluminare furnizează în continuare fundamentul pe care sunt construite modelele avansate.

Modelul de iluminare OpenGL cu pipeline fix însumează un set de componente independente pentru a obține efectul de iluminare general al unui punct de pe suprafața unui obiect. Aceste componente pot fi observate în Figura 1 de mai jos.

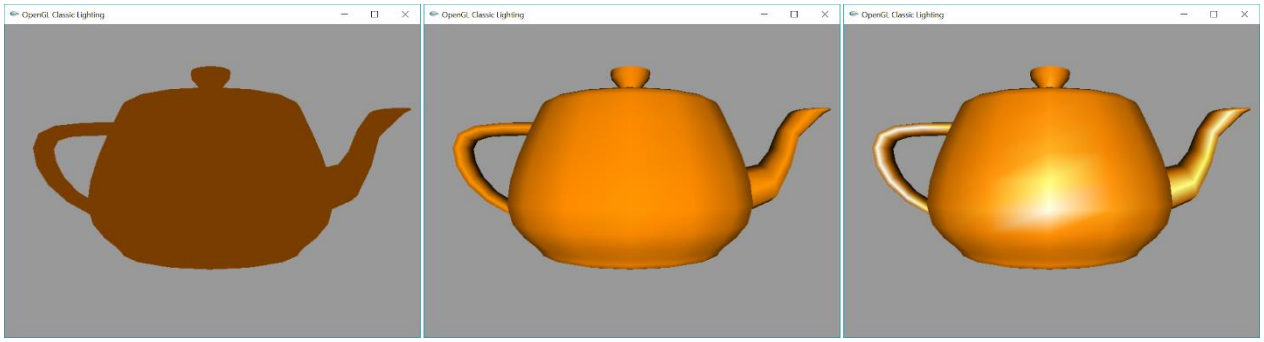

Figura 1 - Componente de lumină OpenGL (de la stânga la dreapta: Ambient, Ambient + Difuz, Ambient + Difuz + Specular)

Lumina ambientală nu vine dintr-o anumită direcție, fiind o aproximare decentă a luminii care există împrăștiată în jurul unei scene. Calculul său nu depinde de poziția spectatorului sau de direcția luminii. Valoarea iluminării ambientale poate fi precalculată ca efect global sau adăugată independent pentru fiecare sursă de lumină.

Lumina difuză este împrăștiată în mod egal în toate direcțiile pentru o sursă de lumină. Cantitatea de lumină difuză care există pe un punct nu depinde de poziția spectatorului, dar depinde de direcția luminii. Intensitatea luminoasa este mai puternică pe suprafețele care sunt orientate spre sursa de lumină și mai slabă pe cele orientate în direcția opusă. Calculul luminii difuze depinde de normala suprafeței și de direcția sursei de lumină, dar nu și de direcția de vizualizare.

*Lumina specular*ă este lumina reflectată direct de către suprafață și se referă la cât de asemănătoare este suprafata (materialul) obiectului cu o oglindă. Intensitatea acestui efect este denumită Shininess (stralucire). Calculul său necesită să se știe cât de apropiată este orientarea suprafeței de reflecția dintre direcția luminii și ochi. Astfel, calculul luminii speculare depinde de normala suprafației, de direcția luminii și de direcția de vizionare.

3 Implementarea funcționalității de iluminare în OpenGL 4

Acum vom construi treptat shaderele necesare pentru implementarea modelului de iluminat clasic OpenGL (iluminat Gouraud) folosind limbajul GLSL. Deoarece modelul clasic calculează iluminarea pentru fiecae vârf, shaderul pe care îl vom schimba treptat este Vertex Shader. Fragment Shader va rămâne același pe parcursul acestui proces, așa cum este prezentat mai jos.

```
//pass-through fragment shader for OpenGL lighting
#version 410 core

uniform vec3 baseColor;

in vec3 color;

out vec4 fColor;

void main()
{
	fColor = vec4(color, 1.0f);
}
```

În teorie, calculul iluminării poate fi efectuat în orice spațiu (sistem de coordonate) din pipeline-ul de transformare. Cu toate acestea, pentru rezultate corecte (și astfel mai realiste), iluminarea ar trebui să fie efectuată în spațiul de coordonate globale sau cel de vizualizare. Ambele vor oferi rezultate de iluminare identice. Pentru acest laborator, vom efectua calculul iluminării în spațiul de vizualizare.

3.1 Lipsa efectului de iluminare

Atunci când nu se utilizează nici un model de iluminare, OpenGL redă fiecare obiect în mod uniform, folosind culoarea sa de bază. Toate efectele de iluminare vor modula această culoare, folosind-o ca bază pentru următoarele calcule de iluminare. Vertex Shader necesar pentru desenarea unui obiect fără efecte de iluminare este prezentat mai jos. Pentru acest laborator, culoarea de bază este furnizată din aplicație ca valoare **uniform**. În aplicațiile din lumea reală, culoarea de bază este, de obicei, furnizată ca atribut al vârfului.

```
//vertex shader for no lighting
#version 410 core

layout(location=0) in vec3 vPosition;
layout(location=1) in vec3 vNormal;

//matrices
uniform mat4 model;
```

```
uniform mat4 view;
uniform mat4 projection;

//lighting
uniform vec3 lightColor;
uniform vec3 baseColor;

out vec3 color;

void main()
{
	//compute final vertex color
	color = baseColor;

	//transform vertex
	gl_Position = projection * view * model * vec4(vPosition, 1.0f);
}
```

3.2 Iluminarea ambientală

Lumina ambientală este uniformă și, prin urmare, nu se schimbă între vârfuri. Putem trimite componenta Ambientală din aplicație ca o variabilă **uniform**. Rezultatul interacțiunii dintre lumina ambientală și culoarea suprafeței pe care o lovește este modelat prin înmulțire. În timpul diferitelor înmulțiri necesare pentru a calcula efectele diferitelor componente luminoase pe o suprafață, valorile componentelor pot să depășească valoarea de 1,0. Cu toate acestea, culoarea finală trebuie să fie saturată la alb (adică nici o componentă de lumină nu trebuie să fie mai mare de 1,0). Din acest motiv se utilizează funcția **min** pentru a atinge această saturație. Adăugările și modificările la vertex shader pentru încorporarea luminii ambientale sunt evidențiate mai jos.

```
//vertex shader for ambient lighting
#version 410 core

layout(location=0) in vec3 vPosition;
layout(location=1) in vec3 vNormal;

//matrices
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

//lighting
uniform vec3 lightColor;
uniform vec3 baseColor;

out vec3 color;

vec3 ambient;
float ambientStrength = 0.2f;

void main()
{
	//compute ambient light
```

```
        ambient = ambientStrength * lightColor;

        //compute final vertex color
        color = min(ambient * baseColor, 1.0f);

        //transform vertex
        gl_Position = projection * view * model * vec4(vPosition, 1.0f);
}
```

3.3 Iluminarea ambientală și difuză

Iluminarea difuză depinde de direcția luminii care atinge punctul pe care îl calculăm în prezent. Deoarece lumina noastră este una **direcțională** (situată la infinit, spre deosebire de sursele de lumină punctiforme care au o poziție 3D definită), toate razele luminoase sunt paralele și au aceeași direcție. Aceasta înseamnă că direcția luminii este constantă pe toate vârfurile și poate fi calculată în aplicație și trimisă ca o variabilă **uniform** în shader. Cantitatea de lumină difuză împrăștiată depinde de orientarea suprafeței (adică de unghiul dintre direcția luminii și normala suprafeței). Cantitatea maximă de lumină împrăștiată este obținută atunci când lumina atinge perpendicular suprafața (adică unghiul dintre normală și direcția luminii este 0,0, astfel cosinusul este 1,0). Pe măsură ce acest unghi crește (suprafața se îndepărtează de lumină), cosinusul scade spre 0,0, astfel încât cantitatea de lumină difuză împrăștiată este mai mică. Cosinul unghiului dintre doi vectori normalizați poate fi calculat folosind produsul scalar (dot product).

Normalele vârfurilor sunt transmise din aplicație (ele sunt pre-calculate în modelul GLM) ca atribut per-vârf (similar cu pozițiile vârfurilor). Deoarece modelul este afectat de transformări (fiecare vârf este înmulțit cu matricea Model, Vedere și Proiecție), normala trebuie supusă acelorași transformări ca și vârful obiectului, altfel iluminarea nu va funcționa corect (adică normala trebuie să se miște împreună cu obiectul). În OpenGL, normalele nu sunt transformate folosind matricea Model-View-Projection, ci folosind așa-numita Matrice Normală. Matricea normală este matricea 3x3 din stânga-sus a inversului transpus al matricei model (dacă se efectuează calculul luminii în spațiul global) sau al matricei Model-View (dacă se efectuează calculul iluminării în spațiul de vizualizare). Aceasta poate fi ușor calculată în aplicație utilizând biblioteca OpenGL matematică (GLM). După calculare, matricea normală este trimisă shader-ului ca variabilă **uniform**. Adăugările la vertex shader pentru a încorpora lumina difuză sunt evidențiate mai jos.

```
//vertex shader for ambient and diffuse lighting
#version 410 core

layout(location=0) in vec3 vPosition;
layout(location=1) in vec3 vNormal;

//matrices
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
uniform mat3 normalMatrix;

//lighting
uniform vec3 lightDir;
uniform vec3 lightColor;
```

```
uniform vec3 baseColor;

out vec3 color;

vec3 ambient;
float ambientStrength = 0.2f;
vec3 diffuse;

void main()
{
	//compute ambient light
	ambient = ambientStrength * lightColor;

	//normalize the light's direction
	vec3 lightDirN = normalize(lightDir);

	//compute eye coordinates for normals (transform normals)
	vec3 normalEye = normalize(normalMatrix * vNormal);

	//compute diffuse light
	diffuse = max(dot(normalEye, lightDirN), 0.0f) * lightColor;

	//compute final vertex color
	color = min((ambient + diffuse) * baseColor, 1.0f);

	//transform vertex
	gl_Position = projection * view * model * vec4(vPosition, 1.0f);
}
```

3.4 Iluminarea ambientală, difuză și specular

Iluminarea specular depinde atât de direcția luminii, cât și de direcția de vizualizare. Componenta specular depinde de cosinusul unghiului (α) dintre direcția de vizualizare (ochi) și reflexia provenită de la lumina (R) (figura 2). Deoarece efectuăm calculul iluminării în coordonatele camerei (de vizualizare), camera este situată la origine, astfel poziția ochiului este (0, 0, 0).

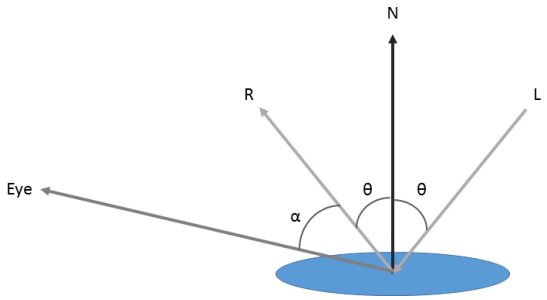

Figura 1 – Calculul iluminării speculare

Shininess (strălucirea) este folosit ca un exponent pentru a obține o măsură a claritățiii căderii unghiulare dintr-o reflecție directă. Cu cât este mai mare exponentul, cu atât devine mai intensă iluminarea specular, deoarece ridicarea unui număr mai mic decât 1.0 la o putere exponențială scade valoarea sa. Unghiurile de reflexie apropiate de 0.0 vor duce la valori apropiate de 1.0 pentru iluminarea specular, în timp ce pentru alte unghiuri aceasta se va îndrepta rapid înspre 0.0. Deoarece lumina specular reprezintă lumina

reflectată, componenta specular nu mai este modulată cu culoarea obiectului. Modificările din vertex shader pentru a încorpora iluminarea specular sunt evidențiate mai jos.

```
//vertex shader for ambient, diffuse and specular lighting
#version 410 core

layout(location=0) in vec3 vPosition;
layout(location=1) in vec3 vNormal;

//matrices
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
uniform mat3 normalMatrix;

//lighting
uniform vec3 lightDir;
uniform vec3 lightColor;
uniform vec3 baseColor;

out vec3 color;

vec3 ambient;
float ambientStrength = 0.2f;
vec3 diffuse;
vec3 specular;
float specularStrength = 0.5f;
float shininess = 32.0f;

void main()
{
	//compute ambient light
	ambient = ambientStrength * lightColor;

	//normalize the light's direction
	vec3 lightDirN = normalize(lightDir);

	//compute eye coordinates for normals
	vec3 normalEye = normalize(normalMatrix * vNormal);

	//compute diffuse light
	diffuse = max(dot(normalEye, lightDirN), 0.0f) * lightColor;

	//compute the vertex position in eye coordinates
	vec4 vertPosEye = view * model * vec4(vPosition, 1.0f);

	//compute the view (Eye) direction (in eye coordinates, the camera is at the origin)
	vec3 viewDir = normalize(- vertPosEye.xyz);

	//compute the light's reflection (the reflect function requires a direction pointing towards the vertex, not away
from it)
	vec3 reflectDir = normalize(reflect(-lightDir, normalEye));

	//compute specular light
```

```
        float specCoeff = pow(max(dot(viewDir, reflectDir), 0.0f), shininess);
        specular = specularStrength * specCoeff * lightColor;

        //compute final vertex color
        color = min((ambient + diffuse) * baseColor + specular, 1.0f);

        //transform vertex
        gl_Position = projection * view * model * vec4(vPosition, 1.0f);
}
```

4 Tutorial

Descărcați resursele disponibile pe pagina web a laboratorului. Prima arhivă (Laboratory 7 resources - part 1.zip) conține o aplicație cu shaderele vertex și fragment necesare pentru a desena un ceainic fără iluminare (a se vedea subsecțiunea 3.1).

Extindeți vertex shader pentru a încorpora iluminatul ambiental în aplicația dvs. (vezi secțiunea 3.2). Aplicația dvs. ar trebui să afișeze ceainicul ca în figura 3 de mai jos.

Figura 2 –Rasterizare cu iluminare ambientală

Extindeţi vertex shader pentru a încorpora iluminarea difuză în aplicaţia dvs. (vezi secţiunea 3.2). Aplicaţia dvs. ar trebui să afişeze ceainicul ca în figura 4 de mai jos.

Figura 3 – Rasterizare cu iluminare ambientală şi difuză

Extindeţi vertex shader pentru a încorpora iluminarea specular în aplicaţia dvs. (vezi secţiunea 3.3). Aplicaţia dvs. ar trebui să afişeze ceainicul ca în figura 4 de mai jos.

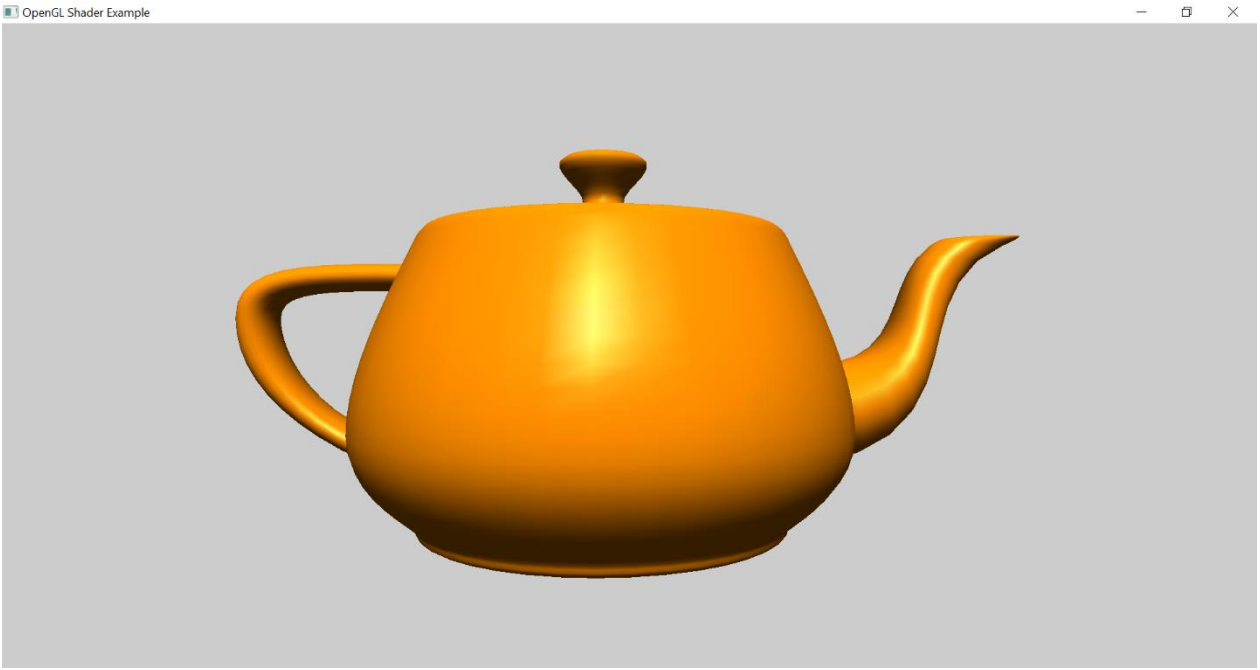

Figura 4 – Rasterizare cu iluminare ambientală, difuză și specular

5 Lectură suplimentară

- OpenGL Programming Guide 8th Edition – Capitolul 7

6 Temă

1. Implementați iluminatul per-vertex OpenGL (Gouraud) pentru aplicația dvs. urmând pașii descriși în secțiunea 4
2. Implementați modelul de iluminare per-pixel (Phong). Iluminarea per-pixel calculează culorile finale din fragment shader, folosind **normalele interpolate**, în loc să folosească normalele vârfului în vertex shader. Comparați calitatea generală a iluminării cu aceea a modelului per-vertex obținut la pasul 1, pentru toate rezoluțiile obiectului ceainic (există mai multe ceainice disponibile, generate folosind 4, 10, 20 și 50 de segmente). Pentru aceast exercițiu, utilizați resursele furnizate în cea de-a doua arhivă (Laboratory 7 resources - part 2.zip) și modificați setul de shader-e shaderPPL (vert / frag).
3. Extindeți implementarea iluminării per-pixel pentru a încorpora proprietățile materialelor pentru obiecte. Proprietățile materialelor pot fi trimise din aplicație ca variabile **uniform**, iar culoarea finală va fi modulată (înmulțită) cu componenta de material corespunzătoare (Ambient, Diffuse și Specular) în locul culorii de bază a obiectului.
4. Extindeți implementarea iluminării per-pixel pentru a încorpora intensitățile componentelor luminoase. În loc să aibă o lumină cu o singură culoare, folosind coeficienți “hard-codați” pentru a decide contribuția fiecărui componentă a luminii, descrieți o sursă de lumină folosind trei componente diferite (ambient, difuz și specular) și folosiți-le pentru a calcula intensitățile finale ale culorii.