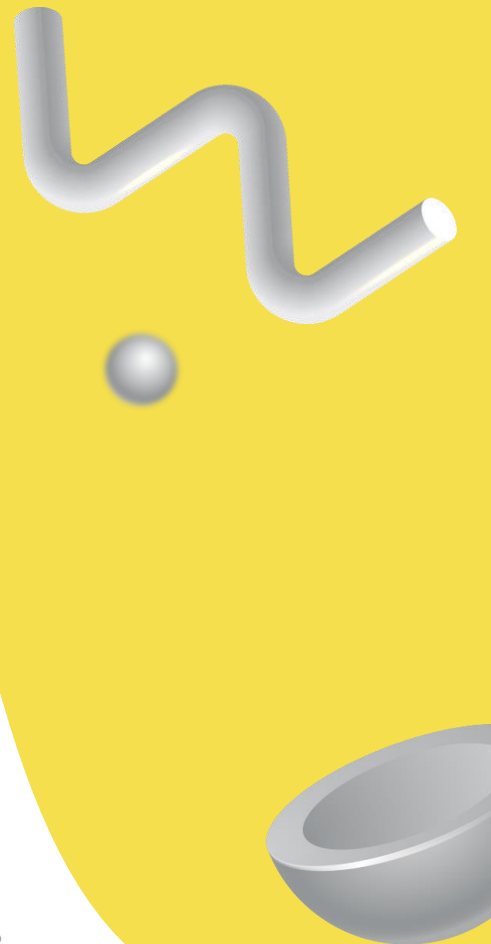


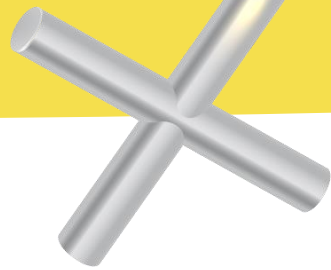
Programarea Aplicațiilor de Simulare

Pipeline de randare și OpenGL

Curs 8

Pătrânjel David-George





Agenda Cursului

01

Recapitulare

Transformări

Pipeline Simplu

03

Efecte comune

Texturare

Modelul de iluminare Phong

Normal mapping

02

Shadere

Atribute

Uniforme

Exemple

04

OpenGL Avansat

Face culling

Blending & transparență

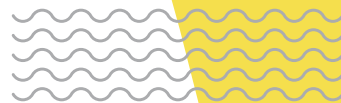
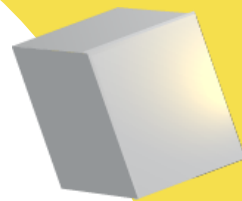
Depth buffers și Frame buffers



01

Recapitulare

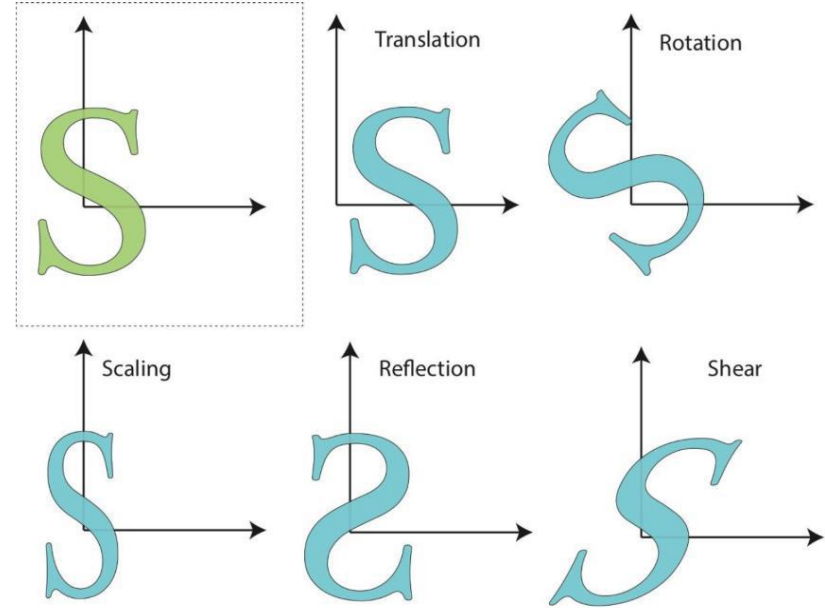
Transformări
Pipeline Simplu





Transformări

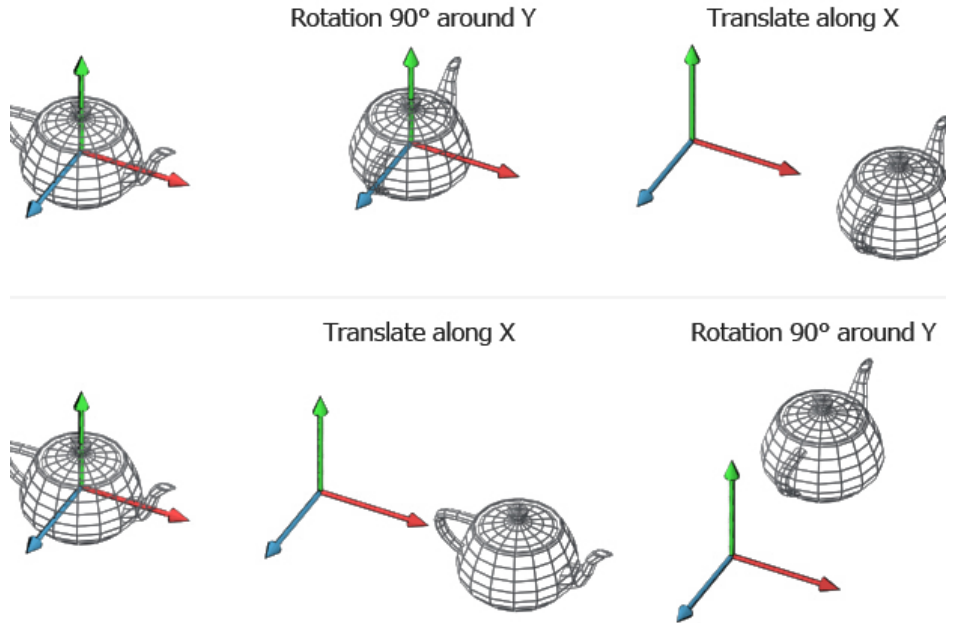
- Mai multe tipuri de transformări.
- Reprezentare prin formă **matriceală** (formulele se găsesc în cursul anterior)
- Pentru a combina mai multe transformări se înmulțesc matricele transformărilor.





Transformări

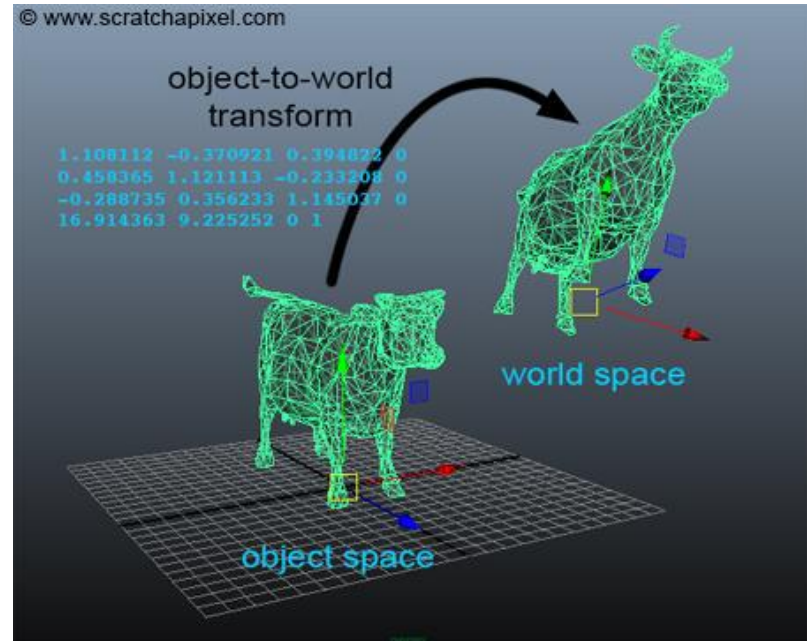
- Ordinea transformărilor contează!



Transformarea de modelare

Transformarea de modelare:

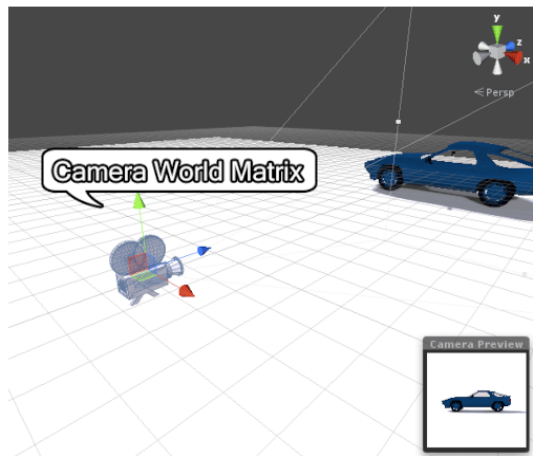
- Se iau toate transformările care se efectuează asupra obiectului, se calculează matricele acestora, iar apoi se înmulțesc. Rezultatul va fi o matrice numită **World Matrix**.
- Această matrice va fi înmulțită cu coordonatele fiecărui vârf al geometriei, astfel transformând coordonatele din **Object Space** în **World Space**.



Transformarea de vizualizare

Transformarea de vizualizare:

- Asupra geometriei aplicăm o transformare astfel încât să pară că aceasta este văzută din perspectiva camerei. Acest spațiu se numește View Space (**Spațiul Observator**).
- Pentru un punct P din World Space în View Space este nevoie să aplicăm transformarea $P' = M_{cam}^{-1} * P$.
- Matricea se numește **Matrice de Vizualizare**.

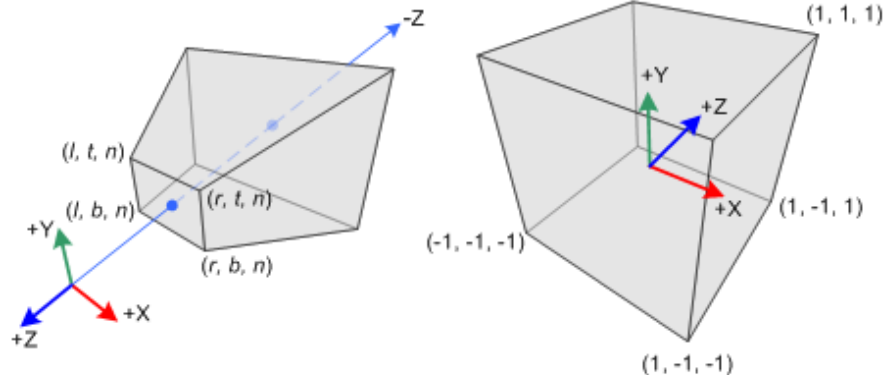


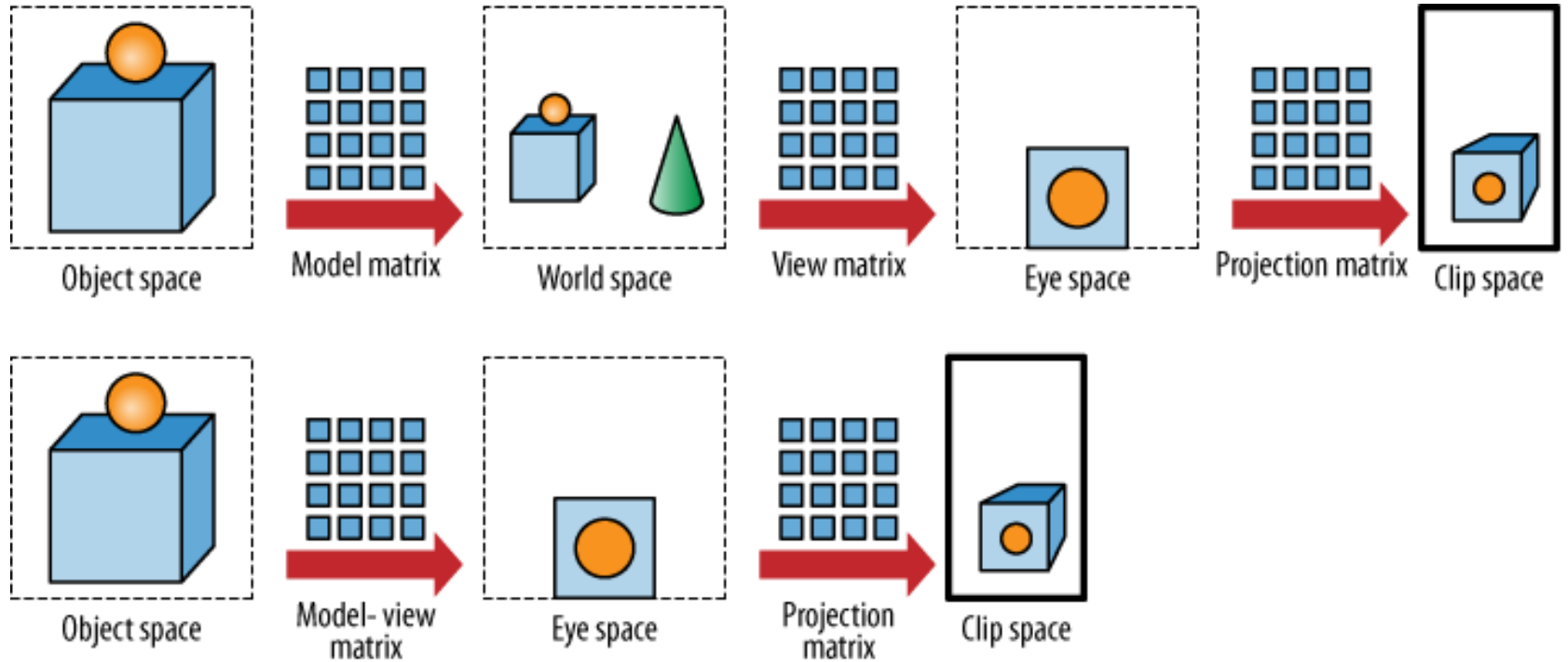
View Matrix

Transformarea de proiecție

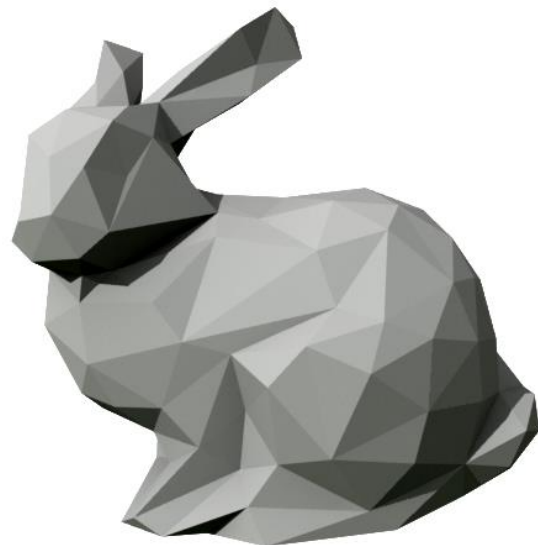
Transformarea de proiecție:

- Pentru trecerea de la View Space la Clip Space se folosește o matrice de 4x4 numită Projection Matrix (**Matrice de Proiecție**)
- În funcție de specificul aplicației, aceasta poate fi construită în mai multe moduri.



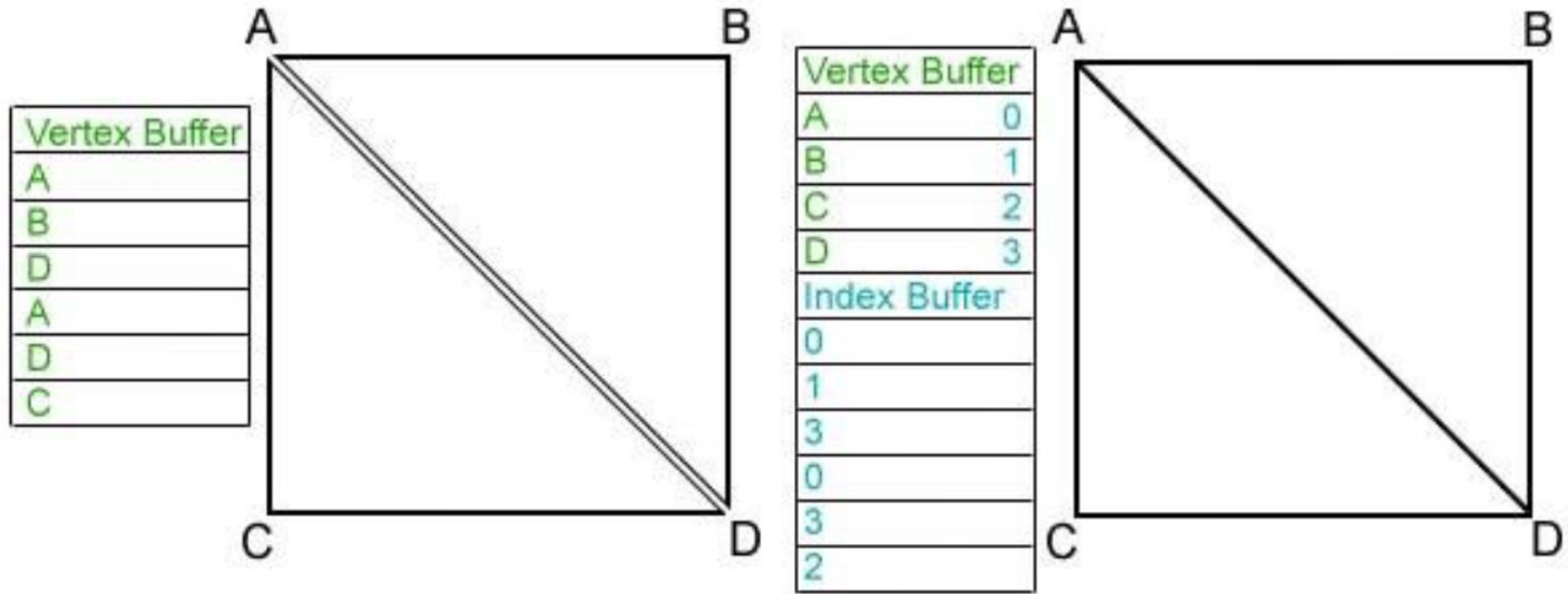


$$P_{clip} = M_{proj} M_{view} M_{world} P_{object}$$



Poligoane

https://en.m.wikipedia.org/wiki/File:Stanford_bunny_gem.png



Vertex (VBO) and Index (IBO) Buffers

Triunghiuri

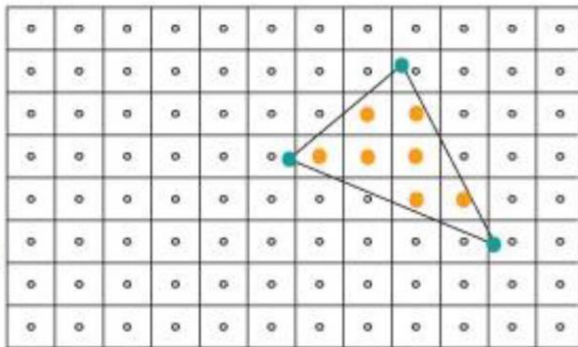
```
float vertices[] = {  
    0.5f, 0.5f, 0.0f, // top right  
    0.5f, -0.5f, 0.0f, // bottom right  
    -0.5f, -0.5f, 0.0f, // bottom left  
    -0.5f, 0.5f, 0.0f // top left };  
unsigned int indices[] = {  
    0, 1, 3, // first triangle  
    1, 2, 3 // second triangle};
```

Transmiterea datelor spre GPU

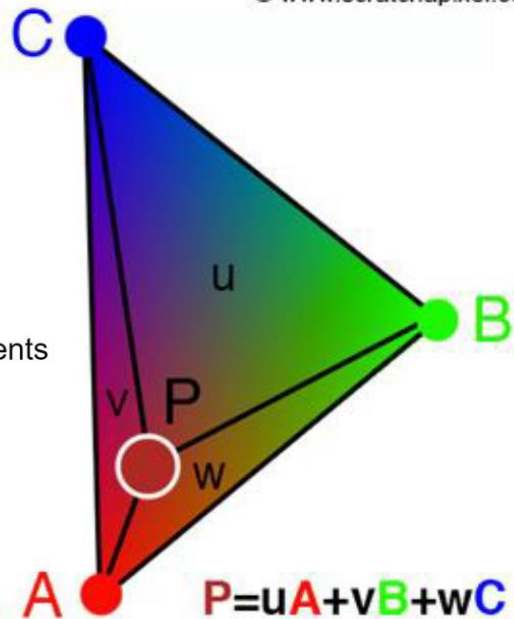
```
unsigned int vbo ;  
glGenBuffers (1 , vbo);  
glBindBuffer(GL_ARRAY_BUFFER, vbo);  
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices , GL_STATIC_DRAW);  
  
unsigned int ebo;  
glGenBuffers(1 , ebo );  
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);  
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices , GL_STATIC_DRAW);
```

Declararea listelor de vârfuri și indecși

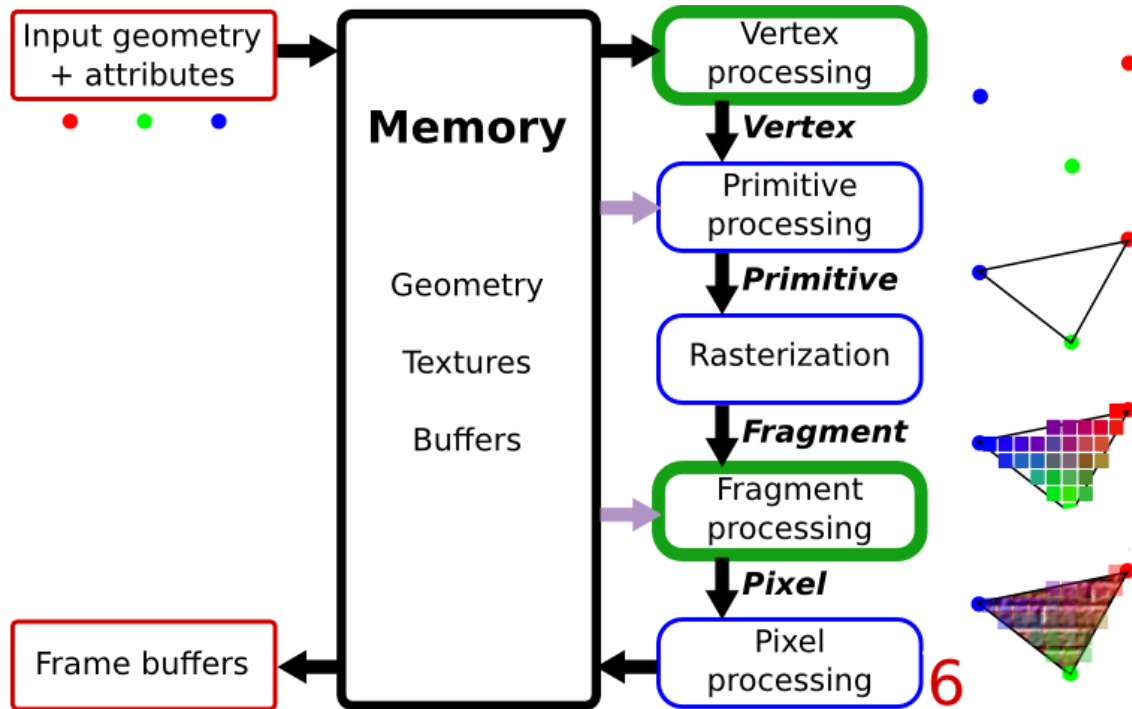
Vertices



Fragments



Rasterizare

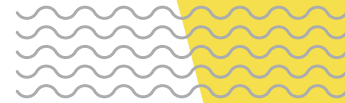
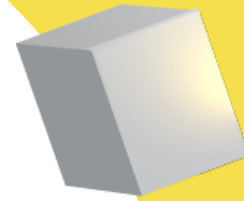


OpenGL Pipeline

02

Shadere

Attribute
Uniform
Exemple





Atribute

- Pe lângă poziție, fiecărui vârf îi putem atribui diferite **proprietăți**.
- Aceste proprietăți se numesc atribute.
- **Exemple de atribute:**
 - Culoare, coordonate de textură, normale, tangente etc.
- Se folosesc **buffere de vertecși** pentru a încărca în memoria plăcii video toate atributele vârfurilor.



Intercalat



Hibrid



SoA –
Structura
de Array-uri



Buffere de vârfuri



Uniforme

- Uniformele sunt argumente pe care le putem trimite shaderelor.
- Au aceeași valoare pentru toate vârfurile/fragmentele din draw call.





Uniforme

```
#version 330
```

```
// Uniform properties
```

```
uniform mat4 Model;
```

```
uniform mat4 View;
```

```
uniform mat4 Projection;
```

```
layout(location = 0) out vec4 out_color;
```

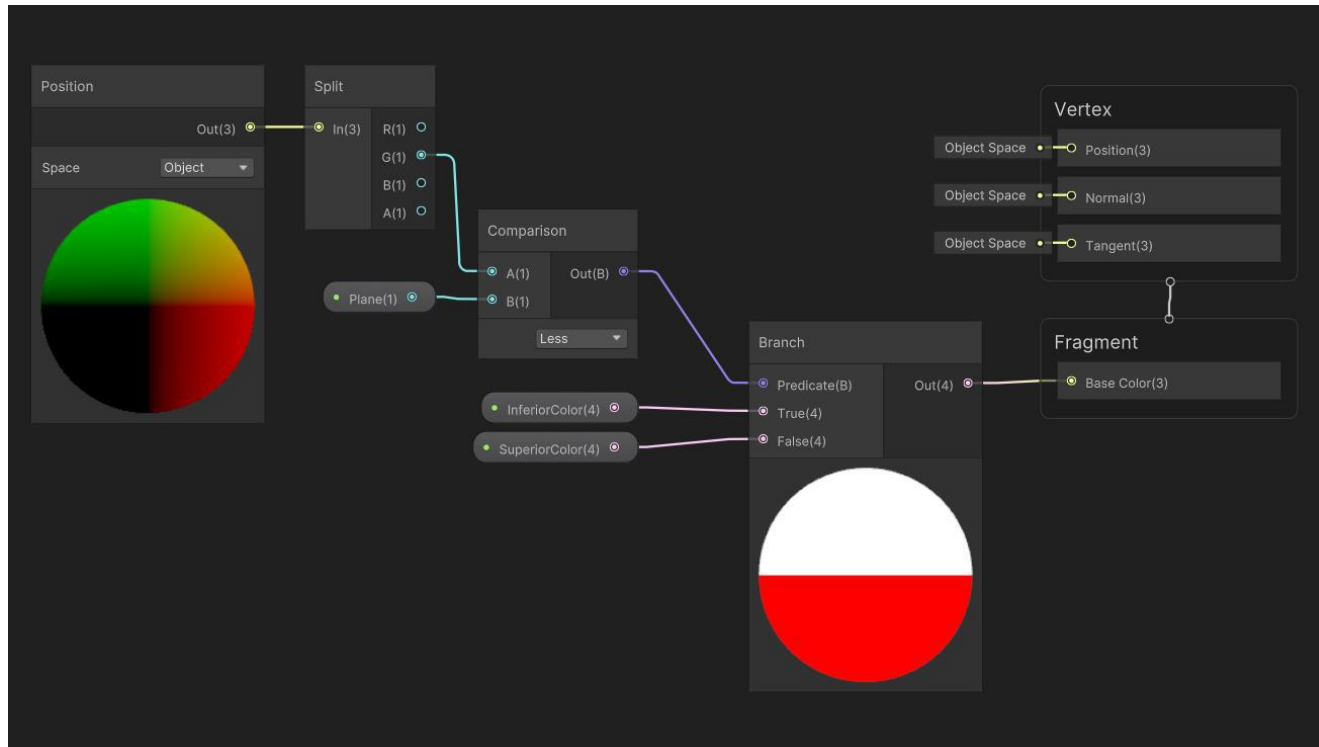
```
void main()
```

```
{
```

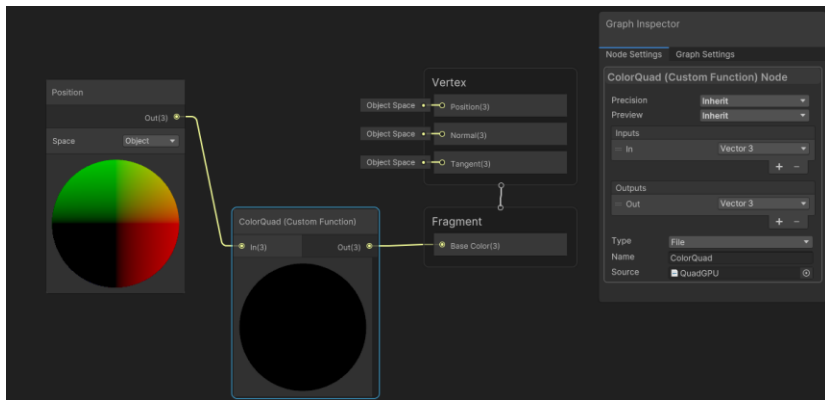
```
    out_color = vec4(1, 0, 0, 0);
```

```
}
```





URP Shader Graph (Unity)



```
void ColorQuad_float(in float3 In, out float3 Out)
{
    if (In.y < 0.5)
        Out = InferiorColor;
    else
        Out = SuperiorColor;
}
```

```
void ColorQuad_half(in half3 In, out half3 Out)
{
    if (In.y < 0.5)
        Out = InferiorColor_half;
    else
        Out = SuperiorColor_half;
}
```



URP Shader Graph (Unity)

Vertex Shader

```
#version 400

in vec3 InputPosition;
out vec3 PositionObject;

uniform mat4 WorldMatrix;
uniform mat4 ViewMatrix;
uniform mat4 ProjectionMatrix;

void main(void)
{
    vec4 worldPos = WorldMatrix * vec4(InputPosition, 1.0);
    vec4 viewPos = ViewMatrix * worldPos;
    gl_Position = ProjectionMatrix * viewPos;

    PositionObject = InputPosition;
}
```

Fragment Shader

```
#version 400

in vec3 PositionObject;

uniform float Plane;
uniform vec3 SuperiorColor;
uniform vec3 InferiorColor;

out vec4 OutputColor;

void main(void)
{
    OutputColor = vec4(SuperiorColor, 1.0);
    if (PositionObject.y > Plane)
        OutputColor = vec4(InferiorColor, 1.0);
}
```



Exemplu OpenGL (GLSL)

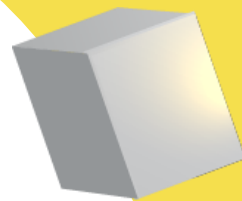
03

Efecte comune

Texturare

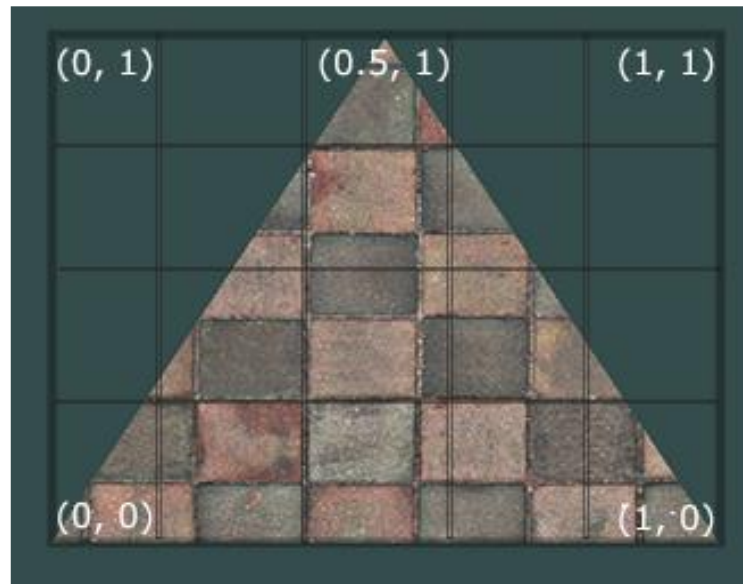
Modelul de iluminare Phong


Normal mapping



Texturi

- O textură este o imagine proiectată peste un poligon.
- Pe un poligon se afișează doar o secțiune a imaginii.
- Pentru acest lucru folosim **coordonate de textură (UV)**





```
float vertices[] = {
    // positions      // colors      // texture coords
    0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f, // top right
    0.5f, -0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, // bottom right
    -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, // bottom left
    -0.5f, 0.5f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f // top left
};

unsigned int vbo;
glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

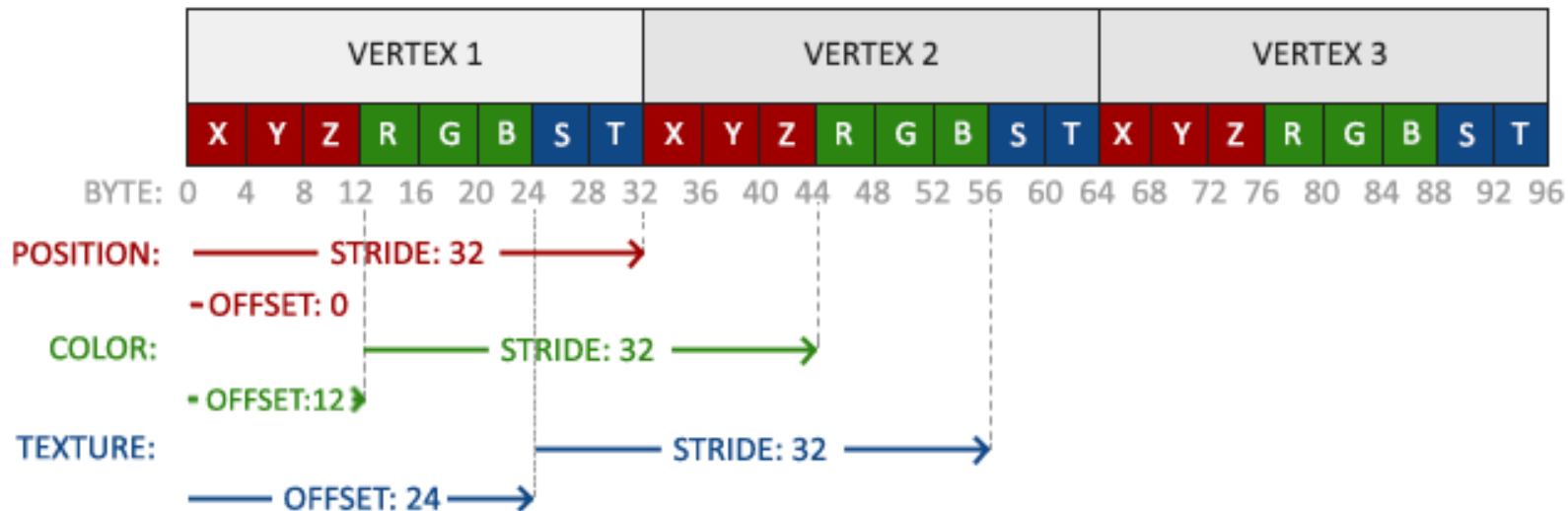
// Position attribute
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);

// Color attribute
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);

// Texture coordinate attribute
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 * sizeof(float)));
glEnableVertexAttribArray(2);
```

Coordonate de texturi

Vârfuri





GL_REPEAT



GL_MIRRORED_REPEAT



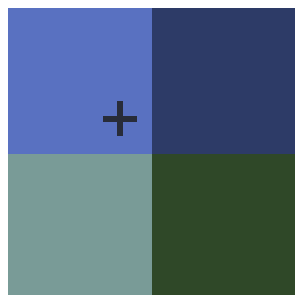
GL_CLAMP_TO_EDGE



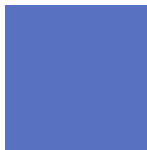
GL_CLAMP_TO_BORDER



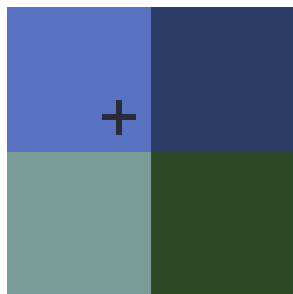
Texture Wrapping



returns



GL_NEAREST



returns



GL_LINEAR



Texture Filtering



MipMapping

<https://gdbooks.gitbooks.io/legacyopengl/content/Chapter7/Mip.html>

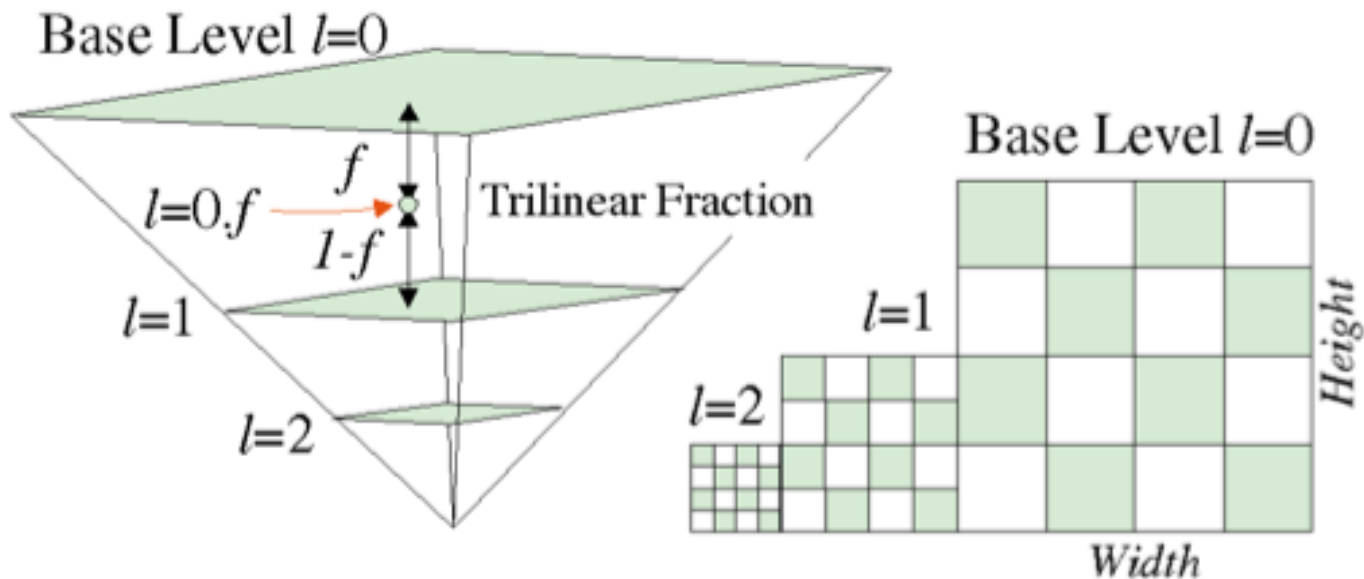
Soluție

Se generează mai multe variante ale texturii la rezoluții înjumătățite pe verticală orizontală . În funcție de distanța față de cameră , se alege o textură la o rezoluție mai mare sau mai mică



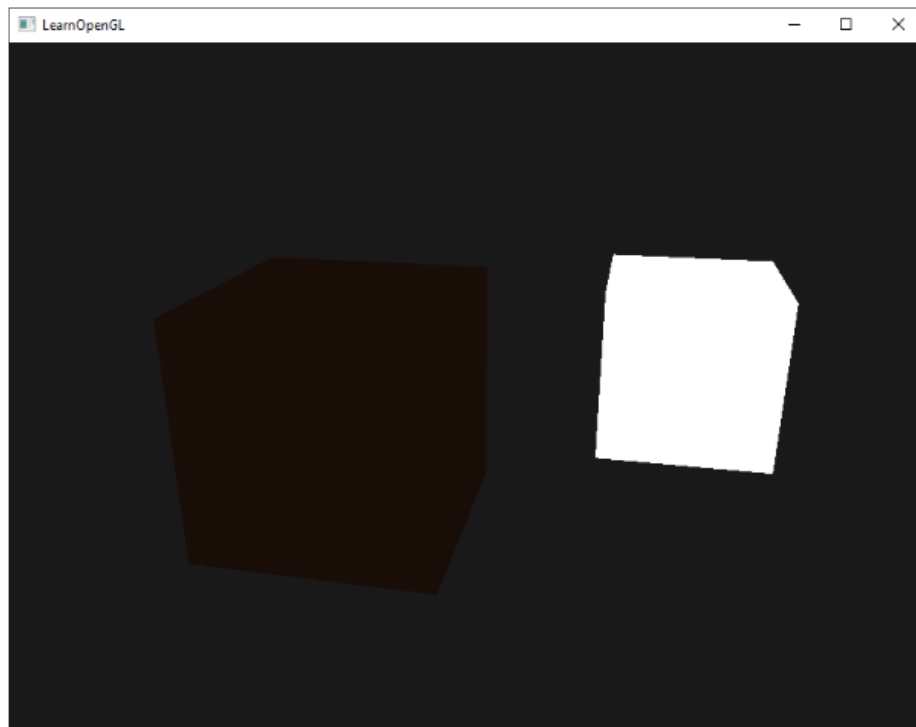
Soluție

Se generează mai multe variante ale texturii la rezoluții înjumătățite pe verticală/orizontală. În funcție de distanța față de cameră, se alege o textură la o rezoluție mai mare sau mai mică



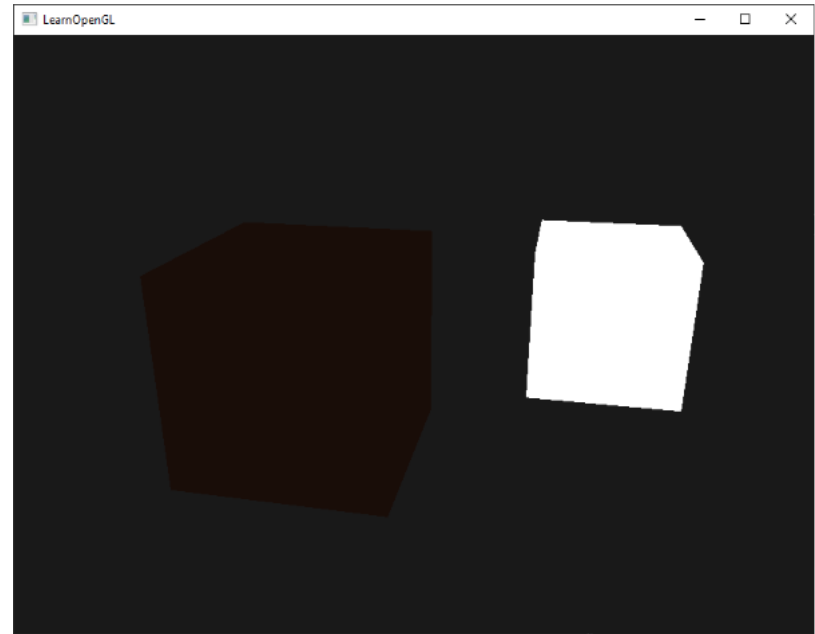
Lumina ambinetală

- Chiar și în întuneric, de regulă tot există câteva raze de lumină care lovesc obiectele și le fac vizibile.
- Putem simula acest tip de lumină folosind o valoare constantă.



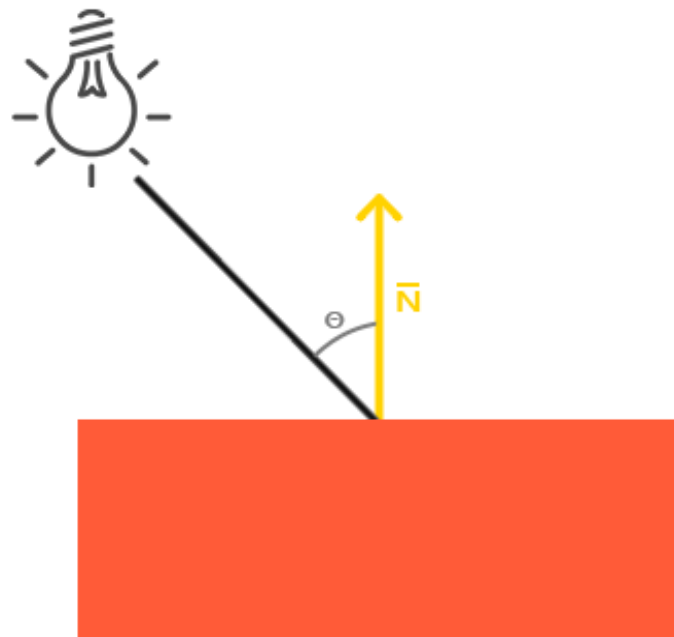
Lumina ambinetală

```
void main()
{
    float ambientStrength = 0.1;
    vec3 ambient = ambientStrength * lightColor;
    vec3 result = ambient * objectColor;
    FragColor =vec4 (result, 1.0);
}
```



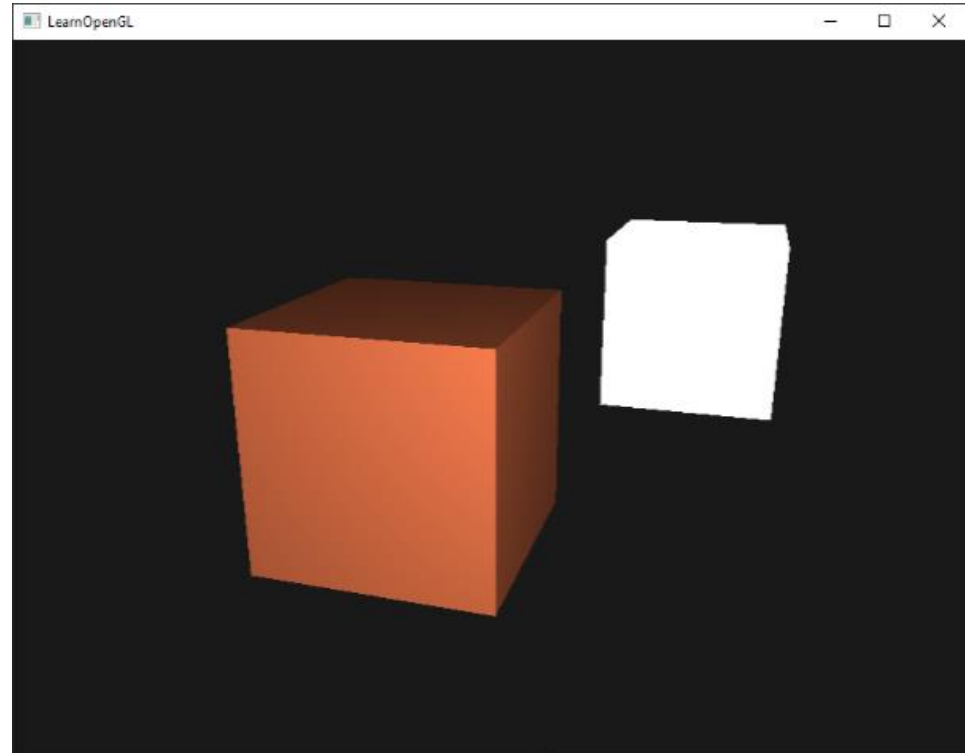
Lumina difuză

- Cu cât o față este mai înclinată către direcția razelor de lumină, cu atât este iluminată mai puternic
- Cea mai semnificativă parte a modelului de iluminare



Lumina difuză

- Se calculează cosinusul unghiului dintre raza de lumină și normala poligonului.
- Trebuie trimise normalele ca atribute ale vertecșilor.



Lumina difuză

```
#version 400

in vec3 Normal;
in vec3 PositionWorld;

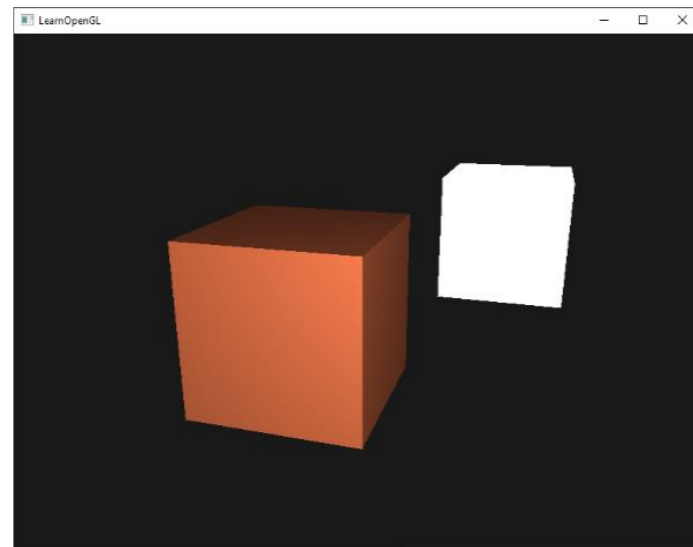
out vec4 FragColor;

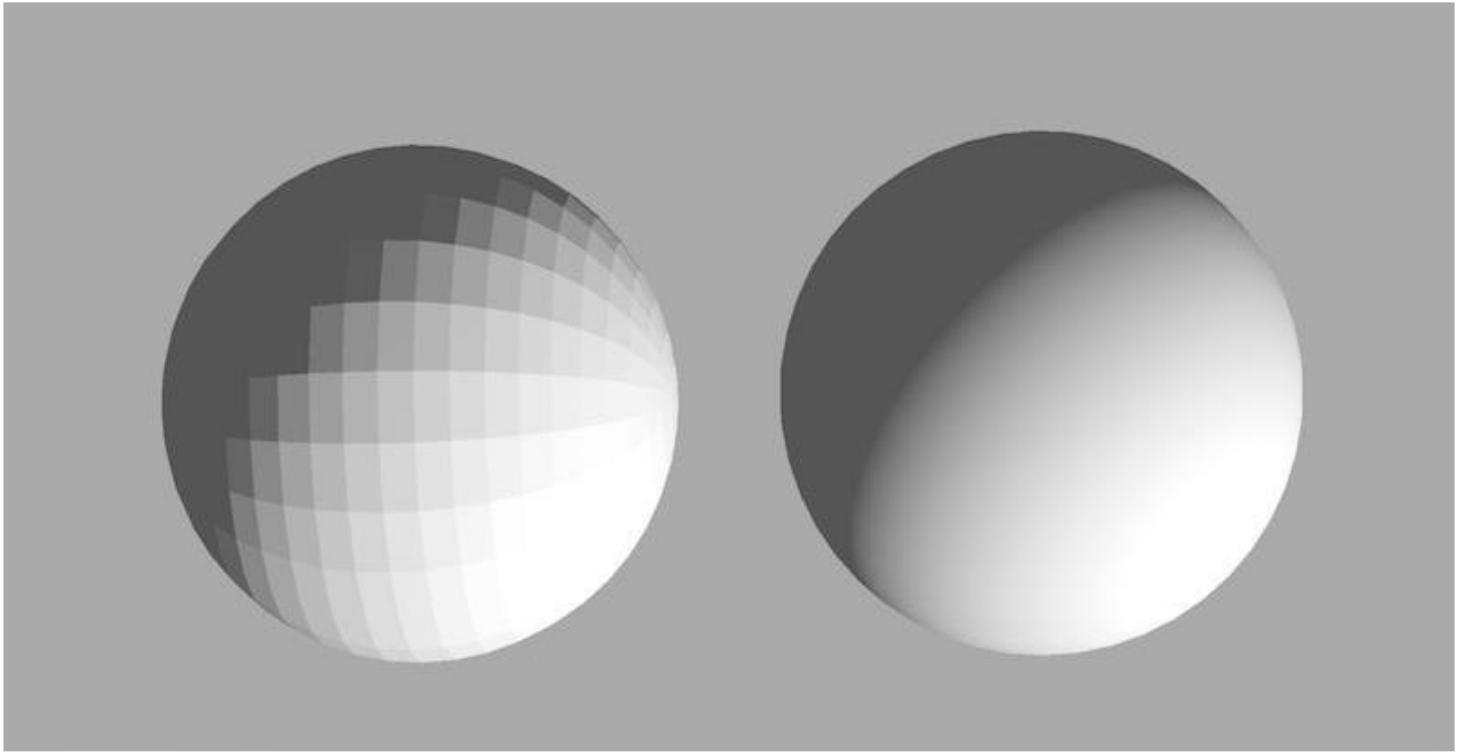
uniform vec3 LightPos;
uniform vec3 LightColor;
uniform vec3 ObjectColor;

void main()
{
    // Ambient lighting
    float ambientStrength = 0.1;
    vec3 ambient = ambientStrength * LightColor;

    // Diffuse lighting
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(LightPos - PositionWorld);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * LightColor;

    // Combine results
    vec3 result = (ambient + diffuse) * ObjectColor;
    FragColor = vec4(result, 1.0);
}
```

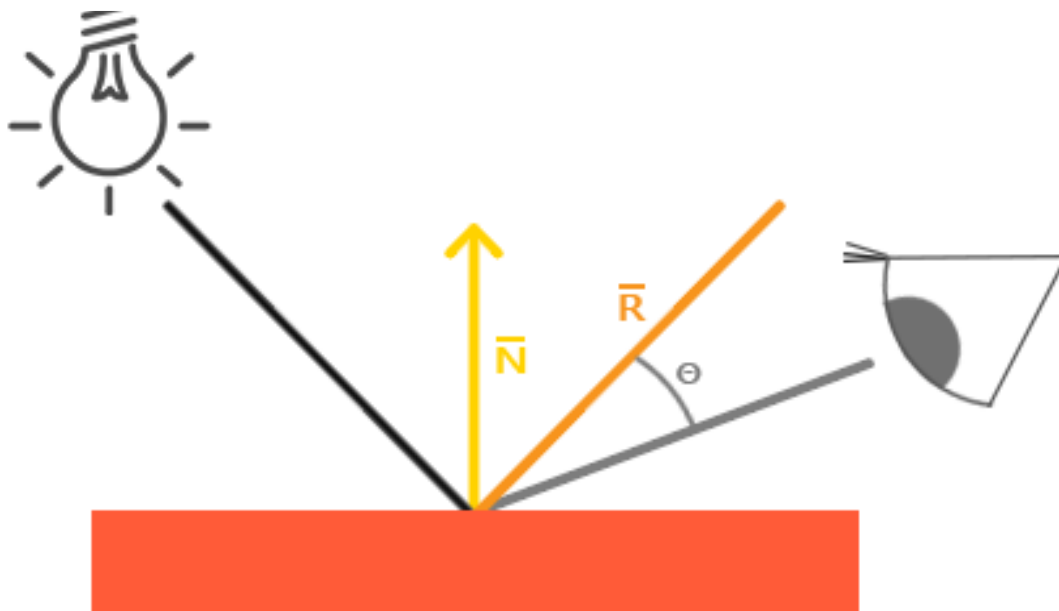




Flat shading vs. smooth shading

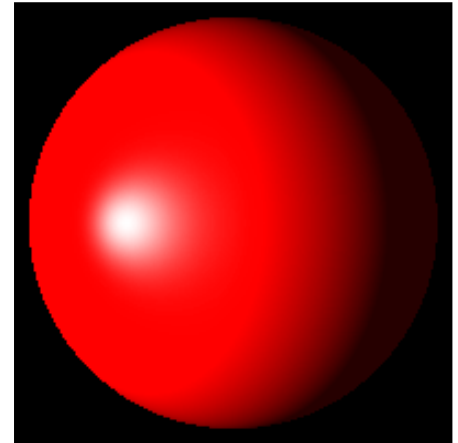
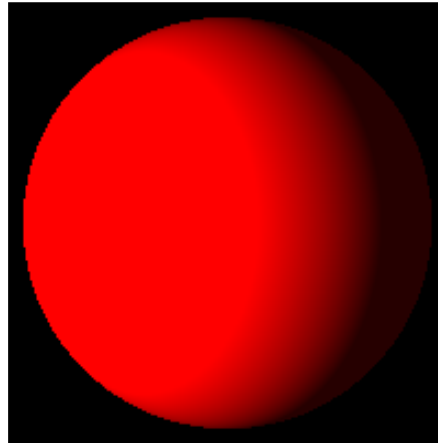
Lumina speculară

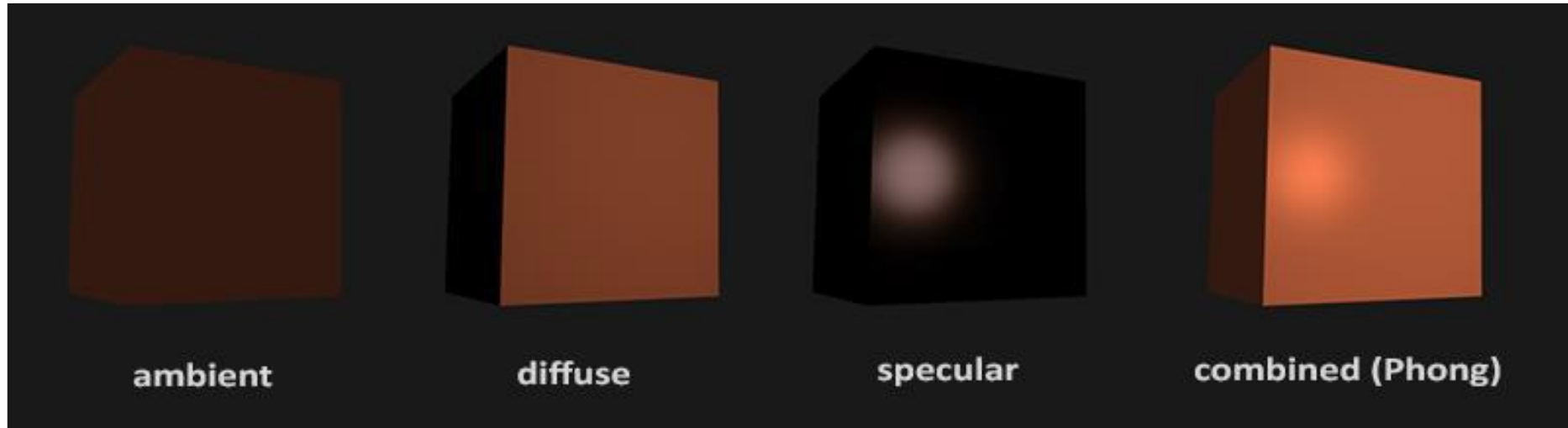
Simulează zonele luminoase ale obiectelor lucioase.



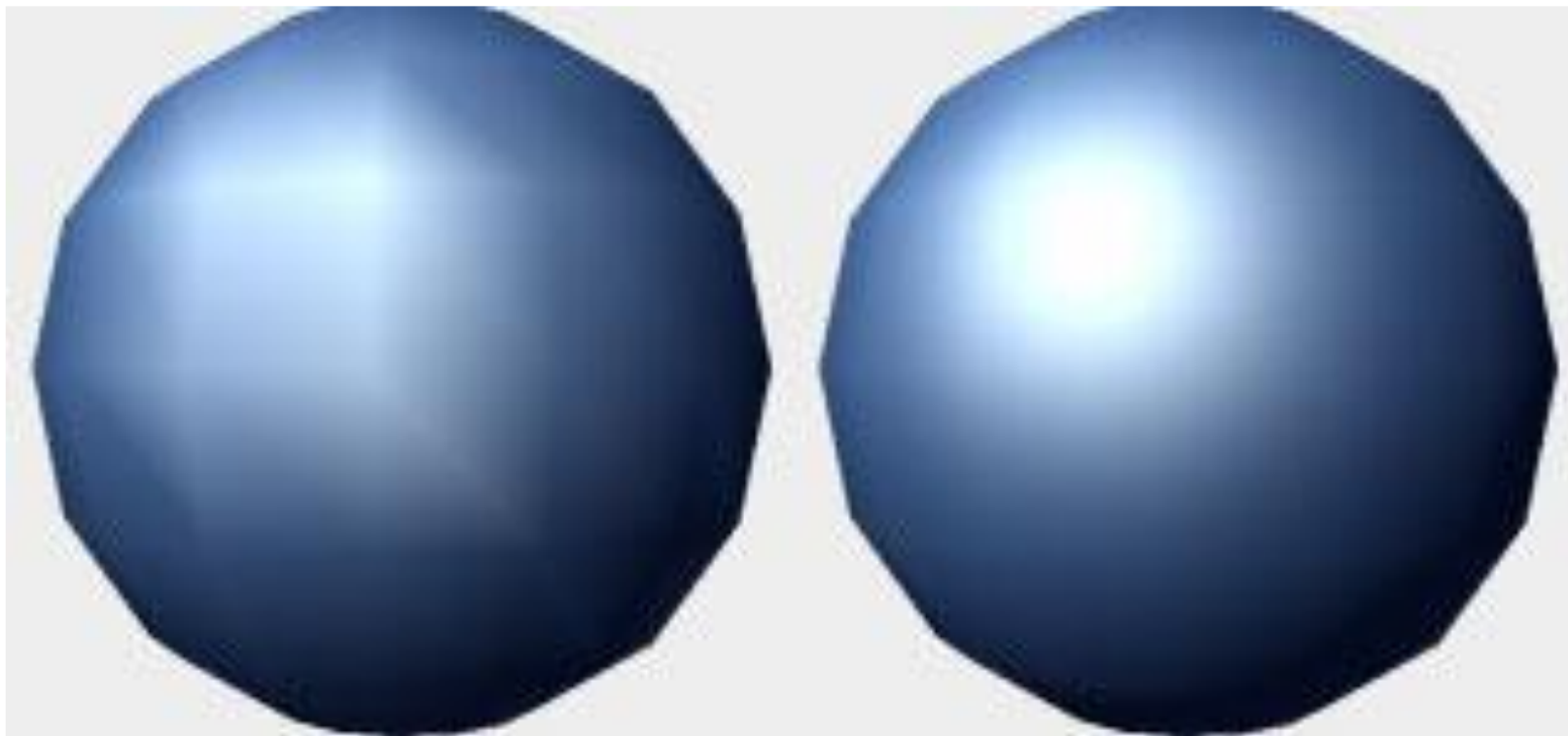
Lumina speculară

```
float specularStrength = 0.5;  
  
vec3 viewDir = normalize(_ViewPos - PositionWorld);  
vec3 reflectDir = reflect(-lightDir, norm);  
  
float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32.0);  
vec3 specular = specularStrength * spec * _LightColor;  
  
vec3 result = (ambient + diffuse + specular) * _ObjectColor;  
FragColor = vec4(result, 1.0);
```

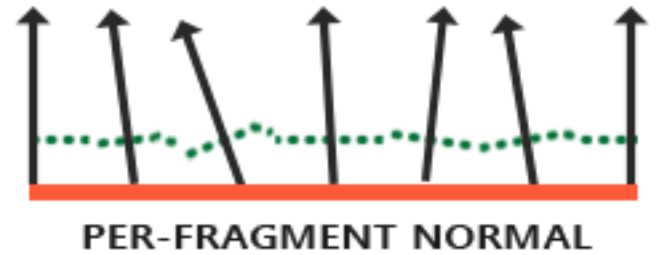
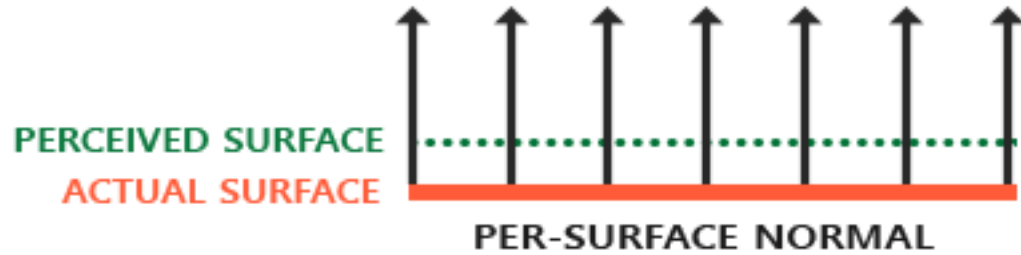




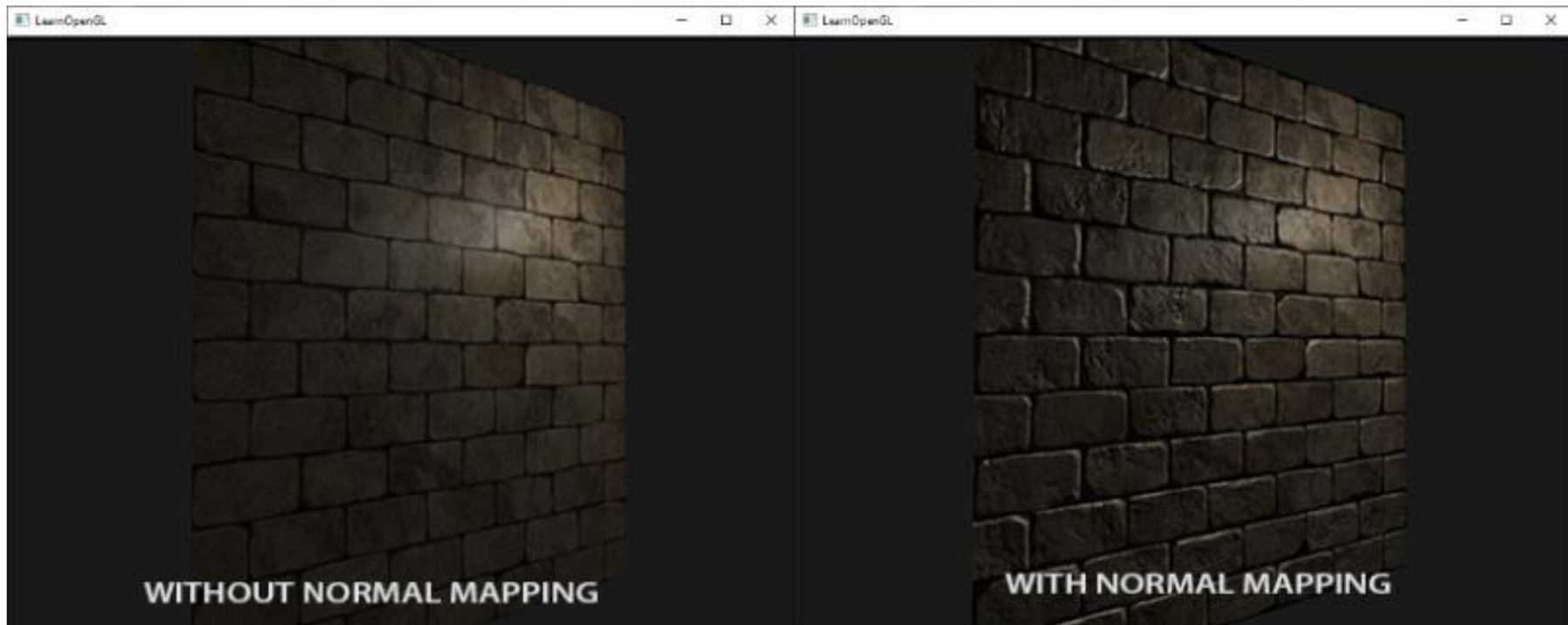
Modelul de iluminare Phong

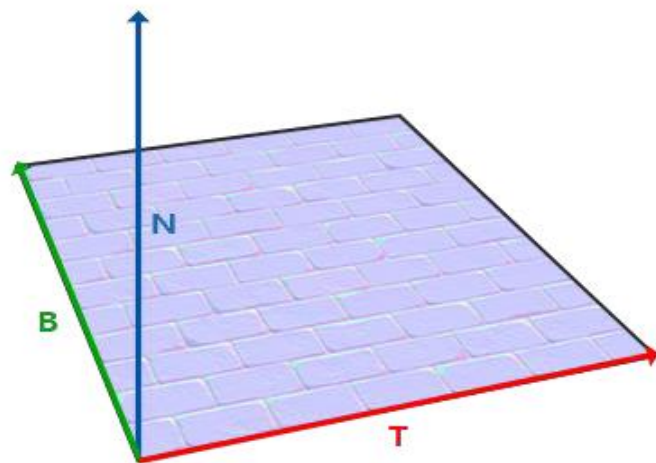
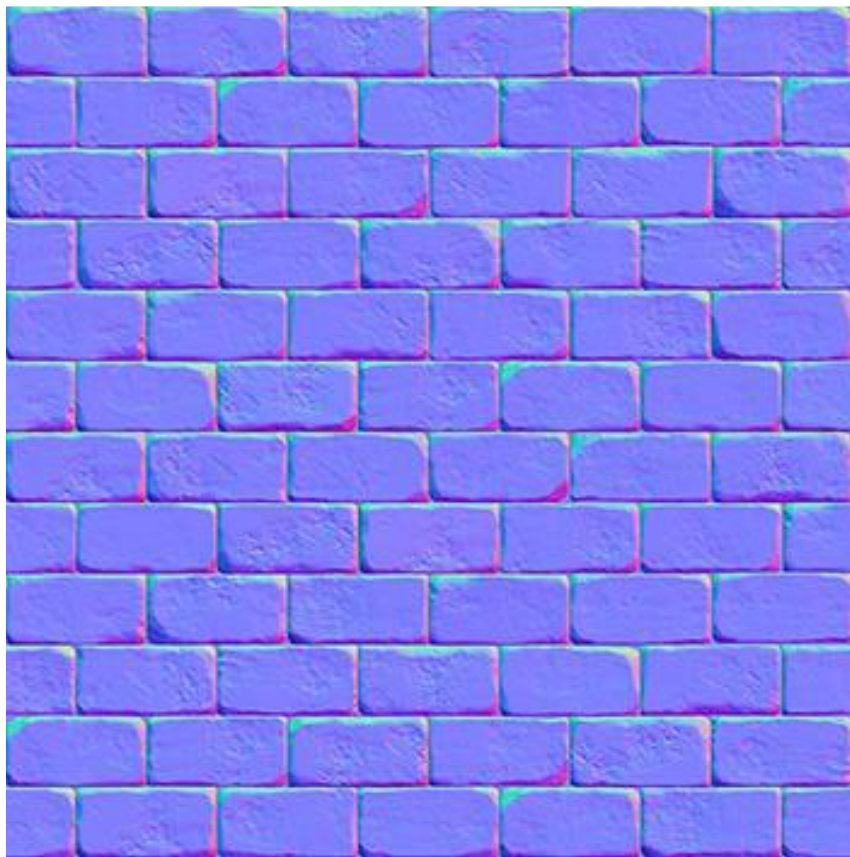


Iluminare per vârf/per fragment



Normal mapping





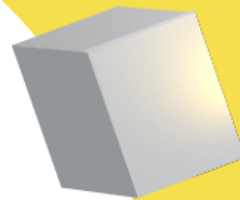
04

OpenGL Avansat

Face culling

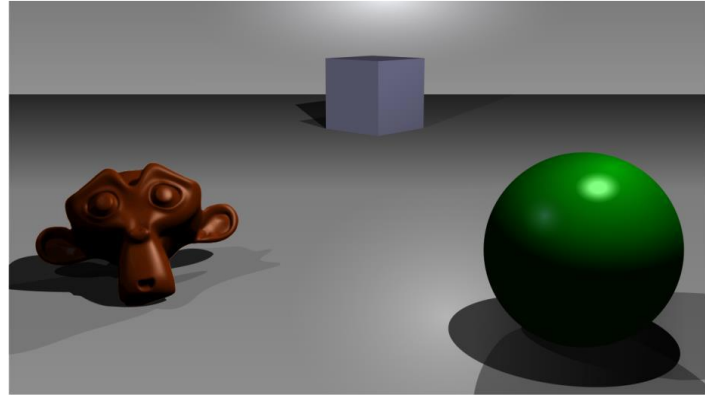
Blending & transparență

Depth buffers și Frame buffers

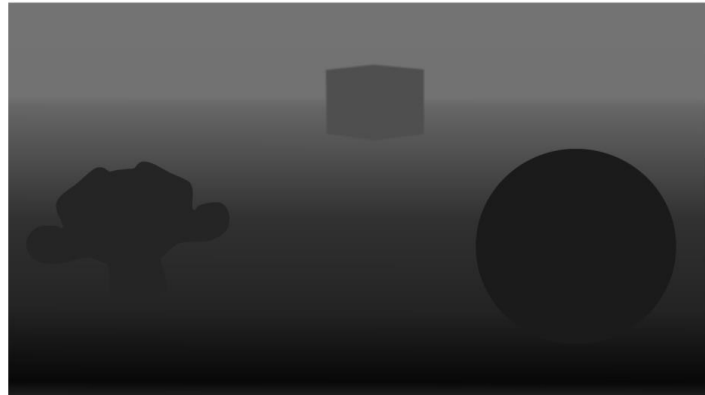


Depth buffer

<https://www.youtube.com/watch?v=zit45k6CUMk>



A simple three-dimensional scene



Z-buffer representation

Blending & Transparență



Full transparent window

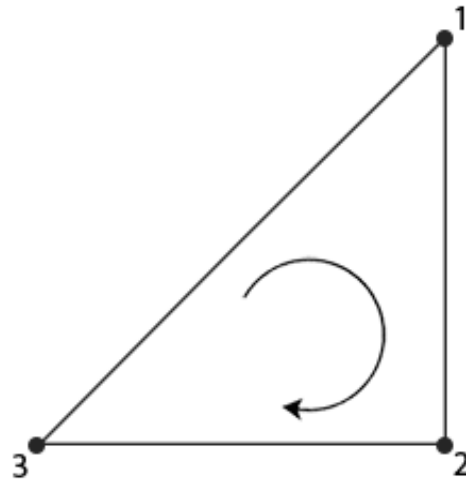


Partially transparent window

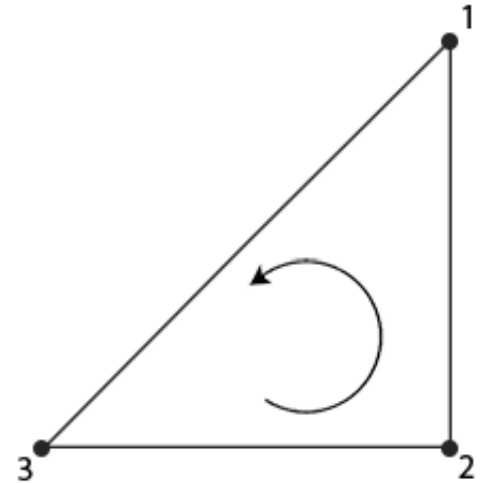
<https://learnopengl.com/Advanced-OpenGL/Blending>

Face culling

<https://learnopengl.com/Advanced-OpenGL/Face-culling>



Clockwise
1 -> 2 -> 3



Counter-clockwise
1 -> 3 -> 2

Post Processing



https://www.youtube.com/watch?v=aGsUU_bvOgw

<https://learnopengl.com/Advanced-Lighting/Bloom>

<https://catlikecoding.com/unity/tutorials/custom-srp/post-processing/>

Resurse

Jeremiah, “Learning DirectX 12 – Lesson 4 – Textures,” 3D Game Engine Programming, Jun. 26, 2021. <https://www.3dgep.com/learning-directx-12-4/>

“Learn OpenGL, extensive tutorial resource for learning Modern OpenGL.”
<https://learnopengl.com/>

“OpenGL 4.0 on Linux Tutorials.” <https://rastertek.com/tutgl4linux.html>

Curs - „Grafică pe calculator” – UB, FMI, Informatică An 3

Curs – „Elemente de Grafică pe Calculator” – UPB, ACS

