

Programarea aplicațiilor de simulare

Efecte de bază





Conținut

01. Recapitulare

- Transformări
- Geometrie
- Pipeline basic

02. Shadere

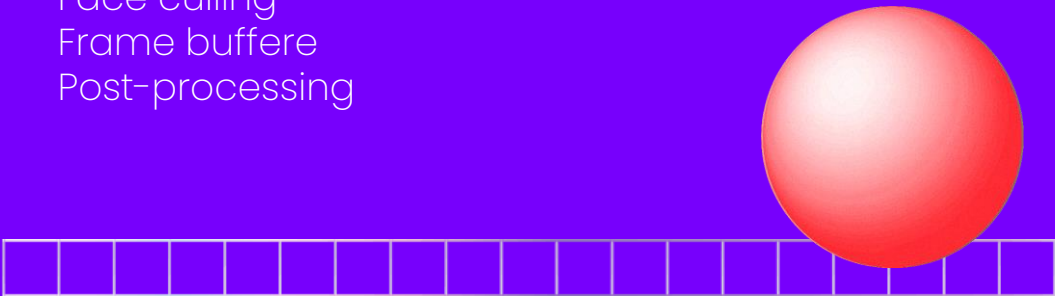
- Atribute
- Uniforme
- Exemple

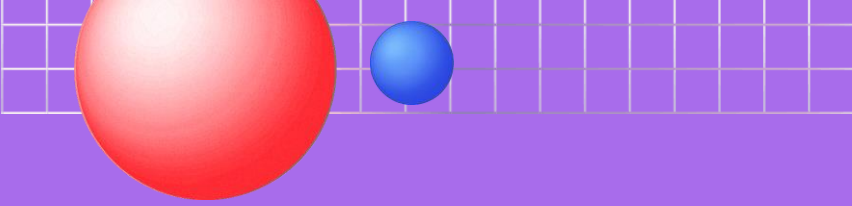
03. Efecte comune

- Texturare
- Modelul de iluminare Phong
- Normal mapping

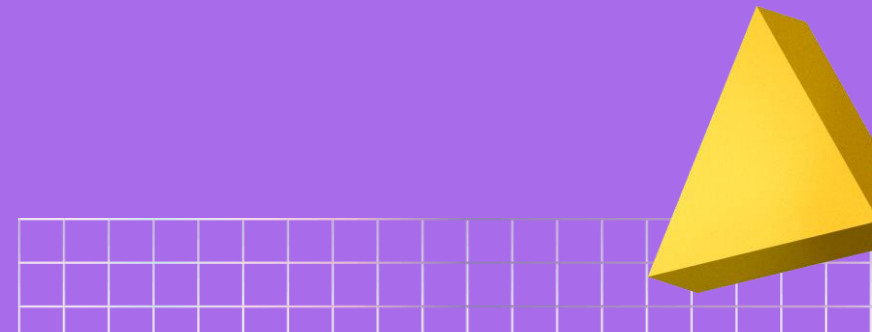
04. OpenGL avansat

- Depth buffere
- Blending & transparență
- Face culling
- Frame buffere
- Post-processing



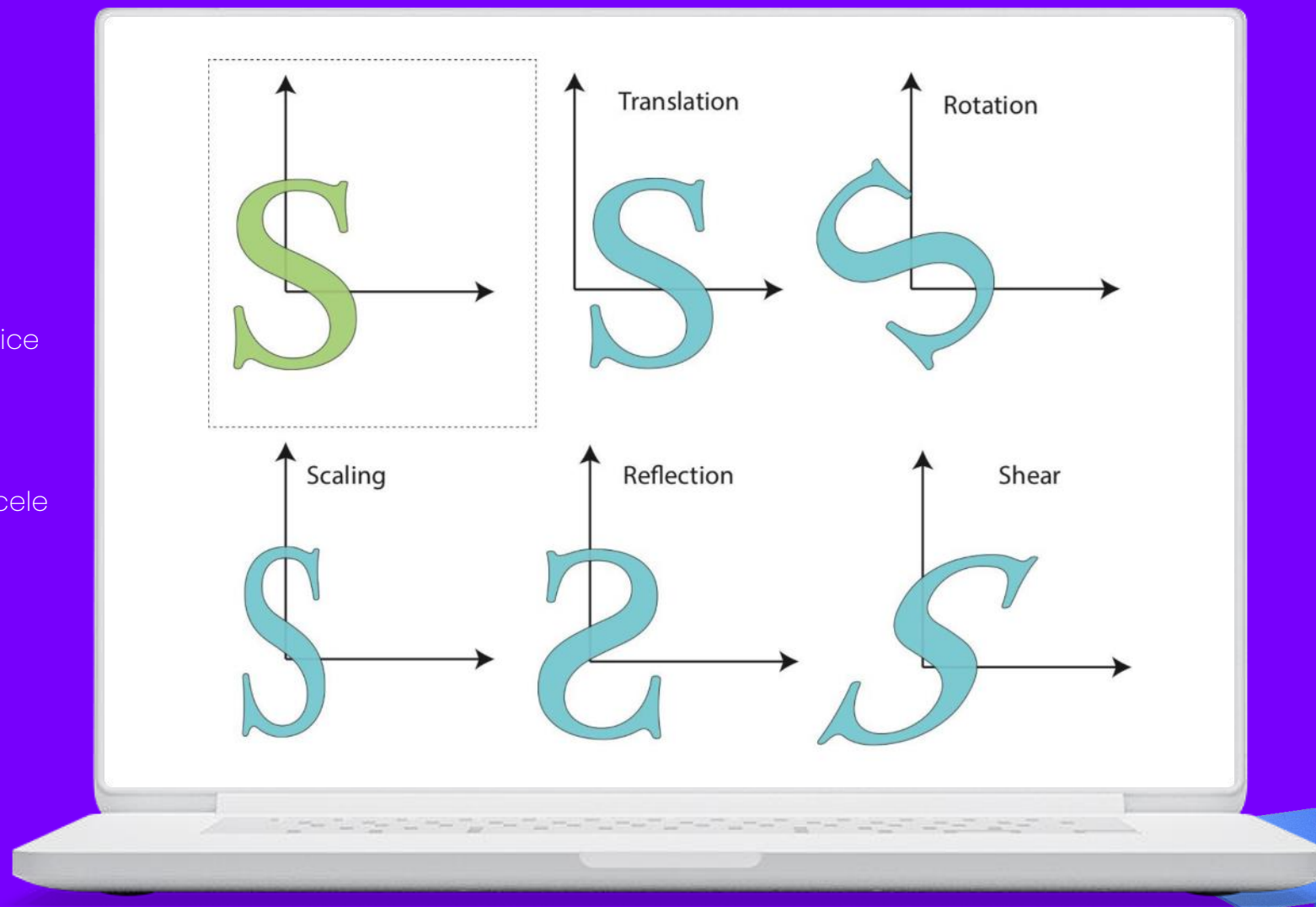


Recapitulare



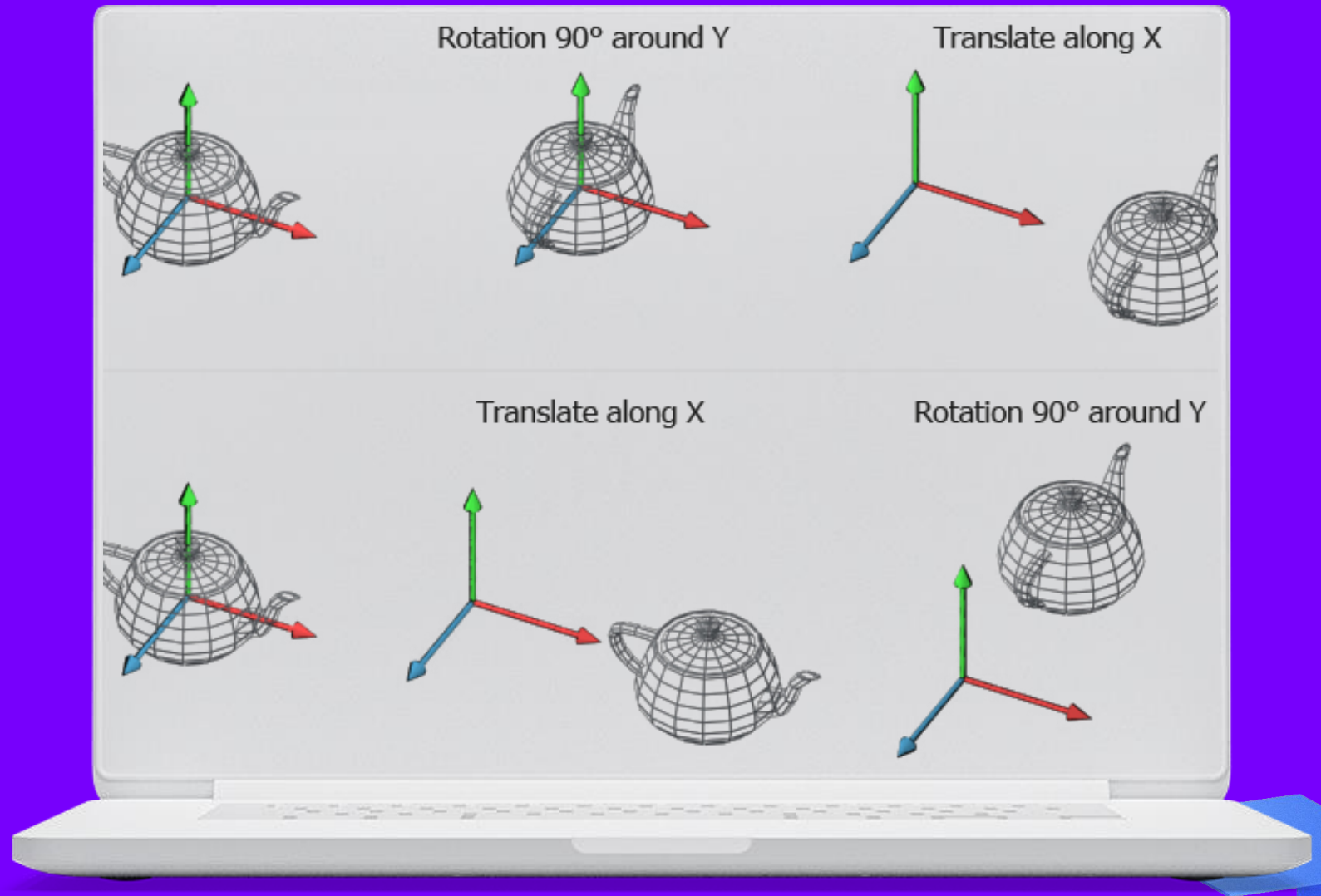
Transformări

- Mai multe tipuri de transformări.
- Pot fi reprezentate folosind matrice (formulele se găsesc în cursul anterior)
- Pentru a combina mai multe transformări se înmulțesc matricele transformărilor.



Transformări

Ordinea transformărilor contează!



World Space

Pentru fiecare obiect se construiește o matrice numită **World Matrix** care transformă coordonatele din **Object Space** în **World Space**.

Astfel, modelele sunt plasate în lume.

Matricea **World Matrix** înglobează transformările aplicate asupra obiectului.



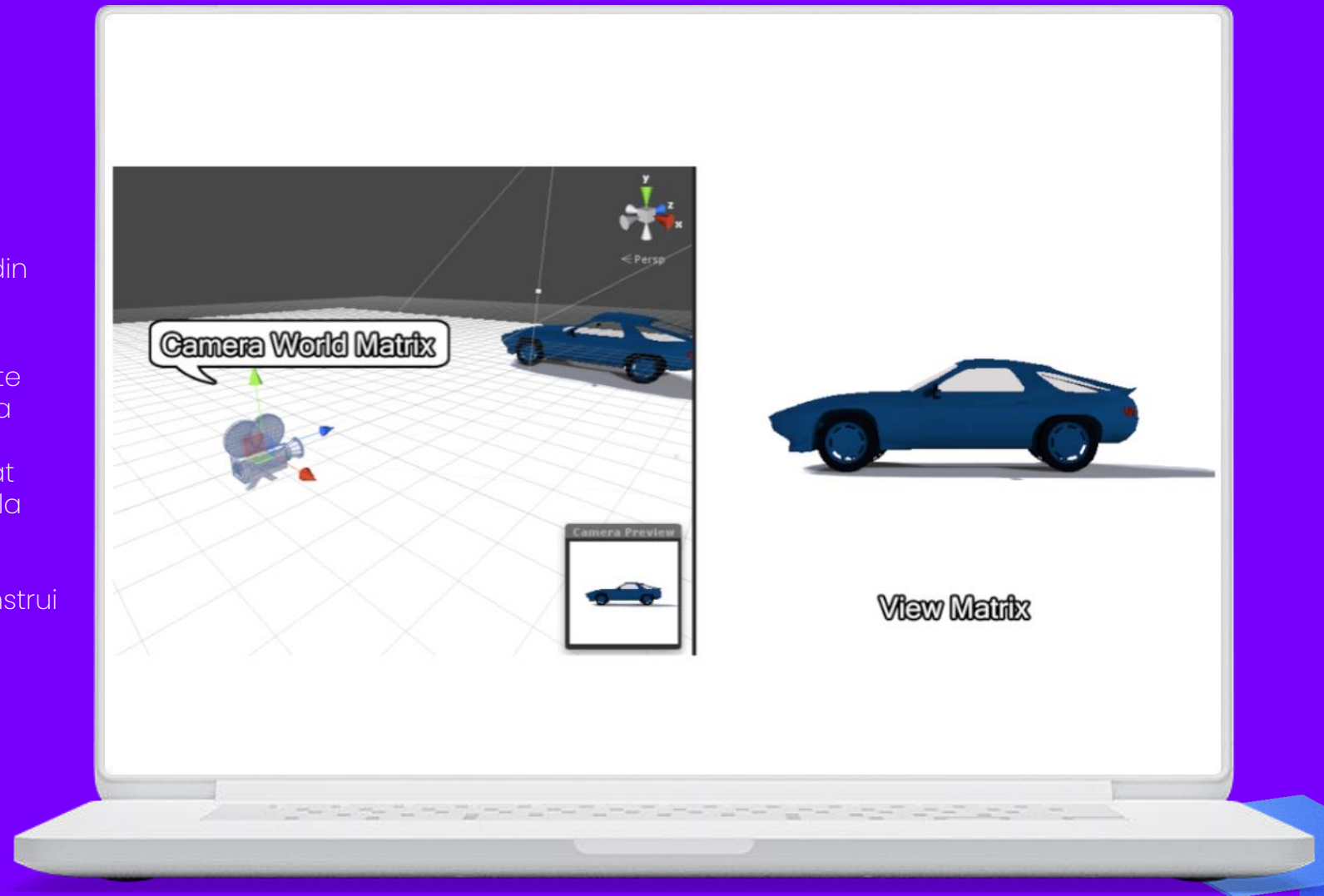
View Space

Coordonatele sunt transformate din **World Space** în **View Space**.

Se folosește **View Matrix**, care este inversa matricei de transformare a camerei. Practic, tot ce se vede în scenă este transformat astfel încât toate coordonatele să fie relative la cameră.

Există mai multe metode de a construi o astfel de matrice.

Dacă vectorii care formează o matrice sunt perpendiculari și au lungimea 1, atunci $M^{-1} = M^T$



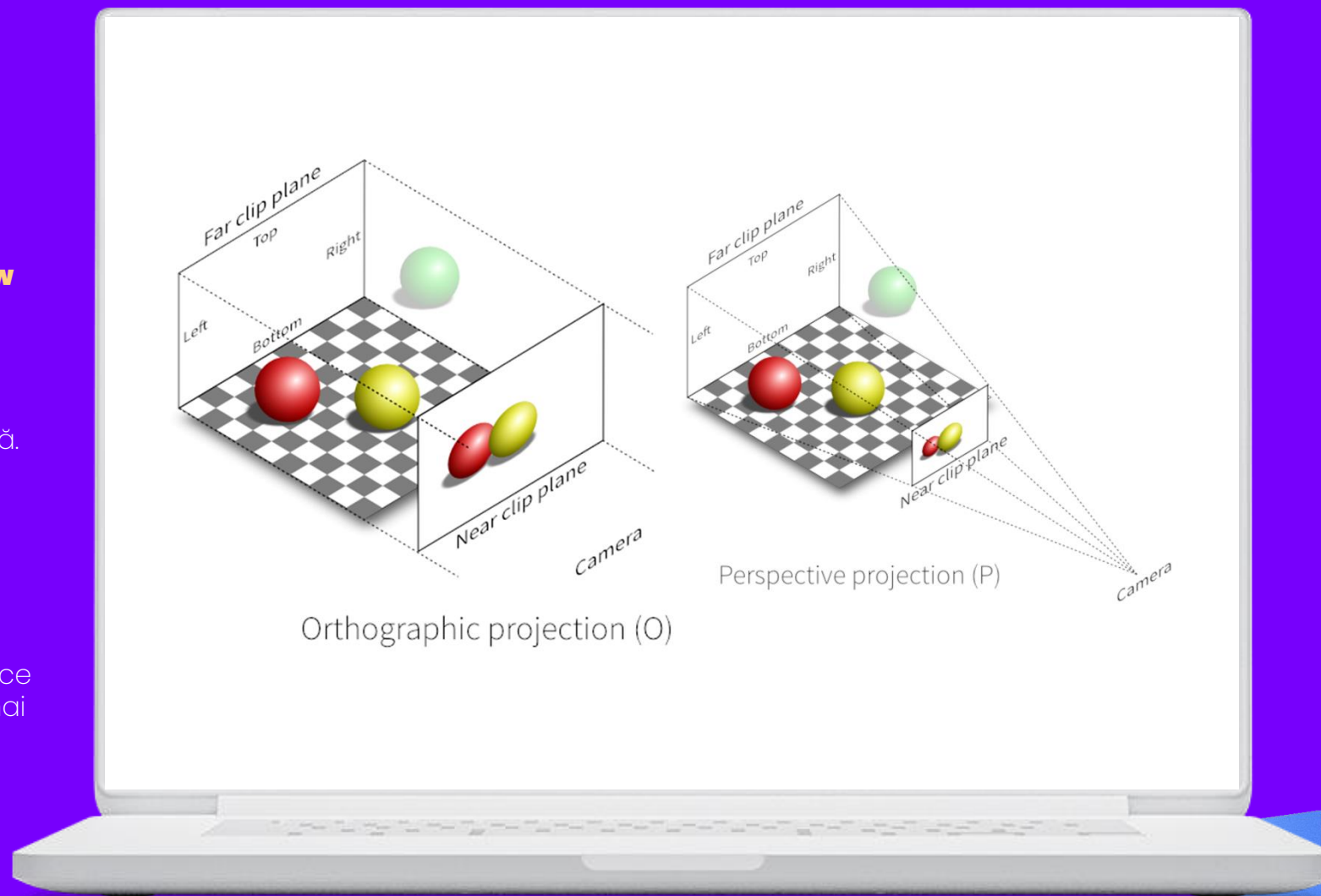
Proiecție

Transformă coordonatele din **View Space** în **Clip Space**.

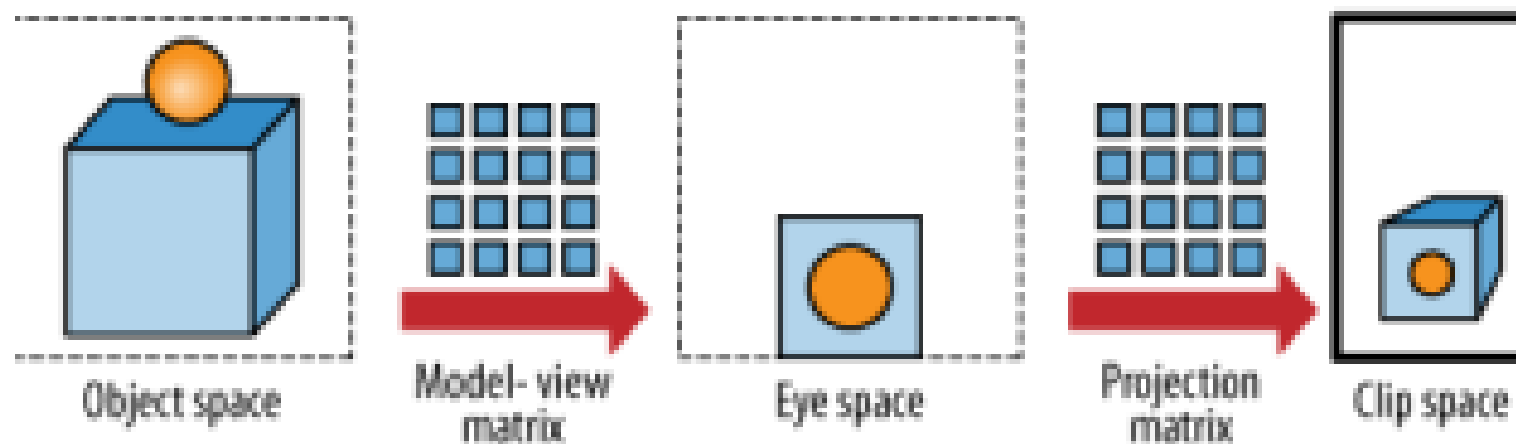
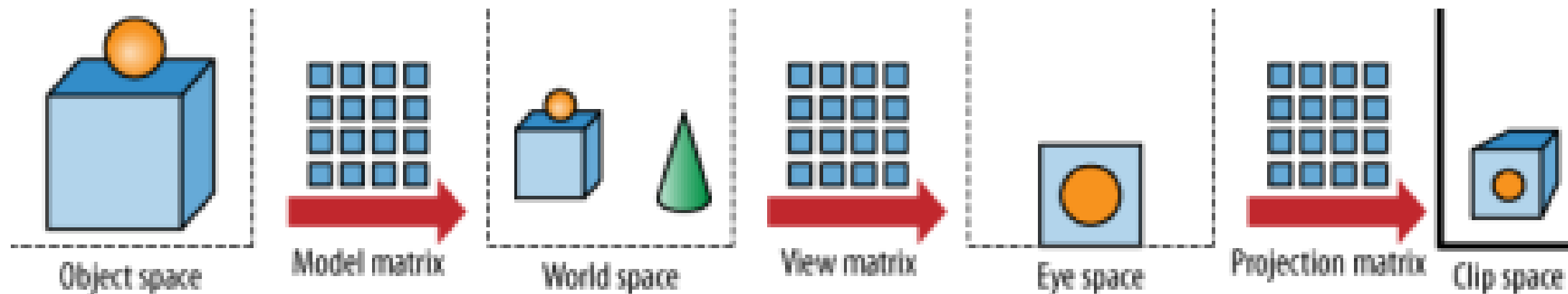
Există două tipuri standard de proiecție: ortografică și perspectivă.

Cea ortografică păstrează liniile paralele la depărtare.

Cea perspectivă este realistă și face ca obiectele depărtate să pară mai mici.

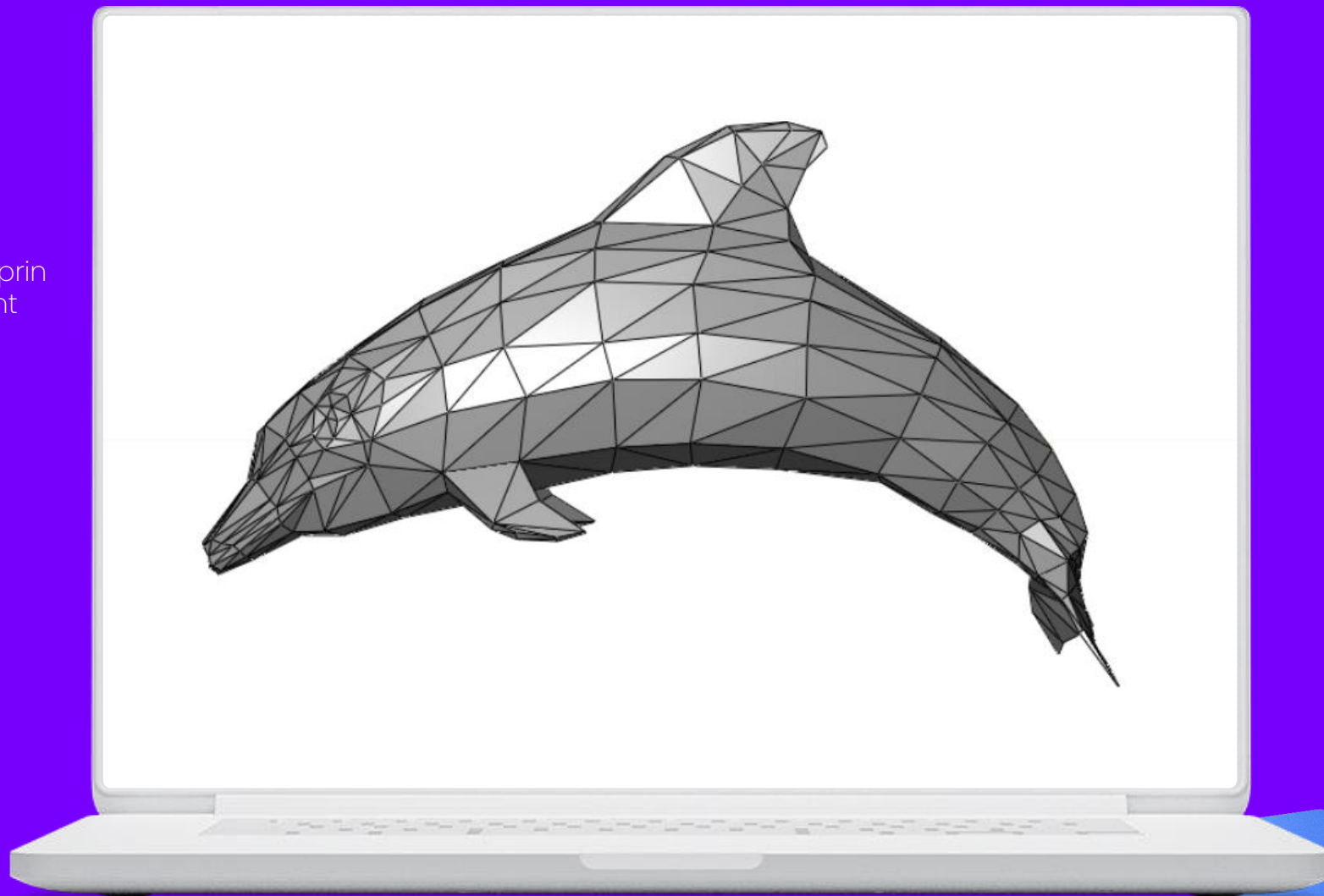


$$P_{clip} = M_{proj}M_{view}M_{world}P_{object}$$

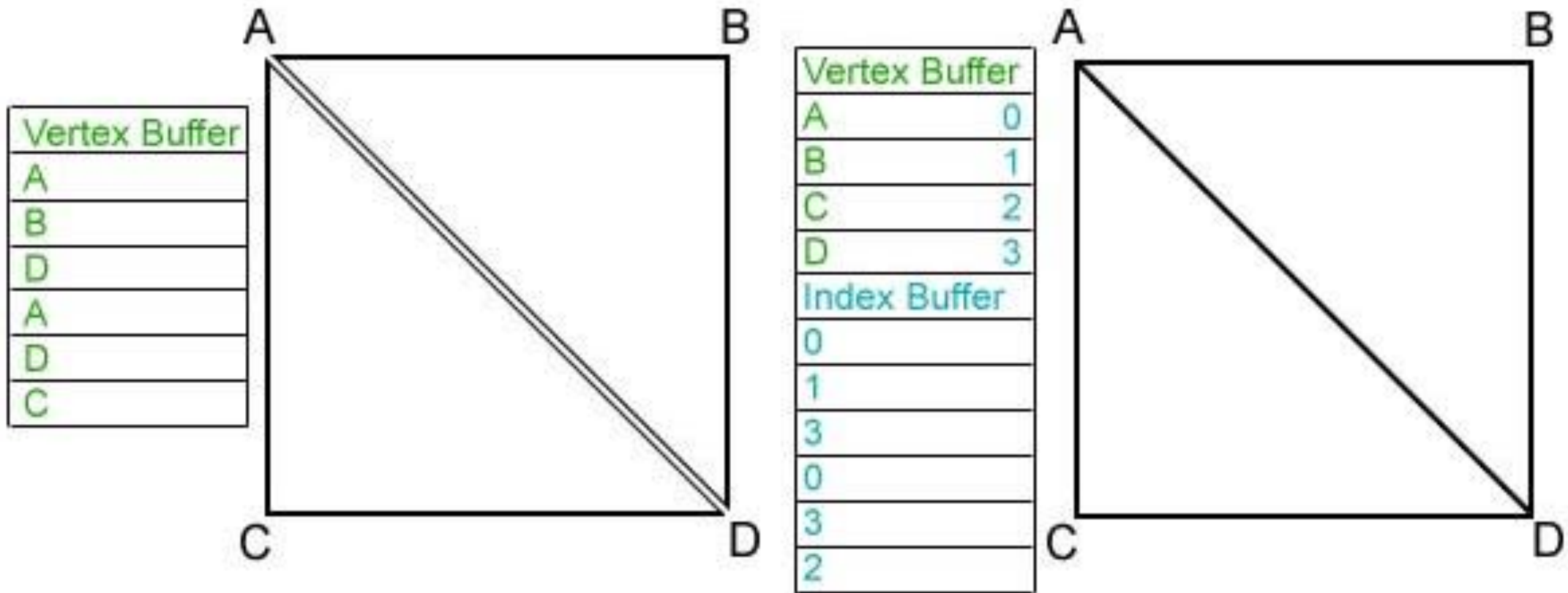


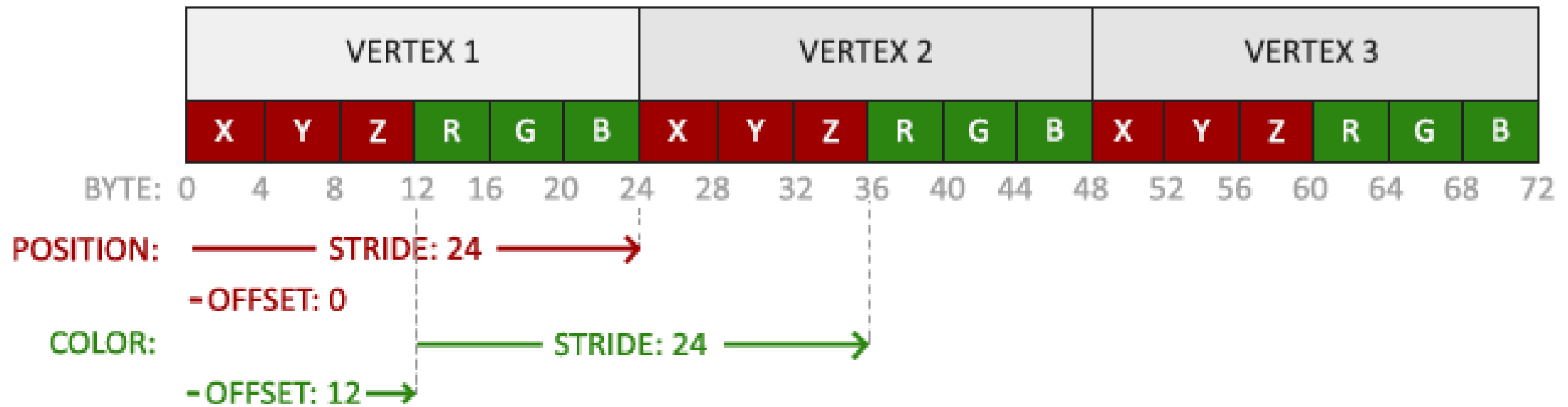
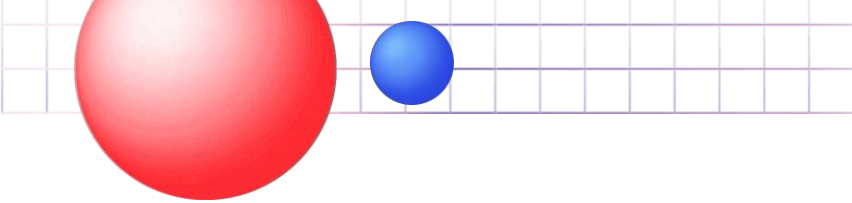
Poligoane

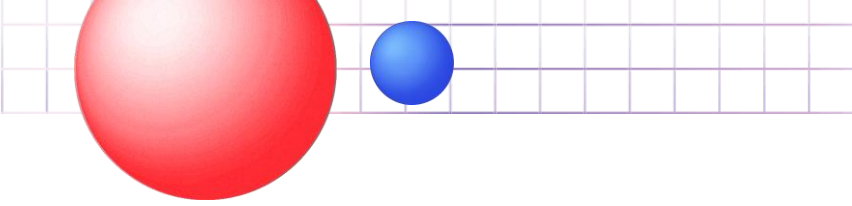
Putem defini obiecte în 2D și în 3D prin definirea poligoanelor din care sunt alcătuite.



Buffere de vertecși și indici

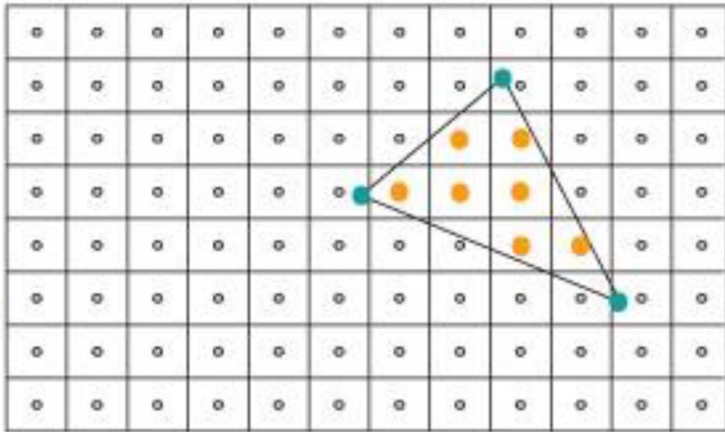




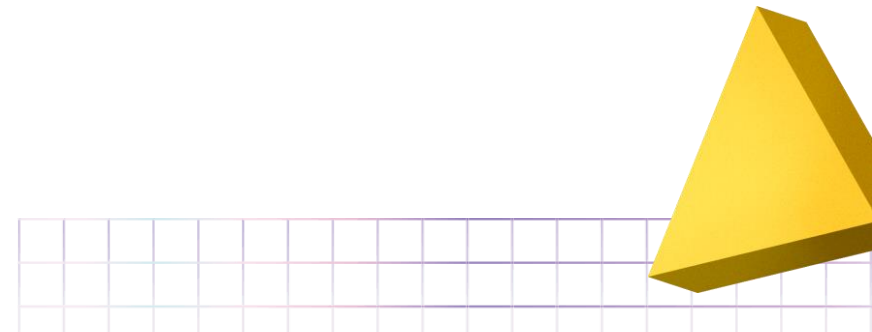
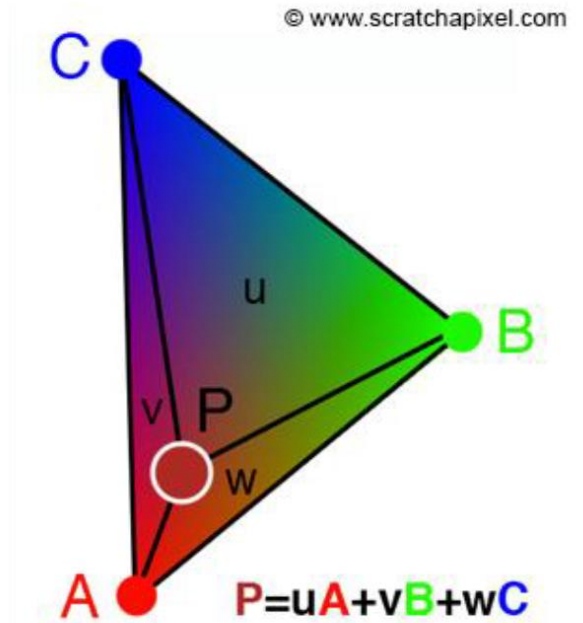


Rasterizzare

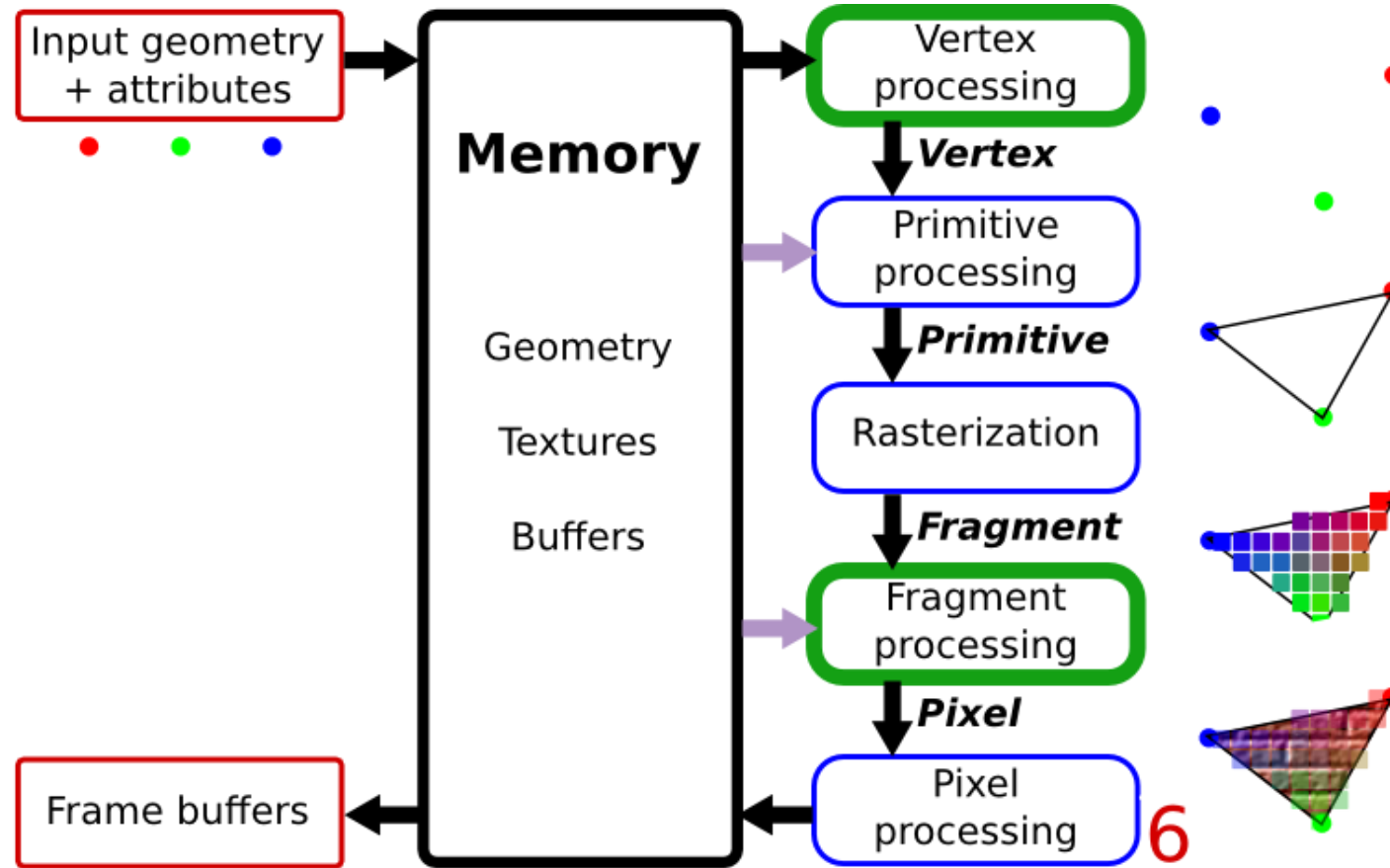
Vertices



Fragments



OpenGL Pipeline



```
/////////////////////////////////////////////////////////////////
// Filename: Color.frag
/////////////////////////////////////////////////////////////////

#version 400

/////////////////////////////////////////////////////////////////
// INPUT VARIABLES //
/////////////////////////////////////////////////////////////////
in vec3 InputPosition;
in vec3 InputColor;

/////////////////////////////////////////////////////////////////
// OUTPUT VARIABLES //
/////////////////////////////////////////////////////////////////
out vec3 Color;

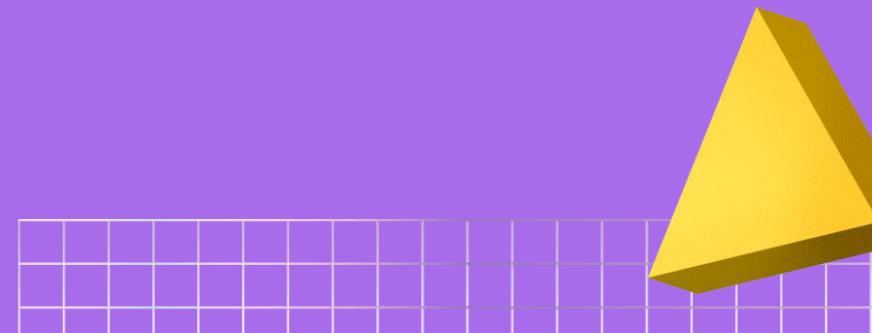
/////////////////////////////////////////////////////////////////
// UNIFORM VARIABLES //
/////////////////////////////////////////////////////////////////
uniform mat4 _WorldMatrix;
uniform mat4 _ViewMatrix;
uniform mat4 _ProjectionMatrix;

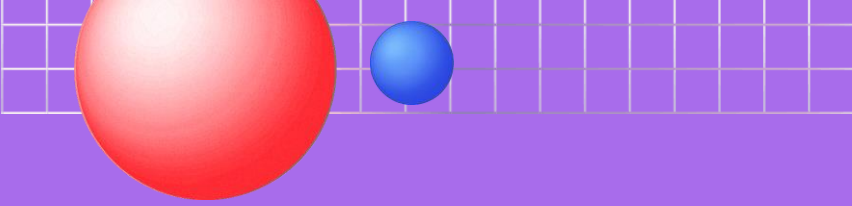
/////////////////////////////////////////////////////////////////
// Vertex Shader
/////////////////////////////////////////////////////////////////
void main(void)
{
    // Calculate the position of the vertex against the world, view, and projection matrices.
    gl_Position = _WorldMatrix * vec4(InputPosition, 1.0f);
    gl_Position = _ViewMatrix * gl_Position;
    gl_Position = _ProjectionMatrix * gl_Position;

    // Store the input color for the pixel shader to use.
    Color = InputColor;
}
```



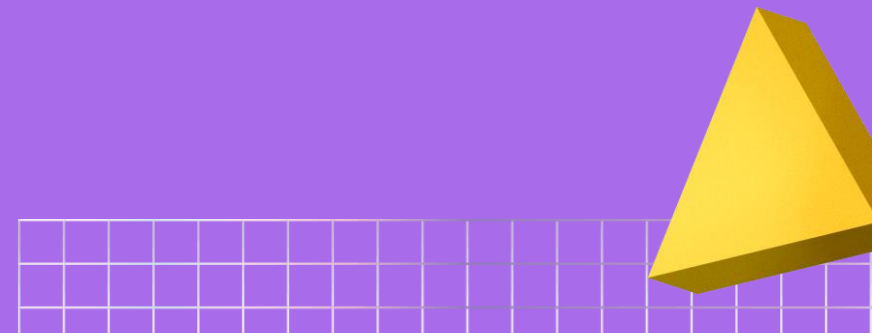

Shadere

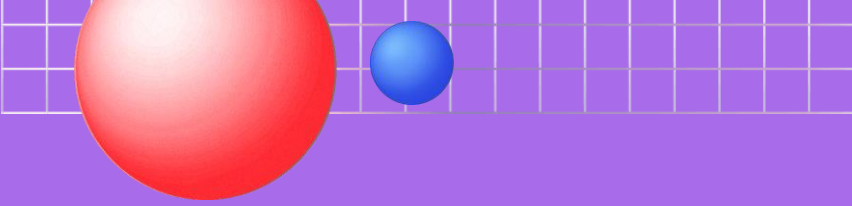




Atribute

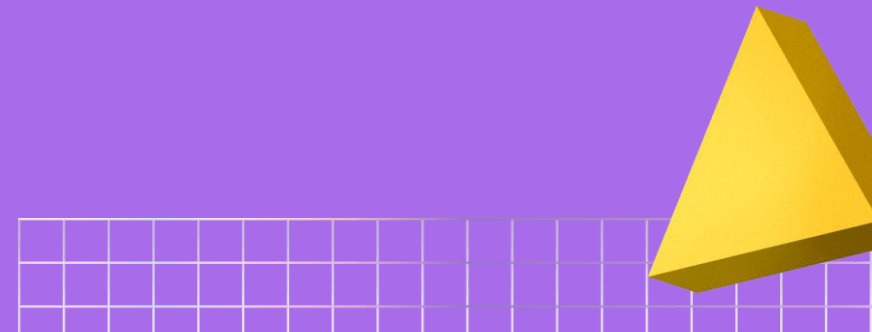
- **Pe lângă poziție, fiecărui vertex îi putem atribui diferite proprietăți.**
- **Aceste proprietăți se numesc atribute.**
- **Exemple de atribute:**
 - ***Culoare, coordonate de textură, normale, tangente, etc..***

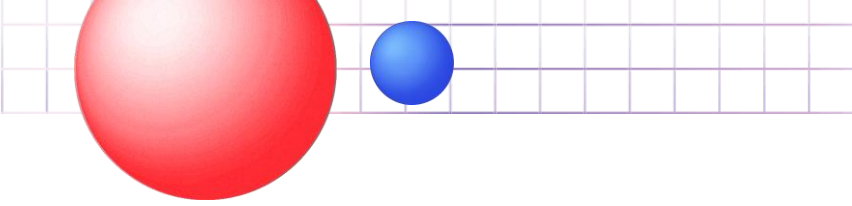




Vertex Buffere

- **Se folosesc Buffere de vertecși pentru a încărca în memoria plăcii video toate attributele vertecșilor**



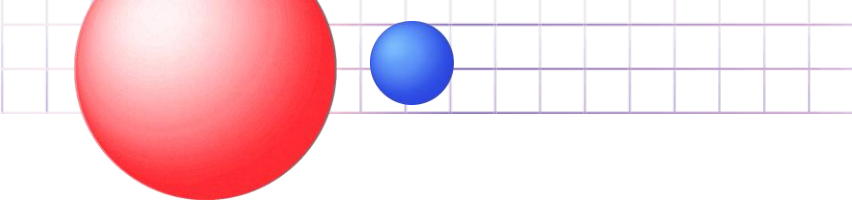


P	N	T	X	P	N	T	X	P	N	T	X	P	N	T	X
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

vs

P	P	P	P	N	N	N	N	T	T	T	T	X	X	X	X
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

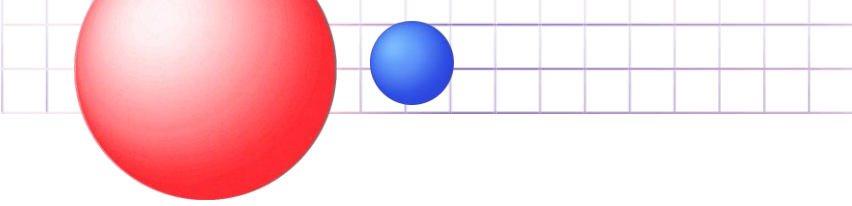




P	P	P	P	N	T	X	N	T	X	N	T	X	N	T	X
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Are sens?

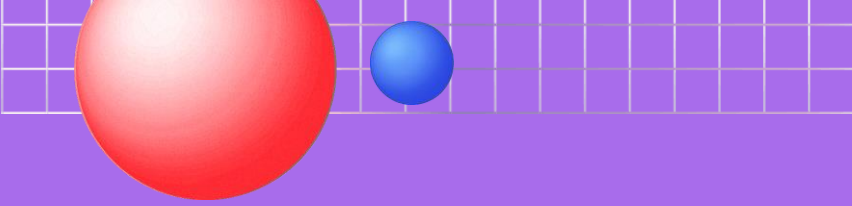




P	P	P	P	N	T	X	N	T	X	N	T	X	N	T	X
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

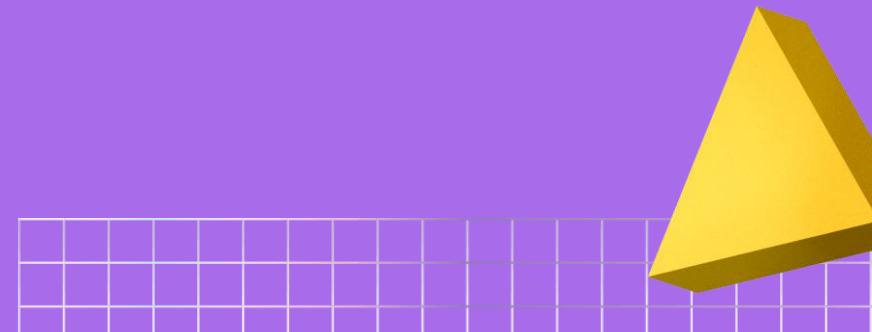
Are sens?





Uniforme

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



Vertex Shader

```
#version 400
layout in vec3 InputPosition;

// The matrices are also uniforms.
uniform mat4 _WorldMatrix;
uniform mat4 _ViewMatrix;
uniform mat4 _ProjectionMatrix;

void main()
{
    gl_Position = _WorldMatrix * vec4(InputPosition, 1.0f);
    gl_Position = _ViewMatrix * gl_Position;
    gl_Position = _ProjectionMatrix * gl_Position;
}
```

Fragment Shader

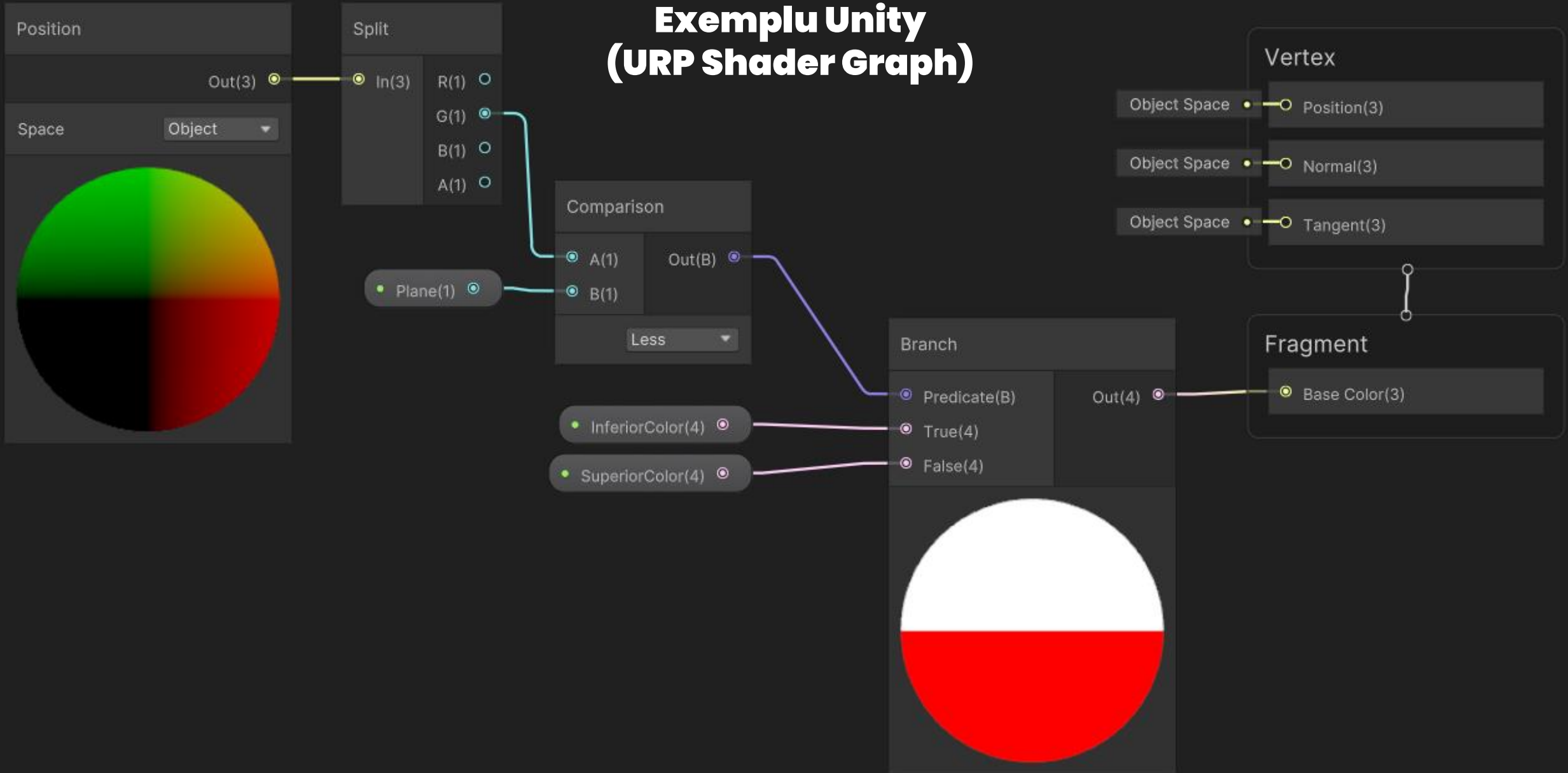
```
#version 400

uniform vec3 Color; // using a uniform for the color of the triangle

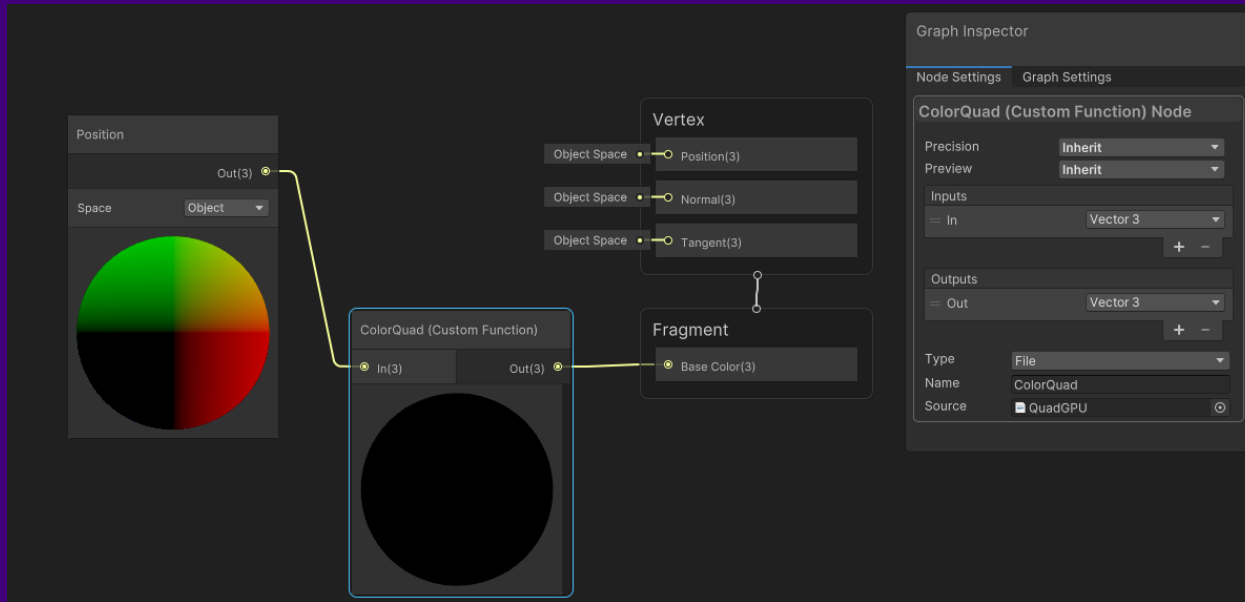
out vec4 OutputColor;

void main(void)
{
    OutputColor = vec4(Color, 1.0f);
}
```

Exemplu Unity (URP Shader Graph)



Sau



```
void ColorQuad_float(in float3 In, out float3 Out)
{
    if (In.y < _Plane)
        Out = _InferiorColor;
    else
        Out = _SuperiorColor;
}
```

```
void ColorQuad_half(in half3 In, out half3 Out)
{
    if (In.y < _Plane)
        Out = _InferiorColor;
    else
        Out = _SuperiorColor;
}
```

Exemplu OpenGL (GLSL)

Vertex Shader

```
#version 400

in vec3 InputPosition;

out vec3 PositionObject;

uniform mat4 _WorldMatrix;
uniform mat4 _ViewMatrix;
uniform mat4 _ProjectionMatrix;

void main(void)
{
    gl_Position = _WorldMatrix * vec4(InputPosition, 1.0f);
    gl_Position = _ViewMatrix * gl_Position;
    gl_Position = _ProjectionMatrix * gl_Position;

    PositionObject = InputPosition;
}
```

Fragment Shader

```
#version 400

in vec3 PositionObject;

uniform float _Plane;
uniform vec3 _InferiorColor;
uniform vec3 _SuperiorColor;

out vec4 OutputColor;

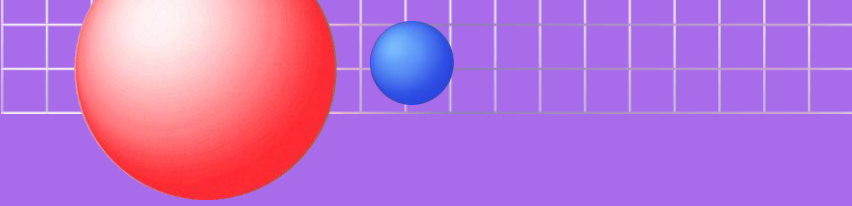
void main(void)
{
    OutputColor = _SuperiorColor;

    if (PositionObject.y > _Plane)
        OutputColor = _InferiorColor;
}
```



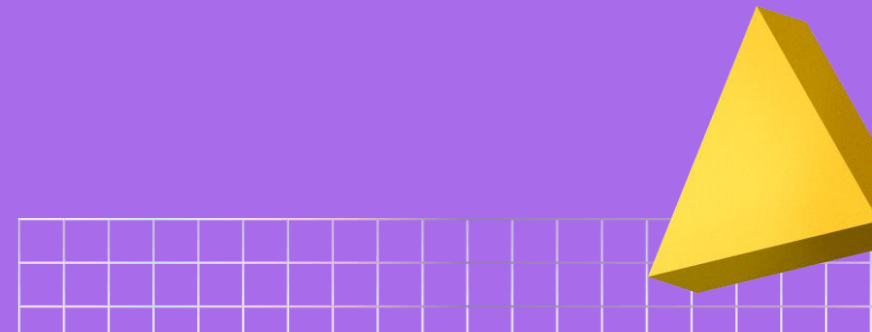
Efecte comune



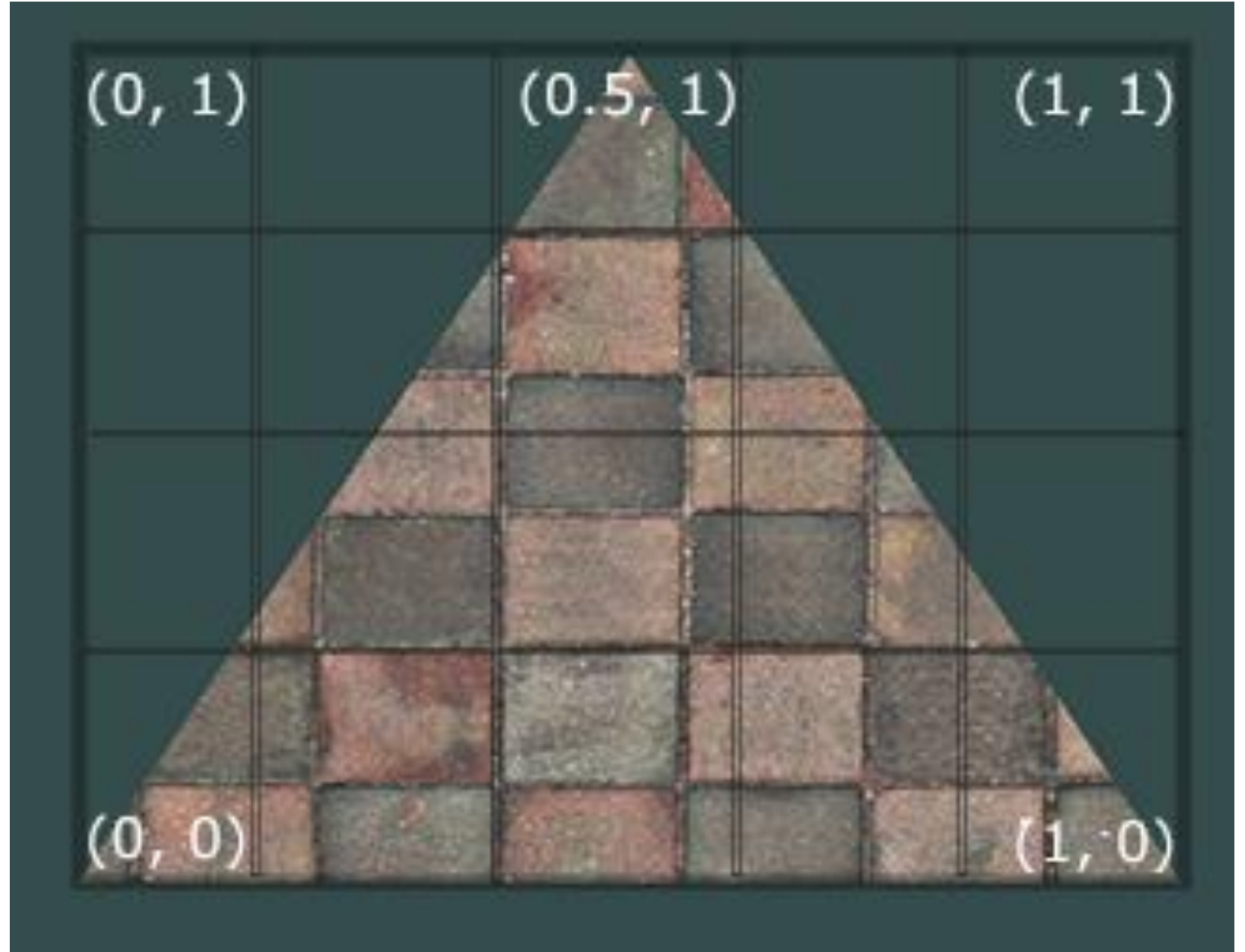


Texturare

- **O textură este o imagine proiectată peste un poligon.**
- **Pe un poligon se afișează doar o secțiune a imaginii.**



Coordonate de textură



Coordonate de textură

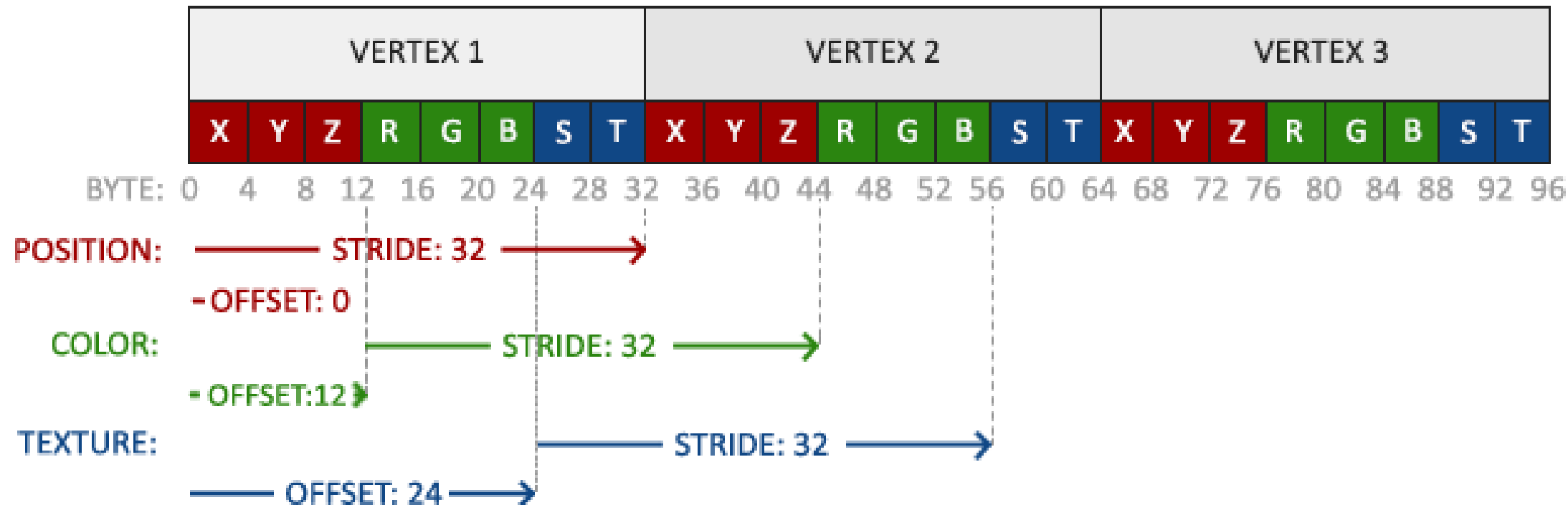
```
float vertices[] =
{
    // positions      // colors      // texture coords
    0.5f,  0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f, // top right
    0.5f, -0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, // bottom right
    -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, // bottom left
    -0.5f,  0.5f,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

- Coordonatele de textură sunt atribuite fiecărui vertex și le încărcăm în buffer-ul de vertecși.

```
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);

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



Texture wrapping



GL_REPEAT



GL_MIRRORED_REPEAT

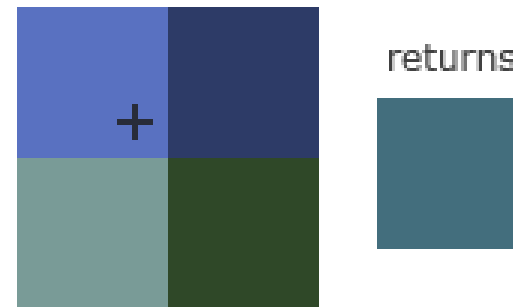
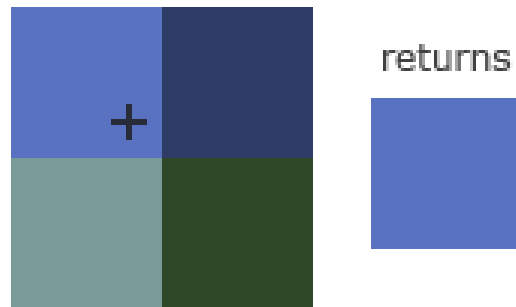


GL_CLAMP_TO_EDGE



GL_CLAMP_TO_BORDER

Texture filtering



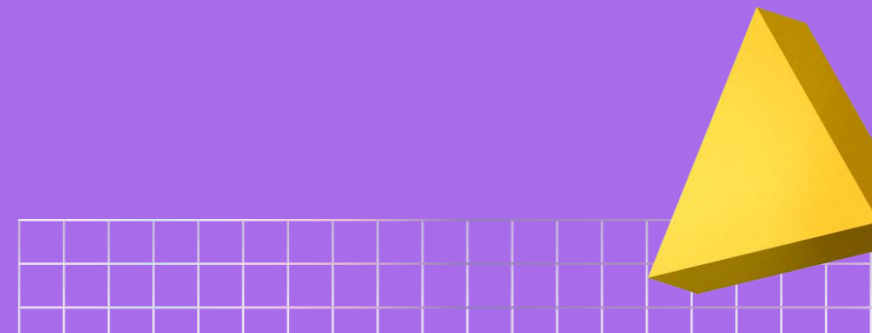
GL_NEAREST



GL_LINEAR



Mipmapping





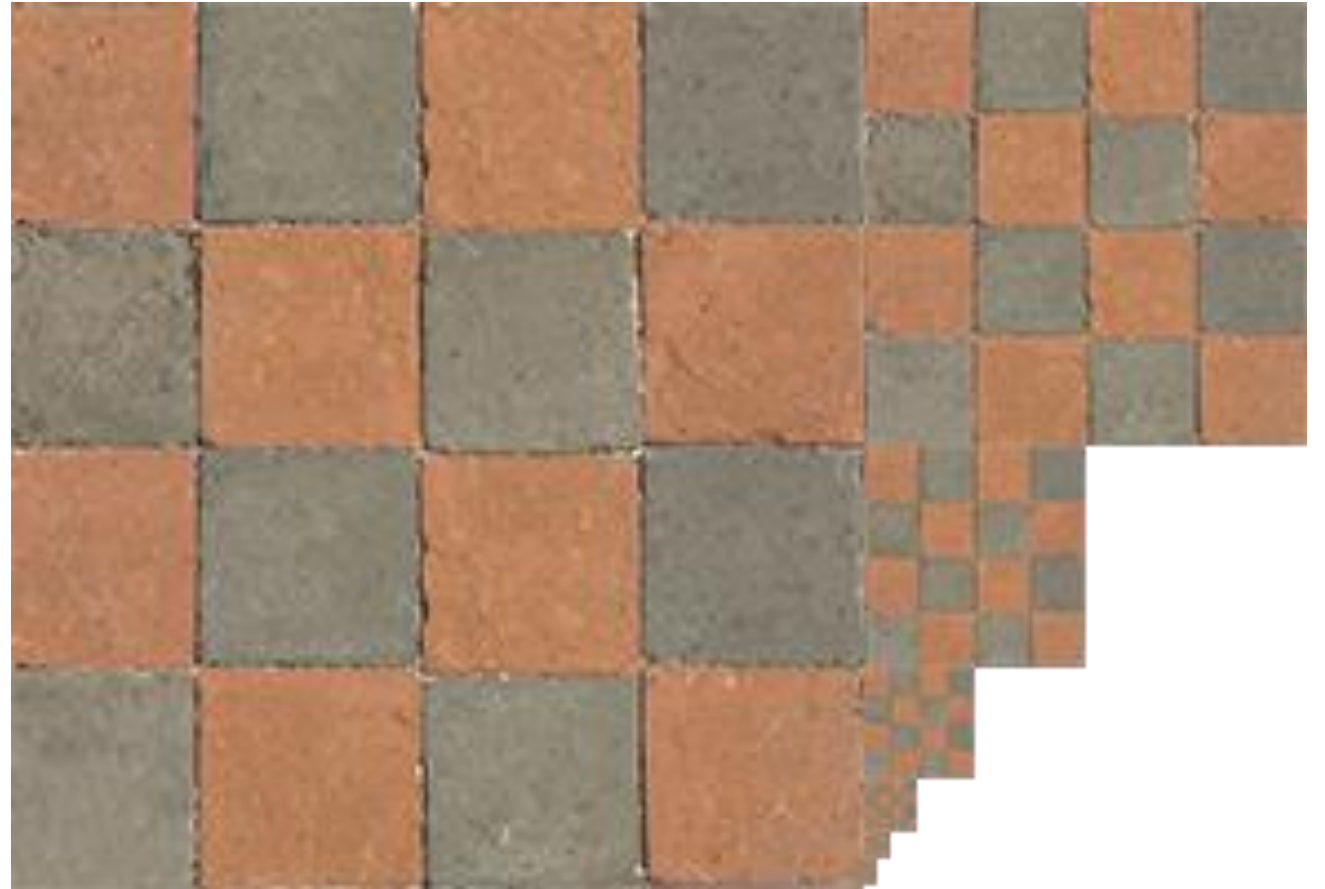
No Mipmapping

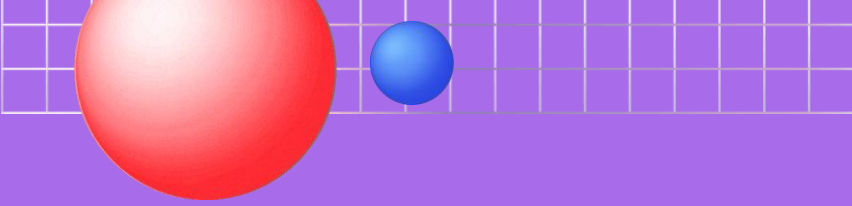


With Mipmapping

Soluție

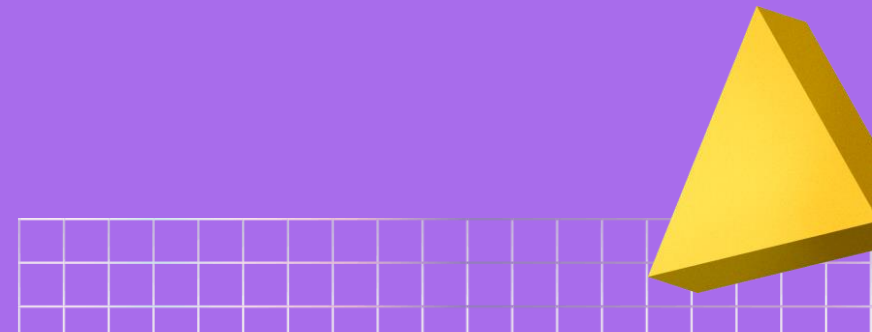
- Se generează mai multe variante ale texturii la rezoluții înjumătățite pe verticală orizontală. În funcție de distanța față de cameră, se alege o textură la o rezoluție mai mare sau mai mică.





Lumină ambientală

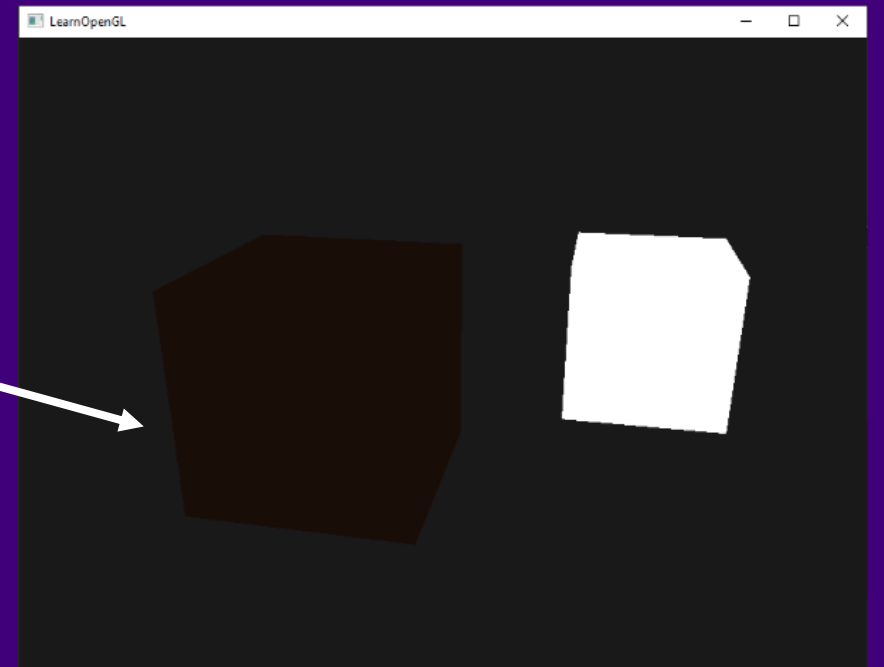
- **Chiar și în întuneric, de regulă tot există câteva raze de lumină care lovesc obiectele și le fac vizibile.**
- **Putem simula acest tip de lumină folosind o valoare constantă.**

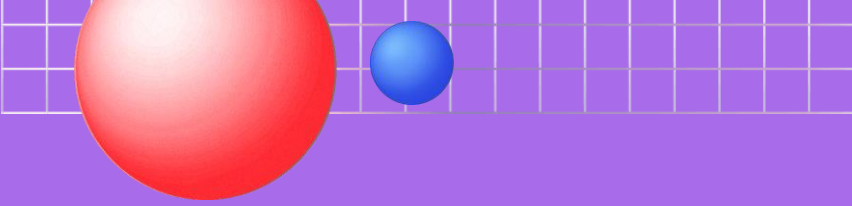


Lumină ambientală

```
void main()
{
    float ambientStrength = 0.1;
    vec3 ambient = ambientStrength * lightColor;

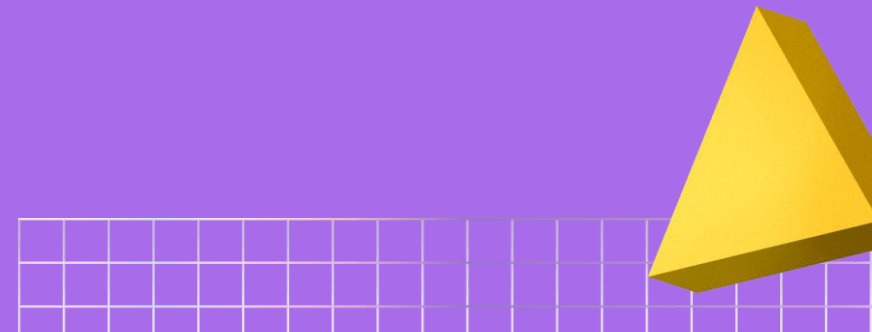
    vec3 result = ambient * objectColor;
    FragColor = vec4(result, 1.0);
}
```

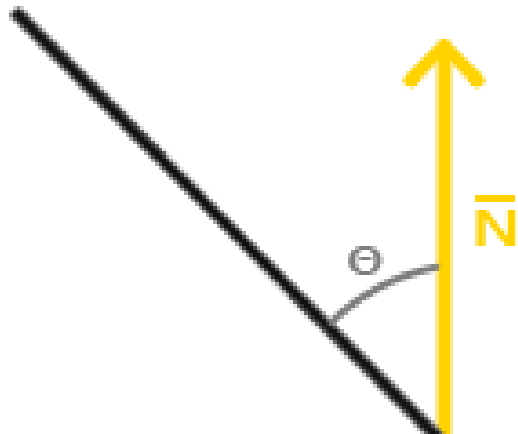
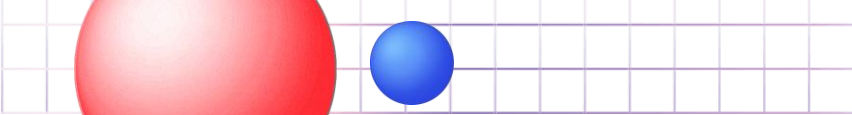


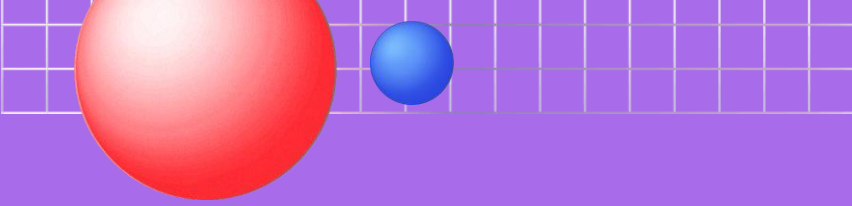


Lumină difuză

- **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**

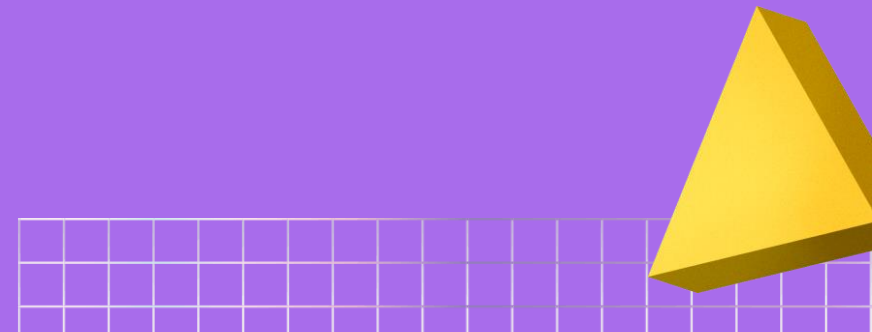






Lumină difuză

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



Lumină difuză

Vertex Shader

```
#version 400
layout in vec3 InputPosition;
layout in vec3 InputNormal;

out vec3 PositionWorld;
out vec3 Normal;

uniform mat4 _WorldMatrix;
uniform mat4 _ViewMatrix;
uniform mat4 _ProjectionMatrix;

void main()
{
    PositionWorld = vec3(_WorldMatrix * vec4(InputPosition, 1.0));
    Normal = mat3(transpose(inverse(_WorldMatrix))) * Normal; // properly transform a normal

    gl_Position = _ProjectionMatrix * _ViewMatrix * vec4(PositionWorld, 1.0);
}
```

Lumină difuză

Fragment Shader

```
#version 400
out vec4 FragColor;

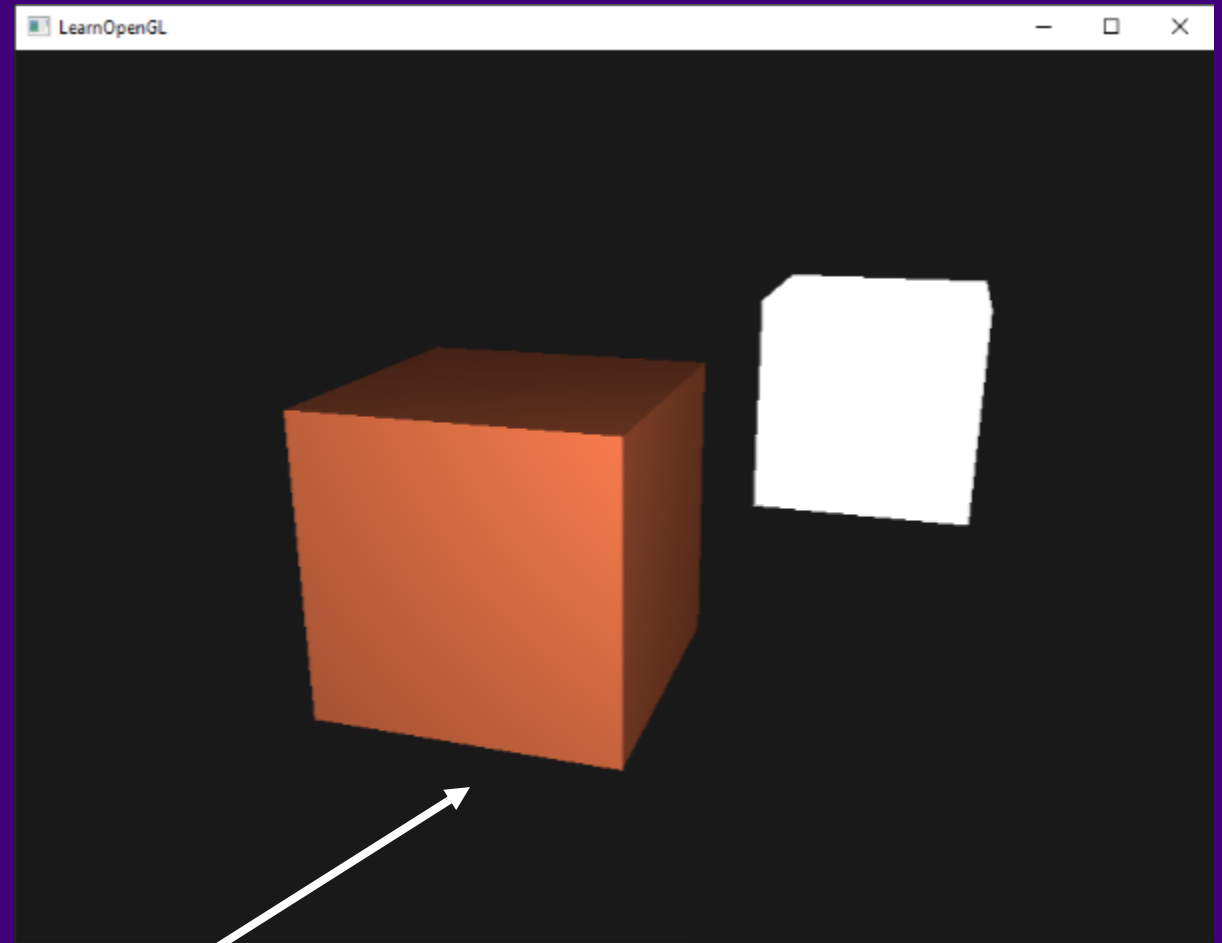
in vec3 Normal;
in vec3 PositionWorld;

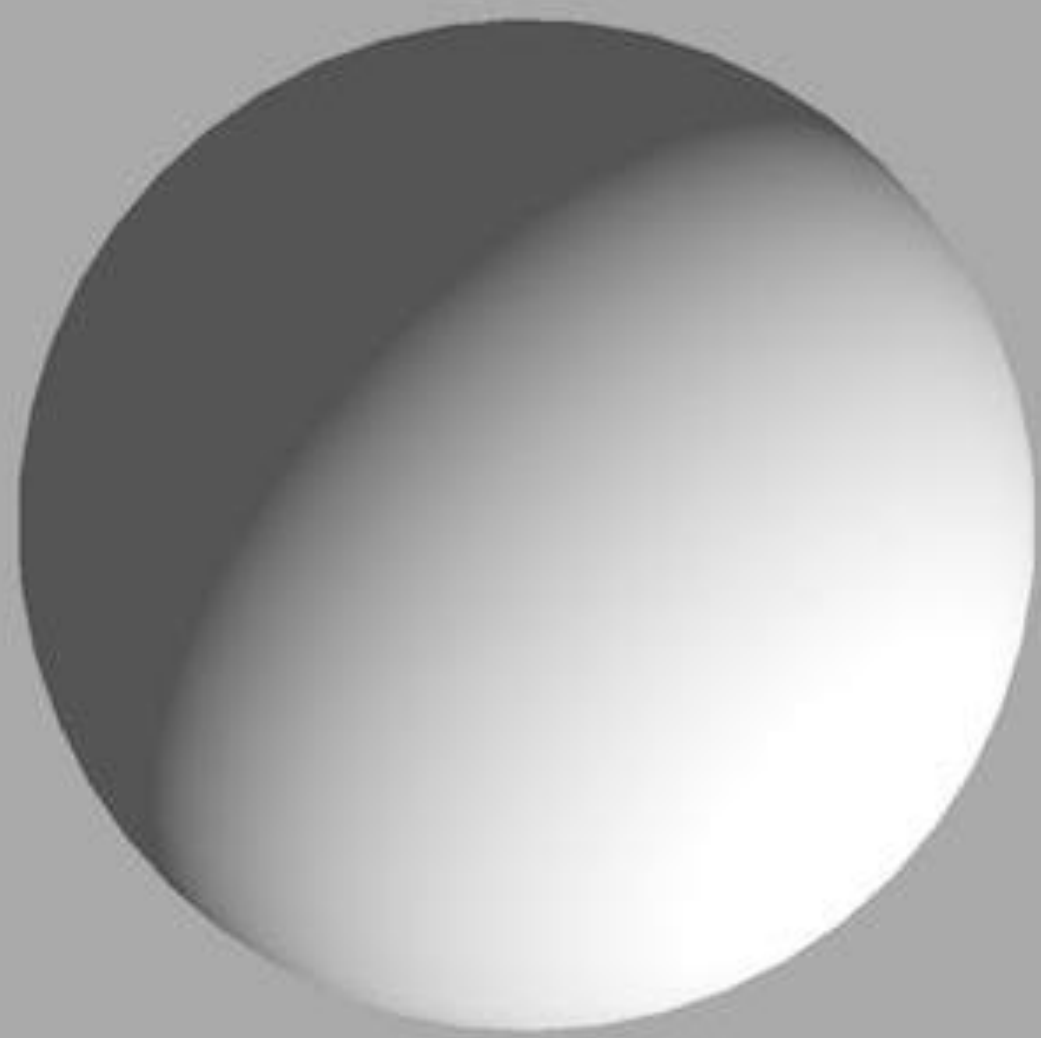
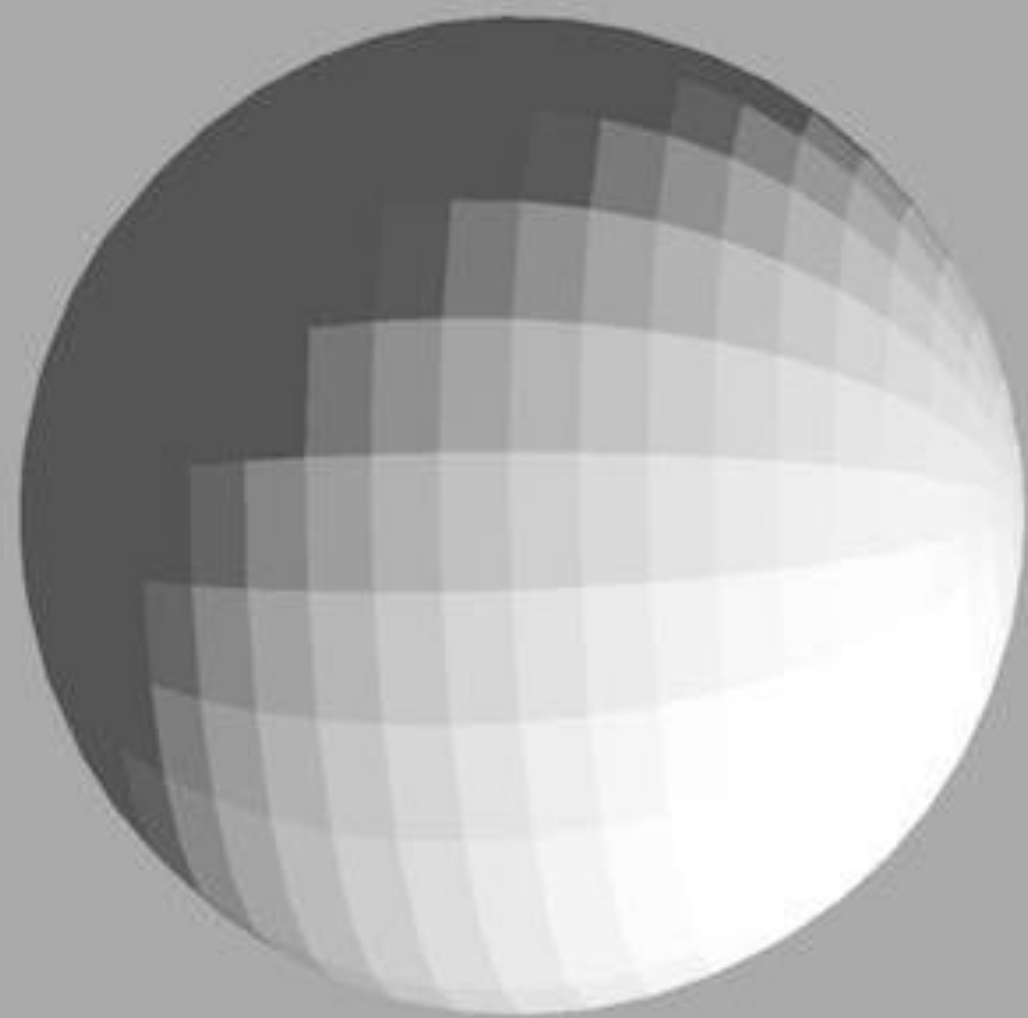
uniform vec3 _LightPos;
uniform vec3 _LightColor;
uniform vec3 _ObjectColor;

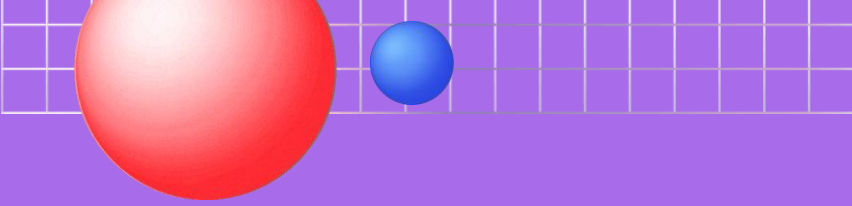
void main()
{
    // ambient
    float ambientStrength = 0.1;
    vec3 ambient = ambientStrength * _LightColor;

    // diffuse
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(_LightPos - PositionWorld);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * _LightColor;

    vec3 result = (ambient + diffuse) * _ObjectColor;
    FragColor = vec4(result, 1.0);
}
```

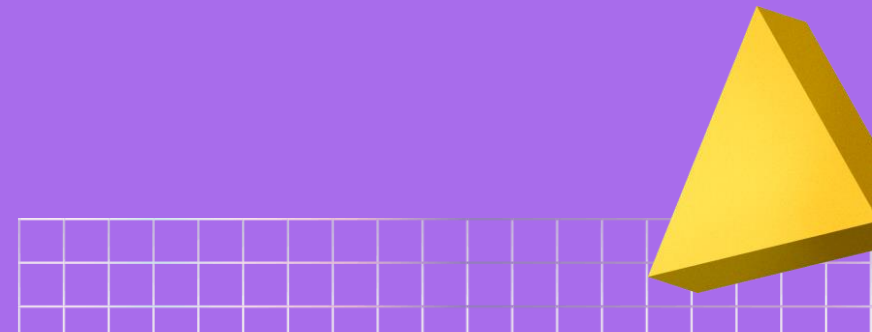


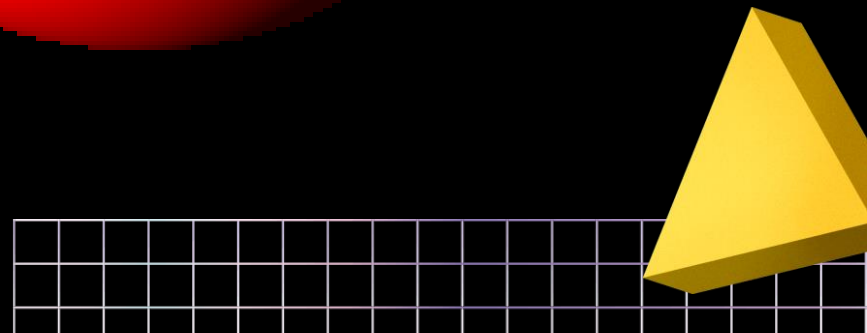
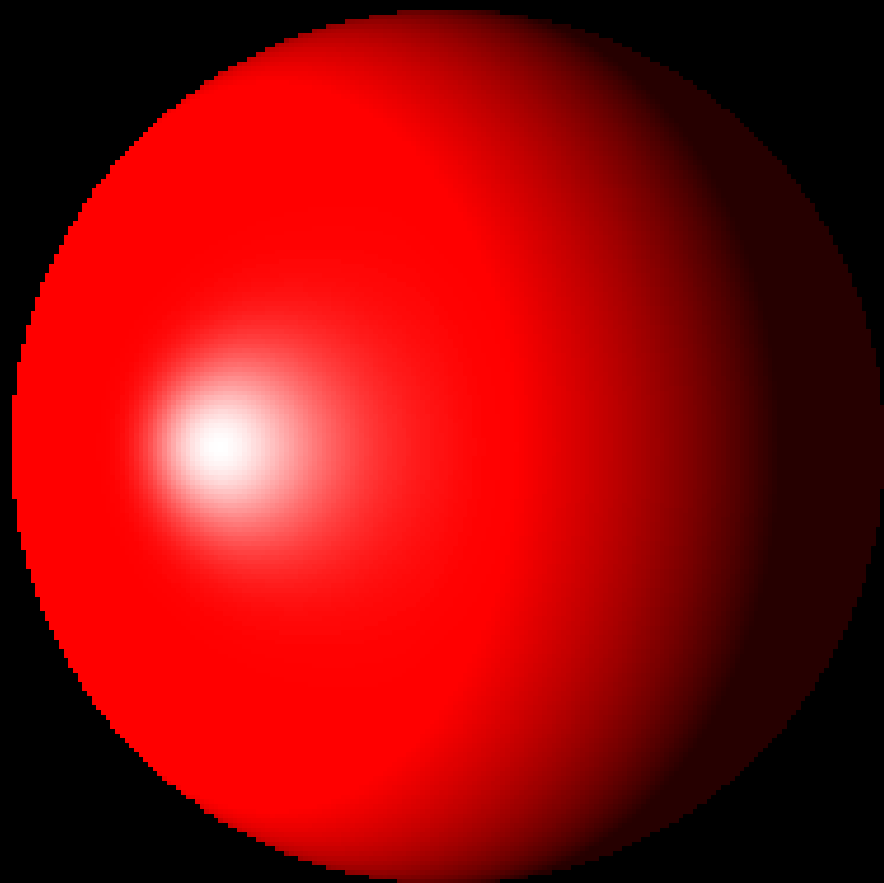
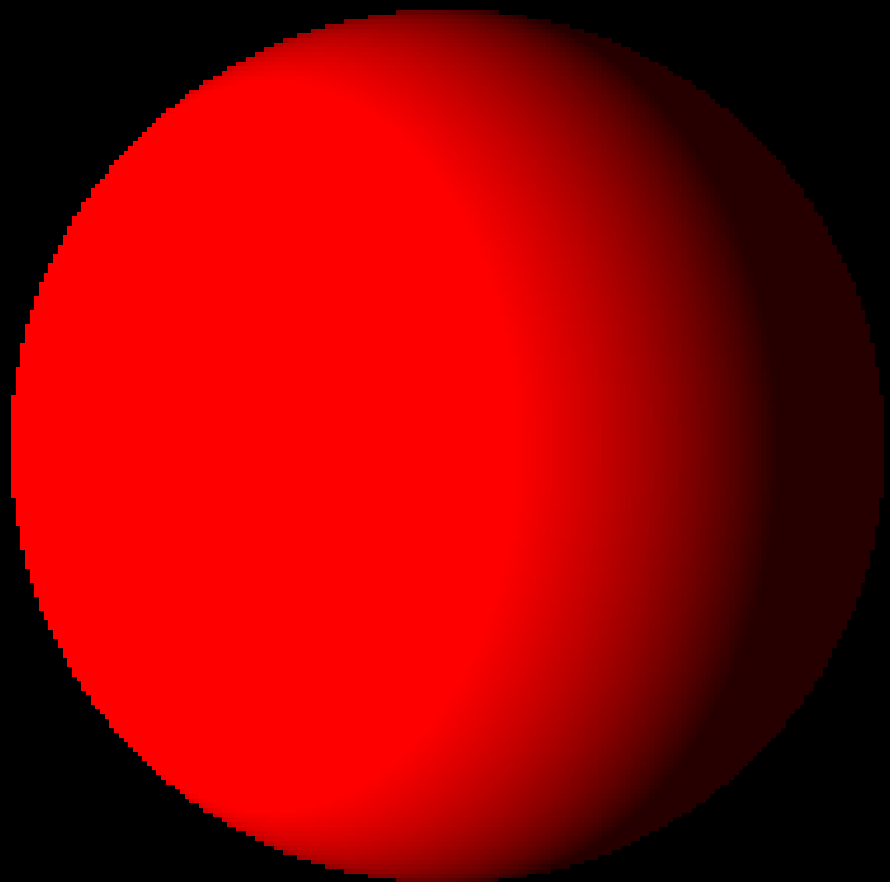
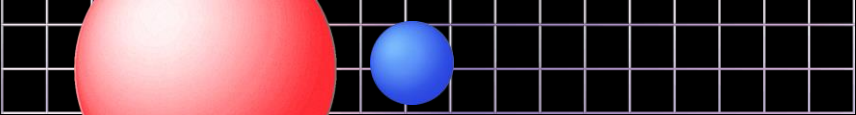


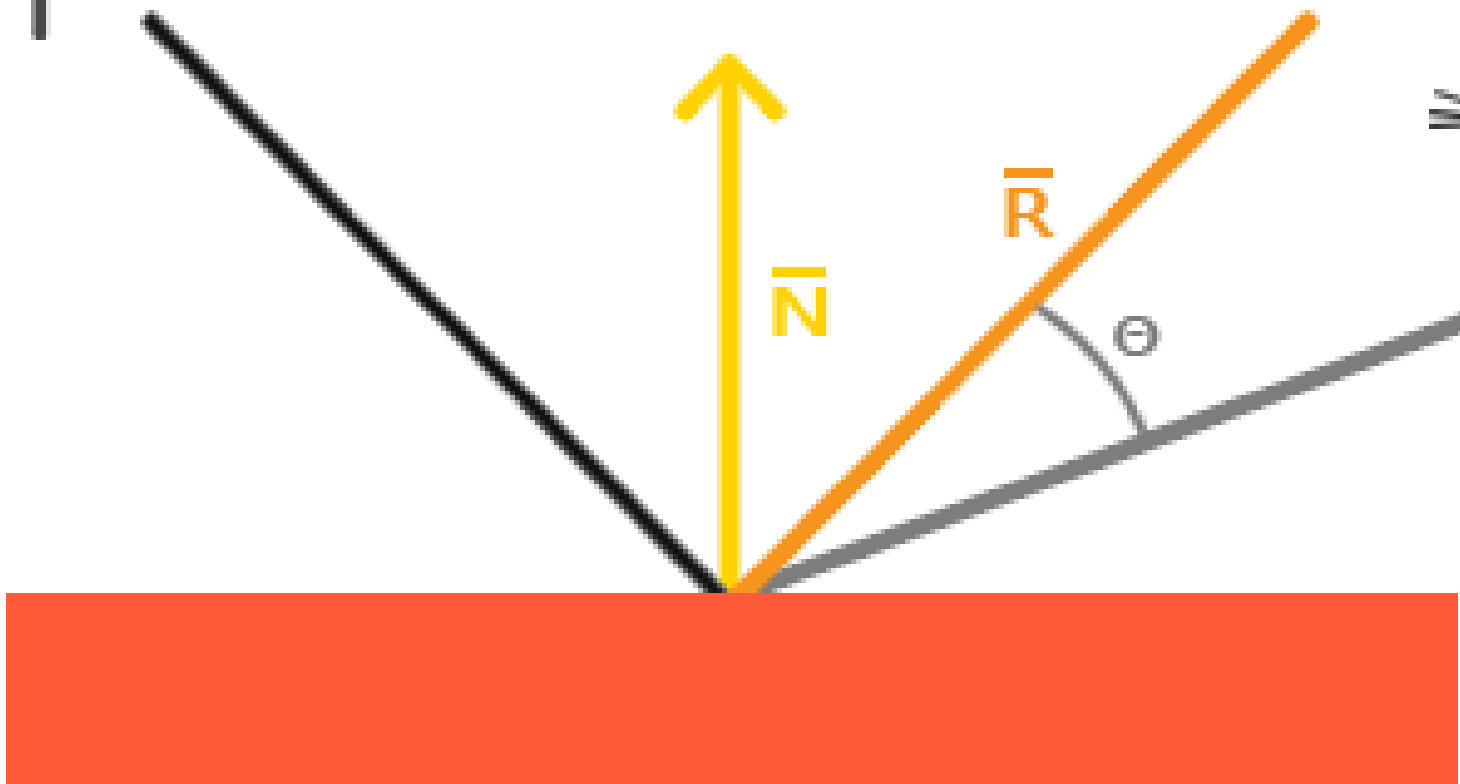
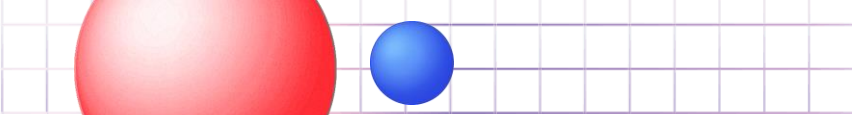


Lumină speculară

- **Simulează zonele luminoase ale obiectelor lucioase.**







Lumină speculară

Fragment Shader

```
#version 400
out vec4 FragColor;

in vec3 Normal;
in vec3 PositionWorld;

uniform vec3 _LightPos;
uniform vec3 _LightColor;
uniform vec3 _ObjectColor;

uniform vec3 _ViewPos;

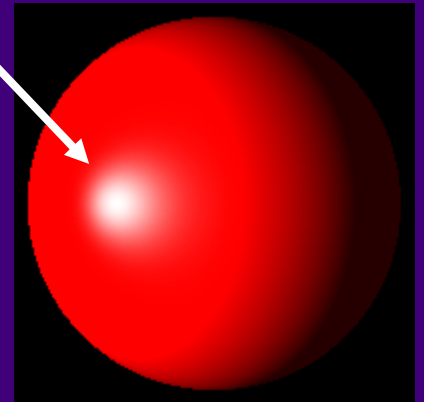
void main()
{
    // ambient
    float ambientStrength = 0.1;
    vec3 ambient = ambientStrength * _LightColor;

    // diffuse
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(_LightPos - PositionWorld);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * _LightColor;
```

```
    // specular
    float specularStrength = 0.5;
    vec3 viewDir = normalize(_ViewPos - PositionWorld);
    vec3 reflectDir = reflect(-lightDir, norm);

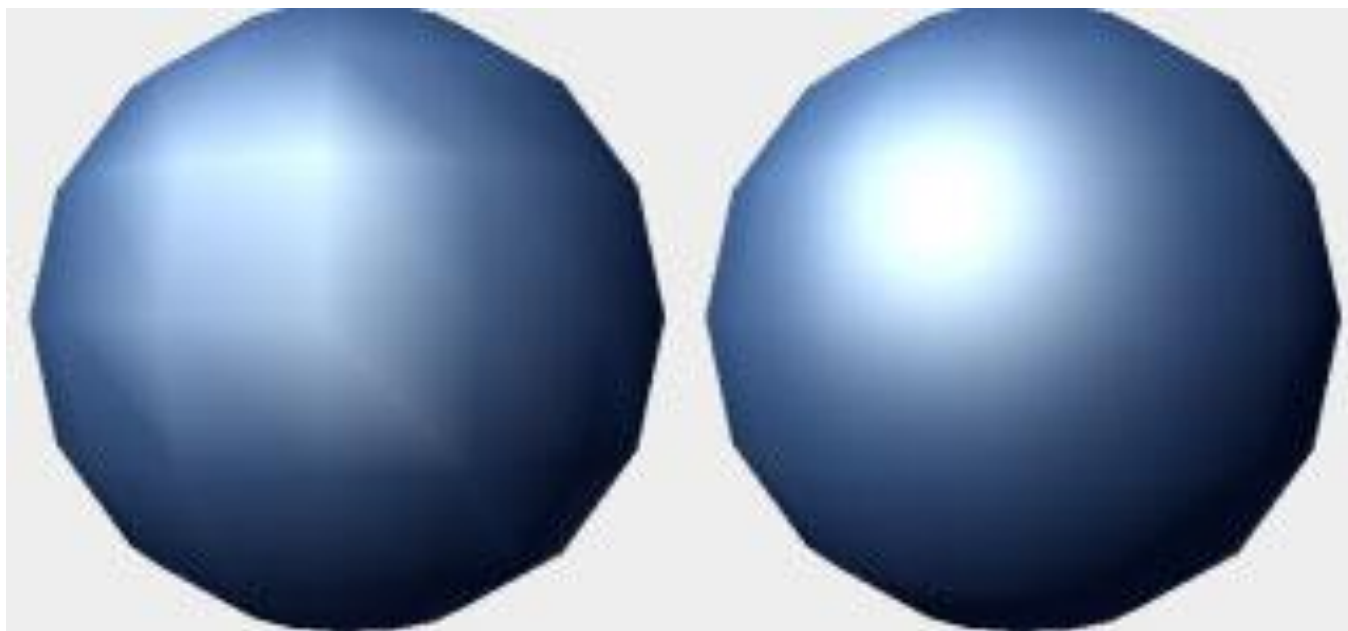
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
    vec3 specular = specularStrength * spec * _LightColor;

    vec3 result = (ambient + diffuse + specular) * _ObjectColor;
    FragColor = vec4(result, 1.0);
}
```



Modelul de iluminare Phong

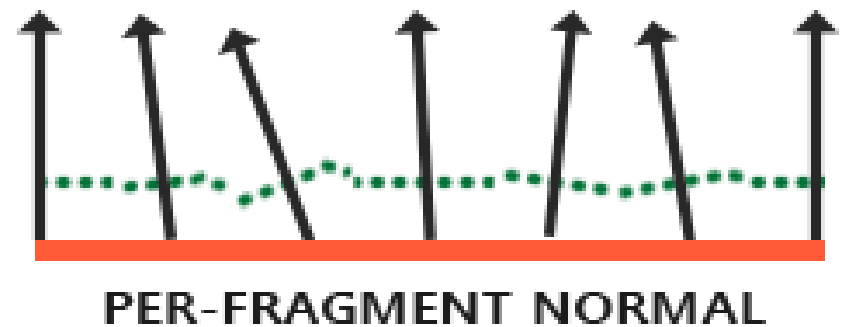
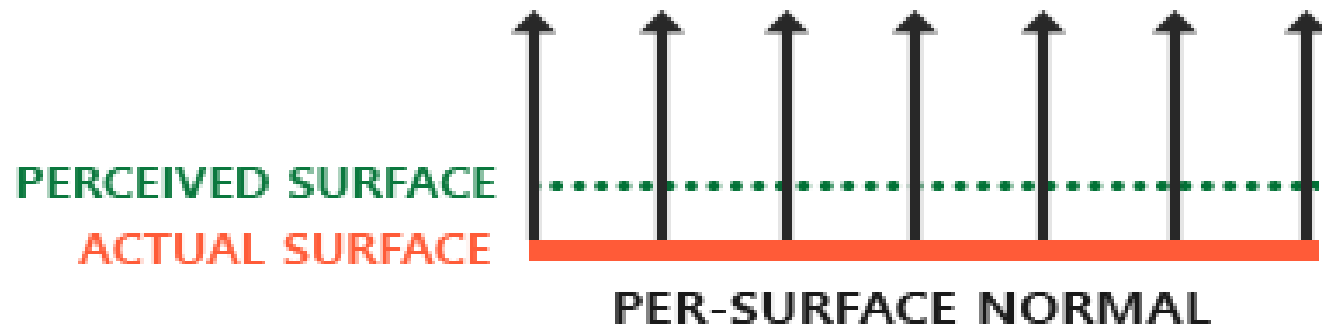


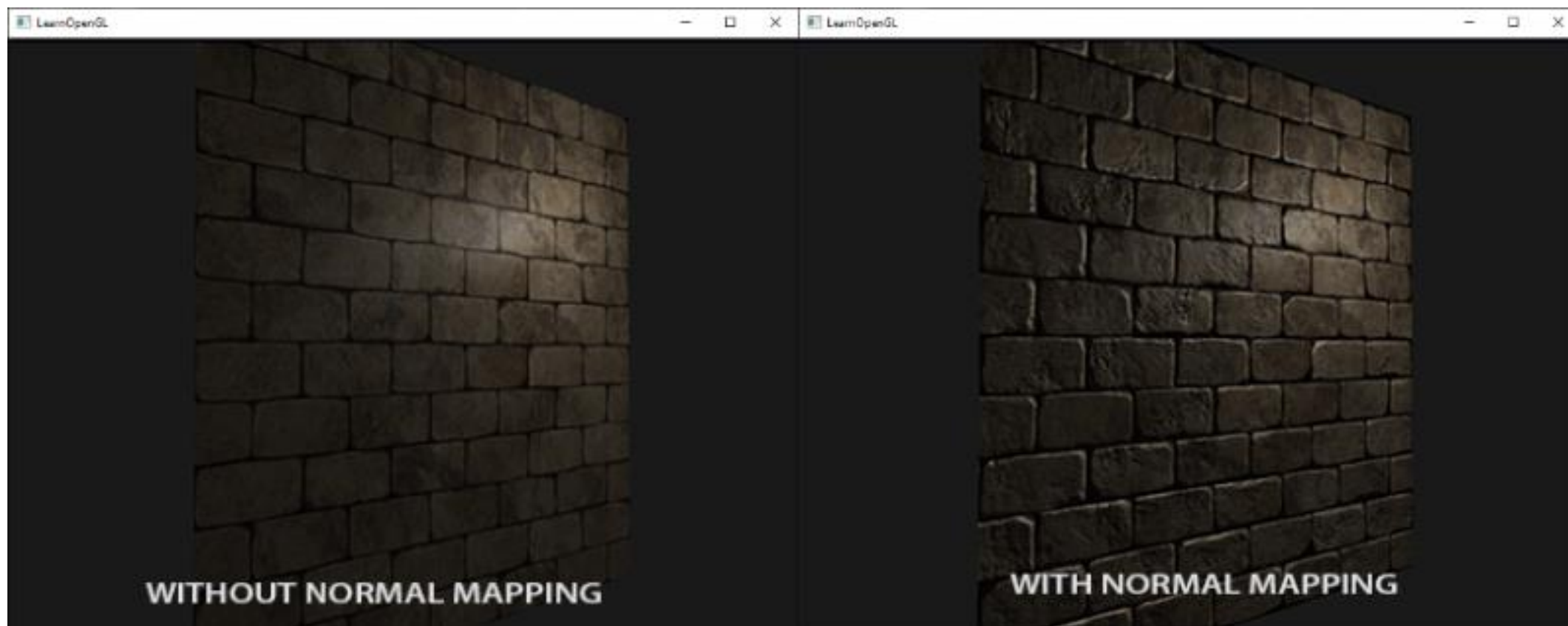


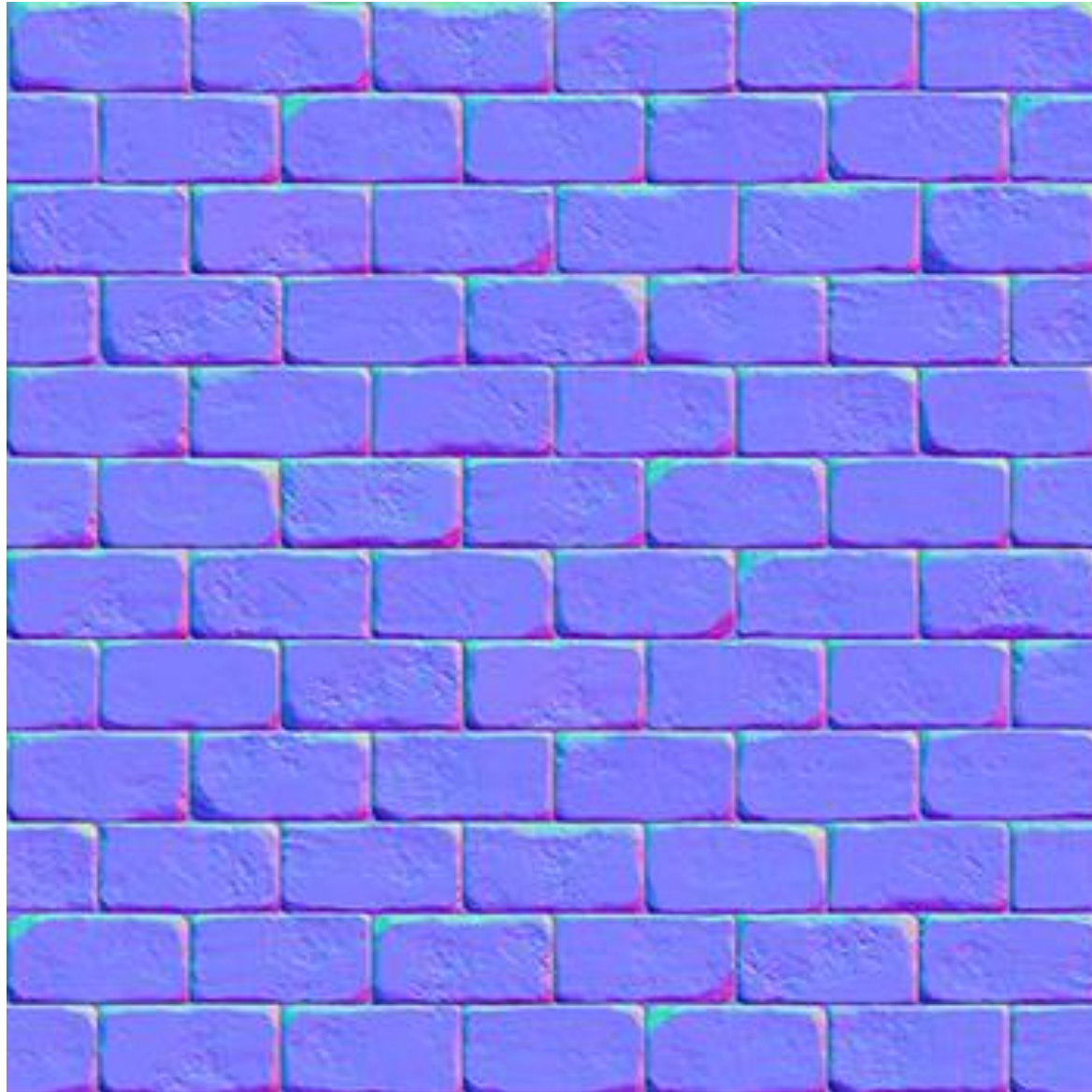
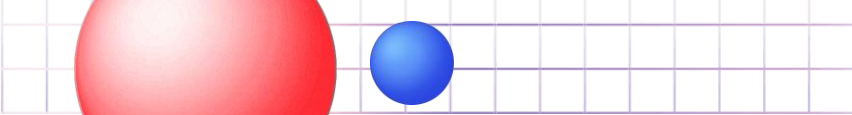
illuminare per vertex

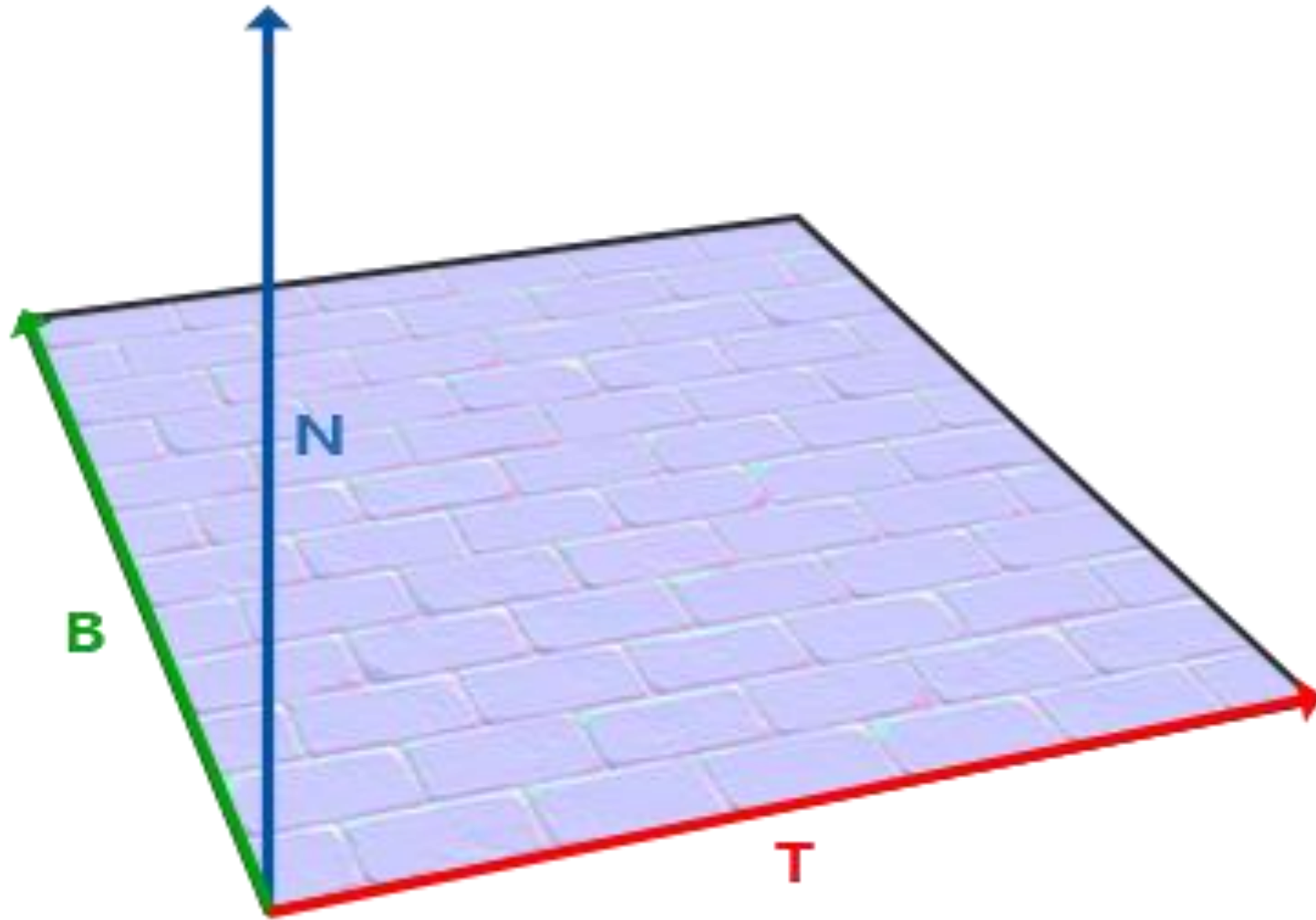
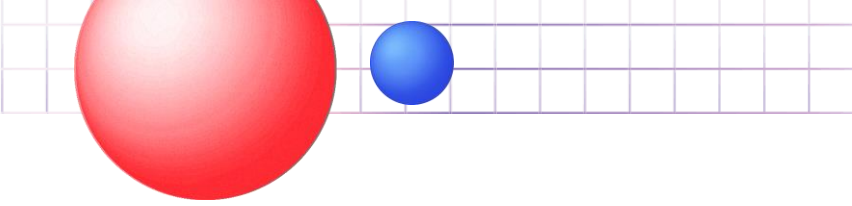
illuminare per fragment

Normal mapping







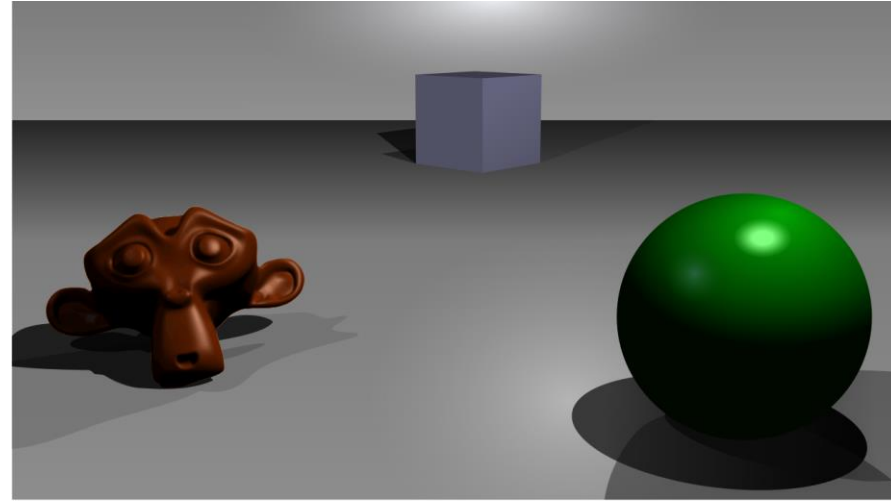




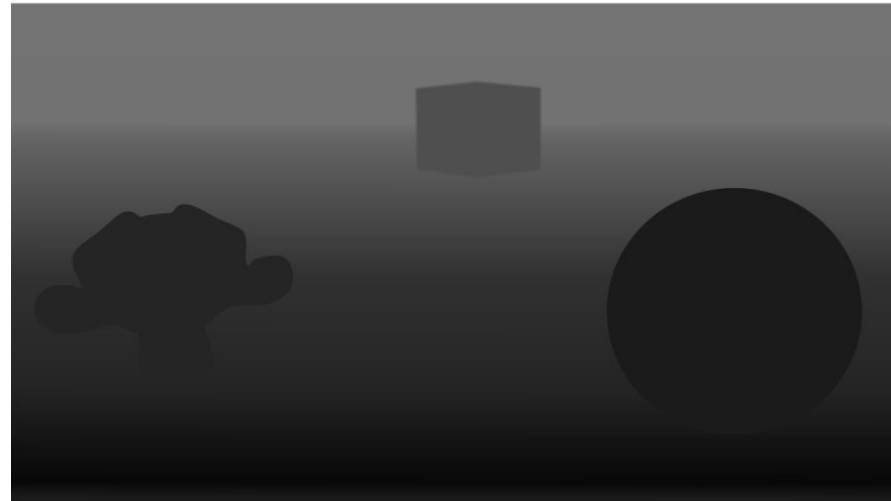
OpenGL Avansat



Depth buffer



A simple three-dimensional scene



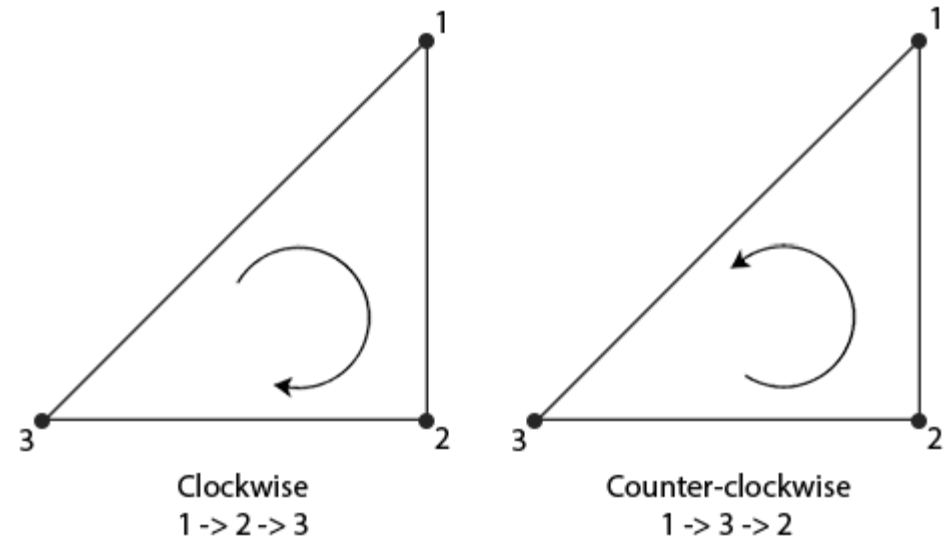
Z-buffer representation



Blending & transparență

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

Face culling



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



Framebuffers

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



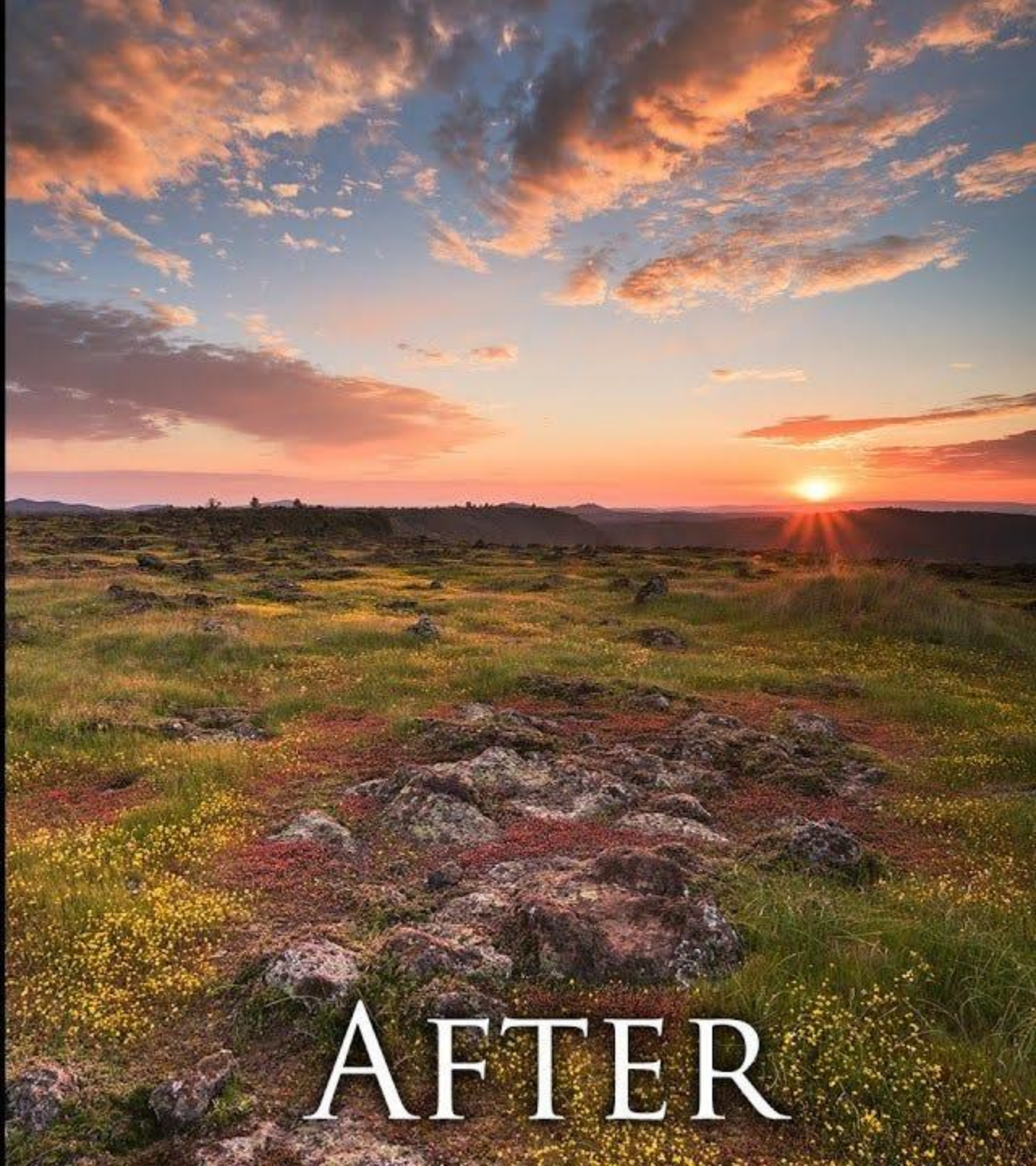
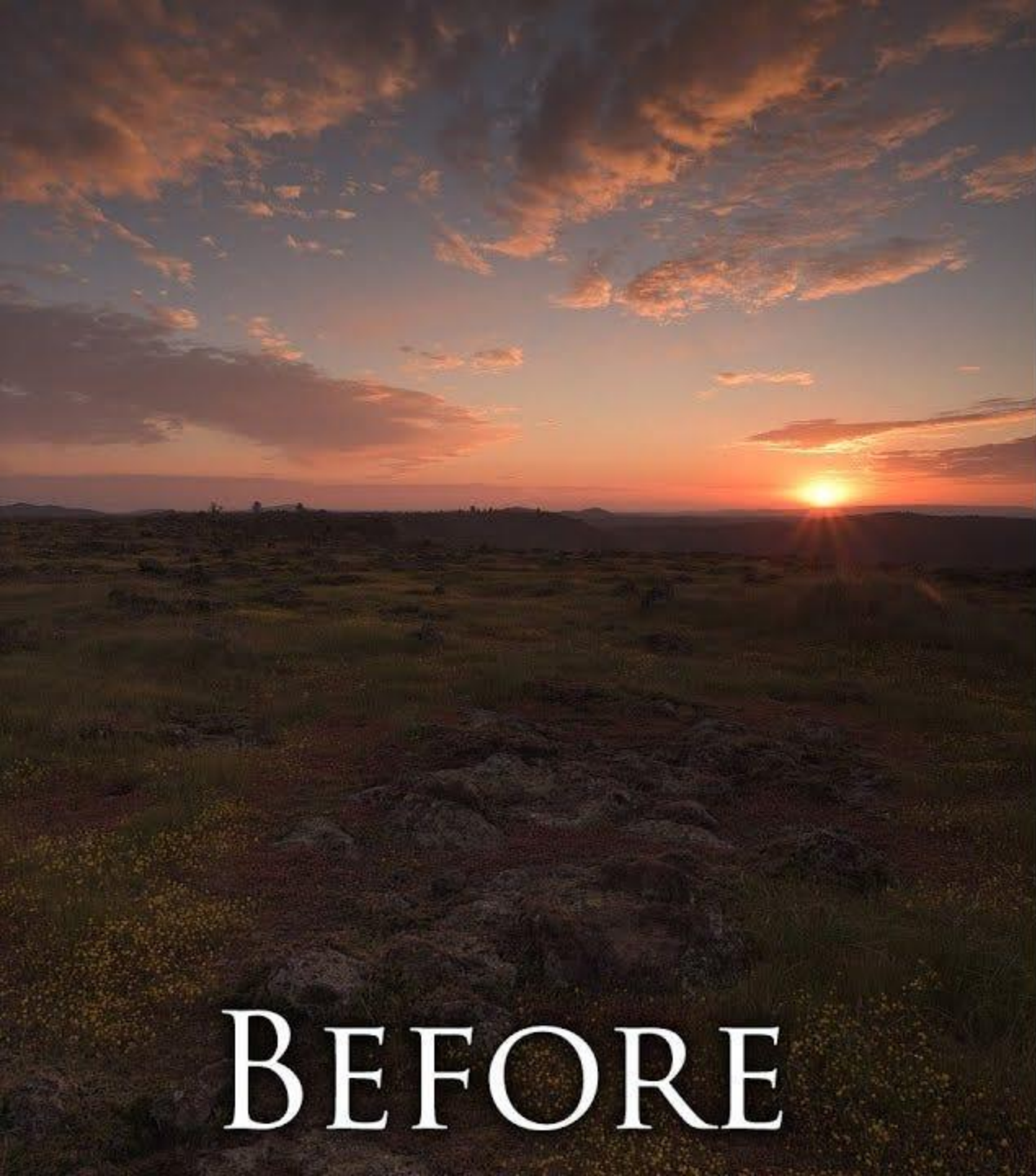
Post Processing

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

<https://github.com/microsoft/DirectXTK/wiki/Writing-custom-shaders>

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

https://youtu.be/aGsUU_bvOgw





Bibliografie/ Resurse

Cursul anterior

Rastertek OpenGL Tutorials

<https://rastertek.com/tutgl4linux.html>

3DGEp Learning DirectX 12 – Lesson 4 - Textures

<https://www.3dgep.com/learning-directx-12-4/>

Learn OpenGL

<https://learnopengl.com/>

