

Nama : Awang Pratama Putra Mulya

NIM : 24060122120039

Laporan Praktikum GKV

Lighting dan Shadow

1. Lighting

What (Apa itu Lighting?)

Pencahayaan dalam grafika komputer dan game development adalah teknik yang digunakan untuk menciptakan nuansa, kedalaman, dan realisme dalam scene. Ini meliputi aspek seperti pencahayaan realistik, pembuatan bayangan, penggunaan global illumination untuk efek yang lebih realistis, HDR lighting untuk rentang dinamis yang lebih luas, ambient occlusion untuk detail tambahan, dan pencahayaan dinamis untuk situasi yang berubah-ubah seperti perubahan waktu atau cuaca dalam game.

Why (Mengapa mengimplementasi/mensimulasikan Lighting?)

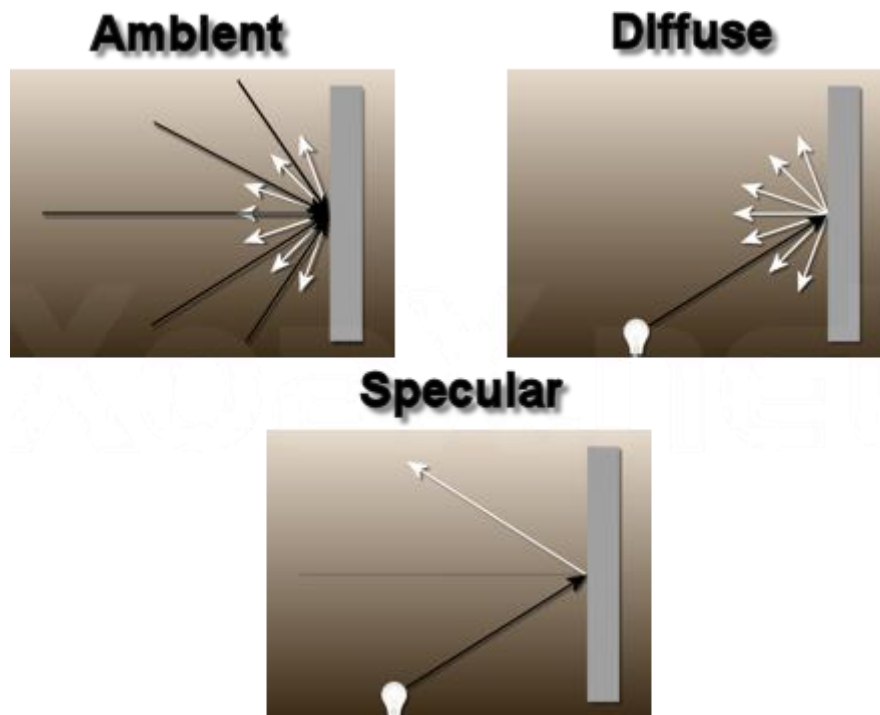
Implementasi pencahayaan dalam grafika komputer penting karena:

1. **Realisme:** Memberikan tampilan yang lebih alami dan realistis pada objek dan lingkungan dalam grafika komputer.
2. **Kedalaman dan Dimensi:** Membantu menentukan kedalaman dan dimensi dalam scene, termasuk pembentukan bayangan dan highlight.
3. **Atmosfer dan Mood:** Mengatur suasana dan mood dalam suatu scene atau permainan dengan menggunakan warna, intensitas, dan arah cahaya.
4. **Memmingkai Fokus:** Mengarahkan perhatian pemain pada objek atau area tertentu dengan memanipulasi pencahayaan untuk menyorot atau menyoroti elemen-elemen penting.
5. **Estetika:** Menambahkan aspek visual yang menarik dan memikat, meningkatkan kualitas keseluruhan dari karya grafis atau permainan.

How (Bagaimana cara mengimplementasikannya?)

Ada beberapa cara untuk mengimplementasikan pencahayaan dalam grafika komputer, antara lain:

1. **Diffuse Lighting:** Pencahayaan difus adalah jenis pencahayaan yang disebabkan oleh cahaya yang tersebar secara merata di semua arah setelah memantul dari permukaan kasar atau tidak rata. Ini adalah jenis pencahayaan yang paling umum ditemui dalam grafika komputer. Untuk mengimplementasikannya, Anda perlu memperhitungkan sudut antara vektor normal permukaan dan arah cahaya yang datang. Intensitas cahaya pada titik tertentu di permukaan akan bergantung pada sudut antara vektor normal dan vektor arah cahaya. Semakin besar sudutnya, semakin kecil intensitas cahayanya.
2. **Specular Lighting:** Pencahayaan spekular adalah jenis pencahayaan yang disebabkan oleh cahaya yang dipantulkan secara teratur dari permukaan yang sangat halus atau mulus. Ini menciptakan efek kilau atau highlight pada permukaan objek. Untuk mengimplementasikannya, Anda perlu memperhitungkan sudut antara arah pandangan kamera (atau mata) dan arah cahaya yang dipantulkan. Efek spekular ini biasanya memiliki intensitas yang tinggi dan terkonsentrasi di sekitar titik di mana sinar cahaya memantul secara langsung ke arah mata.
3. **Ambient Lighting:** Pencahayaan ambien adalah jenis pencahayaan yang merujuk pada cahaya yang tersebar secara merata di sekitar lingkungan, tanpa sumber cahaya yang jelas. Ini adalah cahaya yang dipantulkan atau tersebar dari berbagai permukaan di lingkungan. Dalam implementasinya, Anda biasanya menetapkan nilai ambien yang konstan untuk memberikan efek pencahayaan minimal pada objek, terlepas dari sumber cahaya utama. Pencahayaan ambien digunakan untuk mewakili cahaya yang tersebar di sekitar objek karena pemantulan dan pembiasan cahaya dari berbagai permukaan.



4. **Pencahayaan Terarah (Directional Lighting):** Menggunakan sumber cahaya yang memiliki arah tertentu, seperti sinar matahari, untuk memberikan iluminasi pada objek.
5. **Pencahayaan Titik (Point Lighting):** Mensimulasikan sumber cahaya yang tersebar dari titik tertentu dalam ruang, seperti lampu pijar, yang menghasilkan bayangan dan highlight pada objek.
6. **Pencahayaan Area (Area Lighting):** Menggunakan sumber cahaya yang memiliki area tertentu, seperti layar, untuk menciptakan efek pencahayaan yang lebih halus dan alami.
7. **Pencahayaan Global (Global Illumination):** Memperhitungkan interaksi cahaya yang kompleks antara objek di dalam suatu adegan, termasuk pemantulan dan pembiasan cahaya dari berbagai permukaan.

Implementasi (Diffuse , Specular dan Ambient)

1. Vertex Shader

```
#version 330 core

// Input vertex data, different for all executions of this shader.
layout(location = 0) in vec3 vertexPosition_modelspace;
layout(location = 1) in vec2 vertexUV;
```

```

layout(location = 2) in vec3 vertexNormal_modelspace;

// Output data ; will be interpolated for each fragment.
out vec2 UV;
out vec3 Position_worldspace;
out vec3 Normal_cameraspace;
out vec3 EyeDirection_cameraspace;

// Values that stay constant for the whole mesh.
uniform mat4 MVP;
uniform mat4 V;
uniform mat4 M;

void main(){

    // Output position of the vertex, in clip space : MVP * position
    gl_Position = MVP * vec4(vertexPosition_modelspace,1);

    // Position of the vertex, in worldspace : M * position
    Position_worldspace = (M *
    vec4(vertexPosition_modelspace,1)).xyz;

    // Vector that goes from the vertex to the camera, in camera
    space.
    // In camera space, the camera is at the origin (0,0,0).
    vec3 vertexPosition_cameraspace = ( V * M *
    vec4(vertexPosition_modelspace,1)).xyz;
    EyeDirection_cameraspace = vec3(0,0,0) -
    vertexPosition_cameraspace;

    // Normal of the the vertex, in camera space
    Normal_cameraspace = ( V * M *
    vec4(vertexNormal_modelspace,0)).xyz; // Only correct if
    ModelMatrix does not scale the model ! Use its inverse transpose if
    not.

    // UV of the vertex. No special space for this one.

```

```
    UV = vertexUV;
}
```

2. Fragment Shader

```
#version 330 core

// Interpolated values from the vertex shaders
in vec2 UV;
in vec3 Position_worldspace;
in vec3 Normal_cameraspace;
in vec3 EyeDirection_cameraspace;
in vec3 LightDirection_cameraspace;

// Output data
out vec3 color;

// Values that stay constant for the whole mesh.
uniform sampler2D textureSampler;
uniform mat4 MV;
uniform vec3 LightPosition_worldspace;

void main(){

    // Light emission properties
    // You probably want to put them as uniforms
    vec3 LightColor = vec3(1,1,1);
    float LightPower = 100.0f;

    // Material properties
    vec3 MaterialDiffuseColor = texture( textureSampler, UV ).rgb;
    vec3 MaterialAmbientColor = vec3(0.1,0.1,0.1) *
MaterialDiffuseColor;
    // vec3 MaterialAmbientColor = MaterialDiffuseColor;
    vec3 MaterialSpecularColor = vec3(0.3,0.3,0.3);

    // Distance to the light
```

```

        float distance = length( LightPosition_worldspace -
Position_worldspace );

    // Normal of the computed fragment, in camera space
    vec3 n = normalize( Normal_cameraspace );

    // Direction of the light (from the fragment to the light)
    vec3 l = normalize( LightDirection_cameraspace );

    // Cosine of the angle between the normal and the light
direction,
    // clamped above 0
    // - light is at the vertical of the triangle -> 1
    // - light is perpendicular to the triangle -> 0
    // - light is behind the triangle -> 0
    float cosTheta = clamp( dot( n,l ), 0,1 );

    // Eye vector (towards the camera)
    vec3 E = normalize(EyeDirection_cameraspace);

    // Direction in which the triangle reflects the light
    vec3 R = reflect(-l,n);

    // Cosine of the angle between the Eye vector and the Reflect
vector,
    // clamped to 0
    // - Looking into the reflection -> 1
    // - Looking elsewhere -> < 1
    float cosAlpha = clamp( dot( E,R ), 0,1 );

    color =
        // Ambient : simulates indirect lighting
        MaterialAmbientColor +
        // Diffuse : "color" of the object
        MaterialDiffuseColor * LightColor * LightPower * cosTheta /
(distance*distance) + //; // +
        // Specular : reflective highlight, like a mirror

```

```
MaterialSpecularColor * LightColor * LightPower *  
pow(cosAlpha,5) / (distance*distance);  
  
}
```

2. Shadow

What (Apa itu Shadow?)

Bayangan (shadow) dalam grafika komputer adalah efek visual yang dihasilkan ketika cahaya terhalang oleh suatu objek, menciptakan area gelap di belakang objek tersebut. Dalam dunia nyata, bayangan memberikan informasi tentang kedalaman dan posisi relatif objek dalam ruang. Dalam konteks grafika komputer, pembuatan bayangan membutuhkan pemodelan interaksi kompleks antara cahaya, objek, dan pencahayaan untuk menciptakan ilusi realisme.

Why (Mengapa Menerapkan Bayangan?)

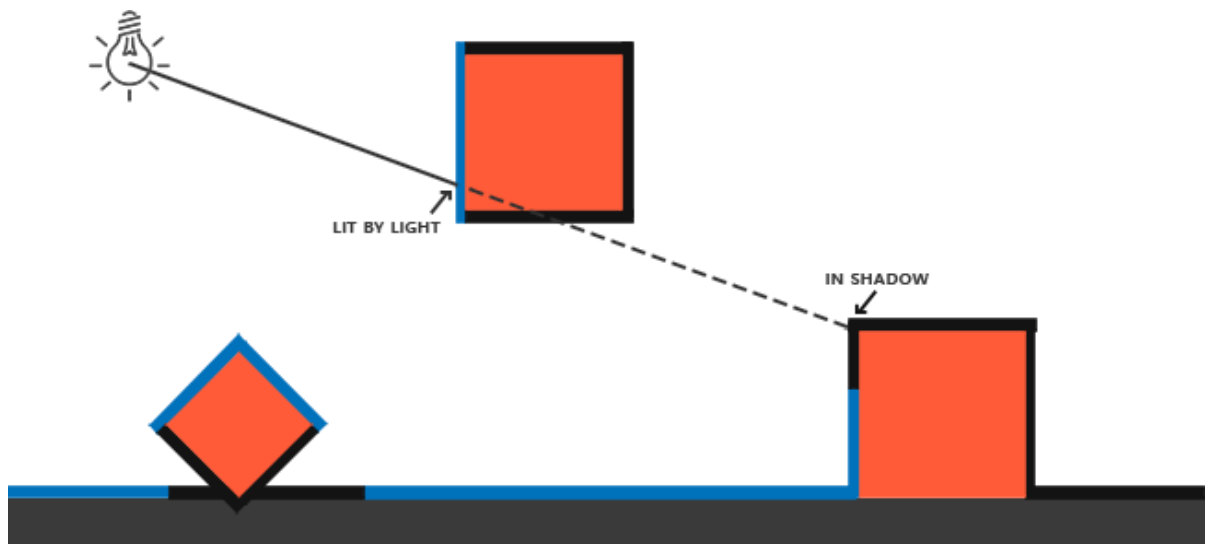
Implementasi bayangan penting karena:

1. **Realisme:** Bayangan meningkatkan realisme visual dalam simulasi grafis dengan menciptakan ilusi kedalaman dan dimensi dalam sebuah adegan.
2. **Kedalaman:** Bayangan membantu memperjelas posisi relatif objek dalam ruang, membantu pemirsa memahami struktur dan jarak antar objek.
3. **Estetika:** Efek bayangan dapat digunakan untuk menciptakan komposisi visual yang menarik dan menambahkan nuansa dramatis atau artistik pada karya grafis.
4. **Informasi Spatial:** Bayangan memberikan informasi penting tentang bentuk dan posisi objek, memungkinkan pengguna untuk membuat keputusan yang lebih tepat dalam aplikasi seperti simulasi, permainan, dan desain.

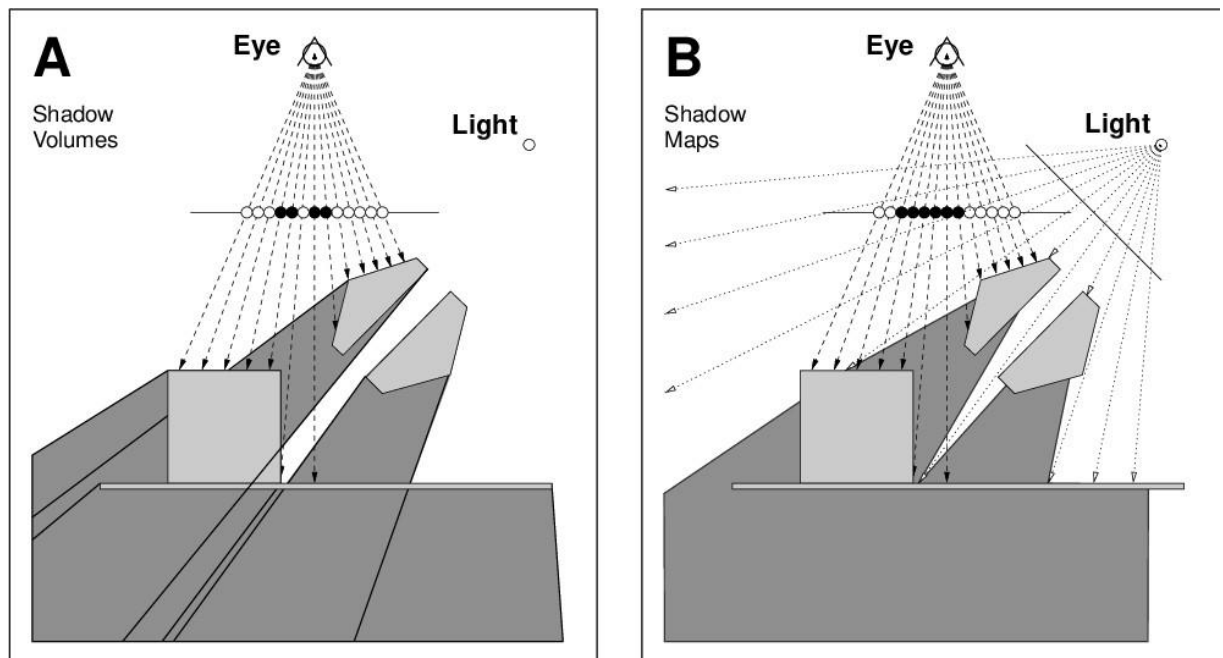
How (Bagaimana Implementasi Bayangan?)

Terdapat beberapa teknik untuk mengimplementasikan bayangan dalam grafika komputer, di antaranya:

1. **Shadow Mapping:** Teknik yang paling umum digunakan untuk membuat bayangan dalam permainan video dan aplikasi 3D. Ini melibatkan membuat peta kedalaman (depth map) dari perspektif sumber cahaya, dan kemudian membandingkan posisi titik pandang pengamat dengan peta kedalaman untuk menentukan apakah suatu titik terkena bayangan atau tidak.



2. **Shadow Volumes:** Teknik ini menggunakan informasi geometris untuk menciptakan volume bayangan di sekitar objek yang memancarkan bayangan. Ini melibatkan pemotongan objek yang menerima bayangan dengan volume bayangan yang dihasilkan oleh sumber cahaya, menciptakan ilusi bayangan di permukaan objek tersebut.



3. **Ray Tracing:** Metode ini menghasilkan bayangan dengan melacak jalur sinar cahaya dari sumber cahaya ke objek, dan kemudian menentukan apakah jalur sinar tersebut terhalangi oleh objek lain di sepanjang jalurnya. Meskipun ray tracing menghasilkan hasil yang sangat realistis, ini seringkali membutuhkan waktu komputasi yang lebih lama daripada teknik lainnya.

Implementasi Shadow

1. Vertex Shader

```
#version 330 core

// Input vertex data, different for all executions of this shader.
layout(location = 0) in vec3 vertexPosition_modelspace;
layout(location = 1) in vec2 vertexUV;
layout(location = 2) in vec3 vertexNormal_modelspace;

out vec2 UV;
out vec3 Position_worldspace;
out vec3 Normal_cameraspace;
out vec3 EyeDirection_cameraspace;
out vec3 LightDirection_cameraspace;
out vec4 Position_lightspace;

uniform mat4 MVP;
uniform mat4 V;
uniform mat4 M;
uniform vec3 LightPosition_worldspace;
```

```

uniform mat4 lightSpaceMatrix;

void main()
{
    // Output position of the vertex, in clip space : MVP * position
    gl_Position = MVP * vec4(vertexPosition_modelspace, 1);

    // Position of the vertex, in worldspace : M * position
    Position_worldspace = (M * vec4(vertexPosition_modelspace,1)).xyz;

    Position_lightspace = lightSpaceMatrix * vec4(Position_worldspace,
1.);
    // Vector that goes from the vertex to the camera, in camera space.
    // In camera space, the camera is at the origin (0,0,0).
    vec3 vertexPosition_cameraspace = ( V * M *
vec4(vertexPosition_modelspace,1)).xyz;
    EyeDirection_cameraspace = vec3(0,0,0) - vertexPosition_cameraspace;

    // Vector that goes from the vertex to the light, in camera space. M
is ommited because it's identity.
    vec3 LightPosition_cameraspace = ( V *
vec4(LightPosition_worldspace,1)).xyz;
    LightDirection_cameraspace = LightPosition_cameraspace +
EyeDirection_cameraspace;

    // Normal of the the vertex, in camera space
    Normal_cameraspace = ( V * M * vec4(vertexNormal_modelspace,0)).xyz;
    // Only correct if ModelMatrix does not scale the model ! Use its
inverse transpose if not.

    // UV of the vertex. No special space for this one.
    UV = vertexUV;
    // vs_out.FragPos = vec3(M * vec4(vertexPosition_modelspace, 1.0));
    // vs_out.Normal = transpose(inverse(mat3(M))) *
vertexNormal_modelspace;
    // UV = vertexUV;
    // vs_out.FragPosLightSpace = lightSpaceMatrix *
vec4(vs_out.FragPos, 1.0);
    // gl_Position = MVP * vec4(vertexPosition_modelspace, 1.0);
}

```

2. Fragment Shader

```

#version 330 core
out vec4 FragColor;

in vec2 UV;
in vec3 Position_worldspace;
in vec3 Normal_cameraspace;
in vec3 EyeDirection_cameraspace;
in vec3 LightDirection_cameraspace;
in vec4 Position_lightspace;

```

```

uniform sampler2D textureSampler;
uniform sampler2D shadowMap;

uniform vec3 LightPosition_worldspace;
uniform vec3 CameraPosition_worldspace;

float ShadowCalculation(vec4 fragPosLightSpace)
{
    // perform perspective divide
    vec3 projCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    // transform to [0,1] range
    projCoords = projCoords * 0.5 + 0.5;
    // get closest depth value from light's perspective (using [0,1] range
    fragPosLight as coords)
    float closestDepth = texture(shadowMap, projCoords.xy).r;
    // get depth of current fragment from light's perspective
    float currentDepth = projCoords.z;
    // calculate bias (based on depth map resolution and slope)
    vec3 normal = normalize(Normal_cameraspace);
    vec3 lightDir = normalize(LightPosition_worldspace -
    Position_worldspace);
    float bias = max(0.05 * (1.0 - dot(normal, lightDir)), 0.005);
    // check whether current frag pos is in shadow
    // float shadow = currentDepth - bias > closestDepth ? 1.0 : 0.0;
    // PCF
    float shadow = 0.0;
    vec2 texelSize = 1.0 / textureSize(shadowMap, 0);
    for(int x = -1; x <= 1; ++x)
    {
        for(int y = -1; y <= 1; ++y)
        {
            float pcfDepth = texture(shadowMap, projCoords.xy + vec2(x, y)
            * texelSize).r;
            shadow += currentDepth - bias > pcfDepth ? 1.0 : 0.0;
        }
    }
    shadow /= 9.0;

    // keep the shadow at 0.0 when outside the far_plane region of the
    light's frustum.
    if(projCoords.z > 1.0)
        shadow = 0.0;

    return shadow;
}

void main()
{
    vec3 LightColor = vec3(1,1,1);
    float LightPower = 100.0f;

    // Material properties
    vec3 MaterialDiffuseColor = texture( textureSampler, UV ).rgb;
    vec3 MaterialAmbientColor = vec3(0.1) * MaterialDiffuseColor;
    vec3 MaterialSpecularColor = vec3(0.3);

    // Distance to the light
    float distance = length( LightPosition_worldspace -
    Position_worldspace );

```

```

// Normal of the computed fragment, in camera space
vec3 n = normalize( Normal_cameraspace );

// Direction of the light (from the fragment to the light)
vec3 l = normalize( LightDirection_cameraspace );
// Cosine of the angle between the normal and the light direction,
// clamped above 0
// - light is at the vertical of the triangle -> 1
// - light is perpendicular to the triangle -> 0
// - light is behind the triangle -> 0
float cosTheta = clamp( dot( n,l ), 0,1 );

// Eye vector (towards the camera)
vec3 E = normalize(EyeDirection_cameraspace);

// Direction in which the triangle reflects the light
vec3 R = reflect(-l,n);

// Cosine of the angle between the Eye vector and the Reflect vector,
// clamped to 0
// - Looking into the reflection -> 1
// - Looking elsewhere -> < 1
float cosAlpha = clamp( dot( E,R ), 0,1 );

// vec3 color = texture(textureSampler, fs_in.TexCoords).rgb;
// vec3 normal = normalize(fs_in.Normal);

// ambient
// vec3 ambient = 0.2 * LightColor;
// // diffuse
// vec3 lightDir = normalize(LightPosition_worldspace - fs_in.FragPos);
// float diff = max(dot(lightDir, normal), 0.0);
// vec3 diffuse = diff * LightColor * LightPower;
// // specular
// vec3 viewDir = normalize(CameraPosition_worldspace - fs_in.FragPos);
// vec3 reflectDir = reflect(-lightDir, normal);
// float spec = 0.0;
// vec3 halfwayDir = normalize(lightDir + CameraPosition_worldspace);
// spec = pow(max(dot(normal, halfwayDir), 0.0), 64.0);
// vec3 specular = spec * LightColor * LightPower;
// calculate shadow
float shadow = ShadowCalculation(Position_lightspace);

vec3 ambient = MaterialAmbientColor;
vec3 diffuse = MaterialDiffuseColor * LightColor * LightPower *
cosTheta / (distance*distance);
vec3 specular = MaterialSpecularColor * LightColor * LightPower *
pow(cosAlpha,5) / (distance*distance);

vec3 lighting = (
    ambient +
    // (1.0 - shadow) *
    (diffuse + specular)
) * MaterialDiffuseColor;
// vec3 lighting = (ambient + (1.0 - shadow) * (diffuse + specular)) *
MaterialDiffuseColor;

FragColor = vec4(lighting, 1.0);

```

}

