TUGAS PRAKTIKUM PERTEMUAN 4 DAN 5

GRAFIKA KOMPUTASI VISUAL

LIGHTING DAN SHADOW

Disusun untuk Memenuhi Tugas Individu pada

Mata Kuliah Grafika dan Komputasi Visual

Disusun Oleh:

Tera Makna Pratiwi (24060122140102)

DEPARTEMEN ILMU KOMPUTER/INFORMATIKA

FAKULTAS SAINS DAN MATEMATIKA

UNIVERSITAS DIPONEGORO

2024

LIGHTING

1. Pengertian Lighting

   Lighting atau pencahayaan dalam grafika dan komputasi visual adalah proses mensimulasikan cahaya di lingkungan digital atau tiga dimensi (3D). Proses ini melibatkan pengaturan sumber cahaya, intensitas, warna, arah, dan posisi cahaya untuk menciptakan efek visual yang diinginkan. Lighting adalah aspek penting yang mempengaruhi penampilan keseluruhan adegan, termasuk warna, tekstur, bayangan, refleksi, dan suasana yang ingin disampaikan. Lighting membantu memberikan kedalaman dan realisme pada objek serta menciptakan nuansa dan emosi yang sesuai dengan kebutuhan desain.

2. Alasan Mengimplementasikan/Mensimulasikan Lighting

   Pencahayaan di dunia nyata sangatlah rumit dan bergantung pada banyak faktor, sesuatu yang tidak dapat kita perhitungkan berdasarkan keterbatasan kekuatan pemrosesan yang kita miliki. Oleh karena itu, pencahayaan di OpenGL didasarkan pada perkiraan realitas menggunakan model sederhana yang lebih mudah diproses dan terlihat relatif serupa. Selain itu terdapat beberapa hal yang menjadi alasan lain seperti:

   - Realistis dan Imersi
     Lighting yang tepat menciptakan tampilan visual yang realistis, mendekati apa yang kita lihat di dunia nyata, sehingga meningkatkan rasa imersi dalam pengalaman visual.
   - Peningkatan Estetika Visual
     Pencahayaan yang tepat dapat meningkatkan estetika visual dengan menciptakan kontras dan mengarahkan perhatian pada elemen penting dalam adegan.
   - Penciptaan Suasana dan Emosi
     Cahaya dapat digunakan untuk mengatur suasana hati atau emosi tertentu dalam adegan, seperti romantis, dramatis, atau menegangkan.
   - Menghadirkan Kedalaman dan Dimensi
     Lighting membantu memberikan kedalaman dan dimensi pada adegan melalui bayangan dan refleksi, membuat objek terlihat lebih hidup.
   - Menekankan Detail dan Tekstur
     Cahaya dapat menyoroti detail dan tekstur pada objek, membantu dalam mengidentifikasi bahan dan permukaan.

3. Cara Implementasi

   1) Vertex Shader

      Kita akan melakukan kalkulasi cahaya lebih banyak di fragment shader. Vertex shader akan digunakan sebagai interpolator dari world space ke camera space. Di dalam vertex shader kita akan menghitung tiap vertex untuk

      1. LightPosition_cameraspace: Arah cahaya ke vertex (untuk sudut insiden)
      2. Normal_cameraspace: Normal di camera space
      3. EyeDirection_cameraspace: Arah dari vertex ke camera (untuk menghitung specular reflection)

```
#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;
out vec3 LightDirection_cameraspace;

// Values that stay constant for the whole mesh.
uniform mat4 MVP;
uniform mat4 V;
uniform mat4 M;
uniform vec3 LightPosition_worldspace;

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;

	// 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;
}
```

Penjelasan code:

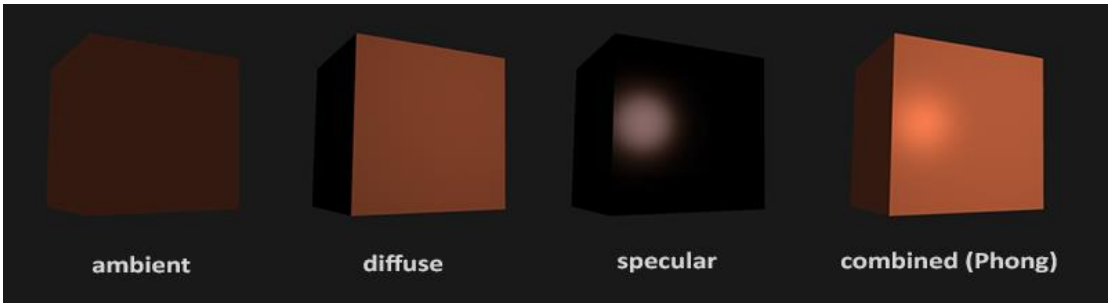
- vertexPosition_modelspace: Posisi vertex dalam model space.
- vertexUV: Koordinat UV dari vertex.
- vertexNormal_modelspace: Normal dari vertex dalam model space.
- UV: Koordinat UV yang diinterpolasi untuk setiap fragmen.
- Position_worldspace: Posisi vertex dalam world space.
- Normal_cameraspace: Normal vertex dalam camera space.
- EyeDirection_cameraspace: Arah dari vertex ke kamera dalam camera space.
- LightDirection_cameraspace: Arah dari vertex ke sumber cahaya dalam camera space.
- MVP: Matriks transformasi Model-View-Projection.
- V: Matriks pandangan (view) yang mentransformasikan world space ke camera space.
- M: Matriks model yang mentransformasikan model space ke world space.
- LightPosition_worldspace: Posisi sumber cahaya dalam world space.
- gl_Position = MVP * vec4(vertexPosition_modelspace,1): Menghitung posisi vertex dalam clip space dengan mengalikan posisi vertex dalam model space dengan matriks MVP. vec4(vertexPosition_modelspace,1) adalah posisi vertex dalam model space sebagai vektor 4D (dengan komponen ke-4 diset ke 1).
- Position_worldspace = (M * vec4(vertexPosition_modelspace,1)).xyz: Menghitung posisi vertex dalam world space dengan mengalikan posisi vertex dalam model space dengan matriks model. Hasilnya kemudian diubah menjadi vektor 3D dengan mengambil komponen xyz.
- vec3 vertexPosition_cameraspace = (V * M * vec4(vertexPosition_modelspace,1)).xyz: Menghitung posisi vertex dalam camera space dengan mengalikan posisi vertex dalam model space dengan matriks model, kemudian mengalikan hasilnya dengan matriks pandangan. Hasilnya kemudian diubah menjadi vektor 3D dengan mengambil komponen xyz.
- EyeDirection_cameraspace = vec3(0,0,0) - vertexPosition_cameraspace: Menghitung arah dari vertex ke kamera dalam camera space dengan mengurangi posisi vertex dalam camera space dari posisi kamera (yang dianggap berada di vec3(0,0,0)).
- vec3 LightPosition_cameraspace = (V * vec4(LightPosition_worldspace,1)).xyz: Menghitung posisi sumber cahaya dalam camera space dengan mengalikan posisi sumber cahaya dalam world space dengan matriks pandangan. Hasilnya kemudian diubah menjadi vektor 3D dengan mengambil komponen xyz.
- LightDirection_cameraspace = LightPosition_cameraspace + EyeDirection_cameraspace: Menghitung arah dari vertex ke sumber cahaya dalam camera space dengan menambahkan arah dari vertex ke kamera (EyeDirection_cameraspace) dengan posisi sumber cahaya dalam camera space (LightPosition_cameraspace).

- Normal_cameraspace = (V * M * vec4(vertexNormal_modelspace,0)).xyz: Menghitung normal vertex dalam camera space dengan mengalikan normal vertex dalam model space dengan matriks model, kemudian mengalikan hasilnya dengan matriks pandangan. Hasilnya kemudian diubah menjadi vektor 3D dengan mengambil komponen xyz. Catatan: penggunaan komponen 0 di komponen ke-4 (vec4(vertexNormal_modelspace,0)) menandakan bahwa normal bukan posisi, sehingga tidak memerlukan translasi.

- UV = vertexUV: Menetapkan UV dari vertex ke output UV yang akan diinterpolasi untuk setiap fragmen.

2) Fragment Shader

Pencahayaan di OpenGL didasarkan pada perkiraan realitas menggunakan model sederhana yang lebih mudah diproses dan terlihat relatif serupa. Model pencahayaan ini didasarkan pada fisika cahaya seperti yang kita pahami. Salah satu model tersebut disebut model pencahayaan Phong. Komponen utama model pencahayaan Phong terdiri dari 3 komponen: pencahayaan ambient, diffuse, dan specular.



- Pencahayaan Diffuse

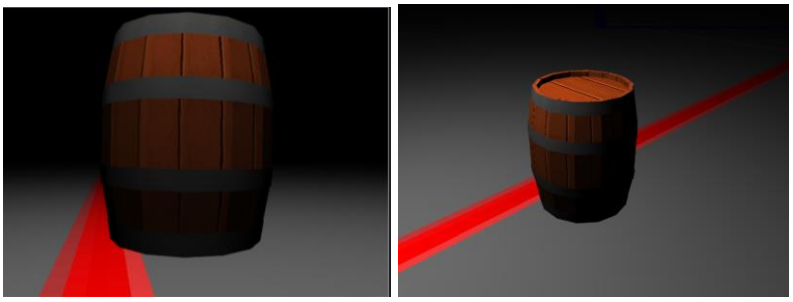
  Pada pencahayaan diffuse kita akan menghitung dari sudut insiden. Ketika sudut insiden lebih kecil(tegak lurus dengan benda) maka cahaya akan lebih mengeluarkan warna benda tersebut. Jadi mensimulasikan dampak arah benda cahaya terhadap suatu benda. Ini adalah komponen model pencahayaan yang paling signifikan secara visual. Semakin banyak bagian suatu objek yang menghadap sumber cahaya, semakin terang objek tersebut. Rumusnya adalah

```
MaterialDiffuseColor * LightColor * LightPower * cosTheta /
(distance*distance)
```

$$\text{DiffuseColor} = \frac{\text{MaterialDiffuseColor} \cdot \text{LightColor} \cdot \text{LightPower} \cdot \cos\theta}{\text{distance}^2}$$

  Keterangan:

  1. MaterialDiffuseColor: Warna yang dipantulkan jika terjadi diffuse (pemantulan baur)
  2. LightColor: Warna dari cahaya yang datang
  3. LightPower: Kekuatan cahayanya
  4. cosTheta: sudut insiden cahaya dan normal (dihitung dengan melakukan perkalian dot , dengan adalah vector normal, dan vector arah cahaya)
  5. distance: jarak posisi vertex ke Cahaya

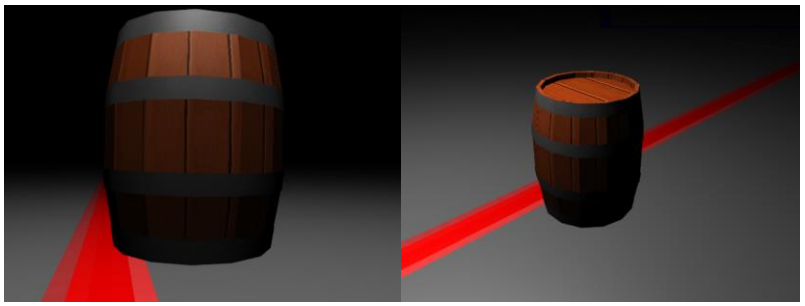


- Pencahayaan Specular
  Mensimulasikan titik terang cahaya yang muncul pada objek berkilau. Sorotan specular lebih condong ke warna cahaya dibandingkan warna objek. Untuk specular kita akan menghitung cahaya yang memantul dan langsung diterima oleh camera. Berikut merupakan rumus dari specular:

```
MaterialSpecularColor * LightColor * LightPower *
pow(cosAlpha,k) / (distance*distance)
```

$$\text{SpecularColor} = \frac{\text{MaterialSpecularColor} \cdot \text{LightColor} \cdot \text{LightPower} \cdot (\cos\alpha)^k}{\text{distance}^2}$$

  Keterangan:
  1. MaterialSpecularolor: Warna yang dipantulkan jika terjadi specular (pemantulan teratur)
  2. LightColor: Warna dari cahaya yang datang
  3. LightPower: Kekuatan cahayanya
  4. cosAlpha: sudut insiden camera dan normal (dihitung dengan melakukan perkalian dot , dengan adalah vector arah camera, dan vector normal yang direfleksikan dengan)
  5. k: angka yang mengatur seberapa besar sudut yang dapat memantulkan cahaya secara teratur
  6. distance: jarak posisi vertex ke Cahaya

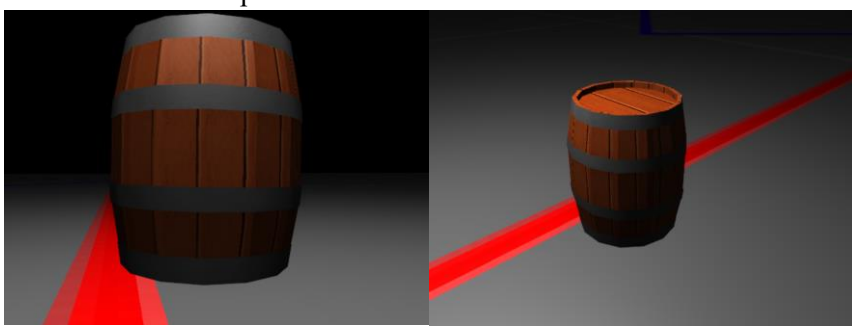


- Pencahayaan Ambient
  Cahaya biasanya tidak datang dari satu sumber cahaya saja, namun dari banyak sumber cahaya yang tersebar di sekitar kita, meski tidak langsung terlihat. Salah satu sifat cahaya adalah ia dapat menyebar dan memantul ke berbagai arah, mencapai titik-titik yang tidak terlihat secara langsung dengan demikian cahaya dapat dipantulkan pada permukaan lain dan berdampak tidak langsung pada pencahayaan suatu objek. Jadi meskipun gelap, biasanya masih ada sedikit cahaya di suatu tempat di dunia (bulan, cahaya di kejauhan) sehingga objek hampir tidak pernah gelap gulita. Untuk mensimulasikan ini kita

menggunakan konstanta pencahayaan sekitar yang selalu memberi warna pada objek. Kita menggunakan warna konstan (cahaya) kecil yang kami tambahkan pada warna akhir yang dihasilkan dari fragmen objek, sehingga membuatnya terlihat seperti selalu ada cahaya yang tersebar meskipun tidak ada sumber cahaya langsung. Kita akan menggunakan 10% warna yang muncul ketika objek berada di ruang terang.

```
MaterialAmbientColor = vec3(0.1,0.1,0.1) * MaterialDiffuseColor;
```

Untuk mengakhiri kalkulasi ini, kita cukup menjumlahkan ketiga nilai yang kita simulasikan ini yakni ambient + diffuse + specular.



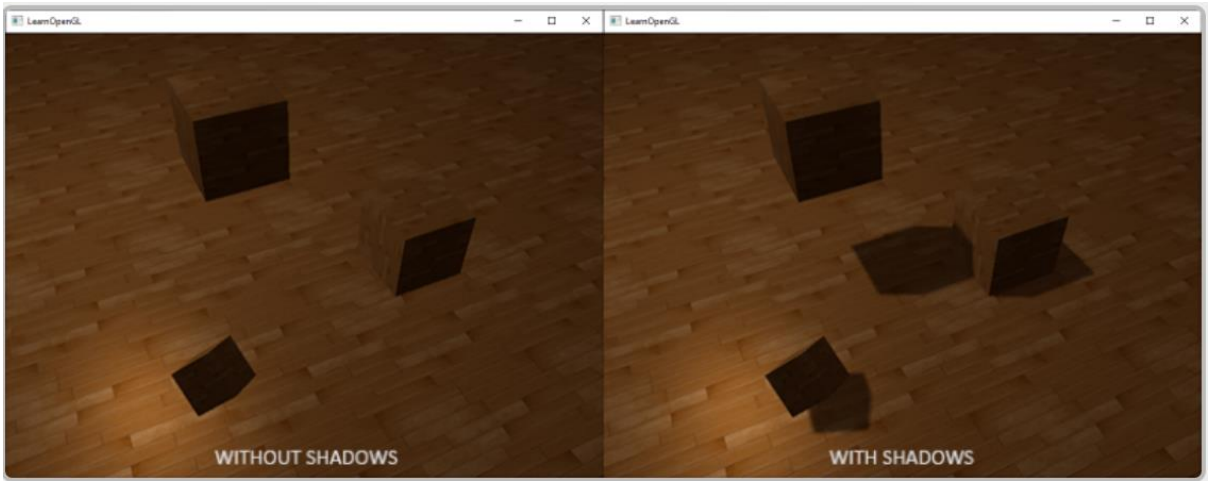
```
#version 330 core

// 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);

}
```

SHADOW

1. Pengertian Shadow

   Bayangan adalah akibat dari tidak adanya cahaya akibat oklusi. Bila berkas cahaya suatu sumber cahaya tidak mengenai suatu benda karena terhalang oleh benda lain, maka benda tersebut berada dalam bayangan. Bayangan menambahkan banyak realisme pada pemandangan yang terang dan memudahkan pemirsa mengamati hubungan spasial antar objek. Mereka memberikan kesan kedalaman yang lebih besar pada pemandangan dan objek kita.



2. Alasan Mengimplementasikan/Mensimulasikan Shadow

   Ada beberapa alasan mengapa mengimplementasikan atau mensimulasikan shadow dalam grafika dan komputasi visual penting:

   - Realistis: Bayangan memberikan efek visual yang realistis dengan meniru perilaku cahaya dan objek di dunia nyata. Ini membantu meningkatkan kualitas visual dan keakuratan adegan.
   - Kedalaman dan Dimensi: Bayangan membantu memberikan kedalaman dan dimensi pada adegan dengan menciptakan kontras antara area terang dan gelap. Ini memberikan tampilan tiga dimensi pada objek dan lingkungan.
   - Keseimbangan Visual: Bayangan dapat menciptakan keseimbangan visual dengan menambahkan detail dan tekstur pada permukaan yang menerima bayangan.
   - Fokus: Bayangan dapat digunakan untuk mengarahkan perhatian penonton ke elemen-elemen penting dalam adegan.
   - Estetika: Bayangan dapat meningkatkan estetika visual dengan memberikan variasi dan kontras dalam adegan, serta menciptakan suasana tertentu.

3. Cara 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;
```

```
// 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;
}
```

Penjelasan code:

- vertexPosition_modelspace: posisi vertex dalam ruang model.
- vertexUV: koordinat tekstur untuk vertex.
- vertexNormal_modelspace: normal permukaan untuk vertex dalam ruang model.
- UV: koordinat tekstur yang akan diteruskan ke fragment shader.
- Position_worldspace: posisi vertex dalam ruang dunia.
- Normal_cameraspace: normal permukaan dalam ruang kamera.
- EyeDirection_cameraspace: arah dari vertex ke kamera dalam ruang kamera.
- MVP: matriks Model-View-Projection yang digunakan untuk menghitung posisi clip space dari vertex.
- V: matriks View yang digunakan untuk transformasi dari ruang model ke ruang kamera.
- M: matriks Model yang digunakan untuk transformasi dari ruang model ke ruang dunia.
- Posisi Clip Space:
    - gl_Position = MVP * vec4(vertexPosition_modelspace,1);

        - Menghitung posisi clip space untuk vertex dengan mengalikan matriks MVP dengan posisi vertex.
    - Posisi Dunia:
        - Position_worldspace = (M * vec4(vertexPosition_modelspace,1)).xyz;
        - Menghitung posisi vertex dalam ruang dunia dengan mengalikan matriks Model (M) dengan posisi vertex.
    - Arah Kamera:
        - EyeDirection_cameraspace = vec3(0,0,0) - vertexPosition_cameraspace;
        - Menghitung arah dari vertex ke kamera dalam ruang kamera.
    - Normal Kamera:
        - Normal_cameraspace = (V * M * vec4(vertexNormal_modelspace,0)).xyz;
        - Menghitung normal permukaan dalam ruang kamera dengan mengalikan matriks View (V) dan Model (M) dengan normal permukaan vertex.
    - UV:
        - UV = vertexUV;
        - Meneruskan koordinat tekstur ke fragment shader.
2) Fragment Shader
    - Shadow Mapping

```
#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);
}
```

Penjelasan code:

- FragColor: warna akhir yang akan diteruskan ke framebuffer.
- UV: koordinat tekstur yang diinterpolasi dari vertex shader.
- Position_worldspace: posisi fragmen dalam ruang dunia.
- Normal_cameraspace: normal permukaan fragmen dalam ruang kamera.
- EyeDirection_cameraspace: arah mata dari fragmen ke kamera dalam ruang kamera.
- LightDirection_cameraspace: arah cahaya dari fragmen ke sumber cahaya dalam ruang kamera.
- Position_lightspace: posisi fragmen dalam ruang cahaya.
- textureSampler: sampler untuk tekstur yang digunakan.
- shadowMap: sampler untuk peta bayangan (shadow map) yang digunakan.
- LightPosition_worldspace: posisi sumber cahaya dalam ruang dunia.
- CameraPosition_worldspace: posisi kamera dalam ruang dunia.

Fungsi ShadowCalculation:

- Menghitung apakah fragmen berada dalam bayangan atau tidak.
- Menggunakan peta bayangan (shadowMap) untuk memeriksa kedalaman fragmen dari perspektif cahaya.
- Menghitung bias untuk mencegah artefak visual.
- Menggunakan metode PCF (Percentage-Closer Filtering) untuk menghaluskan bayangan.
- Mengembalikan nilai bayangan antara 0.0 (tidak ada bayangan) dan 1.0 (sepenuhnya dalam bayangan).

Fungsi main:

- Material Properties:
  - MaterialDiffuseColor: warna difus bahan yang didapat dari tekstur.
  - MaterialAmbientColor: warna ambient bahan, biasanya bernilai rendah.
  - MaterialSpecularColor: warna spekular bahan.
- Menghitung pencahayaan:
  - Menghitung jarak antara fragmen dan sumber cahaya.
  - Menghitung arah cahaya (l) dan arah refleksi (R).
  - Menghitung cosTheta (sudut antara normal permukaan dan arah cahaya) dan cosAlpha (sudut antara arah mata dan arah refleksi).
  - Menghitung bayangan menggunakan fungsi ShadowCalculation.

- Menghitung komponen pencahayaan:
  - ambient: komponen ambient yang rendah.
  - diffuse: komponen difus berdasarkan cosTheta dan distance.
  - specular: komponen spekular berdasarkan cosAlpha dan distance.
- Menggabungkan komponen:
  Komponen-komponen pencahayaan digabungkan untuk menghasilkan warna akhir.
- Mengatur warna akhir:
  - Hasil pencahayaan dikalikan dengan warna bahan (MaterialDiffuseColor).
  - Hasil akhir diatur sebagai FragColor.