Laporan Praktikum GKV
Pertemuan 4 & 5 (Lighting & Shadow)

1. Lighting

- **What**

Lighting/pencahayaan dalam konteks grafika komputer adalah proses atau teknik untuk mengatur distribusi cahaya dalam suatu lingkungan virtual atau adegan grafis. Tujuannya adalah untuk menciptakan tampilan visual yang realistis, menarik, dan memukau bagi pengguna atau pemain. Dalam hal ini, pencahayaan memainkan peran penting dalam membentuk aspek visual dari objek-objek dalam adegan, termasuk warna, tekstur, bayangan, refleksi, dan nuansa keseluruhan. Pencahayaan tidak hanya mempengaruhi bagaimana objek tersebut terlihat, tetapi juga bagaimana mereka berinteraksi satu sama lain dan dengan lingkungan sekitarnya.

Beberapa jenis lighting yang umumnya digunakan:

1. Specular Lighting: Lighting ini menggambarkan pantulan cahaya secara langsung dari sumber cahaya. Ini menciptakan highlight yang tajam pada permukaan objek yang halus dan mengkilap. Misalnya, pada permukaan logam atau air, specular lighting menciptakan refleksi cahaya yang jelas.
2. Diffuse Lighting: Lighting ini menggambarkan penyebaran cahaya pada permukaan objek yang kasar. Ini menciptakan bayangan yang lebih lembut dan lebih merata daripada specular lighting. Misalnya, pada permukaan kayu atau batu, diffuse lighting memberikan efek penyebaran cahaya yang merata.
3. Ambient Lighting: Lighting ini adalah cahaya latar yang tersebar secara merata di seluruh scene. Ini menciptakan pencahayaan umum yang mengisi area-area yang tidak langsung terkena cahaya langsung dari sumber cahaya. Ambient Lighting membantu mengurangi kegelapan yang berlebihan pada area-area yang tidak terkena cahaya langsung.

- **Why**

Lighting akan memberikan dimensi visual yang penting bagi pengalaman pengguna atau pemain. Berikut adalah beberapa alasan mengapa Lighting sangat penting:

1. Realisme: Lighting membantu menciptkan tampilan yang lebih realistis dalam adegan grafis. Dengan mengatur distribusi cahaya yang tepat, objek-objek dapat terlihat lebih hidup dan lebih mirip dengan penampilan asli mereka di dunia nyata.

2. Atmosfer: Lighting mempengaruhi suasana dan atmosfer suatu adegan. Dengan menggunakan lighting yang tepat, pengembang dapat menciptakan berbagai nuansa emosional, mulai dari suasana yang terang dan ceria hingga suasana yang gelap dan misterius.
3. Estetika Visual: Lighting juga berkontribusi pada estetika visual. Dengan menggunakan pencahayaan yang kreatif dan efektif, pengembang dapat menciptakan tampilan yang menarik dan memukau bagi pengguna.
4. Keterbacaan: Lighting yang baik dapat meningkatkan keterbacaan adegan atau antarmuka pengguna. Ini membuat objek atau teks lebih mudah dilihat dan dipahami oleh pengguna, sehingga meningkatkan pengalaman pengguna secara keseluruhan.

- **How**

Menggunakan Lighting yang baik dan benar tentu sangat penting dalam dunia grafika komputer saat ini. Berikut adalah beberapa cara untuk menggunakan lighting dengan tepat dalam aspek yang umum digunakan (diffuse, specular, ambient):

1. Diffuse Lighting:
    - Pastikan sumber cahaya berada dalam posisi yang tepat untuk memberikan pencahayaan yang merata pada objek.
    - Gunakan tekstur dan warna yang sesuai untuk memaksimalkan efek pencahayaan difusi. Permukaan kasar atau tidak rata akan menyebar cahaya dengan lebih merata.
    - JIka memungkinkan, petimbangkan untuk menggunakan teknik pencahayaan global atau pencahayaan tidak lansung untuk meningkatkan kualitas pencahayaan difusi.
2. Specular Lighting:
    - Tentukan posisi dan intensitas sumber cahaya yang sesuai untuk menyoroti bagian-bagian objek yang memiliki permukaan yang halus atau reflektif.
    - Sesuaikan parameter specular seperti kekuatan, ukuran, dan warna highlight untuk mencapai efek yang diinginkan.
    - Pertimbangkan untuk menggunakan teknik seperti pencahayaan lingkungan atau pencahayaan global untuk meningkatkan realisme specular terutama pada permukaan yang reflektif.

3. Ambient Lighting:
    - Tentukan level pencahayaan ambient yang sesuai dengan suasana atau mood yang ingin dicapai dalam adegan.
    - Gunakan pencahayaan ambient dengan hati-hati agar tidak menghilangkan kontras yang diperlukan antara area terang dan gelap dalam adegan.
    - Kombinasikan pencahayaan ambient dengan pencahayaan langsung dan tidak langsung secara proporsional untuk menciptakan pencahayaan keseluruhan yang seimbang dan realistis

- **Code Example (Diffuse, Specular, 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;
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;
}
```

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**

  Shadow dalam grafika komputer adalah fenomena dimana objek memblokir cahaya, menyebabkan area di belakangnya menjadi gelap atau lebih redup. Ketika sebuah objek menerima cahaya, bayangannya muncul di area di belakang objek tersebut. Penanganan bayangan adalah salah satu aspek penting dari realisme visual karena memberikan kedalaman dan dimensi pada adegan.

  Berikut beberapa teknik yang biasanya digunakan:

  1. Shadow Mapping: Melibatkan pembuatan peta kedalaman (depth map) dari perspektif sumber cahaya. Peta kedalaman tersebut kemudian digunakan untuk menentukan apakah titik-titik dalam adegan tertutup atau terkena cahaya, apakah bayangan harus diterapkan di titik tersebut atau tidak. Meskipun shadow mapping efektif, teknik ini memiliki beberapa keterbatasan, seperti aliasing dan artefak bayangan yang mungkin terjadi.

2. Point Shadow/Shadow Volumes: Pembentukan volume bayangan di sekitar objek yang memblokir cahaya. Volume bayangan ini kemudian digunakan untuk menentukan area yang tertutup dari dari sudut pandang sumber cahaya, sehingga bayangan dapat diterapkan dengan benar. Teknik ini memiliki keunggulan tertentu dalam menangani bayangan dari sumber cahaya yang bergerak atau perubahaan geometri yang kompleks.Teknik shadow yang biasa digunakan.

- **Why**

Berikut beberapa alasan mengapa kita harus menggunakan shadow:

1. Memberikan kedalaman: Shadow memberikan kedalaman pada adegan grafis dengan menunjukkan perbedaan antara area yang terkena cahaya dan yang tidak terkena cahaya. Ini membantu menciptakan ilusi ruang dan jarak antara objek-objek dalam adegan.
2. Meningkatkan realisme: Dalam kehidupan nyata, objek memblokir cahaya dan menciptakan bayangan. Dengan menambahkan bayangan pada objek dalam adegan, kita dapat meningkatkan realisme visualnya, membuatnya terasa lebih alami dan meyakinkan.
3. Memberikan informasi spatial: Shadow membantu pengamat untuk memahami struktur dan penempatan objek dalam adegan. Mereka memberikan petunjuk visual tentang posisi dan bentuk objek, yang dapat membantu pemain atau pengguna dalam navigasi atau memecahkan teka-teki dalam permainan.
4. Meningkatkan estetika: Shadow juga dapat digunakan untuk meningkatkan estetika adegan grafis. Mereka dapat menciptakan kontras yang menarik, menambahkan drama, atau membentuk pola menarik pada permukaan dimana bayangan jatuh.
5. Menyampaikan mood dan atmosfer: Lighting dan shadow dapat digunakan bersama-sama untuk menyampaikan mood atau atmosfer tertentu dalam adegan. Shadow yang lembut dan diffuse misalnya, dapat menciptakan suasana yang tenang dan damai, sementara bayangan yang tajam dan kontras dapat menciptakan suasana yang gelap dan mencekam.

- **How**

Menggunakan shadow dengan tepat, baik, dan benar sangat penting dalam konsep grafika komputasi saat ini. Berikut adalah beberapa cara untuk menggunakan teknik shadow secara tepat dan baik:

1. Shadow Mapping
    - Buat peta kedalaman dari perspektif sumber cahaya. Pastikan resolusi peta kedalaman mencukupi untuk menjaga kualitas bayangan.

- Render adegan dari perspektif sumber cahaya dan simpan kedalaman setiap piksel ke dalam peta kedalaman.
- Render adegan dari perspektif kamera utama. Saat melakukan render ini, gunakan informasi peta kedalaman untuk. menentukan apakah piksel tertentu berada dalam bayangan atau tidak.
- Gunakan informasi dari peta kedalaman untuk menentukan bagian mana dari adegan yang harus dianggap dalam bayangan.
- Gunakan teknik sampling dan interpolasi yang tepat untuk mengurangi aliasing dan artefak bayangan.

2. Poin Shadow
    - Tentukan dimana volumes bayangan terbentuk di sekitar objek yang memblokir cahaya.
    - Gunakan teknik seperti z-buffer atau stencil buffer untuk menentukan area yang terkena bayangan.
    - Render volumes bayangan di dalam adegan. Ini bisa berupa volume silinder atau prisma yang mencakup area yang tertutup dari sudut pandang sumber cahaya.
    - Pastikan volumes bayangan terbentuk dengan benar, mengikuti geometri objek yang memblokir cahaya.
    - Render adegan utama dari sudut pandang kamera. Gunakan volumes bayangan yang telah di-render sebelumnya untuk menentukan piksel mana yang berada dalam bayangan.
    - Gunakan informasi dari volumes bayangan untuk menentukan bagian mana dari adegan yang harus dianggap dalam bayangan.
    - Pastikan teknik rendering bayangan yang tepat digunakan untuk mendapatkan hasil yang akurat dan halus

- **Code Example (Shadow mapping)**
  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;      Reylyer, last month • init belum fix

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;      Reylyer, last month • init belum fix
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 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);
}
```