

**LAPORAN PRAKTIKUM
PRAKTIKUM Ke- 4 & 5:
ESSAY PERTEMUAN 4 & 5 MENGENAI LIGHTING DAN SHADOW**



Disusun oleh:
Zahra Nisaa' Fitria Nur'Afifah
24060122140162

**PRAKTIKUM GRAFIKA DAN KOMPUTASI VISUAL
LAB B1**

**DEPARTEMEN INFORMATIKA
FAKULTAS SAINS DAN MATEMATIKA
UNIVERSITAS DIPONEGORO**

2024

LIGHTING DAN SHADOW

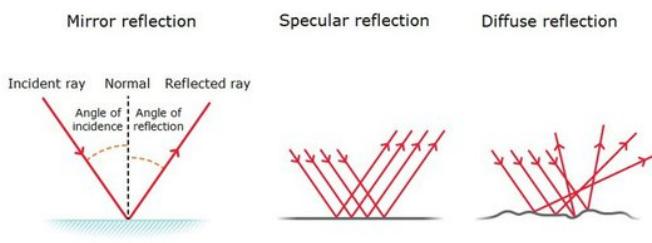
1. Lighting

Pertemuan 4 - Lighting

Pada pertemuan ini kita akan mempelajari bagaimana lighting bekerja di dunia graphics. Tapi sebelum kita dapat memulai mensimulasikan cahaya, kita perlu tahu dahulu bagaimana cahaya bekerja.

Light

Mengingat kembali masa SMA atau SMP(?) Kita pernah belajar mengenai cahaya. kita mempelajari ketika suatu cahaya jatuh kepada suatu permukaan, maka cahaya tersebut akan memantul yang pantulannya dipengaruhi oleh kemulusan dari permukaannya. Kita akan membahas 2 behavior ini.



© The University of Waikato Te Whare Wānanga o Waikato | www.sciencelearn.org.nz

Specular Reflection | Pemantulan teratur

Pada permukaan yang mulus, cahaya yang datang akan dipantulkan. Sesuai hukum pemantulan, sudut datang (angle of incidence) akan sama dengan sudut yang dipantulkan.

Diffuse Reflection | Pemantulan Baur

Dalam bahasa Indonesia, diffuse artinya memencarkan. Ketika cahaya jatuh di permukaan yang kasar, maka cahaya akan memantul dari permukaan dengan sebagian energi dari cahaya akan memancar ke segala arah.

Masih ada beberapa fenomena cahaya ketika menabrak sebuah benda seperti transmisi, refraksi, difraksi, absorpsi, dan scattering. Untuk mempersimpel simulasi cahaya kita, kita hanya akan mensimulasikan sifat refleksi cahaya saja.

Simulasi cahaya di Shader

Kita akan melakukan kalkulasi cahaya lebih banyak di fragment shader. Vertex shader akan digunakan sebagai interpolator dari world space ke camera space.

Vertex shader

Di dalam vertex shader kita akan menghitung tiap vertex untuk

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

Fragment Shader

Diffuse

Simulasi refleksi di fragment shader ini cukup rumit. Kita akan membahas dulu dari diffuse reflection. Diffuse kita akan menghitung dari sudut insiden. Ketika sudut insiden

lebih kecil(tegak lurus dengan benda) maka cahaya akan lebih mengeluarkan warna benda tersebut. Rumus dari diffuse adalah:

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

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

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

Specular

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)
```

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

- MaterialSpecularColor: Warna yang dipantulkan jika terjadi specular (pemantulan teratur)
- LightColor: Warna dari cahaya yang datang
- LightPower: Kekuatan cahayanya
- cosAlpha: sudut insiden camera dan normal (dihitung dengan melakukan perkalian dot , dengan adalah vector arah camera, dan vector normal yang direfleksikan dengan)
- k: angka yang mengatur seberapa besar sudut yang dapat memantulkan cahaya secara teratur
- distance: jarak posisi vertex ke cahaya

Ambient light

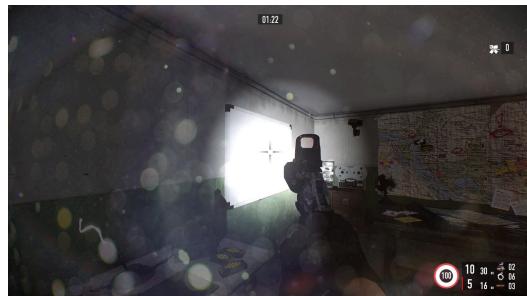
Pada dunia nyata sebenarnya setiap cahaya yang memantul tidak hanya sekali dan hilang, tiap cahaya yang masih memiliki energi akan terus memantul hingga habis. Ini menyebabkan suatu ruangan tertutup tidak 100% gelap ketika ada sumber cahaya yang kecil. Untuk mensimulasikan fenomena ini, kita hanya akan membuat bahwa setiap warna yang ada di objek memiliki tingkat keterangan minimal. Kita akan menggunakan 10% warna yang muncul ketika objek berada di ruang terang.

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

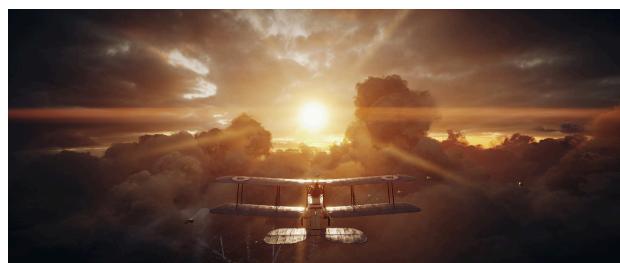
untuk mengakhiri kalkulasi ini, kita cukup menjumlahkan ketiga nilai yang kita simulasikan ini yakni ambient + diffuse + specular.

More advanced topics on light

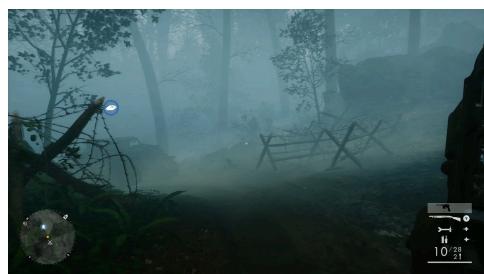
- Lens Dirt



- Lens Flare



- Fog



- Chromatic Abberation



- Skybox



- Bloom



- Depth of Field

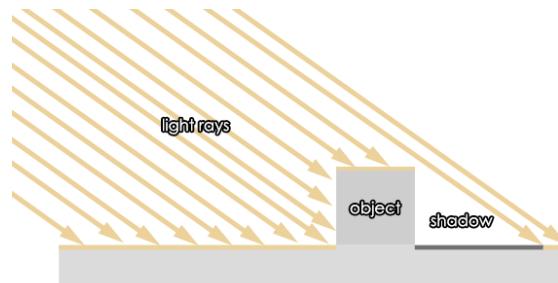


- and more
- SSAO
- HBAO
- SSDO
- HDR
- Global Illumination
- Ray Marching
- Water stuff, refraction and diffraction

2. Shadow

Pertemuan 5 - Shadow

Shadow adalah bayangan atau gambaran dari objek yang tertutup oleh sumber cahaya. Shadow berfungsi untuk menambahkan dimensi dan kedalaman pada gambar, serta membuat objek terlihat lebih realistik dan 3D. Shadow dapat dihasilkan melalui pengolahan cahaya yang berbeda-beda, seperti pencahayaan yang langsung dari sumber cahaya atau pencahayaan yang dibuat oleh objek yang berada di sekitar sumber cahaya.



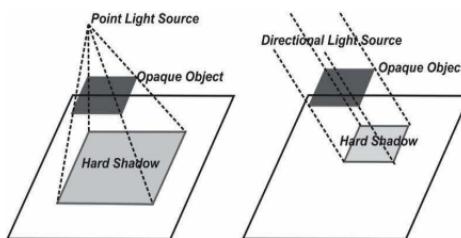
Jika melihat keluar dari sumber cahaya, semua benda yang dilihat akan tampak terkena cahaya. Namun, apa pun yang berada di balik benda-benda itu akan berada

dalam bayangan. Ini adalah prinsip dasar yang digunakan untuk membuat peta bayangan. Tampilan cahaya dirender, menyimpan kedalaman setiap permukaan yang dilihatnya (peta bayangan). Selanjutnya, pemandangan biasa dirender dengan membandingkan kedalaman setiap titik yang digambar (seolah-olah dilihat oleh cahaya, bukan oleh mata) dengan peta kedalaman ini.

Teknik ini kurang akurat dibandingkan volume bayangan , namun peta bayangan dapat menjadi alternatif yang lebih cepat bergantung pada berapa banyak waktu pengisian yang diperlukan untuk salah satu teknik dalam aplikasi tertentu dan oleh karena itu mungkin lebih cocok untuk aplikasi waktunya. Selain itu, peta bayangan tidak memerlukan penggunaan buffer stensil tambahan dan dapat dimodifikasi untuk menghasilkan bayangan dengan tepi yang lembut. Berbeda dengan volume bayangan, keakuratan peta bayangan dibatasi oleh resolusinya.

Bayangan memiliki dua bagian area yaitu umbra dan penumbra. Umbra adalah bagian bayangan objek dengan tingkat kegelapan maksimum dikarenakan cahaya dari sumber cahaya terhalang sepenuhnya oleh objek (occluder). Penumbra, bagian bayangan yang masih terkena cahaya sebagian, yang membentuk batas area dari bayangan dengan perubahan tingkat kegelapan yang halus.

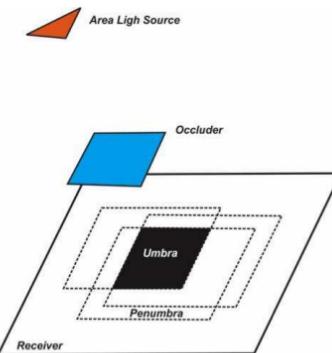
- Hard Shadow dihasilkan oleh point light source. Anatomi hard shadow biasanya hanya terdiri atas bagian umbra saja, lihat Gambar 1. Hard shadow memberikan kesan tidak realistik, karena pada praktiknya point light source ideal tidak ada di dunia nyata



Gambar 1. Hard shadow dihasilkan oleh point light source

- Secara umum Soft Shadow anatominya digambarkan memiliki area tengah yang gelap (umbra) yang dikelilingi batas area dengan perubahan kegelapan yang halus (penumbra) antara daerah bayangan menuju ke daerah tanpa bayangan. Soft shadows dihasilkan oleh area light source jelas nampak lebih

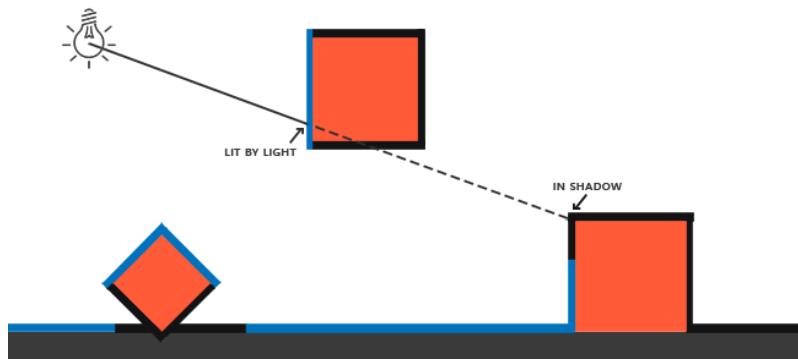
realistik dibandingkan hard shadow, apalagi dengan tingkat kehalusan bayangan yang bervariasi secara dramatis menurut jarak antara sumber cahaya (light source), penghalang (occluder) dan penerima (receiver). Terlihat pada Gambar 2.



Gambar 2. Soft Shadow dihasilkan oleh *area light source*

Pemetaan bayangan

Ide di balik pemetaan bayangan cukup sederhana: kita merender pemandangan dari sudut pandang cahaya dan segala sesuatu yang kita lihat dari sudut pandang cahaya akan menyala dan segala sesuatu yang tidak dapat kita lihat pasti berada dalam bayangan. Bayangkan bagian lantai dengan kotak besar di antara dirinya dan sumber cahaya. Karena sumber cahaya akan melihat kotak ini dan bukan bagian lantai ketika melihat ke arahnya, maka bagian lantai tertentu harus berada dalam bayangan.

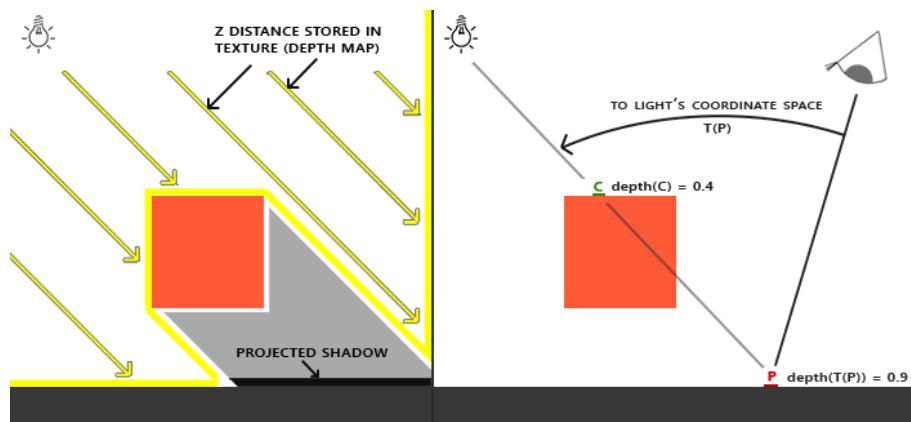


Di sini semua garis biru mewakili pecahan yang dapat dilihat oleh sumber cahaya. Fragmen yang tersumbat ditampilkan sebagai garis hitam: ini dianggap berbayang. Jika kita menggambar garis atau sinar dari sumber cahaya hingga pecahan di kotak paling kanan kita dapat melihat sinar terlebih dahulu mengenai wadah terapung sebelum mengenai wadah paling kanan. Akibatnya, pecahan wadah terapung menyala dan pecahan wadah paling kanan tidak menyala sehingga berada dalam bayangan.

Kita ingin mendapatkan titik pada sinar dimana ia pertama kali menembus suatu benda dan membandingkan titik terdekatnya dengan titik lain pada sinar tersebut. Kami kemudian melakukan tes dasar untuk melihat apakah posisi sinar titik tes lebih jauh ke bawah daripada titik terdekat dan jika demikian, titik tes harus berada dalam

bayangan. Melakukan iterasi melalui kemungkinan ribuan sinar cahaya dari sumber cahaya seperti itu adalah pendekatan yang sangat tidak efisien dan tidak cocok untuk rendering real-time. Kita dapat melakukan hal serupa, tetapi tanpa memancarkan sinar cahaya. Sebagai gantinya, kami menggunakan sesuatu yang cukup kami kenal: buffer kedalaman.

Bahwa nilai dalam buffer kedalaman sesuai dengan kedalaman fragmen yang dijepit ke $[0,1]$ dari sudut pandang kamera. Bagaimana jika kita merender pemandangan dari sudut pandang cahaya dan menyimpan nilai kedalaman yang dihasilkan dalam sebuah tekstur? Dengan cara ini, kita dapat mengambil sampel nilai kedalaman terdekat jika dilihat dari sudut pandang cahaya. Bagaimanapun juga, nilai kedalaman menunjukkan fragmen pertama yang terlihat dari sudut pandang cahaya. Kami menyimpan semua nilai kedalaman ini dalam tekstur yang kami sebut peta kedalaman atau peta bayangan.



3. Shader

Shader adalah program kecil yang dirancang untuk mengontrol bagaimana cahaya, warna, dan tekstur diterapkan pada objek 3D dalam proses rendering.

4. Soal

1. Apa itu lighting?
2. Mengapa mengimplementasi/mensimulasikan lighting itu penting?
3. Bagaimana cara mengimplementasikannya?
4. Apa itu Shadow?
5. Mengapa mengimplementasi/mensimulasikan shadow itu penting?
6. Bagaimana cara mengimplementasikannya?

5. Source Code Mengimplementasikan Lighting

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 omitted 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;
}

```

Source Code Ambient light

```

#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);

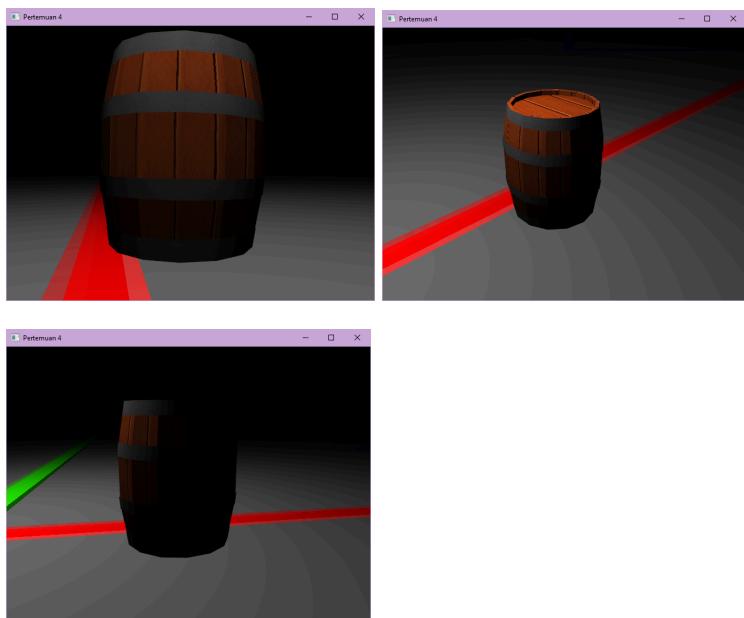
}

```

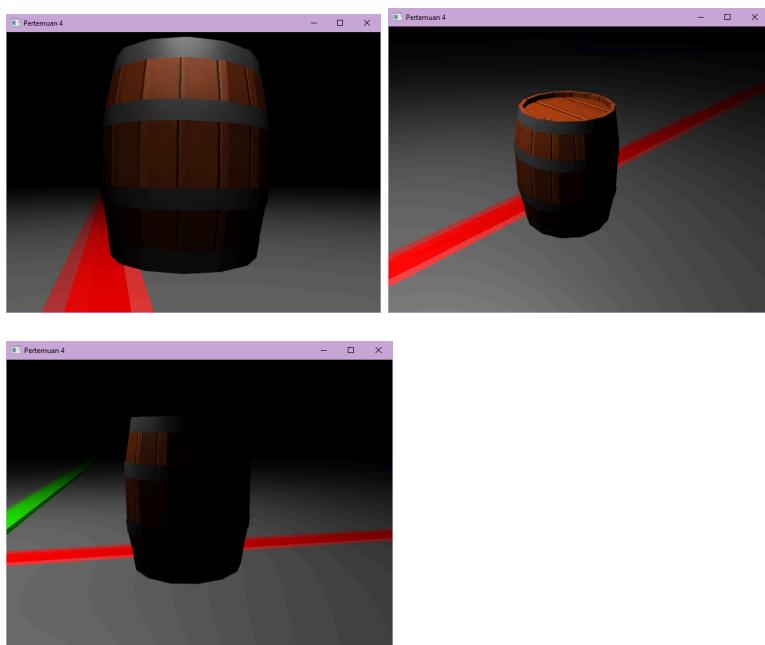
6. Screenshot Program : Input dan Output (Jika Ada)

Gallery

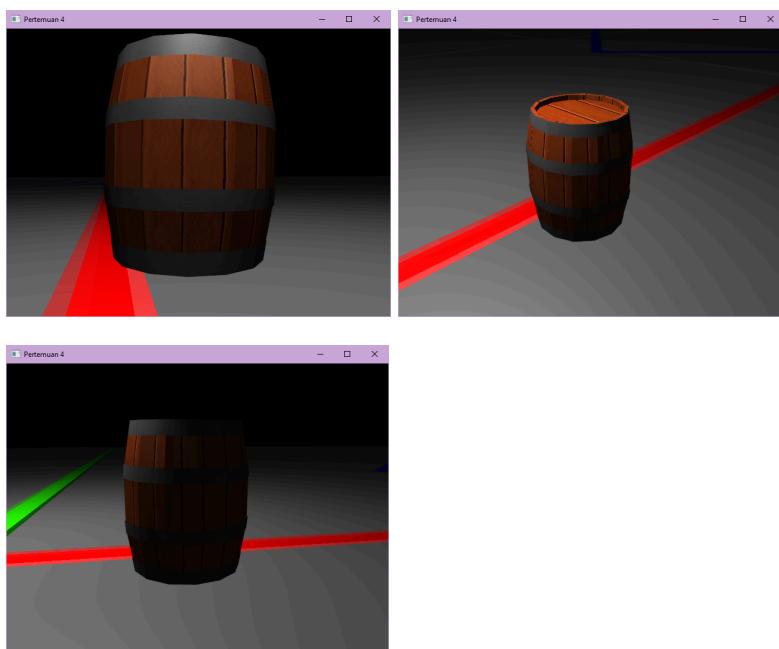
Only Diffuse



Diffuse + Specular



Diffuse + Specular + Ambient



7. Pembahasan Soal Lighting

Lighting merupakan proses atau teknik untuk mensimulasikan efek cahaya di dalam sebuah ruang tiga dimensi (3D) dalam grafika komputer. Pentingnya mengimplementasikan lighting dalam grafika dan komputasi visual komputer yaitu, memberikan tampilan yang lebih realistik pada objek dan adegan 3D, pencahayaan mempengaruhi suasana dan mood dari adegan, menciptakan efek emosional tertentu, dapat digunakan untuk menyoroti atau menekankan bagian-bagian tertentu dari adegan, mempertahankan konsistensi visual antara elemen-elemen dalam adegan.

8. Source Code Mengimplementasikan Shadow

Shadow Mapping

```
#version 330 core
out vec4 FragColor;

in VS_OUT {
    vec3 FragPos;
    vec3 Normal;
    vec2 TexCoords;
    vec4 FragPosLightSpace;
} fs_in;

uniform sampler2D diffuseTexture;
```

```

uniform sampler2D shadowMap;

uniform vec3 lightPos;
uniform vec3 viewPos;

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(fs_in.Normal);
    vec3 lightDir = normalize(lightPos -
fs_in.FragPos);
    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 color = texture(diffuseTexture,
fs_in.TexCoords).rgb;
    vec3 normal = normalize(fs_in.Normal);
    vec3 lightColor = vec3(0.3);
    // ambient
    vec3 ambient = 0.3 * lightColor;
    // diffuse
    vec3 lightDir = normalize(lightPos -
fs_in.FragPos);
    float diff = max(dot(lightDir, normal), 0.0);
    vec3 diffuse = diff * lightColor;
    // specular
    vec3 viewDir = normalize(viewPos -
fs_in.FragPos);
    vec3 reflectDir = reflect(-lightDir, normal);
    float spec = 0.0;
    vec3 halfwayDir = normalize(lightDir + viewDir);
    spec = pow(max(dot(normal, halfwayDir), 0.0),
64.0);
}

```

```

    vec3 specular = spec * lightColor;
    // calculate shadow
    float shadow =
        ShadowCalculation(fs_in.FragPosLightSpace);
    vec3 lighting = (ambient + (1.0 - shadow) *
        (diffuse + specular)) * color;

    FragColor = vec4(lighting, 1.0);
}

```

9. Penjelasan

FragColor:

out vec4 FragColor; mendefinisikan variabel output untuk warna piksel (fragment) yang akan ditampilkan di layar. Piksel akan diberi warna sesuai dengan nilai yang diberikan ke FragColor.

FS_IN:

ni mendefinisikan struktur VS_OUT yang digunakan untuk mengirim data dari vertex shader ke fragment shader. Ini mencakup:

- FragPos: Posisi fragmen dalam ruang.
- Normal: Vektor normal permukaan pada fragmen.
- TexCoords: Koordinat tekstur untuk fragmen.
- FragPosLightSpace: Posisi fragmen dalam ruang cahaya.

Uniforms:

Ini adalah variabel uniform yang digunakan untuk mengirim data dari aplikasi ke shader.

- diffuseTexture: Tekstur untuk difusi objek.
- shadowMap: Tekstur yang berisi data kedalaman (depth) dari sudut pandang cahaya.
- lightPos: Posisi cahaya dalam ruang.
- viewPos: Posisi kamera dalam ruang.

ShadowCalculation:

Ini adalah fungsi yang menghitung apakah sebuah fragmen berada dalam bayangan atau tidak. Ini dilakukan dengan membandingkan kedalaman fragmen yang dihitung dari sudut pandang cahaya dengan kedalaman fragmen terdekat dalam shadowMap. Fungsi ini juga memperhitungkan bias untuk mengatasi artefak bayangan.

Main:

Ini adalah fungsi utama yang dipanggil untuk setiap piksel pada layar. Di dalamnya, pencahayaan ambient, diffuse, specular, dan bayangan dihitung untuk fragmen tersebut.

Ambient lighting:

- Ini adalah pencahayaan konstan yang ada di seluruh lingkungan, yang diberikan oleh `lightColor`.

Diffuse lighting:

- Ini adalah pencahayaan yang disebabkan oleh sumber cahaya yang difus, dihitung dengan memperhitungkan sudut antara normal permukaan dan vektor dari titik cahaya ke titik fragmen.

Specular lighting:

- Ini adalah pencahayaan yang menghasilkan highlight atau kilauan pada permukaan objek, dihitung berdasarkan sudut antara vektor pandang (view vector) dan vektor pantulan cahaya (reflected light vector).

Shadow:

- Ini adalah faktor yang mengurangi intensitas pencahayaan pada objek jika objek berada dalam bayangan. Ini dihitung menggunakan fungsi `ShadowCalculation`.
- Akhirnya, warna akhir dari piksel dihitung dengan menggabungkan pencahayaan ambient, diffuse, dan specular dengan mempertimbangkan efek bayangan. Hasil akhirnya disimpan dalam `FragColor`.

Shader fragmen utama yang akan kita gunakan untuk merender pemandangan menggunakan model pencahayaan Blinn-Phong. Di dalam shader fragmen, kami kemudian menghitung nilai bayangan saat 1.0 fragmen berada dalam bayangan atau 0.0 saat tidak berada dalam bayangan. Komponen difus dan specular yang dihasilkan kemudian dikalikan dengan komponen bayangan tersebut. Karena bayangan jarang benar-benar gelap (akibat hamburan cahaya), kita tidak memasukkan komponen sekitar ke dalam perkalian bayangan.

Di akhir shader fragmen, kita mengalikan kontribusi difus dan specular dengan kebalikan dari komponen bayangan, misalnya berapa banyak fragmen yang tidak berada dalam bayangan. Shader fragmen ini mengambil input tambahan posisi fragmen ruang cahaya dan peta kedalaman yang dihasilkan dari render pass pertama.

10. Pembahasan Soal Shadow

Shadow adalah area gelap yang terbentuk ketika cahaya terhalangi oleh objek, menciptakan perbedaan intensitas cahaya di permukaan atau latar belakang. Pentingnya mengimplementasikan Shadow yaitu, meningkatkan realisme visual dengan menambahkan dimensi dan kedalaman pada objek dan adegan 3D, membantu memperkuat persepsi kedalaman dan ruang dalam adegan, memberikan kontribusi pada atmosfer dan mood dengan menciptakan kontras dan drama visual.

11. Kesimpulan

Pencahayaan dan bayangan merupakan aspek kunci dalam menciptakan grafika komputer yang memukau. Cahaya memungkinkan kita untuk mengungkapkan detail

objek dan lingkungan secara visual, sementara bayangan memberikan kedalaman dan dimensi tambahan yang memperkaya pengalaman visual. Melalui implementasi pencahayaan, kita dapat menonjolkan fitur-fitur penting, menciptakan atmosfer yang sesuai, dan meningkatkan kualitas keseluruhan pengalaman pengguna dalam sebuah gambar. Dengan kombinasi yang cermat antara pencahayaan diffuse, specular, dan ambien, kita dapat menciptakan tampilan yang sangat realistik dan mendalam pada permukaan objek. Dengan demikian, pencahayaan dan bayangan tidak hanya meningkatkan kualitas visual, tetapi juga memperkaya sensasi estetika bagi para pengguna.

Dalam konteks pencahayaan, teknik ini memungkinkan simulasi efek cahaya dalam ruang tiga dimensi. Implementasinya tidak hanya penting untuk mendapatkan tingkat realisme yang tinggi dalam visualisasi, tetapi juga untuk menciptakan atmosfer dan mood yang diinginkan, menyoroti aspek-aspek penting dari objek, dan menjaga konsistensi visual dalam suatu karya. Sementara itu, bayangan membawa elemen tambahan yang vital dengan memperkuat persepsi kedalaman, meningkatkan realisme visual, dan menciptakan suasana yang khas. Menggunakan teknik seperti shadow mapping, ray tracing, atau shadow volumes, implementasi bayangan dapat memperkaya pengalaman pengguna dengan memberikan konteks visual yang lebih kuat dan mendalam.

Dengan menyatukan kedua aspek ini secara efektif, pengguna dapat mencapai pencapaian visual yang mengesankan dan menggugah dalam bidang Grafika Komputer dan Visualisasi (GKV).

DAFTAR PUSTAKA

“Pertemuan 4 - Lighting.”

<https://amygdala.shariyl.cloud/Modul+-+GKV/Pertemuan+4+-+Lighting>, diakses tanggal 18 April 2024.

Suni, F.A., Fathoni. K. (2016). Klasifikasi Shadow Algorithm. Jurnal Teknik Elektro 8(2), 1-4.

<https://media.neliti.com/media/publications/133805-ID-klasifikasi-shadow-algorithm.pdf>, Diakses tanggal 18 April 2024.

“Belajar Pencahayaan dan Bayangan untuk Menghasilkan Karya Seni yang Sempurna.”

<https://design.tutsplus.com/id/articles/improve-your-artwork-by-learning-to-see-light-and-shadow--cms-20282>, diakses tanggal 19 April 2024.

“Shadow Mapping.” <https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping>, diakses tanggal 20 April 2024.

“Advanced Lighting.” <https://learnopengl.com/Advanced-Lighting/Advanced-Lighting>, diakses tanggal 20 April 2024.