LAPORAN UAS GRAFIKA KOMPUTER

Anggota Kelompok :
- Albert Valentino - c14200135
- Samuel Chandra - c14200139
- Andre Cristian Leo - c14200179

- Alur cerita

Di suatu malam ketika ulat sedang tertidur di pohon lewat lah kecoa yang sedang terbang dan tidak sengaja menabrak ulat hingga ia berputar dan terjatuh ke tanah dan tewas. Beberapa waktu kemudian datanglah seekor semut yang sedang mencari makanan ketika dia sampai ke bawah pohon dia melihat ulat yang sudah tewas, ia pun segera mengangkat ulat tersebut untuk dimakan nantinya. (lebih jelasnya cek video demo)

- Third person viewer

Objek diletakkan tepat di depan kamera sehingga terlihat seperti kita menggunakan objek tersebut

```
var temp = new Asset3d("", "", new Vector3(1, 1, 1));
bug1 = temp.createCockroach(-11f, -1.5f, .5f);
bug1.child[0].scaleNew(0.1f, 0.1f, 0.1f, bug1.child[0].objectCenter);
bug1.child[1].scaleNew(0.1f, 0.1f, 0.1f, bug1.child[1].objectCenter);
bug1.rotate(Vector3.Zero, Vector3.UnitY, 90);
```

Ketika kamera digerakkan maka objek juga ikut ditranslasi sehingga seakan akan kita menggunakan objek tersebut untuk bergerak dan menciptakan kesan third person perspective (TPP)

```
#region cameraMovements
if (input.IsKeyDown(Keys.W)) // forward
{
    if (!isOverlaps() && camera.Position.Z > -28.25f)
    {
        camera.Position += Vector3.Normalize(Vector3.Cross(camera.Up, camera.Right)) * cameraSpeed * time;
        bug1.translate(0, 0, -cameraSpeed * (float)args.Time);
    }
    else
    {
        camera.Position -= Vector3.Normalize(Vector3.Cross(camera.Up, camera.Right)) * cameraSpeed * .01f;
        bug1.translate(0, 0, cameraSpeed * (float).01f);
    }

}
```

```
if (input.IsKeyDown(Keys.S)) // backward
{
    if (!isOverlaps() && camera.Position.Z < 28.25f)
    {
        camera.Position -= Vector3.Normalize(Vector3.Cross(camera.Up, camera.Right)) * cameraSpeed * time;
        bug1.translate(0, 0, cameraSpeed * (float)args.Time);
    }
    else
    {
        camera.Position += Vector3.Normalize(Vector3.Cross(camera.Up, camera.Right)) * cameraSpeed * .01f;
        bug1.translate(0, 0, -cameraSpeed * (float).01f);
    }
}
```

```
if (input.IsKeyDown(Keys.A)) // left
{
    if (!isOverlaps() && camera.Position.X > -48)
    {
        camera.Position -= camera.Right * cameraSpeed * time;
        bug1.translate(-cameraSpeed * (float)args.Time, 0f, 0f);
    }
    else
    {
        camera.Position += camera.Right * cameraSpeed * .01f;
        bug1.translate(cameraSpeed * (float).01f, 0f, 0f);
    }
}
```

```
if (input.IsKeyDown(Keys.D)) // right
{
    if (!isOverlaps() && camera.Position.X < 48)
    {
        camera.Position += camera.Right * cameraSpeed * time;
        bug1.translate(cameraSpeed * (float)args.Time, 0f, 0f);
    }
    else
    {
        camera.Position -= camera.Right * cameraSpeed * .01f;
        bug1.translate(-cameraSpeed * (float).01f, 0f, 0f);
    }
}
```

```
if (input.IsKeyDown(Keys.Space)) // up
{
    if (!isOverlaps() && camera.Position.Y < 35)
    {
        camera.Position += camera.Up * cameraSpeed * time;
        bug1.translate(0f, cameraSpeed * (float)args.Time, 0f);
    }
    else
    {
        camera.Position -= camera.Up * cameraSpeed * .01f;
        bug1.translate(0f, -cameraSpeed * (float).01f, 0f);
    }
}
```

```
if (input.IsKeyDown(Keys.Z)) // down
{
    if (!isOverlaps() && camera.Position.Y > -9.25f)
    {
        camera.Position -= camera.Up * cameraSpeed * time;
        bug1.translate(0f, -cameraSpeed * (float)args.Time, 0f);
    }
    else
    {
        camera.Position += camera.Up * cameraSpeed * .01f;
        bug1.translate(0f, cameraSpeed * (float).01f, 0f);
    }
}
```

```
#region cameraRotations
if (KeyboardState.IsKeyDown(Keys.X)) // right
{
    bug1.child[5].rotate(bug1.child[5].objectCenter, Vector3.UnitY, (float)args.Time * -100);
    camera.Yaw += cameraSpeed * (float)args.Time * 10;
    headDir = false;
}
if (KeyboardState.IsKeyDown(Keys.C)) // left
{
    bug1.child[5].rotate(bug1.child[5].objectCenter, Vector3.UnitY, (float)args.Time * 100);
    camera.Yaw -= cameraSpeed * (float)args.Time * 10;
    headDir = false;
}
if (KeyboardState.IsKeyDown(Keys.V)) // down
{
    bug1.child[5].rotate(bug1.child[5].objectCenter, Vector3.UnitX, (float)args.Time * -100);
    camera.Pitch -= cameraSpeed * (float)args.Time * 10;
    headDir = false;
}
if (KeyboardState.IsKeyDown(Keys.B)) // up
{
    bug1.child[5].rotate(bug1.child[5].objectCenter, Vector3.UnitX, (float)args.Time * 100);
    camera.Pitch += cameraSpeed * (float)args.Time * 10;
    headDir = false;
}
#endregion
```

- Shading ambient, diffuse, dan specular

Set titik normal untuk tiap objek yang ada.

```
if (useNormals)
{
    normals.Add(new Vector3(temp_vector.X, temp_vector.Y, temp_vector.Z));
}
```

*titik normal map ellipsoid, torus, hyperboloid elliptic, dan cone

```
if (useNormals)
{
    for (int i = 0; i < tempIndices.Count; i++)
    {
        vertices.Add(tempVertices[(int)tempIndices[i]]);
    }

    for (int i = 0; i < 6; i++)
    {
        normals.Add(-Vector3.UnitZ);
    }

    for (int i = 0; i < 6; i++)
    {
        normals.Add(Vector3.UnitY);
    }

    for (int i = 0; i < 6; i++)
    {
        normals.Add(Vector3.UnitX);
    }

    for (int i = 0; i < 6; i++)
    {
        normals.Add(-Vector3.UnitX);
    }

    for (int i = 0; i < 6; i++)
    {
        normals.Add(Vector3.UnitZ);
    }

    for (int i = 0; i < 6; i++)
    {
        normals.Add(-Vector3.UnitY);
    }
}
```

*titik normal map cuboid

Directional Light dipakai sebagai **bulan.** Memberi penerangan pada seluruh objek di dunia ini.

**cahaya tidak berasal dari objek tersebut, ini hanya cerita nya saja bahwa bulan menerangi seluruh objek pada 3D World ini.*

Variabel dan perhitungan yang diperlukan untuk Directional Light.

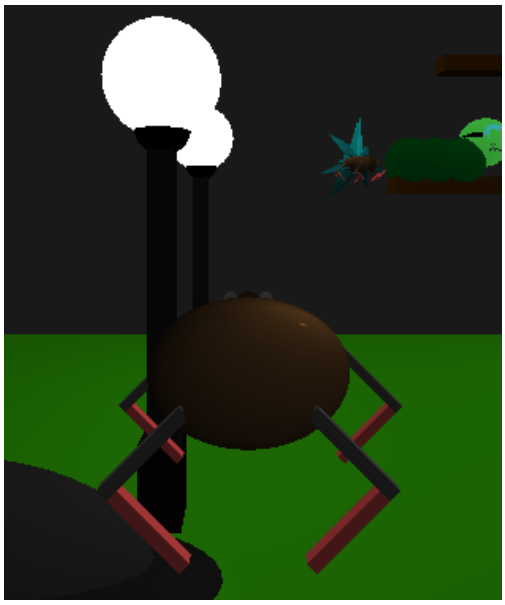
```
struct DirLight {
    vec3 direction;

    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};
uniform DirLight dirLight;
```

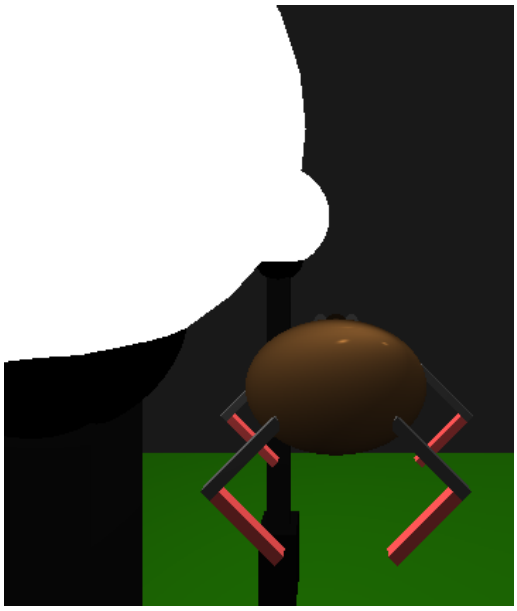
```
vec3 CalcDirLight(DirLight light, vec3 normal, vec3 viewDir)
{
    vec3 lightDir = normalize(-light.direction);

    // diffuse shading
    float diff = max(dot(normal, lightDir), 0.0);

    // specular shading
    vec3 reflectDir = reflect(-lightDir, normal);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), 256);

    // combine results
    vec3 ambient  = light.ambient  * objectColor;
    vec3 diffuse  = light.diffuse  * diff * objectColor;
    vec3 specular = light.specular * spec *
    objectColor;
    return (ambient + diffuse + specular);
}
```

Untuk mengeset tiap variabel yang diperlukan Directional Light.

```
2 references
public void setDirectionalLight(Vector3 direction, Vector3 ambient, Vector3 diffuse, Vector3 specular)
{
    _shader.SetVector3("dirLight.direction", direction);
    _shader.SetVector3("dirLight.ambient", ambient);
    _shader.SetVector3("dirLight.diffuse", diffuse);
    _shader.SetVector3("dirLight.specular", specular);
}
```

Point Lights dipakai pada tiap **tiang lampu**. Memberi penerang pada posisi tertentu sesuai dengan letak tiap tiang lampu yang ada. Jumlah sumber cahaya bisa lebih dari 1.





Gambar kiri saat jauh dari sumber cahaya. Gambar kanan saat dekat sumber cahaya.

```
struct PointLight {
    vec3 position;

    float constant;
    float linear;
    float quadratic;
    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};
// #define NR_POINT_LIGHTS "jumlah point light"
#define NR_POINT_LIGHTS 4
uniform PointLight pointLights[NR_POINT_LIGHTS];
```

Variabel dan perhitungan yang diperlukan untuk Point Lights.

```
vec3 CalcPointLight(PointLight light, vec3 normal, vec3 fragPos, vec3 viewDir)
{
    vec3 lightDir = normalize(light.position - fragPos);

    // diffuse shading
    float diff = max(dot(normal, lightDir), 0.0);

    // specular shading
    vec3 reflectDir = reflect(-lightDir, normal);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0),256);

    // attenuation
    float distance    = length(light.position - fragPos);
    float attenuation = 1.0 / (light.constant + light.linear * distance +
    light.quadratic * (distance * distance));

    //combine results
    vec3 ambient  = light.ambient  * objectColor;
    vec3 diffuse  = light.diffuse  * diff * objectColor;
    vec3 specular = light.specular * spec * objectColor;
    ambient  *= attenuation;
    diffuse  *= attenuation;
    specular *= attenuation;
    return (ambient + diffuse + specular);
}
```

Untuk mengeset tiap variabel yang diperlukan Point Lights.

```
2 references
public void setPointLight(List<Asset3d> positions, Vector3 ambient, Vector3 diffuse, Vector3 specular, float constant, float linear, float quadratic)
{
    for (int i = 0; i < positions.Count; i++)
    {
        _shader.SetVector3($"pointLights[{i}].position", positions[i].objectCenter);
        _shader.SetVector3($"pointLights[{i}].ambient", ambient);
        _shader.SetVector3($"pointLights[{i}].diffuse", diffuse);

        _shader.SetVector3($"pointLights[{i}].specular", specular);
        _shader.SetFloat($"pointLights[{i}].constant", constant);
        _shader.SetFloat($"pointLights[{i}].linear", linear);
        _shader.SetFloat($"pointLights[{i}].quadratic", quadratic);
    }

}
```

Untuk mentotal hasil perhitungan directional light dan tiap point light untuk dimasukkan ke tiap objek pada 3D World.

```
vec3 normal = normalize(normal);
vec3 viewPos = normalize(viewPos - vec3(FragPos));


vec3 result = vec3(0, 0, 0);

// DIRECTIONAL LIGHT
vec3 directionalLight = CalcDirLight(dirLight, normal, viewPos);
result += directionalLight;

// POINT LIGHTS
for(int i = 0; i < NR_POINT_LIGHTS; i++)
    result += CalcPointLight(pointLights[i], normal, vec3(FragPos), viewPos);

outputColor = vec4(result, 1.0);
```