DX1219: PRACTICAL 4

The aim of this practical is to build upon previous light shader and add in different kind of lighting type (Directional, Point, Spot).

Setup

1. I will recommend you to create a new shader for this as there will be quite a bit of changes.
    a. Here are my current properties based on practical 3. Remember to change the name of the shader at the top as well. Feel free to add in your own stuff.

```
Properties
{
    _mainTexture("Albedo", 2D) = "white" {}
    _tint("Tint", Color) = (1,1,1,1)
    _alphaCutoff("Alpha Cutoff", Range(0, 1)) = 0.5

    _lightPosition("Light Position", Vector) = (0,0,0)
    _lightDirection("Light Direction", Vector) = (0,-1,0)
    _lightColor("Light Color", Color) = (1,1,1,1)
    _specularStrength("Specular Strength", Range(0, 1)) = 0.5
    _smoothness("Smoothness", Range(0, 1)) = 0.5
}
```

    b. Here are my uniforms for the initial setup.

```
//Color and Texture
uniform float4 _tint;
uniform sampler2D _mainTexture;
uniform float4 _mainTexture_ST;
uniform float _alphaCutoff;

//Light Data
uniform float3 _lightPosition;S
uniform float3 _lightDirection;
uniform float4 _lightColor;
uniform float _specularStrength;
uniform float _smoothness;
```

    c. Here are my 2 structs.
        i. Application to Vertex Data

```
struct vertexData {
    float4 position: POSITION;
    float2 uv: TEXCOORD0;
    float3 normal: NORMAL;
};
```

        ii. Vertex Shader to Fragment Shader

```
struct vertex2Fragment {
    float4 position: SV_POSITION;
    float2 uv: TEXCOORD0;
    float3 normal: NORMAL;
    float3 worldPosition: POSITION1;
};
```

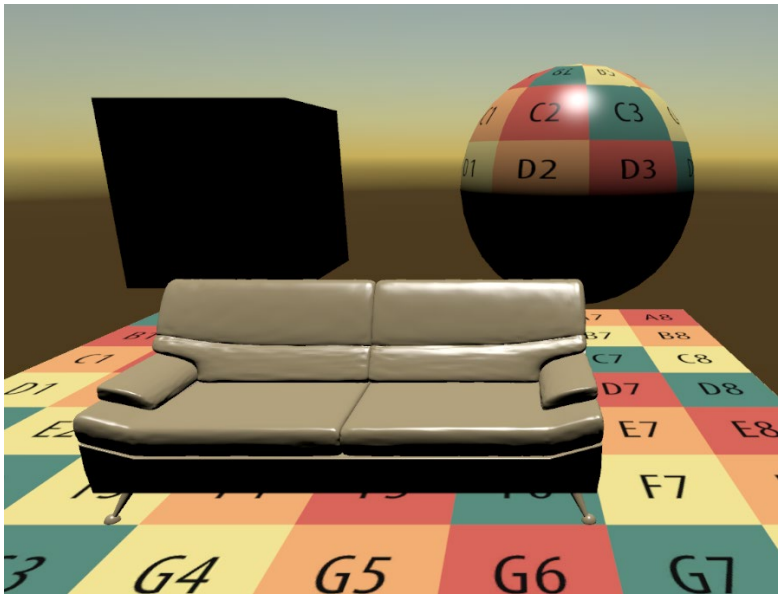
d. Here is my vertex shader.

```
vertex2Fragment MyVertexShader (vertexData vd)
{
    vertex2Fragment v2f;
    v2f.position = UnityObjectToClipPos(vd.position);
    v2f.worldPosition = mul(unity_ObjectToWorld, vd.position);
    v2f.uv = vd.uv * _mainTexture_ST.xy + _mainTexture_ST.zw;
    v2f.normal = UnityObjectToWorldNormal(vd.normal);
    v2f.normal = normalize(v2f.normal);
    return v2f;
}
```

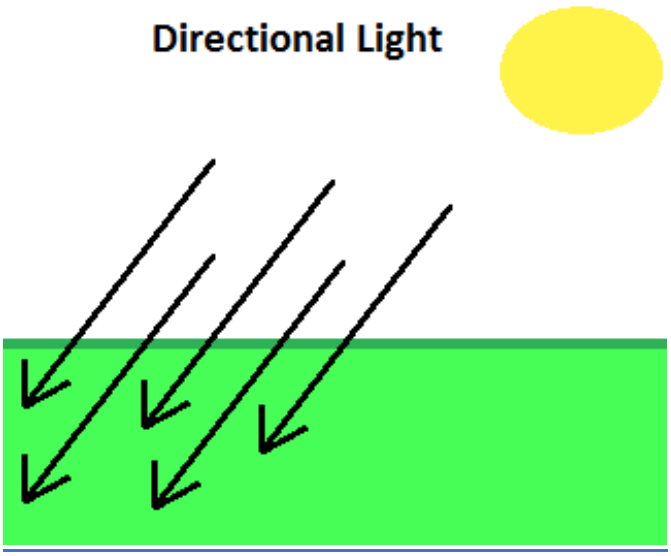
e. Here is the fragment shader.

```
float4 MyFragmentShader(vertex2Fragment v2f) : SV_TARGET
{
    v2f.normal = normalize(v2f.normal);
    float4 albedo = tex2D(_mainTexture, v2f.uv) * _tint;
    float3 viewDirection = normalize(_WorldSpaceCameraPos - v2f.worldPosition);
    float3 halfVector = normalize(viewDirection + -_lightDirection);
    float specular = pow(float(saturate(dot( v2f.normal, halfVector))), _smoothness * 100);
    float3 specularColor = specular * _specularStrength * _lightColor.rgb ;
    float3 diffuse = albedo.rgb * _lightColor.rgb * saturate(dot( v2f.normal, -_lightDirection));
    float3 finalColor = diffuse + specularColor;

    // Return the final color with alpha from the albedo texture.
    return float4(finalColor, albedo.a);
}
```



Directional Light



2. A Directional light is a light source that is so far away that the rays are parallel to each other example, the Sun. Therefore, only the direction vector is important for this type of light and the position is not needed.
3. so Let create our own light type and send it through the properties to shaders. First let add in the following values in the LightObject class .

```
public enum Type
{
    DIRECTIONAL = 0,
    POINT = 1,
    SPOT = 2,
}
[SerializeField]
private Type type;
```

In the SendToShader() function, add this.

```
material.SetInteger("_lightType", (int)type);
```

In the shader properties

```
_lightType("Light Type", Integer) = 1
```

And inside the shader

```
uniform int _lightType;
```

We will also add in a Light Intensity variable

```
[SerializeField]
[Range(0f, 10f)]
private float intensity;
```

In the SendToShader() function, add this.

```
material.SetFloat("_lightIntensity", intensity);
```

In the shader properties

```
_lightIntensity("Light Intensity", float) = 1
```

And inside the shader

```
uniform float _lightIntensity;
```

We already have **_lightPosition** from last week practical as this is important for point and spot light where it will be used to determine where will be lit. **_lightDirection** will be for directional light and spot light.

Change to update to this instead to make use the **_lightDirection**

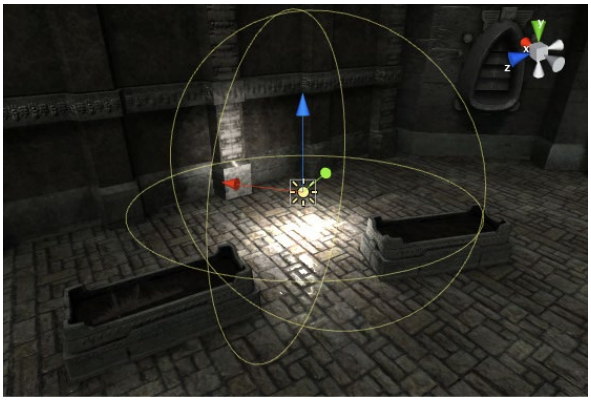
```
float3 finalLightDirection;
if(_lightType == 0)
{
    finalLightDirection = _lightDirection;
}
```

Update to use **finalLightDirection** and **_lightIntensity** instead

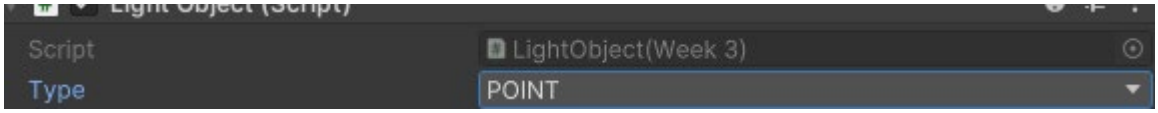
```
float3 viewDirection = normalize(_WorldSpaceCameraPos - v2f.worldPosition);
float3 halfVector = normalize(viewDirection + -finalLightDirection);
float specular = pow(float(saturate(dot( v2f.normal, halfVector))), _smoothness * 100);
float3 specularColor = specular * _specularStrength * _lightColor.rgb ;
float3 diffuse = albedo.rgb * _lightColor.rgb * saturate(dot( v2f.normal, -finalLightDirection));
float3 finalColor = (diffuse + specularColor) * _lightIntensity;
```

Point Light

Directional light is useful for global light that light up the whole scene. However, we want to create light that add like a simple light source such as a glowing orb or lamp. This is known as a point light. A point light is a light source with a given position and shine in all direction but the light rays fade out over distance.

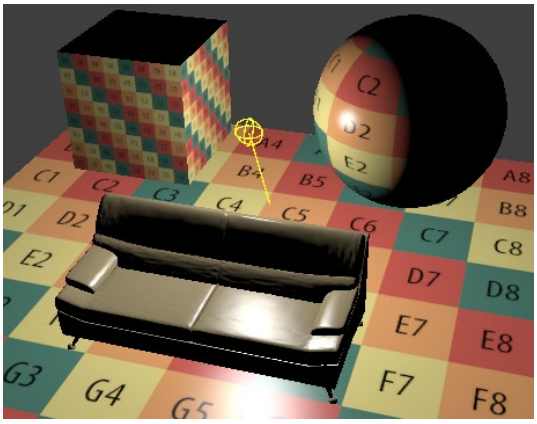


*Remember to change the **light type** to **POINT**



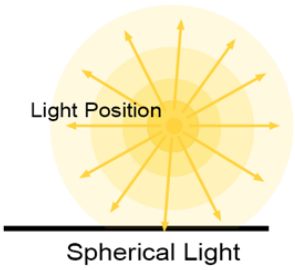

```
//Directional Light
if(_lightType == 0)
{
    finalLightDirection = _lightDirection;
}
//Point Light and Spot Light
else
{
    finalLightDirection = normalize(v2f.worldPosition -_lightPosition);
}
```

You would see something like this



Light Attenuation

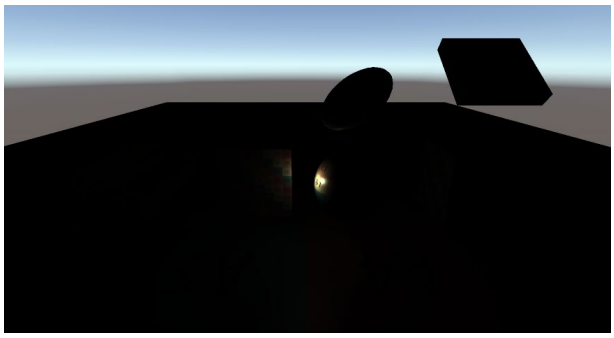
To reduce the intensity of the light over distance a light ray travels is known as attenuation. The current implantation is using a linear equation where objects at a distance is less bright. However linear function looks a bit fake. There are different types of attenuation formula. The surface area of a sphere using radius r is equal to $4\pi r^2$. We can ignore 4π. So, we use $\frac{1}{d^2}$ where d is the light's distance.



Use the attenuation on the final light value in the fragment shader.

```
float3 finalColor = (diffuse + specularColor) * _lightIntensity * attenuation;
```

You will get an effect like this.



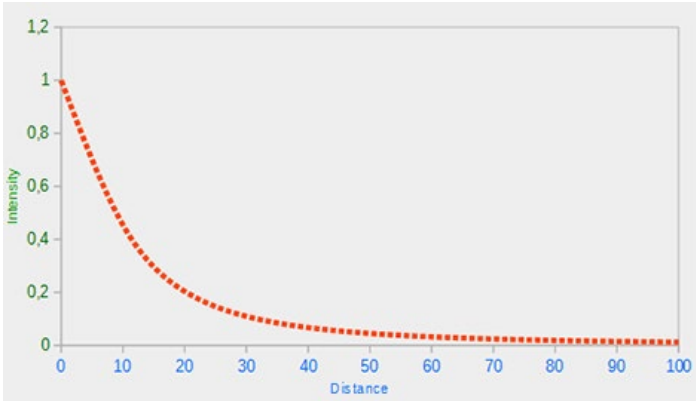
Let improve the formula using an improved **attenuation** formula.

$$\boldsymbol{attenuation} = \frac{\mathbf{1}}{\boldsymbol{k_c} + \boldsymbol{k_l} * \boldsymbol{d} + \boldsymbol{k_q} * \boldsymbol{d^2}}$$

Where k_c is a constant term

k_l , a linear term,

k_q , a quadratic term.



Distance	Constant	Linear	Quadratic
7	1.0	0.7	1.8
13	1.0	0.35	0.44
20	1.0	0.22	0.20
32	1.0	0.14	0.07
50	1.0	0.09	0.032
65	1.0	0.07	0.017
100	1.0	0.045	0.0075
160	1.0	0.027	0.0028
200	1.0	0.022	0.0019
325	1.0	0.014	0.0007
600	1.0	0.007	0.0002
3250	1.0	0.0014	0.000007

Let set the properties and uniform and change the attenuation code

In the light object, create a vector3 variable named attenuation.

x will be our **Attenuation Constant**; y will be our **Attenuation Linear** and Z will be our **Attenuation Quadratic.**

```
[SerializeField]
private Vector3 attenuation = new Vector3(1.0f, 0.09f, 0.032f);
```

In the SendToShader() function, add this.

```
material.SetVector("_attenuation", attenuation);
```

In the shader file add this in the properties.

```
_attenuation("Attenuation", Vector) = (1.0, 0.09, 0.032)
```

And setup the uniform.

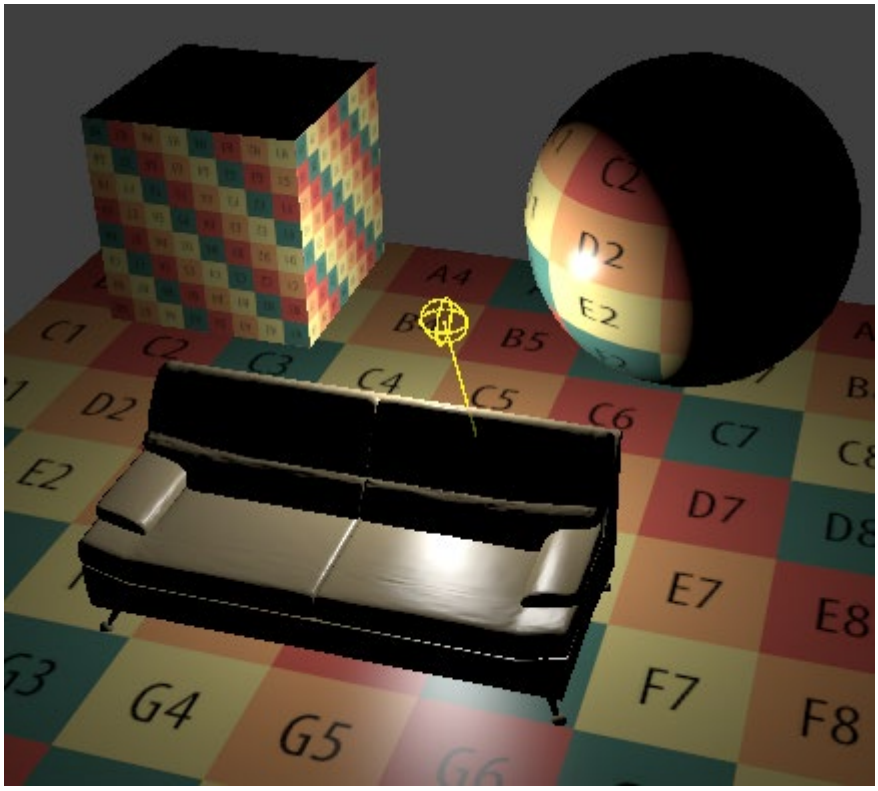
```
uniform float3 _attenuation;
```

Update the attenuation code to be the same the formula above. Remember to take note one of them is an uniform float3, the other is a float without the underscore.

```
float attenuation = 1.0;
```

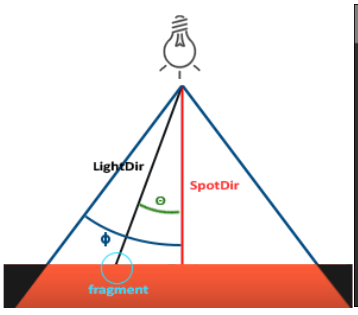
```
finalLightDirection = normalize(v2f.worldPosition -_lightPosition);
float distance = length(v2f.worldPosition -_lightPosition);
attenuation = 1.0 / (_attenuation.x + _attenuation.y * distance
                    + _attenuation.z * distance * distance;
```

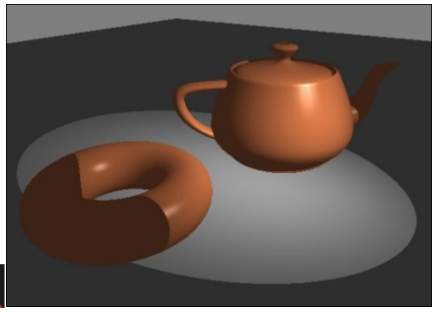
You have a smooth attenuation now. I tried this with an intensity of 5. Try moving the light position as well.



Spot Light

The last type of light is a **Spotlight**. A spotlight is a type of point light so it has a position as well but instead of shooting light rays in all directions. It only shoots them in a specific direction there fore only objects within a certain radius are light and everything else stay dark. A good example of a spotlight will be a street lamp with a cover or a torchlight.





There are 2 angle, one (phi) is the cutoff angle that defines the spotlight radius. The other is the angle between the light direction (the vector from fragment to the light source) to the spot direction (the direction the spot light is looking at).

*Remember to change the **light type** to **SPOT**

Set the properties and uniform for the spot light cut off.

ADD THE PROPERTIES IN LIGHTOBJECT SCRIPT BY YOUR OWN

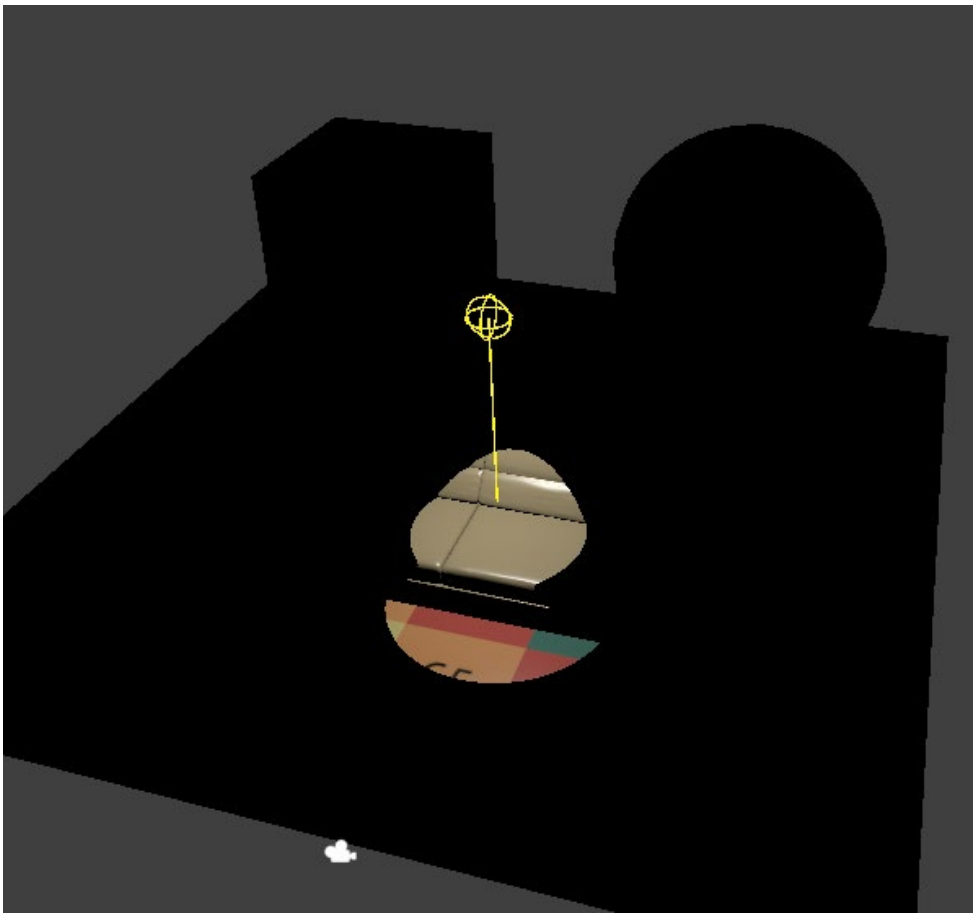
```
_spotLightCutOff("Spot Light CutOff", Range(0, 360)) = 20.0
```

```
uniform float _spotLightCutOff;
```

```
else
{
    finalLightDirection = normalize(v2f.worldPosition -_lightPosition);
    float distance = length(v2f.worldPosition -_lightPosition);
    attenuation = 1.0 / (_attenuation.x + _attenuation.y * distance
                        + _attenuation.z * distance * distance);

    if(_lightType == 2)
    {
        float theta = dot(finalLightDirection, _lightDirection);
        if(theta > cos(radians(_spotLightCutOff)))
        {

        }
        else
        {
            attenuation = 0.0;
        }
    }
}
```



You will notice that the flash light is cut off with hard edges at the angle. Let try making it with soft edges instead. In order to have soft edge we will have an **inner** and **outer** cone. In the inner cone, nothing will change but at the other cone, it would calculate an intensity value between 0.0 to 1.0.

Let set the properties and uniform adding an inner cone cut off angle.

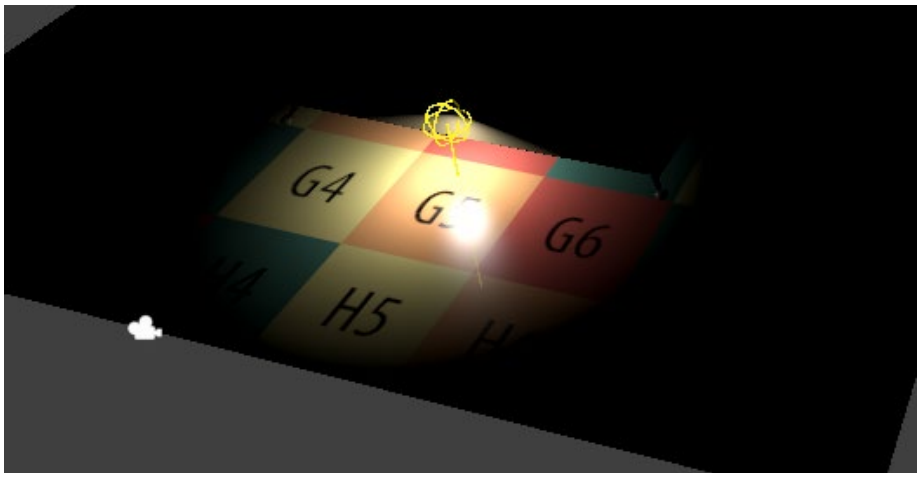
```
_spotLightCutOff("Spot Light CutOff", Range(0, 360)) = 70.0
_spotLightInnerCutOff("Spot Light Inner CutOff", Range(0, 360)) = 25.0
```

```
uniform float _spotLightCutOff;
uniform float _spotLightInnerCutOff;
```

ADD THE PROPERTIES IN LIGHTOBJECT SCRIPT BY YOUR OWN

We will calculate the intensity using the inner and outer angle. We have to clamp the value between 0.0 and 1.0. Also, the inner cutoff has to be smaller than the outer cutoff for it to work properly. If the outer and inner are the same, it would be only hard edges.

```
if(_lightType == 2)
{
    float theta = dot(finalLightDirection, _lightDirection);
    float angle = cos(radians(_spotLightCutOff));
    if(theta > angle)
    {
        float epsilon = cos(radians(_spotLightInnerCutOff)) - angle;
        float intensity = clamp((theta - angle)/epsilon, 0.0, 1.0);
        attenuation *= intensity;
    }
    else
    {
        attenuation = 0.0;
    }
}
```



You will notice that the spot light has smooth soft edges.