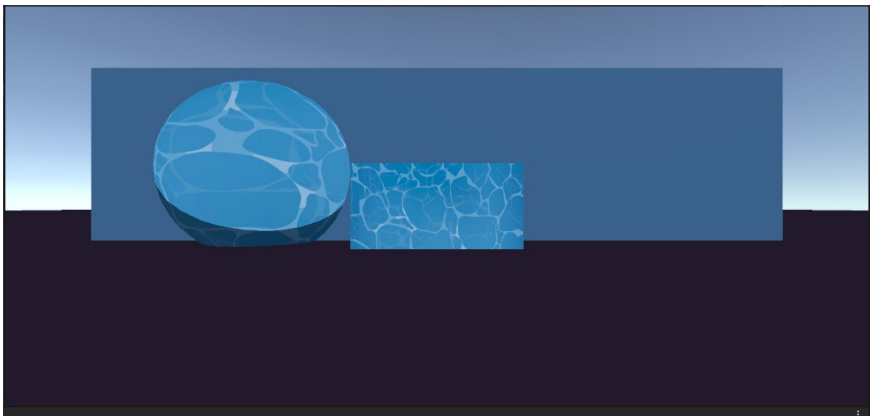
Lit Shader



a) *Diffuse* shaders are used for non-metallic or rough surfaces and non-illuminating surfaces. In simple terms, they determine the general colour of a material when light shines on it: the surface appears uniformly bright from all directions. The Lambertian shader is based on *cos angle = N.L*, where *N* is the surface normal, and *L* refers to the light direction; this means that the amount of light that a surface receives is directly proportional to the angle between the surface normal and the light direction.

The formula can also be rewritten as:

$$\begin{aligned} \text{Diffuse Surface Color} &\propto \text{Incident Light Energy} * \cos\theta \\ &\propto \text{Incident Light Energy} * N.L. \end{aligned}$$

For the peer review, a LambertianShader implements the concepts described above as follows [1]:

```
//To get the normal direction, the surface normal (which is in modal space) needs to be transformed in world space.
// For that reason, the surface normal is multiplied by the unity_WorldToObject matrix.
// The normalise function is used to return a unit vector.
// It represents the direction of the surface.
float3 normalDirection = normalize(mul(float4(v.normal, 0.0), unity_WorldToObject).xyz);

// Light direction relative to the world space, which is normalised to a unit vector
// It represents L or the light direction.
float3 lightDirection = normalize(_WorldSpaceLightPos0.xyz);

// The dotProduct function calculates the light intensity by finding the dot product of the light direction and normal direction.
// Since the dot product can be negative, max is used to filter out vertices where light does not fall
float dotProduct = max(0.0, dot(normalDirection, lightDirection));

// The attenuation variable provides strength of light that falls on the surface.
float attenuation = 1.0;
// The formula below corresponds to Incident Light Energy * N.L
// _LightColour0.xyz gives the colour of the direction light, while _Colour.rgb is the colour of the user-defined variable
float3 diffuseReflection = attenuation * _LightColour0.xyz * _Colour.rgb * dotProduct;
// The vertex is transformed to view space and then clip space.
output.position = UnityObjectToClipPos(v.vertex);
return output;
```

b) The complete lighting equation is given by three times of light: *ambient, specular* and *diffuse* lighting. The whole lighting equation is provided, in simple terms, as ambient_lighting + diffuse_lighting + specular_lighting

Ambient lighting is background light that bounces off everywhere and illuminates every surface part. Its formula is: C_a*[G_a + sum(Atteni*Spoti*Lai)].

Specular lighting is based on the reflective purposes of surfaces. It falls more rapidly across an object's surface. It occurs when light hits an object's surface and reflects toward the camera. Its equation can be given as Specular Lighting = C□ * sum[L□ * (N · H)P * Atten * Spot].

Diffuse shaders are used for non-metallic or rough surfaces and non-illuminating surfaces. In simple terms, they determine the general colour of a material when light shines on it: the surface appears uniformly bright from all directions. Its formula is: sum[Cd*Ld*(N.Ldir)*Atten*Spot].

In the peer-graded review, a FullLightingShader implements the lighting equation as follows [2]:

```
// The normalise function returns a unit vector.
// It represents the direction of the surface.
float3 normalDirection = i.normalDir;

// Light direction relative to the world space, which is normalised to a unit vector
// It represents L or the light direction.
float3 lightDirection = normalize(_WorldSpaceLightPos0.xyz);

// The dotProduct function calculates the light intensity by finding the dot product of the light direction and normal direction.
/ Since the dot product can be negative, max is used to filter out vertices where light does not fall
float dotProduct = max(0.0, dot(normalDirection, lightDirection));

// The attenuation variable provides the strength of light that falls on the surface.
float attenuation = 1.0;
// The formula below corresponds to Incident Light Energy * N.L
// _LightColour0.xyz gives the colour of the direction light, while _Colour.rgb is the colour of the user-defined variable
float3 diffuseReflection = attenuation * _LightColour0.xyz * _Colour.rgb * dotProduct;

// Calculating specular highlights are the bright spots of light that appear on shiny objects when illuminated.
// Reflect - returns the reflection vector given an incident vector and a normal vector.
// return i - 2.0 * n * dot(n, i); is how it can be implemented
// Multiplying the light direction by negative one (-1) gives the opposite direction of the light direction vector.
float3 lightReflectDirection = reflect(-lightDirection, normalDirection);

// The view direction is given by subtracting the camera position from the vertex's position.
// It is then normalised as the direction is required.
float3 viewDirection = normalize(float3(float4(_WorldSpaceCameraPos.xyz, 1.0) - i.posWorld.xyz));

// How much-reflected light can be seen by the camera given by dot product of light reflected direction and the view direction.
float3 lightSeeDirection = max(0.0, dot(lightReflectDirection, viewDirection));

// _Shininess is a user-defined variable
float3 shininessPower = pow(lightSeeDirection, _Shininess);
float3 specularReflection = attenuation * _SpecColour.rgb * shininessPower;

/ Full lighting is calculated via the formula: ambient_lighting + diffuse_lighting + specular_lighting
// For that reason, UNITY_LIGHTMODEL_AMBIENT is added together with diffuse and specular reflections.
float3 finalLight = diffuseReflection + specularReflection + UNITY_LIGHTMODEL_AMBIENT;

return float4(finalLight * _Colour.rgb, 1.0);
```

c) For the extension, a ToonShader implements a toon-effect in the fragment shader as shown below:

```
float3 normal = normalize(i.worldNormal);

// Calculate the amount of light received by the surface from the main directional light
float NdotL = dot(_WorldSpaceLightPos0, normal);

// Adds the ability for the shader to cast and receive shadows
float shadow = SHADOW_ATTENUATION(i);

// Divide the lighting into two bands, light and dark, to create a more realistic toon effect
// The smooth step returns a value between 0 and 1 based on how far this third value is between the bounds
// It blends the transition from 0 to 1.
float lightIntensity = smoothstep(0, 0.01, NdotL * shadow);

// Light intensity is calculated with the directional light
float4 light = lightIntensity * _LightColor0;

/ Calculate specular reflection to model the distinct light from sources
// Fits in with the formula: Specular Lighting = C□ * sum[L□ * (N · H)P * Atten * Spot]
float3 viewDir = normalize(i.viewDir);

// The half vector is between the viewing direction and the light source.
float3 halfVector = normalize(_WorldSpaceLightPos0 + viewDir);

// The strength of the specular reflection is defined as the dot product between the normal of the surface and the half vector.
float NdotH = dot(normal, halfVector);

// Multiply NdotH by lightIntensity to ensure that the reflection is only drawn when the surface is lit
```

```
float specularIntensity = pow(NdotH * lightIntensity, _Glossiness * _Glossiness);

// The smooth step function is used to toonify the reflection and multiply the final output by the user-defined value specular colour.
float specularIntensitySmooth = smoothstep(0.005, 0.01, specularIntensity);
float4 specular = specularIntensitySmooth * _SpecularColour;

// Calculate the rim by taking the dot product of the normal and the view direction and inverting it.
float4 rimDot = 1 - dot(viewDir, normal);

// Toonifyy the thresholding the value with smoothstep. Only display rim light on illuminated surfaces of the object.
float rimIntensity = rimDot * pow(NdotL, _RimThreshold); // The power function scales the rim.
 rimIntensity = smoothstep(_RimAmount - 0.01, _RimAmount + 0.01, rimIntensity);
float4 rim = rimIntensity * _RimColour;

float4 sample = tex2D(_MainTex, i.uv);

// Ambient colour is added to ensure that all surfaces receive light uniformly
// Specular reflection models the individual, distinct reflections made by light sources.
// Specular reflection is view-dependent in that it is affected by the angle at which the surface is viewed.
// Rim lighting is the addition of illumination to the edges of an object to simulate reflected light or backlighting.
// Rim lighting is especially useful for toon shaders to help the object's silhouette stand out among the flat shaded surfaces.
return _Colour * sample * (_AmbientColour + lightIntensity + specular + rim);
```

References

[1] https://medium.com/@deshankalupahana/shaders-in-unity-lambert-and-ambient-ceba9fab6cfa
[2] https://medium.com/@deshankalupahana/shaders-in-unity-specular-ec19de1043ef
[3] https://roystan.net/articles/toon-shader/