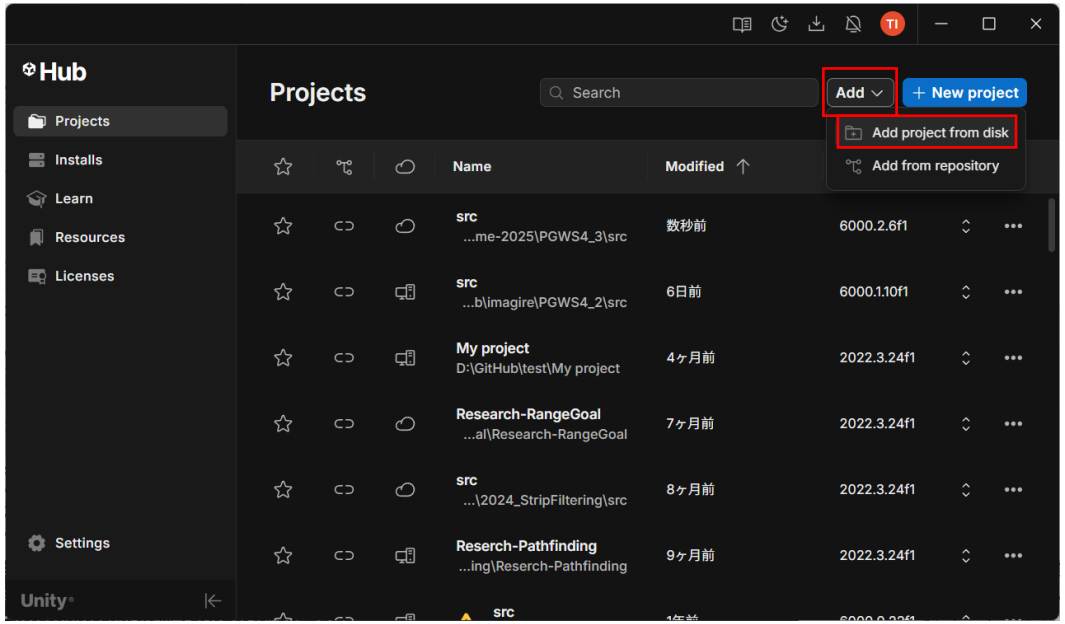
表面反射

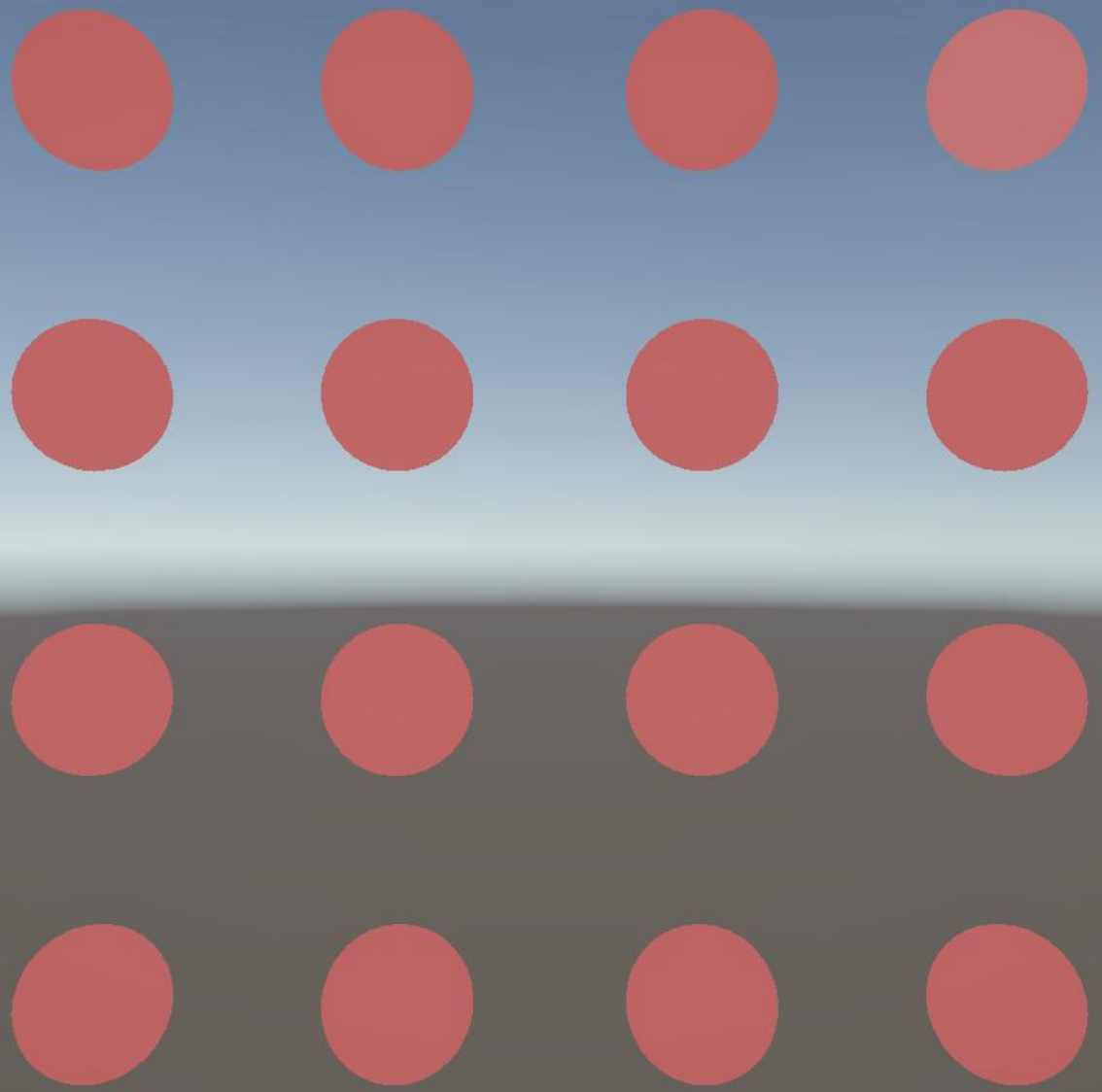
2025年度 プログラムワークショップⅣ (3)

本日の進め方

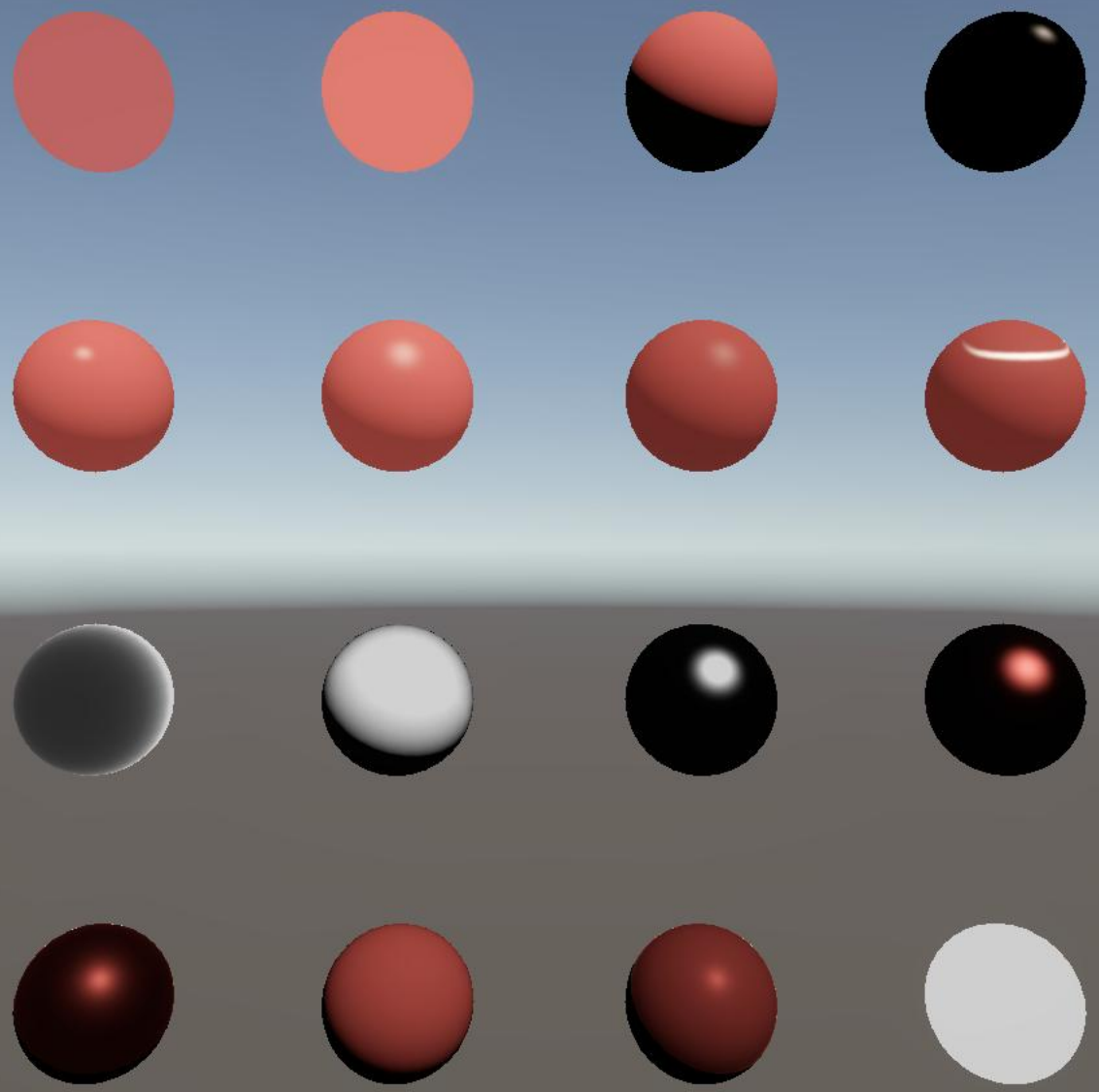
- https://github.com/tpu-game-2025/PGWS4_3
  をfork & Cloneして、「src」フォルダを追加して作業してください



初期状態



ゴール



アジェンダ

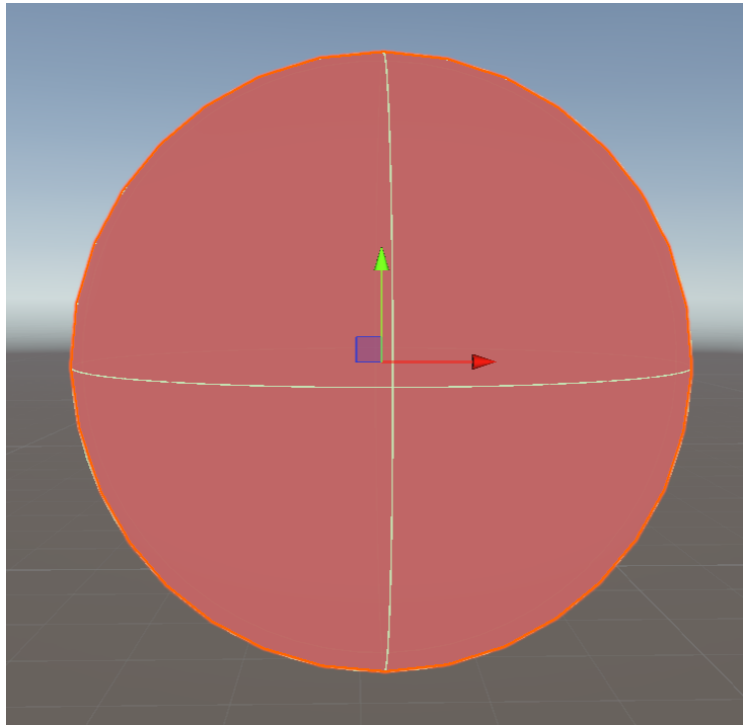
- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

アジェンダ

- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

発光

- 照明計算の視点では、固定色を出力するのは、一定の光が追加されることから、発光(物体が光を出している)と考えられる



Sphere(1)_Emission

固定色の出力:テクスチャの出力を削除

```
3       Properties
4       {
5           [MainColor] _BaseColor("Base Color", Color) = (1, 1, 1, 1)
6           [MainTexture] _BaseMap("Base Map", 2D) = "white"
7       }
```

Inspectorからテクスチャの設定を削除

```
3       Properties
4       {
5           [MainColor] _BaseColor("Base Color", Color) = (1, 1, 1, 1)
6       }
```

```
34          TEXTURE2D(_BaseMap);
35          SAMPLER(sampler_BaseMap);
36
37          CBUFFER_START(UnityPerMaterial)
38              half4 _BaseColor;
39              float4 _BaseMap_ST;
40          CBUFFER_END
```

テクスチャに関連する変数を削除

```
31          CBUFFER_START(UnityPerMaterial)
32              half4 _BaseColor;
33          CBUFFER_END
```

```
50          half4 frag(Varyings IN) : SV_Target
51          {
52              half4 color = SAMPLE_TEXTURE2D(_BaseMap, sampler_BaseMap, IN.uv) * _BaseColor;
53              return color;
54          }
```

テクスチャの読み込みを削除

```
42          half4 frag(Varyings IN) : SV_Target
43          {
44              half4 color = _BaseColor;
45              return color;
46          }
```

テクスチャ座標の伝搬の削除

```
22          struct Attributes
23          {
24              float4 positionOS : POSITION;
25              float2 uv : TEXCOORD0;
26          };
27
28          struct Varyings
29          {
30              float4 positionHCS : SV_POSITION;
31              float2 uv : TEXCOORD0;
32          };
```

使用しない
テクスチャ座標
のメンバーを
構造体から削除

```
21          struct Attributes
22          {
23              float4 positionOS : POSITION;
24          };
25
26          struct Varyings
27          {
28              float4 positionHCS : SV_POSITION;
29          };
```

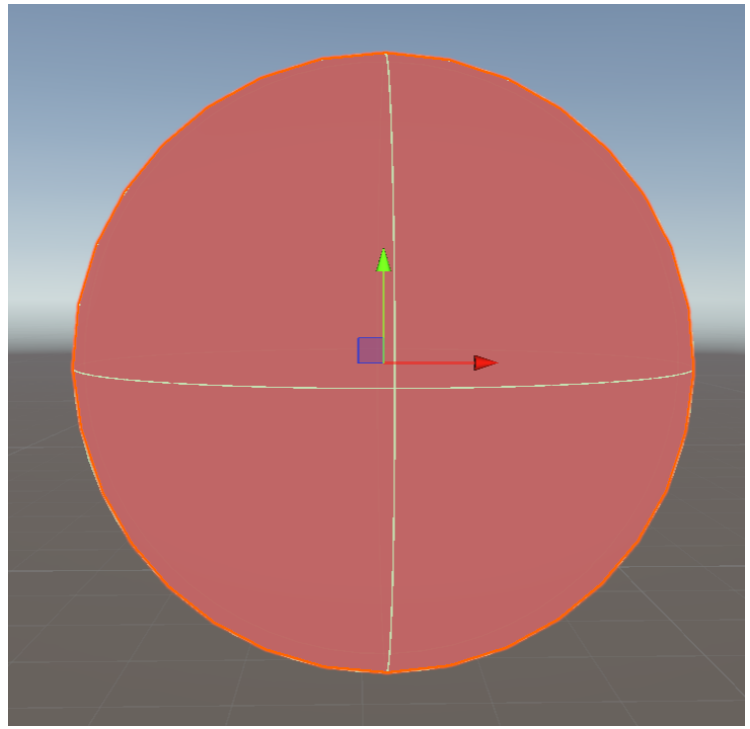
```
42          Varyings vert(Attributes IN)
43          {
44              Varyings OUT;
45              OUT.positionHCS = TransformObjectToHClip(IN.positionOS.xyz);
46              OUT.uv = TRANSFORM_TEX(IN.uv, _BaseMap);
47              return OUT;
48          }
```

ピクセルシェーダで使わないテクスチャ座標
の受け渡しを、頂点シェーダから削除

```
35          Varyings vert(Attributes IN)
36          {
37              Varyings OUT;
38              OUT.positionHCS = TransformObjectToHClip(IN.positionOS.xyz);
39              return OUT;
40          }
```

やってみよう

- Inspectorで色を変えて色を変更できることを確認しよう



Sphere(1)_Emission

アジェンダ

- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

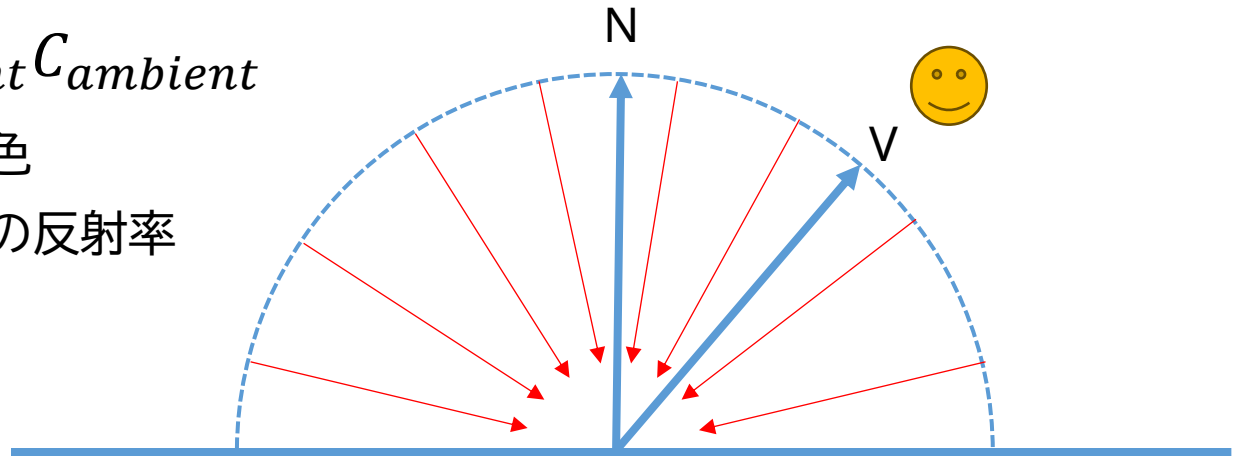
環境光

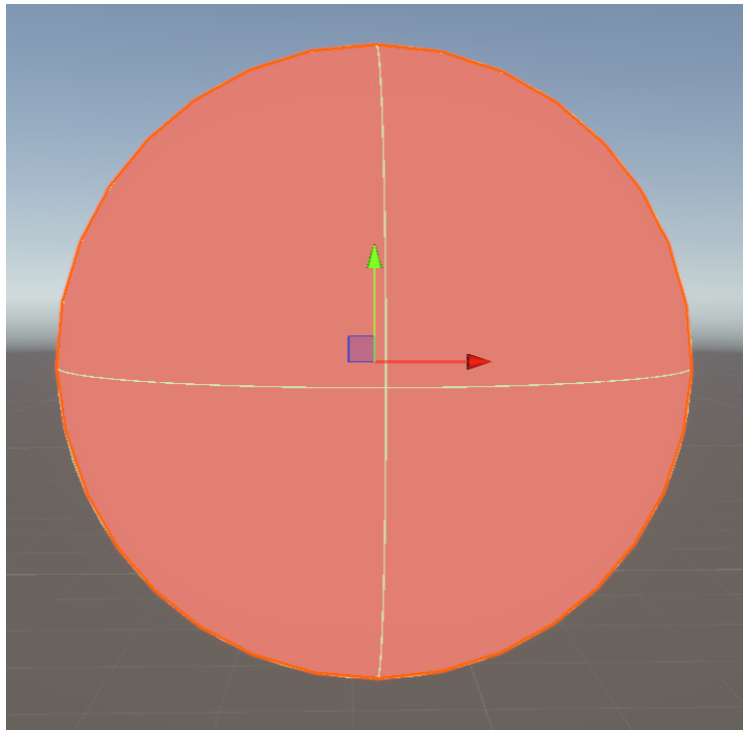
- 光源の光に比例する物体の一様な色
- 簡易的な間接照明
  - 「全方向から同じ強さで（他の物体からの）反射光により照らされている」というモデル
  - また、すべての方向に対して同じ強さで反射する

$$L_{ambient} = C_{light} C_{ambient}$$

C_{light}: 光源の色

$C_{ambient}$: 環境光の反射率





Sphere(2)_AmbientLight

光源情報の取得

- デフォルトで置かれる「Directional Light」オブジェクトの情報

```
19          #include "Packages/com.unity.render-pipelines.universal/ShaderLibrary/Core.hlsl"
20          #include "Packages/com.unity.render-pipelines.universal/ShaderLibrary/Lighting.hlsl"// 追加
```

```
43          half4 frag(Varyings IN) : SV_Target
44          {
45              // 追加
46              Light light = GetMainLight();
```

Use the `GetMainLight` and `GetAdditionalLight` methods to return this struct.

Field	Description
`half3 direction`	The direction of the light.
`half3 color`	The color of the light.
`float distanceAttenuation`	The strength of the light, based on its distance from the object.
`half shadowAttenuation`	The strength of the light, based on whether the object is in shadow.
`uint layerMask`	The **layer mask** of the light.

環境光の計算

- ここでは単に光源の色と固定色「_BaseColor」を乗算する

```
43          half4 frag(Varyings IN) : SV_Target
44          {
45              // 追加
46              Light light = GetMainLight();
47              half3 color = light.color * _BaseColor.rgb;
48              return half4(color, 1);
49          }
```

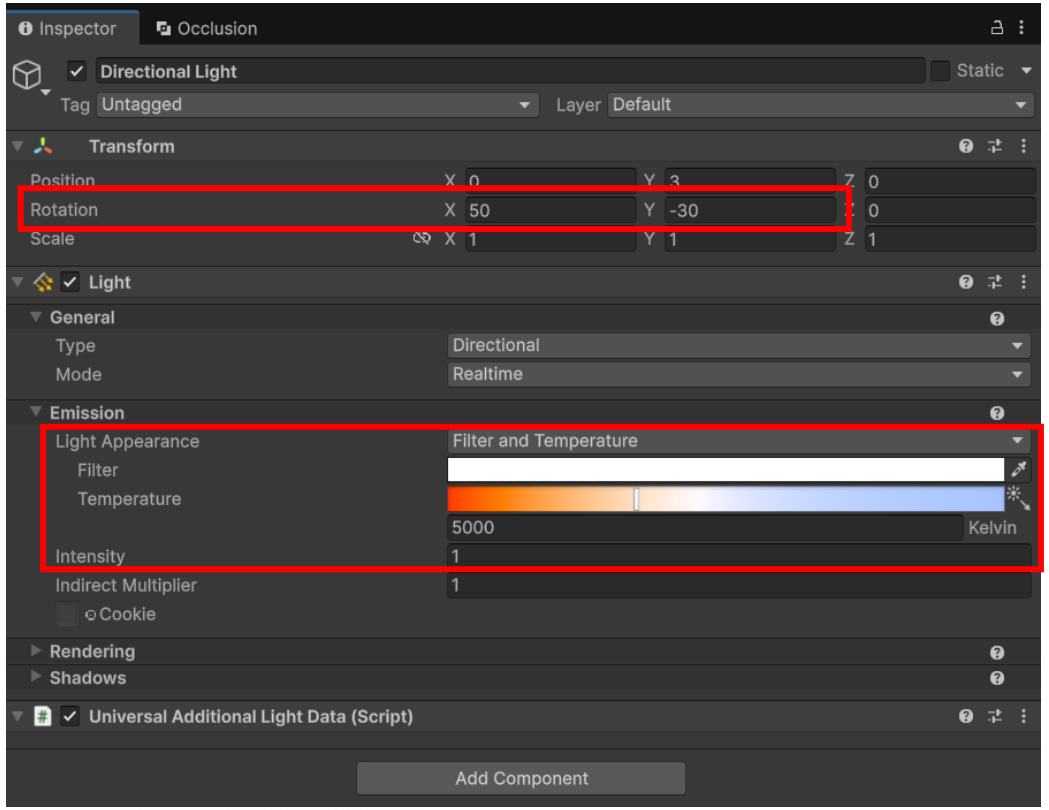
同じ処理

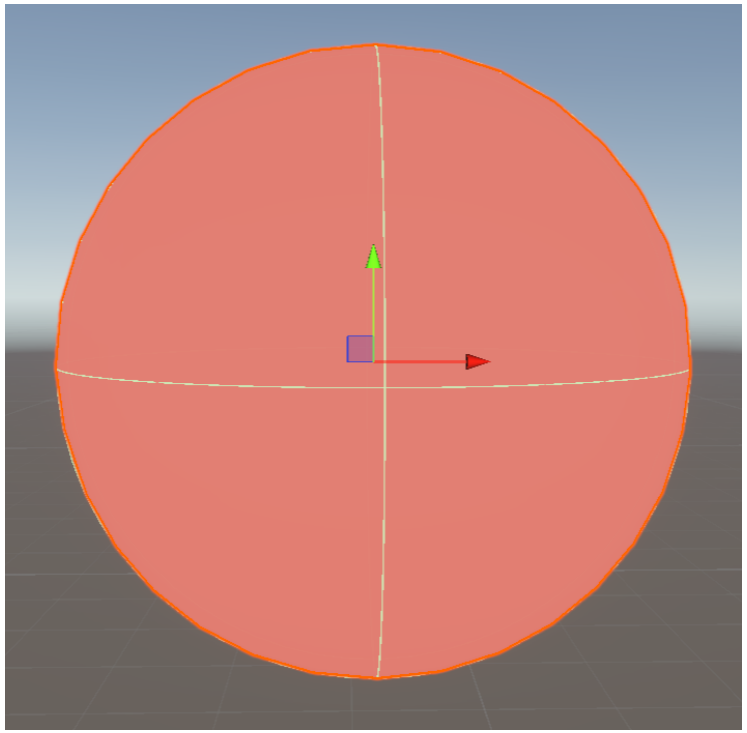
- ベクトルの成分ごとに計算が行われる

```
43          half4 frag(Varyings IN) : SV_Target
44          {
45              // 追加
46              Light light = GetMainLight();
47              half3 color;
48              color.r = light.color.r * _BaseColor.r;
49              color.g = light.color.g * _BaseColor.g;
50              color.b = light.color.b * _BaseColor.b;
51              return half4(color, 1);
52          }
```

やってみよう

- 光の強さ・色を変えた時に物体の色が変わることを確認しよう
  - Emissionのマテリアルはライトの色の影響を受けないことも確認しよう



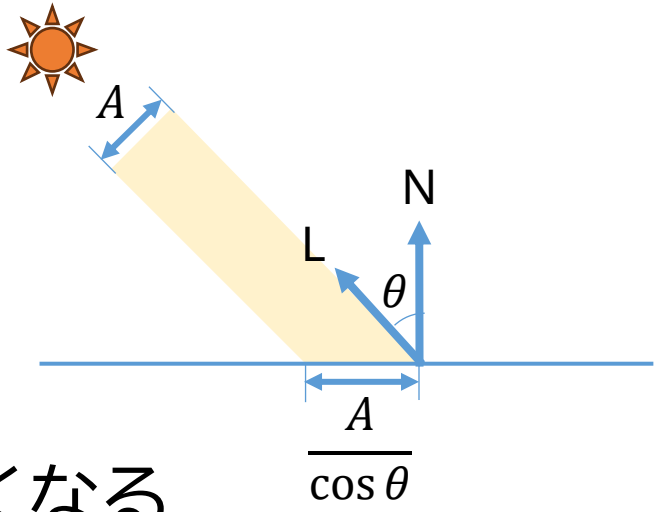


Sphere(2)_AmbientLight

アジェンダ

- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

ランバート拡散反射モデル



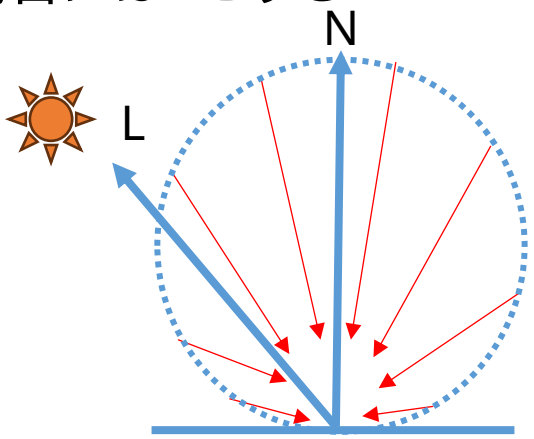
- 表面にあたる光の強さを等方的に反射(ざらざら)
- 光が照らす面積Aは、法線からの傾きθに応じて広くなる
  - 照らす面積: $\frac{A}{\cos\theta}$
  - 面に当る光の強さ: $C_{light}\cos\theta = C_{light}(N \cdot L)$
    - ランバートの余弦則
      - 照らす面積に反比例
    - 余弦が負の(裏から光が当たる)場合には0とする

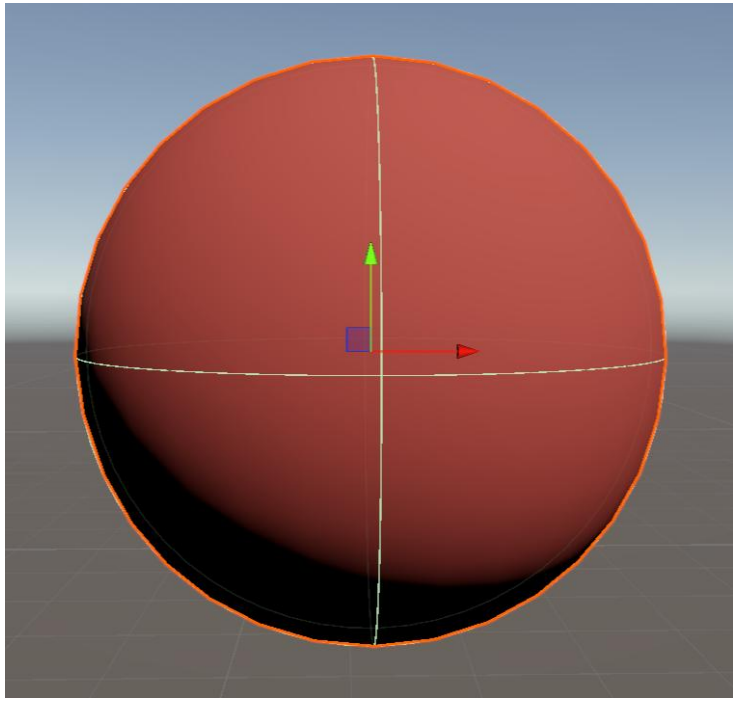
$$L_{lambert} = C_{light} C_{lambert} (N \cdot L)_+$$

C_{light}: 光源の色

$C_{lambert}$: 拡散反射率

$(\cdot)_+$: $max(0,\cdot) = \begin{cases} \cdot & (0 < \cdot) \\ 0 & otherwise \end{cases}$





Sphere(3)_Lambert

法線をピクセルシェーダに渡す

- セマンティクスとして受け取り、渡す
- ワールド空間に変換
  - 光源の向きがワールド空間で渡されるため
  - カメラ空間が望ましい
    - 位置だと精度が落ちる

```
22          struct Attributes
23          {
24              float4 positionOS : POSITION;
25              float3 normal : NORMAL;
26          };
27
28          struct Varyings
29          {
30              float4 positionHCS : SV_POSITION;
31              float3 normal : NORMAL;
32          };
```

```
34          Varyings vert(Attributes IN)
35          {
36              Varyings OUT;
37              OUT.positionHCS = TransformObjectToHClip(IN.positionOS.xyz);
38              OUT.normal = TransformObjectToWorldNormal(IN.normal);
39              return OUT;
40          }
```

環境光の計算

- 法線と光源への向きの余弦を(非負の値に制限して)計算

```
46          half4 frag(Varyings IN) : SV_Target
47          {
48              Light light = GetMainLight();
49              half3 color = light.color * _BaseColor.rgb * max(0, dot(IN.normal, light.direction));
50              return half4(color, 1);
51          }
```

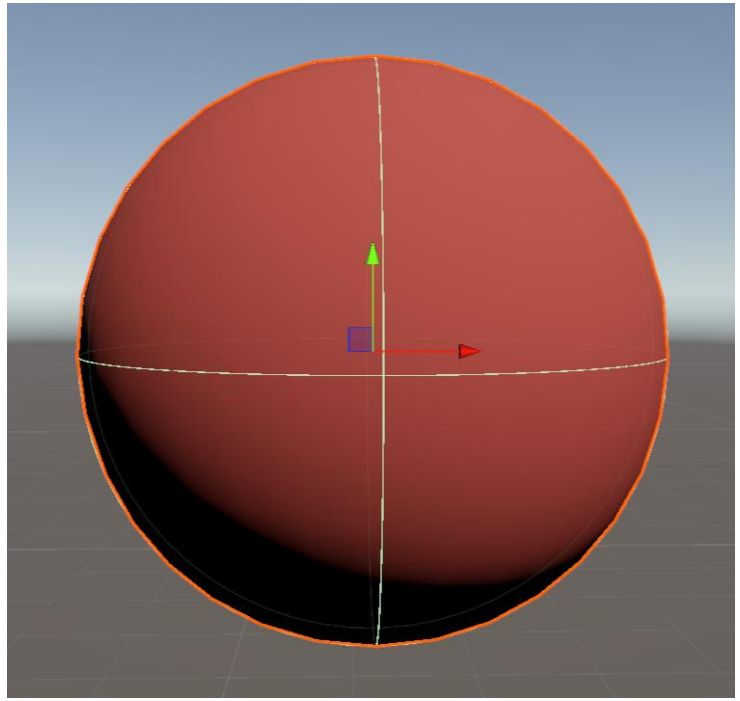
- 同じ計算をする組み込み関数も用意されている
  - 組み込み関数が使えるだけではなく、中身を実装できるようにしておこう

同じ処理

```
46          half4 frag(Varyings IN) : SV_Target
47          {
48              Light light = GetMainLight();
49              half3 color = _BaseColor.rgb * LightingLambert(light.color, light.direction, IN.normal);
50              return half4(color, 1);
51          }
```

やってみよう

- 光の向き・色を変えた時に物体の色が変わることを確認しよう
  - Emissionや環境光のマテリアルはライトの向きの影響を受けないことも確認しよう

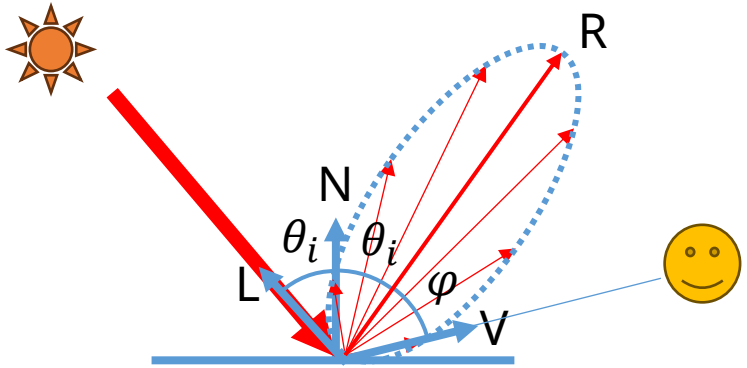


Sphere(3)_Lambert

アジェンダ

- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

(Phongの)鏡面反射光



- てかっているハイライトを入れたい
- 入射方向の正反射方向Rと視線方向Vが近いと明るい
  - 入射方向の正反射方向: $R = reflect(-L, N)$
    - $reflect(i, n) = i - 2\,n\,(i \cdot n)$

$$L_{Phong} = C_{light} C_{Phong} (R \cdot V)^n_+$$

C_{light}: 光源の色

C_{Phong}: 鏡面反射率

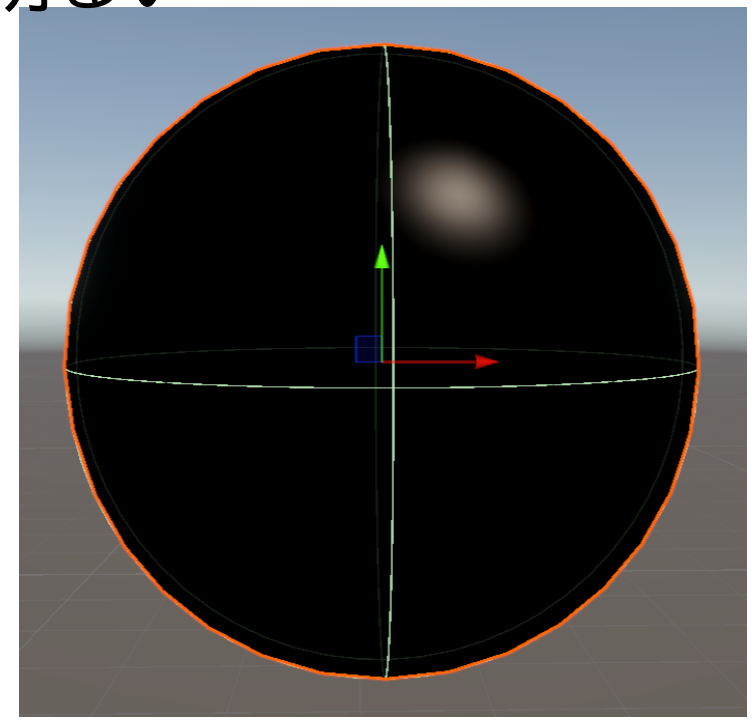
R: 光の正反射方向($R = -L + 2N(L \cdot N)$)

L: ライトベクトル

N: 法線ベクトル

V: ビュー(視線)ベクトル

n: 鏡面反射指数



Sphere(4)_Specular

ビューベクトルの取得

- 位置を渡してカメラの位置との差分から求める
  - カメラ位置はカメラ空間の原点
    - ワールド空間へカメラ原点を変換

```
29          struct Varyings
30          {
31              float4 positionHCS : SV_POSITION;
32              float3 normal : NORMAL;
33              float3 position : TEXCOORD0;
34          };
```

```
41          Varyings vert(Attributes IN)
42          {
43              Varyings OUT;
44              OUT.positionHCS = TransformObjectToHClip(IN.positionOS.xyz);
45              OUT.normal = TransformObjectToWorldNormal(IN.normal);
46              OUT.position = TransformObjectToWorld(IN.positionOS.xyz);
47              return OUT;
48          }
```

```
50          half4 frag(Varyings IN) : SV_Target
51          {
52              Light light = GetMainLight();
53              half3 normal = normalize(IN.normal);// 滑らかにするために正規化し直す
54              half3 view_direction = normalize(TransformViewToWorld(float3(0,0,0)) - IN.position);
```

- 位置を使わない近似も軽量にできるため、よく使われる
  - カメラ空間の正面の向きをワールド空間での方向に変換する
    - 頂点位置ごとの微妙な向きの違いは出せない

近い処理

```
54          half3 view_direction = TransformViewToWorldNormal(float3(0,0,1));// 頂点の位置を見ない近似
```

「Normal」がついている関数は向きを変換(厳密には違う)

パラメータの追加

- 新しいパラメータを追加する
  - Specular Power: 鏡面反射指数
  - Specular Intensity: 鏡面反射係数
    - 塗装などの誘電体は色が乗らないので、今回はスカラーにしてみた

```
3       Properties
4       {
5           _SpecularPower("Specular Power", Range(0.001, 300)) = 80
6           _SpecularIntensity("Specular Intensity", Range(0, 1)) = 0.3
7       }
```

```
36          CBUFFER_START(UnityPerMaterial)
37              half _SpecularPower;
38              half _SpecularIntensity;
39          CBUFFER_END
```

(Phongの)鏡面反射光

- Phongの鏡面反射光の計算に従って鏡面反射を計算

$$L_{Phong} = C_{light} C_{Phong} (R \cdot V)_{+}^{n}$$

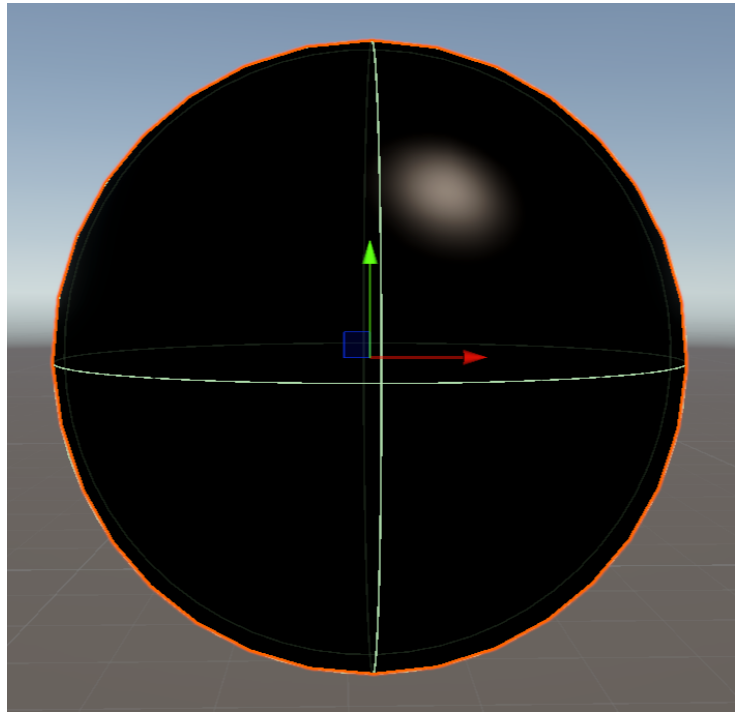
  - 55行目：反射ベクトルの算出
  - 58行目：鏡面反射率の計算
  - 60行目：光源の色を反映

```
50              half4 frag(Varyings IN) : SV_Target
51              {
52                  Light light = GetMainLight();
53                  half3 normal = normalize(IN.normal);// 滑らかにするために正規化し直す
54                  half3 view_direction = normalize(TransformViewToWorld(float3(0,0,0)) - IN.position);
55                  float3 reflected_direction = -light.direction + 2 * normal * dot(light.direction, normal);
56  //                float3 reflected_direction = reflect(-light.direction, normal);// reflect関数を使う場合はこちら
57
58                  half3 specular = _SpecularIntensity * pow(max(0, dot(reflected_direction, view_direction)), _SpecularPower);
59
60                  half3 color = light.color * specular;
61                  return half4(color, 1);
62              }
```

地味に書き換えています(そのままだと汚い)

やってみよう

- 光の向き、視点を変えた時に物体の色が変わることを確認しよう
  - Emissionや環境光、拡散光のマテリアルは視線の向きの影響を受けないことも確認しよう

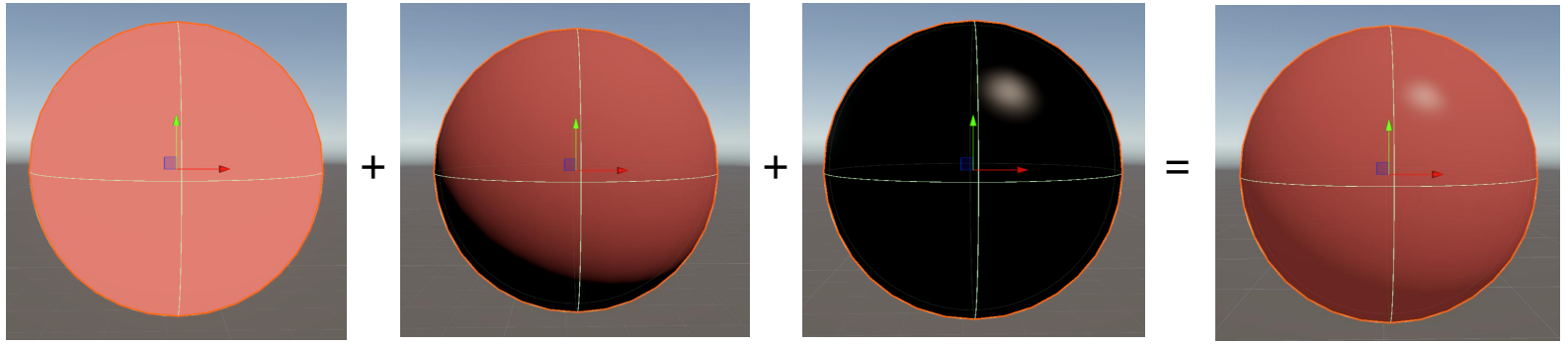


Sphere(4)_Specular

アジェンダ

- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

Phongモデル



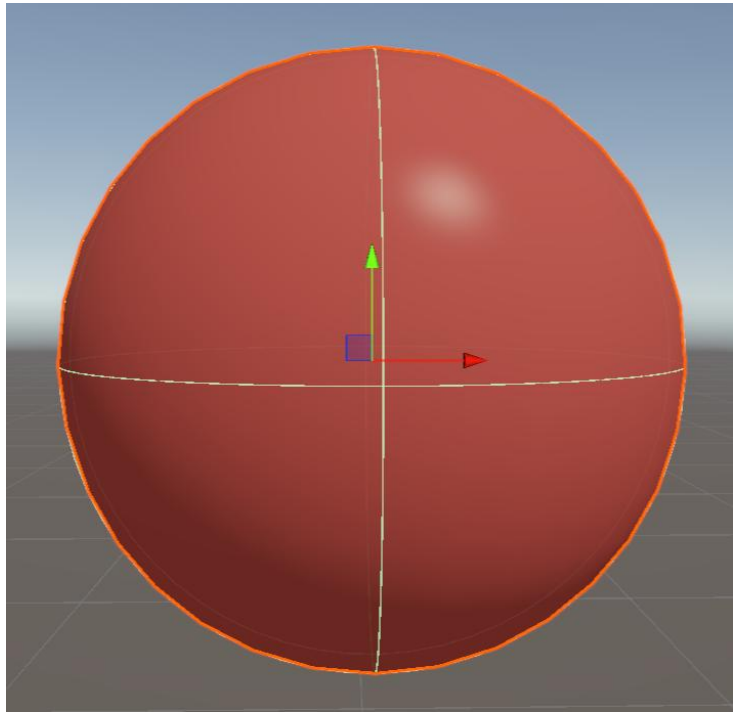
- Phongの鏡面反射とLambert拡散と環境光の合算

$$L_{PhongModel} = L_{ambient} + L_{lambert} + L_{Phong}$$

$$L_{ambient} = C_{light} C_{ambient}$$
$$L_{lambert} = C_{light} C_{lambert} (N \cdot L)_+$$
$$L_{Phong} = C_{light} C_{Phong} (R \cdot V)_+^n$$

- 拡散反射と環境光の色味が同じことが多いのでアルベド(物体の白さ)として統合する事も多い

$$L_{PhongModel} = C_{light}(C_{albedo}(k_{ambient} + k_{lambert}(N \cdot L)_+) + C_{Phong}(R \cdot V)_+^n)$$



C_{albedo}: アルベド(物体の白さ), $k_{ambient}$: 環境光の係数, $k_{Lambert}$: 拡散反射光の係数

Sphere(5)_PhongModel

頂点シェーダ

- 鏡面反射光と同じ
  - 入出力のデータも変更なし

```
25          struct Attributes
26          {
27              float4 positionOS : POSITION;
28              float3 normal : NORMAL;
29          };
30
31          struct Varyings
32          {
33              float4 positionHCS : SV_POSITION;
34              float3 normal : NORMAL;
35              float3 position : TEXCOORD0;
36          };
```

```
45          Varyings vert(Attributes IN)
46          {
47              Varyings OUT;
48              OUT.positionHCS = TransformObjectToHClip(IN.positionOS.xyz);
49              OUT.normal = TransformObjectToWorldNormal(IN.normal);
50              OUT.position = TransformObjectToWorld(IN.positionOS.xyz);
51              return OUT;
52          }
```

ピクセルシェーダ

```
3       Properties
4       {
5           [MainColor] _BaseColor("Base Color", Color) = (1, 1, 1, 1)
6           _AmbientRate("Ambient Rate", Range(0, 1)) = 0.2
7           _SpecularPower("Specular Power", Range(0.001, 300)) = 80
8           _SpecularIntensity("Specular Intensity", Range(0, 1)) = 0.3
9       }
```

```
38      CBUFFER_START(UnityPerMaterial)
39          half4 _BaseColor;
40          half _AmbientRate;
41          half _SpecularPower;
42          half _SpecularIntensity;
43      CBUFFER_END
```

- 明るくなりすぎないように環境光と拡散光の強さを内挿計算で調整

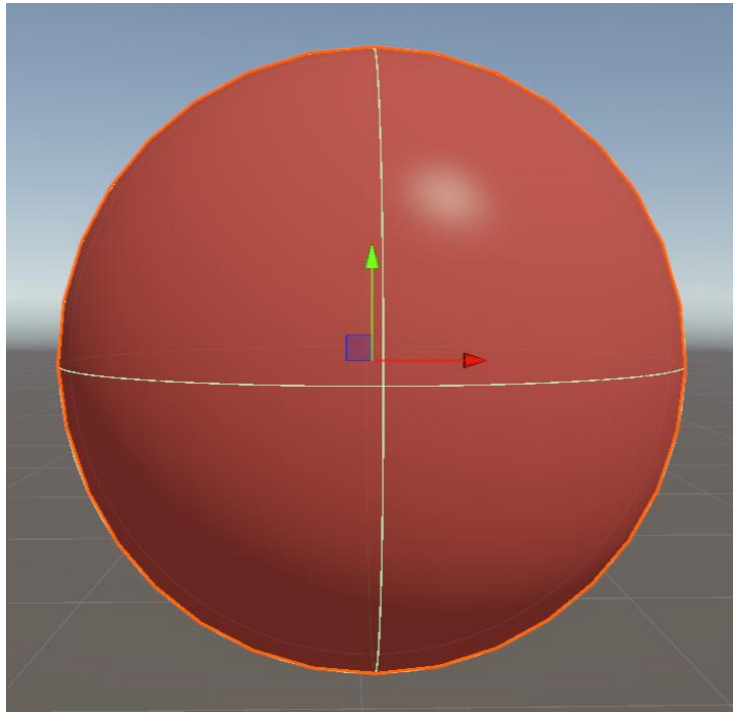
$$C_{ambient} + C_{lambert}(N \cdot L)_+ = C_{albedo}(k_{ambient} + k_{lambert}(N \cdot L)_+)$$
$$= C_{albedo}k_{ambient} + C_{albedo}(1 - k_{ambient})(N \cdot L)_+$$

$$k_{lambert} = 1 - k_{ambient}$$

```
54      half4 frag(Varyings IN) : SV_Target
55      {
56          Light light = GetMainLight();
57          half3 normal = normalize(IN.normal);
58          half3 view_direction = normalize(TransformViewToWorld(float3(0,0,0)) - IN.position);
59          float3 reflected_direction = -light.direction + 2 * normal * dot(light.direction, normal);
60
61          half3 ambient = _BaseColor.rgb;
62          half3 lambert = _BaseColor.rgb * max(0, dot(IN.normal, light.direction));
63          half3 specular = _SpecularIntensity * pow(max(0, dot(reflected_direction, view_direction)), _SpecularPower);
64
65          half3 color = light.color * (specular + lerp(lambert, ambient, _AmbientRate));
66          return half4(color, 1);
67      }
```

やってみよう

- 光の向き、視点を変えた時に物体の色が変わることを確認しよう
  - それぞれのパラメータがどのように影響しているのか確認しよう



Sphere(5)_PhongModel

アジェンダ

- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

レンダリング方程式

- 表面反射の一般化（羊羹のような透け感のある質感は対象外）

$$L_{output}(x;\omega_o) = L_{emissive}(x;\omega_o) + \int_{\Omega_+} d\omega\, f_r(x;\omega \to \omega_o)\cos(N(x),\omega)L(\omega)$$

$$= L_{emissive}(x;\omega_o) + \int_{\Omega} d\omega\, f_r(x;\omega \to \omega_o)(N(x)\cdot\omega)_+ L(\omega)$$

発光　BRDF　Lambert余弦則

BRDF (Bidirectional Reflectance Distribution Function): 双方向反射率分布関数
表面上の各点xにおける、入射方向ωから反射方向ω_oへの光の反射率

物理的なBRDF

- 物理的なBRDFと呼ばれるBRDFは次の特性を持つ
  - 正値性：$0 \leq f_r(x; \omega \rightarrow \omega_o)$
  - ヘルムホルツの相反性：$f_r(x; \omega \rightarrow \omega_o) = f_r(x; \omega_o \rightarrow \omega)$
    - 物理法則の時間反転普遍性
  - エネルギーの保存：$\forall \omega_o, \int_\Omega \ d\omega \ f_r(x; \omega \rightarrow \omega_o)(N(x) \cdot \omega)_+ \leq 1$

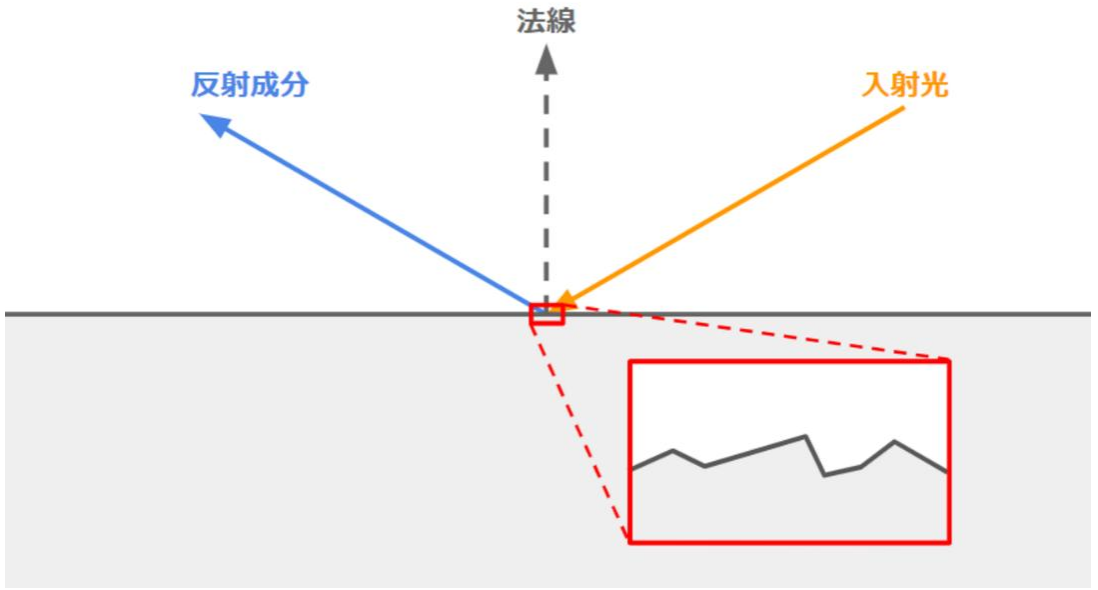
- 注：ただし、これらの特性を満たしていても物理的にありえないBRDFは存在する
  - 上記は、あくまでも物理法則の一部

アジェンダ

- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

マイクロファセットモデル

- この特性を考慮に入れて反射モデルは発展を続けてきた
- マイクロファセットモデル
  - 表面を微視的にみると、小さな鏡面の集まりで構成されている
  - 各鏡面では完全鏡面反射（入射した光は、正反射方向にしか反射しない）
  - 物理的な制約は、マイクロファセットの法線分布の制約などに入ってくる



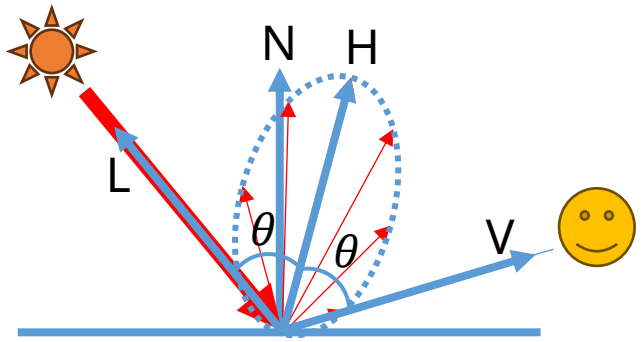
Blinn - Phongの鏡面反射モデル

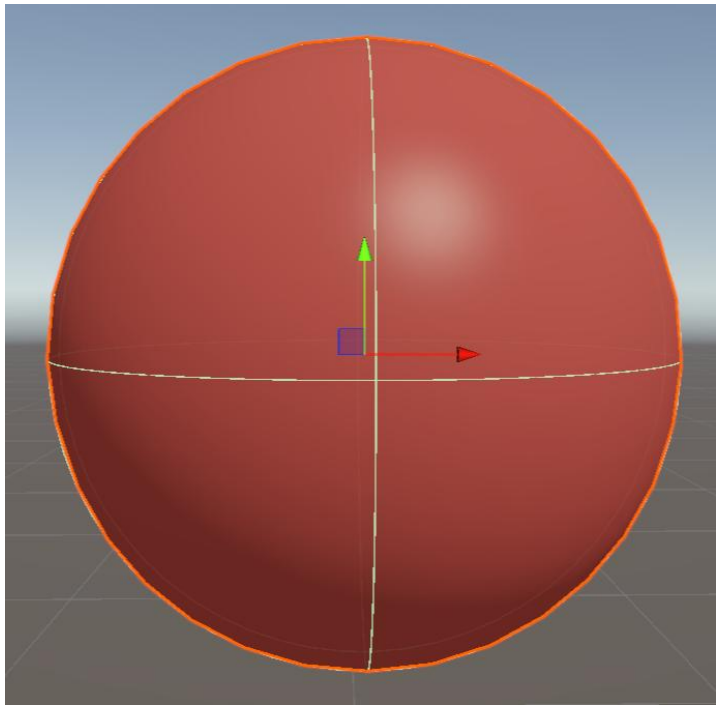
- ハイライトが生じるのは、マイクロファットモデルで、マイクロファセットがいろいろな方向を向いているから
  - いろいろな方向を向いた鏡面の反射の平均が通常みている反射
    - 視線と光源方向の中間が法線の向きと一致していれば完全反射の状況になっているはず（反射が強い状況）

$$L_{BlinnPhong} = C_{light} C_{BlinnPhong} (H \cdot N)_{+}^{n},$$

$H = \frac{L+V}{[L+V]}$: ハーフベクトル

定義からLとVを入れ替えても不変





Sphere(6)_Blinn_Phong

ハーフベクトルを使ったシェーディング

- ハーフベクトルは$L + V$をその大きさで割るのではなく、$L + V$の正規化で導出

```
54              half4 frag(Varyings IN) : SV_Target
55              {
56                  Light light = GetMainLight();
57                  half3 normal = normalize(IN.normal);
58                  half3 view_direction = normalize(TransformViewToWorld(float3(0,0,0)) - IN.position);
59                  float3 half_vector = normalize(view_direction + light.direction);
60                  half HdotN = max(0, dot(half_vector, normal));
61
62                  half3 ambient = _BaseColor.rgb;
63                  half3 lambert =  _BaseColor.rgb * max(0, dot(light.direction, normal));
64                  half3 specular = _SpecularIntensity * pow(HdotN, _SpecularPower);
65
66                  half3 color = light.color * (specular + lerp(lambert, ambient, _AmbientRate));
67                  return half4(color, 1);
68              }
```

組み込み関数：LightingSpecular

- Unityでは、組み込み関数があり、細かな鏡面反射計算を省ける

```
59          float3 half_vector = normalize(view_direction + light.direction);
60          half LdotN = max(0, dot(light.direction, normal));
61          half HdotN = max(0, dot(half_vector, normal));
62
63          half3 ambient = _BaseColor.rgb;
64          half3 lambert = _BaseColor.rgb * LdotN;
65          half3 specular = _SpecularIntensity * pow(HdotN, _SpecularPower);
```

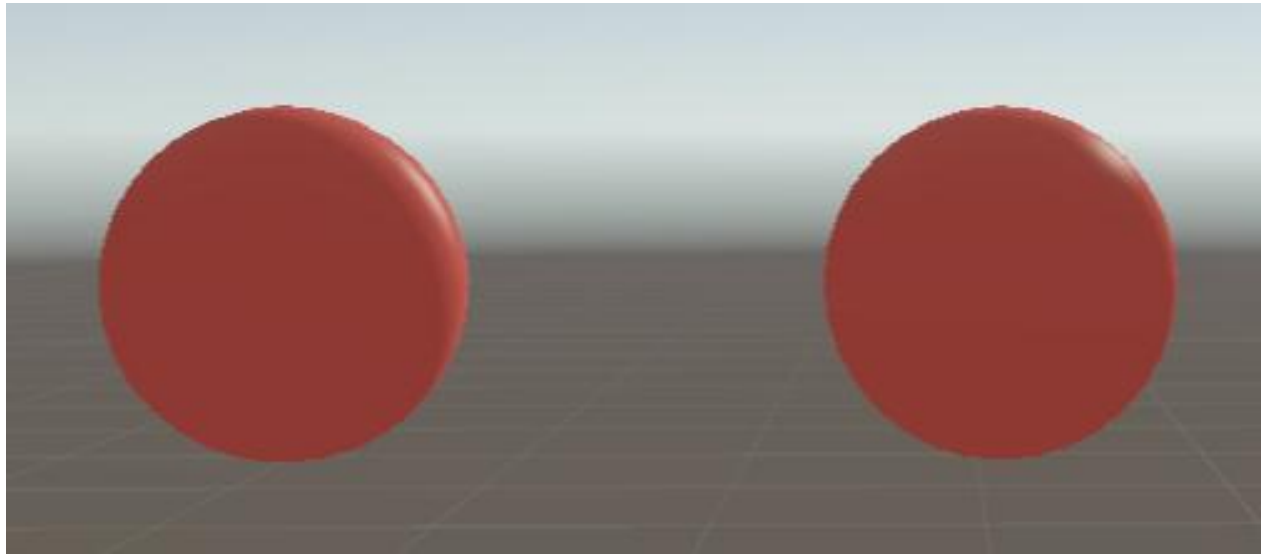
同じ処理

```
64          half3 specular = LightingSpecular(1, light.direction, normal, view_direction,
65              _SpecularIntensity, _SpecularPower);
66
67          half3 color = light.color * (specular + lerp(lambert, ambient, _AmbientRate));
```

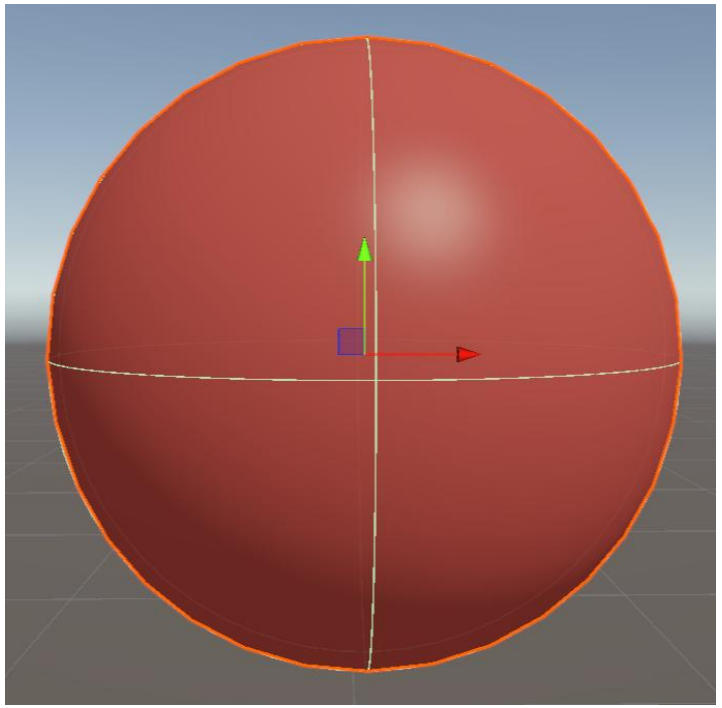
光源の色を入れてまとめることもできる

やってみよう

- 光の向き、視点を変えた時に物体の色が変わることを確認しよう
  - PhongとBlinn-Phongで光の向きを変えた時に、ハイライトがどのように変化するか確認しよう



Phong　　Blinn-Phong



Sphere(6)_Blinn_Phong

アジェンダ

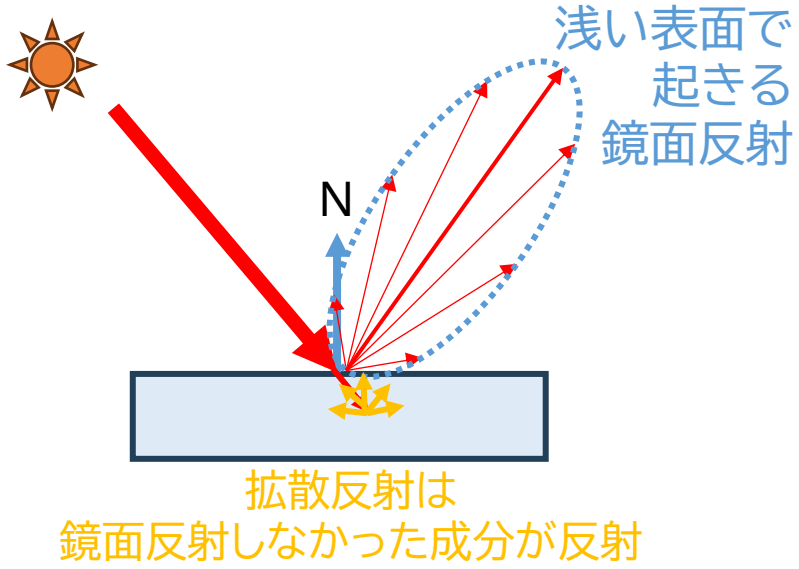
- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

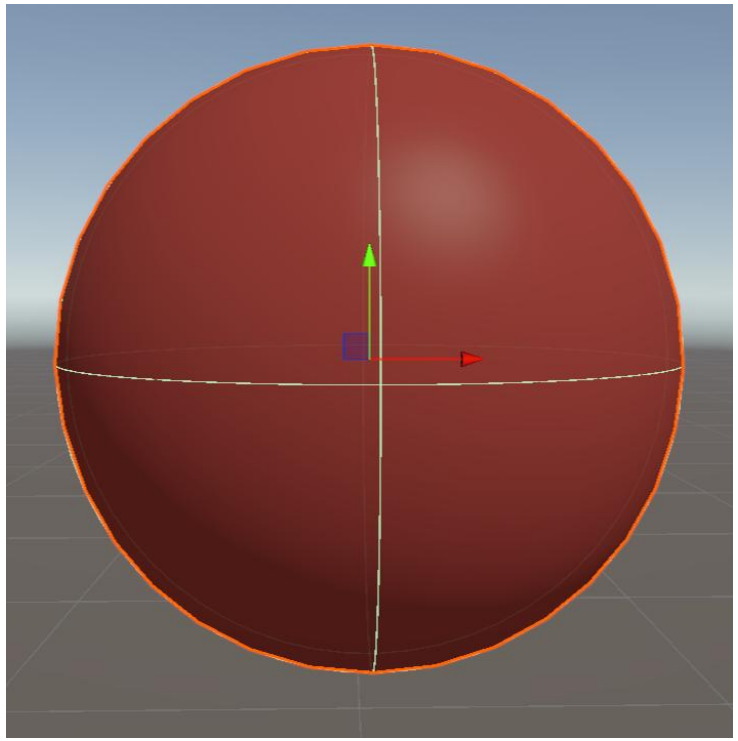
メタリック

- 拡散反射(と環境光)は鏡面反射していない成分の光による反射
  - 鏡面反射があるなら、その分、拡散反射は減るべき

$$L = (1 - k_{Metallic})(L_{lambert} + L_{ambient}) + k_{Metallic} L_{specular}$$
$$= lerp(L_{lambert} + L_{ambient}, L_{specular}, k_{Metallic})$$

金属反射でなくても「メタリック」
というパラメータで呼ばれる





Sphere(7)_Metallic

パラメータ追加

- 変数の追加
  - Properties, 定数バッファ
- パラメータの繁栄
  - ピクセルシェーダ

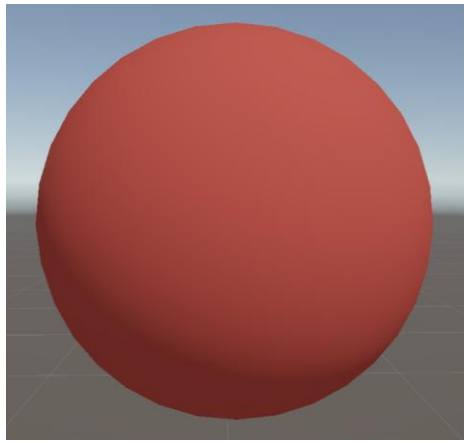
```
3       Properties
4       {
5           [MainColor] _BaseColor("Base Color", Color) = (0.86, 0.39, 0.39, 1)
6           _AmbientRate("Ambient Rate", Range(0, 1)) = 0.2
7           _SpecularPower("Specular Power", Range(0.001, 300)) = 80
8           _SpecularIntensity("Specular Intensity", Range(0, 1)) = 0.3
9           _Metallic("Metallic", Range(0, 1)) = 0.5
10      }
```

```
39          CBUFFER_START(UnityPerMaterial)
40              half4 _BaseColor;
41              half _AmbientRate;
42              half _SpecularPower;
43              half _SpecularIntensity;
44              half _Metallic;
45          CBUFFER_END
```

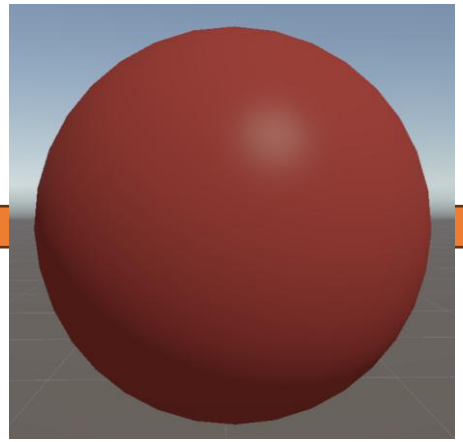
```
56          half4 frag(Varyings IN) : SV_Target
57          {
58              Light light = GetMainLight();
59              half3 normal = normalize(IN.normal);
60              half3 view_direction = normalize(TransformViewToWorld(float3(0,0,0)) - IN.position);
61              float3 half_vector = normalize(view_direction + light.direction);
62              half HdotN = max(0, dot(half_vector, normal));
63
64              half3 ambient = _BaseColor.rgb;
65              half3 lambert = _BaseColor.rgb * max(0, dot(light.direction, normal));
66              half3 specular = _SpecularIntensity * pow(HdotN, _SpecularPower);
67
68              half3 color = light.color *
69                  lerp(lerp(lambert, ambient, _AmbientRate), specular, _Metallic);
70              return half4(color, 1);
71          }
```

やってみよう

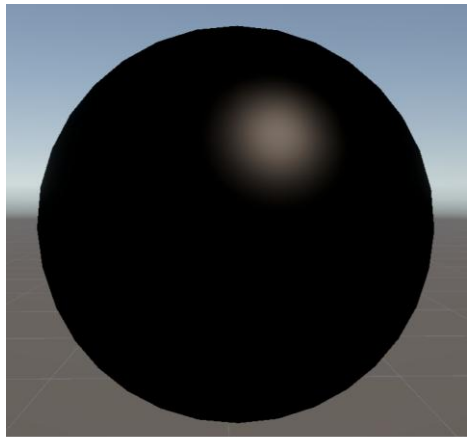
- メタリックパラメータを変えた時に質感が変わることを確認しよう



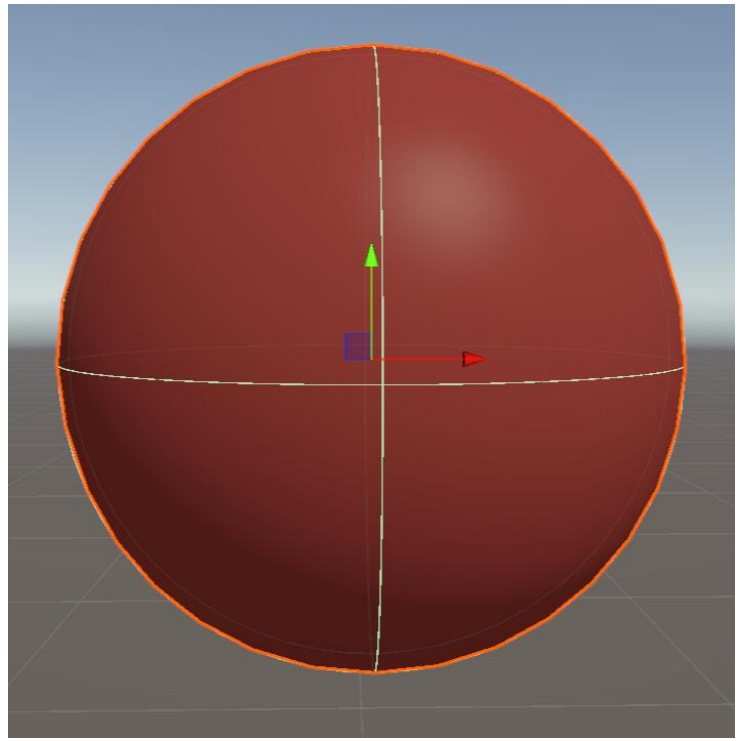
0%メタリック



50%メタリック



100%メタリック



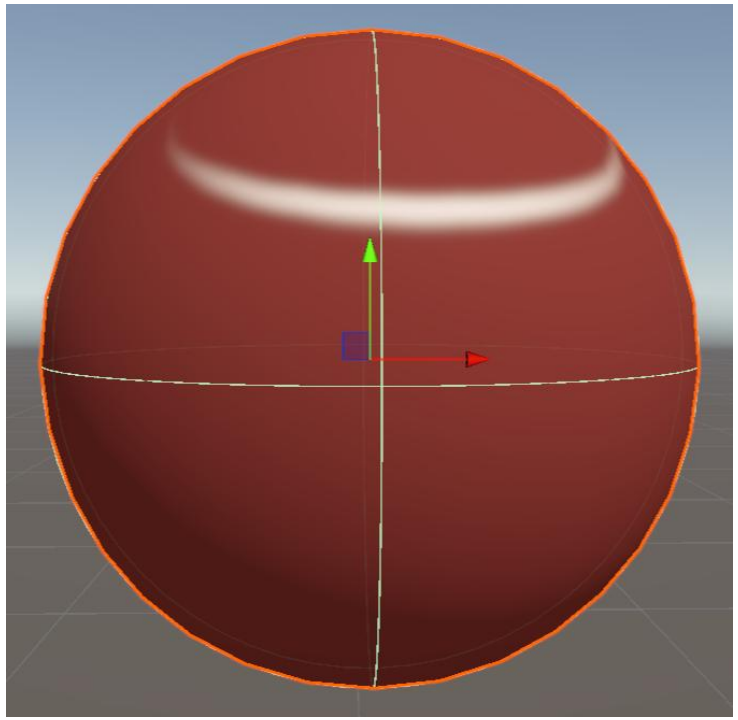
Sphere(7)_Metallic

アジェンダ

- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

異方性反射モデル

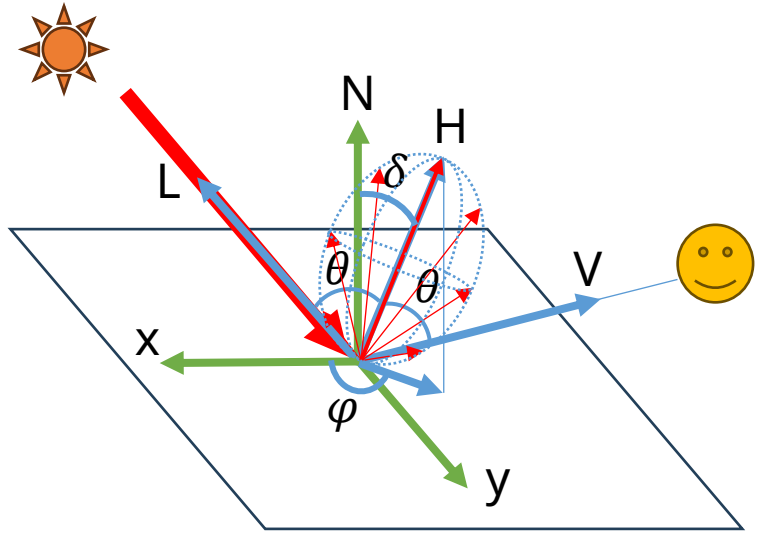
- ハイライトは円形とは限らない
  - 特異な構造をしていると、ハイライトは一方向に伸びる
- 異方性反射：
  - 反射光の成分が入射方向の接線成分にも依存する



Sphere(8)_Anisotropic

image by freeimageslive.co.uk - gratuit
https://pxhere.com/ja/photo/1403200

Walterモデル



- 接方向x, yに対して、独立して粗さを設定し接方向ごとに鏡面反射の粗さを制御する

$$f_r = \rho_s \frac{1}{\sqrt{(L \cdot N)(V \cdot N)}} \frac{e^c}{4\pi\alpha_x\alpha_y}$$

$$c = -\frac{1}{(H \cdot N)^2}\left\{\left(\frac{H \cdot x}{\alpha_x}\right)^2 + \left(\frac{H \cdot y}{\alpha_y}\right)^2\right\}$$

ρ_s: 鏡面反射係数

α_x, α_y: 粗さのパラメータ

- 異方性反射モデルは他にも提案されているが、実装の容易さから今回はWalterモデルを採用した

Walter, B. Notes on the Word BRDF, 2005.

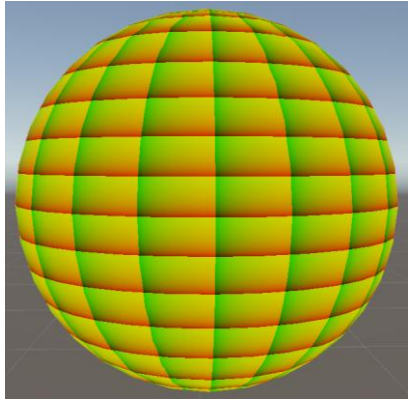
接ベクトル

- 接方向としてテクスチャ座標が増える向きがよく使われる
- 法線ベクトルと同じようにテクスチャ座標の増える向きを取得できる
  - 従法線ベクトル: binormal
    - U(テクスチャ座標のx成分：赤成分)方向の減る方向
  - 接ベクトル: tangent
    - V(テクスチャ座標のy成分：緑成分)方向の減る方向

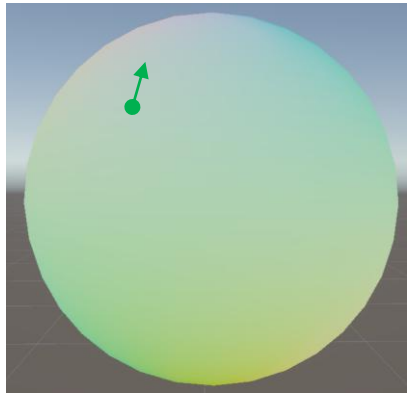
向きはUnityが定めた取り方で、別のツール・環境だと異なりうる

注：これらは、DirectXのセマンティクスに端を発した(リアルタイム)CG特有の命名であり、通常の数学の使われ方とは違う

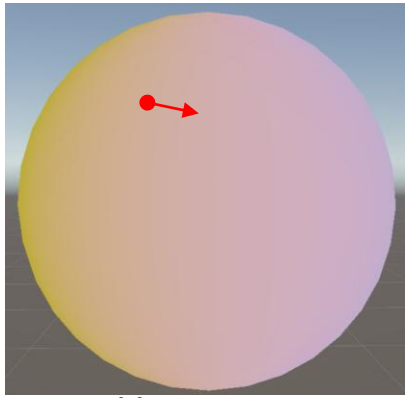
- 通常は、接ベクトルも従法線ベクトルも共に接ベクトルと呼ばれる



テクスチャ座標
（20倍して小数部を取得）



従法線ベクトル



接ベクトル

接ベクトルの取得

- 接ベクトルと従法線ベクトルと法線ベクトルは直交しているので2つがわかれば残りの1つは外積で計算できる
- Unityでは、APIの右手系と左手系の差を吸収するために、接ベクトルを4次元に拡張し、従法線ベクトルの向きの符号をw成分に格納しているらしい

```
27          struct Attributes
28          {
29              float4 positionOS : POSITION;
30              float3 normal : NORMAL;
31              float4 tangent : TANGENT;
32          };
33
34          struct Varyings
35          {
36              float4 positionHCS : SV_POSITION;
37              float3 normal : NORMAL;
38              float4 tangent : TANGENT;
39              float3 position : TEXCOORD0;
40          };
```

```
51          Varyings vert(Attributes IN)
52          {
53              Varyings OUT;
54              OUT.positionHCS = TransformObjectToHClip(IN.positionOS.xyz);
55              OUT.normal = TransformObjectToWorldNormal(IN.normal);
56              OUT.tangent = float4(TransformObjectToWorldNormal(float3(IN.tangent.xyz)).xyz, IN.tangent.w);
57              OUT.position = TransformObjectToWorld(IN.positionOS.xyz);
58              return OUT;
59          }
```

接ベクトルについて法線ベクトルの座標変換を使うのは間違い。
ただ、異方性の拡大縮小でしか差が出ないので、手抜きをする

```
61          half4 frag(Varyings IN) : SV_Target
62          {
63              half3 normal = normalize(IN.normal);// 滑らかにするために正規化し直す
64              half3 binormal = normalize(cross(normal, IN.tangent.xyz) * IN.tangent.w);
65              half3 tangent = cross(binormal, normal) * IN.tangent.w;
```

パラメータの追加

- 接ベクトル方向と従法線ベクトル方向の粗さのパラメータを追加

```
 3        Properties
 4        {
 5            [MainColor] _BaseColor("Base Color", Color) = (0.86, 0.39, 0.39, 1)
 6            _SpecularIntensity("Specular Intensity", Range(0, 1)) = 0.3
 7            _AmbientRate("Ambient Rate", Range(0, 1)) = 0.2
 8            _RoughnessX("Roughness X", Range(0, 1)) = 0.8
 9            _RoughnessY("Roughness Y", Range(0, 1)) = 0.2
10            _Metallic("Metallic", Range(0, 1)) = 0.5
11        }
```

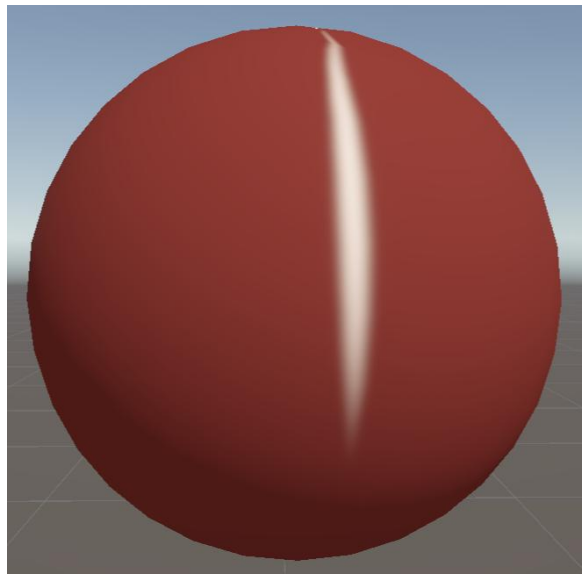
```
42            CBUFFER_START(UnityPerMaterial)
43                half4 _BaseColor;
44                half _SpecularIntensity;
45                half _AmbientRate;
46                half _RoughnessX;
47                half _RoughnessY;
48                half _Metallic;
49            CBUFFER_END
```

ピクセルシェーダ

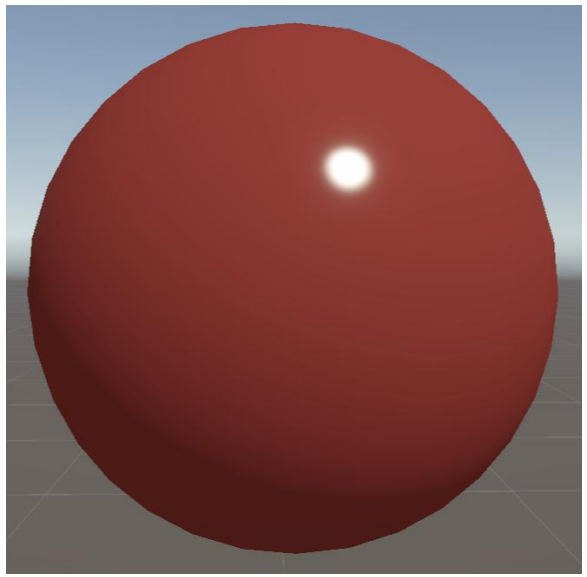
```
61          half4 frag(Varyings IN) : SV_Target
62          {
63              half3 normal = normalize(IN.normal);// 滑らかにするために正規化し直す
64              half3 binormal = normalize(cross(normal, IN.tangent.xyz) * IN.tangent.w);
65              half3 tangent = cross(binormal, normal) * IN.tangent.w;
66
67              Light light = GetMainLight();
68              half3 view_direction = normalize(TransformViewToWorld(float3(0,0,0)) - IN.position);
69              half3 half_vector = normalize(light.direction + view_direction);
70              half VdotN = max(0.000001, dot(view_direction, normal));
71              half LdotN = max(0.000001, dot(light.direction, normal));  発散しないように0になるのを抑える
72              half HdotN = max(0.000001, dot(half_vector, normal));
73
74              half alphaX = _RoughnessX * _RoughnessX; 他の照明モデルに合わせて、粗さ(roughness)の
75              half alphaY = _RoughnessY * _RoughnessY; 自乗αx,αyを鋭さのパラメータに採用
76              half XdotH = dot(tangent, half_vector);
77              half YdotH = dot(binormal, half_vector);
78
79              half3 ambient = _BaseColor.rgb;
80              half3 lambert = _BaseColor.rgb * LdotN; 拡散反射光もLdotNを使って簡潔に表現
81              half c = (XdotH*XdotH/(alphaX*alphaX) + YdotH*YdotH/(alphaY*alphaY))/(HdotN * HdotN);
82              half3 specular = _SpecularIntensity* exp(-c) / sqrt(LdotN * VdotN) / (4 * PI * alphaX * alphaY);
83
84              half3 color = light.color *
85                  lerp(lerp(lambert, ambient, _AmbientRate), specular, _Metallic);
86              return half4(color, 1);
87          }
```

やってみよう

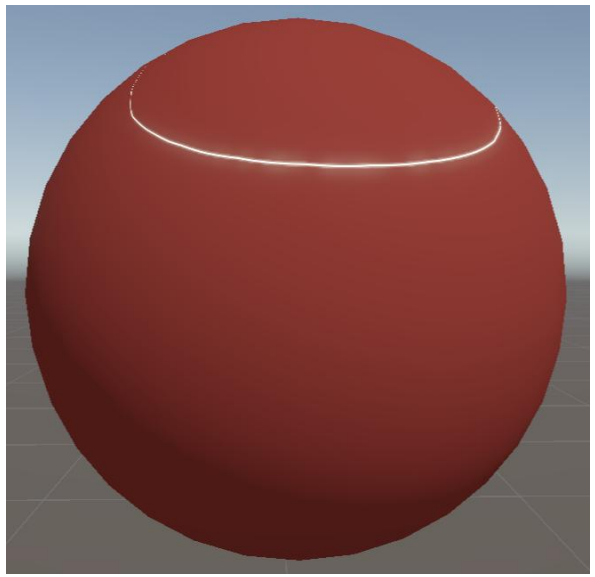
- 各軸方向の粗さを変えた時に質感が変わることを確認しよう



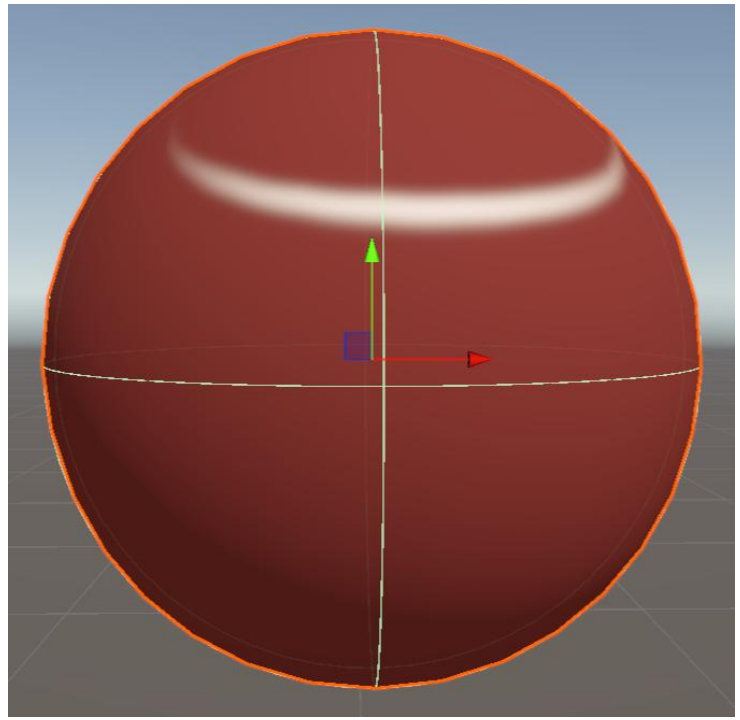
Roughness X：0.2
Roughness Y：0.8



Roughness X：0.2
Roughness Y：0.2



Roughness X：0.99
Roughness Y：0.04



Sphere(8)_Anisotropic

アジェンダ

- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

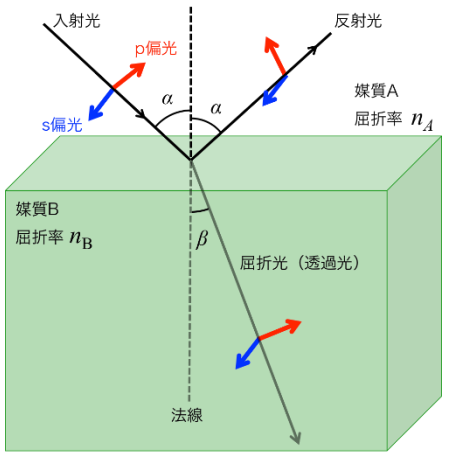
フレネル

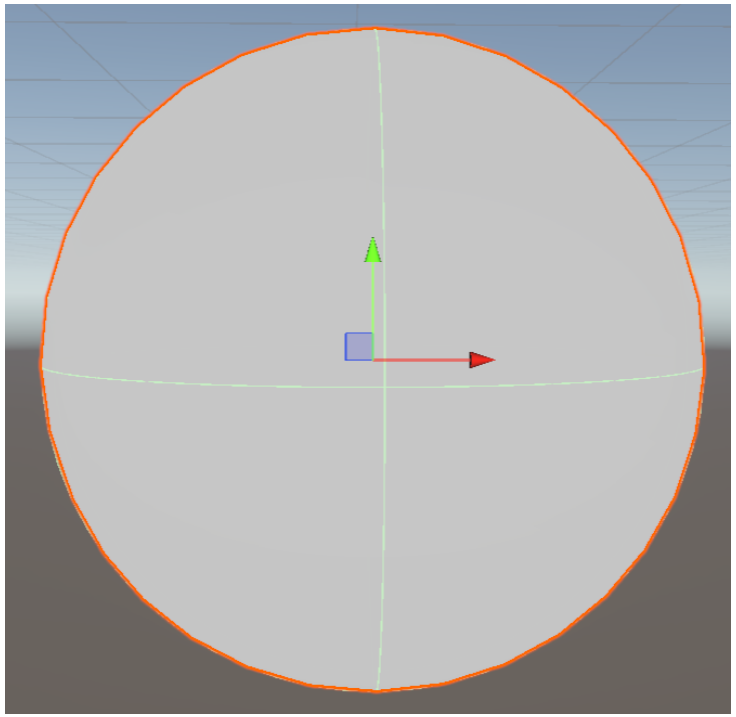
- 表面の反射率は材質で決まる
  - 電磁気学のマックスウェル方程式に従う
  - 周波数ごとに反射率は異なる
    - 金属は鏡面反射に色がつく(正面反射率は大きい)
    - 誘電体は無色で十分な場合が多い(正面反射率は小さい)
- フレネルの式:反射・透過率の関係
- ゲームではSchlick近似が有名

$$F_{\text{Schlick}} = F_0 + (1 - F_0)(1 - V \cdot H)^5$$

- Far Cry 3で使われた手法も良い

$$F_{FarCry3} = F_0 + (1 - F_0)e^{-6V \cdot H}$$





Sphere(9)_Fresnel

https://physics.thick.jp/Experimental_Physics/Section1/1-7.html

Schlick近似の実装

- パラメータの追加
- ピクセルシェーダ

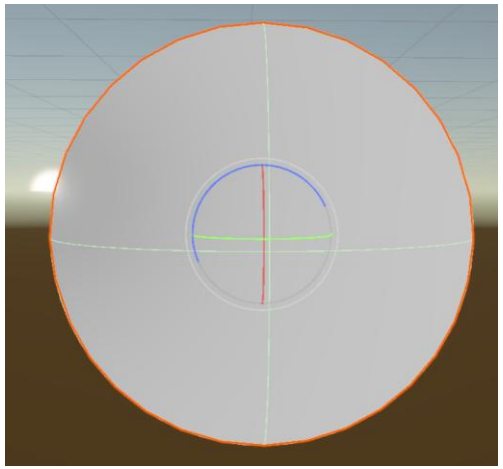
```
3       Properties
4       {
5           _Fresnel0("Fresnel0", Range(0, 0.99999)) = 0.8
6       }
```

```
35          CBUFFER_START(UnityPerMaterial)
36              half _Fresnel0;
37          CBUFFER_END
```

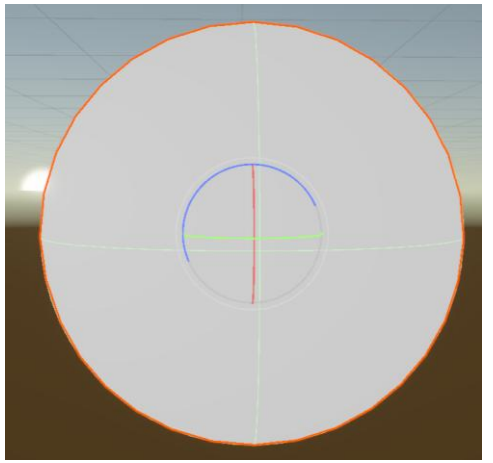
```
63      half4 frag(Varyings IN) : SV_Target
64      {
65          Light light = GetMainLight();
66          half3 normal = normalize(IN.normal);
67          half3 view_direction = normalize(TransformViewToWorld(float3(0,0,0)) - IN.position);
68          half3 half_vector = normalize(light.direction + view_direction);
69          half VdotH = max(0, dot(view_direction, half_vector));
70
71          half F = _Fresnel0 + (1-_Fresnel0) * pow(1 - VdotH, 5);// Schlick近似
72
73          half3 color = F;
74          return half4(color, 1);
75      }
```

他の実装への差し替えの挑戦

- 近似なし(偏光は考慮せず)
  - 光源が正対する場合に違いあり(落ち込む効果が表れる)



近似なし



Schlick近似

```
48              half FresnelReflectanceAverageDielectric(float co, float f0)
49              {
50                  float root_f0 = sqrt(f0);
51                  float n = (1 + root_f0) / (1 - root_f0);
52                  float n2 = n * n;
53
54                  float si2 = 1 - co * co;
55                  float nb = sqrt(n2 - si2);
56                  float bn = nb / n2;
57
58                  float r_s = (co - nb) / (co + nb);
59                  float r_p = (co - bn) / (co + bn);
60                  return 0.5 * (r_s * r_s + r_p * r_p);
61              }
```

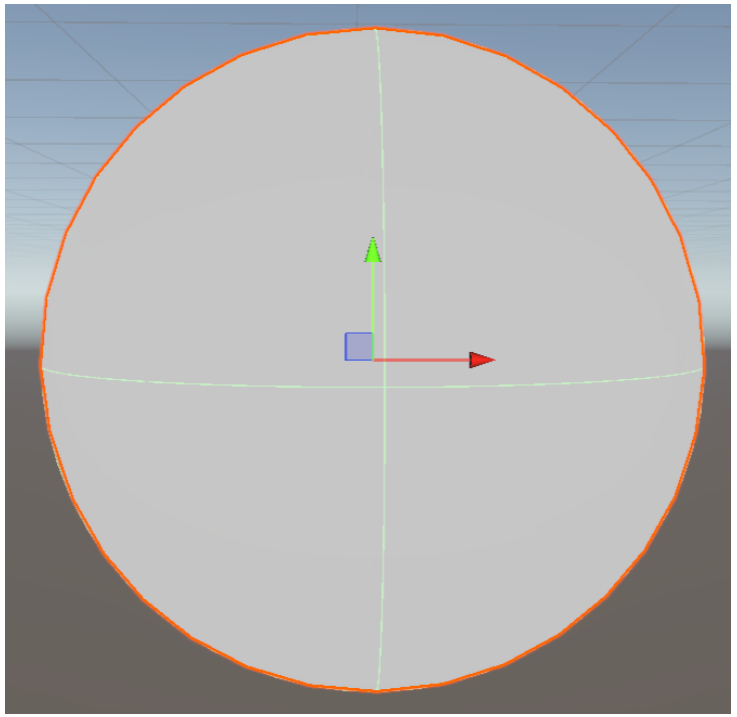
```
72                  half F = FresnelReflectanceAverageDielectric(VdotH, _Fresnel0);// S波P波平均の無近似
```

- Far Cry 3 近似

```
74                      half F = _Fresnel0 + (1-_Fresnel0) * exp(-6 * VdotH);// FarCry3近似
```

やってみよう

- 正面反射率や光源方向を変えた時の明るさの違いを確認しよう
  - 実際に使われるフレネル値(Linear)
    - プラスチック:0.03-0.05
    - ガラス: 0.03-0.08
    - ダイアモンド:0.17
    - 金属: (挑戦:色成分ごとに反射率を変えてみよう)
      - 金: (1.00, 0.71, 0.29)
      - 鉄: (0.56, 0.57, 0.58)
      - アルミニウム: (0.92, 0.92, 0.92)

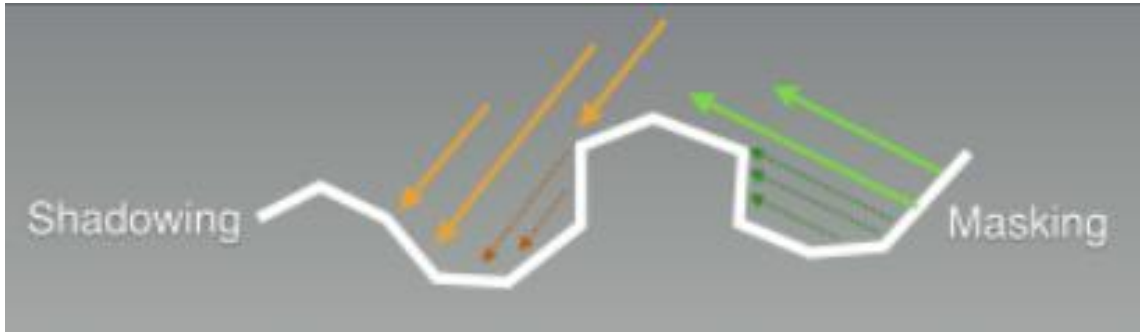


Sphere(9)_Fresnel

アジェンダ

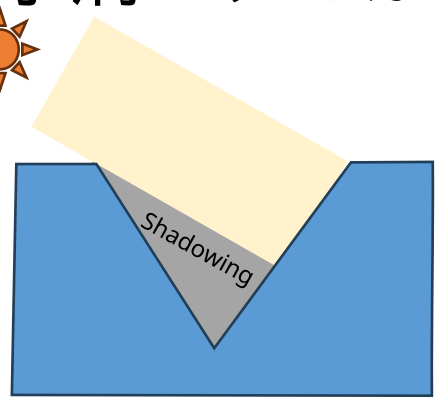
- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

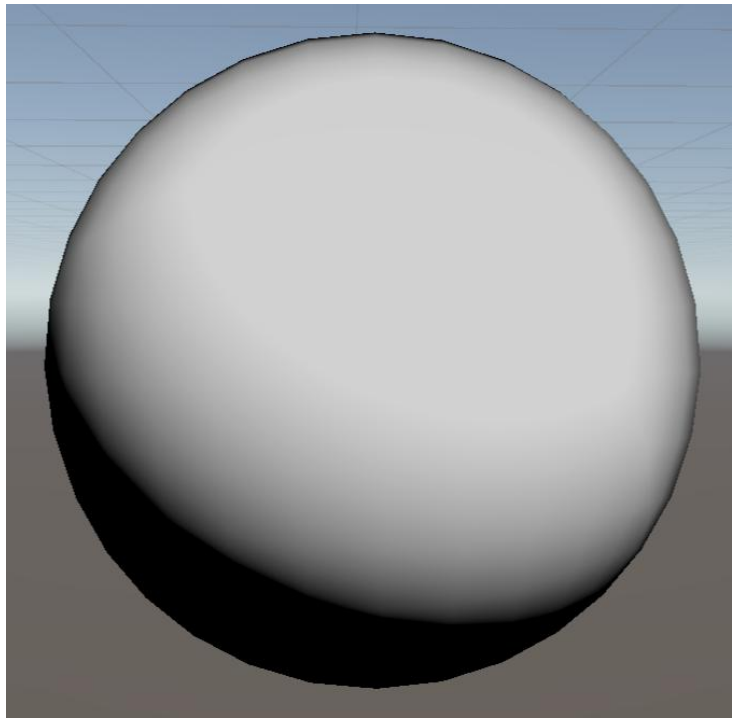
幾何減衰項



- マイクロファセットの反射光が他のマイクロファセットの影響により遮られる
  - シャドウィング：ほかのマイクロファセットに遮られて入射光が届かない
  - マスキング：ほかのマイクロファセットに反射放出光が遮られる
- これらの効果を考慮するためにV字溝モデルが提案された
  - 今回は導出はしない

$$G = \min\left\{1, \frac{2(N\cdot H)(V\cdot N)}{V\cdot H}, \frac{2(N\cdot H)(L\cdot N)}{L\cdot H}\right\}$$





Sphere(10)_GeometryAttenuation

W. McDermott, “The Comprehensive PBR Guide by Allegorithmic.”

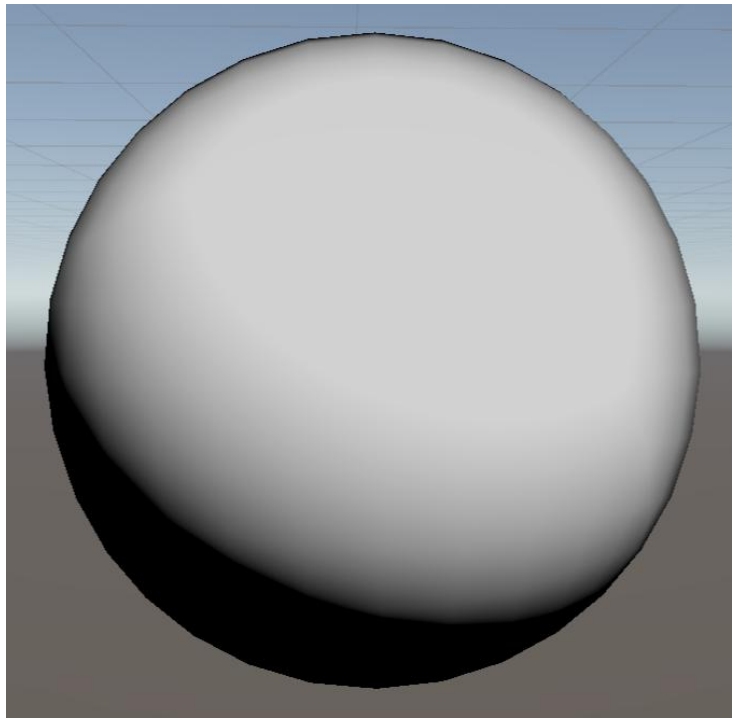
実装

- ピクセルシェーダ
  - 自乗の計算でpowを使うのは重いので、同じものを掛けている

```
48              half4 frag(Varyings IN) : SV_Target
49              {
50                  Light light = GetMainLight();
51                  half3 normal = normalize(IN.normal);
52                  half3 view_direction = normalize(TransformViewToWorld(float3(0,0,0)) - IN.position)
53                  half3 half_vector = normalize(light.direction + view_direction);
54                  half VdotN = max(0, dot(view_direction, normal));
55                  half LdotN = max(0, dot(light.direction, normal));
56                  half HdotN = max(0, dot(half_vector, normal));
57                  half LdotH = max(0, dot(half_vector, light.direction));
58                  half VdotH = max(0, dot(half_vector, view_direction));
59
60                  half G = min(1, 2 * min(HdotN * VdotN / VdotH, HdotN * LdotN / LdotH));
61                  half3 color = G;
62                  return half4(color, 1);
63              }
```

やってみよう

- 光源方向を変えた時の明るさの違いを確認しよう



Sphere(10)_GeometryAttenuation

アジェンダ

- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

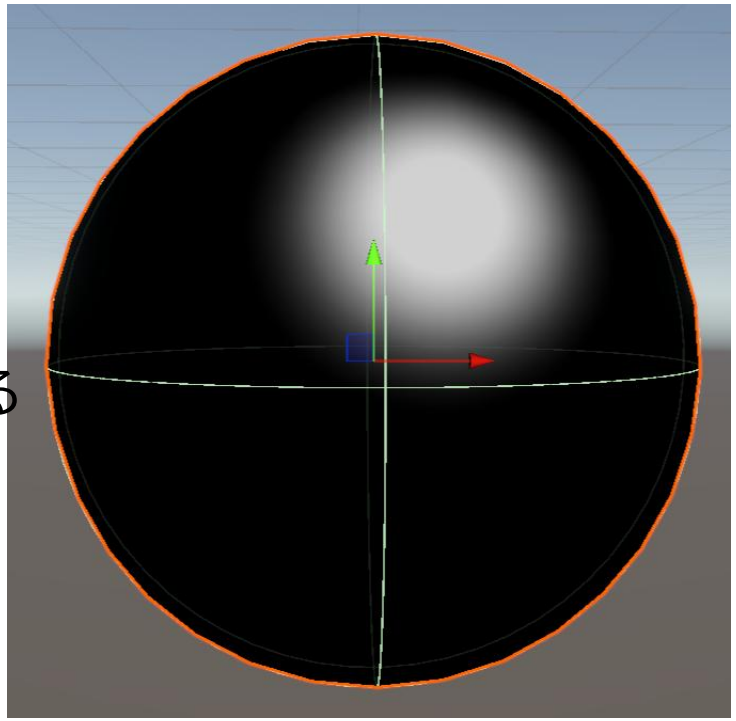
法線分布関数

- 各方向に対する、マイクロファセットが向いている割合
- Beckmann分布関数
  - 金属反射を再現するために使われた分布関数

$$D = \frac{1}{\alpha^2 (H \cdot N)^4} e^{-\left(\frac{\tan(\arccos(H \cdot N))}{\alpha}\right)^2}$$

  - α:向きの平均自乗
    - 大きい:マイクロファセットの向きがバラバラになり粗くなる
    - 小さい:マイクロファセットの向きがそろい鋭いハイライト
- BRDFへの適用

$$f_r = \frac{FDG}{4(N \cdot L)(N \cdot V)}$$



Sphere(11)_DistributionFunction

パラメータの追加

- 表面の粗さ

```
3       Properties
4       {
5           [MainColor] _BaseColor("Base Color", Color) = (0.86, 0.39, 0.39, 1)
6           _Roughness("Roughness", Range(0, 1)) = 0.4
7       }
```

```
36          CBUFFER_START(UnityPerMaterial)
37              half4 _BaseColor;
38              half _Roughness;
39          CBUFFER_END
```

ピクセルシェーダ

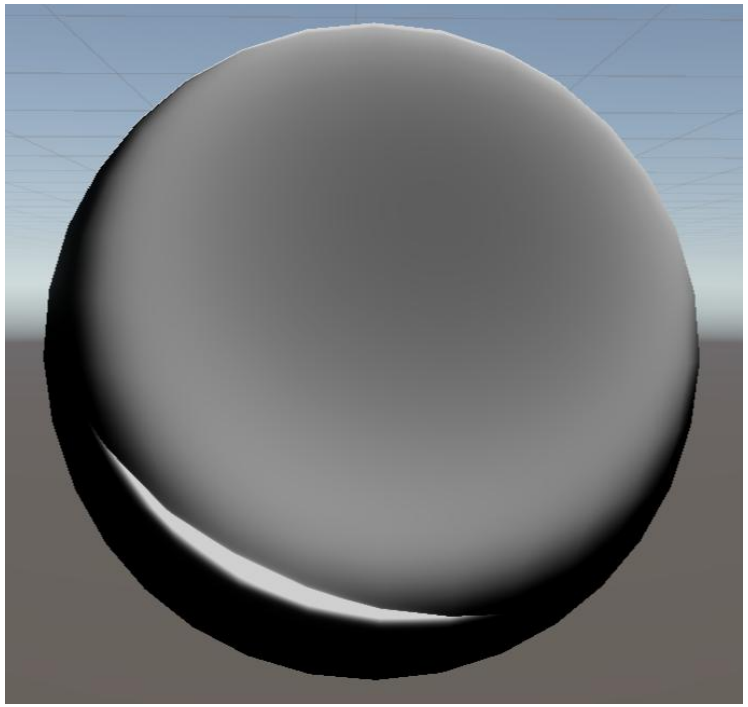
Expの指数の計算に次の式変形を使った

$$\tan^2(\arccos(H\cdot N)) = \frac{\sin^2(\arccos(H\cdot N))}{\cos^2(\arccos(H\cdot N))} = \frac{1-(H\cdot N)^2}{(H\cdot N)^2}$$

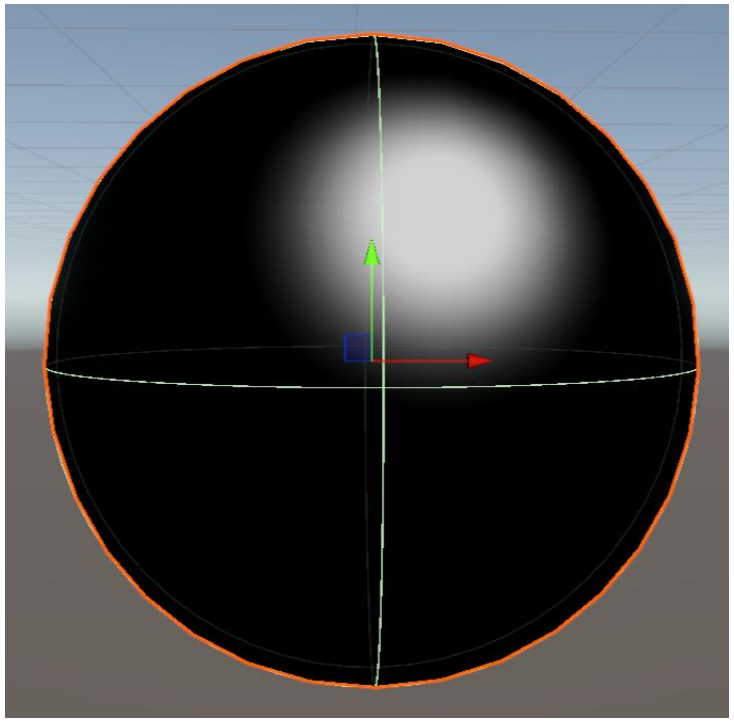
```
50          half4 frag(Varyings IN) : SV_Target
51          {
52              Light light = GetMainLight();
53              half3 normal = normalize(IN.normal);
54              half3 view_direction = normalize(TransformViewToWorld(float3(0,0,0)) - IN.position);
55              half3 half_vector = normalize(light.direction + view_direction);
56              half VdotN = max(0, dot(view_direction, normal));
57              half LdotN = max(0.00001, dot(light.direction, normal));
58              half HdotN = max(0, dot(half_vector, normal));
59                                                          αが向きの平均自乗
60              half alpha2 = _Roughness * _Roughness * _Roughness * _Roughness;
61
62              float D = exp(-(1 - HdotN * HdotN)/(HdotN * HdotN * alpha2))
63                  / (4 * alpha2 * HdotN * HdotN * HdotN * HdotN);
64
65              half3 color = D / (4 * LdotN * VdotN); FやGを除いたBRDF
66              color = saturate(color);  値が大きくなりすぎる
67              return half4(color, 1);
68          }
```

やってみよう

- 粗さのパラメータや光源方向を変えた時の違いを確認しよう



粗すぎるとアーティファクトが発生（後ででなくなる）



Sphere(11)_DistributionFunction

アジェンダ

- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

Cook-Torrance金属反射モデル

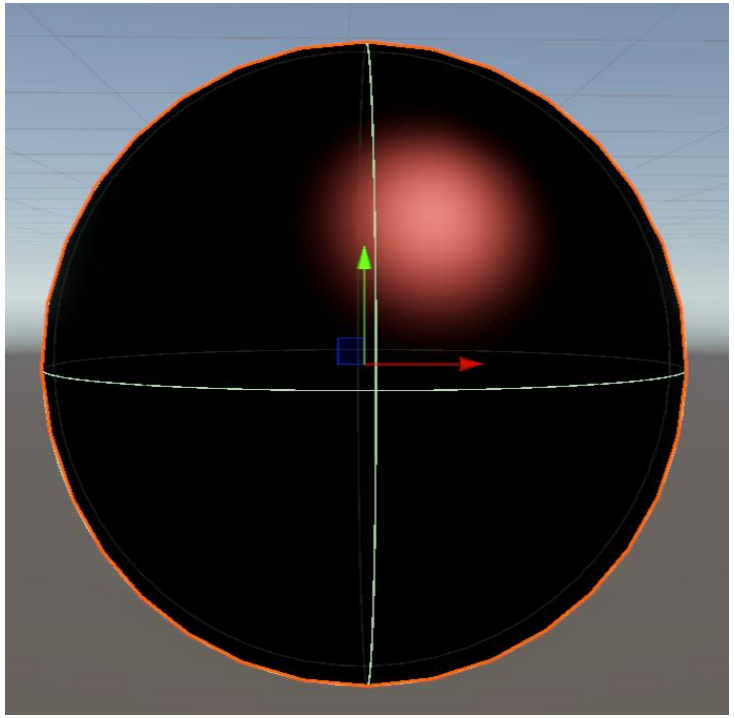
- 今までの計算をまとめたモデル

$$f_r = \frac{FDG}{4(N \cdot L)(N \cdot V)}$$

$$F \sim F_{\text{Schlick}} = F_0 + (1 - F_0)(1 - V \cdot H)^5$$

$$D = \frac{1}{\alpha^2 (H \cdot N)^4} e^{-\left(\frac{\tan(\arccos(H \cdot N))}{\alpha}\right)^2}$$

$$G = \min\left\{1, \frac{2(N \cdot H)(V \cdot N)}{V \cdot H}, \frac{2(N \cdot H)(L \cdot N)}{L \cdot H}\right\}$$



Sphere(12)_CookTorrance

パラメータの追加

- 表面の粗さ
- 正面反射率

```
3       Properties
4       {
5           [MainColor] _BaseColor("Base Color", Color) = (0.86, 0.39, 0.39, 1)
6           _FresnelO("FresnelO", Range(0, 0.99999)) = 0.8
7           _Roughness("Roughness", Range(0.0001, 1)) = 0.4
8       }
```

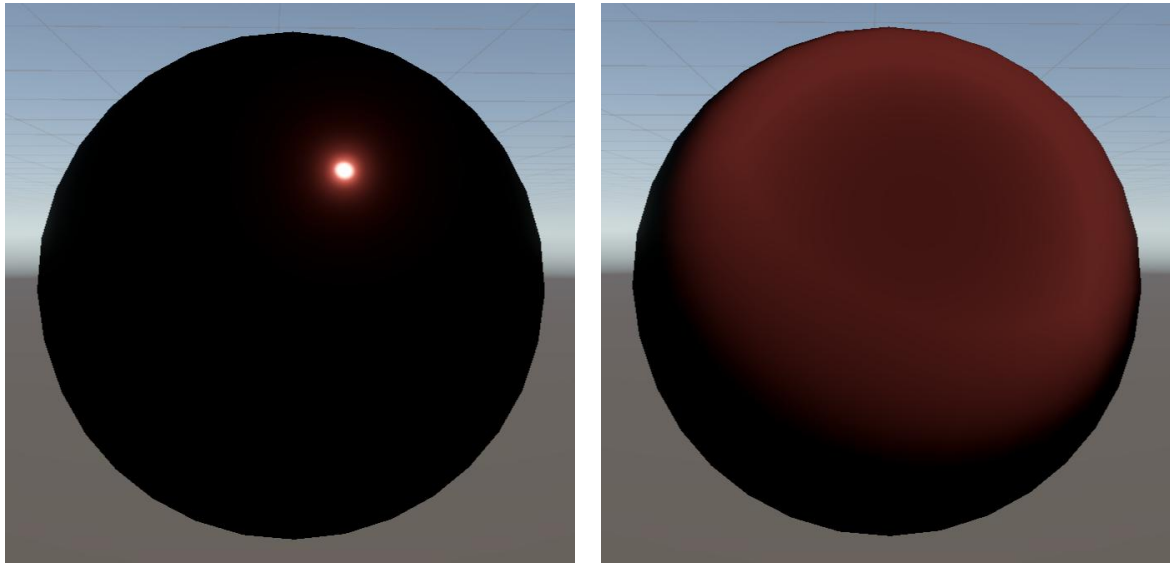
```
37          CBUFFER_START(UnityPerMaterial)
38              half4 _BaseColor;
39              half _FresnelO;
40              half _Roughness;
41          CBUFFER_END
```

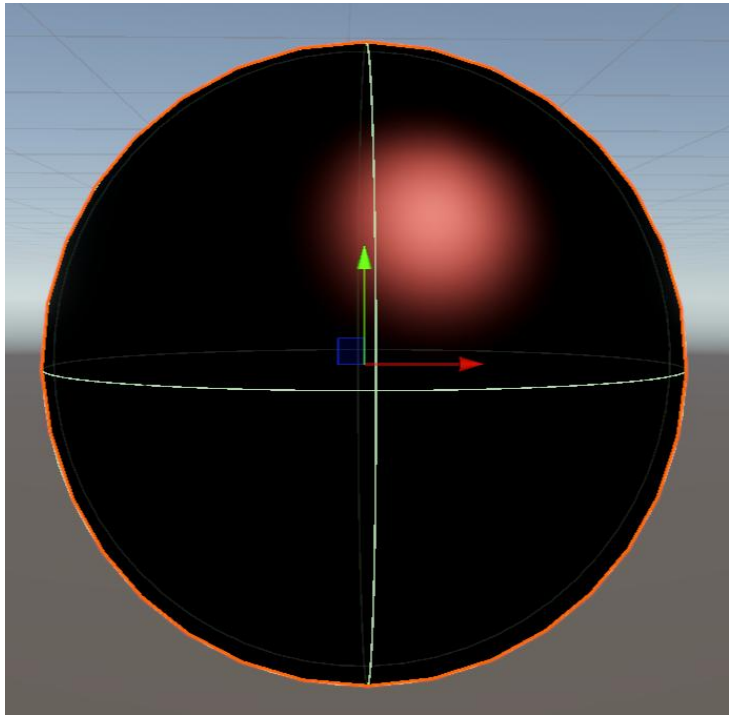
ピクセルシェーダ

```
67          half4 frag(Varyings IN) : SV_Target
68          {
69              Light light = GetMainLight();
70              half3 normal = normalize(IN.normal);
71              half3 view_direction = normalize(TransformViewToWorld(float3(0,0,0)) - IN.position);
72              half3 half_vector = normalize(light.direction + view_direction);
73              half VdotN = max(0.00001, dot(view_direction, normal));
74              half LdotN = max(0.00001, dot(light.direction, normal));  発散しないように0になるのを抑える
75              half HdotN = max(0.00001, dot(half_vector, normal));
76              half LdotH = max(0, dot(half_vector, light.direction));
77              half VdotH = max(0, dot(half_vector, view_direction));
78
79              half alpha2 = _Roughness * _Roughness * _Roughness * _Roughness;
80              float D = exp(-(1 - HdotN * HdotN)/(HdotN * HdotN * alpha2))
81                  / (4 * alpha2 * HdotN * HdotN * HdotN * HdotN);
82
83              half G = min(1, 2 * min(HdotN * VdotN / VdotH, HdotN * LdotN / LdotH));
84              half F = _Fresnel0 + (1-_Fresnel0) * pow(1 - VdotH, 5);// Schlick近似
85              half3 brdf = _BaseColor * D * G * F / (4 * LdotN * VdotN);
86                          鏡面反射に色を乗せた
87              half3 color = light.color * LdotN * brdf; Lambertの余弦則
88              return half4(color, 1);
89          }
```

やってみよう

- 粗さのパラメータや光源方向を変えた時の違いを確認しよう





Sphere(12)_CookTorrance

アジェンダ

- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

より良い反射モデル

- 実際に測定をすると、スペキュラの広がりが大きいことが分かった
- 多くの改良が提案され、次の項がよく使われる

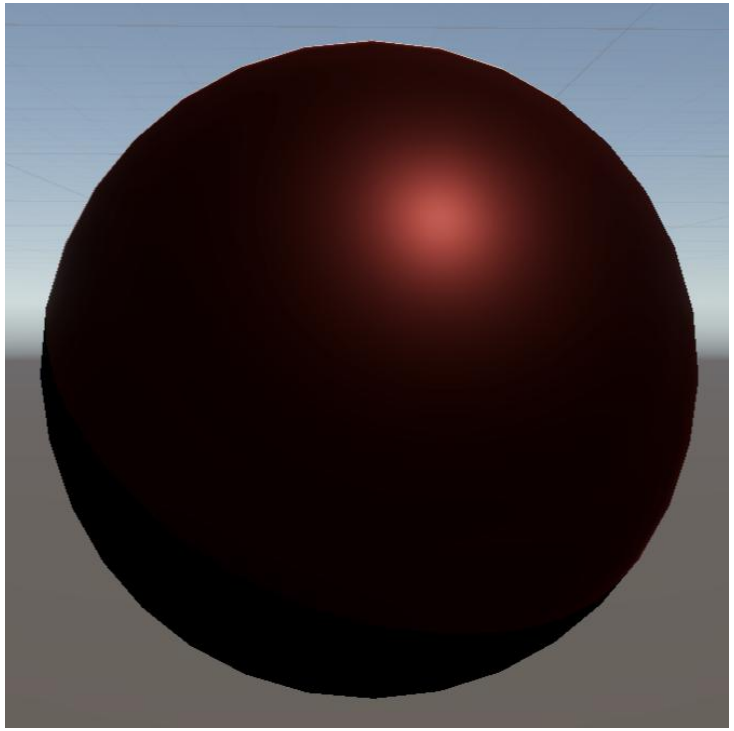
$$f_r = \frac{FDG}{4(N \cdot L)(N \cdot V)}$$

$$D = \frac{\alpha^2}{\pi((H \cdot N)^2(\alpha^2 - 1) + 1)^2}$$

$$G = G_{GGX}(N \cdot L)G_{GGX}(N \cdot V)$$

$$G_{GGX}(c) = \frac{1}{c + \sqrt{\alpha^2 + ac^2 - \alpha^2 c^2}}$$

$$F \sim F_{\text{Schlick}} = F_0 + (1 - F_0)(1 - V \cdot H)^5$$



Sphere(13)_GGX

パラメータの追加(Cook-Torranceと同じ)

- 表面の粗さ
- 正面反射率

```
3       Properties
4       {
5           [MainColor] _BaseColor("Base Color", Color) = (0.86, 0.39, 0.39, 1)
6           _Fresnel0("Fresnel0", Range(0, 1)) = 0.8
7           _Roughness("Roughness", Range(0, 1)) = 0.4
8       }
```

```
37          CBUFFER_START(UnityPerMaterial)
38              half4 _BaseColor;
39              half _Fresnel0;
40              half _Roughness;
41          CBUFFER_END
```

ピクセルシェーダ

```
67          half4 frag(Varyings IN) : SV_Target
68          {
69              Light light = GetMainLight();
70              half3 normal = normalize(IN.normal);
71              half3 view_direction = normalize(TransformViewToWorld(float3(0,0,0)) - IN.position);
72              half3 half_vector = normalize(light.direction + view_direction);
73              half VdotN = max(0.00001, dot(view_direction, normal));
74              half LdotN = max(0.00001, dot(light.direction, normal));
75              half HdotN = max(0.00001, dot(half_vector, normal));
76              half VdotH = max(0.0, dot(view_direction, half_vector));
77
78              half alpha2 = _Roughness * _Roughness * _Roughness * _Roughness;
79              float denom = HdotN * HdotN * (alpha2 - 1.0) + 1.0;
80              float D = alpha2 / (PI * denom * denom);
81
82              half G = min(1, 1
83                      / (VdotN + sqrt(alpha2 + (1.0 - alpha2) * VdotN * VdotN))
84                      / (LdotN + sqrt(alpha2 + (1.0 - alpha2) * LdotN * LdotN)));
85
86              half F = _Fresnel0 + (1-_Fresnel0) * pow(1 - VdotH, 5);// Schlick近似
87
88              half3 brdf = saturate(_BaseColor * D * G * F / (4 * LdotN * VdotN));
89
90              half3 color = light.color * LdotN * brdf;
91              return half4(color, 1);
92          }
```

組み込み関数

- 法線分布関数には同じ関数が用意されている

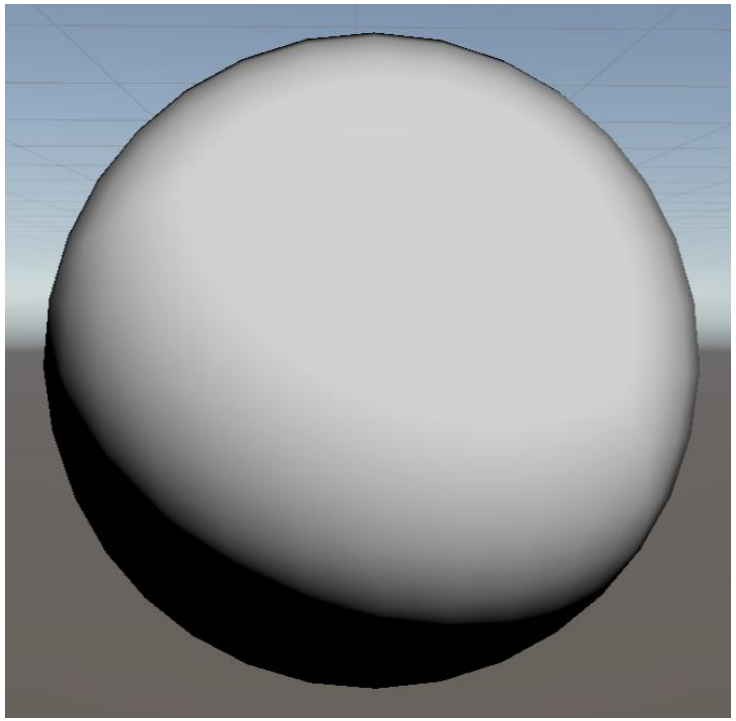
```
79          half alpha2 = _Roughness * _Roughness * _Roughness * _Roughness;
80          float denom = HdotN * HdotN * (alpha2 - 1.0) + 1.0;
81          float D = alpha2 / (PI * denom * denom);
```

同じ処理

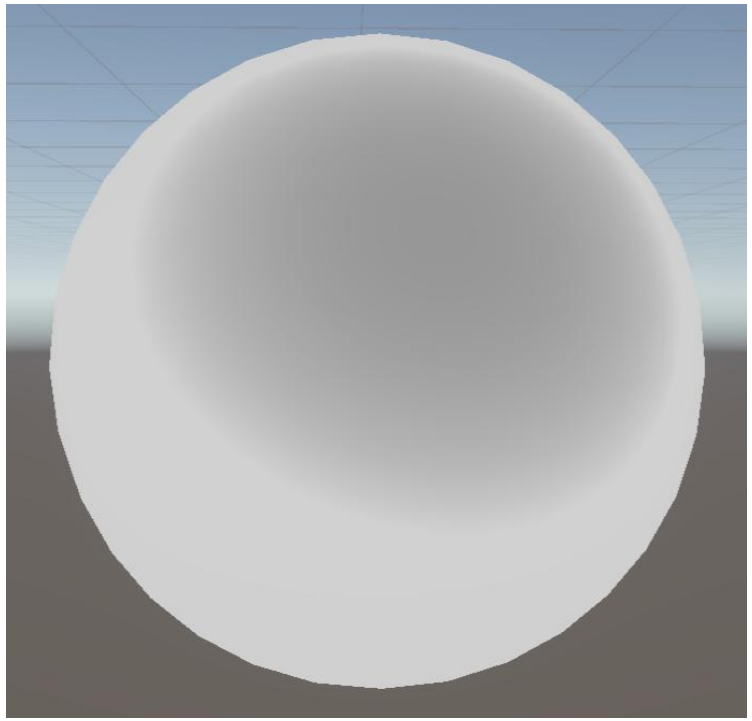
```
79          float D = D_GGX(HdotN, _Roughness * _Roughness);
```

幾何減衰項の違い

- GGXでは横から見ても遮蔽されすぎない
  - マイクロファセットの高さが考慮されている



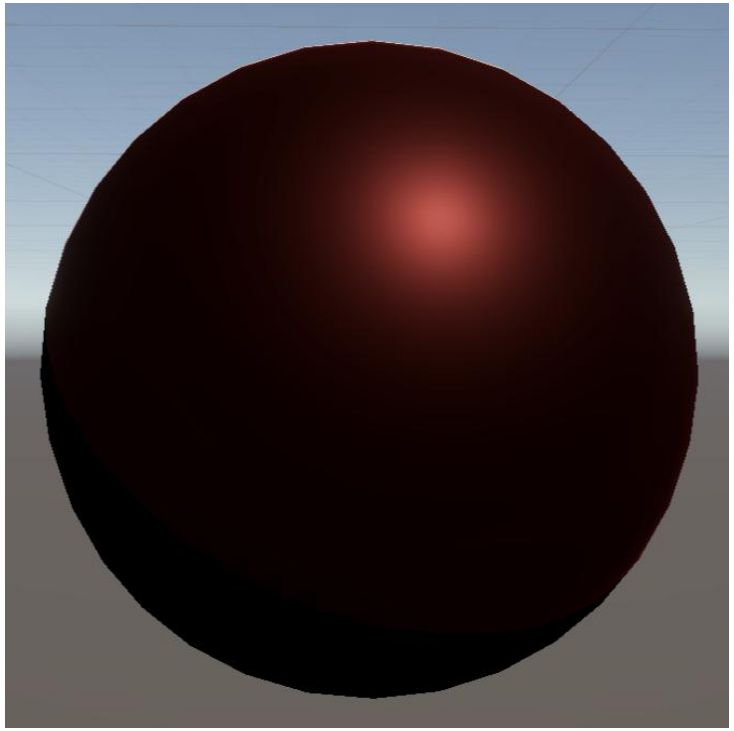
V字溝モデル



GGX

やってみよう

- 粗さのパラメータや光源方向を変えた時の違いを確認しよう



Sphere(13)_GGX

アジェンダ

- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

Frostbite拡散反射モデル

- 鏡面反射光の改善につれて、拡散反射光も改善されてきた
  - Disneyによるモデルが紹介され、さらにFrostbiteエンジン用に改善された
- 拡散反射項は鏡面反射光で使わなかった強度であり、また、ヘルムホルツの相反性を満たすように光の入射方向と視線方向に関して、BRDFを対称な形とする

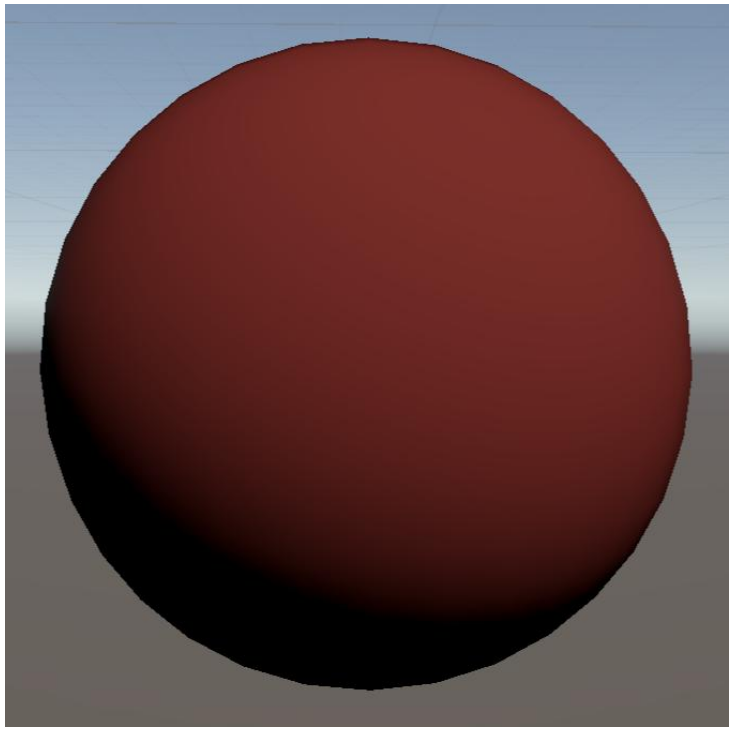
$$f_r = \frac{\sigma}{\pi}\big(1 - F(N \cdot L)\big)\big(1 - F(N \cdot V)\big)$$

$$= \frac{\sigma}{\pi}(1 - F_0 - (1 - F_0)(N \cdot L)^5)(1 - F_0 - (1 - F_0)(N \cdot V)^5)$$

$$= \frac{\sigma}{\pi}(1 - F_0)^2(1 + (-1)(N \cdot L)^5)(1 + (-1)(N \cdot V)^5)$$

- ここで、正面時の反射率を繰り込んで、その分を水平に見た際の反射率で調整する

$$f_r = \frac{\sigma}{\pi}(1 + (F_{D90} - 1)(N \cdot L)^5)(1 + (F_{D90} - 1)(N \cdot V)^5)$$



Sphere(14)_FrostbiteDiffuse

Sébastien Lagarde, Charles de Rousiers, “Moving Frostbite to Physically Based Rendering,” SIGGRAPH 2014 Course: Physically Based Shading in Theory and Practice.

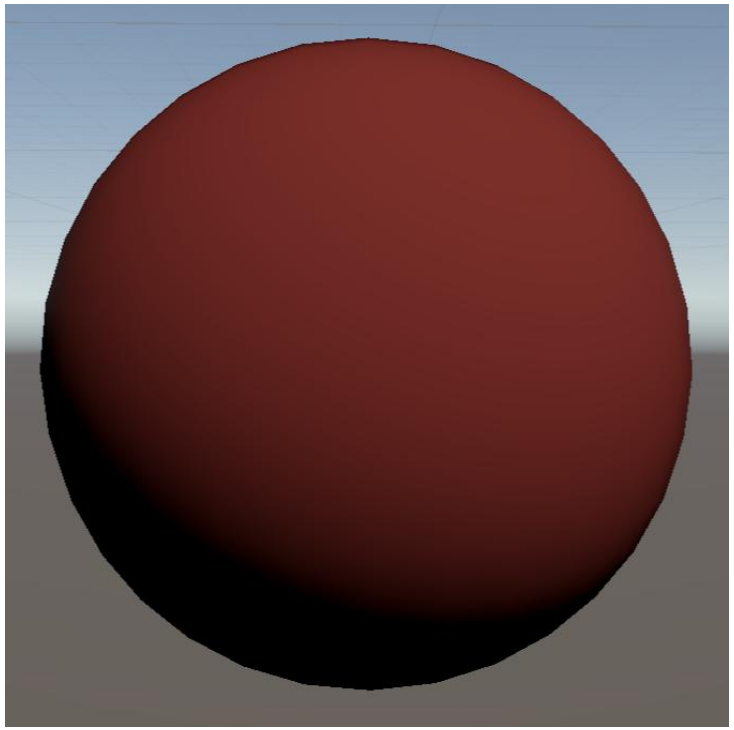
Frostbite エンジンでの拡散反射の実装

```
71  half Fresnel(half f0, half f90, float co)
72  {
73      return f0 + (f90-f0) * pow(1 - co, 5);// Schlick近似
74  }
75
76  // "Moving Frostbite to Physically Based Rendering 3.0"
77  half3 Fr_DisneyDiffuse(half3 albedo, half LdotN, half VdotN, half LdotH, half linearRoughness)
78  {
79      half energyBias = lerp(0.0, 0.5, linearRoughness);
80      half energyFactor = lerp(1.0, 1.0/1.51, linearRoughness);
81      half Fd90 = energyBias + 2.0 * LdotH * LdotH * linearRoughness;
82      half FL = Fresnel(1, Fd90, LdotN);
83      half FV = Fresnel(1, Fd90, VdotN);
84      return (albedo * FL * FV * energyFactor);
85  }
86
87  half4 frag(Varyings IN) : SV_Target
88  {
89      Light light = GetMainLight();
90      half3 normal = normalize(IN.normal);
91      half3 view_direction = normalize(TransformViewToWorld(float3(0,0,0)) - IN.position);
92      half3 half_vector = normalize(light.direction + view_direction);
93      half VdotN = max(0.00001, dot(view_direction, normal));
94      half LdotN = max(0.0, dot(light.direction, normal));
95      half HdotN = max(0.0, dot(half_vector, normal));
96      half LdotH = max(0.0, dot(half_vector, light.direction));
97
98      half3 color = light.color * LdotN
99          * Fr_DisneyDiffuse(_BaseColor, LdotN, VdotN, LdotH, _Roughness * _Roughness) / PI;
100     return half4(color, 1);
101 }
```

公開されている資料から
そのまま持ってきた

やってみよう

- 粗さのパラメータや光源方向を変えた時の違いを確認しよう



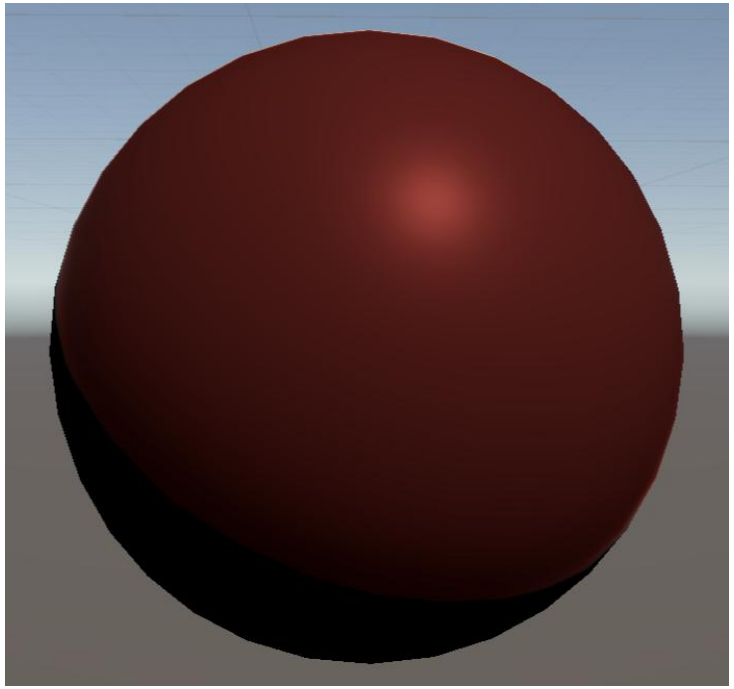
Sphere(14)_FrostbiteDiffuse

アジェンダ

- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

PBR

- いままで説明してきたような考え方が統合されてゲームエンジンでは使われている



Sphere(15)_PBR

パラメータ

- 発光やメタリックを追加

```
 3        Properties
 4        {
 5            [MainColor] _BaseColor("Base Color", Color) = (0.86, 0.39, 0.39, 1)
 6            _SpecularColor("Specular Color", Color) = (1, 1, 1, 1)
 7            _Emission("Emission", Color) = (0, 0, 0, 0)
 8            _Fresnel0("Fresnel0", Range(0, 1)) = 0.8
 9            _Roughness("Roughness", Range(0, 1)) = 0.4
10            _Metallic("Metallic", Range(0, 1)) = 0.6
11        }
```

```
40            CBUFFER_START(UnityPerMaterial)
41                half4 _BaseColor;
42                half4 _SpecularColor;
43                half4 _Emission;
44                half _Fresnel0;
45                half _Roughness;
46                half _Metallic;
47            CBUFFER_END
```

拡散反射

- 前出のまま
- 他のフレネルの手法も用意
  - 近似なしは関数を拡張

```
half FresnelReflectanceAverageDielectric(float co, float f0, float f90)
{
    co = min(0.9999, max(0.000001, co));

    float root_f0 = sqrt(f0);
    float root_f90 = sqrt(f90);
    float n = (root_f90 + root_f0) / (root_f90 - root_f0);
    float n2 = n * n;

    float si2 = 1 - co * co;
    float nb = sqrt(n2 - si2);
    float bn = nb / n2;

    float r_s = (co - nb) / (co + nb);
    float r_p = (co - bn) / (co + bn);
    return 0.5 * f90 * (r_s * r_s + r_p * r_p);
}

half Fresnel(half f0, half f90, float co)
{
//      return FresnelReflectanceAverageDielectric(co, f0, f90);// S波P波平均の無近似
    return f0 + (f90-f0) * pow(1 - co, 5);// Schlick近似
//      return f0 + (f90-f0) * exp(-6 * co);// FarCry3近似
}

// "Moving Frostbite to Physically Based Rendering 3.0"
half3 Fr_DisneyDiffuse(half3 albedo, half LdotN, half VdotN, half LdotH, half linearRoughness)
{
    half energyBias = lerp(0.0, 0.5, linearRoughness);
    half energyFactor = lerp(1.0, 1.0/1.51, linearRoughness);
    half Fd90 = energyBias + 2.0 * LdotH * LdotH * linearRoughness;
    half FL = Fresnel(1, Fd90, LdotN);
    half FV = Fresnel(1, Fd90, VdotN);
    return (albedo * FL * FV * energyFactor);
}
```

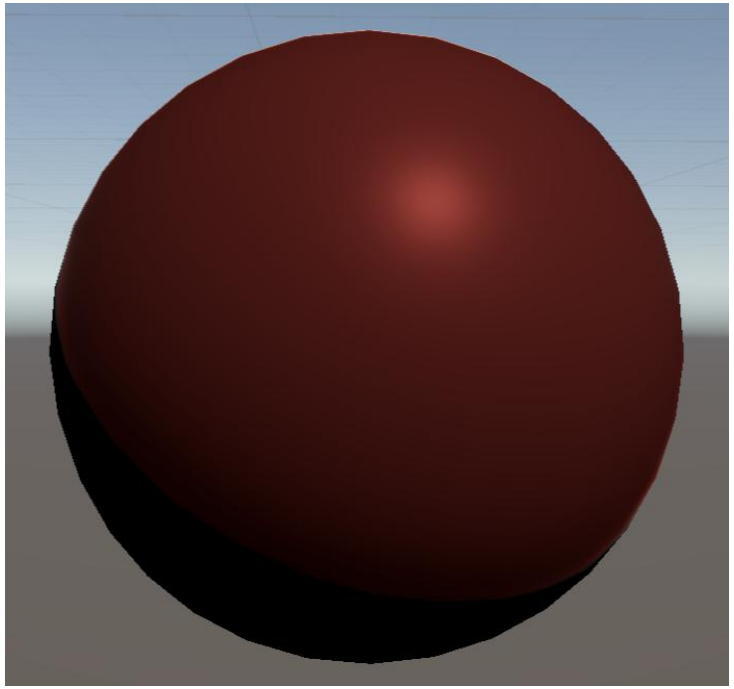
変更点

- Metallic、Emissionを反映
- 幾何減衰項をFrostbiteのものに置き換え
  - マイクロファセットの位置が高いほど、入射・出射方向に対して見える可能性が高くなることを考慮

```
94          // "Moving Frostbite to Physically Based Rendering 3.0"
95          float V_SmithGGXCorrelated(float NdotL, float NdotV, float alphaG2)
96          {
97              // Original formulation of G_SmithGGX Correlated
98              // lambda_v = (-1 + sqrt(alphaG2 * (1-NdotL2) / NdotL2 + 1)) * 0.5f;
99              // lambda_l = (-1 + sqrt(alphaG2 * (1-NdotV2) / NdotV2 + 1)) * 0.5f;
100             // G_SmithGGXCorrelated = 1 / (1 + lambda_v + lambda_l);
101             // V_SmithGGXCorrelated = G_SmithGGXCorrelated / (4.0f * NdotL * NdotV);
102
103             // Caution: the "NdotL *" and "NdotV *" are explicitely inversed, this is not a mistake.
104             float Lambda_GGXV = NdotL * sqrt((-NdotV * alphaG2 + NdotV) * NdotV + alphaG2);
105             float Lambda_GGXL = NdotV * sqrt((-NdotL * alphaG2 + NdotL) * NdotL + alphaG2);
106             return 0.5f / (Lambda_GGXV + Lambda_GGXL);
107         }
108
109         half4 frag(Varyings IN) : SV_Target
110         {
111             Light light = GetMainLight();
112             half3 normal = normalize(IN.normal);
113             half3 view_direction = normalize(TransformViewToWorld(float3(0,0,0)) - IN.position);
114 //            half3 view_direction = TransformViewToWorldNormal(float3(0,0,1));// 頂点の位置を見ない近似
115             half3 half_vector = normalize(light.direction + view_direction);
116             half VdotN = max(0.00001, dot(view_direction, normal));
117             half LdotN = max(0.0, dot(light.direction, normal));
118             half HdotN = max(0.0, dot(half_vector, normal));
119             half LdotH = max(0.0, dot(half_vector, light.direction));
120             half VdotH = max(0.0, dot(half_vector, view_direction));
121
122             half alpha = _Roughness * _Roughness;
123             half3 diffuse = Fr_DisneyDiffuse(_BaseColor, LdotN, VdotN, LdotH, alpha) / PI;
124
125             half alpha2 = alpha * alpha;
126             float D = alpha2 / (PI * pow(HdotN * HdotN * (alpha2 - 1.0) + 1.0, 2.0));
127 //            half G = min(1, 1
128 //                     / (VdotN + sqrt(alpha2 + (1.0 - alpha2) * VdotN * VdotN))
129 //                     / (LdotN + sqrt(alpha2 + (1.0 - alpha2) * LdotN * LdotN)));
130             half G = V_SmithGGXCorrelated(LdotN, VdotN, alpha2);
131             half F = Fresnel(_Fresnel0, 1, VdotH);
132             half3 specular = saturate(_SpecularColor * D * G * F / (4 * LdotN * VdotN));
133
134             half3 color = light.color * LdotN * lerp(diffuse, specular, _Metallic);
135             color += _Emission;// 発光
136             return half4(color, 1);
```

やってみよう

- パラメータや光源方向を変えた時の違いを確認しよう



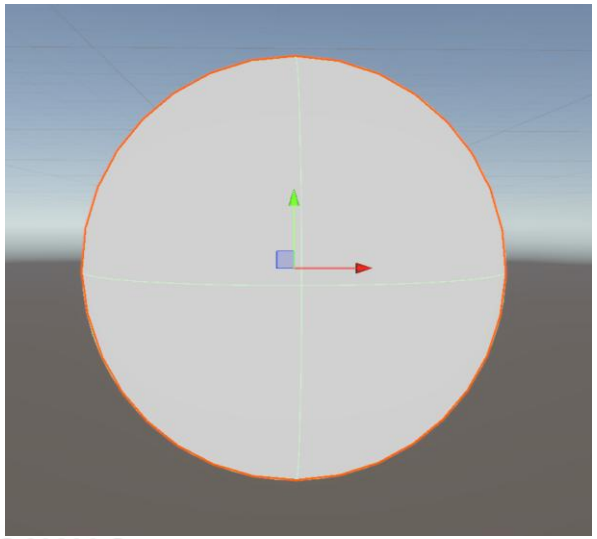
Sphere(15)_PBR

アジェンダ

- おさらい
  - 発光
- Phongの反射モデル
  - 環境光
  - ランバート拡散反射モデル
  - 鏡面反射
  - Phongの反射モデル
- 古典反射モデル
  - ブリンフォン
  - メタリック
  - 異方性反射モデル
- 金属反射
  - フレネル
  - 幾何減衰項
  - Beckman分布関数
  - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR
- 課題

課題

- 今回の授業内容を発展させて「Shader_16_MyShader」を書き替えて素敵なシェーダを書こう
  - PBRの異方性対応
  - テクスチャ対応
  - 他の反射モデル
  - 分散(周波数)への対応
    - トゥーンは今後の授業でやる予定なので×



Shader_16_MyShader

```
Shader "Custom/Shader_16_MyShader"
{
    Properties
    {
        [MainColor] _BaseColor("Base Color", Color) = (1, 1, 1, 1)
        [MainTexture] _BaseMap("Base Map", 2D) = "white"
    }

    SubShader
    {
        Tags { "RenderType" = "Opaque" "RenderPipeline" = "UniversalPipeline" }

        Pass
        {
            HLSLPROGRAM

            #pragma vertex vert
            #pragma fragment frag

            #include "Packages/com.unity.render-pipelines.universal/ShaderLibrary/Core.hlsl"

            struct Attributes
            {
                float4 positionOS : POSITION;
                float2 uv : TEXCOORD0;
            };

            struct Varyings
            {
                float4 positionHCS : SV_POSITION;
                float2 uv : TEXCOORD0;
            };

            TEXTURE2D(_BaseMap);
            SAMPLER(sampler_BaseMap);

            CBUFFER_START(UnityPerMaterial)
                half4 _BaseColor;
                float4 _BaseMap_ST;
            CBUFFER_END

            Varyings vert(Attributes IN)
            {
                Varyings OUT;
                OUT.positionHCS = TransformObjectToHClip(IN.positionOS.xyz);
                OUT.uv = TRANSFORM_TEX(IN.uv, _BaseMap);
                return OUT;
            }

            half4 frag(Varyings IN) : SV_Target
            {
                half4 color = SAMPLE_TEXTURE2D(_BaseMap, sampler_BaseMap, IN.uv) * _BaseColor;
                return color;
            }
            ENDHLSL
        }
    }
}
```

まとめ

- CG基礎
  - 発光
- 古典的な反射モデル
  - Phongの反射モデル
    - 環境光
    - ランバート拡散反射モデル
    - 鏡面反射
    - Phongの反射モデル
  - 古典反射モデル
    - ブリンフォン
    - メタリック
    - 異方性反射モデル
  - 金属反射
    - フレネル
    - 幾何減衰項
    - Beckman分布関数
    - Cook-Torrance金属反射モデル
- 現代的な反射モデル
  - GGX分布
  - Frostbite拡散反射モデル
  - PBR

最後に

- もっと知りたいと思った事を教えてください

今日の内容が何の役に立つんですかという質問がありがちですが
- あなたは足し算が何の役に立つか聞くんですか？
- プログラマによくくる要求は、「あのゲームでやっていることを再現してほしい」です