基礎からはじめる物理ベースレンダリング

mebiusbox software

2018 年 12 月 16 日

目次

1．はじめに

物理ベースレンダリングを勉強しようと資料を探してみると，いきなりレンダリング方程式や BRDF の解説があったり，金属か非金属か，表面は荒いかどうかといった材質の話だったりといった印象でした．極端な話，鏡面反射にクック・トランスの反射モデルを使えばいいよねという感じでした．ところが，資料を見ていくうちに，放射輝度（Radiance），放射照度（Irradiance）や，BSSDF とか様々な用語が出て来るわけです．そこで，光とは何なのか，光の特性はどのようなものか，放射エネルギーに関する基本知識，光が物体表面にぶつかったときに発生する物理現象を必要なものだけ選んでまとめてみました．わかりやすいように図も多く入れたつもりです．今回の目標はレンダリング方程式です．この内容は主にグラフィック関係のプログラマに向けて書いてあります．

2．物理ベースレンダリング

物理ベースレンダリング（Physicaly-based rendering：PBR）とは物体表面における光の反射や媒質内における散乱などの物理現象、光源からシーンを経てカメラに入射する光の伝搬などを計測して数式でモデル化したものを用いてレンダリングすることです。物理ベースレンダリングに使用されるモデルは様々で、物理現象を詳細に表現したモデルほど現実と見分けがつかないほどの映像を作成することが可能ですが、計算にとても時間がかかります。
物理ベースレンダリングで使用するモデルを理解するために、光に関する物理学の基礎を知る必要があります。

3．光

光は電磁波の一種です．電磁波は電場と磁場の変化によって作られる波のことで，光のエネルギーを放出または伝達します．この現象を放射といいます．太陽や電球などの光源から電磁波が発生し，大気中で散乱や吸収されながら直進し，物体表面にぶつかって反射が起こり，私たちの目に届きます．
光は媒質によって伝播されます．媒質となる物体を媒体といいます．

3.1 吸収と散乱

散乱とは光が微小の粒子にぶつかったときに，直進する方向を変えることです．方向は媒体の材質に応じて変化します．吸収とは光のエネルギーが物質との相互作用によって，他の形のエネルギー（主に熱エネルギー）に変わることです．
媒体の厚さは吸収や散乱される度合いに大きく影響します。

3.2 光の特性

3.2.1 直進性

光は電磁波の一種なので、障害物がなく均一な物体の中を通る限りは直進します。光の速度は 1 秒間に 30 万 km という速さです。

3.2.2 反射性

光は鏡や研磨された金属の表面や白いものの表面で反射します（鏡面反射）。このとき，反射の法則により，完全に平坦な表面上においては、入ってきた光と反射する光の角度が等しくなります（完全反射）。しかし、表面に凹凸がある場合はいろんな方向に反射されます．これを乱反射といいます．

3.2.3 屈折性

光は通り抜ける物体によって速度が変わります。そのため、密度の違う物質の境界では光が折れ曲がって進むように見えます。これを屈折と呼びます。

3.2.4 吸収性

物質には特定の波長の光を吸収する性質があります。たとえば、赤いものは、白色光があたると赤以外の波長の光を吸収して赤を反射させるために人間の目には赤く見えます。また、明るい色ほど吸収せず、逆に黒はすべての光を吸収します。光が吸収されると多くの場合、そのエネルギーは熱に変わります。変化した量に応じて光の強さは減衰し、光の色はその波長に応じて吸収された光量だけ変化しますが、光線の方向は変わりません。

3.3 光のエネルギー

太陽や照明の光が物体の表面に当たると光の特性により、反射されたり、屈折したり、吸収されたりします。光は何度も反射や屈折を繰り返して、私たちの眼に色として視覚されます。色は人間の視覚が作りだす心理的な量で、物理的な量のことではないそうです。光のもつ物理的なエネルギーは光学において放射量といいます。

> 人間の眼は光の波長に応じて感度が異なり、波長の違いは色として知覚されます。人間の眼を通した光の量を測光量といい、測光学（Photometry）の扱いになります。

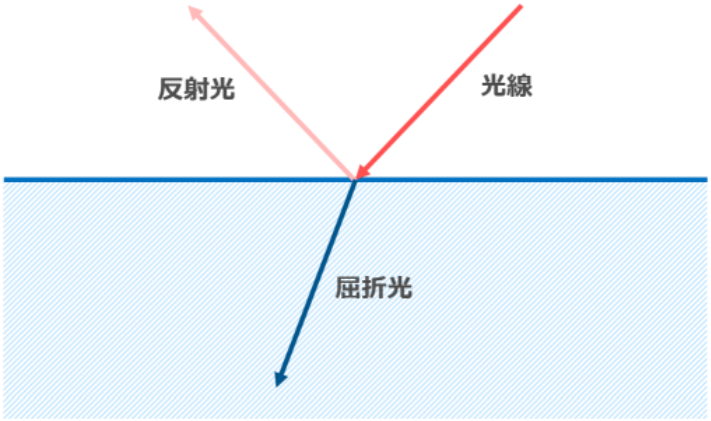
> 物体の表面はそれ自身が発光することがあります。この場合は太陽や照明の光が当たって反射していなくても物体の表面から光が放出されます。

3.4 光線

光線がある表面に当たると以下のいずれか、もしくは両方が発生します。

① 光線が表面で反射し、異なる方向へ進んでいきます。これは反射の法則に従えば反射角は入射角に等しくなります
② 光線は表面で速度が変わり、屈折して一方の媒体からもう一方の媒体へと通り抜けます。

つまり、光線はある表面に当たると、反射する方向と屈折する方向の２つに分かれることになります。



反射や屈折した光は、最終的にいずれかの媒体によって吸収されます。

3.5 反射光

物体表面に当たる光線は入射光（Incident light）、その当たる角度は入射角（Angle of Incidence）といいます。また，光線がある表面に当たって反射された光線は反射光（Reflected Light）、その角度は反射角（Angle of Reflection）といいます。

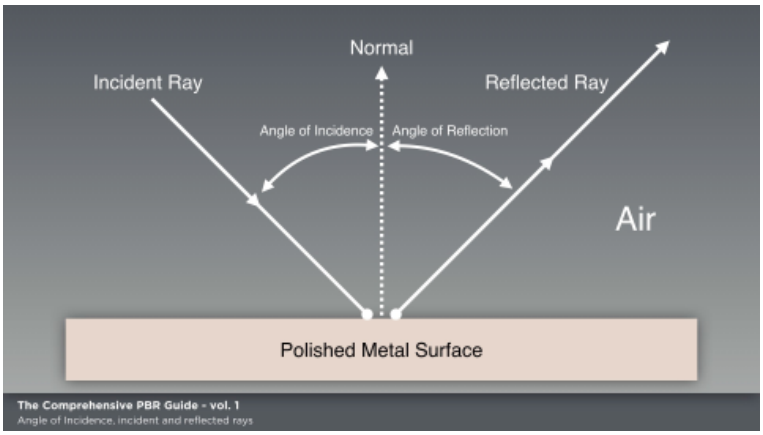


Fig.1: Wes McDermott「The Comprehensive PBR Guide by Allegorithmic - vol. 1 Light and Matter : The theory of Physically-Based Rendering and Shading」©Allegorithmic

3.6 鏡面反射光

物体表面に当たって反射された光は鏡面反射光と呼びます。鏡面反射する光の方向は物体表面の凹凸によって変化します．もし完全に平坦で凸凹のない表面にぶつかると，反射した光線の角度と入射角は同じになります．これを反射の法則といいます．通常，物体表面には多少凹凸があるため，反射する光の方向は鏡面反射方向に拡散し，反射した光はぼやけて見えます．一方，物体表面がなめらかなときは収束して，反射した光は鮮明になります

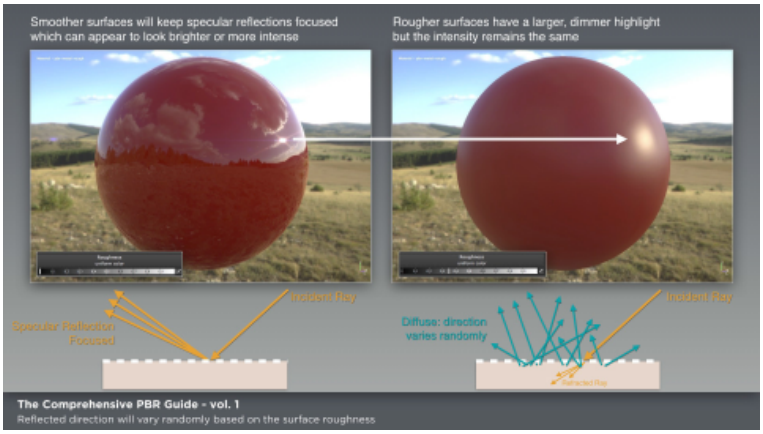


Fig.2: Wes McDermott「The Comprehensive PBR Guide by Allegorithmic - vol. 1 Light and Matter : The theory of Physically-Based Rendering and Shading」©Allegorithmic

3.7 拡散反射光

光線はある媒体から別の媒体に入ってくることがあります．このとき媒体と別の媒体の境界で屈折します．屈折して別の媒体に入った光はその媒体の内部で何度も散乱されたり，吸収されてしまいます．このとき吸収されずに散乱された光がまた元の媒体に出射されることがあります（媒体の境界でまた屈折します）。これを拡散反射と呼びます。つまり、拡散反射光とは屈折した光のことになります。

媒体から別の媒体に入射した光が元の媒体に戻ってくるときに，別の媒体から出射される位置は，入射した位置とは限りません．これが表面下散乱という物理現象を起こしています．例えば，人間の皮膚内では光が進めば吸収されていきますので，皮膚内で進む距離が長くなると光が弱くなっていきます．光源に向かって手をかざしてみると皮膚の薄いところがより透けて見えます．これは皮膚内に入射した光が外に出射されるまでに皮膚内を進んだ距離が短いためで，この距離はその媒体の厚さに影響を受けると言えます．

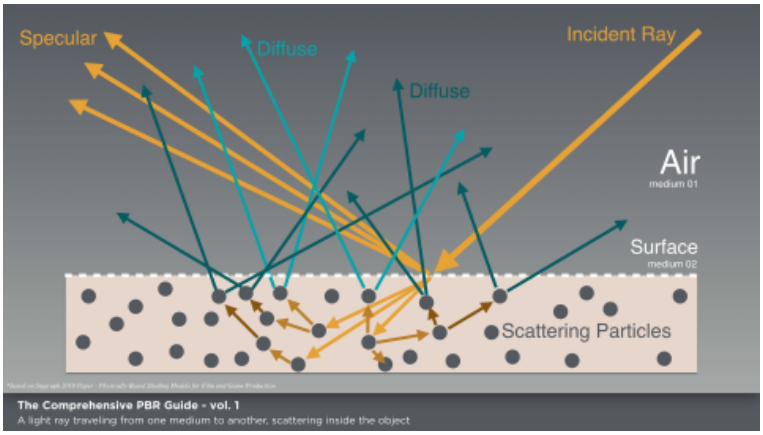


Fig.3: Wes McDermott「The Comprehensive PBR Guide by Allegorithmic - vol. 1 Light and Matter : The theory of Physically-Based Rendering and Shading」©Allegorithmic

3.8 微小面

光線が物体表面にぶつかって反射する方向は，物体表面の凹凸に依存します．粗い物体表面を拡大して見てみると，そこには肉眼では確認できないほど小さな凹凸が見られます．この小さな凹凸は微小面（microfacet）と呼ばれる小さい面が集まっているものとして考え，微小面の傾きから物体表面の粗さを計算します．各微小面は完全鏡面反射すると想定します．微小面により光が物体表面に到達する直前や，光が物体表面で反射した直後に，反射点の近傍の物体表面によって遮られるという現象が起きま

す．この現象をローカルオクリュージョンといいます．ローカルオクリュージョンには，セルフシャドウイングとセルフマスキングがあります．セルフシャドウイングは光が物体表面に到達するまでの間に物体表面によって遮られる現象のことで，セルフマスキングは光が物体表面で反射された後に物体表面から遠ざかる間に遮られる現象のことです．

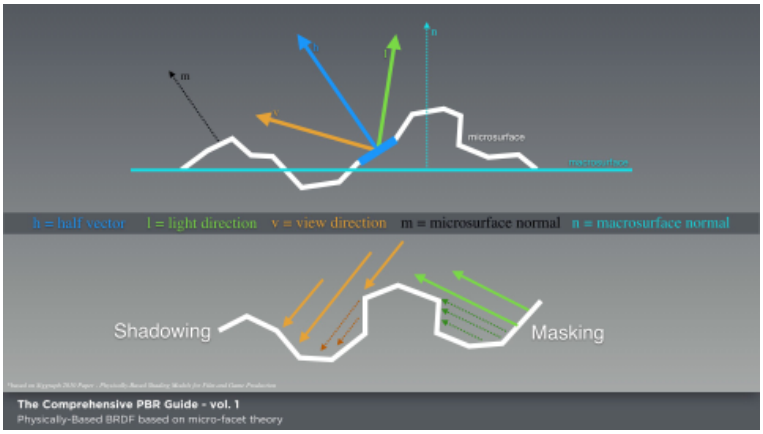


Fig.4: Wes McDermott「The Comprehensive PBR Guide by Allegorithmic - vol. 1 Light and Matter : The theory of Physically-Based Rendering and Shading」 ©Allegorithmic

微小面は光の向きと視線の向きのちょうど中間の方向に向かって光を反射すれば，その微小面から反射された光が見えるということになりますが，ローカルオクリュージョンによって光が遮られる場合もあります．物体表面が粗く凹凸があるということは，微小面の傾きがばらばらであり，これによって光が拡散します．

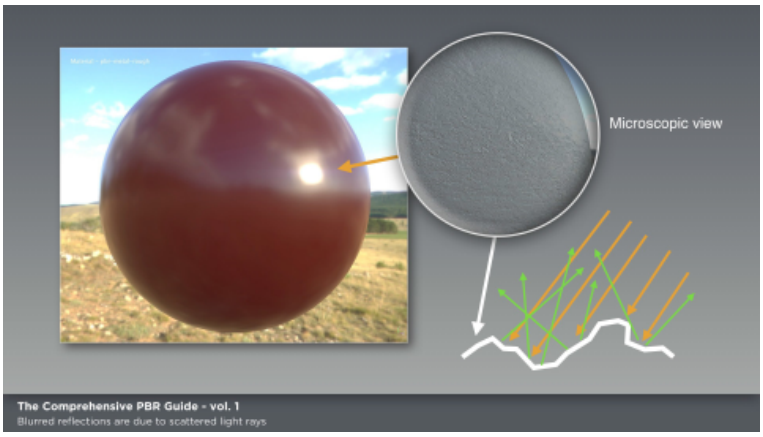


Fig.5: Wes McDermott「The Comprehensive PBR Guide by Allegorithmic - vol. 1 Light and Matter : The theory of Physically-Based Rendering and Shading」 ©Allegorithmic

3.9 記号一覧

ここで利用する記号と意味の表です

記号	説明
x	位置
x'	入射光の位置
$\vec{n}$	x における法線（常に正規化されている）
$\vec{\omega}$	方向（表面から離れる方向）
$\vec{\omega}'$	放射輝度（radiance）の入射方向（表面から離れる方向）
$d\vec{\omega}$	立体角の微分
(θ, ϕ)	極座標系における方向
L	放射輝度
$L(x, \vec{\omega})$	位置 x における $\vec{\omega}$ 方向の放射輝度
$L(x, \vec{\omega}')$	位置 x に $\vec{\omega}$ 方向から入射する放射輝度
$L(x' \rightarrow x)$	位置 x' から x 方向に出射される放射輝度
L_e	放射された（emitted）放射輝度
L_r	反射された（reflected）放射輝度
L_i	入射する（incident）放射輝度
Φ	放射束（flux）
E	放射照度（irradiance）
f_r	双方向反射率分布関数（BRDF）
f_d	拡散（diffuse）BRDF
f_s	鏡面（specular）BRDF
ρ	反射率（reflectance）
Ω	方向を表す半球（hemisphere）
η	屈折率（index of refraction）

3.10 放射束

光の基本単位は光子（photon）です。放射エネルギー（radiant energy）Q は、光子が集まったエネルギーです。このエネルギーを単位時間あたりで表したものを放射束（Flux）といいます。Φ で表記します。

$$\Phi = \frac{\Delta Q}{\Delta d} = \frac{dQ}{dt} \tag{1}$$

3.11 放射照度（Irradiance）

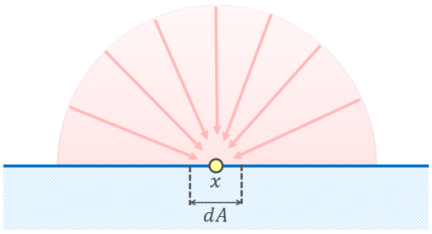
放射照度 E は単位面積あたりの放射束のことです．ある面積 A に到達する放射束を Φ とすると、放射照度 E は次のようになります。

$$E = \frac{\Phi}{A} \tag{2}$$

ある位置 x に入ってくる放射照度は次のようになります。

$$E(x) = \frac{d\Phi}{dA} \tag{3}$$

dA は単位面積で，位置 x の法線方向に垂直な微小面の面積のことです．

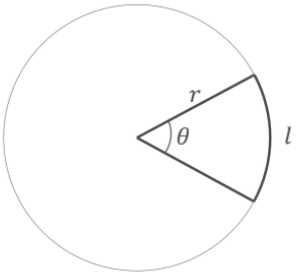


放射照度は入ってくる放射束のことですが、物体表面から出ていく放射束のことを放射発散度（radiant exitance）と呼び、Mと記述します。また、放射発散度はラジオシティ（radiocity）Bとしても知られています。

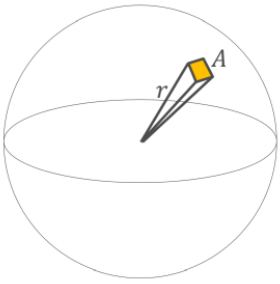
3.12 立体角

立体角（solid angle）は、方向と光線の角度的な「大きさ」を表します。立体角の単位はステラジアン（sr）です．立体角は平面角を３次元に拡張したものと考えることができます．平面角 θ は，円上に張られた弧の長さ l を円の半径 r で割ったものです．

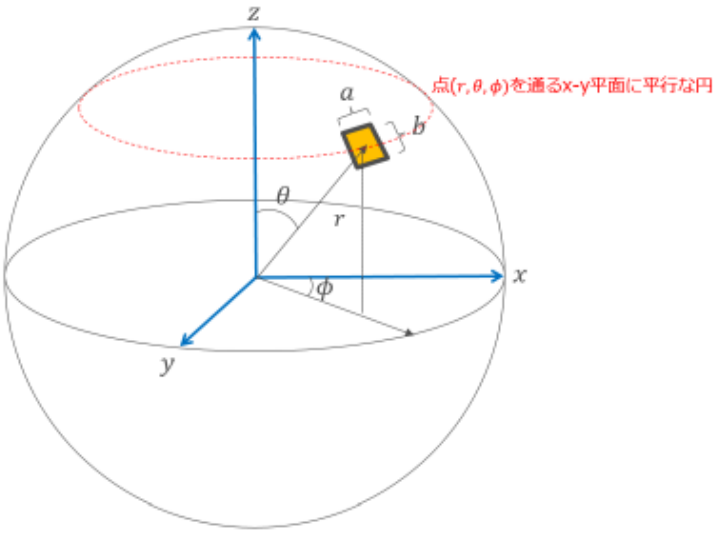
$$\theta = \frac{l}{r} \tag{4}$$



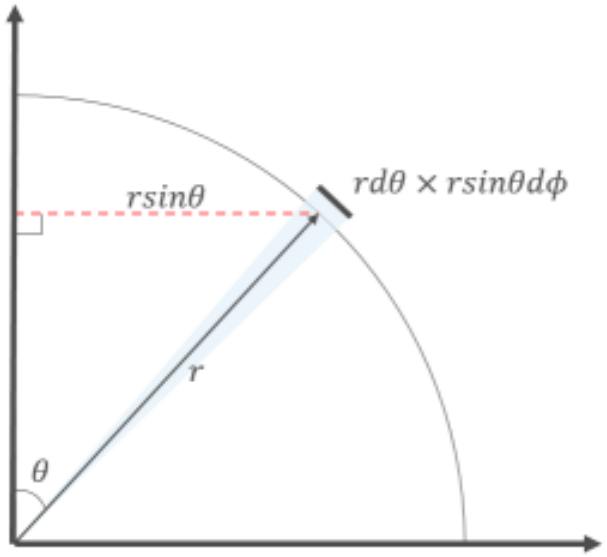
これを３次元で考えると，半径 r の球面上の面積 A に対応する立体角 ω は $\omega = A/r^2$ です．球の面積は $4\pi r^2$ ですから，球全体の立体角は 4π ステラジアンになります．



この方向と大きさは、球面座標（θ, ϕ）を用いて表現することができます。ここで，点 $< r, \theta, \phi >$ における微小表面積を考えてみます．



極角 θ における円（点 $<r,\theta,\phi>$ を通る x-y 平面に平行な円：図の赤い円）の半径は $r\sin\theta$ ですので，この円上の方位角方向の微小弧長 a は $r\sin\theta d\phi$ となります．そして，極角方向の微小弧長 b は $rd\theta$ となります．



最終的に球面上の点 $<r,\theta,\phi>$ における微小表面積は次のようになります．

$$\begin{aligned} dA &= ab \\ &= r\sin\theta d\phi \times rd\theta \\ &= r^2\sin\theta d\theta d\phi \end{aligned} \tag{5}$$

半径 r の球面上の面積 A に対する立体角 ω は，$\omega = A/r^2$ でしたので，微小立体角は次のようになります．

$$
\begin{aligned}
d\vec{\omega} &= \frac{dA}{r^2} \\
&= \frac{r^2 \sin\theta d\theta d\phi}{r^2} \\
&= \sin\theta d\theta d\phi
\end{aligned} \tag{6}
$$

3.13 放射強度

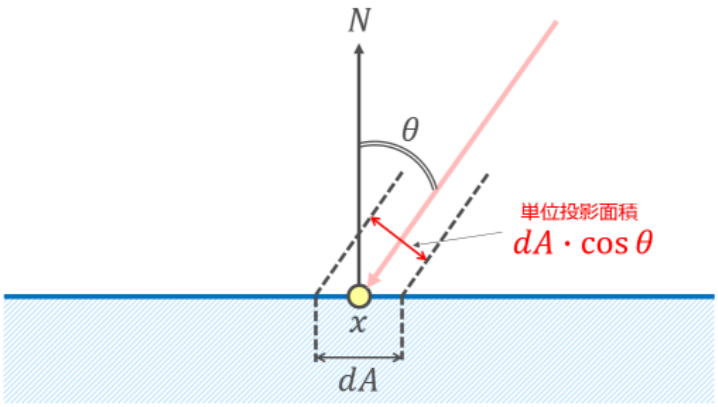
放射強度（radiant intensity）I は、単位立体角（ω）あたりの放射束です。

$$
I(\vec{\omega}) = \frac{d\Phi}{d\vec{\omega}} \tag{7}
$$

放射強度は一定の方向に、どの程度の放射束が放出されているのかを表します。同じ 100W の電球でも、笠をつけて一定方向に光を集中させることによって、強い放射強度を得ることができます。

3.14 放射輝度

放射輝度（radiance）L は、単位立体角あたり、単位投影面積あたりの放射束のことです。単位投影面積 $\cos\theta dA$ とは，光が進む方向に直交する面へ単位面積を投影したもので，放射輝度はこの単位投影面積で放射強度を割った値になります．



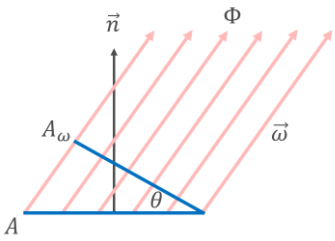
$$
L(x,\vec{\omega}) = \frac{d^2\Phi}{\cos\theta dA d\vec{\omega}} \tag{8}
$$

物体表面における入射してくる放射輝度がわかっていれば、半球面上のすべての方向 Ω と物体表面の領域 A で積分することにより、放射束を計算することができます。

$$
\Phi = \int_A \int_\Omega L(x,\vec{\omega}')(\vec{\omega}' \cdot \vec{n}) d\vec{\omega}' dx \tag{9}
$$

3.15 放射照度と放射輝度の関係

ある領域 A を通過する放射束 Φ があるとすると，ある領域の放射照度 E は Φ/A となります．また同じ位置で，同じ放射束が通過する $\vec{\omega}$ 方向と直交する面の領域を考えると，この領域の面積 A_ω は $A\cos\theta$ となります．これはランバートのコサイン則として知られています．



よって，この領域における放射照度は次のようになります．

$$E_{\vec{\omega}} = \frac{\Phi}{A_{\vec{\omega}}} = \frac{\Phi}{A\cos\theta} \tag{10}$$

これに放射照度の関係を入れると次のようになります．

$$E = E_{\vec{\omega}} \cos\theta \tag{11}$$

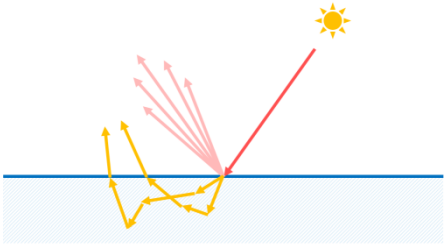
点 x における放射照度は，半球面上のあらゆる方向から入射した放射輝度を足し合わせたものと考えられます．放射照度と放射輝度との関係により，放射輝度に入射角の余弦（$\cos\theta$）を掛けることで放射照度に変換することができます．入射角の余弦は，法線ベクトルと入射方向の単位ベクトルとの内積 $\vec{\omega}' \cdot \vec{n}$ で求められ，一般的にコサイン項と呼ばれています．これらをまとめると次のようになります．

$$E(x) = \int_\Omega L_i(x, \vec{\omega}')(\vec{\omega}' \cdot \vec{n}) d\vec{\omega}' \tag{12}$$

4．双方向散乱面反射率分布関数

4.1 双方向散乱面反射率分布関数

光がある媒体から別の媒体へ屈折して入射し，内部で散乱して光が入射した位置とは違う場所から出射していくことで，表面下散乱の物理現象が発生します．これは皮膚などの半透明（translucent）な材質でよく起こる現象です．この散乱は、双方向散乱面反射率分布関数（bidirectional scattering surface reflectance distribution function：BSSRDF）で表すことができます．

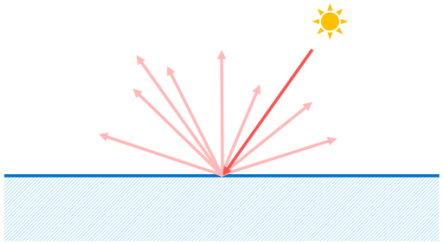


BSSRDF S は、位置 x において $\vec{\omega}$ 方向に反射された放射輝度の微分 dL_r を，位置 x' に方向 $\vec{\omega}$ から入射する放射束の微分 $d\Phi_i$ で表せます．

$$S(x, \vec{\omega}, x', \vec{\omega}') = \frac{dL_r(x, \vec{\omega})}{d\Phi_i(x', \vec{\omega}')} \tag{13}$$

4.2 双方向反射率分布関数

BSSRDF では入射する位置と出射する位置を別々に考えた関数ですが，入射する位置と出射する位置が同じだと想定して，BSSRDF を単純化したのが双方向反射率分布関数（bidirectional reflectance distribution function：BRDF）です．



BRDF f_r は，反射される放射輝度と放射照度の関係を以下のように定義しています．

$$f_r(x,\vec{\omega}',\vec{\omega}) = \frac{dL_r(x,\vec{\omega})}{dE_i(x,\vec{\omega}')} = \frac{dL_r(x,\vec{\omega})}{L_i(x,\vec{\omega}')(\vec{\omega}'\cdot\vec{n})d\vec{\omega}'} \tag{14}$$

BRDF は，点 x に $\vec{\omega}'$ 方向から入射した光のどれだけが，$\vec{\omega}$ 方向に反射されるかを表す割合です．一般的に反射の特徴は，「物体表面の材質」と「光の当たり方」の２つで決まってきます．「物体表面の材質」の部分は BRDF，「光の当たり方」の部分はコサイン項で定義します．

物体表面上のある位置に入射する放射輝度がわかっていれば，それがあらゆる方向に反射していく放射輝度を計算することができます．これは，入射する放射輝度 L_i を積分することによって求められます．

$$L_r(x,\vec{\omega}) = \int_\Omega f_r(x,\vec{\omega}',\vec{\omega})dE(x,\vec{\omega}') = \int_\Omega f_r(x,\vec{\omega}',\vec{\omega})L_i(x,\vec{\omega}')(\vec{\omega}'\cdot\vec{n})d\vec{\omega} \tag{15}$$

BRDF の物理的に重要な性質が２つあります．

4.2.1 相反性

光が進む方向に依存しないというヘルムホルツ（Helmholtz）の相反性（reciprocity）の法則を満たしている必要があります．つまり，光が入射する方向と出射する方向が入れ替わっても BRDF の結果は変わらないことを意味しています．

$$f_r(x,\vec{\omega}',\vec{\omega}) = f_r(x,\vec{\omega},\vec{\omega}') \tag{16}$$

4.2.2 エネルギー保存則

物体表面は，それが受けた以上の光を反射することはできず，以下の式を満たさなければなりません．

$$\int_\Omega f_r(x,\vec{\omega}',\vec{\omega})(\vec{\omega}'\cdot\vec{n})d\vec{\omega}' \leq 1, \forall\vec{\omega} \tag{17}$$

4.3 反射率

物体表面に入射する放射束と反射される放射束の比のことで，物体表面の反射率（reflectance）ρ で表します．

$$\rho(x) = \frac{d\Phi_r(x)}{d\Phi_i(x)} \tag{18}$$

物理法則に基づく場合，この値は０～１の範囲でなければなりません．

4.4 BRDF と反射モデル

BRDF は光のエネルギーを用いて厳密に定義する物理量です．この BRDF を用いて実際に反射される放射輝度を計算できるようにしたものが反射モデルです．BRDF が放射照度と放射輝度の比と定義されていること，放射照度が法線と光源方向のなす角の余弦に依存していることから，反射される放射輝度は BRDF に余弦項 $\cos\theta$ を掛けたもので，この形が一般的な反射モデルです．

光反射の過程は物理的に複雑で，物体表面の反射特性を詳細にモデル化した反射モデルほど複雑で，計算する時間も多くなっていきます．

4.5 正規化 BRDF

BRDF の性質であるエネルギー保存則は，ある位置 x において放出される放射輝度が入射される放射輝度（放射照度）を超えてはいけないことでした．BRDF に基づいた反射モデルよりも以前によく使われていたランバート反射モデルやフォン反射モデルなどはエネルギー保存則を満たしていません．そこで，それらの反射モデルを正規化することで，エネルギー保存則を満たし，BRDF に基づいた反射モデルとして使用することができます．
例えば，BRDF に基づいていないランバード反射モデルは次のようになっています（ρ_d は拡散反射率）．

$$L_{lambert} = \rho_d \cos\theta \tag{19}$$

これを BRDF とコサイン項に分けると BRDF は ρ_d となります．エネルギー保存則を考慮すると次の式を満たさなければなりません．

$$\int_\Omega \rho_d \cos\theta d\vec{\omega} \leq 1 \tag{20}$$

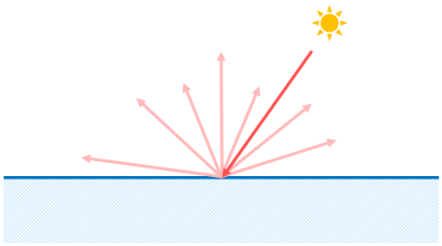
$\cos\theta$ の半球積分は π なので（詳細は拡散反射のところで説明），拡散反射率 ρ_d を 1 とすると，結果は π となり明らかに 1 を超えてしまいます．そこで，拡散反射率を π で割ることでエネルギー保存則を満たすことになります．

$$\int_\Omega \frac{\rho_d}{\pi} \cos\theta d\vec{\omega} \leq 1 \tag{21}$$

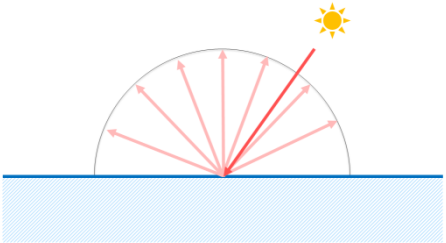
つまり，エネルギー保存則を満たしたランバート反射の BRDF は $\frac{\rho_d}{\pi}$ となり，この BRDF は正規化されていることになります．

4.6 拡散反射

物体表面に光がぶつかると，光の一部は屈折して物体の中に入ってから物体表面に放出されることがあります．このとき，BRDF では入射した光と同じ位置から物体表面に放出されるとします．放出される光の方向は不規則なため，全方向に放出されると考えることもできます．



ランバート反射（Lambertian diffuse reflection）では入射光の向きにかかわらず，反射輝度はすべての方向に対して一定になります．ランバート反射を引き起こす物体表面をランバート面といいます．



$$L_r(x,\vec{\omega}) = f_{r,d}\int_\Omega dE_i(x,\vec{\omega}') = f_{r,d}(x)E_i(x) \tag{22}$$

この関係を用いて，ランバート面の拡散反射率 ρ_d を求めると，以下のようになります．

$$\rho_d(x) = \frac{d\Phi_r(x)}{d\Phi_l(x)} = \frac{L_r(x)dA\int_\Omega(\vec{\omega}\cdot\vec{n})d\vec{\omega}}{E_i(x)dA} \tag{23}$$

$$L_r(x) = f_{r,d}E(x) \tag{24}$$

$$\rho_d(x) = f_{r,d}(x)\int_\Omega(\vec{n}\cdot\vec{\omega}') = \pi f_{r,d}(x) \tag{25}$$

この ρ_d をディフューズアルベド（diffuse albedo），単にアルベドともいいます．

整理すると，ランバート反射の BRDF $f_{r,d}$ は次のようになります．

$$f_{r,d}(x) = \frac{\rho_d(x)}{\pi} \tag{26}$$

4.6.1 cos Θの半球積分

ランバート反射面ではすべての方向に一様に反射しますので，微小立体角あたりの放射光は $\cos\theta$ に比例します．半球上のすべての方向に反射する光の総量は $\cos\theta$ を積分することで求められ，次のようになります．

$$\int_\Omega(\vec{n}\cdot\vec{\omega}')d\vec{\omega} = \pi \tag{27}$$

つまり，$\cos\theta$ を半球積分すると π になります．この計算は次のようになっています．

法線ベクトルと入射ベクトルの内積はコサイン項と同じです．

$$(\vec{n}\cdot\vec{\omega}') = \cos\theta \tag{28}$$

半球の積分は球面座標系の積分に書き直せます．

$$\int_\Omega f(\theta,\phi)d\vec{\omega}' = \int_0^{2\pi}\int_0^{\pi/2} f(\theta,\phi)\sin\theta d\theta d\phi \tag{29}$$

整理すると

$$\int_\Omega (\vec{n}\cdot\vec{\omega})d\vec{\omega}' = \int_0^{2\pi}\int_0^{\frac{\pi}{2}} \cos\theta\sin\theta d\theta d\phi$$

$$= \int_0^{2\pi} d\phi \int_0^{\pi/2} \cos\theta\sin\theta d\theta$$

$$= 2\pi\int_0^{2\pi} \cos\theta\sin\theta d\theta$$

2倍角の公式：$\sin 2\theta = 2\sin\theta\cos\theta$

$$= 2\pi\int_0^{2\pi} \frac{\sin 2\theta}{2} d\theta$$

$$= \pi\int_0^{2\pi} \sin 2\theta d\theta$$

$t = 2\theta, \frac{dt}{d\theta} = 2, d\theta = \frac{1}{2}dt$

$$= \pi\int_0^{\pi} \sin t\frac{1}{2}dt$$

$$= \pi\frac{1}{2}[-\cos t]_0^{\pi}$$

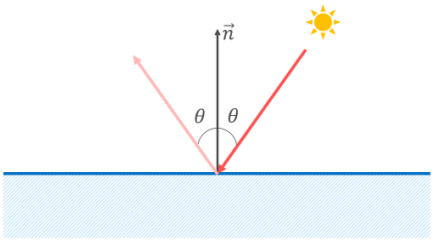
$$= \pi\frac{1}{2}(-\cos\pi - (-(\cos 0))$$

$$= \pi\frac{1}{2}(1+1) = \pi\frac{1}{2}(2)$$

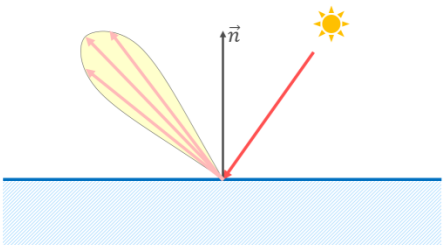
$$= \pi$$

4.7 鏡面反射

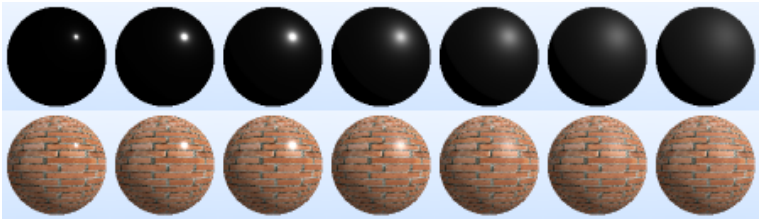
鏡面反射は入射光を鏡面反射方向に反射します．



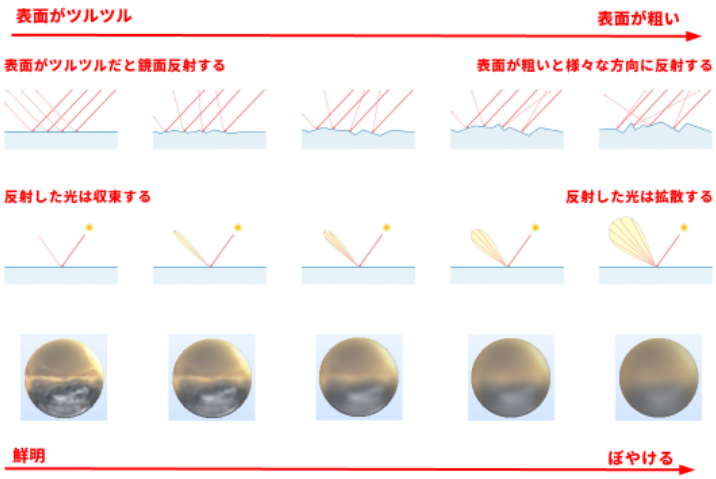
通常，物体表面にはある程度の粗さがあるため，鏡面反射方向だけでなく，鏡面反射方向に若干拡散します．この部分を光沢といいます．



次の図は物体表面の粗さと光沢の関係を表しています．右にいくほど表面は粗くなり，光沢部分が広がって鏡面反射した光（ハイライト）がぼやけていきます．



また，たとえば金属のような鏡面反射率が高いものは周囲の景色が映りこんで見えます．このときの映り込んだ景色も表面の粗さによって見え方が変わってきます．



鏡面反射によって反射された放射輝度は次のようになります．

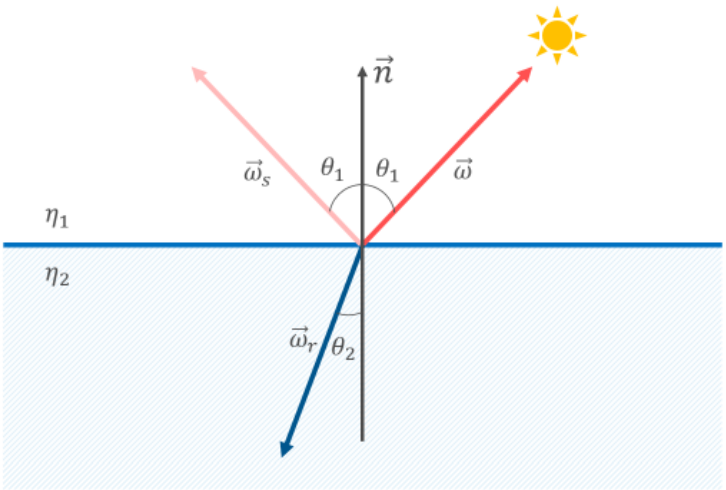
$$L_r(x,\vec{\omega}_s) = \rho_s(x) L_i(x,\vec{\omega}') \tag{30}$$

完全な鏡面反射における反射方向 $\vec{\omega}_s$ は，次のようになります．

$$\vec{\omega}_s = 2(\vec{\omega}' \cdot \vec{n})\vec{n} - \vec{\omega}' \tag{31}$$

4.7.1 反射と屈折

光が物体表面にぶつかると光は反射光と屈折光にわかれますが，このとき，屈折する方向はスネルの法則で求められます．また，入射光が反射光と屈折光にどれくらい分配されるかはフレネルの方程式で求められます．



4.7.2 フレネルの方程式

屈折率 η_1 の媒質中の光線が，屈折率 η_2 の物質にぶつかるとき，光の反射量は以下のように計算することができます．

$$\rho_{\parallel} = \frac{\eta_2 \cos\theta_1 - \eta_1 \cos\theta_2}{\eta_2 \cos\theta_1 + \eta_1 \cos\theta_2} \tag{32}$$

$$\rho_{\perp} = \frac{\eta_1 \cos\theta_1 - \eta_2 \cos\theta_2}{\eta_1 \cos\theta_1 + \eta_2 \cos\theta_2} \tag{33}$$

フレネルの方程式は偏光された光を対象としています．偏光とは特定の方向に振動している光のことです．普通の光は，あらゆる方向に振動している光が混ざっています．

$\rho_{\parallel}$ は入射面に対して平行な電界を有する光の反射係数（reflection coefficient）であり，$\rho_{\perp}$ は直交するそれになります．

物質ごとの屈折率（index of refracction）は計測されて，書籍やインターネット上で見つけることができます．

RefractiveIndex.INFO

偏光されていない光に対しては鏡面反射率（reflectance）は以下のようになります．

$$F_r(\theta) = \frac{1}{2}(\rho_{\parallel}^2 + \rho_{\perp}^2) = \frac{d\Phi_r}{d\Phi_i} \tag{34}$$

フレネルの反射係数は Shlick の近似式がよく使われます．

$$F_r(\theta) \approx F_0 + (1 - F_0)(1 - \cos\theta)^5 \tag{35}$$

F_0 は，垂直入射におけるフレネル反射係数です．

$$F_0 = \frac{(\eta_1 - \eta_2)^2}{(\eta_1 + \eta_2)^2} \tag{36}$$

4.7.3 屈折

光が媒質の境界で別の媒質側へ進むとき，光の進行方向が変わる現象が起こり，これを屈折と呼びます．

光がある媒質を透過する速度を v とするとき，真空中の光速 c と媒質中の光速との比は

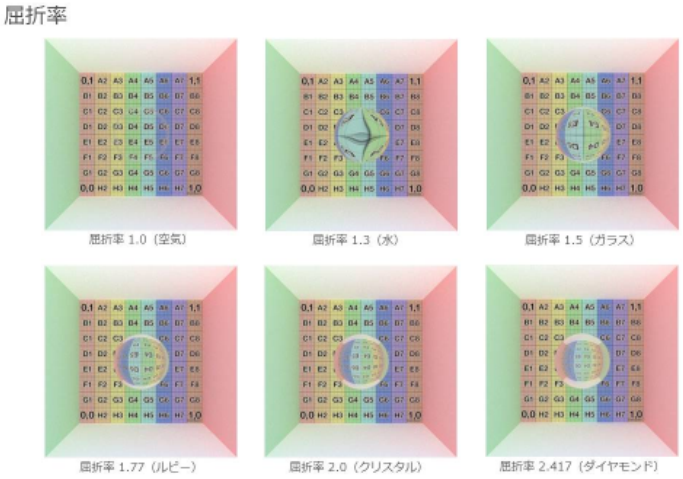
$$\eta = \frac{c}{v} \tag{37}$$

となります．この η がその媒質の屈折率です．

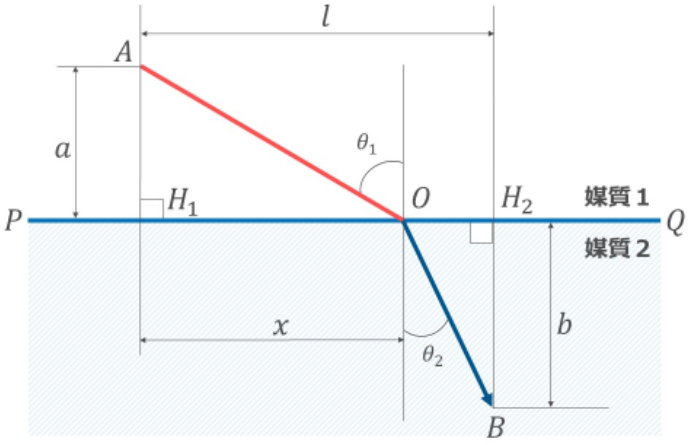
入射角と屈折角の関係は，屈折前の媒質の屈折率 η_1 と，屈折後の媒質の屈折率 η_2 からスネルの法則（Snell's law）を用いて計算することができます．

$$\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2 \tag{38}$$

θ_2 は屈折角です．



4.7.4　スネルの法則



PQ を媒質の境界として，媒質 1 内の点 A から境界 PQ 上の点 O に達して屈折し，媒質 2 内の点 B に進むとします．媒質 1 での光速を v_1，媒質 2 での光速を v_2，真空中の光速を c とすれば

$$\eta_1 = \frac{c}{v_1} \tag{39}$$

$$\eta_2 = \frac{c}{v_2} \tag{40}$$

となります．点 A と点 B から境界 PQ に下ろした垂線の足を H_1, H_2 としたとき

$$H_1H_2 = l \tag{41}$$

$$AH_1 = a \tag{42}$$

$$BH_2 = b \tag{43}$$

と定義します．点 H_1 から点 O までの距離を x として，この x を求めて点 O の位置を特定します．

AO 間を光が進むのにかかる時間は

$$t_{AO} = \frac{AO}{v_1} = \frac{\eta_1}{c} AO \tag{44}$$

また，OB 間を光が進むのにかかる時間は

$$t_{OB} = \frac{OB}{v_2} = \frac{\eta_2}{c} OB \tag{45}$$

となります．したがって，光が AOB 間を進むのにかかる時間は次のようになります．

$$t = t_{AO} + t_{OB} = \frac{1}{c}(\eta_1 AO + \eta_2 OB) \tag{46}$$

AO と OB はピタゴラスの定理から

$$AO = \sqrt{x^2 + a^2} \tag{47}$$

$$OB = \sqrt{(l-x)^2 + b^2} \tag{48}$$

だとわかります．整理すると次のようになります．

$$t = \frac{1}{c}(\eta_1 \sqrt{x^2 + a^2} + \eta_2 \sqrt{(l-x)^2 + b^2} \tag{49}$$

フェルマーの原理によると，「光が媒質中を進む経路は，その間を進行するのにかかる時間が最小となる経路である」といえます．すなわち，光は AOB 間を進むのにかかる時間 t が最小となる経路を通ると考え，さきほどの式 (49) の t が最小となるのは

$$\frac{dt}{dx} = 0 \tag{50}$$

を満たすときです．式 (1) を代入すると次のようになります．

$$\frac{dt}{dx} = \frac{d}{dx}\left\{\frac{1}{c}(\eta_1 \sqrt{x^2 + a^2} + \eta_2 \sqrt{(l-x)^2 + b^2})\right\} = 0 \tag{51}$$

1/c は定数なので外に出せます．

$$\frac{dt}{dx} = \frac{1}{c}\left(\eta_1 \sqrt{x^2 + a^2} + \eta_2 \sqrt{(l-x)^2 + b^2}\right)' = 0 \tag{52}$$

和の微分ですので，η_1 と η_2 のある項をそれぞれ x で微分して足し合わせます．

$$\begin{aligned}
\eta_1 \sqrt{x^2 + a^2}' &= \eta_1 \frac{(x^2 + a^2)'}{2\sqrt{x^2 + a^2}} \\
&= \eta_1 \frac{2x}{2\sqrt{x^2 + a^2}} \\
&= \eta_1 \frac{x}{\sqrt{x^2 + a^2}} \\
\eta_2 \sqrt{(l-x)^2 + b^2}' &= \eta_2 \frac{((l-x)^2 + b^2)'}{2\sqrt{(l-x)^2 + b^2}} \\
&= \eta_2 \frac{(l^2 - 2lx + x^2 + b^2)'}{2\sqrt{(l-x)^2 + b^2}} \\
&= \eta_2 \frac{-2(l-x)}{2\sqrt{(l-x)^2 + b^2}} \\
&= \eta_2 \frac{-(l-x)}{\sqrt{(l-x)^2 + b^2}}
\end{aligned}$$

整理すると

$$\frac{\eta_1}{c}\frac{x}{\sqrt{x^2+a^2}}-\frac{\eta_2}{c}\frac{l-x}{\sqrt{(l-x)^2+b^2}}=0 \tag{53}$$

図から $\sin\theta_1, \sin\theta_2$ を定義すると

$$\sin\theta_1=\frac{H_1O}{AO}=\frac{x}{\sqrt{x^2+a^2}} \tag{54}$$

$$\sin\theta_2=\frac{H_2O}{OB}=\frac{l-x}{\sqrt{(l-x)^2+b^2}} \tag{55}$$

となりますので，先ほどの式 (53) を書き換えて移項し，c を両辺にかけて消すと

$$\eta_1\sin\theta_1=\eta_2\sin\theta_2 \tag{56}$$

となります．これがスネルの法則の式です．

4.7.5 全反射

スネルの法則の式を変形して，

$$\sin\theta_2=\frac{\eta_1}{\eta_2}\sin\theta_a \tag{57}$$

とするとき，$\eta_1<\eta_2$ ならば，$\eta_1/\eta_2<1$ となります．また，$0<\sin\theta_1<1$ であり，式 (57) から $\sin\theta_2$ は

$$0<\sin\theta_2<1 \tag{58}$$

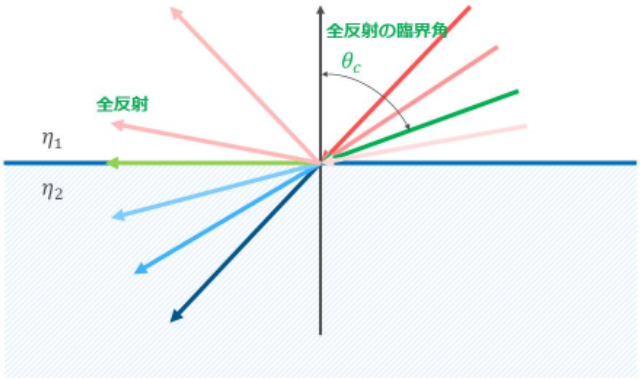
となりますから，式 (57) を満たす屈折角 θ_2 が必ず存在することになります．逆に，$\eta_1>\eta_2$ の場合は，$\eta_1/\eta_2>1$ なので，式 (57) において，$\sin\theta_1$ が大きいと，$\sin\theta_2>1$ となり解が得られない場合があります．入射角 θ_1 を次第に大きくしていくとき，

$$\sin\theta_2=1 \tag{59}$$

すなわち，屈折角 θ_2 が 90° となり，屈折光が発生しなくなる限界の入射角を θ_c とすれば，

$$\sin^{-1}\frac{\eta_2}{\eta_1} \tag{60}$$

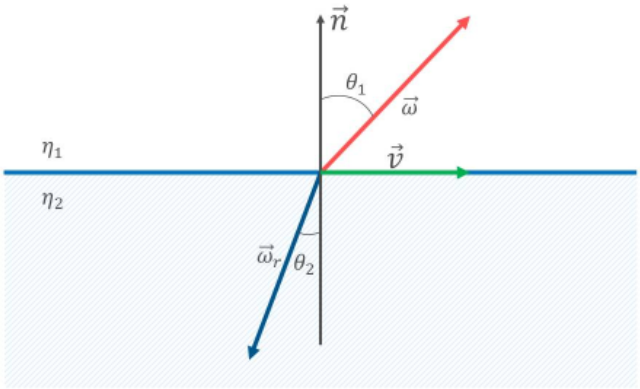
と表せます．下図のように入射角が θ_c を超えると全部の光を反射します．これを全反射といいます．また，この屈折光が発生しなくなる限界の入射角 θ_c を全反射の臨界角といいます．



4.7.6 屈折光の方向

屈折光の方向はスネルの法則を使って求めることができます.

入射ベクトルと法線ベクトルを含む面があるとし，その面上で法線ベクトルと直交している単位ベクトルを $\vec{v}$ とします.



この単位ベクトルと屈折ベクトル $\vec{\omega}_r$ の関係を表すと次のようになります.

$$\vec{\omega}_r = -\vec{v}\sin\theta_2 - \vec{n}\cos\theta_2 \tag{61}$$

入射ベクトルと法線ベクトルを使って，$\vec{v}$ の関係を表すと

$$\vec{v}\sin\theta_1 = \vec{\omega} - (\vec{\omega}\cdot\vec{n})\vec{n} \tag{62}$$

これを先ほどの式に代入すると

$$\vec{\omega}_r = -\frac{\vec{\omega} - (\vec{\omega}\cdot\vec{n})\vec{n}}{\sin\theta_1}\sin\theta_2 - \vec{n}\cos\theta_2 \tag{63}$$

スネルの法則を使って $\sin\theta_1$ を書き換えると

$$\vec{\omega}_r = -\frac{\vec{\omega} - (\vec{\omega}\cdot\vec{n})\vec{n}}{\frac{\eta_2}{\eta_1}\sin\theta_2}\sin\theta_2 - \vec{n}\cos\theta_2 \tag{64}$$

$$= -\frac{\eta_1}{\eta_2}(\vec{\omega} - (\vec{\omega}\cdot\vec{n})\vec{n}) - \vec{n}\cos\theta_2 \tag{65}$$

次に $\cos\theta_2$ を変形していきます．まず $\cos^2\theta = 1 - sin^2\theta$ を使って

$$\vec{\omega}_r = -\frac{\eta_1}{\eta_2}(\vec{\omega} - (\vec{\omega}\cdot\vec{n})\vec{n}) - \vec{n}\sqrt{1-\sin^2\theta_2} \tag{66}$$

スネルの法則を使って $\sin^2\theta_2$ を $\sin^2\theta_1$ に変えます．

$$\vec{\omega}_r = -\frac{\eta_1}{\eta_2}(\vec{\omega} - (\vec{\omega}\cdot\vec{n})\vec{n}) - \vec{n}\sqrt{1-\left(\frac{\eta_1}{\eta_2}\right)^2\sin^2\theta_1} \tag{67}$$

そして，$\sin^2\theta_1$ を $1-\cos^2\theta_1$ に置き換えます．

$$\vec{\omega}_r = -\frac{\eta_1}{\eta_2}(\vec{\omega} - (\vec{\omega}\cdot\vec{n})\vec{n}) - \vec{n}\sqrt{1-\left(\frac{\eta_1}{\eta_2}\right)^2(1-\cos^2\theta_1)} \tag{68}$$

$\cos\theta_1$ は入射ベクトルと法線ベクトルの内積で求まります．

$$\cos\theta_1 = \vec{\omega}\cdot\vec{n} \tag{69}$$

これを代入すると

$$\vec{\omega}_r = -\frac{\eta_1}{\eta_2}(\vec{\omega} - (\vec{\omega}\cdot\vec{n})\vec{n}) - \vec{n}\sqrt{1-\left(\frac{\eta_1}{\eta_2}\right)^2(1-(\vec{\omega}\cdot\vec{n})^2)} \tag{70}$$

となります．これが屈折光の方向を計算する式です．

4.7.7 屈折光の量

屈折光が運ぶ光の量は $1-F_r$ になります．

4.8 様々な BRDF モデル

拡散反射や鏡面反射など各反射に適した BRDF が提唱されています．

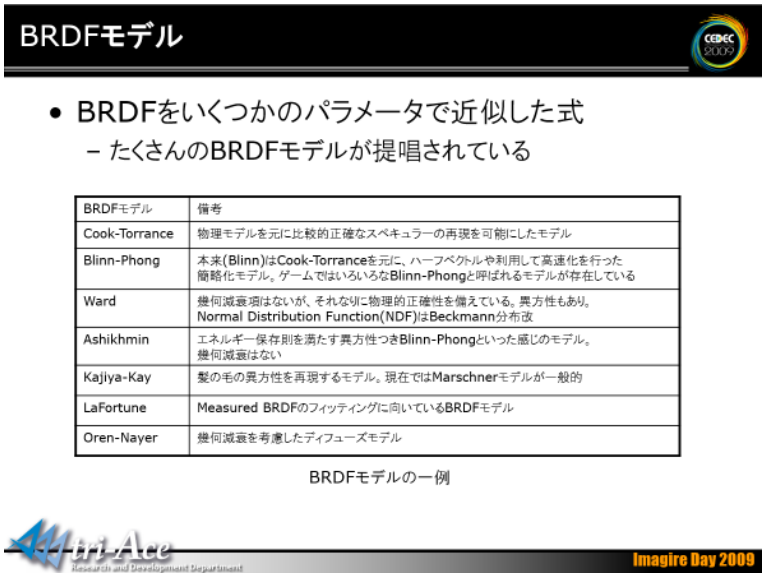


Fig.6: 五反田義治「本当のＨＤＲ表現へ〜アーティストにも知ってほしいリフレクタンスの基礎〜」 ©tri-Ace, 2009

4.9 マイクロファセット BRDF

鏡面反射の BRDF としてクック・トランスのマイクロファセット BRDF は物理ベースの BRDF としてよく使われています．この BRDF は次のように定義されています．

$$f_r = \frac{DGF}{4\cos\theta_i \cos\theta_r} \tag{71}$$

D はマイクロファセット分布関数，G は幾何減衰項，F はフレネルです．

ここでは記号を次のように定義します．

記号	説明
n	法線ベクトル
l	ライトベクトル
v	視線ベクトル
h	ライトベクトルと視線ベクトルの中間ベクトル
α	ラフネスの２乗

これらの記号を使って，先ほどの式を整理すると次のようになります．

$$f_r = \frac{D(h)G(l,v)F(v,h)}{4(n\cdot l)(n\cdot v)} \tag{72}$$

4.9.1 マイクロファセット分布関数

法線を中心とした半球内の任意の方向を向いた微小面の分布関数です．微小面がどれだけ任意の方向を向いているかを表します．この任意の方向は入射方向の逆ベクトルと，出射方向のベクトルの中間ベクトルが使われます．マイクロファセット分布関数も様々な関数が提唱されていますが，よく使われるものは GGX(Throwbridge-Reiz) です．GGX はエネルギー保存が考慮されている NDF（Normal Distribution Function）です．

$$D_{GGX}(h) = \frac{\alpha^2}{\pi((n\cdot h)^2(\alpha^2-1)+1)^2} \tag{73}$$

4.9.2 幾何減衰項

幾何減衰項（Geometry Attenuation Factor）は，微小面によるローカルオクリュージョンでの隠蔽減衰を表したものです．よく使われるのは Smith モデルに，Schlick の近似式を使ったものです．

$$k = \frac{\alpha}{2} \tag{74}$$

$$G_{Schlick}(v) = \frac{n \cdot v}{(n \cdot v)(1-k)+k} \tag{75}$$

$$G_{Smith}(l,v) = G_{Schlick}(l)G_{Schlick}(v) \tag{76}$$

4.9.3 フレネル

フレネルは Schlick の近似式を使います．

$$F_{Schlick}(v,h) = F_0 + (1-F_0)(1-v \cdot h)^5 \tag{77}$$

4.9.4 "4" か "π " か

Walter らの論文「Microfacet Models for Refraction through Rough Surface」によればマイクロファセット BRDF は以下のように定義されています．

$$f_r = \frac{DGF}{4\cos\theta_i \cos\theta_r} \tag{78}$$

クック・トランスの論文「A Reflectance Model for Computer Graphics」では BRDF は次のように定義されています．

$$f_r = \frac{F}{\pi}\frac{DG}{\cos\theta_i \cos\theta_r} \tag{79}$$

クック・トランスの BRDF としてどちらともよく使われているみたいなのでややこしいですが，基本的に Walter らのマイクロファセット BRDF を使うことで問題なさそうです．後者の式はよくみるとフレネルを π で割っているように見えます．論文によると，フレネル項は計測されたデータや Schlick 近似の場合，垂直方向に入射してきたときの反射率になるので，ランバート反射のように反射率を π で割ります．では，前者の方はというと，これも論文によれば，マイクロファセット分布関数Dの正規化が後者のときとは異なっているからです（例えば同じ Beckmann でも前者と後者の論文では数式が違う）．

5．レンダリング方程式

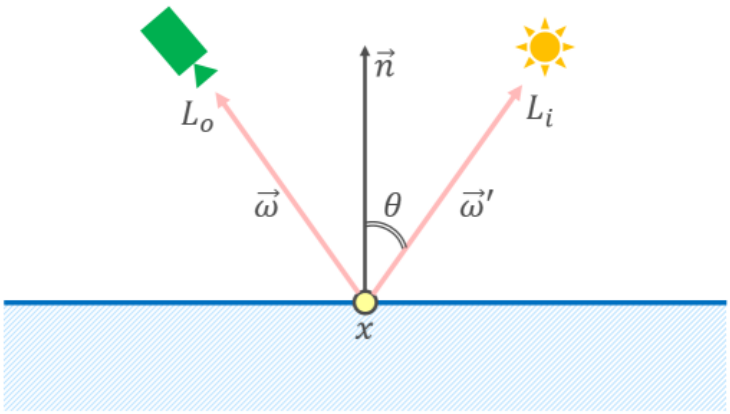
点 x から $\vec{\omega}$ 方向に出射される（outgoing）放射輝度 L_o は，放射される（emitted）放射輝度 L_e と，反射される（reflected）放射輝度 L_r の和です．

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + L_r(x,\vec{\omega}) \tag{80}$$

点 x における放射照度と，$\vec{\omega}$' 方向にどれだけ反射するかを表す BRDF を用いて，$\vec{\omega}$ 方向に反射される放射輝度 $L_r(x,\vec{\omega})$ を書き直すと次のようになります．

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + \int_\Omega f_r(x,\vec{\omega}',\vec{\omega}) L_i(x,\vec{\omega}')(\vec{\omega}'\cdot\vec{n})d\vec{\omega} \tag{81}$$

これがレンダリング方程式です．



レンダリング方程式の反射成分を見てみると，BRDF，放射輝度，コサイン項の３つのパラメータで成り立っていることがわかります．

$$L_r(x,\vec{\omega}') = \int_\Omega \underbrace{f_r(x,\vec{\omega}',\vec{\omega})}_{\text{BRDF}}\ \underbrace{L_i(x,\vec{\omega}')}_{\text{放射輝度}}\ \underbrace{(\vec{\omega}'\cdot\vec{n})}_{\text{コサイン項}} d\vec{\omega}$$

5.1 物理ベースレンダリング方程式の例

これまでのまとめで，拡散反射と鏡面反射を考慮したレンダリング方程式を作ってみます．自己放射はしないとします．単純に拡散反射と鏡面反射を足すと次のようになります．

$$L_r(x,\vec{\omega}') = f_d(x,\vec{\omega}',\vec{\omega})L_i(x,\vec{\omega}')(\vec{\omega}'\cdot\vec{n}) + f_s(x,\vec{\omega}',\vec{\omega})L_i(x,\vec{\omega}')(\vec{\omega}'\cdot\vec{n}) \tag{82}$$

$$= (f_d(x,\vec{\omega}',\vec{\omega}) + f_s(x,\vec{\omega}',\vec{\omega}))L_i(x,\vec{\omega}')(\vec{\omega}'\cdot\vec{n}) \tag{83}$$

しかし，これではエネルギー保存則が満たされていません．放射照度のうち鏡面反射される量がどれくらいかは鏡面反射率 F_r で表せられ，鏡面反射率はフレネルの方程式から求まります．拡散反射は屈折光で屈折光が運ぶ光の量は $1-Fr$ でした．鏡面反射率 F_r を使ってエネルギー保存則を満たすようにすると，次のようになります．

$$L_r(x,\vec{\omega}') = (f_d(x,\vec{\omega}',\vec{\omega})(1-F_r) + f_s(x,\vec{\omega}',\vec{\omega})F_r)L_i(x,\vec{\omega}')(\vec{\omega}'\cdot\vec{n}) \tag{84}$$

鏡面反射率を求めるのは簡単ではないので，思い切って拡散反射成分と鏡面反射成分の割合をパラメータ化します．つまり $F_r = k$ とし，以下のようにします．

$$L_r(x,\vec{\omega}') = (f_d(x,\vec{\omega}',\vec{\omega})(1-k) + f_s(x,\vec{\omega}',\vec{\omega})k)L_i(x,\vec{\omega}')(\vec{\omega}'\cdot\vec{n}) \tag{85}$$

この k はメタルネスとかメタリックと呼ばれるパラメータです．金属か非金属かで直感的に操作することができます．マイクロファセット BRDF にはラフネスという物体表面の粗さを表すパラメータがあります．このメタルネスとラフネスは物理ベースレンダリングにおいて，とても重要なパラメータになっています．アンリアルエンジン４などで扱っている物理ベースのマテリアルには必ずあるパラメータです．

最終的なレンダリング方程式は次のようになります．

$$L_r(x,\vec{\omega}') = \int_\Omega (f_d(x,\vec{\omega}',\vec{\omega})(1-k) + f_s(x,\vec{\omega}',\vec{\omega})k)L_i(x,\vec{\omega}')(\vec{\omega}'\cdot\vec{n})d\vec{\omega} \tag{86}$$

6．実装

これは前回の「物理からはじめる物理ベースレンダリング」の実装編になります．WebGL の JS ライブラリ「Three.js」を使って，実際に物理ベースレンダリングを行います．Three.js には物理ベースのマテリアルが標準で用意されていますが，ここではシェーダを独自に作成してレンダリングします．シェーダ言語は GLSL です．実装編の読者は GLSL もしくは何らかのシェーダ言語でコーディングしたことがある人を想定しています．また，Three.js について説明を行いませんので，興味のある方は調べてみてください．

6.1 Three.js について

Three.js は JavaScript 3D ライブラリです．WebGL に対応しており，機能も充実しているため手軽に 3D シーンをレンダリングすることが出来ます．ライブラリは以下から入手することが出来ます．

https://threejs.org/

「download」をクリックすればサンプルコードを含めたソースコードをダウンロード出来ます．

6.2 ソースコード

実装するサンプルのソースコードは以下から入手することができます．

https://github.com/mebiusbox/pbr

今回のファイル構成は次のようになっています．

```
pbr/
    js/
    three.min.js
    Detector.js
    loadFiles.js
    libs/dat.gui.min.js
    libs/stats.min.js
    geometries/TeapotBufferGeometry.js
    controls/OrbitControls.js
    shaders/
    pbr_vert.glsl
    pbr_frag.glsl
    index.html
```

js フォルダの loadFiles.js は独自のスクリプトで，それ以外は Three.js ライブラリの一部です．

6.3 頂点シェーダ

まず大事なことですが，ライティング計算はカメラ座標系で行います．これは Three.js もライティング計算はカメラ座標系になっているからです．Three.js で独自のシェーダを使う場合は THREE.ShaderMaterial を使います．THREE.ShaderMaterial を使うと，カメラやオブジェクトから変換行列を計算して自動で渡してくれます．また，ジオメトリの頂点属性も位置や法線，UV 値を自動で追加してくれます．これは大変便利です．もし，接法線や従法線を頂点に追加した場合は自分で attribute を宣言する必要があります．また，自分で定義したい場合は THREE.RawShaderMaterial を使います．

以下は THREE.ShaderMaterial で作るとシェーダの先頭に自動で付与されるコードです．

```
precision highp float;
precision highp int;
#define SHADER_NAME ShaderMaterial
#define VERTEX_TEXTURES
#define GAMMA_FACTOR 2
#define MAX_BONES 0
#define BONE_TEXTURE
#define NUM_CLIPPING_PLANES 0
uniform mat4 modelMatrix;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform mat4 viewMatrix;
uniform mat3 normalMatrix;
uniform vec3 cameraPosition;
attribute vec3 position;
attribute vec3 normal;
attribute vec2 uv;
#ifdef USE_COLOR
    attribute vec3 color;
#endif
#ifdef USE_MORPHTARGETS
    attribute vec3 morphTarget0;
    attribute vec3 morphTarget1;
    attribute vec3 morphTarget2;
    attribute vec3 morphTarget3;
    #ifdef USE_MORPHNORMALS
        attribute vec3 morphNormal0;
        attribute vec3 morphNormal1;
    attribute vec3 morphNormal2;
    attribute vec3 morphNormal3;
    #else
        attribute vec3 morphTarget4;
        attribute vec3 morphTarget5;
        attribute vec3 morphTarget6;
        attribute vec3 morphTarget7;
    #endif
#endif
#ifdef USE_SKINNING
    attribute vec4 skinIndex;
    attribute vec4 skinWeight;
#endif
```

そして pbr_vert.glsl の内容は次のようになっています．

```
varying vec3 vViewPosition;
varying vec3 vNormal;
void main() {
  vec4 mvPosition = modelViewMatrix * vec4(position, 1.0);
```

```
  gl_Position = projectionMatrix * mvPosition;
  vViewPosition = -mvPosition.xyz;
  vNormal = normalMatrix * normal;
}
```

varying はフラグメントシェーダに渡す変数を定義します．vViewPosition は頂点位置から視点位置までのベクトルです．mvPosition は頂点座標をカメラ座標系に変換したもので，その値をマイナスにしたものを vViewPosition に入れています．カメラ座標系で原点はカメラ位置になりますので，頂点位置の符号を反転すると，頂点位置から原点へ向かうベクトル，つまり頂点位置からカメラに向かう視線ベクトルになります．vNormal は normalMatrix で法線を変換したものです．normalMatrix は modelViewMatrix の逆行列の転置行列が入っており，normalMatrix = transpose(inverse(modelViewMatrix)) と同じです．なぜ法線を modelViewMatrix で変換してはいけないのかはここでは説明しませんが，modelViewMatrix で変換すると正しい法線にならないからです．

6.4 フラグメントシェーダ

次にフラグメントシェーダです．ここで物理ベースレンダリングの処理を行っています．まずは，Three.js がフラグメントシェーダに自動で付与するコードは次のようになっています．

```
precision highp float;
precision highp int;
#define SHADER_NAME ShaderMaterial
#define GAMMA_FACTOR 2
#define NUM_CLIPPING_PLANES 0
#define UNION_CLIPPING_PLANES 0
uniform mat4 viewMatrix;
uniform vec3 cameraPosition;
#define TONE_MAPPING
#define saturate(a) clamp( a, 0.0, 1.0 )
uniform float toneMappingExposure;
uniform float toneMappingWhitePoint;
vec3 LinearToneMapping( vec3 color ) {
    return toneMappingExposure * color;
}
vec3 ReinhardToneMapping( vec3 color ) {
    color *= toneMappingExposure;
    return saturate( color / ( vec3( 1.0 ) + color ) );
}
#define Uncharted2Helper( x ) max( ( ( x * ( 0.15 * x + 0.10 * 0.50 ) + 0.20 * 0.02 ) / ( x * ( 0.15 * x
+ 0.50 ) + 0.20 * 0.30 ) ) - 0.02 / 0.30, vec3( 0.0 ) )
vec3 Uncharted2ToneMapping( vec3 color ) {
    color *= toneMappingExposure;
    return saturate( Uncharted2Helper( color ) / Uncharted2Helper( vec3( toneMappingWhitePoint ) ) );
}
vec3 OptimizedCineonToneMapping( vec3 color ) {
    color *= toneMappingExposure;
    color = max( vec3( 0.0 ), color - 0.004 );
    return pow( ( color * ( 6.2 * color + 0.5 ) ) / ( color * ( 6.2 * color + 1.7 ) + 0.06 ), vec3( 2.2 )
     );
}

vec3 toneMapping( vec3 color ) { return LinearToneMapping( color ); }

vec4 LinearToLinear( in vec4 value ) {
    return value;
}
vec4 GammaToLinear( in vec4 value, in float gammaFactor ) {
    return vec4( pow( value.xyz, vec3( gammaFactor ) ), value.w );
}
vec4 LinearToGamma( in vec4 value, in float gammaFactor ) {
```

```
    return vec4( pow( value.xyz, vec3( 1.0 / gammaFactor ) ), value.w );
}
vec4 sRGBToLinear( in vec4 value ) {
    return vec4( mix( pow( value.rgb * 0.9478672986 + vec3( 0.0521327014 ), vec3( 2.4 ) ), value.rgb *
    0.0773993808, vec3( lessThanEqual( value.rgb, vec3( 0.04045 ) ) ) ), value.w );
}
vec4 LinearTosRGB( in vec4 value ) {
    return vec4( mix( pow( value.rgb, vec3( 0.41666 ) ) * 1.055 - vec3( 0.055 ), value.rgb * 12.92, vec3(
     lessThanEqual( value.rgb, vec3( 0.0031308 ) ) ) ), value.w );
}
vec4 RGBEToLinear( in vec4 value ) {
    return vec4( value.rgb * exp2( value.a * 255.0 - 128.0 ), 1.0 );
}
vec4 LinearToRGBE( in vec4 value ) {
    float maxComponent = max( max( value.r, value.g ), value.b );
    float fExp = clamp( ceil( log2( maxComponent ) ), -128.0, 127.0 );
    return vec4( value.rgb / exp2( fExp ), ( fExp + 128.0 ) / 255.0 );
}
vec4 RGBMToLinear( in vec4 value, in float maxRange ) {
    return vec4( value.xyz * value.w * maxRange, 1.0 );
}
vec4 LinearToRGBM( in vec4 value, in float maxRange ) {
    float maxRGB = max( value.x, max( value.g, value.b ) );
    float M      = clamp( maxRGB / maxRange, 0.0, 1.0 );
    M            = ceil( M * 255.0 ) / 255.0;
    return vec4( value.rgb / ( M * maxRange ), M );
}
vec4 RGBDToLinear( in vec4 value, in float maxRange ) {
    return vec4( value.rgb * ( ( maxRange / 255.0 ) / value.a ), 1.0 );
}
vec4 LinearToRGBD( in vec4 value, in float maxRange ) {
    float maxRGB = max( value.x, max( value.g, value.b ) );
    float D      = max( maxRange / maxRGB, 1.0 );
    D            = min( floor( D ) / 255.0, 1.0 );
    return vec4( value.rgb * ( D * ( 255.0 / maxRange ) ), D );
}
const mat3 cLogLuvM = mat3( 0.2209, 0.3390, 0.4184, 0.1138, 0.6780, 0.7319, 0.0102, 0.1130, 0.2969 );
vec4 LinearToLogLuv( in vec4 value )  {
    vec3 Xp_Y_XYZp = value.rgb * cLogLuvM;
    Xp_Y_XYZp = max(Xp_Y_XYZp, vec3(1e-6, 1e-6, 1e-6));
    vec4 vResult;
    vResult.xy = Xp_Y_XYZp.xy / Xp_Y_XYZp.z;
    float Le = 2.0 * log2(Xp_Y_XYZp.y) + 127.0;
    vResult.w = fract(Le);
    vResult.z = (Le - (floor(vResult.w*255.0))/255.0)/255.0;
    return vResult;
}
const mat3 cLogLuvInverseM = mat3( 6.0014, -2.7008, -1.7996, -1.3320, 3.1029, -5.7721, 0.3008, -1.0882,
5.6268 );
vec4 LogLuvToLinear( in vec4 value ) {
    float Le = value.z * 255.0 + value.w;
    vec3 Xp_Y_XYZp;
    Xp_Y_XYZp.y = exp2((Le - 127.0) / 2.0);
    Xp_Y_XYZp.z = Xp_Y_XYZp.y / value.y;
    Xp_Y_XYZp.x = value.x * Xp_Y_XYZp.z;
    vec3 vRGB = Xp_Y_XYZp.rgb * cLogLuvInverseM;
    return vec4( max(vRGB, 0.0), 1.0 );
}

vec4 mapTexelToLinear( vec4 value ) { return LinearToLinear( value ); }
vec4 envMapTexelToLinear( vec4 value ) { return LinearToLinear( value ); }
vec4 emissiveMapTexelToLinear( vec4 value ) { return LinearToLinear( value ); }
vec4 linearToOutputTexel( vec4 value ) { return LinearToLinear( value ); }
```

ビュー行列，カメラ位置，トーンマッピング，ガンマ変換，テクスチャの RGBE, RGBM, RGBD 形式の変換などが定義されています．

6.5 幾何情報

ライティングの計算に必要なものは，「物体表面の位置と法線，視線への向き」「放射照度」「物体表面の材質」です．その内「物体表面の位置と法線，視線への向き」は幾何情報のことで，次のように定義します．

```
struct GeometricContext {
  vec3 position;
  vec3 normal;
  vec3 viewDir;
};
```

position は位置，normal は法線，viewDir は視線への向きです．頂点シェーダから渡される vViewPosition と vNormal で幾何情報を設定します．

```
GeometricContext geometry;
geometry.position = -vViewPosition;
geometry.normal = normalize(vNormal);
geometry.viewDir = normalize(vViewPosition);
```

vViewPosition はカメラ座標での，表面位置からカメラへの視線ベクトルでした．それをまたマイナスにすると表面位置になります．そして，視線ベクトルと法線は正規化しておきます．

6.5.1 放射照度

レンダリング方程式では放射照度（放射輝度とコサイン項の積）が必要です．今回扱う光源は平行光源，点光源，スポットライトです．この光源からある点（＝１ピクセル）に入ってくる放射輝度を求めます．平行光源からの入射光はどの位置からでも入射してくる方向は同じですが，点光源とスポットライトは光源の位置から入射してきます．これらの光源は物体表面に光源の強さ分の放射輝度が放射されるように設定しています．ですが，例えば拡散反射のランバートモデルで考えてみると，π で割られるので，光源の強さよりも弱い放射輝度が入射することになります．結論として，これらの光源に対して，π を掛けることで調整します．この平行光源や点光源，スポットライトのことを punctual light と言います．
入射光（ここでは光源から入ってくる光）は IncidentLight と呼びました．入射光は入射してくる向き，光の波長（＝色）の情報を持っています．これを定義すると

```
struct IncidentLight {
  vec3 direction;
  vec3 color;
};
```

となります．平行光源，点光源，スポットライトから入射光を求められれば，入射光から放射照度の計算とを分けることができます．点光源とスポットライトは減衰しますので，光源の位置と物体表面の位置から減衰率を求める必要があります．また，光源から光が届かないとわかれば，その光源のライティング計算をする必要がなくなりますので，IncidentLight に visible 項目を追加し，光が届くときに真となるようにします．
まず，平行光源，点光源，スポットライトの定義をします．

```
struct DirectionalLight {
  vec3 direction;
  vec3 color;
};
```

```
struct PointLight {
  vec3 position;
  vec3 color;
  float distance;
  float decay;
};

struct SpotLight {
  vec3 position;
  vec3 direction;
  vec3 color;
  float distance;
  float decay;
  float coneCos;
  float penumbraCos;
};
```

position は光源の位置，color は光の色，distance は光源からの光が届く距離，decay は減衰率，coneCos は光源から出射する光線の幅，penumbraCos は光源から出射する光線の減衰幅です．coneCos と penumbraCos はシェーダに渡すときに Math.cos で計算したものになっています．

各光源から入射光を求めるコードは次のようになっています．

```
bool testLightInRange(const in float lightDistance, const in float cutoffDistance) {
  return any(bvec2(cutoffDistance == 0.0, lightDistance < cutoffDistance));
}

float punctualLightIntensityToIrradianceFactor(const in float lightDistance, const in float
cutoffDistance, const in float decayExponent) {
  if (decayExponent > 0.0) {
    return pow(saturate(-lightDistance / cutoffDistance + 1.0), decayExponent);
  }

  return 1.0;
}

void getDirectionalDirectLightIrradiance(const in DirectionalLight directionalLight, const in
GeometricContext geometry, out IncidentLight directLight) {
  directLight.color = directionalLight.color;
  directLight.direction = directionalLight.direction;
  directLight.visible = true;
}

void getPointDirectLightIrradiance(const in PointLight pointLight, const in GeometricContext geometry,
out IncidentLight directLight) {
  vec3 L = pointLight.position - geometry.position;
  directLight.direction = normalize(L);

  float lightDistance = length(L);
  if (testLightInRange(lightDistance, pointLight.distance)) {
    directLight.color = pointLight.color;
    directLight.color *= punctualLightIntensityToIrradianceFactor(lightDistance, pointLight.distance,
    pointLight.decay);
    directLight.visible = true;
  } else {
    directLight.color = vec3(0.0);
    directLight.visible = false;
  }
}

void getSpotDirectLightIrradiance(const in SpotLight spotLight, const in GeometricContext geometry, out
IncidentLight directLight) {
  vec3 L = spotLight.position - geometry.position;
  directLight.direction = normalize(L);
```

```
  float lightDistance = length(L);
  float angleCos = dot(directLight.direction, spotLight.direction);

  if (all(bvec2(angleCos > spotLight.coneCos, testLightInRange(lightDistance, spotLight.distance)))) {
    float spotEffect = smoothstep(spotLight.coneCos, spotLight.penumbraCos, angleCos);
    directLight.color = spotLight.color;
    directLight.color *= spotEffect * punctualLightIntensityToIrradianceFactor(lightDistance, spotLight.
    distance, spotLight.decay);
    directLight.visible = true;
  } else {
    directLight.color = vec3(0.0);
    directLight.visible = false;
  }
}
```

光源からの光が届くかどうかは testLightInRange で，光源からの減衰率計算は punctualLightIntensityToIrradianceFactor で行っています．各光源から求めた入射光に，π を掛けて，さらにコサイン項を掛けると放射輝度になります．

6.5.2 物体表面の材質

物体表面の材質は「入射光のうち拡散反射になる割合＝ディフューズリフレクタンス」「入射光のうち鏡面反射になる割合＝スペキュラーリフレクタンス」「物体表面の粗さ」になります．これを定義すると次のようになります．

```
struct Material {
  vec3 diffuseColor;
  vec3 specularColor;
  float specularRoughness;
};
```

diffuseColor はディフューズリフレクタンス，specularColor はスペキュラーリフレクタンス，specularRoughness は表面の粗さです．

これらは，metallic，roughness，albedo というパラメータから計算します．metallic は金属か非金属かを直感的に設定するパラメータで，鏡面反射率です．roughness は物体表面の粗さです．albedo は波長ごとの反射能で，つまりは色（RGB）ごとの反射能です．この当たりのコードは次のようになっています．

```
Material material;
material.diffuseColor = mix(albedo, vec3(0.0), metallic);
material.specularColor = mix(vec3(0.04), albedo, metallic);
material.specularRoughness = roughness;
```

スペキュラーリフレクタンスの 0.04 は，非金属でも 4%(0.04) は鏡面反射させるということです．4% という数字は一般的な不導体をカバーするそうです．

6.5.3 BRDF

次に拡散反射 BRDF と鏡面反射 BRDF の実装です．ここでは前回説明した BRDF を使います．つまり，拡散反射 BRDF には正規化ランバートモデルを，鏡面反射 BRDF にはクック・トランスのマイクロファセットモデルを使います．マイクロファセットモデルの分布関数は GGX, 幾何減衰項は Smith モデルの Schlick 近似，フレネルには Schlick の近似式を使います．

各数式をまとめておくと

拡散反射 BRDF（正規化ランバートモデル）

$$f_d = \frac{\rho_d}{\pi}$$

鏡面反射 BRDF（クック・トランスのマイクロファセットモデル）

$$f_r = \frac{D(h)G(l,v)F(v,h)}{4(n\cdot l)(n\cdot v)}$$

マイクロファセット分布関数

$$D_{GGX}(h) = \frac{\alpha^2}{\pi((n\cdot h)^2(\alpha^2-1)+1)^2}$$

幾何減衰項

$$k = \frac{\alpha}{2} \tag{87}$$

$$G_{Schlick}(v) = \frac{n\cdot v}{(n\cdot v)(1-k)+k} \tag{88}$$

$$G_{Smith}(l,v) = G_{Schlick}(l)G_{Schlick}(v) \tag{89}$$

フレネル

$$F_{Schlick}(v,h) = F_0 + (1-F_0)(1-v\cdot h)^5$$

これらをコード化すると次のようになります.

```
// Normalized Lambert
vec3 DiffuseBRDF(vec3 diffuseColor) {
  return diffuseColor / PI;
}

vec3 F_Schlick(vec3 specularColor, vec3 H, vec3 V) {
  return (specularColor + (1.0 - specularColor) * pow(1.0 - saturate(dot(V,H)), 5.0));
}

float D_GGX(float a, float dotNH) {
  float a2 = a*a;
  float dotNH2 = dotNH*dotNH;
  float d = dotNH2 * (a2 - 1.0) + 1.0;
  return a2 / (PI * d * d);
}

float G_Smith_Schlick_GGX(float a, float dotNV, float dotNL) {
  float k = a*a*0.5 + EPSILON;
  float gl = dotNL / (dotNL * (1.0 - k) + k);
  float gv = dotNV / (dotNV * (1.0 - k) + k);
  return gl*gv;
}

// Cook-Torrance
vec3 SpecularBRDF(const in IncidentLight directLight, const in GeometricContext geometry, vec3
specularColor, float roughnessFactor) {

  vec3 N = geometry.normal;
  vec3 V = geometry.viewDir;
  vec3 L = directLight.direction;

  float dotNL = saturate(dot(N,L));
  float dotNV = saturate(dot(N,V));
  vec3 H = normalize(L+V);
  float dotNH = saturate(dot(N,H));
  float dotVH = saturate(dot(V,H));
```

```
  float dotLV = saturate(dot(L,V));
  float a = roughnessFactor * roughnessFactor;

  float D = D_GGX(a, dotNH);
  float G = G_Smith_Schlick_GGX(a, dotNV, dotNL);
  vec3 F = F_Schlick(specularColor, V, H);
  return (F*(G*D))/(4.0*dotNL*dotNV+EPSILON);
}
```

所々に EPSILON があります．これはゼロ除算を防ぐためのおまじないと思ってください．

6.5.4 レンダリング方程式

点 x から $\vec{\omega}$ 方向に出射される（outgoing）放射輝度 L_o は，放射される（emitted）放射輝度 L_e と，反射される（reflected）放射輝度 L_r の和でした．

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + L_r(x,\vec{\omega})$$

そして，レンダリング方程式は以下のとおりです．

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + \int_\Omega f_r(x,\vec{\omega}',\vec{\omega}) L_i(x,\vec{\omega}')(\vec{\omega}'\cdot\vec{n})d\vec{\omega}$$

レンダリング方程式の反射成分は，BRDF，放射輝度，コサイン項です．コサイン項は物体表面の法線とライトベクトルの内積なので，幾何情報と入射光から求まります．これで必要な情報は全て揃いました．

$$L_r(x,\vec{\omega}') = \int_\Omega \underbrace{f_r(x,\vec{\omega}',\vec{\omega})}_{\text{BRDF}} \underbrace{L_i(x,\vec{\omega}')}_{\text{放射輝度}} \underbrace{(\vec{\omega}'\cdot\vec{n})}_{\text{コサイン項}} d\vec{\omega}$$

反射成分は光源の数だけ求めることになります．反射される放射輝度を ReflectedLight として定義します．

```
struct ReflectedLight {
  vec3 directDiffuse;
  vec3 directSpecular;
};
```

directDiffuse は拡散反射成分，directSpecular は鏡面反射成分です．また，将来的な話になりますがグローバルイルミネーションも考慮すると，間接光による拡散反射成分と鏡面反射成分も含まれます．

```
struct ReflectedLight {
  vec3 directDiffuse;
  vec3 directSpecular;
  vec3 indirectDiffuse;
  vec3 indirectSpecular;
};
```

まとめると，レンダリング方程式を解くために必要なパラメータ「入射光」「幾何情報」「物体表面の材質」を渡して，「反射光」を求める処理を実装します．これは次のようになります．

```
// RenderEquations(RE)
void RE_Direct(const in IncidentLight directLight, const in GeometricContext geometry, const in Material
material, inout ReflectedLight reflectedLight) {
```

```
  float dotNL = saturate(dot(geometry.normal, directLight.direction));
  vec3 irradiance = dotNL * directLight.color;

  // punctual light
  irradiance *= PI;

  reflectedLight.directDiffuse += irradiance * DiffuseBRDF(material.diffuseColor);
  reflectedLight.directSpecular += irradiance * SpecularBRDF(directLight, geometry, material.
  specularColor, material.specularRoughness);
}
```

1つ1つ見ていくと

```
float dotNL = saturate(dot(geometry.normal, directLight.direction));
```

ここでコサイン項を計算しています.

```
vec3 irradiance = dotNL * directLight.color;
```

放射輝度とコサイン項を掛けて放射照度に変換しています.

```
// punctual light
irradiance *= PI;
```

puncutal light には π を掛けて調整します.

```
reflectedLight.directDiffuse += irradiance * DiffuseBRDF(material.diffuseColor);
reflectedLight.directSpecular += irradiance * SpecularBRDF(directLight, geometry, material.specularColor,
 material.specularRoughness);
```

拡散反射成分と鏡面反射成分を計算します.

そして，これを光源の数だけ処理を行います.

```
ReflectedLight reflectedLight = ReflectedLight(vec3(0.0), vec3(0.0), vec3(0.0), vec3(0.0));
vec3 emissive = vec3(0.0);
float opacity = 1.0;

IncidentLight directLight;

// point light
for (int i=0; i<LIGHT_MAX; ++i) {
  if (i >= numPointLights) break;
  getPointDirectLightIrradiance(pointLights[i], geometry, directLight);
  if (directLight.visible) {
    RE_Direct(directLight, geometry, material, reflectedLight);
  }
}

// spot light
for (int i=0; i<LIGHT_MAX; ++i) {
  if (i >= numSpotLights) break;
  getSpotDirectLightIrradiance(spotLights[i], geometry, directLight);
  if (directLight.visible) {
    RE_Direct(directLight, geometry, material, reflectedLight);
  }
}

// directional light
```

```
for (int i=0; i<LIGHT_MAX; ++i) {
  if (i >= numDirectionalLights) break;
  getDirectionalDirectLightIrradiance(directionalLights[i], geometry, directLight);
  RE_Direct(directLight, geometry, material, reflectedLight);
}
```

最後に自己放射と反射成分の総和が，最終的に出力される放射輝度になります．

```
vec3 outgoingLight = emissive + reflectedLight.directDiffuse + reflectedLight.directSpecular +
reflectedLight.indirectDiffuse + reflectedLight.indirectSpecular;
gl_FragColor = vec4(outgoingLight, opacity);
```

6.5.5 最終コード

フラグメントシェーダの内容は以下のとおりです．

```
varying vec3 vViewPosition;
varying vec3 vNormal;

// uniforms
uniform float metallic;
uniform float roughness;
uniform vec3 albedo;

// defines
#define PI 3.14159265359
#define PI2 6.28318530718
#define RECIPROCAL_PI 0.31830988618
#define RECIPROCAL_PI2 0.15915494
#define LOG2 1.442695
#define EPSILON 1e-6

struct IncidentLight {
  vec3 color;
  vec3 direction;
  bool visible;
};

struct ReflectedLight {
  vec3 directDiffuse;
  vec3 directSpecular;
  vec3 indirectDiffuse;
  vec3 indirectSpecular;
};

struct GeometricContext {
  vec3 position;
  vec3 normal;
  vec3 viewDir;
};

struct Material {
  vec3 diffuseColor;
  float specularRoughness;
  vec3 specularColor;
};

// lights

bool testLightInRange(const in float lightDistance, const in float cutoffDistance) {
  return any(bvec2(cutoffDistance == 0.0, lightDistance < cutoffDistance));
}
```

```
float punctualLightIntensityToIrradianceFactor(const in float lightDistance, const in float
cutoffDistance, const in float decayExponent) {
  if (decayExponent > 0.0) {
    return pow(saturate(-lightDistance / cutoffDistance + 1.0), decayExponent);
  }

  return 1.0;
}

struct DirectionalLight {
  vec3 direction;
  vec3 color;
};

void getDirectionalDirectLightIrradiance(const in DirectionalLight directionalLight, const in
GeometricContext geometry, out IncidentLight directLight) {
  directLight.color = directionalLight.color;
  directLight.direction = directionalLight.direction;
  directLight.visible = true;
}

struct PointLight {
  vec3 position;
  vec3 color;
  float distance;
  float decay;
};

void getPointDirectLightIrradiance(const in PointLight pointLight, const in GeometricContext geometry,
out IncidentLight directLight) {
  vec3 L = pointLight.position - geometry.position;
  directLight.direction = normalize(L);

  float lightDistance = length(L);
  if (testLightInRange(lightDistance, pointLight.distance)) {
    directLight.color = pointLight.color;
    directLight.color *= punctualLightIntensityToIrradianceFactor(lightDistance, pointLight.distance,
    pointLight.decay);
    directLight.visible = true;
  } else {
    directLight.color = vec3(0.0);
    directLight.visible = false;
  }
}

struct SpotLight {
  vec3 position;
  vec3 direction;
  vec3 color;
  float distance;
  float decay;
  float coneCos;
  float penumbraCos;
};

void getSpotDirectLightIrradiance(const in SpotLight spotLight, const in GeometricContext geometry, out
IncidentLight directLight) {
  vec3 L = spotLight.position - geometry.position;
  directLight.direction = normalize(L);

  float lightDistance = length(L);
  float angleCos = dot(directLight.direction, spotLight.direction);

  if (all(bvec2(angleCos > spotLight.coneCos, testLightInRange(lightDistance, spotLight.distance)))) {
    float spotEffect = smoothstep(spotLight.coneCos, spotLight.penumbraCos, angleCos);
    directLight.color = spotLight.color;
    directLight.color *= spotEffect * punctualLightIntensityToIrradianceFactor(lightDistance, spotLight.
```

```
    distance, spotLight.decay);
    directLight.visible = true;
  } else {
    directLight.color = vec3(0.0);
    directLight.visible = false;
  }
}

// light uniforms
#define LIGHT_MAX 4
uniform DirectionalLight directionalLights[LIGHT_MAX];
uniform PointLight pointLights[LIGHT_MAX];
uniform SpotLight spotLights[LIGHT_MAX];
uniform int numDirectionalLights;
uniform int numPointLights;
uniform int numSpotLights;

// BRDFs

// Normalized Lambert
vec3 DiffuseBRDF(vec3 diffuseColor) {
  return diffuseColor / PI;
}

vec3 F_Schlick(vec3 specularColor, vec3 H, vec3 V) {
  return (specularColor + (1.0 - specularColor) * pow(1.0 - saturate(dot(V,H)), 5.0));
}

float D_GGX(float a, float dotNH) {
  float a2 = a*a;
  float dotNH2 = dotNH*dotNH;
  float d = dotNH2 * (a2 - 1.0) + 1.0;
  return a2 / (PI * d * d);
}

float G_Smith_Schlick_GGX(float a, float dotNV, float dotNL) {
  float k = a*a*0.5 + EPSILON;
  float gl = dotNL / (dotNL * (1.0 - k) + k);
  float gv = dotNV / (dotNV * (1.0 - k) + k);
  return gl*gv;
}

// Cook-Torrance
vec3 SpecularBRDF(const in IncidentLight directLight, const in GeometricContext geometry, vec3
specularColor, float roughnessFactor) {

  vec3 N = geometry.normal;
  vec3 V = geometry.viewDir;
  vec3 L = directLight.direction;

  float dotNL = saturate(dot(N,L));
  float dotNV = saturate(dot(N,V));
  vec3 H = normalize(L+V);
  float dotNH = saturate(dot(N,H));
  float dotVH = saturate(dot(V,H));
  float dotLV = saturate(dot(L,V));
  float a = roughnessFactor * roughnessFactor;

  float D = D_GGX(a, dotNH);
  float G = G_Smith_Schlick_GGX(a, dotNV, dotNL);
  vec3 F = F_Schlick(specularColor, V, H);
  return (F*(G*D))/(4.0*dotNL*dotNV+EPSILON);
}

// RenderEquations(RE)
void RE_Direct(const in IncidentLight directLight, const in GeometricContext geometry, const in Material
material, inout ReflectedLight reflectedLight) {
```

```
  float dotNL = saturate(dot(geometry.normal, directLight.direction));
  vec3 irradiance = dotNL * directLight.color;

  // punctual light
  irradiance *= PI;

  reflectedLight.directDiffuse += irradiance * DiffuseBRDF(material.diffuseColor);
  reflectedLight.directSpecular += irradiance * SpecularBRDF(directLight, geometry, material.
  specularColor, material.specularRoughness);
}

void main() {
  GeometricContext geometry;
  geometry.position = -vViewPosition;
  geometry.normal = normalize(vNormal);
  geometry.viewDir = normalize(vViewPosition);

  Material material;
  material.diffuseColor = mix(albedo, vec3(0.0), metallic);
  material.specularColor = mix(vec3(0.04), albedo, metallic);
  material.specularRoughness = roughness;

  // Lighting

  ReflectedLight reflectedLight = ReflectedLight(vec3(0.0), vec3(0.0), vec3(0.0), vec3(0.0));
  vec3 emissive = vec3(0.0);
  float opacity = 1.0;

  IncidentLight directLight;

  // point light
  for (int i=0; i<LIGHT_MAX; ++i) {
    if (i >= numPointLights) break;
    getPointDirectLightIrradiance(pointLights[i], geometry, directLight);
    if (directLight.visible) {
      RE_Direct(directLight, geometry, material, reflectedLight);
    }
  }

  // spot light
  for (int i=0; i<LIGHT_MAX; ++i) {
    if (i >= numSpotLights) break;
    getSpotDirectLightIrradiance(spotLights[i], geometry, directLight);
    if (directLight.visible) {
      RE_Direct(directLight, geometry, material, reflectedLight);
    }
  }

  // directional light
  for (int i=0; i<LIGHT_MAX; ++i) {
    if (i >= numDirectionalLights) break;
    getDirectionalDirectLightIrradiance(directionalLights[i], geometry, directLight);
    RE_Direct(directLight, geometry, material, reflectedLight);
  }

  vec3 outgoingLight = emissive + reflectedLight.directDiffuse + reflectedLight.directSpecular +
  reflectedLight.indirectDiffuse + reflectedLight.indirectSpecular;

  gl_FragColor = vec4(outgoingLight, opacity);
}
```

6.6 サンプル

動作するサンプルを用意しました．Three.js には物理ベースレンダリングを行うマテリアルがありますが，それと切り替えられるようになっています．フレネル関係で多少違うところがありますが，ほとんど変わらないはずです．

http://mebiusbox.github.io/contents/pbr/

7．最後に

光の基本から始まり，放射照度，放射輝度といった光の物理的なエネルギー，光の反射特性，BRDF，そして，レンダリング方程式と説明してきました．物理ベースレンダリングとは結局のところ，物体の表面に当たった放射エネルギー（放射照度）が，ある方向（カメラや目）に向かって反射する放射エネルギー（放射輝度）を計算するときに，相反性とエネルギー保存則を満たした BRDF の反射モデルを使っていることということになるでしょうか．ただし，この２つの物理的性質を満たしていても，物理的に正しいというわけではありません．

今回は拡散反射モデルと鏡面反射モデルを使った物理ベースレンダリングを実装しました．これで最低限の物理ベースレンダリングが実装されたことになったと思います．ここから，テクスチャを貼って質感を上げたり，拡散反射モデルにオーレン・ネイヤーモデルを使ったり，鏡面反射モデルは異方性を扱ったものなど多くの反射モデルがあります．また，グローバルイルミネーションでライトプローブやイラディアンスボリュームを使って間接光の拡散反射を計算した場合は reflectedLight.indirectDiffuse に，間接光の鏡面反射によって周りの景色が映り込むようなものは reflectedLight.indirectSpecular に入ってくるでしょう．シャドウマップなどで影の処理をする場合は光源から入射光を求めたときに，incidentLight.color に影の係数を掛けてから反射される輝度を求めるといったことが出来ます．

物理ベースレンダリングの基礎から実装まで，解説を書いてみましたが，いかがだったでしょうか．これからも勉強して理解を深めつつ，記事の修正・加筆，新しい内容について書いていこうかなと思います．少しでも参考になれば幸いです．

参考文献

[1] Robert L. Cook, Kenneth E. Torrance「A Reflection Model for Computer Graphics」, 1982

[2] Bruce Walter, Stephen R. Marschner, Hongsong Li, Kenneth E. Torrance「Microfacet Models for Refraction through Rough Surfaces」, 2007

[3] Brian Karis「Real Shading in Unreal Engine 4」, 2013

[4] Wes McDermott「The Comprehensive PBR Guide by Allegorithmic - vol. 1 Light and Matter : The theory of Physically-Based Rendering and Shading」, https://www.allegorithmic.com/pbr-guide

[5] Wes McDermott「The Comprehensive PBR Guide by Allegorithmic - vol. 2 Light and Matter : Practical guidelines for creating PBR textures」, https://www.allegorithmic.com/pbr-guide

[6] Wes McDermott 著，小林信行訳「総合 PBR ガイド by Allegorithmic - vol. 1 光と物質：物理ベースレンダリングとシェーディングの理論」, 2015, https://www.slideshare.net/nyaakobayashi/pbr-guide-vol1jp（２０１７年６月２９日閲覧）

[7] Wes McDermott 著，小林信行訳「総合 PBR ガイド by Allegorithmic - vol. 2 光と物質：PBR のテクスチャを作成するための実用的なガイドライン」, 2015, https://www.slideshare.net/nyaakobayashi/70100srgb180255（２０１７年６月２９日閲覧）

[8] Henrik Wann Jenson 著，苗村健訳「フォトンマッピングー実写に迫るコンピュータグラフィックス」オーム社，2002

[9] 倉地紀子「CG Magic：レンダリング」オーム社，2007

[10] CG-ARTS 協会「コンピュータグラフィックス」CG-ARTS 協会，2015

[11] CG-ARTS 協会「ディジタル画像処理」CG-ARTS 協会，2015

[12]「ニュートン別冊「光は電磁波」を実感 光とは何か？」ニュートンプレス，2007

[13] Eric Lengyel 著，狩野智英訳「ゲームプログラミングのための３ D グラフィックス数学」ボーンデジタル社，2006

[14] 西川善司「ゲーム制作者になるための３ D グラフィックス技術」インプレスジャパン，2009

[15] 向川康博「反射・散乱の計測とモデル化」『コンピュータビジョン最先端ガイド４』アドコム・メディア，2011，p.121

[16] Eugene Hecht 著，尾崎義治・朝倉利光訳「ヘクト 光学Ｉ ー基礎と幾何光学ー」丸善出版，2010

[17] CGWORLD，永田豊志「CG ＆映像しくみ事典」ワークスコーポレーション，2009

[18] JEREMY BIRN 著，株式会社 B スプラウト訳「ライティング＆レンダリング」ボーンデジタル，2014

[19] 山本醍田他「Computer Graphics Gems JP 2015 コンピュータグラフィックス技術の最前線」ボーンデジタル，2015

[20] pnlybubbles「物理ベースレンダリング 1」, http://qiita.com/pnlybubbles/items/416a3273cd245d1a1c04（２０１７年６月２９日閲覧）

[21] shikihuiku「物理ベースレンダリング (1)」, https://shikihuiku.wordpress.com/2012/07/03/%E7%89%A9%E7%90%86%E3%83%99%E3%83%BC%E3%82%B9%E3%83%AC%E3%83%B3%E3%83%80%E3%83%AA%E3%83%B3%E3%82%B01/（２０１７年６月２９日閲覧）

[22] shocker「放射輝度 (Radiance)」, http://rayspace.xyz/CG/contents/radiance.html（２０１７年６月２９日

閲覧）

[23] Ushio「レイトレーシングにおける放射輝度って結局何なのか？」, http://qiita.com/Ushio/items/e2752fa10e093a1b4cdd（２０１７年7月１日閲覧）

[24] hole「物理ベースレンダラ edupt 解説」, http://www.slideshare.net/h013/edupt-kaisetsu-22852235（２０１７年6月２９日閲覧）

[25] hole「双方向パストレーシングレンダラ edubpt 解説」, http://www.slideshare.net/h013/edubpt-v100（２０１７年6月２９日閲覧）

[26] A.R Khadka「BSDF」, https://anishrkhadkablog.wordpress.com/tag/bsdf/（２０１７年6月３０日閲覧）

[27] 五反田義治「本当のＨＤＲ表現へ〜アーティストにも知ってほしいリフレクタンスの基礎〜」, http://research.tri-ace.com/Data/BasicOfReflectance.ppt（２０１７年7月１日閲覧）

[28] 床井浩平「ゲームグラフィックス特論 第６回陰影付け」, http://www.wakayama-u.ac.jp/~tokoi/lecture/gg/ggnote06.pdf（２０１７年7月１日閲覧）

[29] 床井浩平「ゲームグラフィックス特論 第９回高度な陰影付け」, http://www.wakayama-u.ac.jp/~tokoi/lecture/gg/ggnote09.pdf（２０１７年7月１日閲覧）

[30] 山口裕也「FINAL FANTASY 零式 HD にみる新しい HD リマスター〜GPU 最適化編〜」

[31] 堀内敏行「光技術入門」東京電機大学出版局，2014

[32] Naty Hoffman「Background: Physics and Math of Shading」2013

[33] hanecci「従来のゲーム向けライト (ポイントライト, ディレクショナルライト) での BRDF の利用, http://d.hatena.ne.jp/hanecci/20130604/p1