第10章 陰影付け

10.1 陰影付けモデル

10.1.1 反射光と形状の知覚

ものの形は、照明の反射光によって視覚的に知覚することができます。反射光が全く無い状態では、ものの形を見ることはできません (図 115)。

だからと言って物体にでたらめに色を付けても、立体として知覚することはできません。反射光の強さは、現実に光源の光が物体表面で反射され、視点に到達するプロセスに沿って求める必要があります。陰影付けモデルは、この反射のプロセスをモデル化したものです。

反射光がないとき　　反射光があるとき

図 115 反射光の有無

10.1.2 光源のモデル化

3DCG、特に高速な処理が求められるゲームグラフィックスやリアルタイムレンダリングの領

域では、光源に平行光線、点光源、それにスポットライトの三種類がよく用いられます (図 116)。これらの光源の共通点は、いずれも大きさを持たない点であることです。平行光線は光源が無限の彼方にある場合であり、仮に光源が有限の大きさを持っていたとしても、光の反射面から見れば大きさを持ちません。またスポットライトは、光の拡散方向に指向性を持たせた点光源であると言えます。このような光源のモデルは **Omni Light (全方向光源)** といいます。

光源の大きさが点であれば、光源は光の放射面積を持たないことになります。光が姿の見えないところから放射されることは、現実にはあり得ません。しかし、このような光源のモデルでも、実用上は「それらしい」陰影を得ることができます。何より、このモデルは計算量が少なくすみます。光源の大きさを考慮に入れた陰影付けは、非常に時間がかかります。

その一方で、このようなおおざっぱな「近似」は、当然「誤差」を生みます。これは、生成された映像の CG 臭さ、リアリティの低さとして知覚されます。これは面光源などの光源の面積を考慮したモデルを用いることによって大きく改善されます。しかし、それがすべてではありません。また、リアリティを追求すればするほど計算量も増加します。リアリティとパフォーマンス (表示性能) は、トレードオフの関係にあります。

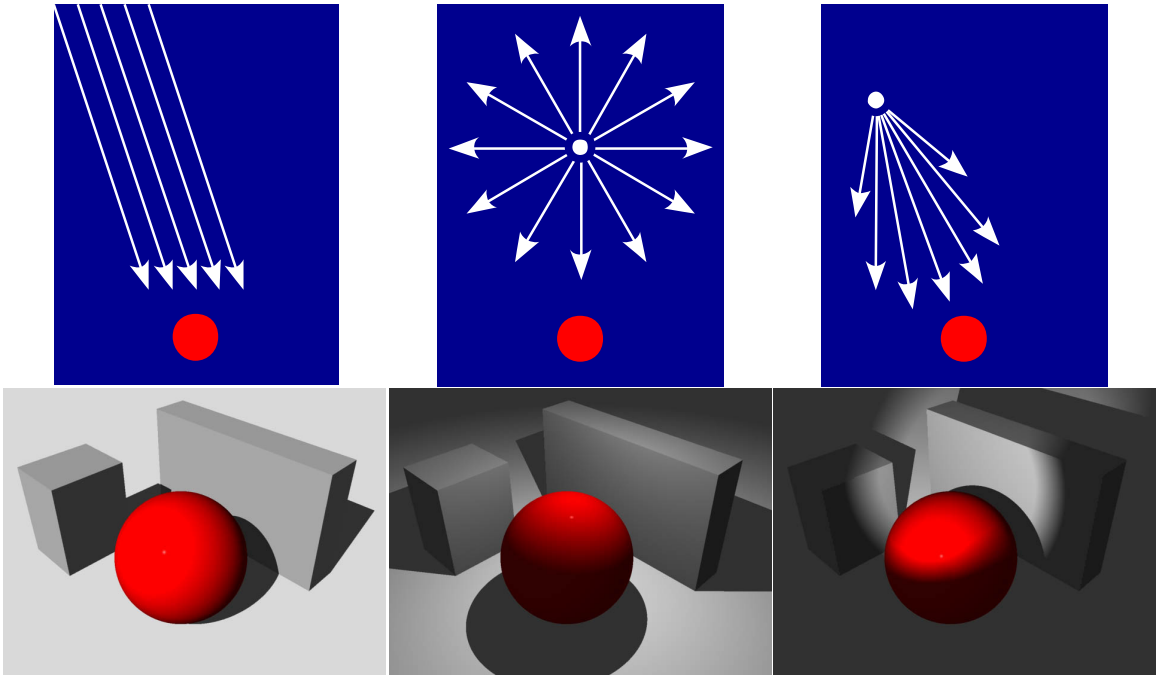


図 116 基本的な光源

10.1.3 二色性反射モデル

二色性反射モデルでは、反射光を、すべての方向に均等に放射される**拡散反射光**と、入射光の正反射方向を軸に反射する**鏡面反射光**の二つの成分の和として捉えます。

- **拡散反射光** (diffuse)

光源からの入射光は、良く知られている通り、物体表面、あるいは性質の異なる光の媒質の境界面において、屈折して内部に進入する成分と、正反射する成分に分かれます (図 117)。

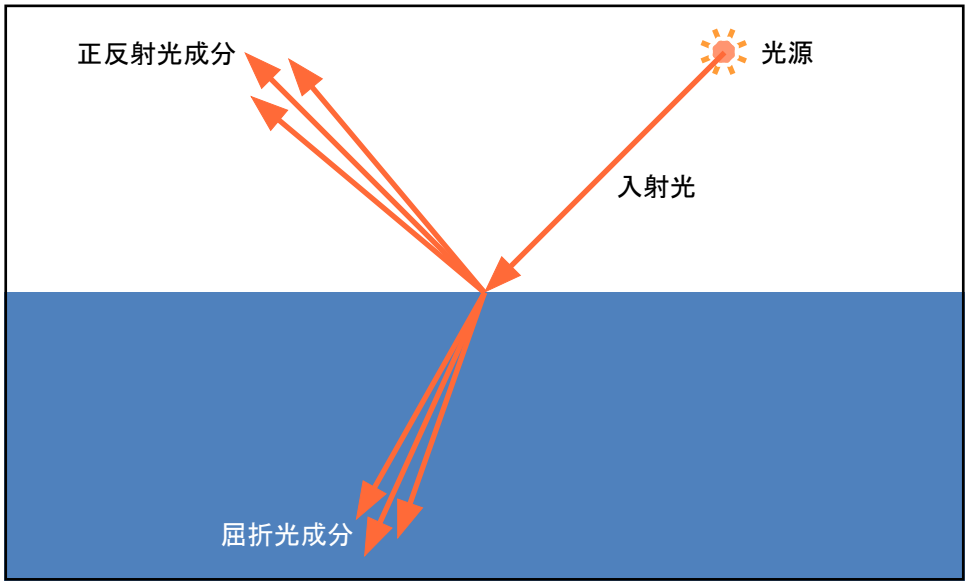


図 117 屈折光と正反射光

このうち屈折光の成分は、物体の材質が非金属であれば、物体の内部で**散乱**と**吸収**を繰り返した後、再び**光の入射点から**外部に放出されると考えます。これを**拡散反射光**といいます (図 118)。

ここで**散乱**は、光が異なる光学特性を持つ材質の境界面などの光学的な不連続性に出会うことで発生する現象であり、これによって光は向きを変えるものの、光の量は変化しないと考えます。これに対して**吸収**は、物質の内部で発生する現象であり、光が熱などの他の種類のエネルギーに変換されることによって、光が消失することをいいます。

拡散反射光は、物体内部の物体内部での吸収によって、入射光に含まれる波長の一部の成分が消失しています。この結果、拡散反射光には物体の色がつきます。また、この光は物体内部で散乱を繰り返したことによって指向性を失っており、すべての方向に対して均等に放射されます。

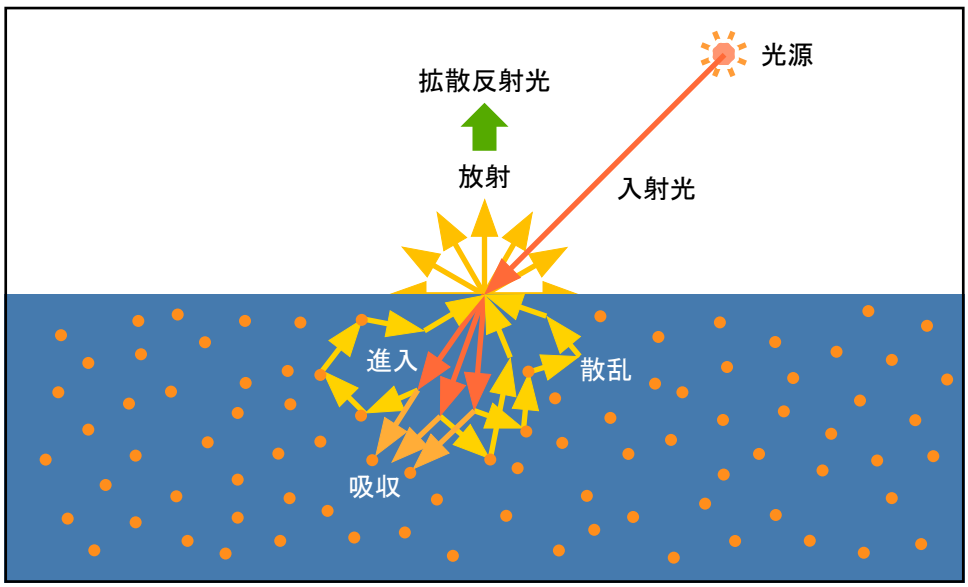


図 118 拡散反射光

拡散反射光の強度は入射光の強度に比例しますが、視点方向 (観測方向) には依存しません。また入射光の強度は、入射光の密度に比例します。いま、入射光を断面積 d の平行光線だと考えます。光の反射は物体表面の微視的な空間でモデル化されるので、光源が有限の位置にあっても、十分遠くにある (すなわち、平行光線である) と見なすことができます。

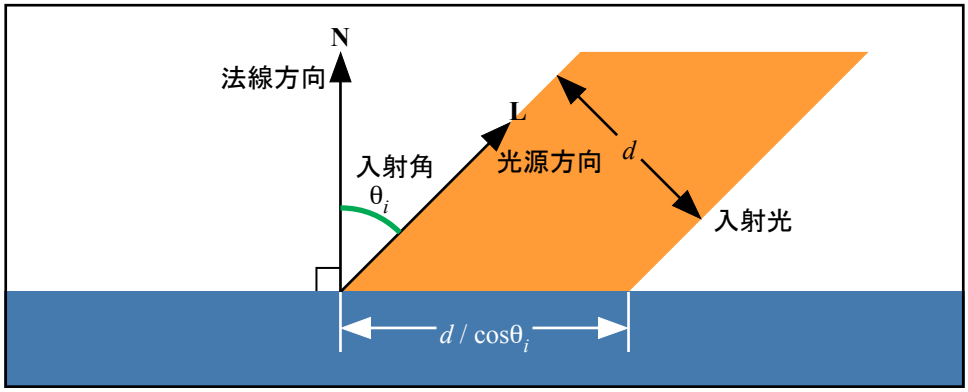


図 119 Lambert の余弦法則

いま、物体表面の法線方向に対する入射光の入射角を θ_i とすれば、断面積 d の光が照射する面積は $d/\cos\theta_i$ になります (図 119)。したがって、この入射光の物体表面における密度は $\cos\theta_i$ に比例します。これを **Lambert の余弦法則**と言います。物体表面の法線単位ベクトルを $\mathbf{N}$、入射光の方向を $\mathbf{L}$ とすれば、$\cos\theta_i = \mathbf{N}\cdot\mathbf{L}$ です。

したがって、物体表面の材質の拡散反射係数を K_{diff}、光源強度の拡散反射光成分を L_{diff} としたとき、拡散反射光の強度 I_{diff} は次式により求めることができます。

$$I_{diff} = \cos\theta_i K_{diff} \otimes L_{diff} = (\mathbf{N}\cdot\mathbf{L})K_{diff} \otimes L_{diff} \tag{82}$$

陰影付けを RGB 色空間で行うなら、この $K_{diff}, L_{diff}, I_{diff}$ はともに RGB の三つの要素を持ちます。$\otimes$ は RGB の要素ごとの積の演算子です。なお、L_{diff} は光源強度の拡散反射光成分ですが、実際の光源にそのような成分があるわけではありません。

$\mathbf{N}$ と $\mathbf{L}$ のなす角が $\pi/2$ を超えると $\cos\theta_i = \mathbf{N}\cdot\mathbf{L} < 0$ となってしまいますが、これは入射光が物体表面の裏側から入射していることになります。これは物体が不透明なら起こりませんし、半透明であっても別の処理になりますので、その場合はこれを 0 とします。

$$I_{diff} = \max(\mathbf{N}\cdot\mathbf{L}, 0)K_{diff} \otimes L_{diff} \tag{83}$$

図 120 にこのモデルによる拡散反射光の陰影の例を示します。

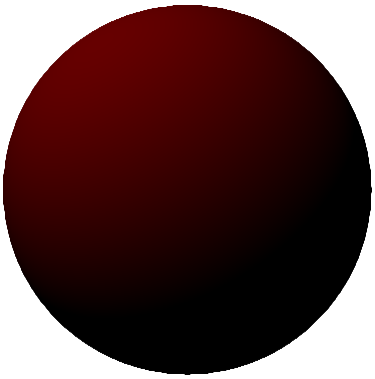


図 120 拡散反射光による陰影

● 鏡面反射光 (specular)

入射光が物体表面で正反射した成分のうち、視点に届くものを、鏡面反射光といいます。

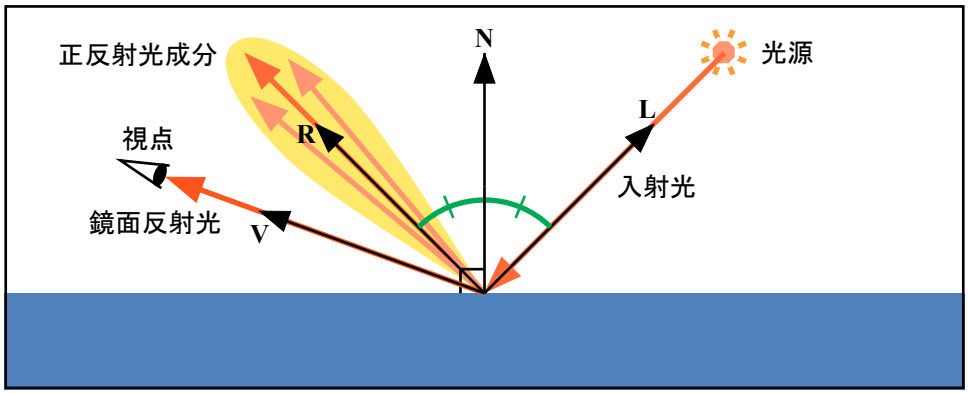


図 121 鏡面反射光

正反射光は、物体表面の粗さによって、入射光の正反射方向を軸に分布すると考えます。そうしないと、光源が点であると仮定しているために、視点が正反射方向に無ければ、鏡面反射光が観測されないことになってしまいます。

また、鏡面反射光は物体の表面での反射であり、光が物体の内部を経由しません。そのため、物体の材質が非金属であれば、反射光は物体の色の影響を受けず、光源の色がそのまま反映されます。これにより、物体表面上にハイライトを生成して、物体を輝かせて見せる効果を得ます。

いま、図 122 のように物体表面の法線単位ベクトルを $\mathbf{N}$ とし、光源が単位ベクトル $\mathbf{L}$ の方向にあるとすれば、$\mathbf{L}$ の正反射ベクトル $\mathbf{R}$ は次式で求めることができます。

$$\mathbf{R} = 2(\mathbf{N} \cdot \mathbf{L})\mathbf{N} - \mathbf{L} \tag{84}$$

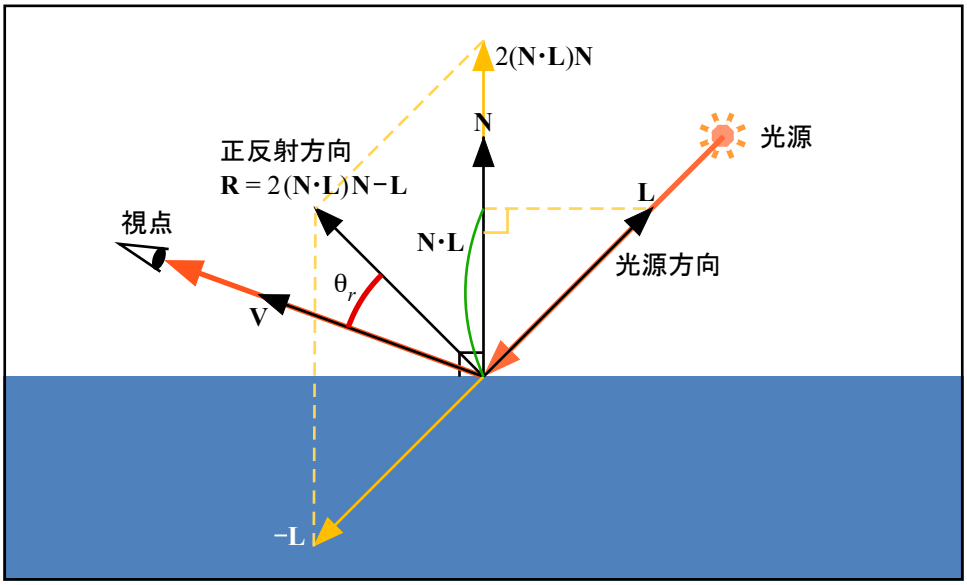


図 122 正反射方向

鏡面反射光の強度は、この **R** と視点の方向の単位ベクトル **V** とのなす角 θ_r にもとづいて決定します。**R** と **V** が近いほど反射光が強くなることを $\cos\theta_r = \mathbf{R}\cdot\mathbf{V}$ でモデル化します。さらに、正反射光の広がりを、このべき乗で表現します。

いま、物体表面の材質の鏡面反射係数を K_{spec}、光源強度の拡散反射光成分を L_{spec} としたとき、拡散反射光の強度 I_{spec} は次式により求めることができます。これを **Phong の陰影付けモデル**といいます。

$$I_{spec} = \cos^{K_{shi}}\theta_i K_{spec} \otimes L_{spec} = (\mathbf{R}\cdot\mathbf{V})^{K_{shi}} K_{spec} \otimes L_{spec} \tag{85}$$

ここで K_{shi} は正反射光の広がりを制御する係数で、これが大きいほど正反射光は鋭くなり、物体表面上に現れるハイライトは小さくなります。これを**輝き係数** (shininess) と呼びます。

また拡散反射光と同様に、**R** と **V** のなす角が $\pi/2$ を超えると $\cos\theta_r = \mathbf{R}\cdot\mathbf{V} < 0$ となってしまいますが、これも入射光が物体表面の裏側から入射していることになります。この場合も、これを 0 とします。

$$I_{spec} = \max(\mathbf{R}\cdot\mathbf{V}, 0)^{K_{shi}} K_{spec} \otimes L_{spec} \tag{86}$$

図 123 にこのモデルによる拡散反射光の陰影の例を示します。

図 123 Phong の陰影付けモデルによる鏡面反射光の陰影

鏡面反射光のモデルには、入射光の正反射光と視線方向との関係にもとづくもののほかに、図 124 のように光源の方向単位ベクトル **L** と視線の方向 **V** の中間ベクトル **H** と物体表面の法線単位ベクトル **N** とのなす角 θ_h にもとづいて決定する方法もあります。中間ベクトル **H** は、次式により求めることができます。

$$\mathbf{H} = \frac{\mathbf{L} + \mathbf{V}}{|\mathbf{L} + \mathbf{V}|} \tag{87}$$

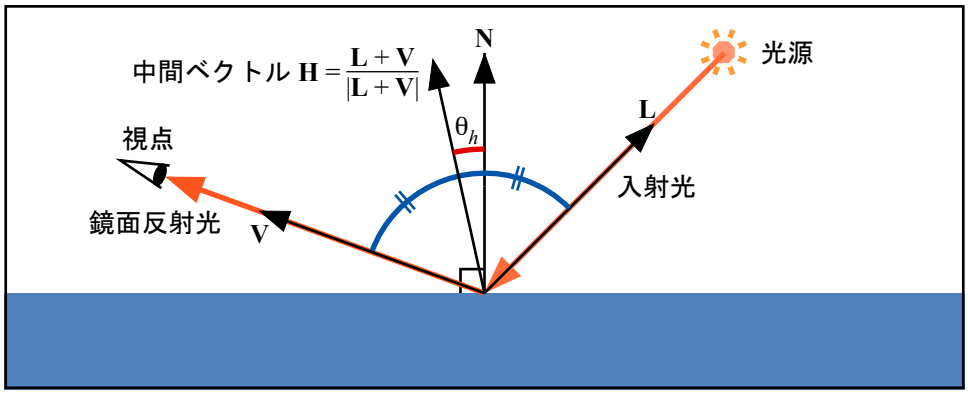


図 124 中間ベクトルにもとづく鏡面反射光

このモデルの鏡面反射光の強度は、この **H** と物体表面の法線単位ベクトル **N** とのなす角 θ_h にもとづいて決定します。これも **N** と **H** が近いほど反射光が強くなることを $\cos\theta_h = \mathbf{N}\cdot\mathbf{H}$ でモデル化し、正反射光の広がりをそのべき乗で表現します。これは **Blinn-Phong の陰影付けモデル**、あるいは単に **Blinn の陰影付けモデル**と呼ばれます。

$$I_{spec} = \cos^{K_{shi}}\theta_i K_{spec} \otimes L_{spec} = (\mathbf{N}\cdot\mathbf{H})^{K_{shi}} K_{spec} \otimes L_{spec} \tag{88}$$

これも **N** と **H** のなす角が $\pi/2$ を超えて $\cos\theta_h = \mathbf{R}\cdot\mathbf{V} < 0$ となる場合は 0 とします。これは視点か光源のどちらかが物体表面の背後にある場合です。

$$I_{spec} = \max(\mathbf{N}\cdot\mathbf{H}, 0)^{K_{shi}} K_{spec} \otimes L_{spec} \tag{89}$$

このモデルは OpenGL にプログラマブルシェーダが導入される前の固定機能パイプラインで

用いられていました。このモデルによる陰影を、図 125 に示します。

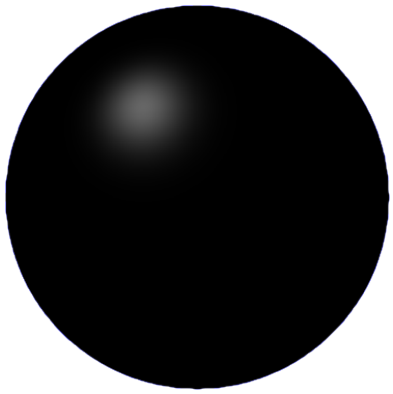


図 125 Blinn の陰影付けモデルによる鏡面反射光の陰影

Phong のモデルは、円形の光源の映り込みをモデル化したものだと捉えることができます。これに対して Blinn のモデルは、物体表面の凹凸の分布をモデル化したものだと捉えることができます。したがって、Phong のモデルはハイライトの形状は光源の形状を反映したものという予備知識と一致しますが、現実のハイライトの形状は Blinn のモデルに似たものになっています。

● **輝き係数** (shininess)

物体表面は完全な平面ではなく、微小な凹凸が存在します。鏡面反射光の広がりは、この凹凸による物体表面の滑らかさの知覚を制御します。

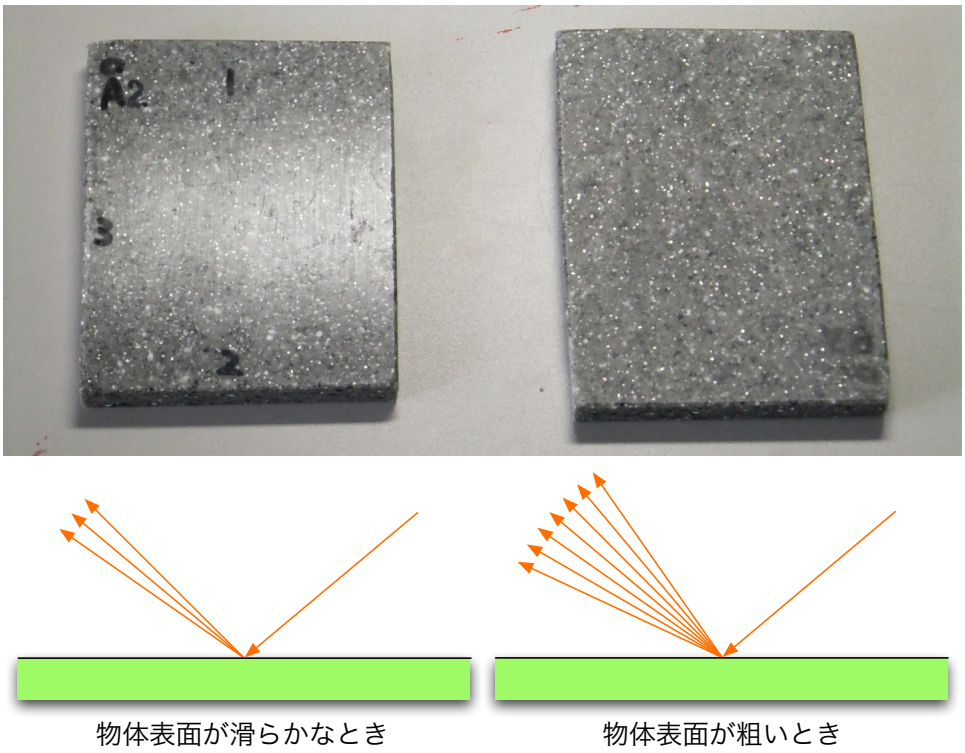


図 126 物体表面の滑らかさとハイライト

図 126 は砥粒の吹き付けによるマット仕上げを施した人造大理石 (樹脂) の表面を示します。表面が滑らかなほど、はっきりとしたハイライトが現れています。砥粒の吹き付けのような方法で付けられた物体表面の凹凸の方向分布は、ほぼ正規表現に従います。

鏡面反射光のモデル化に用いられている $\cos^n\theta$ のモデル、いわゆる cosine モデルの概形は、この正規分布に良く似ています。輝き係数 K_{shi} が大きくなるほど分布が急峻になります。

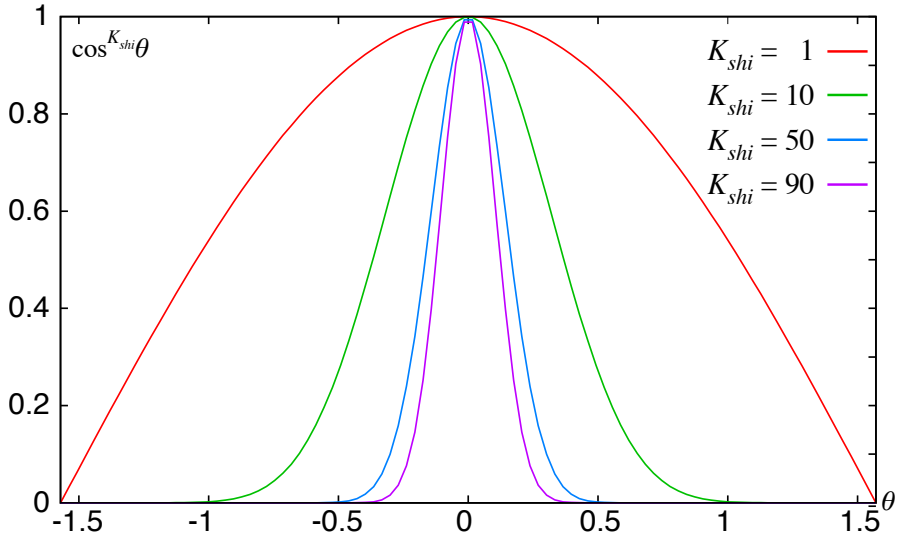


図 127 cosine モデル

このため、K_{shi} が大きくなるほど、ハイライトが小さくなります。

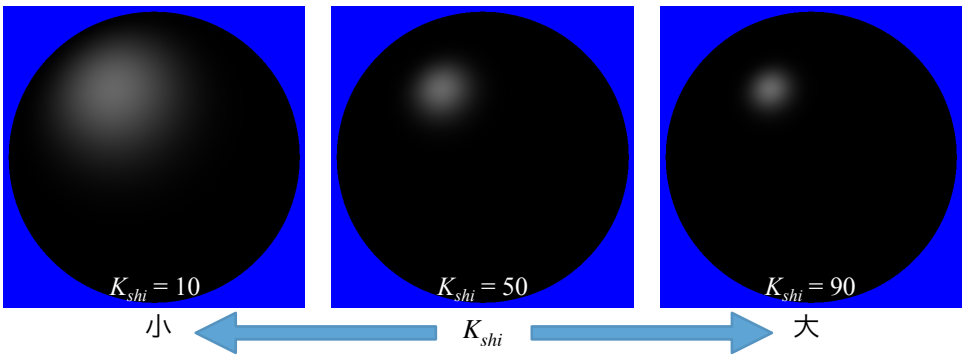


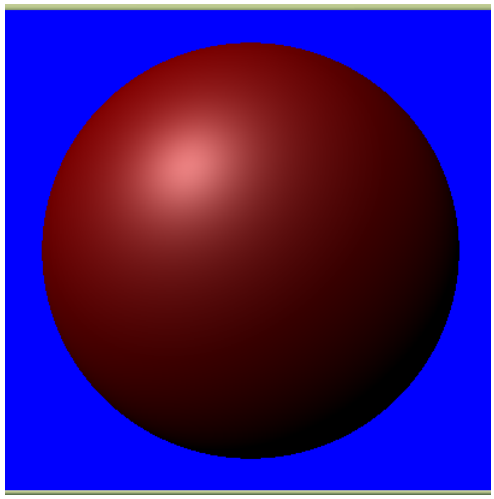
図 128 輝き係数とハイライトの大きさ

10.1.4 その他の鏡面反射関数

● Shlick の近似

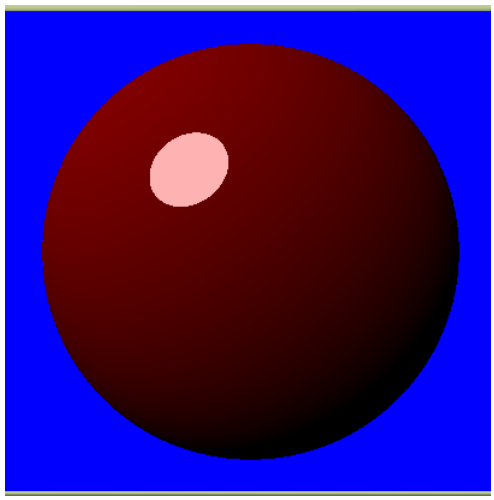
$$t = \cos\theta_r \quad \text{or} \quad t = \cos\theta_h$$

$$I_{spec} = \frac{t}{K_{shi} - tK_{shi} + t} K_{spec} \otimes L_{spec} \tag{90}$$



● しきい値 *t*

$$I_{spec} = \lceil \max(\mathbf{N} \cdot \mathbf{H}, 0) - t \rceil K_{spec} \otimes L_{spec} \tag{91}$$



● **環境光** (ambient)

拡散反射光や鏡面反射光は、光源から物体表面に直接届く**直接光**の反射光です。物体表面には直接光の他に、他の物体で反射した**間接光**も届きます。

しかし、間接光の到来経路は非常に複雑であり、ある面から放射された反射光が再びその面に到達する場合もあります。また天空光のように、天空全体から到来する光もあります。このように物体表面には、ありとあらゆる方向から、無限の経路で光が到達します。この精密な計算をリアルタイムに行うことは困難です。

そこで、ゲームグラフィックスやリアルタイムレンダリングでは、一般にこれを定数で表します。このような光を**環境光**といいます。

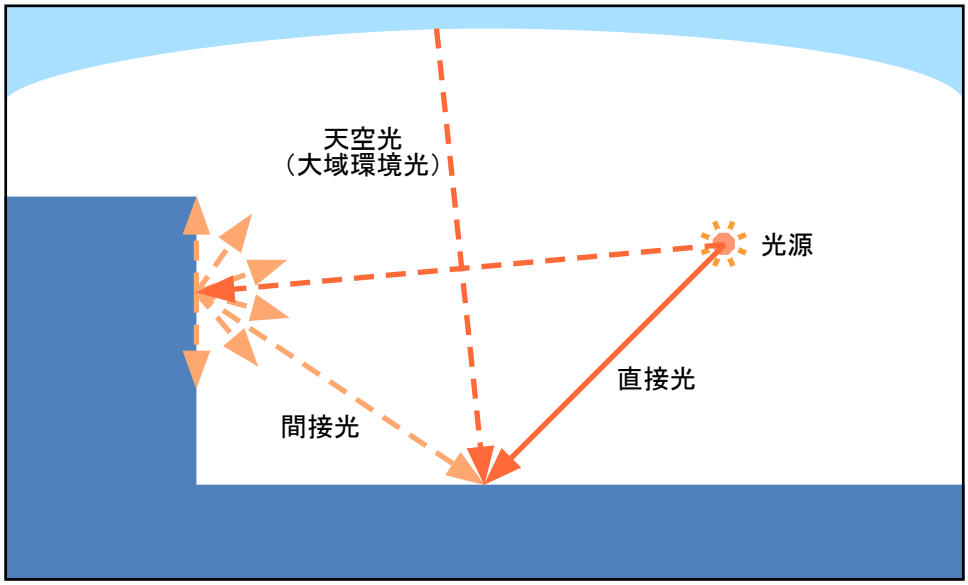


図 129 間接光と天空光

環境光はすべての方向から到来したひかるを総括したものですから、その反射光強度は方向に依存しません。物体表面の材質の環境光に対する反射係数を K_{amb}、光源の環境光の成分の強度を L_{amb} とすると、環境光の反射光強度は、次式により求めることができます。

$$I_{amb} = K_{amb} \otimes L_{amb} \tag{92}$$

一般的に環境光に対する反射係数 K_{amb} は、拡散反射係数 K_{diff} と一致させます。この環境光による陰影は、物体表面の向きや光源・視点の方向に依存しないため、図 130 のように一様なものとなります。

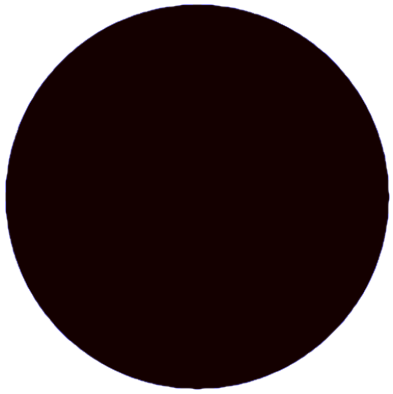


図 130 環境光による陰影

● 照明方程式

最終的な反射光の強度 I_{tot} は、拡散反射光の強度 I_{diff}、鏡面反射光の強度 I_{spec}、および環境光の反射光の強度 I_{amb} を合計したものになります。

$$I_{tot} = I_{amb} + I_{diff} + I_{spec} \tag{93}$$

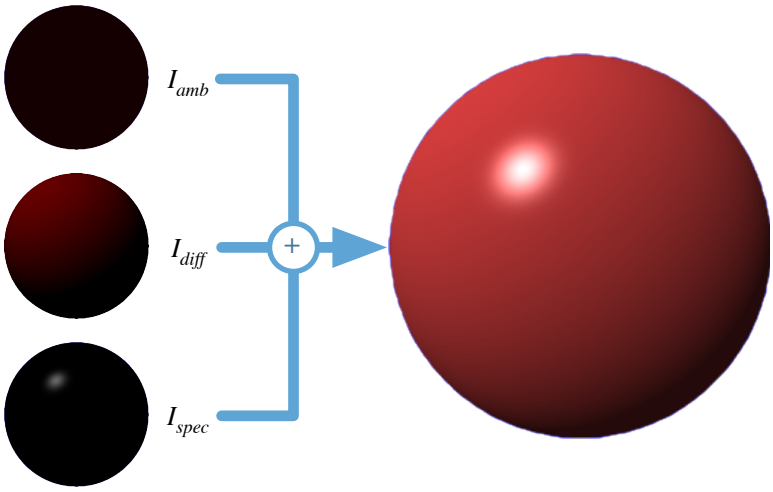


図 131 最終的な陰影

10.1.5 距離減衰係数

光源が点光源の場合は、光の放射面積が距離の二乗に比例します。したがって点光源の光の密度は、光源からの距離の二乗に反比例します。

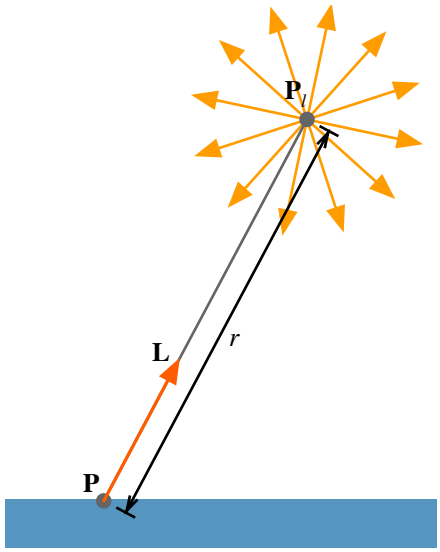


図 132 点光源までの距離

いま、照射点の位置を $\mathbf{P}$、点光源の位置を $\mathbf{P}_l$ とします (図 132)。すると、$\mathbf{P}$ における、この距離による減衰の割合 (距離減衰係数) d は、次式で求められます。

$$r = |\mathbf{P}_l - \mathbf{P}| \tag{94}$$

$$d = \frac{1}{r^2} \tag{95}$$

このモデルは物理的には正しいのですが、実際に使用すると距離に対する光の減衰が急すぎて、照明結果は非常に暗いものになります。これは真っ暗な部屋の中でロウソクを点けても、ロウソ

クの周囲は明るいのに部屋は全然明るくならないことからも分かります。

そこで、一般には次の式が用いられます。

$$d = \frac{1}{s_c + s_l r + s_q r^2} \tag{96}$$

ここで s_c は距離に依存しない成分の割合、s_l は距離に反比例する成分の割合、s_q は距離の二乗に反比例する成分の割合を表します。$s_c = 1, s_l = 0, s_q = 0$ なら光源の明るさは距離に依存せず、$s_c = 0, s_l = 1, s_q = 0$ なら距離に反比例し、$s_c = 0, s_l = 0, s_q = 1$ なら二乗に反比例します。

距離減衰を考慮した照明方程式は、次のようになります。

$$I_{tot} = I_{amb} + d(I_{diff} + I_{spec}) \tag{97}$$

10.1.6 スポットライト

スポットライトは光の照射範囲を制限した点光源です。照射範囲の制限には、Phong の陰影付けモデルと同様に、cosine モデルの分布が使用できます。

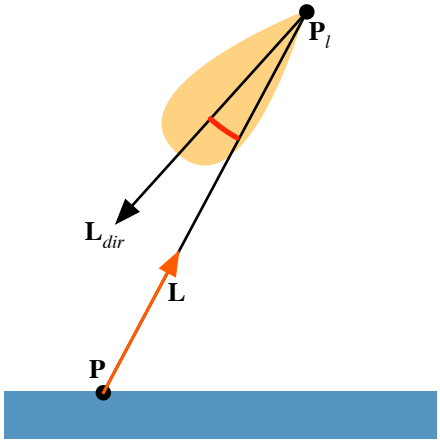


図 133 スポットライト

照射位置 $\mathbf{P}$ における光源方向を $\mathbf{L}$、スポットライトの照射方向を $\mathbf{L}_{dir}$ とするとき、スポットライトの係数 c_{spot} は次式により求めることができます。ここで L_{exp} はスポットライトの広がりを制御する係数で、これが大きいほどスポットライトが鋭くなります。

$$c_{spot} = \max(-\mathbf{L} \cdot \mathbf{L}_{dir}, 0)^{L_{exp}} \tag{98}$$

$$I_{tot} = c_{spot}(I_{amb} + d(i_{diff} + I_{spec})) \tag{99}$$

このスポットライトのモデルでは、光の照射範囲の外周のぼけの量を制御することができません。くっきりとしたスポットライトを得るには、別のモデルを用いる必要があります。

10.1.7 大域環境光

これまで述べて来た照明方程式の環境光の反射光成分 I_{amb} は、その光源からの間接構成分をまとめたものだと考え、その強さは光源の強さに依存しています。天空光など、光源に依存しな

い環境光 (背景光) は、これとは別に求めます。これを大域環境光といいます。また、物体自体が発光する、いわゆる自己発光を表現したい場合もあります。

大域環境光の強さを L_{glob}、物体表面の大域環境光に対する反射係数を K_{amb} とすると照明方程式は次式のようになります。

$$I_{tot} = K_{amb} \otimes L_{glob} + L_{emi} + c_{spot}\left\{I_{amb} + d\left(I_{diff} + I_{spec}\right)\right\} \tag{100}$$

光源が n 個ある場合は、次式のようになります。

$$I_{tot} = K_{amb} \otimes L_{glob} + L_{emi} + \sum_{k=1}^{n} c_{spot}^{k}\left\{I_{amb}^{k} + d^{k}\left(I_{diff}^{k} + I_{spec}^{k}\right)\right\} \tag{101}$$

なお、このモデルでは物体の自己発光による光が他の物体を照明することは考慮していません。

10.2 ポリゴンに色を付ける

10.2.1 面全体の陰影

これまでに述べてきた陰影付けモデルは、物体表面上の一点における反射光の強度を求めるものでした。図形を描くには、図形の表面全体について、陰影を求める必要があります。これは本来、物体表面の画面への投影像の一点一点について反射光の強度を求めることによって実現されます。しかし、陰影の計算には多くの実数計算を含まれるため、これはコストの高い処理になります。

そこで反射光強度の計算は頂点についてのみ行い、面内の陰影は頂点における反射光強度を補間して求めるという手段が用いられます。

- **頂点法線ベクトル**
- **法線の変換行列**

10.2.2 頂点属性の補間

図形の各頂点には、位置や色、法線ベクトル、テクスチャ座標など、様々な頂点属性を割り当てることができます。バーテックスシェーダでは、これらをもとに組み込み変数 gl_Position に値を代入し、次のラスタライザのステージでその値を頂点とする図形の塗り潰しを行います。そのほかの頂点属性は、バーテックスシェーダの out 変数に代入することによってラスタライザによって補間され、塗り潰す画素を処理するフラグメントシェーダに補間値として渡されます。

いま、図 134 のように三角形の各頂点に $(\mathbf{P}_0, \mathbf{C}_0), (\mathbf{P}_1, \mathbf{C}_1), (\mathbf{P}_2, \mathbf{C}_2)$ という頂点属性が割り当てられているとします。

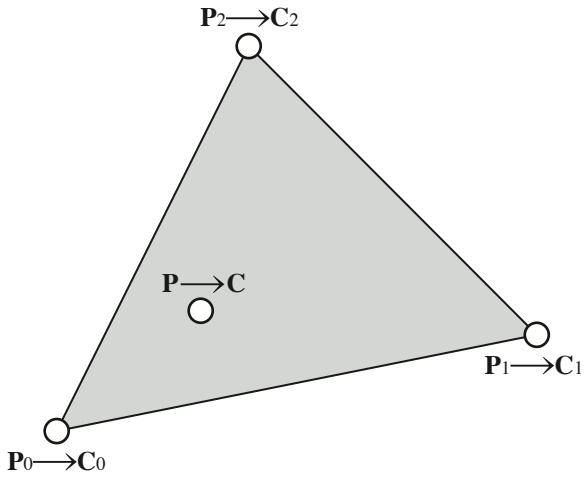


図 134 頂点属性の対応付け

このとき、$\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2$ を頂点とする三角形の内部の点 $\mathbf{P}$ における、頂点属性 $\mathbf{C}_0, \mathbf{C}_1, \mathbf{C}_2$ の補間値 $\mathbf{C}$ は、次のようにして求めることができます。

三角形の二辺に沿ったベクトル $\mathbf{P}_1 - \mathbf{P}_0, \mathbf{P}_2 - \mathbf{P}_0$ をそれぞれ $\mathbf{u}, \mathbf{v}$ とします (図 135)。

$$\begin{cases} \mathbf{u} = \mathbf{P}_1 - \mathbf{P}_0 \\ \mathbf{v} = \mathbf{P}_2 - \mathbf{P}_0 \end{cases} \tag{102}$$

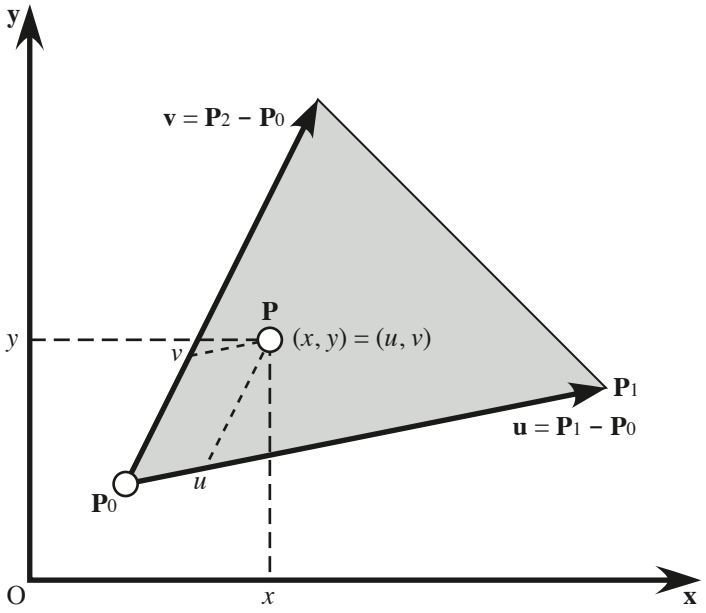


図 135 三角形内の点の座標

点 $\mathbf{P}$ の位置を、$\mathbf{P}_0$ を原点とし $\mathbf{u}, \mathbf{v}$ を軸とする座標系を用いて、次のように表します。

$$\mathbf{P} = x\mathbf{x} + y\mathbf{y} = u\mathbf{u} + v\mathbf{v} + \mathbf{P}_0 \tag{103}$$

ここで $\mathbf{x} = (1, 0), \mathbf{y} = (0, 1), \mathbf{u} = (x_u, y_u), \mathbf{v} = (x_v, y_v), \mathbf{P}_0 = (x_0, y_0)$ とすれば、(103) 式は次の連立方程式になります。

$$\begin{cases} x = ux_u + vx_v + x_0 \\ y = uy_u + vy_v + y_0 \end{cases} \tag{104}$$

これを u, v について解きます。

$$\begin{cases} u = \dfrac{(x - x_0)y_v - (y - y_0)x_v}{x_u y_v - x_v y_u} \\ v = \dfrac{(y - y_0)x_u - (x - x_0)y_u}{x_u y_v - x_v y_u} \end{cases} \tag{105}$$

この u, v を使って三角形の頂点 $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$ のそれぞれに割り当てられた頂点属性 $\mathbf{C}_0$, $\mathbf{C}_1$, $\mathbf{C}_2$ を補間します (図 136)。

$$\begin{aligned} \mathbf{C} &= u(\mathbf{C}_1 - \mathbf{C}_0) + v(\mathbf{C}_2 - \mathbf{C}_0) + \mathbf{C}_0 \\ &= (1 - u - v)\mathbf{C}_0 + u\mathbf{C}_1 + v\mathbf{C}_2 \end{aligned} \tag{106}$$

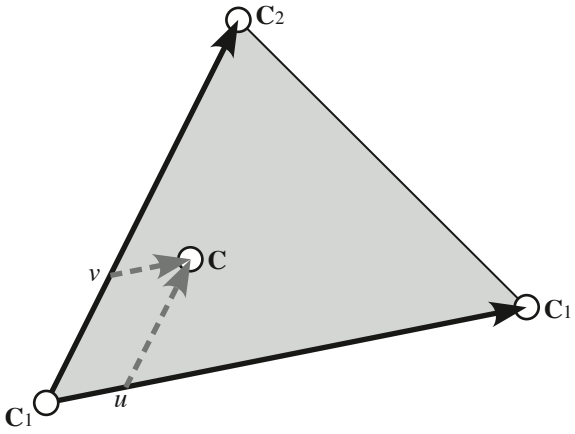


図 136 頂点属性の補間

しかし、図 137 の左のように矩形を二つの三角形で表現している場合、これを右のように変形すると、矩形内部の補間結果は三角形に沿って曲がってしまいます。

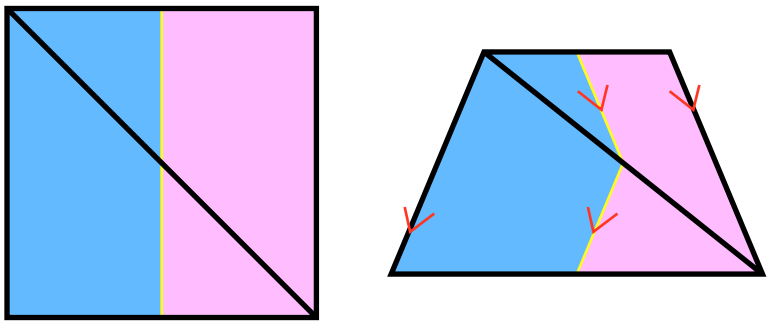


図 137 透視投影の影響

このような変形は、三次元形状を透視投影した場合に現れます。この影響により、物体がスクリーンと平行な軸で回転する場合などに、物体表面の陰影やテクスチャが、物体表面上を動いているように見える場合があります。

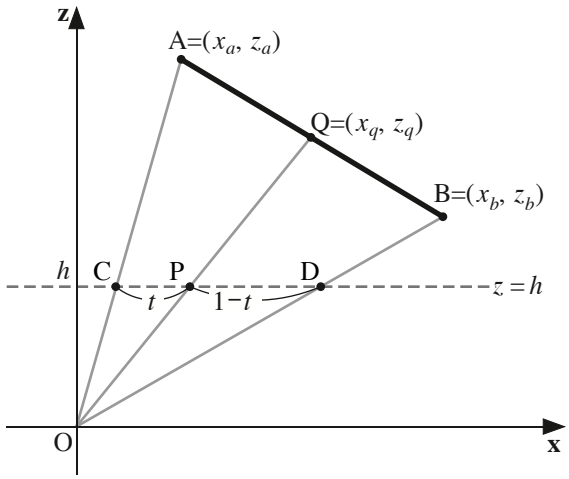


図 138 線形補間の逆透視投影

そこで、この補間を同次座標系で行うことを考えます。図 138 において、二点 A = (x_a, z_a)、B = (x_b, z_b) を端点とする線分 AB の $z = h$ への投影 CD を $t : (1 - t)$ で内分する点 P を、AB 上に逆投影した位置 Q = (x_q, z_q) は、次式により求めることができます。

$$\begin{cases} x_q = \dfrac{\dfrac{x_a}{z_a}(1-t) + \dfrac{x_b}{z_b}t}{\dfrac{1}{z_a}(1-t) + \dfrac{1}{z_b}t} \\ z_q = \dfrac{1}{\dfrac{1}{z_a}(1-t) + \dfrac{1}{z_b}t} \end{cases} \tag{107}$$

すなわち、z_q は z_a, z_b の逆数を線形補間したものの逆数であり、x_q は x_a, x_b のそれぞれを z_a, z_b で割ったものの線形補間に z_q を掛けたものになります。

したがって同次座標 (x, y, z, w) の場合は、その実座標は $(x / w, y / w, z / w)$ なので、頂点属性 $\mathbf{C}_0$, $\mathbf{C}_1, \mathbf{C}_2$ をそれぞれの**頂点位置の w 要素**で割ったものを u, v で線形補間し、それに頂点位置の w 要素の逆数を u, v で線形補間したものの逆数を掛けます (図 139)。

$$\begin{aligned} \mathbf{C} &= w\left\{(1-u-v)\frac{\mathbf{C}_0}{w_0} + u\frac{\mathbf{C}_1}{w_1} + v\frac{\mathbf{C}_2}{w_2}\right\} \\ w &= \frac{1}{(1-u-v)\dfrac{1}{w_0} + u\dfrac{1}{w_1} + v\dfrac{1}{w_2}} \end{aligned} \tag{108}$$

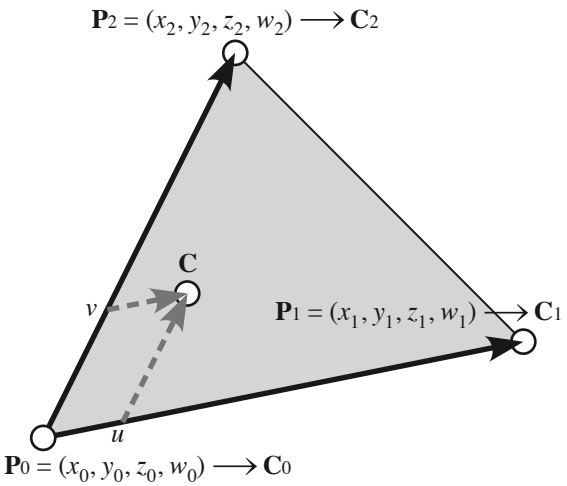


図 139 同次座標の線形補間

10.2.3 における陰影付け

陰影付け (シェーディング、shading) とは、CG において、物体に光を当てたときの物体表面の反射光強度を求めることをいいます。このためには、物体の表面に光の反射係数、すなわち色などの材質を設定し、物体表面の形状、光源の位置や方向、特性などをもとに、物体表面と光との相互作用を計算します。

GPU で陰影付けを行う場合は、CPU から図形の頂点属性として、頂点位置の他に、頂点の法線ベクトルを送ります。このとき、物体表面の光の反射係数などの材質情報は描画単位内では変更されない uniform 変数に設定します。材質の制御を描画単位より細かく行う場合は、これも頂点属性として GPU に送るか、テクスチャなどを用います。

GPU では、一般的にバーテックスシェーダにおいて、頂点の座標計算とともに照明計算を行い、得られた反射光強度を次のステージに送ります。次のステージがラスタライザであれば、ラスタライザは図形の塗り潰し時に頂点色を補間し、処理対象の画素の陰影を決定します。

10.2.4 頂点色を指定した図形の描画

まず、以下のシェーダのソースプログラムによる図形の描画について考えます。バーテックスシェーダでは、頂点属性として頂点の位置を受け取る in 変数 pv に加えて、頂点の色を受け取る cv という変数を追加しています。これをそのまま out 変数 vc に代入し、次のステージに送ります。ジオメトリシェーダを使用しなければ、次のステージはラスタライザです。

```
#version 150 core
in vec4 pv;
in vec4 cv;
uniform mat4 mc;
out vec4 vc;
void main(void)
{
  vc = cv;
  gl_Position = mc * pv;
}
```

フラグメントシェーダでは、前のステージ、すなわちラスタライザにより補間された頂点の色を in 変数 vc により受け取り、これをそのまま出力するカラーバッファに対応した out 変数 fc に格納します。

```
#version 150 core
in vec4 vc;
out vec4 fc;
void main(void)
{
  fc = vc;
}
```

ここで注意して欲しいのは、バーテックスシェーダの out 変数 vc と、フラグメントシェーダの in 変数 vc が、同じものを表すわけでは無い点です (図 140)。フラグメントシェーダの in 変数は、頂点ごとに出力されるバーテックスシェーダの out 変数を、図形内で補間したものです。

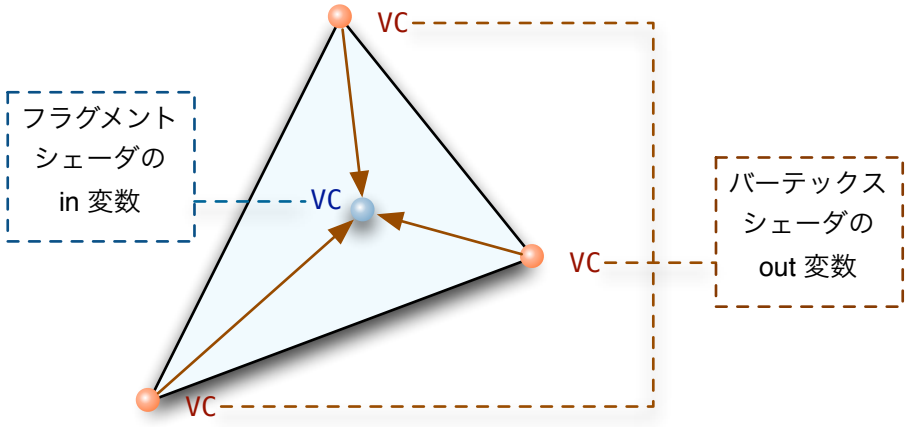


図 140 ラスタライザによる頂点属性の補間

これらのシェーダのソースプログラムを読み込んで、プログラムオブジェクトを作成します。読み込んだシェーダのシェーダのソースプログラムをコンパイルし、作成されたシェーダオブジェクトを glCreateShader() で作成されたプログラムオブジェクトに組み込んだ後、リンクします。その際、頂点属性の入力に使う in 変数のインデックスや、画素の色の出力先であるカラーバッファの番号を指定します。以下では、これらの両方ともに 0 を指定しています。

```
// プログラムオブジェクトの作成
GLuint program = glCreateProgram();

… (ソースプログラムの読み込み，コンパイル，取り付け等)

// プログラムオブジェクトのリンク
glBindAttribLocation(program, 0, "pv");    // in 変数のインデックス
glBindFragDataLocation(program, 0, "fc");  // 出力先のカラーバッファ番号
glLinkProgram(program);
```

in 変数のインデックスは、プログラムオブジェクトのリンク後に glGetAttribLocation() を使って取り出すこともできます。このソースプログラムでは、頂点属性として、頂点の位置を受け取る上記の pv に加えて、頂点の色を受け取る cv という変数をシェーダのソースプログラムに追

加しています。このインデックスを下記の手続きにより取り出して、cvLoc に保存します。

```
// in (attribute) 変数 cv のインデックスの検索（見つからなければ -1）
GLint cvLoc = glGetAttribLocation(program, ”cv");
```

uniform 変数のインデックスは、プログラムオブジェクトから glGetUniformLocation() を使って取り出します。

```
// uniform 変数 mc のインデックスの検索（見つからなければ -1）
GLint mcLoc = glGetUniformLocation(program, "mc");
```

図形の描画は頂点配列オブジェクト (Vertex Array Object, VAO) を介して行います。まず、頂点配列オブジェクトを作成し、それを結合します。

```
// 頂点配列オブジェクト
GLuint vao;
glGenVertexArrays(1, &vao);
glBindVertexArray(vao);
```

この状態で、頂点バッファオブジェクトを作成します。頂点の位置 pv と頂点の色 cv の二つの頂点属性を使用しますから、頂点バッファオブジェクトも二つ用意します。

```
// 頂点バッファオブジェクト
GLuint vbo[2];
glGenBuffers(2, vbo);
```

この一つ目の頂点バッファオブジェクトに、頂点の位置のデータを転送します。

```
// 頂点の位置
static const GLfloat pv[][3] =
{
  { -1.0f, -0.8660254f, 0.0f },
  {  1.0f, -0.8660254f, 0.0f },
  {  0.0f,  0.8660254f, 0.0f },
};

// 頂点の数
static const int points = sizeof pv / sizeof pv[0];

// 一つ目の頂点バッファオブジェクトに頂点の位置のデータを転送する
glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);  // 一つ目の頂点バッファオブジェクト
glBufferData(GL_ARRAY_BUFFER, sizeof pv, pv, GL_STATIC_DRAW);
```

また、この頂点バッファオブジェクトに格納されている頂点属性を、インデックスが 0 の in 変数から取り出せるようにします。ここでは in 変数 pv のインデックスを 0 にしています。

```
// この頂点バッファオブジェクトを index == 0 の in 変数から取得する
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(0);
```

また、二つ目の頂点バッファオブジェクトに、頂点の色のデータを転送します。

```
// 頂点の色
static const GLfloat cv[][3] =
{
  { 1.0f, 0.0f, 0.0f },  // 赤
```

```
    { 0.0f, 1.0f, 0.0f },  // 緑
    { 0.0f, 0.0f, 1.0f },  // 青
  };

  // 二つ目の頂点バッファオブジェクトに色のデータを転送する
  glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);  // 二つ目の頂点バッファオブジェクト
  glBufferData(GL_ARRAY_BUFFER, sizeof cv, cv, GL_STATIC_DRAW);
```

この頂点バッファオブジェクトに格納されている頂点属性は、cvLoc に保存しておいたインデックスの in 変数から取り出せるようにします。cvLoc には in 変数 cv のインデックスが保存されています。

```
  // この頂点バッファオブジェクトを index == cvLoc の in 変数から取得する
  glVertexAttribPointer(cvLoc, 3, GL_FLOAT, GL_FALSE, 0, 0);
  glEnableVertexAttribArray(cvLoc);
```

図形の描画は、頂点配列オブジェクトを指定して行います。

```
  // シェーダプログラムの選択
  glUseProgram(program);

  // uniform 変数 mc（モデルビュー・投影変換行列）を設定する
  glUniformMatrix4fv(mcLoc, 1, GL_FALSE, mc);

  // 描画に使う頂点配列オブジェクトの指定
  glBindVertexArray(vao);

  // 図形の描画
  glDrawArrays(GL_TRIANGLES, 0, points);
```

このように頂点ごとに色を指定した場合に表示される図形を図 141 に示します。この場合、各頂点に設定した色は図形内で補間され、画素の色に用いられます。

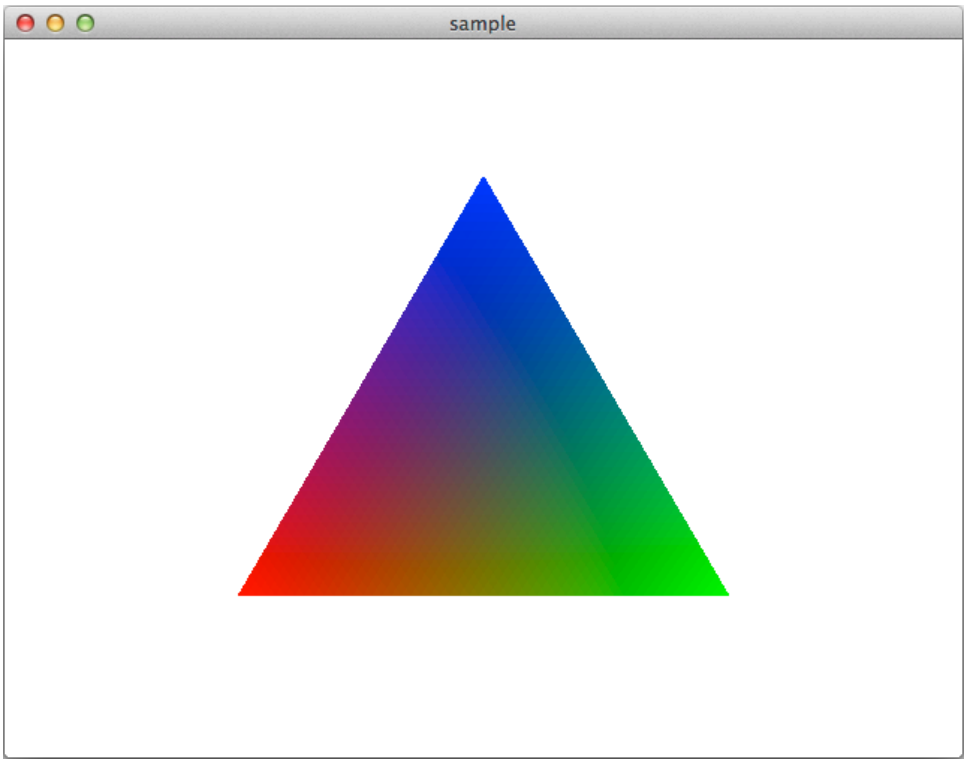


図 141 頂点色を指定した図形の描画

10.2.5 OpenGL における補間

OpenGL / GLSL のラスタライザは、標準でこの透視投影変換を考慮した補間を行います。なお、out / in 変数の宣言に noperspective という修飾子を付けることにより、透視投影変換の影響を考慮しない補間を行うことができます (標準は smooth)。また、flat という修飾子を付けると、補間は行われなくなります。この場合、最後に指定された頂点の頂点属性が用いられます。

```
noperspective out vec4 vc;  // バーテックスシェーダから次のステージへの出力
noperspective in vec4 vc;   // フラグメントシェーダの入力
```