レイトレーシングによるコンピュータグラフィクス入門
- 光の反射・屈折の計算 -

金森 由博*

2014 年 4 月 23 日

1　課題

シーン定義ファイル reflect.cfg (鏡面反射用) および refract.cfg (反射・屈折用) を読み込んで、課題のページの参照画像のような結果が得られるよう、RecursiveRayTracer.cpp というファイルの traceRec 関数を実装しよう。

2　光の反射・屈折

前回のレジュメ「物体の陰影の計算」では、物体の反射特性として、拡散反射と光沢反射の二つを扱った。もちろん、実世界の素材は拡散反射と光沢反射だけで表現できるわけではない。ここでは、前回のレジュメでは扱わなかった、鏡で起こる光の反射 (完全鏡面反射) や、ガラスや水面で起こる光の屈折を対象とする。

2.1　光の反射 (完全鏡面反射)

鏡のように、表面がつるつるしていて、周囲の物体が映り込むような物体を考えよう。前回のレジュメで説明した通り、このような物体の表面では完全鏡面反射が起こっている。つまり入射光は、面でちょうど折り返されたような一方向にのみ、反射される。光の入射方向を $\mathbf{L}$、面の法線ベクトル (単位ベクトル) を $\mathbf{N}$、光の反射方向を $\mathbf{R}$ とすると、

$$\mathbf{R} = \mathbf{L} - 2(\mathbf{L} \cdot \mathbf{N})\mathbf{N} \tag{1}$$

となる。なお、$(\mathbf{L} \cdot \mathbf{N})$ はベクトル $\mathbf{L}$ と $\mathbf{N}$ の内積である。

レイトレーシングでは、視点から見える物体の明るさを計算するわけだが、完全鏡面での明るさ $\mathbf{I}$ は次のように計算できる。

$$\mathbf{I} = \mathbf{k}_r \mathbf{I}_r \tag{2}$$

ここで、$\mathbf{k}_r$ は反射係数、$\mathbf{I}_r$ は光を追跡していって (再帰的な計算を行って) 得られる反射光の明るさである。$\mathbf{k}_r$ および $\mathbf{I}_r$ のそれぞれが RGB ごとに異なる値を持っているとして、RGB ごとに別々に乗算を行う。$\mathbf{I}_r$ を求めるためには、光が様々な物体の表面で何度も何度も反射・屈折する様子をずっと追跡していかなければな

* kanamori@cs.tsukuba.ac.jp

らない。とはいえ、反射・屈折を繰り返すうちに光の強さは減衰するので、レイトレーシングでは通常、反射・屈折を一定回数繰り返すと計算を打ち切る。さらにこの実験では、単純化のために[*1]、拡散面や光沢面にレイ (視線) が当たったら光の追跡計算を打ち切る。つまり、レイが当たった点で計算された明るさ (その点で、光源からの入射光を元に計算された明るさ) を持つ光が、他の物体の表面で反射・屈折して、最終的に視点に届く、ということにする。

2.2 光の屈折

次に、ガラスや水面で起こる光の屈折について考えよう (図 1)。光が屈折率の異なる物質に入射すると、反射と屈折 (光の向きが変わる) が起こる。屈折率は、物質によって固有の値 (空気:1.00、水:1.33、ガラス:1.5〜1.7) を持っている。いま、屈折率 n_1 の物質 1 の中から屈折率 n_2 の物質 2 の中へ、光が入射するとする。このとき、入射角 θ_1 と屈折角 θ_2 の間にはスネル (Snell) の法則が成り立つ。

$$n_1 \sin\theta_1 = n_2 \sin\theta_2 \tag{3}$$

これより、光の反射方向 $\mathbf{R}$ および屈折方向 $\mathbf{T}$ は、次の式で得られる。

$$\mathbf{T} = \frac{1}{n}\left\{\mathbf{L} + (c-g)\mathbf{N}\right\}, \tag{4}$$

$$\mathbf{R} = \mathbf{L} + 2c\mathbf{N}, \tag{5}$$

ここで、$n = n_2/n_1$ は相対屈折率であり、

$$c = \cos\theta_1 = -(\mathbf{L}\cdot\mathbf{N}), \tag{6}$$

$$g = \sqrt{n^2 + c^2 - 1} \tag{7}$$

である。

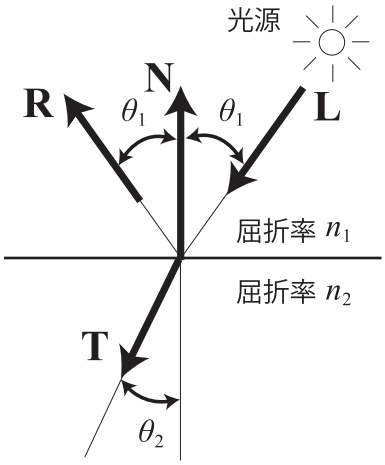

図 1　光の屈折.

[*1] 局所照明 (local illumination) モデルを採用したレイトレーシングでは一般的な方法である。まじめに計算しようとすれば、レイ (視線) が拡散面や光沢面に当たった場合、光が入射してくるいろいろな方向 (拡散面ならすべての方向、光沢面なら主にハイライトに貢献しやすい入射方向) に対してレイを飛ばして、計算を継続することになる。これがいわゆる大域照明 (global illumination) で行われる計算である。

ただし光が浅い角度で入射してくると、屈折が起きずにすべての光が反射される、という現象が起きる (図 2)。これを全反射という。上の式で言えば $g \leq 0$ となるときに起こる。また、屈折を計算する場合には面の法線ベクトル $\mathbf{N}$ とレイのなす角に注意する必要がある。例えば図 3 のように、空気中からガラス玉に光が入射するときは $\theta_1 \leq 90°$ であるが、ガラス玉の面で屈折した光が再び空気中に出て行く入射点では $\theta_2 > 90°$ になる。これらを考慮してレイが内部に入るときと外部に出るときとを判断し、相対屈折率を計算する。

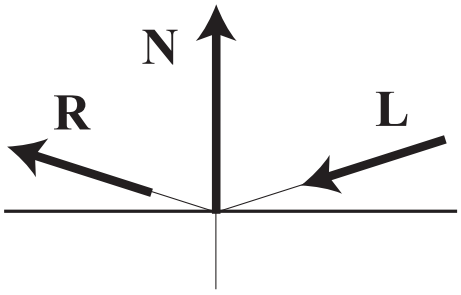

図 2　全反射. 光が浅い角度で入射すると, 屈折は起きずに反射だけになる.

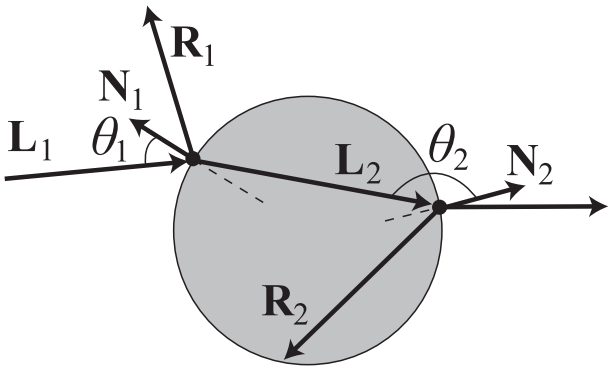

図 3　屈折の計算では、入射光と法線ベクトルのなす角によって、レイが物体の外側から入射するか内側から入射するかを判定できる.

光が反射・屈折する点での明るさは、次の式で計算できる。

$$\mathbf{I} = k_t \mathbf{I}_t + k_r \mathbf{I}_r \tag{8}$$

ここで、k_t は透過率 (transmittance)、$\mathbf{I}_t$ は透過した光の明るさ、k_r は反射率、$\mathbf{I}_r$ は反射した光の明るさである。透過光および反射光の計算は、再帰的な光の追跡計算を行う必要がある。k_t および k_r との間には、フレネル (Fresnel) の法則が成り立つ。

$$k_r + k_t = 1, \tag{9}$$

$$k_r = \frac{1}{2}\left\{\frac{\sin^2(\theta_1 - \theta_2)}{\sin^2(\theta_1 + \theta_2)} + \frac{\tan^2(\theta_1 - \theta_2)}{\tan^2(\theta_1 + \theta_2)}\right\} \tag{10}$$

この式をさらに変形すると、

$$k_r = \frac{1}{2}\left\{\left(\frac{c-g}{c+g}\right)^2 + \left(\frac{n^2c-g}{n^2c+g}\right)^2\right\} \tag{11}$$

となる。この実験では、この式を簡略化した Schlick の近似式を用いる。

$$k_r = r_0 + (1 - r_0)(1 - c)^5, \tag{12}$$

$$r_0 = \left(\frac{n-1}{n+1}\right)^2 \tag{13}$$

3 プログラムについて

再帰的なレイトレーシングのプログラムにおいて、(完全鏡面での) 反射・屈折の計算はなかなか厄介である。反射・屈折した光の方向ベクトルが正しく計算できているか、反射係数などの係数が正しく計算できているかなど確認すべきことが多い。この実装ができればより見栄えのする画像が生成できるようになるので、根気強く取り組んでほしい。

反射・屈折を扱うためにプログラムで変更すべきところは、`RecursiveRayTracer.cpp` ファイルの `traceRec` 関数の中である。`traceRec` 関数は、レイ (視線) と衝突した点における、物体の面の色を計算する関数である。RGB の要素ごとに色を計算するので、返り値の型は `vec3` になっている。

RecursiveRayTracer.cpp

```
vec3 RecursiveRayTracer::traceRec(const Ray &ray, int recursionDepth)
{
    /* ... (略) ... */

    const vec3 hitPos = ray.getPosition( record.t );
    Material *material = record.material;

    if (material->getMaterialType() == Material::Perfect_Specular_Type)
    {
        // TODO: fill here

        return vec3(0,0,0);
    }
    else if (material->getMaterialType() == Material::Specular_Refraction_Type)
    {
        // TODO: fill here

        return vec3(0,0,0);
    }

    /* ... (略) ... */
}
```

`if` および `else if` のブロックには、それぞれ完全鏡面反射 (`Material::Perfect_Specular_Type` と書いてあるブロック) および屈折 (`Material::Specular_Refraction_Type` と書いてあるブロック) を扱うた

めの処理を記述する。どちらも再帰的な計算を行う必要がある。つまり、このブロック内で traceRec 関数を呼び出して計算を行う。このとき、関数が無限に呼び出されてしまうのを防ぐために、再帰の深さを表す引数 recursionDepth を 1 だけ増やして呼び出すようにする。再帰を打ち切る上限の深さは、グローバル変数 max_recursion_depth で決められている。traceRec 関数の一つ目の引数には、レイを表す Ray クラスのデータを与える必要がある (詳しくは Ray.h ファイルを参照)。Ray クラスは、レイの始点と単位方向ベクトルから定義される。この場合、始点は hitPos で与えられている。方向ベクトルは、反射や屈折に基づいて計算すればよい。まとめると、traceRec 関数を再帰的に呼び出すには、次のように記述すればよい。

RecursiveRayTracer.cpp

```
// d は、反射・屈折に基づいて計算された、新しい単位方向ベクトル
... = traceRec( Ray(hitPos, d), recursionDepth+1 );
```

反射や屈折の計算には、レイの入射方向のベクトルと、交点での法線ベクトルが必要になる。レイの入射方向のベクトルは、ray.getUnitDir() という関数呼び出しで得られる。法線ベクトルは、レイと交点の情報を記録した record という構造体のメンバ変数となっている。record.normal が交点での法線ベクトルである。

式 (2) の完全鏡面のための反射係数 k_r は、材質を表す変数 material から得られる。material は、材質を表す基底クラスである Material クラスのポインタなので、これを完全鏡面反射を表す PerfectSpecularMaterial クラスのポインタにキャストする必要がある。そして、反射係数を得るための getSpecularCoeff 関数を呼べばよい。まとめると次のようになる。

RecursiveRayTracer.cpp

```
const vec3 kr = ((PerfectSpecularMaterial *)material)->getSpecularCoeff();
```

同様に、式 (8) の屈折率を得るには、material を屈折を表す SpecularRefractionMaterial クラスのポインタにキャストして、getRefractionIndex 関数を呼べばよい。

RecursiveRayTracer.cpp

```
const Real eta = ((SpecularRefractionMaterial *)material)->getRefractionIndex();
```

屈折の計算について、課題のプログラムでは空気と他の物質との境界面での屈折のみを対象としている。つまり、片方の物質の屈折率は 1 であると仮定している。他の物質の屈折率を η だとすると、相対屈折率 n は、光が空気中から他の物質に入射するとき $n = \eta$ で、逆に他の物質から空気中に出射するとき $n = 1/\eta$ となる。

屈折の際、光が空気中から他の物質に入るのか出るのかを判定するには、入射ベクトル (レイの方向ベクトル) と交点での法線ベクトルの内積を計算すればよい。課題のプログラムでは、交点での法線ベクトルは物体の外向きになるように計算されている。したがって、空気中から他の物質に入る場合は内積は負、逆に空気中に出る場合は内積が正となる。なお、光が空気中に出るときには、境界面での法線ベクトルが外向きになっているので、法線ベクトルの方向を反転 (法線ベクトルにマイナスをつければよい) してから残りの計算を行うこと。

反射光および屈折光の方向ベクトルを計算するには、MyAlgebra.h に定義されている、reflect 関数および refract 関数を利用すると便利である。

MyAlgebra.h

```
vec3 i; // 光の入射ベクトル
vec3 r = reflect(i, n); // n は単位法線ベクトル

Real eta = 1.5f; // 相対屈折率
vec3 r = refract(i, n, eta); // n は単位法線ベクトル、全反射の場合 r はゼロベクトル
```

以上をまとめると、光の完全鏡面反射および屈折の計算の擬似コードはそれぞれ次のようになる。

光の完全鏡面反射の計算の擬似コード

レイが交差した物体の反射率 $\mathbf{k}_r$ を取得;
正反射の方向ベクトル $\mathbf{d}$ を計算;
$\mathbf{d}$ の方向へ再帰的にレイを追跡し交点の色 $\mathbf{I}_r$ を取得;
反射率 $\mathbf{k}_r$ と交点の色 $\mathbf{I}_r$ を要素ごとに乗算した結果を返す;

光の屈折の計算の擬似コード

レイが交差した物体の屈折率 η を取得;
交点での法線ベクトル $\mathbf{N}$ を取得;
レイの入射方向 $\mathbf{L}$ を取得;
if (レイが物体内部から空気中に出る場合) {
　$\eta = 1/\eta$;
　$\mathbf{N} = -\mathbf{N}$;
}
if (全反射の場合) {
　光が完全鏡面反射した先でレイが交差する点の色 $\mathbf{I}_r$ を再帰的に計算して返す;
} **else** {
　フレネル係数 k_r を計算;
　光が完全鏡面反射した先でレイが交差する点の色 $\mathbf{I}_r$ を再帰的に計算;
　光が屈折した先でレイが交差する点の色 $\mathbf{I}_t$ を再帰的に計算;
　$\mathbf{I}_r$ と $\mathbf{I}_t$ をフレネル係数 k_r でブレンドして返す;
}

屈折の計算結果のデバッグはなかなかやりづらい。反射成分のみ、屈折成分のみを計算して不自然な画像になっていないか確認してから、両者をフレネル係数でブレンドして出力するとよい。