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
- テクスチャマッピング -

金森 由博*

2014 年 4 月 30 日

これまでは、物体表面は無地で何も模様がない、としてきた。ここでは、物体表面に画像や模様を貼り付ける技術である、**テクスチャマッピング** (texture mapping) を扱い、物体の見た目のバリエーションを増やす。

1 課題

ソースコード `TexturedPlane.cpp`, `TexturedSphere.cpp`, `SolidTexturedSphere.cpp` それぞれの `hit` 関数のうち、テクスチャ座標の計算部分を実装し、テクスチャマッピングを実現しよう。

2 テクスチャマッピング

テクスチャマッピングとは、3 次元の交点の座標から、対応する画像 (**テクスチャ画像**または単に**テクスチャ**と呼ぶ) のピクセル位置から、色を取得することである (図 1)。3 次元の点と画像上の点との対応関係をどのように与えるかによって、様々なマッピング (mapping; 写像) が考えられる。マッピングを定義するには、物体表面上の各点に対応する、画像上の座標を与えてやればよい。この画像上の座標のことを、**テクスチャ座標**と呼ぶ。テクスチャ座標は、画像のサイズに依存しないよう、$[0, 1]$ の範囲に正規化された 2 次元座標として計算される。慣例として、横方向のテクスチャ座標は $u \in [0, 1]$ (または s)、縦方向のテクスチャ座標は $v \in [0, 1]$ (または t) で記述されることが多い。u 座標と v 座標をあわせて uv 座標などと呼ばれる。

テクスチャの与え方は、画像ファイルを読み込む、あるいは適当な関数を使って計算する、などの方法がある。画像ファイルを読み込む場合、任意の模様を指定することができるが、画像の解像度が低かったり一部を拡大したりすると模様の粗い部分が見えてしまう場合がある。一方、関数を用いて計算する場合、計算負荷が高く模様の指定に制限はあるものの、いくら拡大しても鮮明な模様を得ることができる。

2.1 ソリッドテクスチャ

テクスチャとして与えるデータは、2 次元の画像に限らない。3 次元的なデータを与えて、交点の 3 次元座標との対応関係を考えることもできる。このとき与える 3 次元データを、**ソリッドテクスチャ** (solid texture) と呼ぶ。ソリッドテクスチャを 3 次元の画像データとして与えることもできるが、メモリ消費量が多いため、適当な関数を用いて計算により模様を得る方法がより好まれる。

* kanamori@cs.tsukuba.ac.jp

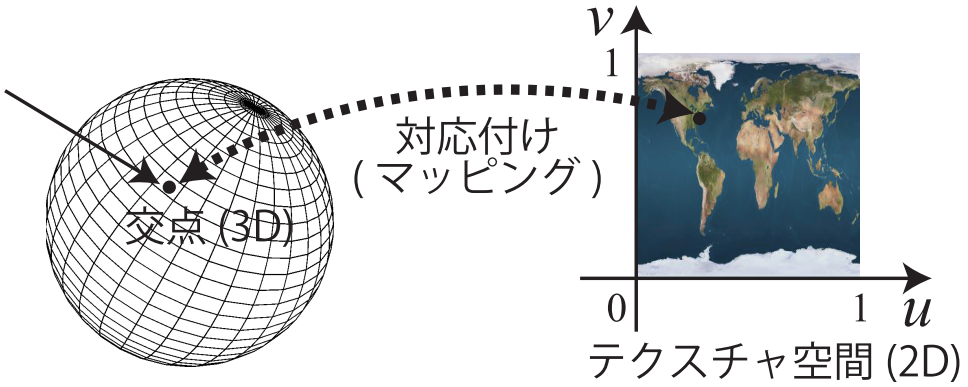

図 1　テクスチャマッピング (2D の場合).

2.2　プロシージャルテクスチャ

大理石、木目、雲、炎など、自然界には複雑な模様を持つものが多い。このような模様を表現するための手法として、Ken Perlin の開発した Perlin Noise が有名である[*1]。Perlin Noise を用いたものに限らず、画像データによってではなく計算手続きによって得られるテクスチャのことをプロシージャルテクスチャ (procedural texture; 手続き的テクスチャ) と呼ぶ。

3　プログラムについて

この実験のプログラムでテクスチャマッピングを行うには、テクスチャマッピングのデータを提供するクラスと、テクスチャ座標を持つ形状クラスの両方を実装する必要がある。前者はほぼ実装済みであるが、テクスチャマッピングを施した物体表面の色を計算するために、`TexturedMaterial.h` の `calcReflectedLight` 関数は自分で実装する必要がある。これは `PhongMaterial` と同様に実装してもらえばよい。なおこの実験では、テクスチャによって得られた色を、拡散反射の反射係数として計算する。プログラムで提供されているテクスチャの種類は下記の通りである：

`ImageTexture`　画像読み込みによるテクスチャ。
`CheckeredTexture`　チェッカー (格子) 模様のテクスチャ。
`SolidCheckeredTexture`　チェッカー模様のソリッドテクスチャ。
`MarbleTexture`　Perlin Noise による大理石のような模様のソリッドテクスチャ。
`WoodenTexture`　Perlin Noise による木目のような模様のソリッドテクスチャ。
`NoiseTexture`　Perlin Noise によるぼんやりした模様のソリッドテクスチャ。
`TurbulenceTexture`　細かさの異なる Perlin Noise を重ね合わせたソリッドテクスチャ。
`DerivTurbulenceTexture`　上記の `TurbulenceTexture` の変種。

これらの指定の仕方は、シーン定義ファイル `texture_sample.txt` などを参照してほしい。一方、テクスチャ座標を持つ形状クラスについては、以下のファイルの `hit` 関数に手を加える必要がある：

[*1] Ken Perlin はこの業績によって、アカデミー技術部門賞を受賞した。

`TexturedPlane.cpp`　テクスチャマッピングを施した平面。テクスチャ座標は周期的に設定される。周期はメンバ変数 `texture_scale` によって決定される。

`TexturedSphere.cpp`　テクスチャマッピングを施した球面。世界地図のように、球面を平面に展開して得られる座標系でテクスチャ座標を計算する。つまり、経度/緯度が u, v 座標に対応する。

`SolidTexturedSphere.cpp`　ソリッドテクスチャマッピングを施した球面。球を囲む立方体の、各辺の長さが 1 である、としたときの、立方体内部の点の位置からテクスチャ座標が計算できる。

以下、それぞれについて、実装方法を解説する。

3.1 平面のテクスチャ座標 (`TexturedPlane.cpp`)

この実験では、平面上のテクスチャ座標 $[0,1]\times[0,1]$ を次のように定義する (図 2)。まず平面上に、ある基準点 $\mathbf{o}'$ を決める。平面の単位法線ベクトルを $\hat{\mathbf{n}}$ (プログラムでは `normal`) として、それに直交する単位ベクトル $\hat{\mathbf{b}}$ (プログラムではメンバ変数 `binormal`)、$\hat{\mathbf{t}}$ (プログラムではメンバ変数 `tangent`) を用意し、これらを平面上の座標系の軸とする。これらの軸に沿って、基準点 $\mathbf{o}'$ を頂点のひとつとする一辺の長さが s (プログラムではメンバ変数 `texture_scale`) の正方形領域を考える。その正方形内部のテクスチャ座標が $[0,1]\times[0,1]$ の範囲にあるものとする。正方形の外側については、さらに正方形を繰り返し並べる。この正方形がテクスチャ画像に対応する。

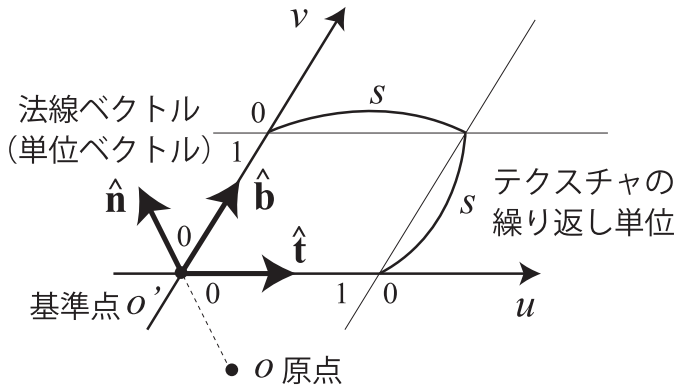

図 2　平面上でのテクスチャ座標の定義.

レイと平面が交差する交点を $\mathbf{p}$ とすると、点 $\mathbf{p}$ のテクスチャ座標は次のように計算できる。まず、点 $\mathbf{p}$ を、平面上の座標系で表現する。u 方向、v 方向の座標をそれぞれ u_p, v_p とすると、これらは、基準点 $\mathbf{o}'$ から点 $\mathbf{p}$ に向かうベクトルを、軸を表す単位ベクトル $\hat{\mathbf{t}}$, $\hat{\mathbf{b}}$ に射影することで得られる。

$$\begin{aligned} u_p &= (\mathbf{p}-\mathbf{o}')\cdot\hat{\mathbf{t}}, \\ v_p &= (\mathbf{p}-\mathbf{o}')\cdot\hat{\mathbf{b}} \end{aligned} \tag{1}$$

これら u_p, v_p は、まだ [0,1] の範囲の値ではない。そこで、これらの座標をテクスチャの繰り返し単位の長さである s で割り、その小数部分をテクスチャ座標 (u,v) として取り出す。

$$\begin{aligned} u &= u_p/s - \lfloor u_p/s \rfloor, \\ v &= v_p/s - \lfloor v_p/s \rfloor \end{aligned} \tag{2}$$

ここで、$\lfloor u_p/s \rfloor$ および $\lfloor v_p/s \rfloor$ は、u_p/s および v_p/s を超えない最大の整数である。

上記の (1) 式で必要な、基準点 $\mathbf{o}'$ は、平面上の点であれば任意に選べる。この実験では簡単のため、原点 $\mathbf{o}$ をこの平面に射影した点とする。この基準点 $\mathbf{o}'$ を以下のようにして求める。基準点 $\mathbf{o}'$ は平面上の点なので、次の平面の定義式を満たす。$\mathbf{x}$ を平面上の任意の点の座標として

$$\mathbf{x} \cdot \hat{\mathbf{n}} = d \tag{3}$$

ここで d は定数である。原点 $\mathbf{o}$ を法線ベクトル $\hat{\mathbf{n}}$ の方向に長さ t だけ移動させた点が基準点 $\mathbf{o}'$ だとして (3) 式に代入すると、

$$\begin{aligned} \mathbf{o}' &= t\,\hat{\mathbf{n}} + \mathbf{o} \\ &= t\,\hat{\mathbf{n}} \end{aligned} \tag{4}$$

となる。(4) 式を (3) 式に代入すると、

$$\begin{aligned} (t\hat{\mathbf{n}}) \cdot \hat{\mathbf{n}} &= d \\ t &= d \end{aligned} \tag{5}$$

ここで、$\hat{\mathbf{n}}$ が単位ベクトルであることを用いた。この結果を (4) 式に代入すれば、基準点 $\mathbf{o}'$ が求まる。

$$\mathbf{o}' = d\,\hat{\mathbf{n}} \tag{6}$$

3.2 球面のテクスチャ座標 (TexturedSphere.cpp)

球面を 2D 平面に対応付ける方法はいくつかあるが、この実験では、世界地図のように球の緯度と経度によってテクスチャ座標を決める。地球儀を想像してほしい。緯度・経度を radian 単位で考える。緯度は、赤道を 0 radian として、北極が $\frac{\pi}{2}$ radian、南極が $-\frac{\pi}{2}$ radian である。経度は、北極と南極をつなぐ基準線 (地球の場合はグリニッジ天文台を通る子午線) を 0 radian として、$-\pi$ radian から π radian までである。

この実験では、図 3 のような座標系を考え、北極・南極が y 軸上にあるものとする。球面をパラメータ化するには、単位法線ベクトル $\hat{\mathbf{n}}$ が扱いやすい。$\hat{\mathbf{n}}$ は中心が原点で半径 1 の球 (単位球) の球面上の点の座標と考えられる。

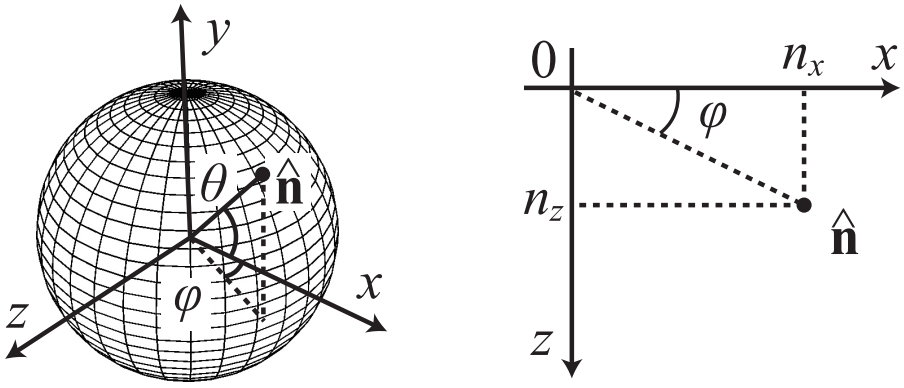

図 3　この実験で扱う球の座標系.

経度 ϕ および緯度 θ は、次のように計算できる。

$$\phi = \tan^{-1}\frac{n_z}{n_x}, \quad \theta = \sin^{-1} n_y \tag{7}$$

ここで、経度 ϕ は x 軸の正の向きを 0 とした。なお、n_z/n_x は $n_x = 0$ のときにゼロ除算が発生してしまうので、計算には atan2 を使うとよい。上述の通り、$\phi \in [-\pi, \pi]$ および $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ であるので、$[0, 1]$ の範囲にする必要がある。

$$u = \frac{\phi + \pi}{2\pi}, \quad v = \frac{\theta + \frac{\pi}{2}}{\pi} \tag{8}$$

3.3 ソリッドテクスチャマッピングのための球のテクスチャ座標 (SolidTexturedSphere.cpp)

前述の通り、法線ベクトル $\hat{\mathbf{n}} \in [-1, 1]^3$ は単位球の球面上の点の座標を表している。これを使ってソリッドテクスチャマッピングのためのテクスチャ座標 $(u, v, w) \in [0, 1]^3$ を計算できる。

$$u = \frac{n_x + 1}{2}, \quad v = \frac{n_y + 1}{2}, \quad w = \frac{n_z + 1}{2} \tag{9}$$

4 プログラムについて

平面でのテクスチャ座標の計算で現れる、浮動小数点数 x を超えない最大の整数 $\lfloor x \rfloor$ の計算には注意が必要である。例えば、浮動小数点数 x を int 型にキャストして求めようとすると、x が負の数の場合に問題が起きる。なぜなら int 型へのキャストは普通、0 に近い整数値を与えるためである。例えば、$x = -2.7818$ という値に対して (int)x は-2 となる。x を超えない最大の整数は正しくは-3 である。x が負の数であっても正しい結果を得るには、数学関数の floor 関数を使えばよい。

実験に使うシーン定義ファイルは textures.cfg である。このファイルには次のような記述がある。

textures.cfg

```
/* ...（略）... */

sphere {
  center = (0, 0, 0);
  radius = 2.0;

  /* "texture_include.cfg" で定義済みのテクスチャ */
  aSolidChecker;
//  aNoise;
//  aMarble;
//  aWood;
//  anImage;
//  aTurbulence;
//  aDturbulence;
```

```
}

plane
{
  normal = (0,1,0);
  distance = -2.0;

  tex_scale = 5.0;

  aChecker;
}

/* ...（略）... */
```

平面 plane については、この設定でチェッカー模様を表すマテリアル aChecker が設定されている。球 sphere については、標準でチェッカー模様のソリッドテクスチャを表すマテリアル aSolidChecker が設定されている。他のテクスチャを試すには、コメントアウトされているものを外せばよい。ここに挙げてあるものはそれぞれ、単純な Perlin Noise のテクスチャ aNoise、大理石模様のテクスチャ aMarble、木目調のテクスチャ aWood、画像テクスチャ anImage、乱流テクスチャ aTurbulence、乱流テクスチャに微分を加えた aDturbulence である。