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
- 形状のインスタンス化と座標変換 -

金森 由博*

2014 年 6 月 4 日

1 課題

ソースコード `GeometryInstance.cpp` ファイルにおいて、形状に対する座標変換を実現しよう。

2 形状のインスタンス化と座標変換

同じ形状データを、3D シーン中にいくつも配置したいとき、何度もデータを読み込んだのではメモリを浪費してしまう。特に、形状データのデータサイズが大きい場合、例えば三角形数の多いメッシュなどでは特に問題である。このような場合は、形状データは 1 つ分のメモリだけ確保して、他は座標変換やマテリアルを変えて[*1]同じ形状データを使いまわすのがよい。これを形状のインスタンス化 (*geometry instancing*) という。この実験のプログラム特有の事情として、三角形メッシュにはインスタンス化と座標変換が必要である。三角形メッシュの大きさはものによってまちまちなので、この実験のプログラムでは、位置と大きさを、基準となる位置と大きさになるように調整している[*2]。物体を任意の大きさに拡大縮小したり、回転したり平行移動したりできるよう、座標変換の計算を実装しよう。

3 同次座標 (斉次座標)

まずは座標変換について復習しよう。3 次元の物体の座標は 3 次元ベクトルで表現できる。物体表面の点 $\mathbf{p}$ が、3×3 の回転行列 R と平行移動ベクトル $\mathbf{t}$ によって座標変換されて点 $\mathbf{p}'$ に移動するとすると、

$$\mathbf{p}' = R\,\mathbf{p} + \mathbf{t} \tag{1}$$

である。さらに点 $\mathbf{p}'$ に、例えば回転行列 R' と平行移動ベクトル $\mathbf{t}'$ で座標変換して点 $\mathbf{p}''$ に移動するとすると、

$$\begin{aligned} \mathbf{p}'' &= R'\,\mathbf{p}' + \mathbf{t}' &(2)\\ &= R'(R\,\mathbf{p} + \mathbf{t}) + \mathbf{t}' &(3) \end{aligned}$$

* kanamori@cs.tsukuba.ac.jp

*1 現状のプログラムでは、座標変換はサポートしているが、残念ながらマテリアルの変更はサポートしてない。

*2 具体的には、バウンディングボックスの大きさが、1 辺の長さが 1 であるような立方体に収まるよう一様に拡大縮小し、バウンディングボックスの x,y,z 座標が最小である端点が原点と一致するよう平行移動している。

となる。これにさらに座標変換が適用されると、3×3 行列と平行移動ベクトルが増えて煩雑になる。実は、このような座標変換をもっと簡単に扱う方法がある。3 次元ベクトルに、もうひとつ要素を足して 4 次元ベクトルとし、とりあえず 4 つ目の要素を 1 とする。以降、列ベクトルを表現するためにベクトルに転置の記号 T をつけておく。

$$\mathbf{p} = (x \;\; y \;\; z \;\; 1)^T \qquad \mathbf{p}' = (x' \;\; y' \;\; z' \;\; 1)^T \tag{4}$$

座標変換を 4×4 行列で表現する。先ほどの回転行列 R と平行移動ベクトル $\mathbf{t}$ による変換行列を M とすると

$$M = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \tag{5}$$

と書くことができる ($\mathbf{0}$ はゼロベクトル)。すると座標変換を

$$\mathbf{p}' = M\mathbf{p} \tag{6}$$

と書けるようになる。回転行列 R' と平行移動ベクトル $\mathbf{t}'$ による座標変換 M' を追加しても

$$\mathbf{p}'' = M'\,\mathbf{p}' \tag{7}$$

$$= M'M\,\mathbf{p} \tag{8}$$

と単純に書ける。ここでさらに、上記の 4 次元ベクトルに対して、「4 つ目の要素ですべての要素を除算しても、除算する前のベクトルと同一視する」という規則を設けよう。すなわち、

$$(x \;\; y \;\; z \;\; w)^T = (\frac{x}{w} \;\; \frac{y}{w} \;\; \frac{z}{w} \;\; 1)^T \tag{9}$$

と見なす、ということである。実はこのように決めると、$w = 0$ とすることで無限遠の点 (あるいは位置ベクトルではなく方向ベクトル) を表現したり、本来は線形変換ではない射影変換 (平行投影や透視投影など) を線形変換として記述したりすることができるようになる。以上のように、3 次元ベクトルに対して 4 次元ベクトル、2 次元ベクトルに対して 3 次元ベクトル、というように、d 次元ベクトルに対して $(d+1)$ 次元ベクトルで記述された座標のことを**同次座標** (または**斉次座標**; *homogeneous coordinates*) と呼ぶ。座標変換の扱いが容易になることから、OpenGL や DirectX などグラフィクスライブラリで標準的に用いられている。この実験のプログラムでも、同次座標によって座標変換を記述することにする。

4　レイの座標変換

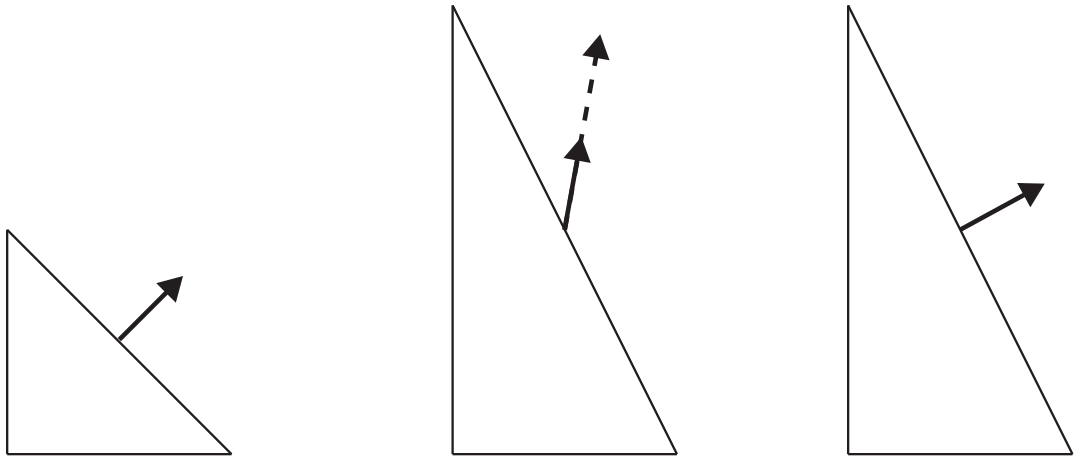
物体を座標変換するといっても、例えば三角形メッシュの各頂点を座標変換してメモリに記憶するのでは、結局メモリ消費量が増えてしまう。また例えば、数式で表現された球を引き伸ばして楕円体にしたい場合、三角形メッシュのように頂点を座標変換したくても、数式で表現された球にはそもそも頂点が存在しない。そこで、物体に座標変換を適用する代わりに、レイに対して適当な座標変換を適用して交差判定を行う。

では、レイにはどのような座標変換を施せばよいだろうか？　レイが座標変換適用後の形状と交差する点を $\mathbf{p}'$ とする。レイの始点を $\mathbf{o}$, 単位方向ベクトルを $\hat{\mathbf{d}}$, パラメータを t とすると

$$\mathbf{p}' = \mathbf{o} + t\,\hat{\mathbf{d}} \tag{10}$$

と書ける。一方、座標変換を 4×4 行列 M で表すと、ある点 $\mathbf{p}$ に対して M を適用することで $\mathbf{p}'$ になった点、すなわち M を適用する前の点 $\mathbf{p}$ が存在するはずで、

$$\mathbf{p}' = M\,\mathbf{p} \tag{11}$$



(a) 三角形の斜辺の法線ベクトル　(b) 法線ベクトルの y 成分を 2 倍　(c) 本当の法線ベクトル

図 1　法線ベクトルと座標変換.

と書ける。ここで、座標変換 M を適用する前の点 $\mathbf{p}$ については、物体とレイとの交差判定の計算方法がわかっているはずである。そこで点 $\mathbf{p}$ についてのレイの式を導出したい。式 (11) の両辺に M^{-1} を掛けると

$$M^{-1}\,\mathbf{p}' = \mathbf{p} \tag{12}$$

なので、式 (10) を代入して

$$\begin{aligned}\mathbf{p} &= M^{-1}\,(\mathbf{o} + t\,\hat{\mathbf{d}}) &(13)\\ &= M^{-1}\,\mathbf{o} + t\,M^{-1}\,\hat{\mathbf{d}} &(14)\end{aligned}$$

となる。つまり、レイの始点 $\mathbf{o}$ と単位方向ベクトル $\hat{\mathbf{d}}$ をそれぞれ M^{-1} で変換してから交点を求めれば、交点でのパラメータ t が計算できる、ということである。ただし、方向ベクトル $\hat{\mathbf{d}}$ の w 座標はゼロとする。

実は、まだこれだけでは十分ではない。ある点 $\mathbf{p}$ が M で座標変換されるとき、その点での法線ベクトル $\mathbf{n}$ (単位ベクトル) はどうなるかを考えてみよう。簡単のため、2 次元の例で説明する。図 1(a) のような直角二等辺三角形の斜辺の法線ベクトルは、明らかに $(1/\sqrt{2}, 1/\sqrt{2})^T$ である。では、この三角形を y 方向に 2 倍拡大してみよう。法線ベクトルの y 成分も同じように 2 倍してやると、「法線ベクトルは面 (この場合は 2 次元なので辺) と垂直である」という条件が満たされなくなってしまう。以上からわかることは、法線ベクトルは点の座標変換とは異なる変換を施す必要がある、ということである。

では法線ベクトルにどのような変換を施せばよいかというと、M^{-T} (4×4 行列 M の逆行列 M^{-1} を転置した行列) を適用すればよい。ただし、M の 3×3 成分がスケール成分を含まなければ、つまり回転行列 R(直交行列) であれば、$R^{-1} = R^T$ であるから $R^{-T} = R$、つまり行列 R をそのまま掛ければよい。詳細を知りたい人はこのレジュメの付録を参照してほしい。なお、交点での法線ベクトル $\mathbf{n}$ は正規化されているはずであるが、M に拡大縮小が入っていると、$M^{-T}\,\mathbf{n}$ は単位ベクトルではなくなってしまう。M^{-T} で座標変換後にベクトルを正規化する必要がある。

さらにまた、この実験のプログラム特有の修正が必要である。実験のプログラムでは、レイの方向ベクトルは単位ベクトルであると仮定している。座標変換されたレイの方向ベクトルは $M^{-1}\,\hat{\mathbf{d}}$ であるが、もし M に拡大縮小が入っていると、$M^{-1}\,\hat{\mathbf{d}}$ は単位ベクトルではなくなってしまう。そこで、方向ベクトルが単位ベク

トルになるよう、細工を施す。式 (14) より

$$\mathbf{p} = M^{-1}\,\mathbf{o} + tM^{-1}\,\hat{\mathbf{d}} \tag{15}$$

$$= M^{-1}\mathbf{o} + (t\,\|M^{-1}\,\hat{\mathbf{d}}\|)\frac{M^{-1}\,\hat{\mathbf{d}}}{\|M^{-1}\hat{\mathbf{d}}\|} \tag{16}$$

$$= M^{-1}\mathbf{o} + t'\,\hat{\mathbf{d}}' \tag{17}$$

ここで、$t' = t\,\|M^{-1}\hat{\mathbf{d}}\|$, $\hat{\mathbf{d}}' = \frac{M^{-1}\hat{\mathbf{d}}}{\|M^{-1}\hat{\mathbf{d}}\|}$ とおいた。t' と $\hat{\mathbf{d}}'$ を使って交差判定を行い、交点が見つかったら、

$$t = \frac{t'}{\|M^{-1}\hat{\mathbf{d}}\|} \tag{18}$$

によって補正すれば、交点での正しいパラメータ t が求まる。これとともに HitRecord オブジェクトに入っている交点の座標も、正しいパラメータ t を使って再計算する必要がある。これには Ray クラスの getPosition 関数を使うとよい。

以上をまとめると、次のような手順で計算すればよい。

hit 関数の擬似コード

```
レイの始点、方向ベクトルを座標変換する;
座標変換した方向ベクトルの大きさを記録しつつ、正規化する;
レイと物体との交差判定を行う;
if (交点が見つかった) {
    見つかった法線ベクトルを座標変換し、正規化する;
    レイのパラメータを、元の方向ベクトルの大きさで補正する;
    レイのパラメータから、交点の座標を再計算する;
    マテリアルをセットする;
    return true;
}

return false;
```

座標変換のためのメンバ変数として、GeometryInstance クラスには座標変換行列の逆行列 m_InverseTransformMatrix が用意されているので、座標変換にはこれを用いる。行列は mat4 クラスである。hit 関数と同様に、shadowHit 関数も実装する。

mat4 クラスについて補足する。メンバ変数は、16 個のスカラー値 a00, a10, ..., a33 と、要素数 16 個の配列 mat_array の共用体になっている。要素の並び順が**列優先** (*column-major*) になっていることに注意してほしい。すなわち、配列の要素の並び順を見ると、0 列目の要素が 0 行から 3 行まで埋まったら次の 1 列目の要素が並ぶ、という風になっている。なので例えば、mat4 クラスのコンストラクタに

```
mat4 M( f0,  f1,  f2,  f3,
        f4,  f5,  f6,  f7,
        f8,  f9, f10, f11,
       f12, f13, f14, f15);
```

と書いたら、4×4 行列 M の内部的には

```
f0  f4   f8  f12
f1  f5   f9  f13
f2  f6  f10  f14
f3  f7  f11  f15
```

という風に並んでいることになる。ちなみに、行列内の要素の並び順を列優先ではなく**行優先** (*row-major*) にしているライブラリも世の中にはある。どちらがよいかは、列ベクトル・行ベクトルのどちらに高速にアクセスできるようにしたいかなどによる。`mat4` クラスの使い方はレジュメ「ベクトル・マトリクスライブラリの使用方法」を見てほしい。

設定ファイルから座標変換を読み取ったら、座標変換行列に反映する。例えば、点 $\mathbf{v}$ にスケーリング行列 S、回転行列 R、平行移動行列 T をこの順に適用して点 $\mathbf{v}'$ が得られるとすると

$$\mathbf{v}' = T\,R\,S\,\mathbf{v} \tag{19}$$

である。座標変換行列について見ると、S の左に R, R の左に T が掛けられている。つまり、既存の行列に左から行列を掛けると新たな座標変換行列が得られる。ただし、実験のプログラムでは、メンバ変数として保持されているのが座標変換行列の逆行列なので、逆変換を考える。

$$S^{-1}\,R^{-1}\,T^{-1}\,\mathbf{v}' = \mathbf{v} \tag{20}$$

これを見ると、S^{-1} の右に R^{-1}, R^{-1} の右に T^{-1} が掛けられている。つまり、既存の行列に右から逆行列を掛けることで、それらをまとめた新たな逆変換が得られていることがわかる。実装も、既存の逆行列 `m_InverseTransformMatrix` に、新たな座標変換の逆行列を、右から掛けるようにすればよい。`GeometryInstance` クラスの以下の関数を実装しよう。

- `transform(const vec4 &col0, const vec4 &col1, const vec4 &col2, const vec4 &col3)`
  座標変換行列 M の各列が引数で与えられた場合に呼び出される関数。
- `transform( const mat4 &M )`
  座標変換行列 M が与えられた場合に呼び出される関数。
- `translate( float x, float y, float z )`
  平行移動ベクトル $(x\ y\ z)^T$ が与えられた場合に呼び出される関数。
- `translate( const vec3 &t )`
  平行移動ベクトル t が与えられた場合に呼び出される関数。
- `scale( float sx, float sy, float sz )`
  x, y, z 軸方向の拡大縮小率 `sx`, `sy`, `sz` が与えられた場合に呼び出される関数。
- `rotate( float angle, float x, float y, float z )`
  回転軸 $(x\ y\ z)^T$ (※正規化されているとは限らない) と、その軸周りの回転角度 `angle` (度数単位で指定されるのでラジアン単位に変換する必要あり) が与えられた場合に呼び出される関数。

付録: 法線ベクトルに施す座標変換について

図 1 の例で見たように、座標変換の行列 M がスケール成分を含む場合、法線ベクトル $\mathbf{n}$ に M を掛けると、法線ベクトルが面と直交しなくなってしまう。では、法線ベクトルにはどのような行列を適用すればよいか考

えてみよう。法線ベクトル $\mathbf{n}$ に対応する、面の接線ベクトルを $\mathbf{t}$ とおく。これらは直行するので内積はゼロである。

$$\mathbf{n} \cdot \mathbf{t} = 0 \tag{21}$$

ここで、接線ベクトルに行列 M を適用したベクトルを $\mathbf{t}'$ とする。法線ベクトルについても、行列 M に相当する行列 G を適用し、その結果得られるベクトルが $\mathbf{n}'$ だとする。

$$\mathbf{t}' = M\,\mathbf{t} \qquad \mathbf{n}' = G\,\mathbf{n} \tag{22}$$

座標変換後の法線ベクトル $\mathbf{n}'$ と接線ベクトル $\mathbf{t}'$ が直交しているとすると、これらの内積はやはりゼロである。

$$\mathbf{n}' \cdot \mathbf{t}' = 0 \tag{23}$$

これに式 (22) を代入する。

$$(G\,\mathbf{n}) \cdot (M\,\mathbf{t}) = 0 \tag{24}$$

$$\mathbf{n}^T\,G^T M\,\mathbf{t} = 0 \tag{25}$$

ここで式 (21) より、$\mathbf{n} \cdot \mathbf{t} = \mathbf{n}^T\,\mathbf{t} = 0$ であるから、$G^T M = I$ (I は単位行列) となれば上の式が成り立つ。よって

$$G = (M^{-1})^T \tag{26}$$

となる。すなわち、法線ベクトル $\mathbf{n}$ には行列 M^{-T} を適用すればよい、ということである。