学生実験４：コンピュータグラフィックス（CG）
— 投影変換とシェーディング —

担当：舩冨卓哉 （TA:井上仁, 清水渚佐）

2012 年度版 (Ver 1.1)

1　はじめに

本演習では、三次元データを二次元画像で表現する際の基本技術を修得することを目的として、実際に計算機上で三次元物体の可視化を行うプログラムを作成し、その挙動を確認する。別の言い方をすれば、これは VRML ブラウザのレンダリング部分の技術を学び、ソフトウェアを作成する演習とも言える。

計算機上の三次元データを二次元画像に投影変換する過程において、得られる画像は主に以下の３つの要因によって決まる。

(1) 撮像方法 (カメラモデル)

(2) 観測物体 (形状、表面の反射率、位置)

(3) 光源 (色、輝度、位置)

このうち、(1) は幾何問題に帰着される。また、(2), (3) は物理現象に関連が深く、これらを扱う処理を一般に “シェーディング” という。本稿では、(1) については 3章で述べ、(2), (3) については 8章で述べるが、演習では特に (2), (3) に焦点を当てる。

なお、コース履修に当たって、課題などの指示は WWW に掲示する。また、更新・訂正情報や本テキストの全文も掲載する。WWW 上と本テキストで相違がある場合は、WWW 上の文書の内容を有効とする。

URL： http://www.mm.media.kyoto-u.ac.jp/education/le4cg/

2　準備

2.1　画像のデータ構造と表示

計算機内にある三次元物体を可視化する際には、計算機上のウィンドウシステムを通じて画像表示を行うことになる。

計算機で扱う画像は、画素が二次元配列で並べられたデータ構造で表すことができる。画素は物体がどのような色や明るさで見えるかを規定するが、計算機上ではこれを光の三原色である RGB

の成分によって表す。本演習ではこの RGB それぞれを 256 段階の輝度で表す方法を採用する。これで 2^{24} 通りの色（いわゆる 1670 万色）を表現できることになる。厳密には、モニタの色温度やブライトネス・コントラスト等の調整によって画面に表示される色は変化するので、色の再現性が重要な用途ではモニタの色調整から行う必要があるが、本演習ではそこまでは考慮せず、画像の生成までを対象とする。

2.2 画像フォーマット

コンピュータで扱う画像形式については現在様々な形式が提案されている。本演習ではこのうち、最も画像フォーマットが簡単な ppm 形式を利用する。この形式は、RGB それぞれに 8bits をとった画素あたり 24bits のカラー画像を表現できる。

ppm 形式については、man ページを参照されたい。この形式のファイルの拡張子は一般に ppm とするのが普通であり、xview などの画像表示ツールを用いて表示することができる。

```
ppm(5)                                                    ppm(5)

NAME
       ppm - portable pixmap file format

DESCRIPTION
       The  portable pixmap format is a lowest common denominator
       color image file format.  The definition is as follows:

       - A "magic number" for identifying the file type.   A  ppm
         file's magic number is the two characters "P3".

       - Whitespace (blanks, TABs, CRs, LFs).

       - A width, formatted as ASCII characters in decimal.

       - Whitespace.

       - A height, again in ASCII decimal.

       - Whitespace.

       - The  maximum color-component value, again in ASCII deci-
         mal.

       - Whitespace.

       - Width * height pixels, each three ASCII  decimal  values
         between  0  and the specified maximum value, starting at
```

the top-left corner of the pixmap, proceeding in normal English reading order. The three values for each pixel represent red, green, and blue, respectively; a value of 0 means that color is off, and the maximum value means that color is maxxed out.

- Characters from a "#" to the next end-of-line are ignored (comments).

- No line should be longer than 70 characters.

Here is an example of a small pixmap in this format:

```
P3
# feep.ppm
4 4
15
 0  0  0    0  0  0    0  0  0   15  0 15
 0  0  0    0 15  7    0  0  0    0  0  0
 0  0  0    0  0  0    0 15  7    0  0  0
15  0 15    0  0  0    0  0  0    0  0  0
```

Programs that read this format should be as lenient as possible, accepting anything that looks remotely like a pixmap.

There is also a variant on the format, available by setting the RAWBITS option at compile time. This variant is different in the following ways:

- The "magic number" is "P6" instead of "P3".

- The pixel values are stored as plain bytes, instead of ASCII decimal.

- Whitespace is not allowed in the pixels area, and only a single character of whitespace (typically a newline) is allowed after the maxval.

- The files are smaller and many times faster to read and write.

Note that this raw format can only be used for maxvals less than or equal to 255. If you use the ppm library and

```
try to write a file with a larger maxval, it will automat-
ically fall back on the slower but more general plain for-
mat.
```

一般に ppm 形式では表示時にデータの初めのほうがモニタ表示時に上に来て、後のほうが下に来るのが普通である。

この形式を用いて画像を X 座標、Y 座標で表現すると、X 軸は右向き、Y 軸は下向きとなる。これに対して後述する世界座標系では、Y 軸を上向きにするのが一般的であり、その場合は軸の向きが逆になることに注意されたい。

2.3 三次元物体の表現

計算機上で三次元世界を表現するには、三次元物体を計算機上のデータとして表現しなければならない。このための三次元物体の表現法には、大きく分けて “ソリッドモデル” と “サーフェスモデル” がある。ソリッドモデルは物体をそのまま立体として表現する。例えば単位球を表現する場合、これを $x^2 + y^2 + z^2 \leq 1$ のような形で三次元の “塊” として表現するような表現方法がこれに相当する。実際にソリッドモデルを表現するには関数モデルや離散モデル（“ボクセル” という単位立方体の集合で表す方法等）など様々な表現方法がある。ソリッドモデルは一部の CAD や数値計算シミュレーション、科学技術計算などでよく用いられる。しかし、CG のように物体の見かけを表現する目的に限った場合、外からは見えない物体内部の属性は必ずしも必要でない。そこで、外から見える部分である物体表面についての情報だけを表現するのがサーフェスモデルである。

サーフェスモデルにおいて、物体表面の幾何情報を表す形態には関数モデルと離散モデルがある。関数モデルは物体表面をパラメトリック曲面によって表現するもので、ベジェ曲面や B-Spline 面、さらにその発展形である NURBS や、メタボール、超二次関数、さらにはフラクタクル関数等で表現する。これらはその特性をよく理解すれば数少ないパラメータで曲面の形状を制御できて便利な一方、各々の関数形式を越えた曲面を表現することは困難である。これに対して、離散モデルは一般に “ポリゴンモデリング”、“パッチモデリング” と呼ばれている手法で、物体表面を微小な多角形の面（ポリゴン）の集合で表現するものである。理論上は面を微細にしていけばどのような面でも十分な精度で近似できるという利点を持つ。その一方でパラメータ数が膨大（基本的に頂点の数 $\times 3$）になるという欠点をもつが、現在の計算機環境では汎用性に富むポリゴンモデルが利用されることが多い。これは、優秀なレンダリングアルゴリズムとハードウェアの出現によるところが大きいと思われる。

本演習でもポリゴンモデルを採用し、三次元物体の形状を表すために、オープンな規格の一つである VRML ver 2.0 の構文の一部を利用する。

本演習で最低限用いるフォーマットは以下のみである。

- appearance
  - Material
    - diffuseColor：拡散反射係数
    - specularColor：鏡面反射係数
    - ambientIntensity：環境光係数

        * shininess：鏡面反射強度

- geometry
    - IndexedFaceSet

VRML のファイルはテキストファイルであり、通常拡張子が wrl である。次にサンプルを示す。

```
#VRML V2.0 utf8
Shape {
 appearance Appearance {
  material Material{
   diffuseColor    1.0 1.0 0.0
   specularColor   1.0 0 0
   ambientIntensity 0.4
   shininess 0.2
  }
 }
 geometry IndexedFaceSet {
  coord Coordinate {
   point [
    0.0     1.0     2.0,
    2.0     0.0     3.0,
    2.0     2.0     2.0,
    1.5     6.0     0.5,
    1.0     0.0     0.0
   ]
  }
  coordIndex [
    1, 2, 4, -1,
    0, 2, 3, -1,
    0, 4, 2, -1
  ]
 }
}
```

Coordinate ノード内の point フィールドのカンマで区切られたセットが X,Y,Z 座標を示し、一つの頂点の三次元座標を表現する。頂点番号は、一つ目から順に番号 $0, 1, 2, \cdots$ となる。IndexedFaceSet ノード内の coordIndex フィールド内の各インデックスは-1 で区切られ、それぞれ Coordinate の頂点の番号の並びによって一つの面を表現している。本例の一行目 (1,2,4,-1,) は、頂点 1 と頂点 2 と頂点 4 とで構成される三角形の面を表す。

本演習では、この頂点の並びによって面の裏表の区別をつける。すなわち、頂点の並びが反時計回りに見える側を表（法線方向）と定義する。VRML 2.0 ではポリゴンが多角形でも構わないが、問題を簡単にするため、本演習ではポリゴンは全て三角形とする。

Material ノードについてはシェーディング処理の項で各フィールドの意味を詳しく述べるが、各フィールド（上記の例では diffuseColor）に続いて [0.0, 1.0] の範囲に正規化された (R,G,B) の値で色を表現する。ただし、shininess だけは 1 要素のみが後に続く。

3 投影変換

我々が三次元世界を視るとき、その情報は目の水晶体を通して網膜上に結ばれた二次元の像として得られる。このような三次元世界の二次元への変換は、カメラの撮像過程でも同様である。このカメラの撮像過程のモデル化として代表的なものに、“平行投影” と “透視投影” がある。

まず、三次元世界全体を考える。三次元世界を表現するために、三軸の直交座標系（以後これを “世界座標系” と呼ぶ）を採用する。ここでは三軸を右手系の順序で X,Y,Z とする。この世界の中に物体が存在することになる。光源も存在するが、本章ではカメラと物体の幾何的な関係のみを考えているため、光源についての議論は省略する（これについては 8章で述べる）。

三次元世界の画像を生成することは、この三次元世界を二次元平面に投影変換する処理に他ならない。ただし、得る画像の大きさは有限なので、投影する際に空間のどの部分を投影するかを意識する必要がある。以後、この投影される平面のことを “画像平面”、画像平面上で実際に画像として可視化する領域を “撮像領域” と呼ぶ。

また、これ以降は座標系は同次座標系[1]で表現する。同次座標系は空間を表す三要素のベクトルに一要素を加えたもので、空間の射影問題を解くときに都合が良い。三次元ユークリッド空間上の一点 $(x_i, y_i, z_i)^T$ は同次座標系では $(x_i, y_i, z_i, 1)^T$ で表現できる。また、同次座標 $(x_a, y_a, z_a, \gamma)^T$ は各要素を γ で割れば、その三次元ユークリッド空間上の点の位置となる。

3.1 平行投影

平行投影（“直交投影”、“正射影” とも呼ばれる）とは、空間内にあるベクトルを考え、それに平行に物体上の一点を画像平面 I 上の一点へ投影する方法である。最も簡単な場合の例として、視線方向が Z 軸方向、画像平面が $z = z_I$ である場合を図 1に示す。

このとき、空間内の一点 $P = (x_p, y_p, z_p, 1)^T$ は、画像平面 I 上の一点 $P' = (x'_p, y'_p, z_I, 1)^T$ に投影される。これは、

$$\begin{pmatrix} x'_p \\ y'_p \\ z_I \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & z_I \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} \tag{1}$$

で表現される。ただし、点 P を画像平面 I にプロットする際には z_I は必要ない。また、図 1において $z_n \leq z \leq z_f$ を満たす z 値を持つ物体のみが投影対象となり、画像上に表される。さらに、画像平面上の撮像領域のみが画像として可視化される。

この方法は非常にアルゴリズムが簡単であり、計算量も少ないが、昔の日本画のように、奥にある物体も手前にある物体も同じ大きさに描画されることになるため、実際の CG に用いられるよりは、CAD の見取り図のように、各辺の寸法を図で表現するような目的に利用されることが多い。

[1] “斉次座標系” とも呼ばれる。この座標系に従えば、移動と回転の複合を一つの行列にまとめられたり、透視投影変換を一つの行列で表現することなどが可能になる。

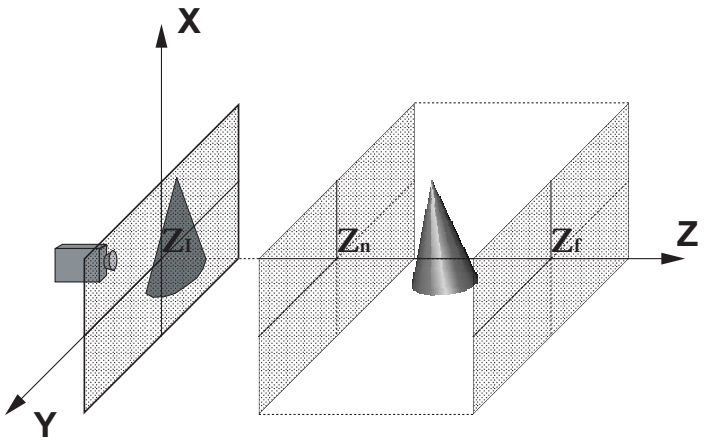

図 1: 平行投影

3.2 透視投影

透視投影（“中心投影” とも呼ばれる）を考えるとき、一番分かりやすいのがピンホールカメラを思い浮かべることである（図 2参照)。簡単のため、いまカメラのピンホールが点 $(0,0,0)$ にあり、カメラの方向は $(0,0,1)$ を向いているものとする。また、画像平面 I が $z = z_I (z_I > 0)$ に設定されるものとする。実際のピンホールカメラでは画像平面は $z = -z_I$ になるが、計算理論上はピンホールのどちら側に画像平面 I があっても変わりはない。このようにカメラを基準として定められる座標系を、以後 “カメラ座標系” と呼ぶ。

このとき、空間内の一点 $P = (x_p, y_p, z_p, 1)^T$ は、画像平面上の一点 $P' = (x'_p, y'_p, z_I, 1)^T$ に投影されるが、この P' を求めるために、まず

$$P'' = \begin{pmatrix} x_p \\ y_p \\ z_p \\ \frac{z_p}{z_I} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{z_I} & 0 \end{bmatrix} \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} \tag{2}$$

という行列で点 P'' を求める。P'' は P' と同一のユークリッド座標に射影される点である。本式の結果の P'' は同次座標表現なので、画像平面 I 上の座標を得るために第四要素 $\frac{z_p}{z_I}$ で P'' の全項を割ってみると、次式のように (x'_p, y'_p, z_I) が求められる。

$$\frac{1}{\frac{z_p}{z_I}} P'' = \frac{1}{\frac{z_p}{z_I}} \begin{pmatrix} x_p \\ y_p \\ z_p \\ \frac{z_p}{z_I} \end{pmatrix} = \begin{pmatrix} \frac{z_I}{z_p} x_p \\ \frac{z_I}{z_p} y_p \\ z_I \\ 1 \end{pmatrix} = \begin{pmatrix} x'_p \\ y'_p \\ z_I \\ 1 \end{pmatrix} = P' \tag{3}$$

画像平面上には撮像領域が設定されるので、透視投影では画像に投影される範囲は図 2に示されるような撮像領域を上面とした四角錐台の内部のみとなる。$|z_I|$ の距離は “焦点距離” とも呼ばれ、撮像領域の大きさを一定とした場合、焦点距離を短くとればパースペクティブはきつくなり、遠近法が強調された画像となる。

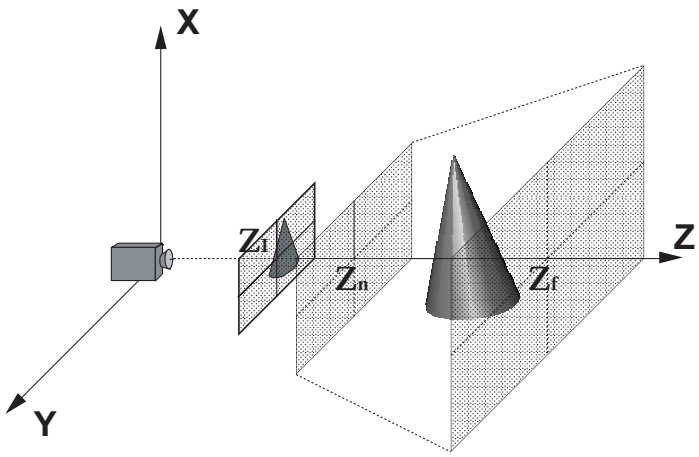

図 2: 透視投影

4 三次元の移動・回転・拡大縮小

前章では、ピンホールの位置をそのカメラ座標系の原点、カメラ方向を (0,0,1) としたが、一般にはこれらは世界座標系の上に設定され、物体モデルも世界座標系の上に設定される。前節の投影変換を可能にするためには、世界座標系に設定された物体モデルをカメラ座標系で表現しなくてはならない。

一般に、三次元座標系の移動、回転、拡大縮小は、同次座標表現を用いると正方行列の演算のみで表現できる。ここでこれらの行列表現は全て逆行列を持つことに注意しよう。つまり、全ての移動・回転・拡大縮小は、どのような組み合わせも、あらかじめ唯一つの行列として表現しておくことが可能であり、その逆演算は、その逆行列のみを求めればよいことになる。

下記で T は (t_x, t_y, t_z) の移動を表現し、R_x, R_y, R_z は X,Y,Z 軸回りの $\theta_x, \theta_y, \theta_z$ 回転、S は X,Y,Z 軸方向への (s_x, s_y, s_z) 倍の拡大縮小を表す[2]。

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

$$R_y = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

[2]移動の正負、回転の正負に注意すること。

$$R_z = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

5　処理ポリゴン数の削減

一般に、物体モデルのポリゴン数は膨大であり、全てのポリゴンを処理するには莫大な計算量が必要となる。そこで、レンダリングに必要がないポリゴンを、レンダリング処理の早い段階で処理対象から効率よく外すことが重要になる。

物体モデルのデータの中には、平行投影／透視投影時に撮影されないポリゴンが含まれる。画像平面上の撮像領域から完全に逸脱した場所に投影されるポリゴンはレンダリングする必要がないので、処理の対象から削除する。

また通常は、物体の表側のみしか可視化する必要がない。もちろん、半透明な物体がある場合は別であるが、本演習では扱わない。そこで、面が視点に表を向けているかどうかを判定し、表を向いていない面を処理の対象から削除する。この判定は、そのポリゴンの法線 $\vec{n_p}$ とそのポリゴンからカメラへの視線ベクトル $\vec{e_p}$ との成す角度が $\pi/2$ 未満であるかどうかを調べることで実現できる。このとき、二つのベクトルの成す角度自体が必要なわけではないので、内積を使うと、上記の判定だけならば次のように少ない計算コストで行える。

$$\vec{n_p} \cdot \vec{e_p} \geq 0 \tag{9}$$

この方法は “cull face” または “backfacing” といい、3D ポリゴンブラウザではこれを切り替えられるようになっている場合もある。

6　ラスタライズとクリッピング

空間上の三角形ポリゴンは、3章の投影変換により画像平面上の三角形に変換される。これを実際に画像上に描画するためには、この実数表現である画像平面上の三角形を、ディジタル化された画素の集合に変換しなくてはならない。これを実現する方法はいろいろ考えられるが、リアルタイムでの CG 表示を念頭におくと、ハードウェアによる高速化等が可能なアルゴリズムにしておくことが望ましい。つまり、単純な計算に分解しやすいアルゴリズムや並列性の高いアルゴリズムを考えることが、チップ製造の容易さや高速化に結び付く。

本実験では、これを “ラスタライズ” と “クリッピング” という技法を用いて実現するが、これが高速化に有利な理由については各自がプログラム作成を行いながら考えてみてほしい。

さて、三角形を画素に分解するのがラスタライズで、これは 1 画素が画像平面上の三角形に対してどれほどの大きさになるかを計算し、三角形を X 軸方向に沿ってラスタ分解するものである。この X 軸方向に沿った 1 列を “スキャンライン” と呼び、スキャンラインごとに描画画素を決定する。

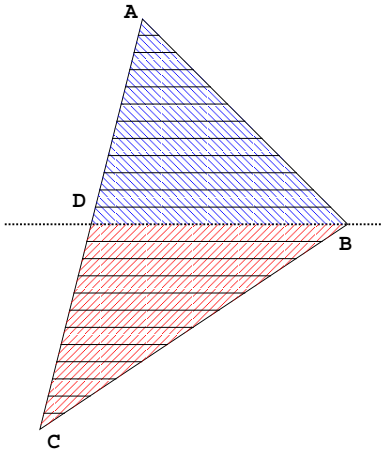

図 3: 三角形の Y 軸方向の分解

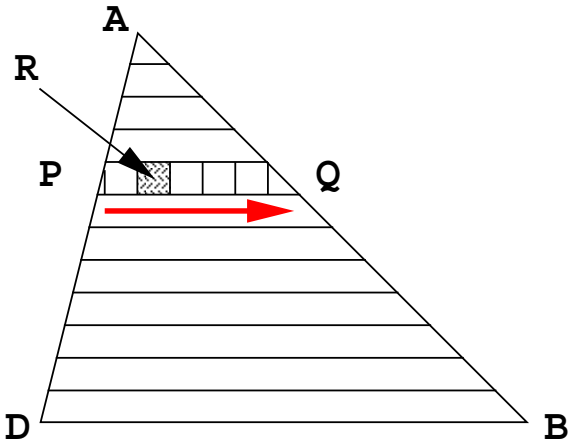

図 4: 三角形のラスタライズ

ラスタライズを簡単にする一つの方法は、画像平面上の三角形の頂点を Y 座標の値によってソートし、真中の値をもつ頂点によって三角形を二つに分割する方法であろう。例えば図 3の場合、三角形 ABC は三角形 ADB と三角形 CDB に分割される。

各三角形 ADB,CDB の各頂点の座標の計算は透視投影（並行投影）で計算できる。また、図 4の辺 PQ がスキャンラインであり、PQ 上の点 R の値は頂点同士の内分比によって簡単に求められる。点 A の座標を $(x_a, y_a)^t$ などと表すと、計算式は以下のようになる。

$$\begin{pmatrix} x_p \\ y_p \end{pmatrix} = (1-s)\cdot\begin{pmatrix} x_a \\ y_a \end{pmatrix} + s\cdot\begin{pmatrix} x_d \\ y_d \end{pmatrix} \tag{10}$$

$$\begin{pmatrix} x_q \\ y_q \end{pmatrix} = (1-s)\cdot\begin{pmatrix} x_a \\ y_a \end{pmatrix} + s\cdot\begin{pmatrix} x_b \\ y_b \end{pmatrix} \tag{11}$$

$$s = \frac{AP}{AD} = \frac{AQ}{AB} \tag{12}$$

$$\begin{pmatrix} x_r \\ y_r \end{pmatrix} = (1-t)\cdot\begin{pmatrix} x_p \\ y_p \end{pmatrix} + t\cdot\begin{pmatrix} x_q \\ y_q \end{pmatrix} \tag{13}$$

$$t = \frac{PR}{PQ} \tag{14}$$

また、この画素計算において、画像の大きさを保持しておいて、一部が画像からはみ出すようなポリゴンの場合には、画像からはみだす部分については処理を行わないようにする。この処理を “クリッピング” という。

なお、本章で述べた処理において、実数表現の三角形を量子化された画素上へ変換する処理は、実際には、工夫を凝らさないと、輪郭線のギザギザが目立つ “ジャギー” といわれる現象を引き起こす。しかし本演習ではジャギーの発生に対して対策を施さなくともよい。

7 隠面処理とZバッファ

表を向いていて撮像領域に投影されるポリゴンをラスタライズして画素に変換していっても、実際にどの部分が可視状態か（どの面のどの部分が視点に一番近いか）を考慮しなくては、正常なCGにならない。このように描画する点が他のポリゴンによって隠蔽されていないかどうかを確認する処理を “隠面処理” という。これを判定する方法については様々なアルゴリズムが考案されているが、ここでは “Zバッファ法” について説明する。

Zバッファ法は隠面処理アルゴリズムの中ではもっとも簡単な部類に属する。まず、画素毎に n ビットの “Zバッファ” を用意する。物体や投影パラメータ、ハードウェアにもよるが、10ビットから32ビット程度取られていることが多い。Zバッファには、現在描画されている画素の、もとの空間におけるz座標値が保存される。

本演習では、カメラを原点に設定し、奥行きのz座標は遠いほど正になるように座標系をとっているので、z値が大きいほど遠くにある物体を表すことになる。

まず、処理の最初の段階において、各画素のZバッファに、そのZバッファのとり得る最大値を代入しておく。あとは、ポリゴンごとに、その内部の各画素を描画するときに、その画素のもとの空間におけるz座標を調べる。もしこのz座標が、その画素の現在のZバッファ値よりも小さければ、その点をその画素に描画すると共に、Zバッファの値をそのz座標で置き換える。このときのZバッファの値との比較を高速化するためには、透視投影の四角錐に上底と下底を設け、その間の距離を n ビットで表現できるように正規化して整数比較するのがよい（実数での大小比較より整数での大小比較のほうが計算量がかからない）。ただし本演習では、実装の簡便さのため、プログラム実装時には単純に実数値比較してよい。

なお、Zバッファの値の比較は、ラスタライズと同時に行うと効率がよい。これは、プログラムのループ構造は全く同一でよいからである。

Zバッファ法はポリゴンの処理順序（データの登場順序）を問わず、しかも画素単位で計算結果が求められるが、メモリを大量に消費するという問題点がある。しかし、ハードウェア化に適した手法なので、現在のCGハードウェアのほとんどが本方式を採用している。

8 シェーディング

CGにおいて、三次元空間中の物体のどの点が画像上のどの位置に描画されることになるのかについては、すでに3章および6章で述べた。ではその点にどのような色 (画素値) を設定すれば

よいのであろうか？ これを光と物体の特性と視点との関係から決定する処理が “シェーディング” である。

シェーディングのためには、光源の特性と物体の光の反射に関する属性、および環境モデル（大気）が必要になる。ただし、本演習では大気は透明であると仮定するので、環境モデルについては考慮しない。

8.1 光源モデル

光源には、大別して “平行光線” と “点光源” がある。平行光線は太陽のようなもので、無限遠に位置した点光源と見なすこともできる。この場合、光源方向と光の強さのみがモデルとして必要である。

一方、点光源はある一点から光が発せられているような光源であり、その点と物体との距離が明るさに大きく影響する。いま、点光源から発せられる光の強さを I、それが物体のある一点に到達した時点での光の強さを I_p、その点と点光源との距離を d とすると、$I_p \propto \frac{I}{d^2}$ の関係がある。この点光源のモデルを拡張すれば、蛍光灯のような線光源やパネルのような面光源をモデル化することも可能である。

本演習では以下のうち、一種類以上の光源を利用することとする。

(1) 平行光源

- 減衰しない。
- 光の強さは $[0.0, 1.0]$ の範囲に正規化された (R,G,B) で表現する。

(2) 点光源

- $I_p \propto \frac{I}{d^2}$ で減衰する（適宜変更した減衰モデルを用いてもよい）。
- 点光源の配置位置において、光の強さは $[0.0, 1.0]$ の範囲に正規化された (R,G,B) で表現する。

なお、光源自身が画像に投影される状況はモデル化が困難であるので、光源を画像に表示することは少ないが、もし表示する場合には、シェーディングの計算に用いる上のような光源モデルとは別に、光源自体の見え方のモデルを用意する場合が多い。

8.2 反射

光のエネルギーが物体の表面に到達すると、吸収／反射／透過（の幾つか）が同時に起こる。本節以降では、このうちの反射のみについて考える。

CG において、物体表面での光の反射は、“拡散反射” と “鏡面反射” の二種類の反射によってモデル化される場合が多い。これを “二色性反射モデル” という。

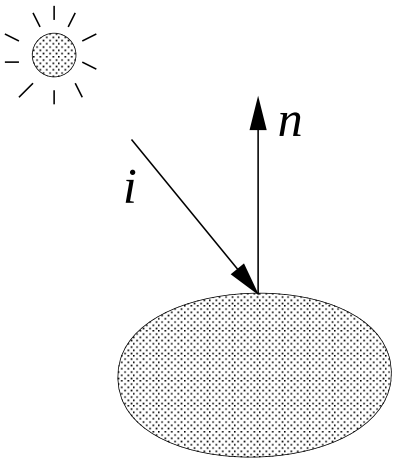

図 5: 拡散反射

8.2.1 拡散反射

拡散反射とは、光が物体表面の微細構造の中で、ランダムな反射を繰り返した後、再び表面から外に向かっていく場合をモデル化したものである。この拡散反射による反射光の強さは視点方向に依存しないため、物体上の各点の明るさは、同じ点ならばそれを見る方向に依らず同じ明るさとなる。これは特に布や木目などに顕著な傾向とされている。

今、物体のある点 P に対する入射光の方向を $\vec{i}$、強さを I_l、P での物体表面の法線ベクトルを $\vec{n}$ とする (図 5) [3]。ただしベクトルは全て単位ベクトルとする。Lambert の法則により、P における反射光の強さ I_d は、

$$I_d = -(\vec{i} \cdot \vec{n}) k_d I_l \tag{15}$$

で与えられる。ここで $k_d (0 \leq k_d \leq 1)$ は拡散反射係数 (VRML:diffuseColor) で、物体に固有の値である。また、$-(\vec{i} \cdot \vec{n})$ は $\vec{i}$ と $\vec{n}$ の成す角の余弦であり、$-(\vec{i} \cdot \vec{n})$ が正値でないときは反射はないとする。

8.2.2 鏡面反射

鏡面反射は、金属表面などでハイライトを生じるような反射をモデル化したものである。物体上でハイライトを生じる位置は、照明状況が変わらなくても、物体を見る方向を変えれば移動することからもわかるように、鏡面反射による反射光は、拡散反射の場合とは異なり、物体から視点位置に向かう視点方向によって変化する。鏡面反射は複雑な物理的属性に基づくが、本演習では、OpenGL でモデル化されている比較的簡単な鏡面反射モデルを用いる。いま、物体上の点 P から視点位置に向かう単位方向ベクトルを $\vec{e}$ で表すと、このモデルでは、P における視点方向への反射光の強さ I_s を次式で表現する。

[3] 三角形の法線は外積を用いると簡単に求められる。

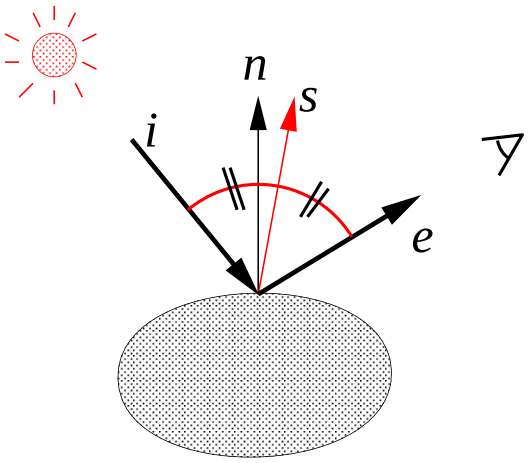

図 6: 鏡面反射

$$I_s = (\vec{s}\cdot\vec{n})^m k_s I_l \tag{16}$$

$$\vec{s} = \frac{\vec{e}-\vec{i}}{|\vec{e}-\vec{i}|} \tag{17}$$

ここで、I_l は入射光の強さ、$\vec{i}$ は光源から点 P への入射光の単位方向ベクトル、$\vec{n}$ は P における物体表面の単位法線ベクトル、k_s は物体の鏡面反射係数 (VRML:specularColor)($0 \leq k_s \leq 1$)、$m(\geq 0)$（実際には 1.0 以上でないとほとんど意味はない）は鏡面反射の強度 (VRML:shininess) を表す (図 6)。鏡面反射も拡散反射同様、$(\vec{s}\cdot\vec{n})$ が正値でないときは反射はないとするのが普通である。

8.2.3 環境光・環境反射

物体のシェーディングは基本的に拡散反射と鏡面反射に基づいて計算可能であるが、現実世界で問題になるのは入射光がどのようにして光源から物体表面に到達したかである。先にモデル化したような平行光源や点光源から物体表面に直接到達する光（直接光）は当然入射光の支配的な成分となるが、その他にも、それらの光源からの光が他の物体表面でいったん反射した後、物体表面に到達してくる光（間接光）も入射光に含まれる。このように物体の面同士が、それぞれの反射光によって互いを照らし合う現象を “相互反射” という。相互反射が存在する場合、物体表面は、直接光のみで照らされている場合と比べ、その分だけ明るくなる。この相互反射に起因する物体の明るさを厳密に計算する方法も提案されてはいるが、膨大な計算コストを要する。

そこで、相互反射に起因する物体の明るさを非常に大まかな近似で表現したものが “環境光” あるいは “環境反射” と呼ばれるものである。これは相互反射によって物体表面に入射する入射光を環境光 I_l（定数）でモデル化し、これによる物体表面の反射光 I_e を次式で表す。

$$I_e = k_a I_l \tag{18}$$

ここで、k_a は物体に固有の環境光係数 (VRML:ambientIntensity) で、 $0 \leq k_a \leq 1$ である。

8.2.4 反射のまとめ

拡散反射、鏡面反射、環境反射はそれぞれ RGB 成分毎に独立に計算し、複数の光源がある場合はその総和が最終的な物体の色となる。CG で画像を作成する際には、このときの RGB がそれぞれ各 8 ビットの量子化範囲を逸脱しないように調整しておく必要がある。

8.3 ポリゴンに対するシェーディング

前節までで述べた光と反射の計算モデルは、全て物体上の一点を対象にしたものであったが、物体上の全ての点について上記の各計算を行うと計算量は膨大なものになる。一方、本演習のようにポリゴンを用いた三次元物体の表現方法を採用している場合、物体の表現単位はポリゴンである。そこで、前節までの計算モデルをポリゴンに適用して各画素の色を決定するための方法について述べる。

8.3.1 コンスタントシェーディング

最も簡単な方法は、一つのポリゴンについてただ一つの面法線ベクトルと光源ベクトル、視点ベクトルを用い、そのポリゴンに対応する領域全てを単一色 (一定色:Constant color) で塗りつぶす方法である。反射モデルの計算に出てくる三角関数の計算がポリゴンごとにしか行われないので、計算量は少なくて済む。

もちろん実際にはポリゴンは大きさをもつので、その範囲内の各点に対する視点方向は同一ではなく (透視投影の場合)、また光源方向も同一ではない (点光源の場合)。しかし、一般に画像に対してポリゴンの投影される領域は比較的小さいので、この方法では同一ポリゴン上の各点毎の視点方向や光源方向の違いを無視する。その結果、このシェーディング法によって生成される CG 画像は、同じポリゴンは同一色で描画されることになるため、物体は、微小面から成る多面体に見える。

8.3.2 グーローシェーディング

ポリゴンによる物体モデルは通常、より滑らかな物体表面を平面で近似して表現したものであるので、その描画結果が多面体に見えてしまうのは望ましいことではない。そこで、物体の描画結果において、物体表面の各点で滑らかに色が移り変わるようにする方法として “グーローシェーディング (Gouraud Shading)” がある。この方法では、各面の輝度値 (色) を計算した後、隣り合う面同士の輝度値を用いて双一次関数で輝度値を補間し、滑らかな輝度値の変化になるようにする。

実際の計算は次のようになる。まず各頂点において面の法線ベクトルを求める。頂点の法線ベクトルは、その頂点に隣接する面全ての面法線ベクトルの平均をとる[4]。その法線ベクトルをもとに、頂点の輝度値 (色) を反射モデルに基づいて計算する。いま、投影されたポリゴン ABC について、辺 AB, AC 上の点を P, Q、辺 PQ 上の任意の点を R とする。

点 P, Q での輝度値 $I(P), I(Q)$ は、次式で表される。

$$I(P) = (1-s)\cdot I(A) + s\cdot I(B) \tag{19}$$

$$s = \frac{AP}{AB} \tag{20}$$

[4] 平均以外にも、立体角などで荷重平均をとる方法などが考えられる。

$$I(Q) = (1-t)\cdot I(A) + t\cdot I(C) \quad (21)$$
$$t = \frac{AQ}{AC} \quad (22)$$

これをもとに、点 R の輝度値 $I(R)$ は次式で計算される。

$$I(R) = (1-u)\cdot I(P) + u\cdot I(Q) \quad (23)$$
$$u = \frac{PR}{PQ} \quad (24)$$

点 P,Q が同一スキャンラインになるように s,t を決定すれば、ラスタライズアルゴリズムと同一走査で点 R の値を計算しシェーディングしていくことが可能になる。

8.3.3 フォーンシェーディング

グーローシェーディングは、コンスタントシェーディングに比べ、計算量の増加が少ない割に滑らかな外観を与えるので、三次元ポリゴンブラウザにはよく用いられるが、問題点もある。その一つは、双一次補間の対象が輝度値であるので、必ずしも面の法線ベクトルの向きを反映しないことである。この問題の解決を目指したのが “フォーンシェーディング (Phong Shading)” である。

フォーンシェーディングでは双一次補間の対象を輝度値の代わりに法線ベクトルとする。あとはグーローシェーディングとほぼ同様であるが、各点において法線ベクトルの値が全て異なるため、各点毎に反射モデルに基づく計算が必要になる。計算量は増大するが、特に鏡面反射などが多い場合には効果を発揮する。

8.4 テキスチャマッピング

通常、反射係数はポリゴンごとに 1 組与えることになるが、細かい模様が単一部分平面上に載っているような現実の物体をモデル化する場合には、そのような方法ではその模様中の同色領域ごとにポリゴンを用意することが必要となる。しかしこれではポリゴンモデルのデータサイズが大きくなるだけでなく、法線ベクトルの計算などにも無駄が発生する。

そこで、ポリゴンごとに反射係数を 1 組与える代わりに、そのポリゴンに合わせた反射係数マップを与えることを考える。この反射係数マップは、“テキスチャマップ” と呼ばれる。

テキスチャマッピングの問題点は、ラスタライズの過程で常にテキスチャマップを参照しなくてはならないことである。そのため、テキスチャマップを格納するメモリには非常な高速性が要求される。また、ポリゴンがどのような大きさで表示されるかが予測できない場合、どのようなテキスチャマップの解像度までを用意しておかなくてはならないかという検討が必要になる。

参考文献

[1] David F. Rogers, 山口富士夫監修,“実践コンピュータグラフィックス 基礎手続きと応用”, 日刊工業新聞社,ISBN4-526-02143-1,C3034

[2] David F.Rogers & J.Alan Adams, 山口富士夫監修,“コンピュータグラフィックス 第二版”, 日刊工業新聞社,ISBN4-526-03288-3,C3050

[3] 金谷健一, “画像理解 — 3 次元認識の数理—”, 森北出版, ISBN4-627-82140-9

[4] Jakkie Neider & Tom Devis & Mason Woo, “Open GL^{TM} Programming Guide (日本語版)”, 星雲社,ISBN4-7952-9645-6,C3055

[5] Linux Magazine PS2 Linux プログラミング, vol.4, No.2 - vol4. No.10 (2002 年 2 月号 - 2002 年 10 月号), 2002.

[6] 日経 CG, vol.115-118, (1996 年 4-6 月号,9 月号),ISSN0912-1609