コンピュータグラフィックスＳ 演習資料

第２回　ポリゴンモデルの描画

九州工業大学 情報工学部 システム創成情報工学科
講義担当：尾下真樹

1. 準備：前回の演習

本日の演習は、前回の演習で作成したプログラムを引き続き拡張していく。
もし、前回の演習を行っていない、もしくは、前回の演習で作成したプログラムを無くしてしまった場合は、最初のサンプルプログラムを Moodle からダウンロードし直して、前回の演習の内容を行った上で、今回の演習を行うこと。なお、前回の課題で自分が Moodle に提出したファイルをダウンロードすると、コメント中の日本語の文字コードが自動的に変換されて文字化けすることがあるので、そのような文字化けしたソースファイルは使わないこと。

2. 基本的なポリゴンモデルの描画

ポリゴンモデルの描画の演習に入る前に、まずは、GLUT を使った基本的なポリゴンモデルの描画を試してみる。GLUT には、あらかじめ用意されたポリゴンモデルを描画するための関数がいくつか用意されている。

以下のように、これまでのプログラムの、ポリゴンを描画していた処理をコメントアウトし、立方体を描画する GLUT の関数を呼び出してみる。glutSolidCube(GLdouble *size*) は、立方体をポリゴンモデルとして描画する関数である。引数 *size* には、立方体の辺の長さを指定する。また、立方体を描画する前に、色を指定している。

```
//
//   ウィンドウ再描画時に呼ばれるコールバック関数
//
void   display( void )
{
        ・・・・・・

        // 変換行列を設定（物体のモデル座標系→カメラ座標系）
        //（物体が (0.0, 1.0, 0.0) の位置にあり、静止しているとする）
        glTranslatef( 0.0, 1.0, 0.0 );
/*
        // 物体（１枚のポリゴン）を描画
        glBegin( GL_TRIANGLES );
                glColor3f( 0.0, 0.0, 1.0 );
                glNormal3f( 0.0, 0.0, 1.0 );
                glVertex3f(-1.0, 1.0, 0.0 );
                glVertex3f( 0.0,-1.0, 0.0 );
                glVertex3f( 1.0, 0.5, 0.0 );
        glEnd();
*/
        // 立方体を描画
        glColor3f( 1.0, 0.0, 0.0 );
        glutSolidCube( 1.5f );

        ・・・・・・
}
```

プログラムを上記のように修正し、コンパイル、実行して、立方体が描画されることを確認せよ。

次は、球を描画してみる。glutSolidSphere(GLdouble *radius*, GLdouble *slices*, GLdouble *stacks*) は、球を近似するポリゴンモデルを生成し、描画する関数である。引数、*slices*, *stacks* により、球をどれだけ細かいポリゴンモデルで近似するかを指定することができる。これらの値を大きくすることで、より滑らかな球が描画されることになる（ただし、その分、処理時間がかかることになる）。

```
void   display( void )
{
        ・・・・・・
/*
        // 立方体を描画
        glColor3f( 1.0, 0.0, 0.0 );
        glutSolidCube( 1.5f );
*/
        // 球を描画
        glColor3f( 1.0, 0.0, 0.0 );
        glutSolidSphere( 1.0, 8, 8 );

        ・・・・・・
}
```

プログラムを上記のように修正し、球（を近似したポリゴンモデル）が描画されることを確認せよ。
次に、球をより細かいポリゴンモデルで近似して描画してみる。

```
void   display( void )
{
        ・・・・・・

        // 球を描画
        glColor3f( 1.0, 0.0, 0.0 );
        glutSolidSphere( 1.0, 16, 16 );

        ・・・・・・
}
```

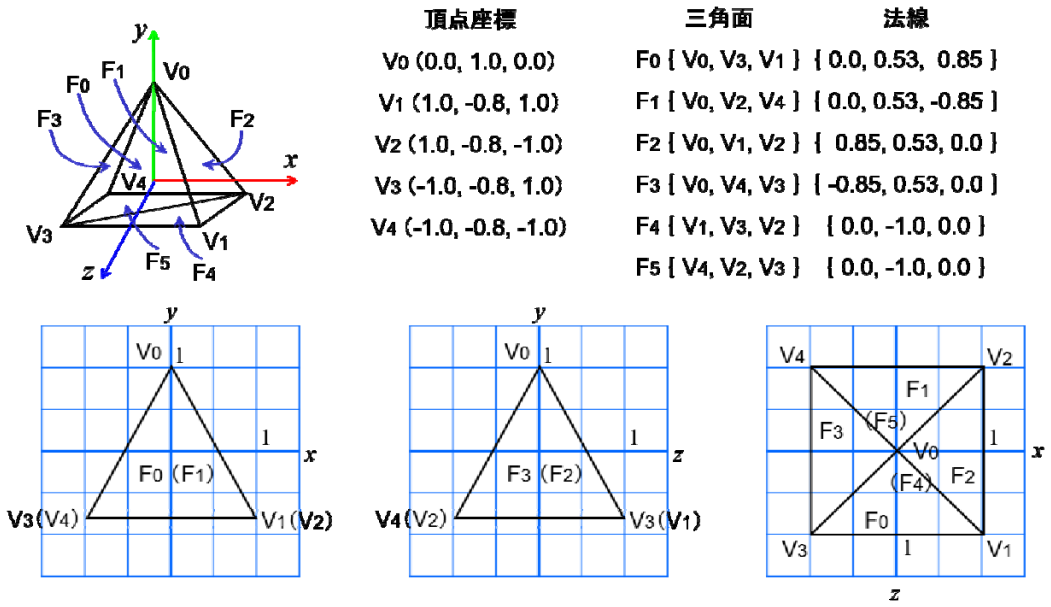
上記のように引数の値を変更し、再度実行して、描画されるポリゴンモデルがどのように変化するかを確認せよ。

3. ポリゴンモデルの描画

次に、1 枚のポリゴンの代わりに、もう少し複雑なポリゴンモデル（四角すい）の描画を行う。
OpenGL の関数を使ってポリゴンを描画する方法として、以下のようないくつかのやり方がある。

1. glVertex() 関数に直接頂点座標を記述する方法
2. 頂点データの配列を使う方法
3. 頂点データとインデックスの配列を使う方法
4. 頂点配列とインデックス配列を使う方法
5. 頂点配列を使う方法

本章では、基本的な最初の 3 つの方法を使って、以下のような、四角すいのポリゴンモデルを描画する。
（残りの 2 つの方法については、後日、余裕があれば、別の演習で扱う。）



3.1. 方法 1: glVertex() 関数に直接頂点座標を記述する方法

最も単純は、ここまでのプログラムと同様、glVertex() 関数の呼び出しをポリゴンモデルの頂点数分記述することで、OpenGL にポリゴンデータを渡す方法である。
以下のように、この方法を使って、四角すいを描画するための関数を追加する。(C では関数の呼び出しよりも前に関数が定義されていなければいけないので、dysplay 関数よりも上の適当な場所に追加する。)
この関数では、glBegin(GL_TRIANGES) ～ glEnd() を使用し、頂点座標と面の法線を指定することで、6 枚の三角面として四角すいを描画する。
なお、以下のプログラムリスト (list2-1.txt) は、講義のページに用意されているので、プログラムを修正するときには、用意されているプログラムリストをコピー&ペーストするなどして利用しても構わない。

List2-1.txt

```
//
//   角すいの描画（頂点データを関数内に直接記述する方法）
//
void   renderPyramid1()
{
        glBegin( GL_TRIANGLES );
                // +Z 方向の面
                glNormal3f( 0.0, 0.53, 0.85 );
                glVertex3f( 0.0, 1.0, 0.0 );
                glVertex3f(-1.0,-0.8, 1.0 );
                glVertex3f( 1.0,-0.8, 1.0 );

                // -Z 方向の面
                glNormal3f( 0.0, 0.53,-0.85 );
                glVertex3f( 0.0, 1.0, 0.0 );
                glVertex3f( 1.0,-0.8,-1.0 );
                glVertex3f(-1.0,-0.8,-1.0 );

                // +X 方向の面
                glNormal3f( 0.85, 0.53, 0.0 );
                glVertex3f( 0.0, 1.0, 0.0 );
                glVertex3f( 1.0,-0.8, 1.0 );
                glVertex3f( 1.0,-0.8,-1.0 );

                // -X 方向の面
                glNormal3f(-0.85, 0.53, 0.0 );
```

```
                    glVertex3f( 0.0, 1.0, 0.0 );
                    glVertex3f(-1.0,-0.8,-1.0 );
                    glVertex3f(-1.0,-0.8, 1.0 );

                    // 底面 1
                    glNormal3f( 0.0,-1.0, 0.0 );
                    glVertex3f( 1.0,-0.8, 1.0 );
                    glVertex3f(-1.0,-0.8, 1.0 );
                    glVertex3f( 1.0,-0.8,-1.0 );

                    // 底面 2
                    glNormal3f( 0.0,-1.0, 0.0 );
                    glVertex3f(-1.0,-0.8,-1.0 );
                    glVertex3f( 1.0,-0.8,-1.0 );
                    glVertex3f(-1.0,-0.8, 1.0 );
          glEnd();
}
```

次に、この関数を描画関数（dysplay() 関数）から呼び出す。
これまでの描画処理を削除（コメントアウト）して、四角すいを描画する関数を呼び出すように書き換える。

```
//
//   ウィンドウ再描画時に呼ばれるコールバック関数
//
void   display( void )
{
          ・・・・・・

          // 変換行列を設定（物体のモデル座標系→カメラ座標系）
          //（物体が (0.0, 1.0, 0.0) の位置にあり、静止しているとする）
          glTranslatef( 0.0, 1.0, 0.0 );
/*
          // 今までの描画処理は全てコメントアウト
*/
          // 角すいの描画
          glColor3f( 1.0, 0.0, 0.0 );
          renderPyramid1();

          ・・・・・・
}
```

以上の処理の追加により、四角すいが描画されることを、コンパイル・実行して確認する。

3.2. 方法 2: 頂点データの配列を使う方法

方法 1 では、全ての頂点ごとに glVertex() 関数の呼び出しを記述する必要があり、ポリゴン数が大きくなるとプログラムが長くなってしまう。そのため、ポリゴンデータの修正が難しいという問題がある。
また、ポリゴンデータがプログラム中に直接記述されているため、例えば、将来的にポリゴンデータをファイルから読み込んで表示するようなこともできない。
そこで、頂点データを配列に格納しておき、その配列のデータを使って描画を行う。

まず、以下のような、ポリゴンデータを構成する頂点配列を、renderPyramid() 関数の前にグローバル変数として定義する。ここでは、四角すいを構成する三角面 8 枚分について、各頂点の座標と法線の配列を、それぞれ配列に記述している。

List2-2.txt

```
//
//   角すいの全頂点配列
```

```
//

// 全頂点数
const int    num_full_vertices = 18;

// 全頂点の頂点座標
static float    pyramid_full_vertices[][ 3 ] = {
            { 0.0, 1.0, 0.0 }, {-1.0,-0.8, 1.0 }, { 1.0,-0.8, 1.0 },     // +Z 方向の面
            { 0.0, 1.0, 0.0 }, { 1.0,-0.8,-1.0 }, {-1.0,-0.8,-1.0 },     // -Z 方向の面
            { 0.0, 1.0, 0.0 }, { 1.0,-0.8, 1.0 }, { 1.0,-0.8,-1.0 },     // +X 方向の面
            { 0.0, 1.0, 0.0 }, {-1.0,-0.8,-1.0 }, {-1.0,-0.8, 1.0 },     // -X 方向の面
            { 1.0,-0.8, 1.0 }, {-1.0,-0.8, 1.0 }, { 1.0,-0.8,-1.0 },     // 底面 1
            {-1.0,-0.8,-1.0 }, { 1.0,-0.8,-1.0 }, {-1.0,-0.8, 1.0 } }; // 底面 1

// 全頂点の法線ベクトル
static float    pyramid_full_normals[][ 3 ] = {
            { 0.00, 0.53, 0.85 }, { 0.00, 0.53, 0.85 }, { 0.00, 0.53, 0.85 },     // +Z 方向の面
            { 0.00, 0.53,-0.85 }, { 0.00, 0.53,-0.85 }, { 0.00, 0.53,-0.85 },     // -Z 方向の面
            { 0.85, 0.53, 0.00 }, { 0.85, 0.53, 0.00 }, { 0.85, 0.53, 0.00 },     // +X 方向の面
            {-0.85, 0.53, 0.00 }, {-0.85, 0.53, 0.00 }, {-0.85, 0.53, 0.00 },     // -X 方向の面
            { 0.00,-1.00, 0.00 }, { 0.00,-1.00, 0.00 }, { 0.00,-1.00, 0.00 },     // 底面 1
            { 0.00,-1.00, 0.00 }, { 0.00,-1.00, 0.00 }, { 0.00,-1.00, 0.00 } }; // 底面 1
```

次に、これらの配列を使って四角すいを描画する、renderPyramid2() 関数を追加する。

```
//
//   角すいを描画（方法２：頂点データの配列を使う方法）
//
void   renderPyramid2()
{
            int    i;

            glBegin( GL_TRIANGLES );
            for ( i=0; i<num_full_vertices; i++ )
            {
                        glNormal3f( pyramid_full_normals[i][0], pyramid_full_normals[i][1], pyramid_full_normals[i][2] );
                        glVertex3f( [        ?        ], [        ?        ], [        ?        ] );
            }
            glEnd();
}
```

上のプログラムでは、頂点座標を指定する部分を空欄にしているので、各自、正しく描画が処理されるように、プログラムを追加すること。（適切な配列の要素を、関数の引数に指定する。）
また、追加した新しい描画関数（renderPyramid2() 関数）が呼び出されるように、描画処理も書き換える。

```
//
//   ウィンドウ再描画時に呼ばれるコールバック関数
//
void   display( void )
{
            ・・・・・・

            // 角すいの描画
            glColor3f( 1.0, 0.0, 0.0 );
//          renderPyramid1();    // 削除またはコメントアウト
            renderPyramid2();

            ・・・・・・
}
```

以上の処理により、前のプログラムと同様に四角すいが描画されることを、コンパイル・実行して確認せよ。
プログラムの実行結果自体は前の方法と変わらないが、このようなプログラムにすることで、ポリゴンデー

タと描画処理が分離されるので、ポリゴンデータの修正などがやりやすくなる。

3.3. 方法 3: 頂点データとインデックスの配列を使う方法

ポリゴンモデルのデータを記述しやすくするため、同じ四角すいのポリゴンデータを、頂点の配列と三角面インデックスの配列に分けて定義し、それらを使って描画する方法を試してみる。
以下のような、ポリゴンデータを表す配列を、renderPyramid() 関数の前にグローバル変数として定義する。

List2-3.txt

```
//
//   角すいの頂点配列＋三角面インデックス配列
//

// 角すいの頂点数
const int   num_pyramid_vertices = 5;

// 角すいの三角面数
const int   num_pyramid_triangles = 6;

// 角すいの頂点座標の配列
float   pyramid_vertices[][ 3 ] = {
          { 0.0, 1.0, 0.0 },
          { 1.0,-0.8, 1.0 },
          { 1.0,-0.8,-1.0 },
          {-1.0,-0.8, 1.0 },
          {-1.0,-0.8,-1.0 }
};

// 三角面インデックス（各三角面を構成する頂点の頂点番号）の配列
int   pyramid_tri_index[][ 3 ] = {
          { 0,3,1 }, { 0,2,4 }, { 0,1,2 }, { 0,4,3 }, { 1,3,2 }, { 4,2,3 }
};

// 三角面の法線ベクトルの配列（三角面を構成する頂点座標から計算）
float   pyramid_tri_normals[][ 3 ] = {
          { 0.00, 0.53, 0.85 }, // +Z 方向の面
          { 0.00, 0.53,-0.85 }, // -Z 方向の面
          { 0.85, 0.53, 0.00 }, // +X 方向の面
          {-0.85, 0.53, 0.00 }, // -X 方向の面
          { 0.00,-1.00, 0.00 }, // -Y 方向の面（底面 1）
          { 0.00,-1.00, 0.00 }  // -Y 方向の面（底面 2）
};
```

次に、これらの配列を使って四角すいを描画する、renderPyramid3() 関数を追加する。

```
//
//   角すいの描画（頂点の座標データ＋三角面インデックス＋三角面の法線・色データの配列を使用）
//
void   renderPyramid3()
{
          int   i, j;
          int   v_no;

          glBegin( GL_TRIANGLES );
          for ( i=0; i<num_pyramid_triangles; i++ )
          {
                    glNormal3f( pyramid_tri_normals[i][0], pyramid_tri_normals[i][1], pyramid_tri_normals[i][2] );
                    for ( j=0; j<3; j++ )
                    {
                              v_no = pyramid_tri_index[ i ][ j ];
                              glVertex3f(  ?  ,  ?  ,  ?  );
                    }
```

```
        }
        glEnd();
}
```

上のプログラムでも、頂点座標を指定する部分を空欄にしているので、各自、正しく描画が処理されるように、プログラムを追加すること。（適切な配列の要素を、関数の引数に指定する。）

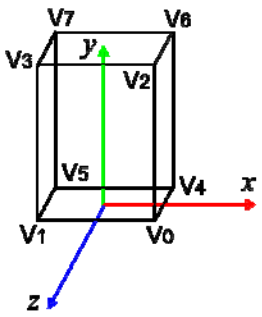
また、新しい関数が呼び出されるように、描画処理も書き換える。

```
//
//   ウィンドウ再描画時に呼ばれるコールバック関数
//
void   display( void )
{
        ・・・・・・

        // 角すいの描画
        glColor3f( 1.0, 0.0, 0.0 );
//      renderPyramid1();    // 削除またはコメントアウト
//      renderPyramid2();    // 削除またはコメントアウト
        renderPyramid3();

        ・・・・・・
}
```

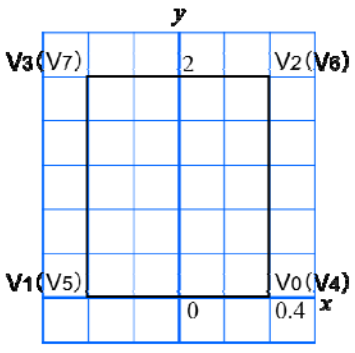
同様に、以上の処理により、同じ四角すいが描画されることを、コンパイル・実行して確認せよ。

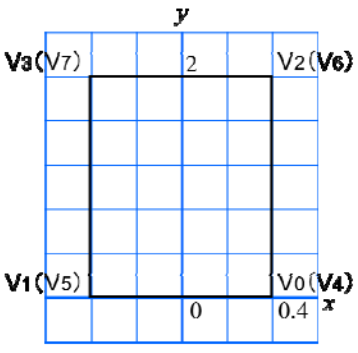
4. 別のポリゴンモデルの描画（直方体）

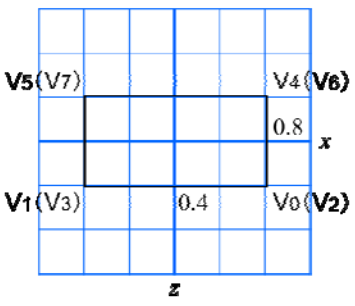
次に、これまでに学習した方法の練習として、今度は、以下のような別のポリゴンモデル（直方体）を描画してみる。



頂点座標	四角面	法線
V0 (0.8, 0.0, 0.4)	[V2, V3, V1 , V0]	[0.0, 0.0, 1.0]
V1 (-0.8, 0.0, 0.4)	[V7, V6, V4 , V5]	[0.0, 0.0, -1.0]
V2 (0.8, 2.0, 0.4)	[V2, V0, V4 , V6]	[1.0, 0.0, 0.0]
V3 (- 0.8, 2.0, 0.4)	[V3, V7, V5 , V1]	[-1.0, 0.0, 0.0]
V4 (0.8, 0.0, -0.4)	[?]	[?]
V5 (-0.8, 0.0, -0.4)	[V0, V1, V5 , V4]	[0.0, -1.0, 0.0]
V6 (0.8, 2.0, -0.4)		
V7 (- 0.8, 2.0, -0.4)		







ここでは、前章で学習した方法 1～方法 3 のうち、直方体のポリゴンモデルを最も容易に記述できる方法 3 のやり方を使って描画を行う。また、今回のポリゴンモデルは全て四角面で構成されるので、三角面の代わりに、四角面を使って描画する。

まずは、以下のように、直方体の頂点データ＋四角面インデックスデータを定義する。
直方体は全ての面が四角面で構成されるので、6 枚の四角面として描画する。
四角面を用いることで、さきほどの四角すいと異なるのは、四角面のインデックス（cube_index）が、4 個の頂点をとることである。また、各面の色のデータも配列として表現するように追加する。

List2-4.txt

```
//
//   直方体の頂点配列＋四角面インデックス配列
//
const int   num_cube_vertices = 8;
const int   num_cube_quads = 6;

// 頂点座標の配列
float   cube_vertices[][ 3 ] = {
          { 0.8, 0.0, 0.4 }, // 0
          {-0.8, 0.0, 0.4 }, // 1
          { 0.8, 2.0, 0.4 }, // 2
          {-0.8, 2.0, 0.4 }, // 3
          { 0.8, 0.0,-0.4 }, // 4
          {-0.8, 0.0,-0.4 }, // 5
          { 0.8, 2.0,-0.4 }, // 6
          {-0.8, 2.0,-0.4 }, // 7
};

// 四角面インデックス（各四角面を構成する頂点の頂点番号）の配列
int   cube_index[][ 4 ] = {
          { 2,3,1,0 }, { 7,6,4,5 }, { 2,0,4,6 }, { 3,7,5,1 }, {    ?    }, { 0,1,5,4 } };

// 四角面の法線ベクトルの配列（四角面を構成する頂点座標から計算）
float   cube_normals[][ 3 ] = {
          { 0.00, 0.00, 1.00 },
          { 0.00, 0.00,-1.00 },
          { 1.00, 0.00, 0.00 },
          {-1.00, 0.00, 0.00 },
          {          ?          },
          { 0.00,-1.00, 0.00 } };

// 四角面のカラーの配列
float   cube_colors[][ 3 ] = {
          { 0.00, 1.00, 0.00 },
          { 1.00, 0.00, 1.00 },
          { 1.00, 0.00, 0.00 },
          { 0.00, 1.00, 1.00 },
          { 0.00, 0.00, 1.00 },
          { 1.00, 1.00, 0.00 } };

//
//   直方体を描画（頂点の座標データ＋四角面インデックス＋四角面の法線・色データの配列）
//
void   renderCube()
{
          int   i, j;
          int   v_no;

          glBegin( GL_QUADS );
          for ( i=0; i<num_cube_quads; i++ )
          {
                    glNormal3f( cube_normals[i][0], cube_normals[i][1], cube_normals[i][2] );
                    glColor3f( cube_colors[i][0], cube_colors[i][1], cube_colors[i][2] );
```

```
                for ( j=0; j<4; j++ )
                {
                        v_no = cube_index[ i ][ j ];
                        glVertex3f( cube_vertices[ v_no ][0], cube_vertices[ v_no ][1], cube_vertices[ v_no ][2] );
                }
        }
        glEnd();
}
```

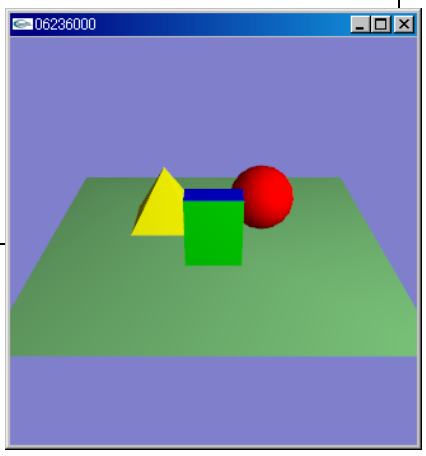
なお、上のプログラムでは、1 枚の四角面の頂点番号と法線を空欄にしているので、各自、抜けている四角面が正しく描画されるような頂点番号の組、及び、法線ベクトルを追加すること。

5. ポリゴンモデルの描画のまとめ

最後に、今回の演習で扱ったポリゴンモデルをまとめて描画してみる。
変換行列を利用して、３つのポリゴンモデルを異なる位置に描画する。変換行列については後日の講義・演習で学習するので、ひとまず今回は、以下のサンプルプログラムをそのまま使用する。

List2-5.txt

```
void   display( void )
{
        ・・・・・・
/*
        もとの変換行列の設定はコメントアウト
        // 変換行列を設定（物体のモデル座標系→カメラ座標系）
        //（物体が (0.0, 1.0, 0.0) の位置にあり、静止しているとする）
        glTranslatef( 0.0, 1.0, 0.0 );
*/
/*
        // 今までの描画処理も全てコメントアウト
*/

        // 球を描画
        glPushMatrix();
                glTranslatef( 1.5, 1.0, -1.0 );
                glColor3f( 1.0, 0.0, 0.0 );
                        ?
        glPopMatrix();

        // 角すいの描画
        glPushMatrix();
                glTranslatef( -1.5, 1.0, -1.0 );
                glColor3f(    ?    );
                renderPyramid3();
        glPopMatrix();

        // 直方体の描画
        glPushMatrix();
                glTranslatef( 0.0, 0.0, 1.0 );
                renderCube();
        glPopMatrix();

        ・・・・・・
}
```



右のスクリーンショットと同様の画面を実現するように、上記のプログラム中の空欄を埋めよ。
プログラムを作成したら、正しい結果が描画されるか、実行して確認する。