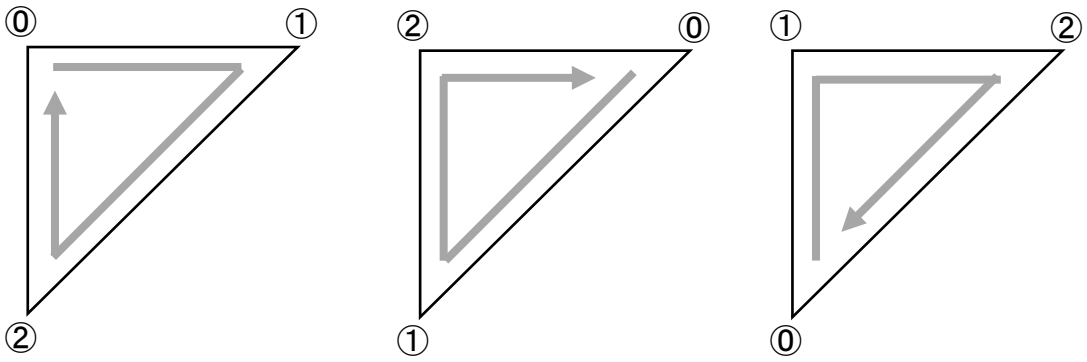
後期課題2

5日で理解する3Dプログラミング 「ポリゴン編」

■ポリゴンの描画

ポリゴンの描画は三角形を一つの単位とします。従って4角形を表示するには三角形2枚のポリゴンを描画します。
尚、DirectX の場合、時計回りに描画点を指定します。

例) 時計回りであればどこから開始しても良い。



ポリゴン描画の基本形

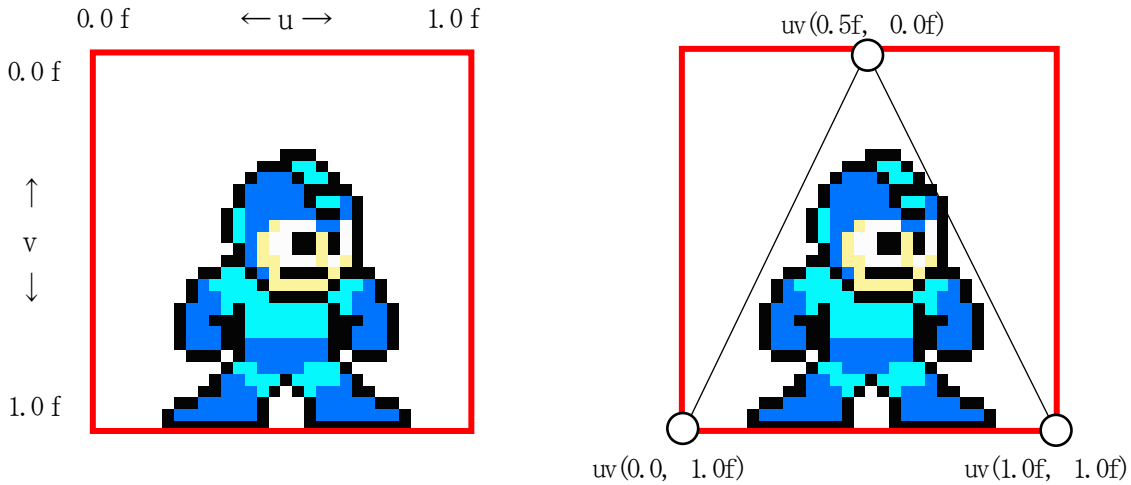
- 「頂点の座標」・・・頂点の座標。(x, y, z)の三次元空間の座標で設定する。
- 「法線」・・・・・・・・ポリゴンになった時の向きをベクトルで表す。ライティングに影響される。
- 「頂点の色」・・・・ディフューズカラー(拡散光色)。基本的なポリゴンの色。
- 「反射色」・・・・・・スペキュラーカラー(反射色)。
- 「uv」・・・・・・・・・・テクスチャ座標。左上を(u:0.0f, v:0.0f)、右下を(u:1.0f, v:1.0f)の座標系で指定する。

頂点データを扱う構造体
VERTEX3D 型

```
VECTOR pos;      // (x, y, z)で頂点座標を設定する。
VECTOR norm;     // (x, y, z)で法線ベクトルを保持する。
COLOR_U8 dif;    // (r, g, b, α)で色を管理する。
COLOR_U8 spc;    // (r, g, b, α)で色を管理する。
float u, v;      // u(横),v(縦)で 0.0f～1.0f でテクスチャ座標を指定する。
```

UVの基本形

UVの指定は、ポリゴンの頂点の位置を画像の割合いで指定するイメージになります。
左上を基準に、0～1の値で位置を示します。



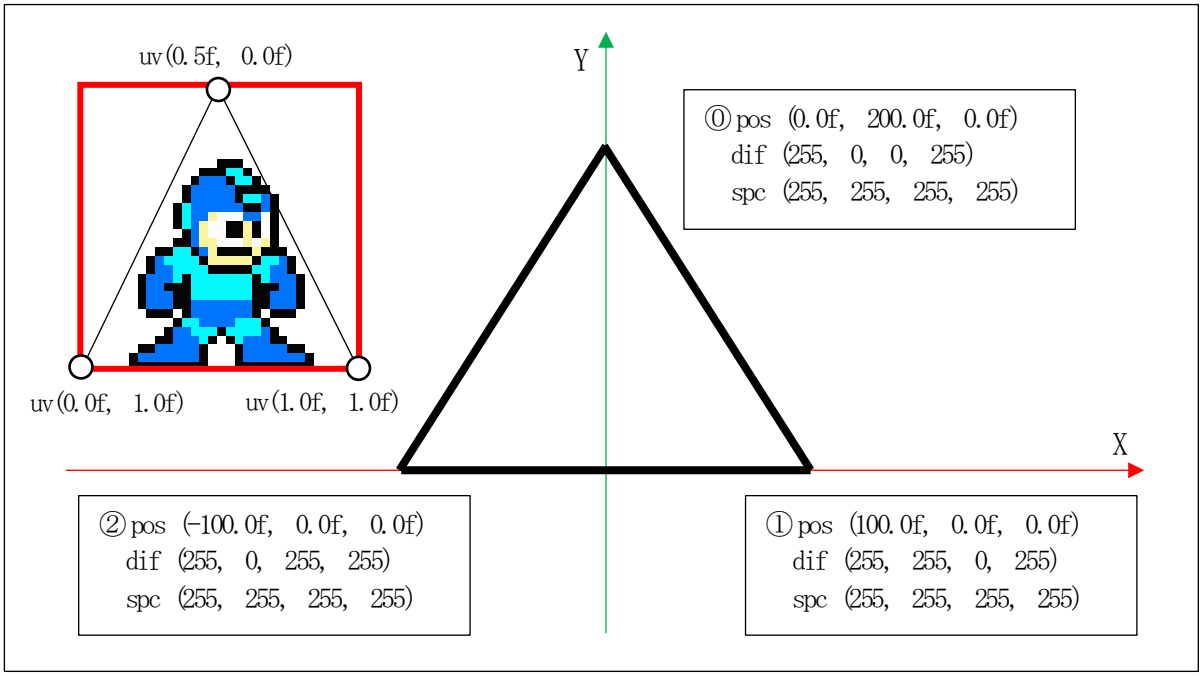
ポリゴン描画の基本形

3D空間に「頂点データ（VERTEX3D の配列）」を元に「描画する枚数分」のポリゴンを描画します。
ポリゴン1枚に3つの頂点が必要なので、頂点データの配列は描画する枚数×3個の頂点データが必要です。
又、テクスチャーを貼る事も出来ますが貼り付ける画像のサイズは2のn乗のピクセルサイズの必要があるので、
8、16、32、64、128、256、512、1024などのサイズになります。※2048以上はスペック的にNG。
尚、DrivationGraph や LoadDivGraph などで読み込んだ画像は使用できないので注意が必要です。

３D空間に三角形ポリゴンを描画する。
int DrawPolygon3D(頂点配列のポインタ, 描画するポリゴン数, 画像のハンドル, 透明フラグ);

頂点配列のポインタ・・・VERTEX3D 型の配列で 3 頂点単位で配列を作成する。
描画するポリゴン数・・・3 頂点で 1 単位で描画する枚数となる。
画像のハンドル・・・・・LoadGraph で取得した int 型の ID。
※サイズは(8, 16, 32, 64, 128, 256, 512, 1024)
透明フラグ・・・・・・・png 形式などで透明情報を有効にするか。

1）演習① 図の様な三角形を描画する



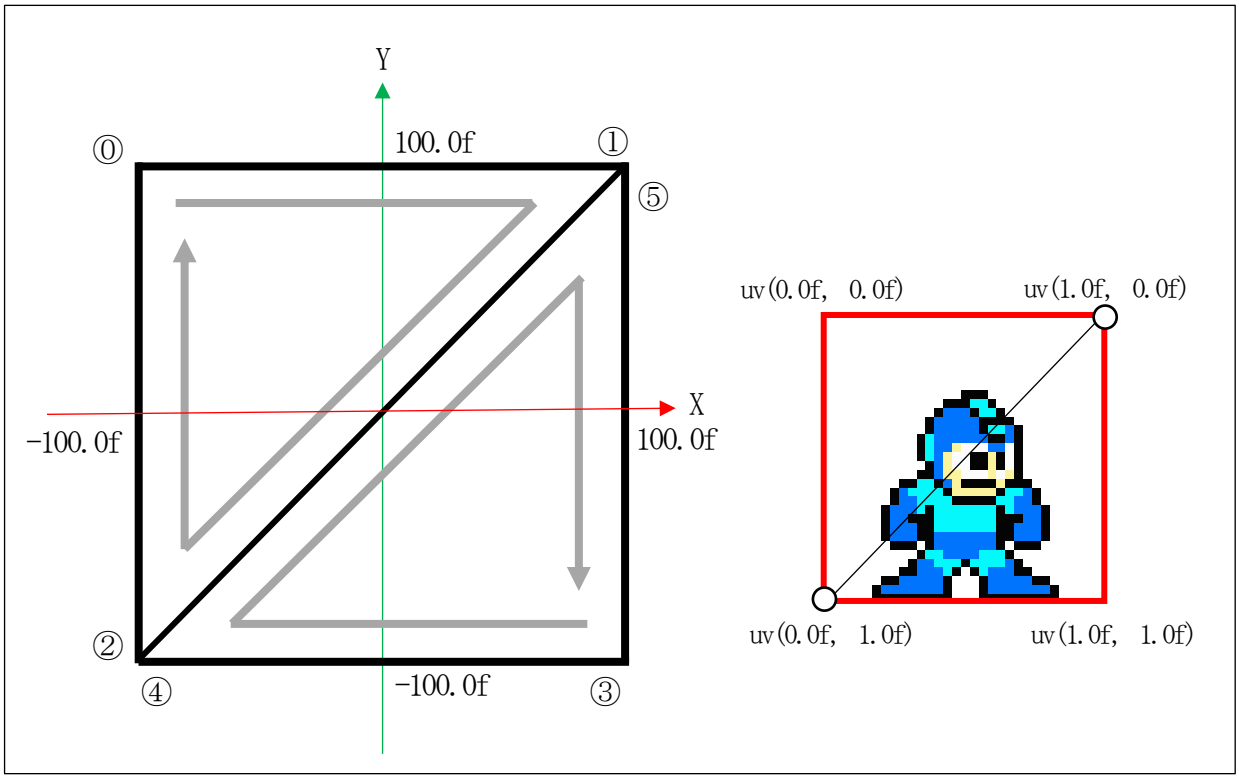
■使用変数

```
VERTEX3D Vertex[3]; // 3点分の頂点ﾃﾞｰﾀ配列
int tecture;        // 画像ﾊﾝﾄﾞﾙ用
```

■変数初期化、描画

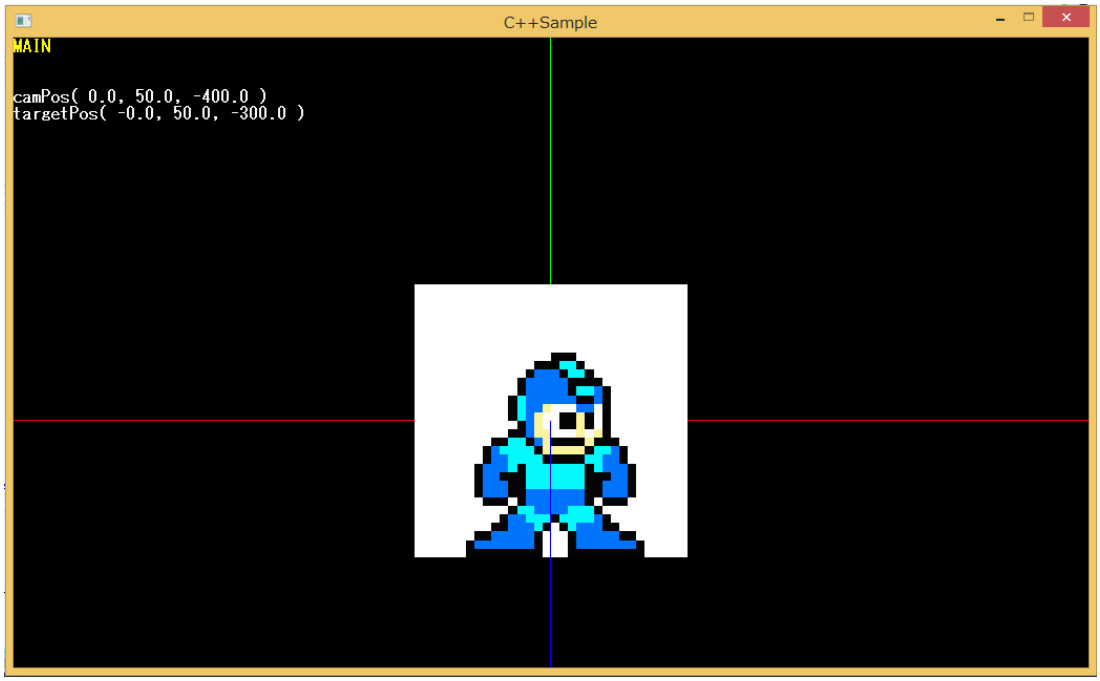
```
texture = LoadGraph("image/sample.png");    // 画像読み込み(ﾌｧｲﾙ名等は任意)

Vertex[0].pos = VGet(0.0f, 200.0f, 0.0f);
Vertex[0].dif = GetColorU8(255, 0, 0, 255);
Vertex[0].spc = GetColorU8(255, 255, 255, 255);
Vertex[0].u = 0.5f;
Vertex[0].v = 0.0f;

Vertex[1].pos = VGet(100.0f, 0.0f, 0.0f);
Vertex[1].dif = GetColorU8(255, 255, 0, 255);
Vertex[1].spc = GetColorU8(255, 255, 255, 255);
Vertex[1].u = 1.0f;
Vertex[1].v = 1.0f;

Vertex[2].pos = VGet(-100.0f, 0.0f, 0.0f);
Vertex[2].dif = GetColorU8(255, 0, 255, 255);
Vertex[2].spc = GetColorU8(255, 255, 255, 255);
Vertex[2].u = 0.0f;
Vertex[2].v = 1.0f;

// ﾎﾟﾘｺﾞﾝ描画
DrawPolygon3D(Vertex, 1, texture, false);
```

2)演習② 図の様な四角形を描画する



ポリゴンがライトの影響を受けて暗い感じになる場合は「ライトを無効」にするセッティングがあります。

```
// ﾗｲﾃｨﾝｸﾞをしない
SetUseLighting(false);
```



■頂点座標の「移動」

ポリゴンを上下左右などに移動させるには、それぞれの頂点座標に移動量を加算すれば良い。

頂点の移動後のX座標　＝　頂点のX座標　＋　移動量
頂点の移動後のY座標　＝　頂点のY座標　＋　移動量
頂点の移動後のZ座標　＝　頂点のZ座標　＋　移動量

例）上下左右でポリゴンを移動させる。

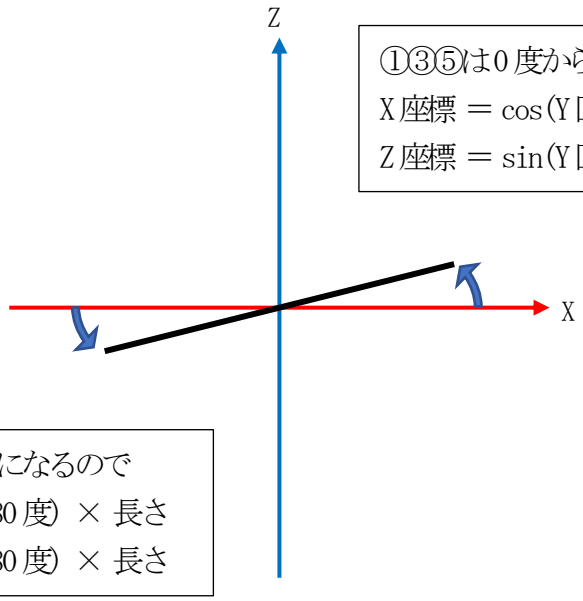
処理　※頂点⓪のみ上下左右移動

```
// 移動座標確定
if(KeyMng::GetInstance().newKey[P1_RIGHT]) Vertex[0].pos.x += 2.0f;
if(KeyMng::GetInstance().newKey[P1_LEFT]) Vertex[0].pos.x -= 2.0f;
if(KeyMng::GetInstance().newKey[P1_UP]) Vertex[0].pos.y += 2.0f;
if(KeyMng::GetInstance().newKey[P1_DOWN]) Vertex[0].pos.y -= 2.0f;
```

■頂点座標の「回転」

ポリゴンを回転させる場合には、それぞれの頂点座標を三角関数を使用して軸回転を行う。

（Y軸回転の場合）
回転後の頂点のX座標 ＝ ｃｏｓ（Y軸の回転量）× 移動量
回転後の頂点のZ座標 ＝ ｓｉｎ（Y軸の回転量）× 移動量



①③⑤は0度からの回転になるので
X座標 ＝ cos(Y回転量) × 長さ
Z座標 ＝ sin(Y回転量) × 長さ

⓪②④は180度からの回転になるので
X座標 ＝ cos(Y回転量+180度) × 長さ
Z座標 ＝ sin(Y回転量+180度) × 長さ

□頂点座標の「移動」「回転」を両方同時に行う。

移動と回転を同時に行うには、下記の手順を行う。

①移動量を確定

↓

②回転量を確定

↓

②回転後のポリゴンの座標を計算する。
（それぞれの頂点の座標を算出）

↓

③移動後のポリゴンの座標を計算する。

↓

④ポリゴンを描画する。

キー入力により移動後の座標を確定させる。

キー入力により回転後のＹ軸の回転値を確定させる。

原点を基準に回転後の頂点座標を計算する。

回転後の頂点に移動後の座標を加算する。

最終的な座標を毎回計算する。

課題）

原点を中心にXY平面に描いた（－Z方に向いた）４角形ポリゴンを、ポリゴンの中心からY軸回転しながら上下左右に移動させる様に完成させなさい。

■軸回転

X、Y、Z軸の回転を複合すると、これまでの軸回転の考えでは**うまくいきません。**

それぞれの回転の状態をまとめる場合は「行列式」が有効です。行列を使用すると「回転」「移動」「拡大縮小」をひとまとめにできるので、3D空間の管理には管理が容易になります。

ですので**「行列は絶対に利用」**しましょう！

□X軸に角度分回転した時の頂点座標

X軸にθ回転

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

(x, y, z)の座標をX軸にθ分回転した時の(X, Y, Z)

```
X = ( x * 1 ) + ( y * 0 ) + ( z * 0 );
Y = ( x * 0 ) + ( y * cosθ ) + ( z * -sinθ );
Z = ( x * 0 ) + ( y * sinθ ) + ( z * cosθ );
```

□Y 軸に角度分回転した時の頂点座標

Y軸にθ回転

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

(x, y, z)の座標をY軸にθ分回転した時の(X, Y, Z)

```
X = ( x * cosθ ) + ( y * 0 ) + ( z * sinθ );
Y = ( x * 0 ) + ( y * 1 ) + ( z * 0 );
Z = ( x * -sinθ ) + ( y * 0 ) + ( z * cosθ );
```

□Z 軸に角度分回転した時の頂点座標

Z軸にθ回転

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(x, y, z)の座標をZ軸にθ分回転した時の(X, Y, Z)

```
X = ( x * cosθ ) + ( y * -sinθ ) + ( z * 0 );
Y = ( x * sinθ ) + ( y * cosθ ) + ( z * 0 );
Z = ( x * 0 ) + ( y * 0 ) + ( z * 1 );
```

□【参考】任意軸に角度分回転した時の頂点座標

任意軸(NX, NY, NZ)でθ回転

$$\begin{bmatrix} (NX*NX)*(1-\cos\theta)+\cos\theta & (NX*NY)*(1-\cos\theta)-NZ*\sin\theta & (NX*NZ)*(1-\cos\theta)+NY*\sin\theta \\ (NY*NX)*(1-\cos\theta)+NZ*\sin\theta & (NY*NY)*(1-\cos\theta)+\cos\theta & (NY*NZ)*(1-\cos\theta)-NX*\sin\theta \\ (NZ*NX)*(1-\cos\theta)+NY*\sin\theta & (NZ*NY)*(1-\cos\theta)+NX*\sin\theta & (NZ*NZ)*(1-\cos\theta)+\cos\theta \end{bmatrix}$$

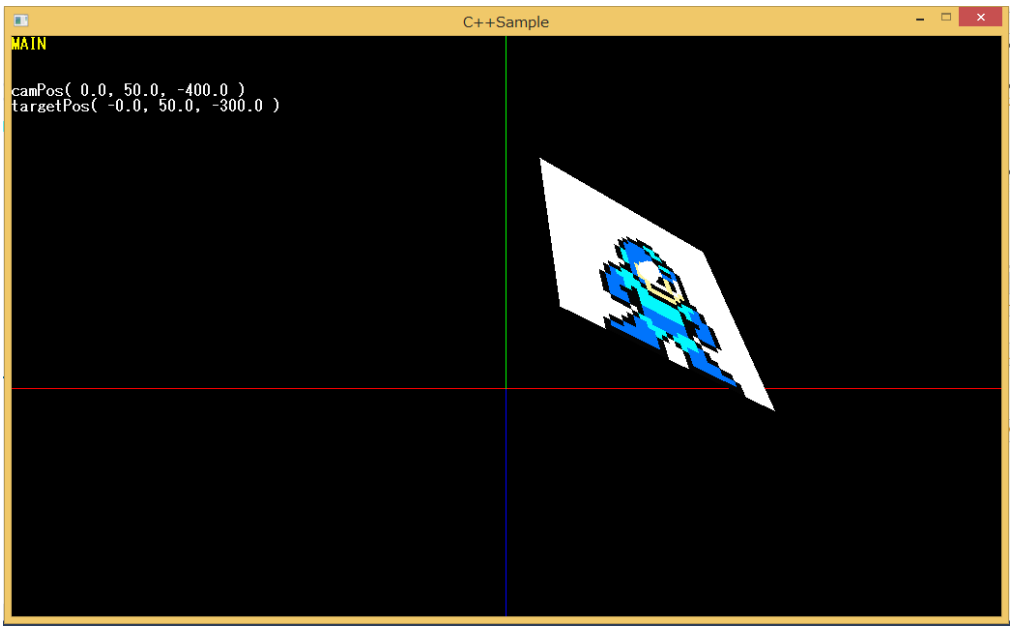
(x, y, z)の座標を(NX, NY, NZ)軸にθ分回転した時の(X, Y, Z)

```
X =   x * ( ( NX * NX ) * ( 1 - cosθ ) + cosθ )
    + y * ( ( NX * NY ) * ( 1 - cosθ ) - NZ * sinθ )
    + z * ( ( NX * NZ ) * ( 1 - cosθ ) + NY * sinθ );

Y =   x * ( ( NY * NX ) * ( 1 - cosθ ) + NZ * sinθ )
    + y * ( ( NY * NY ) * ( 1 - cosθ ) + cosθ )
    + z * ( ( NY * NZ ) * ( 1 - cosθ ) - NX * sinθ );

Z =   x * ( ( NZ * NX ) * ( 1 - cosθ ) + NY * sinθ )
    + y * ( ( NZ * NY ) * ( 1 - cosθ ) + NX * sinθ )
    + z * ( ( NZ * NZ ) * ( 1 - cosθ ) + cosθ );
```

課題)

　原点を中心にXY平面に描いた(－Z方に向いた)４角形ポリゴンを、ポリゴンの中心から「Y軸回転」「X軸回転」「Z軸回転」し、なおかつ上下左右に移動させる様に完成させなさい。

※常に画面から見たX、Y、Z軸での回転。



「A」［D］・・・・・・・・Y軸回転
「W」「S」・・・・・・・・X軸回転
「左Shift」「左Ctrl」・・・Z軸回転
「カーソルキー」・・・・・ 画面上での上下左右

■行列を使おう

行列を使うにはMATRIX型の構造体(４×４行列)を使用します。
上記の計算を自前で行うにはあまりにも大変なので、ここは潔く「行列」と「行列の計算式」を使用しましょう。

4×4の行列を準備する
MATRIX型

m・・・[4][4]の二次元配列でfloat値を管理する。

m[0][0]	m[0][1]	m[0][2]	m[0][3]
m[1][0]	m[1][1]	m[1][2]	m[1][3]
m[2][0]	m[2][1]	m[2][2]	m[2][3]
m[3][0]	m[3][1]	m[3][2]	m[3][3]

X軸の回転後の回転行列を取得する
MATRIX MGetRotX(回転値);

例)
```
MATRIX matrix = MGetRotX( rolX );  // X軸にrolX回転した回転行列を返す。
```

Y軸の回転後の回転行列を取得する
MATRIX MGetRotY(回転値);

例)
```
MATRIX matrix = MGetRotY( rolY );  // Y軸にrolY回転した回転行列を返す。
```

Z軸の回転後の回転行列を取得する
MATRIX MGetRotZ(回転値);

例)
```
MATRIX matrix = MGetRotZ( rolZ );  // Z軸にrolZ回転した回転行列を返す。
```

平行移動行列を取得する
MATRIX MGetTranslate(平行移動ベクトル);

例)
```
MATRIX matrix = MGetTranslate( vector );  // vectorの平行移動を行う行列を返す。
```

行列の合成（行列の掛け算）
MATRIX MMult(行列A, 行列B);

例）
MATRIX matC = MMult(matA, matB); // matA とmatB を合成した結果をmatC に入れる。

行列を使ったベクトルの変換
VECTOR VTransform(変換後のベクトル, 変換に使用する行列);

例）
VECTOR vect = VTransform(vector, mat); // vector をmat で変換してvect に入れる。

例）回転量(rol.x, rol.y, rol.z)、移動量(pos.x, pos.y, pos.z)で変換した頂点座標(x, y, z)を算出する。

```
MATRIX transformMatrix; // 回転・移動行列
VECTOR rol;             // 回転値
VECTOR pos;             // 移動値
VERTEX3D Vertex[6];     // 頂点ﾃﾞｰﾀ

// ※初期化は省略

// 行列に回転を合成
transformMatrix = MGetRotX(rol.x);  // X軸回転行列
transformMatrix = MMult(transformMatrix, MGetRotY(rol.y));  // Y軸回転を合成
transformMatrix = MMult(transformMatrix, MGetRotZ(rol.z));  // Z軸回転を合成

// 行列に移動を合成
transformMatrix = MMult(transformMatrix, MGetTranslate(VGet(pos.x, pos.y, pos.z)));

// 頂点座標に「移動・回転行列」に合わせて変換
Vertex[0].pos = VTransform(VGet(-100.0f,  100.0f, 0.0f), transformMatrix);
Vertex[1].pos = VTransform(VGet( 100.0f,  100.0f, 0.0f), transformMatrix);
Vertex[2].pos = VTransform(VGet(-100.0f, -100.0f, 0.0f), transformMatrix);
Vertex[3].pos = VTransform(VGet( 100.0f, -100.0f, 0.0f), transformMatrix);
Vertex[4].pos = Vertex[2].pos;
Vertex[5].pos = Vertex[1].pos;
```