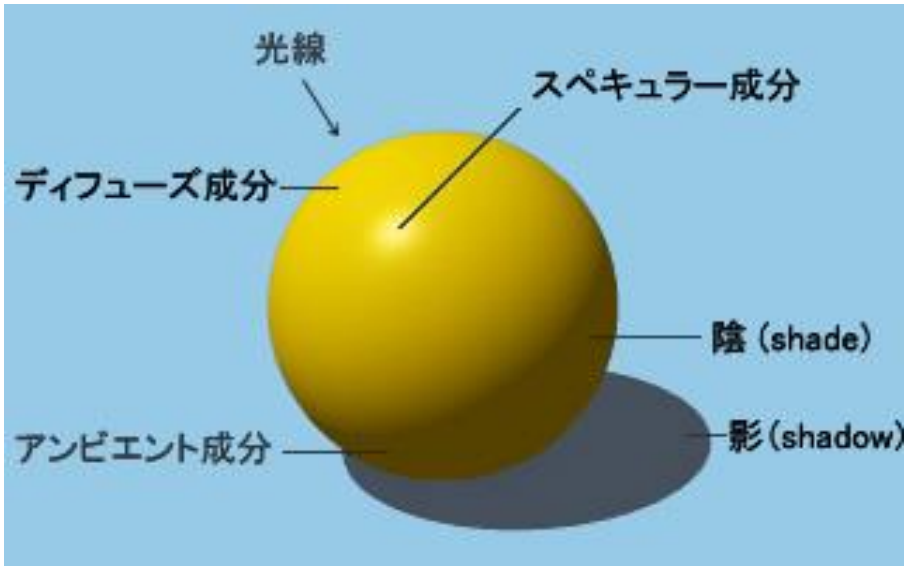
20 日で理解する3Dプログラミング 「その 3:プリミティブ」

■そもそも 3DCG の仕組みとは？

1)レンダリングをして画像を得るためには、物体の色やシェーディングと言われる属性を与える必要があります。



2)オブジェクトに、面の向きに応じた陰（shade）をつける事をシェーディングという。※shadow ではありません。

・アンビエント（ambient）

周辺光の強さ。光源からの光が全く当たらない陰の部分はこの成分だけで明るさが決まります。
0%で真っ黒になる。立体の置かれた環境に合わせて調整します。

・ディフューズ（diffuse）

物体に当たった光の拡散反射を示します。粗さを大きくすると反射を抑えてツヤ消しとなります。
一般的にはランバート法と呼ばれる入射角の cos に比例する計算を行います（他にトゥーンなどがある）。

・スペキュラー（specular）

表面の光沢の反射成分。ハイライトの強さとなり光る部分のサイズも変えられます。

・スペキュラーの強さ(pow)

スペキュラーハイライトの角度範囲を決定する値です。値が小さい程範囲が狭くなります。

これらの値により質感の表現ができる。
・プラスチック系　　：スペキュラーを鈍く弱くする。
・ガラス系　　：アンビエントとディフューズを弱くしスペキュラーを鋭くする。
・光沢のある金属系　：アンビエントとディフューズを弱くしスペキュラーを強く鋭くする。

3)他にもあります。

・エミッシブ（emissive）

自己発光色の設定で、ライトに影響されない光？となります。物体が光っている様に見えるイメージです。
通常では、ライトを使用しないと真っ暗な物体になりますが、自己発光の設定で色を付ける事ができます。

■プリミティブ(基本図形)「球型」

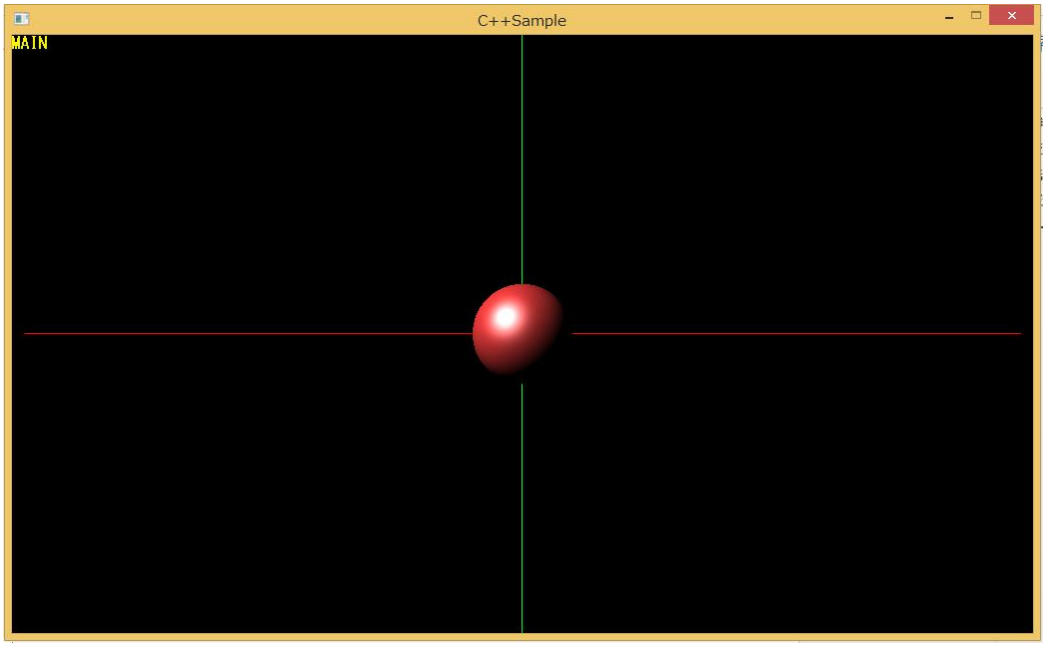
球の描画

int DrawSphere3D(中心座標, 球の半径, ポリゴンの細かさ, 色, 反射の色, 塗りつぶすか);

```
VECTOR pos;             // 中心座標(x, y, z)
float r;                // 半径
int divNum;             // 細かさ
unsigned int difColor;  // 物体色：ﾃﾞｨﾌｭｰｽﾞ(R, G, B)
unsigned int spcColor;  // 反射色：ｽﾍﾟｷｭﾗｰ(R, G, B)
int fillFlag;           // 塗りつぶし(0：線のみ、1:塗り有り)
```

プログラム例

```
pos = VGet(0.0f, 0.0f, 0.0f);
r = 50.0f;
divNum = 32;
difColor = GetColor(255, 64, 64);
spcColor = GetColor(255, 255, 255);
fillFlag = 1;

DrawSphere3D(pos, r, divNum, difColor, spcColor, fillFlag);
```



■プリミティブ(基本図形)「カプセル型」

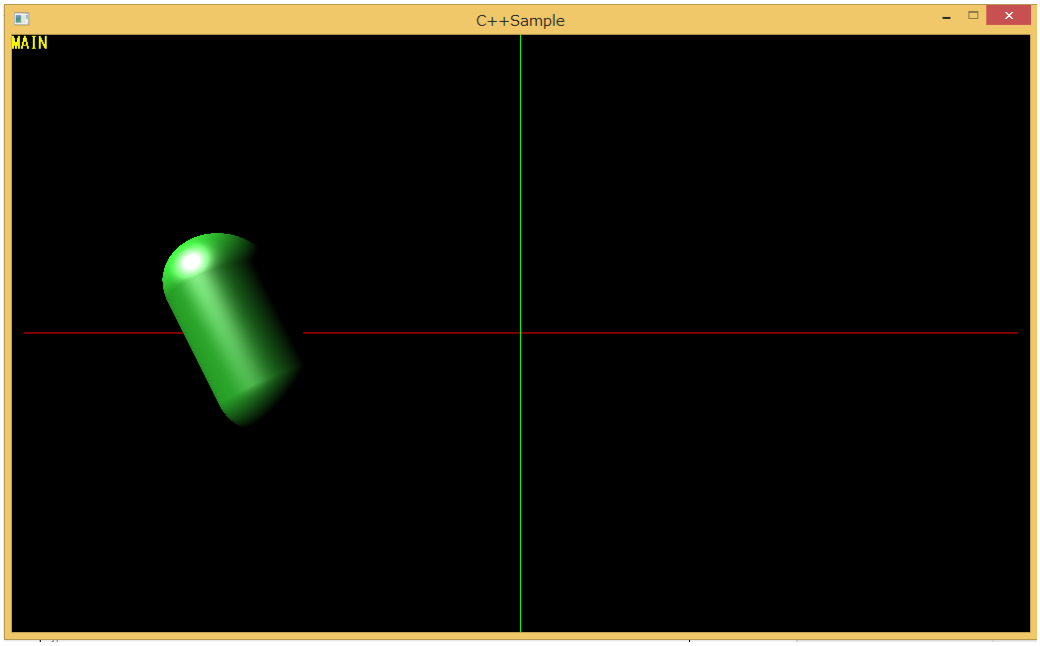
カプセルの描画

int DrawCapsule3D(1 点目の座標, 2 点目の座標, 幅, 細かさ, 色, 反射の色, 塗りつぶすか);

```
VECTOR pos1;              // 1点目の中心座標(x, y, z)
VECTOR pos2;              // 2点目の中心座標(x, y, z)
float r;                  // ｶﾌﾟｾﾙの幅(半径)、先端の半球の半径
int divNum;               // 細かさ
unsigned int difColor;    // 物体色：ﾃﾞｨﾌｭｰｽﾞ(R, G, B)
unsigned int spcColor;    // 反射色：ｽﾍﾟｷｭﾗｰ(R, G, B)
int fillFlag;             // 塗りつぶし(0：線のみ、1:塗り有り)
```

プログラム例

```
pos1 = VGet(-300.0f, 50.0f, 0.0f);
pos2 = VGet(-250.0f, -50.0f, 0.0f);
r = 50.0f;
divNum = 32;
difColor = GetColor(64, 255, 64);
spcColor = GetColor(255, 255, 255);
fillFlag = 1;

DrawCapsule3D(pos1, pos2, r, divNum, difColor, spcColor, fillFlag);
```



■プリミティブ(基本図形)「円錐型」

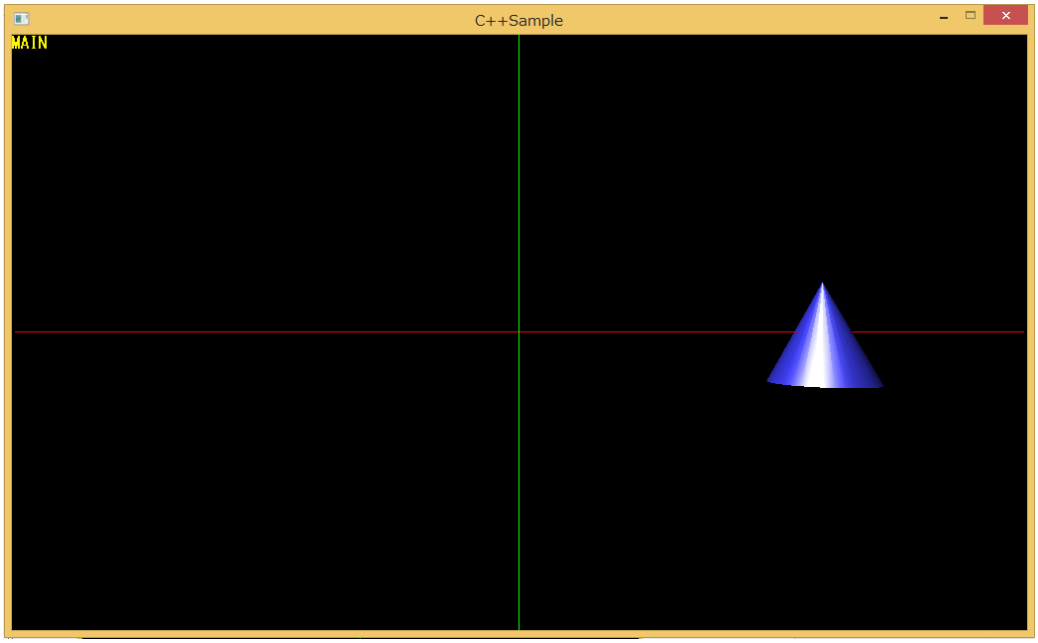
円錐の描画

int DrawCone3D(頂点座標, 底面の中心座標, 底面の半径, 細かさ, 色, 反射の色, 塗りつぶすか);

```
VECTOR top;             // 頂点座標(x, y, z)
VECTOR bottom;          // 底面の中心座標(x, y, z)
float r;                // 底面の半径
int divNum;             // 細かさ
unsigned int difColor;  // 物体色：ﾃﾞｨﾌｭｰｽﾞ(R, G, B)
unsigned int spcColor;  // 反射色：ｽﾍﾟｷｭﾗｰ(R, G, B)
int fillFlag;           // 塗りつぶし(0：線のみ、1:塗り有り)
```

プログラム例

```
top = VGet(300.0f, 50.0f, 0.0f);
bottom = VGet(300.0f, -50.0f, 0.0f);
r = 50.0f;
divNum = 32;
difColor = GetColor(64, 64, 255);
spcColor = GetColor(255, 255, 255);
fillFlag = 1;

DrawCone3D(top, bottom, r, divNum, difColor, spcColor, fillFlag);
```



□マテリアルパラメータ全部設定する方法

各パラメータのメンバー変数がセットになった MATERIALPARAM 構造体を使用し「SetMaterialPara()関数」で一気に設定する事ができます。

1)マテリアル関係をひとまとめにした構造体

```
// ―― マテリアル構造体での変数定義
MATERIALPARAM 変数名;
```

2)3D で使用する COLOR_F 型で色指定する場合に使用する。

```
// ―― 浮動小数点型の値を取得(もしくは設定)する
GetColorF(float red, float green, float blue, float alpha);

float Red  :赤成分の輝度( 0.0f ～ 1.0f )
float Green  :緑成分の輝度( 0.0f ～ 1.0f )
float Blue  :青成分の輝度( 0.0f ～ 1.0f )
float Alpha  :アルファ成分( 0.0f ～ 1.0f )
```

※0～255 ではなく 0.0f～1.0f なので注意

3)マテリアル構造体でのメンバー変数の役割り。

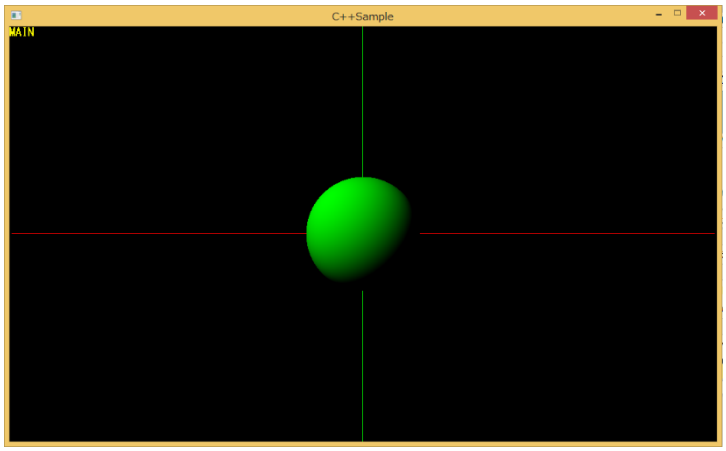
```
GetColorF Diffuse;   // 拡散光色。初期状態では無視される。
                     // ※SetMaterialUseVertDefColor()で使用するかの設定。

GetColorF Ambient; // 環境光色。ライトの色と掛け合わせて使用される。

GetColorF Emissive; // 自己発光色。ライトがなくても光る。

GetColorF Specular; // 反射光色。初期状態では無視される。
                    // ※SetMaterialUseVertSpcColor()で使用するかの設定。

float Power;        // スペキュラハイライトの角度範囲を決定する。
```

4)マテリアルパラメータを設定する。

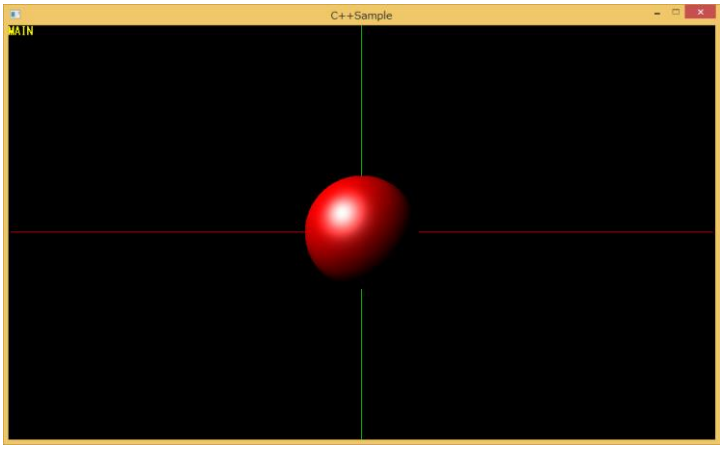
```
// ―― マテリアルパラメータによる設定
int SetMaterialParam(MATERIALPARAM Material);
```

5)設定の一例。

```
// ----- ﾏﾃﾘｱﾙの個別設定
MATERIALPARAM Material; // 変数定義

Material.Diffuse = GetColorF(0.0f, 1.0f, 0.0f, 1.0f);
Material.Specular = GetColorF(0.0f, 0.0f, 0.0f, 0.0f);
Material.Ambient = GetColorF(0.0f, 0.0f, 0.0f, 0.0f);
Material.Emissive = GetColorF(0.0f, 0.0f, 0.0f, 0.0f);
Material.Power = 20.0f;

SetMaterialUseVertDifColor(false);  // 頂点ﾃﾞｰﾀのﾃﾞｨﾌｭｰｽﾞでなくﾏﾃﾘｱﾙ設定の方を使用する
SetMaterialUseVertSpcColor(false);  // 頂点ﾃﾞｰﾀのｽﾍﾟｷｭﾗｰでなくﾏﾃﾘｱﾙ設定の方を使用する
SetMaterialParam(Material);         // ﾏﾃﾘｱﾙﾃﾞｰﾀのｾｯﾄ

// ----- 3D空間上に球を描画する
VECTOR pos = VGet(0.0f, 0.0f, 0.0f);                    // ﾓﾃﾞﾙﾃﾞｰﾀの中心座標
unsigned int difColor = GetColor(255, 0, 0);            // 頂点ﾃﾞｰﾀのﾃﾞｨﾌｭｰｽﾞをｾｯﾄ
unsigned int spcColor = GetColor(255, 255, 255);        // 頂点ﾃﾞｰﾀのｽﾍﾟｷｭﾗｰをｾｯﾄ

DrawSphere3D(pos, 80.0f, 32, difColor, spcColor, true); // ﾓﾃﾞﾙﾃﾞｰﾀの描画
```



頂点データの色設定を使用した場合　SetMaterialUseVertDifColor(False);



マテリアルの色設定を使用した場合　SetMaterialUseVertDifColor(true);

■プリミティブ(基本図形)「三角形の板」

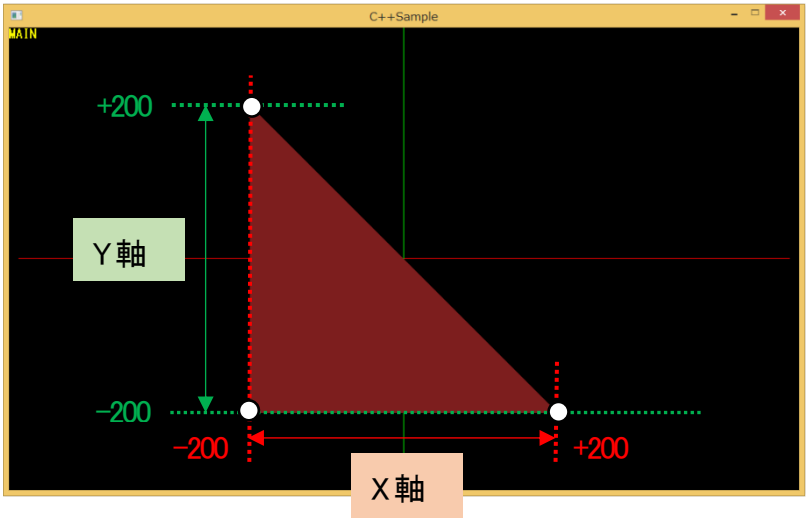
三角形の描画

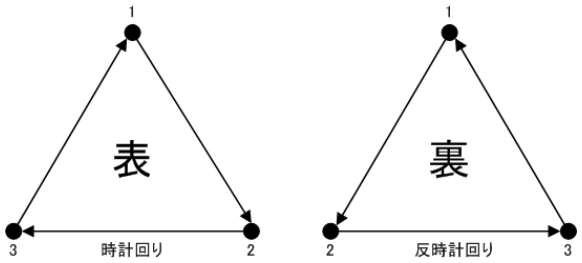
int DrawTriangle3D(頂点１の座標, 頂点２の座標, 頂点３の座標, 色, 塗りつぶすかどうか);

```
VECTOR pos1;        // 三角形の頂点1の座標(x, y, z)
VECTOR pos2;        // 三角形の頂点2の座標(x, y, z)
VECTOR pos3;        // 三角形の頂点3の座標(x, y, z)
unsigned int color; // 三角形の色(R, G, B)
int fillFlag;       // 塗り潰すかどうか(0:線のみ、1:塗る)
```

プログラム例

```
pos1 = VGet( 200.0f, -200.0f, 0.0f);
pos2 = VGet(-200.0f, -200.0f, 0.0f);
pos3 = VGet(-200.0f,  200.0f, 0.0f);
color = GetColor(125, 30, 30);
fillFlag = 1;
DrawTriangle3D(pos1, pos2, pos3, color, fillFlag);
```





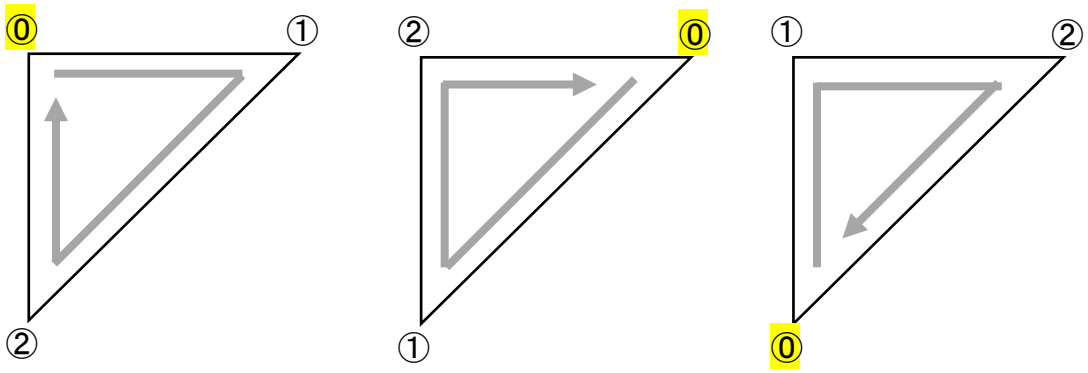
三角形の座標は時計回りの順番で指定します！

※時計回りで描画した方が表になります。右手座標系の場合は反対に半時計回りになります。

■プリミティブ(基本図形)「ポリゴン」

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。
3D 空間に引数 Vertex が示す頂点配列を元に PolygonNum 個の三角形ポリゴンの集合を描画します。
ポリゴン一枚に3つの頂点が必要なので、Vertex が示す配列には PolygonNum × 3個の頂点データが必要になります。

例) 時計回りであればどこから開始しても良いです。



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。

VERTEX3D 型　　頂点データを扱う構造体

```
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;   // ディフューズカラー(R, G, B, α)で色を管理する。
COLOR_U8 spc;   // スペキュラーカラー(R, G, B, α)で色を管理する。
float u, v;     // u(横),v(縦)とし 0.0f～1.0f でテクスチャ座標を指定する。

※uv は、指定したい画像の左上端 u = 0.0f、v = 0.0f、右下端 u = 1.0f、v = 1.0f とした
  座標で指定します。
```

COLOR_U8 構造体

COLOR_U8 は unsigned char r, g, b, a を持つ構造体で、輝度をそれぞれ 0 ～ 255 で表現します。
int r, g, b, a を引数として取り、それをそのまま COLOR_U8 構造体に代入して戻り値として返してくる関数 GetColorU8 を使用すると代入処理を簡素に書くことが出来ます。

GetColorU8 関数

GetColorU8 構造体の値を簡単に作成するための関数です。

int Red ： 取得したいカラー値の赤成分の輝度(0 ～ 255)
int Green ： 取得したいカラー値の緑成分の輝度(0 ～ 255)
int Blue ： 取得したいカラー値の青成分の輝度(0 ～ 255)
int Alpha ： 取得したいカラー値のアルファ成分(0 ～ 255)

例)
VERTEX3D vertex;
vertex.dif = GetColorU8(赤,　緑　青,　α); // VERTEX3D のメンバ変数 dif は COLOR_U8 型の変数

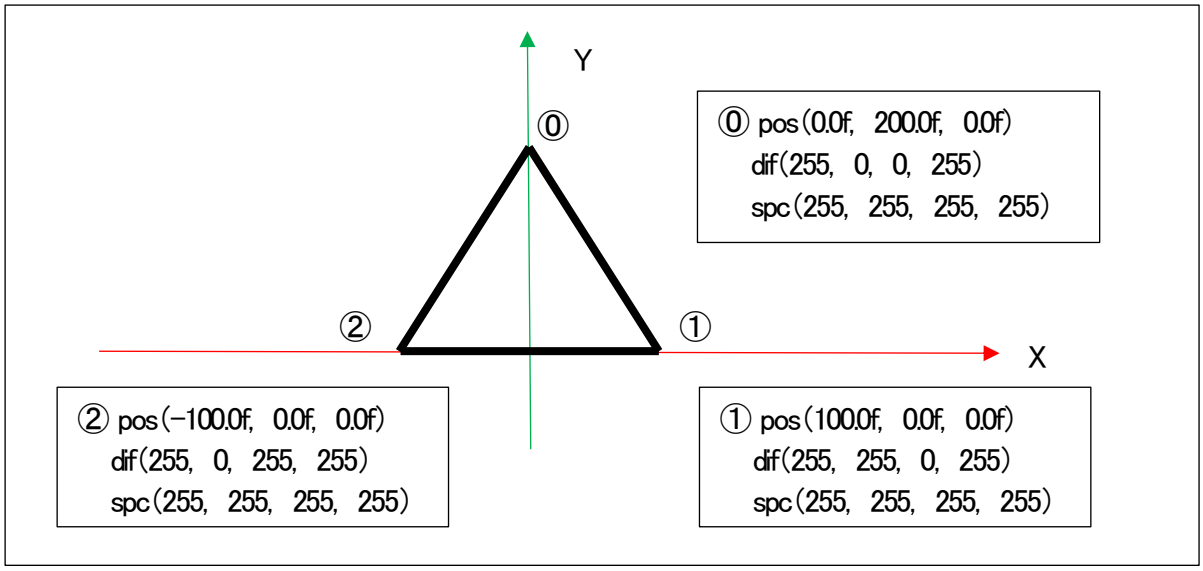
■ポリゴン描画命令

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。

※テクスチャー画像のサイズは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。

1）演習① 図の様な三角形を描画してみよう



■変数の準備

```
VERTEX3D vertex[3]; // 3点分の頂点データ配列
int texture;        // 画像ハンドル用
```

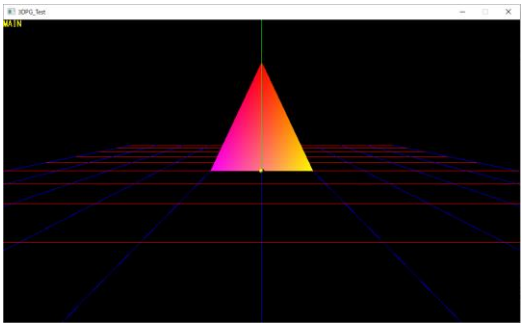
■変数初期化、描画

```
void Primitive::Init()
{
    texture = LoadGraph("texture/sample.png");

    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[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[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);
}

void Primitive::Render()
{
    SetUseLighting(false);

    DrawPolygon3D(vertex, 1, DX_NONE_GRAPH, false);

    SetUseLighting(true);
}
```

画像を張らない場合は DX_NONE_GRAPH

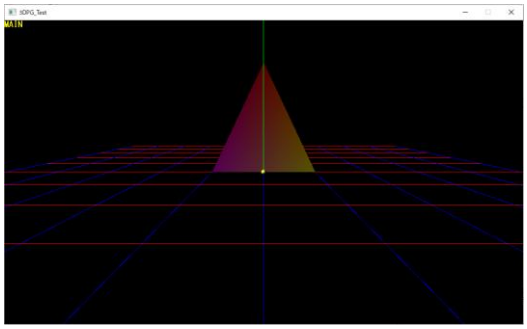
■ライトの影響

3D 描画をする際にライティング計算を行うかどうかを設定します。
頂点カラーとテクスチャカラーの色だけを反映させたい場合は、ライトの設定を無視して描画します。



ライト設定 OFF

```
SetUseLighting(false);
```



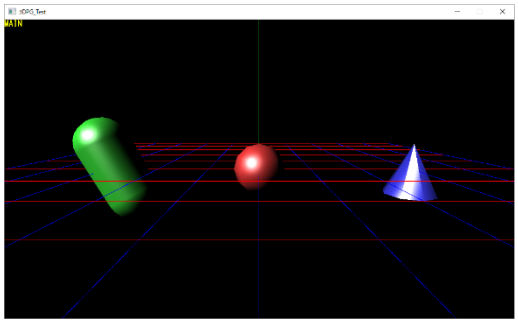
ライト設定 ON

```
SetUseLighting(true);
```

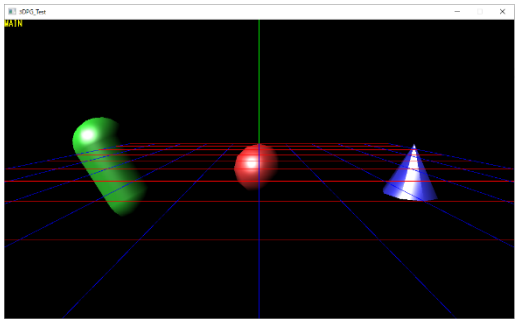
ライトの影響を受けるので、ライトの方向に依存した明るさ表現になります。図の状態は光の方向にポリゴンが向いていないので暗い感じに表示されています。

■Z バッファ

Zバッファは、主に3D空間にある物を前後関係を意識せずに都合の良い順番で描画するために使用するものです。2D の時は、描画する順番を変える事で前後関係を設定していましたが、Z バッファではカメラからの距離に応じて自動的に前後関係をかき分ける事が出来るようになります。



Z バッファ ON



Z バッファ OFF

Z バッファを有効にしてかつ、Z バッファを利用して描画するという2つの設定を行います

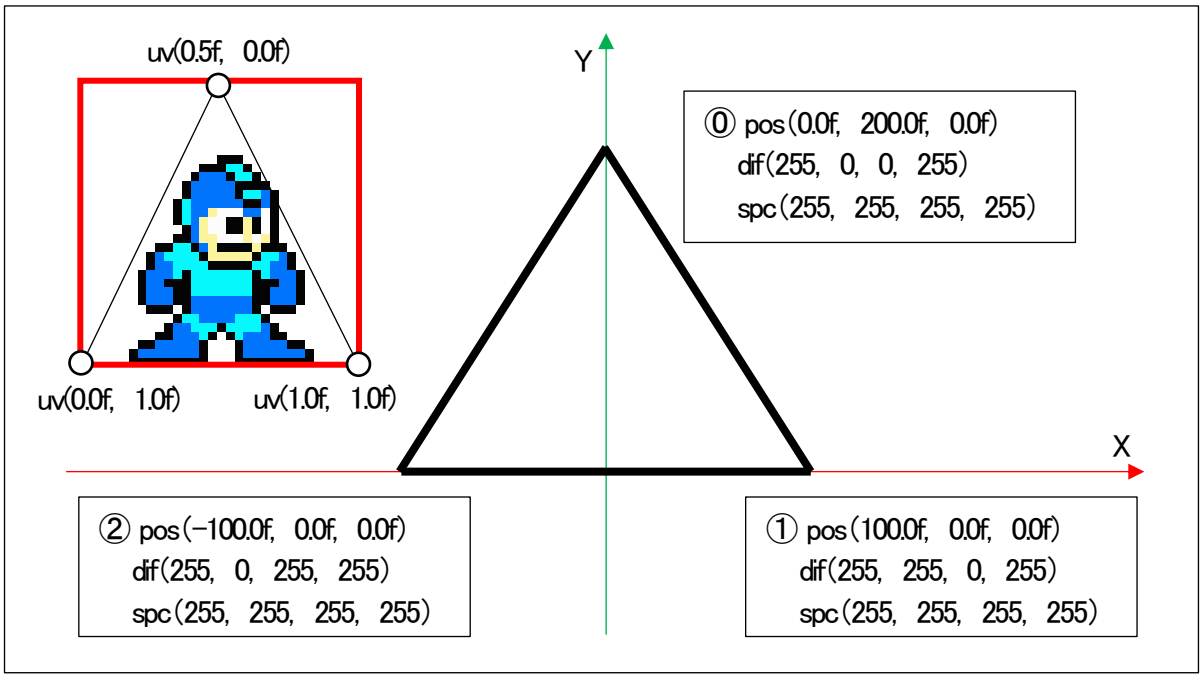
```
// Zバッファを有効にする
SetUseZBuffer3D(true);

// Zバッファへの書き込みを有効にする
SetWriteZBuffer3D(true);
```

■ポリゴンの裏表

ポリゴンの裏表の事をバックカリングと言い、バックカリングとは裏面になっているポリゴンを書かない処理のことです。
基本的に見えない部分になりますので、描画しない事でパフォーマンスの向上を図ります。

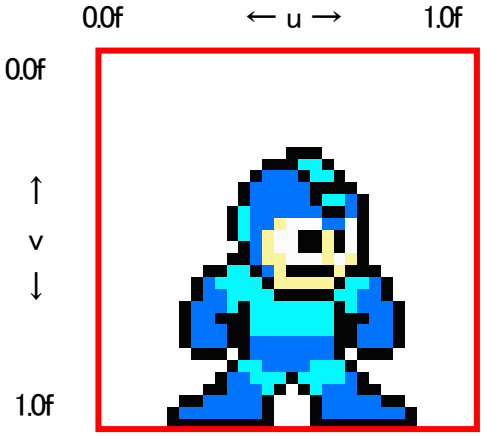
```
// バックカリングを有効にする
SetUseBackCulling(true);
```

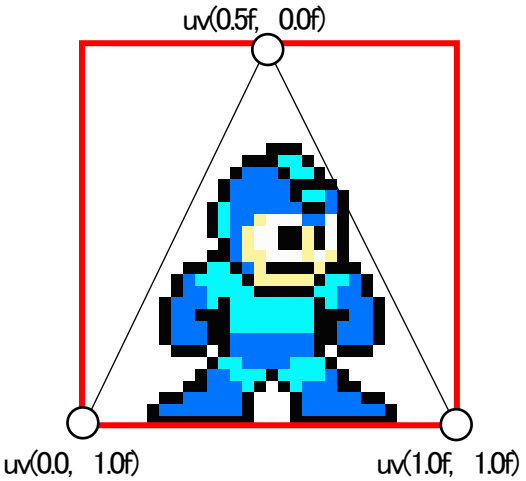
2）演習② 三角形のポリゴンにテクスチャーを貼ってみよう



■UVの基本形

テクスチャー画像の指定は、UVでポリゴンの頂点の位置を画像の場所を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。





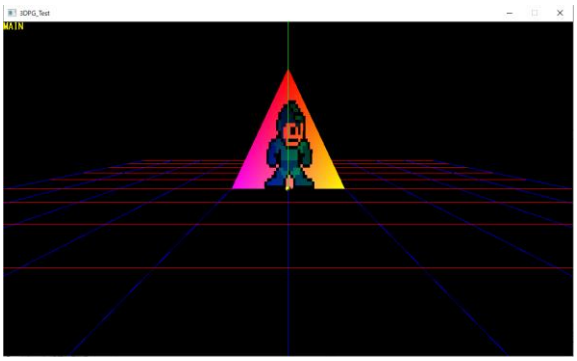
uv を使用してテクスチャを貼ってみる

```
void Primitive::Init()
{
    texture = LoadGraph("texture/sample.png");

    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[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[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[0].u = 0.5f;
    vertex[0].v = 0.0f;

    vertex[1].u = 1.0f;
    vertex[1].v = 1.0f;

    vertex[2].u = 0.0f;
    vertex[2].v = 1.0f;
}

void Primitive::Render()
{
    SetUseLighting(false);

    DrawPolygon3D(vertex, 1, texture, false);

    SetUseLighting(true);
}
```

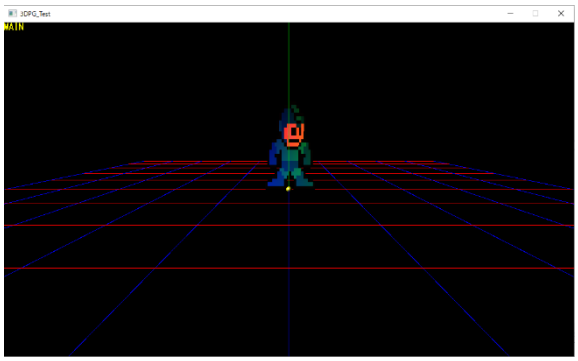
uv の設定

グラフィックハンドル

■テクスチャの透過処理

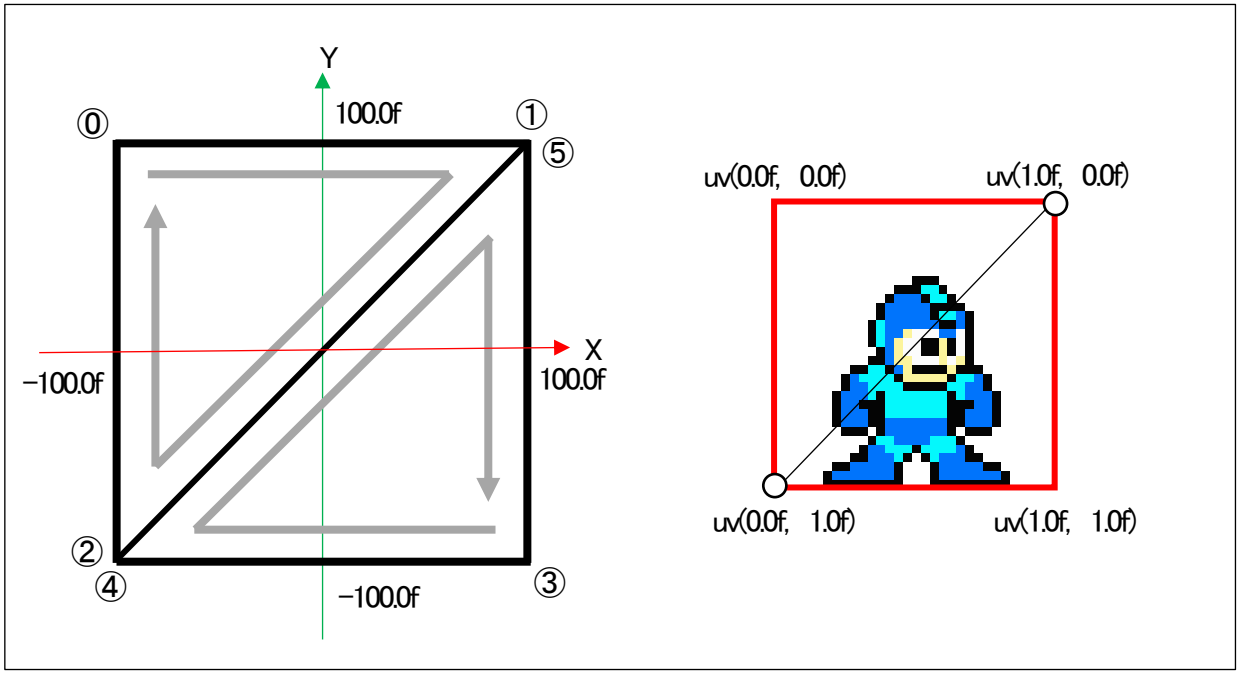


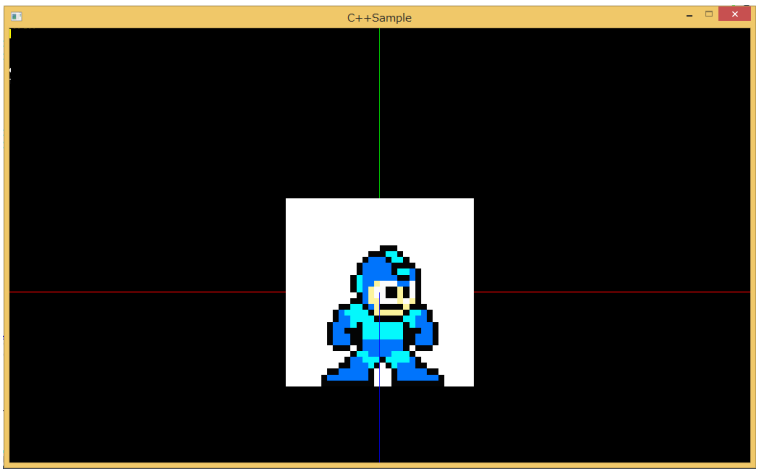
透過設定なし



透過設定あり

3）演習③ 図の様な四角形を描画する





プログラム例

```
void Primitive::Init()
{
    texture = LoadGraph("texture/sample.png");

    vertex[0].pos =
    vertex[0].dif =
    vertex[0].spc =
    vertex[0].u =
    vertex[0].v =

    vertex[1].pos =
    vertex[1].dif =
    vertex[1].spc =
    vertex[1].u =
    vertex[1].v =

    vertex[2].pos =
    vertex[2].dif =
    vertex[2].spc =
    vertex[2].u =
    vertex[2].v =

    vertex[3].pos =
    vertex[3].dif =
    vertex[3].spc =
    vertex[3].u =
    vertex[3].v =

    vertex[4].pos =
    vertex[4].dif =
    vertex[4].spc =
    vertex[4].u =
    vertex[4].v =

    vertex[5].pos =
    vertex[5].dif =
    vertex[5].spc =
    vertex[5].u =
    vertex[5].v =
}
```

0番の設定

1番の設定

2番の設定

3番の設定

4番の設定

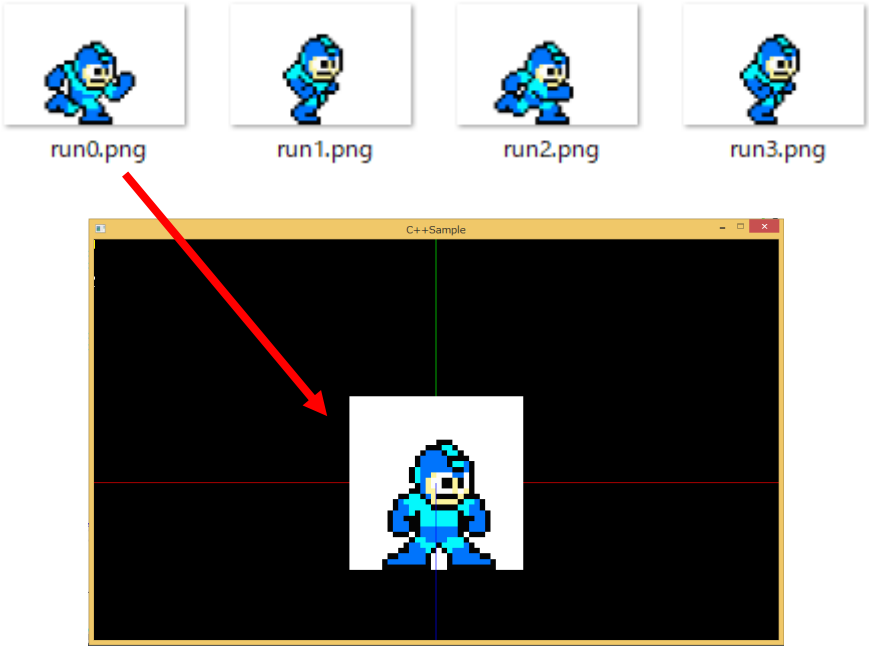
5番の設定

```
void Primitive::Render()
{
    SetUseLighting(false);

    DrawPolygon3D(vertex, 2, texture, true);

    SetUseLighting(true);
}
```

三角形2枚分なので「2」

5）演習⑤ 2D アニメーションの描画

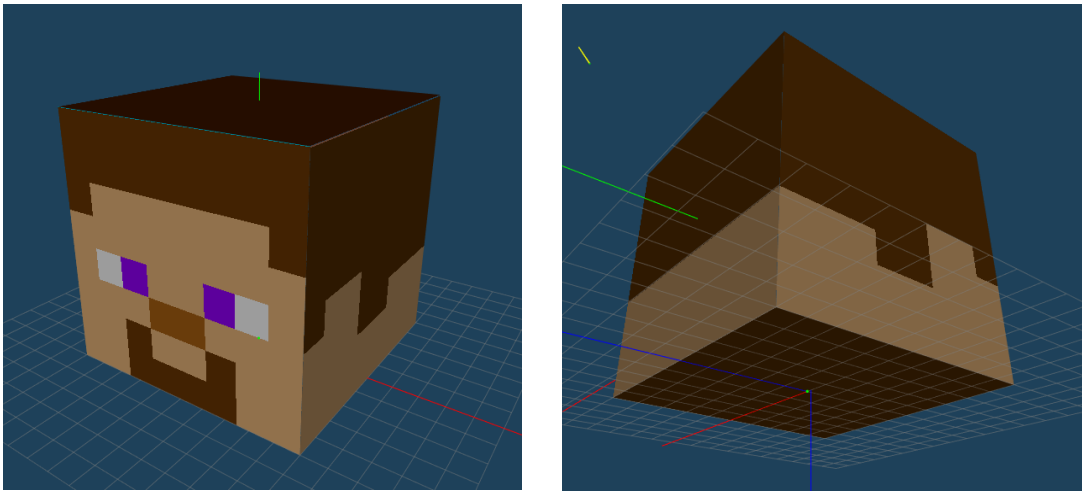
複数の2D 画像を1枚のポリゴンに入れ替えて描画する事で2D アニメーションを再現する事ができます。



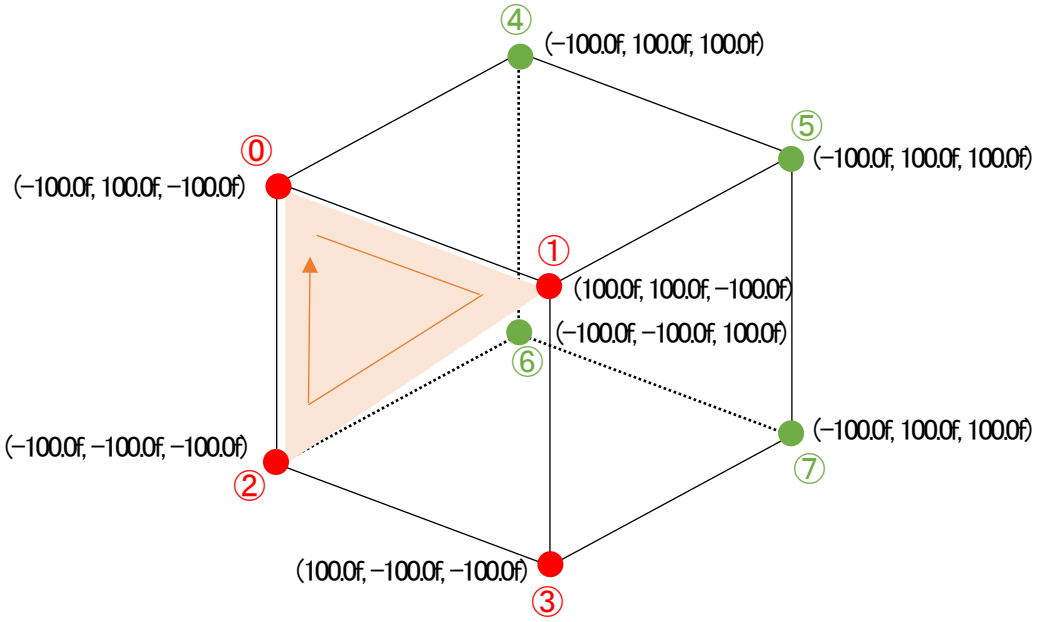
6）演習⑥ 6面体を描画する

1面の描画を応用して6面体の作成をしてみましょう。サンプルのテクスチャーはキャラクターのヘッドとサイコロを準備していますが、オリジナルで準備したものでも可能です。

※変数の数が多いので、座標毎に並べても良いですし、自作の関数を作成するなどの対応をしても良いです。



8つの頂点によって構成されるので、あらかじめ index として値を保持しておくいて利用する事も可能です。



ポリゴンの指定順番

正面・・・⓪→①→②、③→②→①
背面・・・⑤→④→⑦、⑥→⑦→④
左側面・・①→⑤→③、⑦→③→⑤
上面・・・①→⓪→⑤、④→⑤→⓪
右側面・・④→⓪→⑥、②→⑥→⓪
底面・・・②→③→⑥、⑦→⑥→③

UV の指定はポリゴンの向に注意して行います。