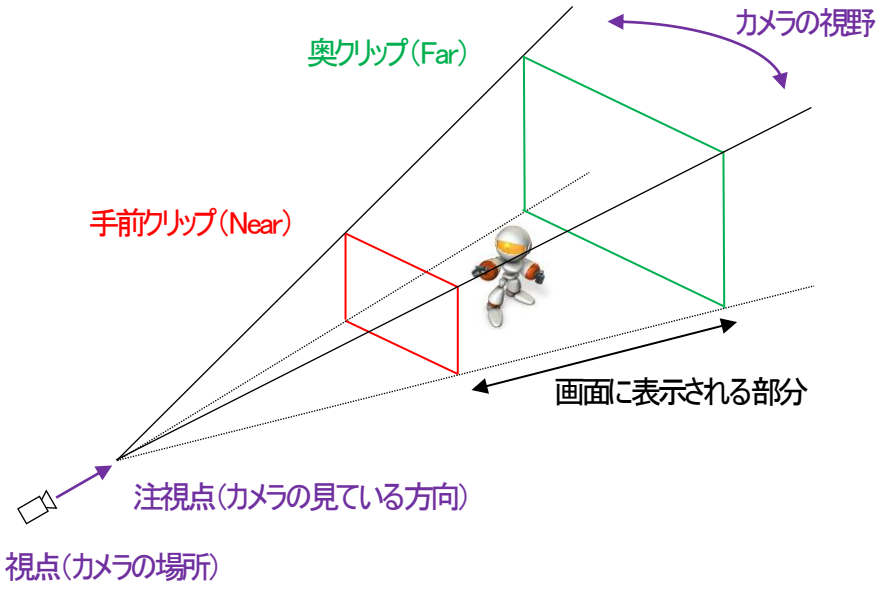
ゲームプログラミング応用Ⅰ／ゲームプログラミング実践Ⅰ

20 日で理解する3Dプログラミング 「その 2:カメラ」

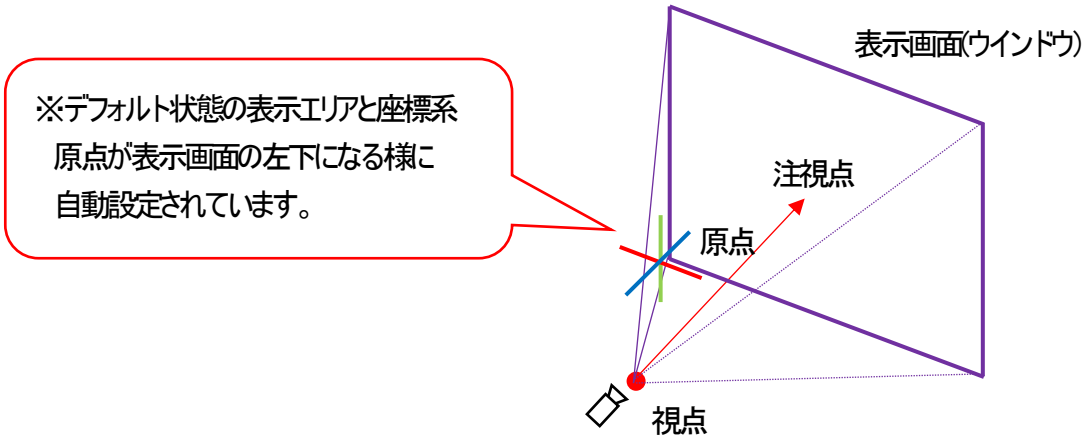
■カメラの設定

3DCG の画面は「カメラ」そのものと言えます。カメラの見ている部分が実際の画面に反映される様になっています。



【カメラ設定の基本形】

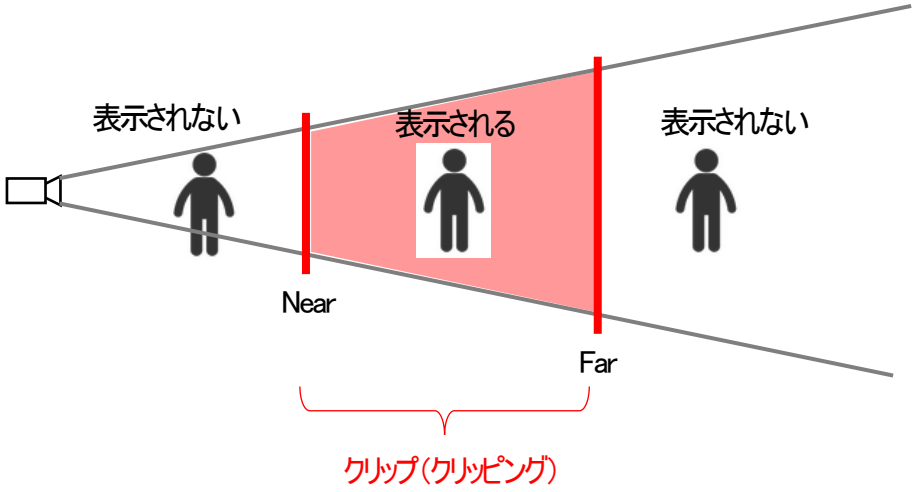
・「カメラの座標」・・・カメラの設置位置を(x, y, z)の三次元空間の座標で設定する。
・「カメラ上方向」・・・カメラの上方向を決定する。同じ場所でも傾けて撮影すると違いが出るイメージ。
・「見ている方向」・・・カメラが見る方向を、注視点として(x, y, z)の座標で設定する。
・「カメラの視野角」・・遠近法(遠いものほど小さく見える)を設定し、視野を広げたり狭くしたりする。
・「手前クリップ」・・撮影するエリアの「一番手前」を距離(座標)で設定する。
・「奥クリップ」・・・撮影するエリアの「一番奥」を距離(座標)で設定する。



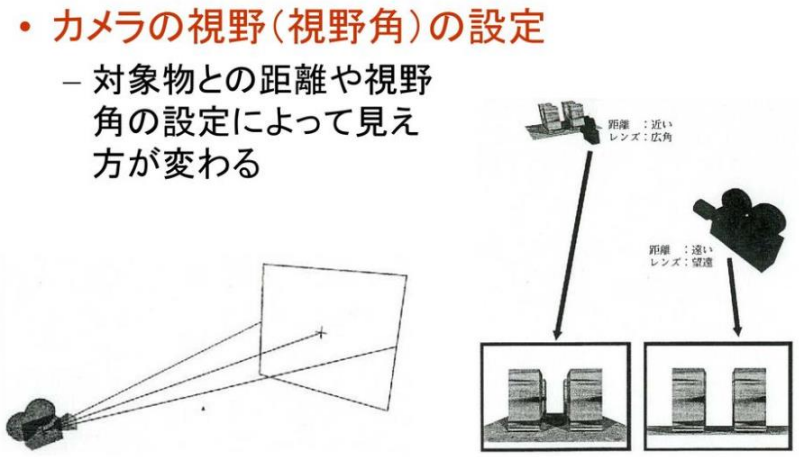
【カメラの「手前」「奥」の限界点の設定】

カメラに映る範囲（奥行）は、Near（ニア）とFar（ファー）の間のエリアとなります。このエリアの事を「視錐台」と言い、視錐台の外のものは描画されません。NearとFarで手前と奥の限界点を設定でき、エリアを広げると広大な世界を構築できますが、その分、処理が増加して描画などの負荷がかかりますので、程よい範囲設定が必要となります。

尚、NearとFarで切り取られたエリアを「クリップ」「クリッピング」と言います。



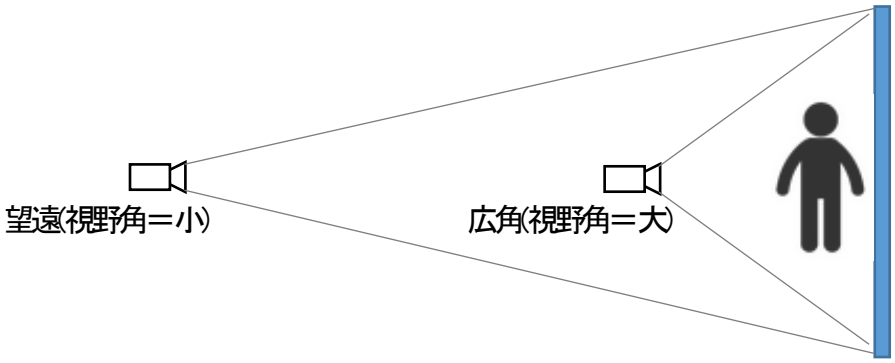
【視野角（遠近法） Field of vision = Fov】



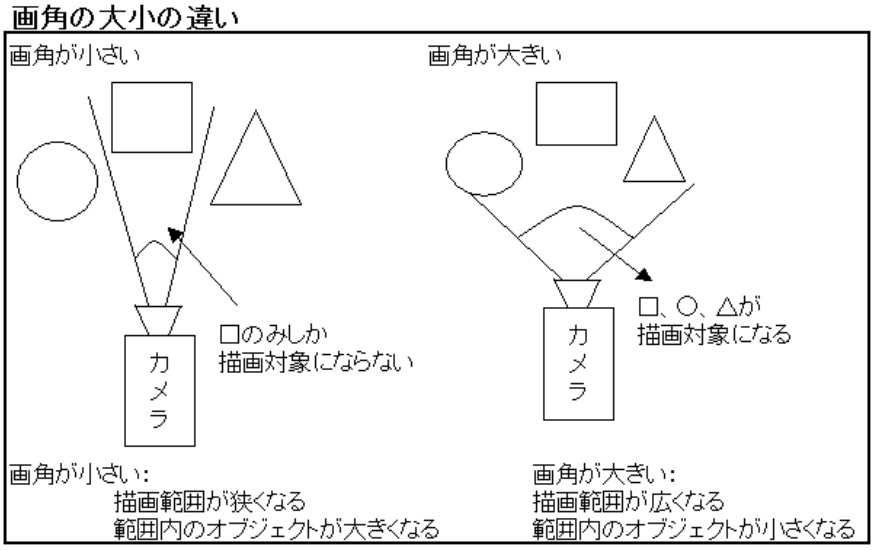
・広角 ・・・・ 実際のカメラでは焦点距離が短いとも言いますが、視野が広く多くの情報を収める事ができます。
立体感（パース）が強くかかる感じになります。

・望遠 ・・・・ 望遠と聞くと「拡大する」というイメージがありますが、視野が狭い為結果的に対象物が大きく見えます。
立体感（パース）が失われ平坦な画像になります。

画角の違うカメラで対象物を同じサイズ感で映す為には、対象物への距離を変える必要があります。

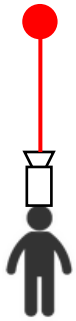


同じカメラ位置だと画角によって映し出す領域に差がでます。



【一人称(First Person)と三人称(Third Person)視点の考え方】

一人称視点の場合



注視点

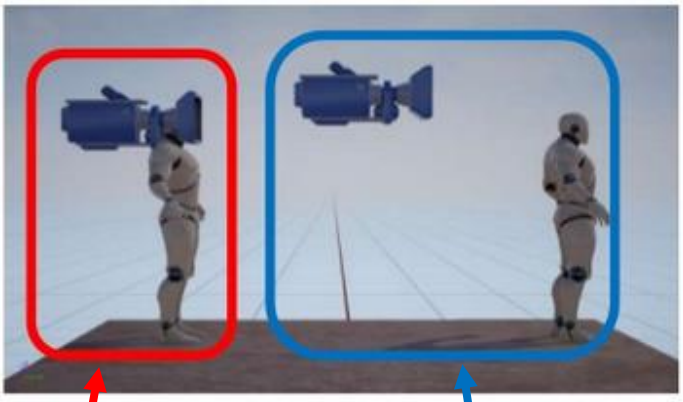
プレイヤー(の目の位置)
＝カメラ座標

三人称視点の場合

プレイヤー(の延長線)
＝注視点

カメラ座標

イメージ図



一人称視点のカメラ位置
(First Person)

三人称視点のカメラ位置
(Third Person)

■カメラの実装

カメラ制御用のカメラクラスを作成します。複数のカメラを構成する場合は管理用のクラスを作成します。

Camera.h

```
#pragma once

class Camera
{
private:
    VECTOR cameraPos;   // カメラ位置
    VECTOR targetPos;   // 注視点
    VECTOR cameraUpVec; // カメラの上方向

    float fov;      // 視野角
    float clipNear; // 手前クリップ
    float clipFar;  // 奥クリップ

public:
    Camera();
    ~Camera() {}
    void Init();
    void Update();
};
```

【カメラの視点、注視点、上方向を設定する関数】

int SetCameraPositionAndTargetAndUpVec(VECTOR Position, VECTOR Target, VECTOR Up) ;

カメラの視点、注視点、上方向を設定する

VECTOR Position ： カメラの位置
VECTOR Target ： カメラの注視点(見ている座標)
VECTOR Up ： カメラの上方向

カメラは移動したり向きを変えたりする場合は、座標や注視点が変化するので Update()に書きます。
今回は基本設定値としてひとまず Init()で定義しますが、カメラセットの関数は Update()に書いておきます。

Camera.cpp

```
#include "Dxlib.h"
#include <math.h>
#include "common.h"
#include "Camera.h"

Camera::Camera()
{
    Init();
}

void Camera::Init()
{
    // ----- カメラの基本設定
    cameraPos = VGet(0.0f, 100.0f, -500.0f);
    targetPos = VGet(0.0f, 0.0f, 0.0f);
    cameraUpVec = VGet(0.0f, 1.0f, 0.0f);
}

void Camera::Update()
{
    // カメラセット
    SetCameraPositionAndTargetAndUpVec(cameraPos, targetPos, cameraUpVec);
}
```

GameTask.cpp

```
GameTask::GameTask()
{
    SystemInit();    // 最初の初期化

    camera = new Camera();  // カメラ生成
}

GameTask::~GameTask()
{
    delete camera;
}
```

```
void GameTask::GameMainLoop()
{
    camera->Update();

    // ---------- グリッド線

    // ---------- XYZ基本軸


    DrawString(0, 0, "MAIN", 0xffff00);
}
```

グリッド線などが見にくい場合は背景色を設定しておくと見やすくなって作業が容易になります。
一度設定するだけで良いので GameTask のコンストラクタ時などに書いておくと良いです。

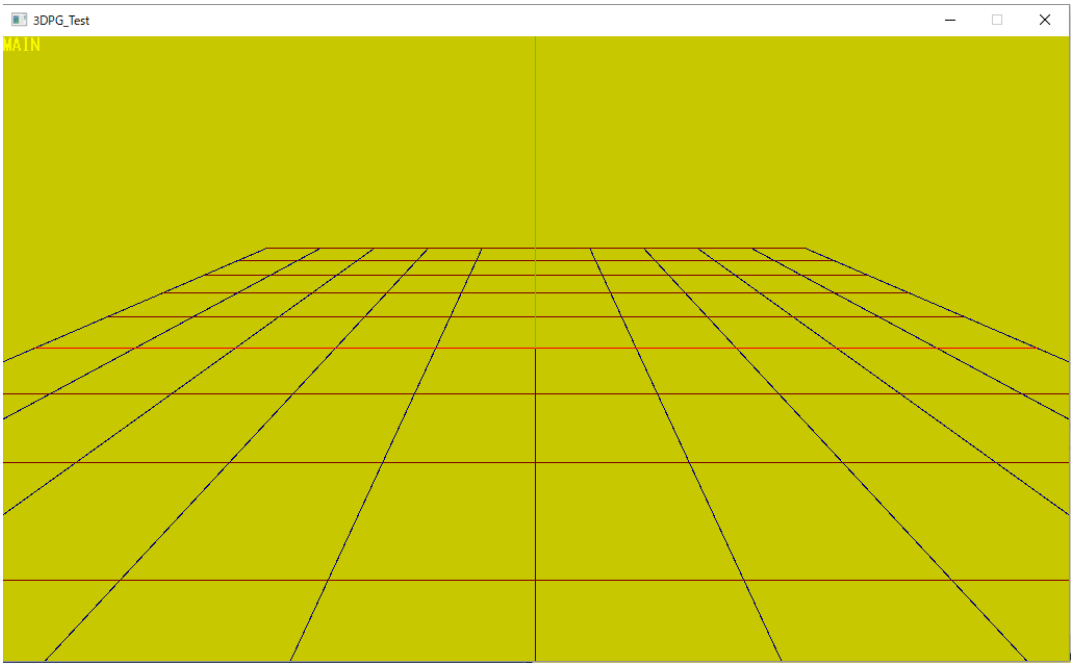
【背景色を設定する関数】

int SetBackgroundColor(int Red, int Green, int Blue) ;

画面の背景色を設定する

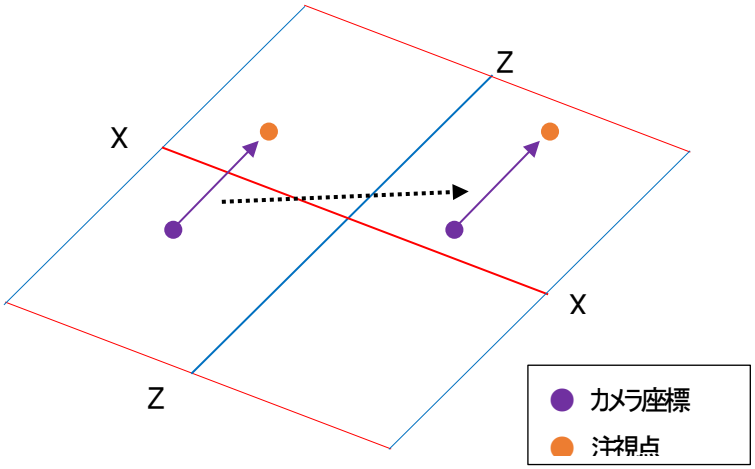
Red ： 背景色の赤成分(０～２５５)
Green ： 背景色の緑成分(０～２５５)
Blue ： 背景色の青成分(０～２５５)

■カメラ基本完成



□カメラを XZ 軸方向に前後左右に動かす

カメラを動かす場合は「カメラ座標」と「注視点」を移動させます。カメラの上方向を Y 座標とすると、前後左右の移動は XZ 平面で行う事になります。上下の動きを考えなければ2D の移動と同様となります。



1)【演習】キー入力によりカメラを XZ 平面上に前後左右に移動させる

例)WSAD キーで「カメラ位置」「注視点」を前後左右に動かす
if(CheckHitKey(KEY_INPUT_W))
{
　カメラ位置を移動(XZ 平面)
　注視点を移動(XZ 平面)
}

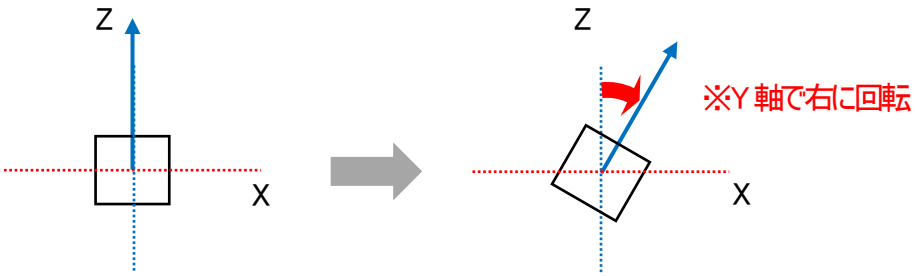
「W」「S」で前後に移動する例

Camera.cpp

```
void Camera::Update()
{
    // カメラの前後左右の移動
    if (CheckHitKey(KEY_INPUT_W))
    {
        cameraPos.z += 5.0f;
        targetPos.z += 5.0f;
    }
    if (CheckHitKey(KEY_INPUT_S))
    {
        cameraPos.z -= 5.0f;
        targetPos.z -= 5.0f;
    }
}
```

カメラの左右移動を追加して、前後左右の4方向に動かせたら OK です。

□カメラを左右に回転させる

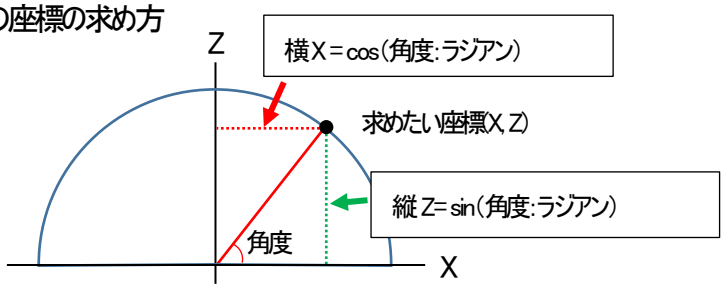


1)【演習】キー入力によりカメラをＹ軸回転させる。

カメラを回転させるために必要なもの
・Ｙ軸の回転角度(360 度) ※アナログ時計を上から見て針の動きのイメージ

・カメラの位置・・・・・前後左右の時と同等
・注視点・・・・・・・・・・Ｙ軸の回転角度を基に向いている先の任意の点を求める

回転後の座標の求め方



「←」「→」でＹ軸を基準に左右に回転する例

Camera.cpp

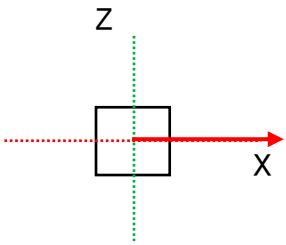
```
//  カメラの回転(カメラ座標を中心)
if (CheckHitKey(KEY_INPUT_RIGHT))
{
    camRot.y += (DX_PI_F/180)*0.5f;
}
if (CheckHitKey(KEY_INPUT_LEFT))
{
    camRot.y -= (DX_PI_F/180)*0.5f;
}
```

注視点に黄色の球を配置して場所の確認をする

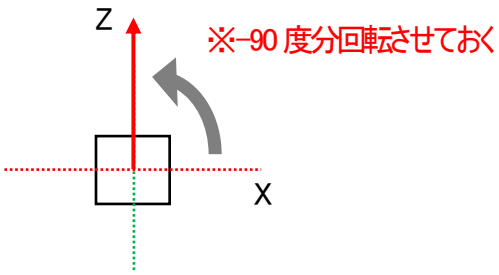
```
DrawSphere3D(targetPos, 5.0f, 8, 0xffff00, 0xffffff, 1);
```

ヒント）Z軸を真っすぐ見た場合、デフォルトの角度０は時計でいう３時の方向になっているので、０度の時に
真正面（Ｚの＋方向）を向くように補正が必要です。

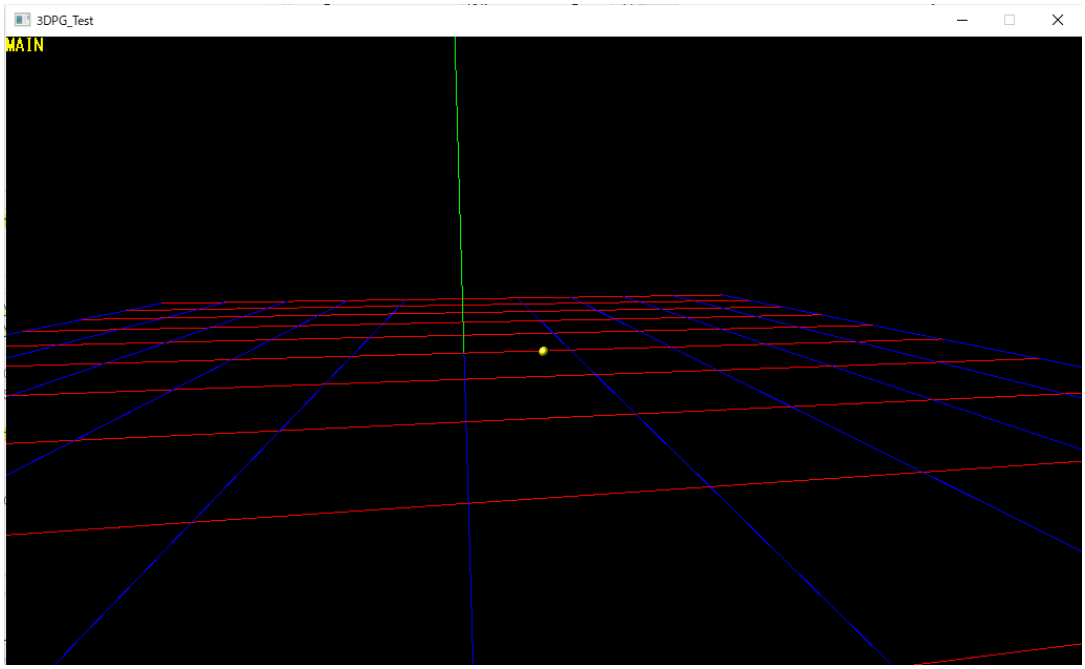
■ 初期値(0 度)のＹ軸の向き



■ ０度の時に上に向く様に補正



カメラを左右に回転させられたら OK です。



カメラの XZ 平面の前後左右の移動とＹ軸基準の左右回転のプログラム例。

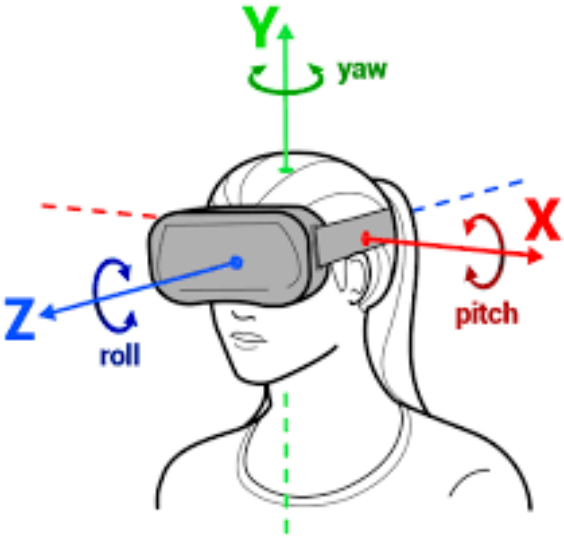
```
void Camera::Update()
{
    // カメラの前後左右の移動
    if (CheckHitKey(KEY_INPUT_W))
    {
        cameraPos.z += 5.0f;
        targetPos.z += 5.0f;
    }
    if (CheckHitKey(KEY_INPUT_S))
    {
        cameraPos.z -= 5.0f;
        targetPos.z -= 5.0f;
    }
    if (CheckHitKey(KEY_INPUT_A))
    {
        cameraPos.x -= 5.0f;
        targetPos.x -= 5.0f;
    }
    if (CheckHitKey(KEY_INPUT_D))
    {
        cameraPos.x += 5.0f;
        targetPos.x += 5.0f;
    }

    //  カメラの回転(カメラ座標を中心)
    if (CheckHitKey(KEY_INPUT_RIGHT))
    {
        camRol.y += (DX_PI_F / 180) * 0.5f;
    }
    if (CheckHitKey(KEY_INPUT_LEFT))
    {
        camRol.y -= (DX_PI_F / 180) * 0.5f;
    }

    targetPos.x = cameraPos.x + cosf(camRol.y) * camLength;
    targetPos.z = cameraPos.z - sinf(camRol.y) * camLength;

    // カメラセット
    SetCameraPositionAndTargetAndUpVec(cameraPos, targetPos, cameraUpVec);

    // 注視点に黄色の球を配置
    DrawSphere3D(targetPos, 5.0f, 8, 0xffff00, 0xffffff, 1);
}
```

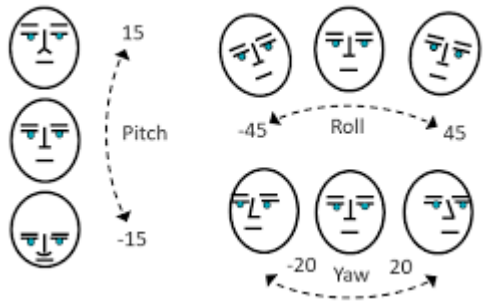
□カメラの「X 軸回転」「Y軸回転」「Z軸回転」で設定する場合

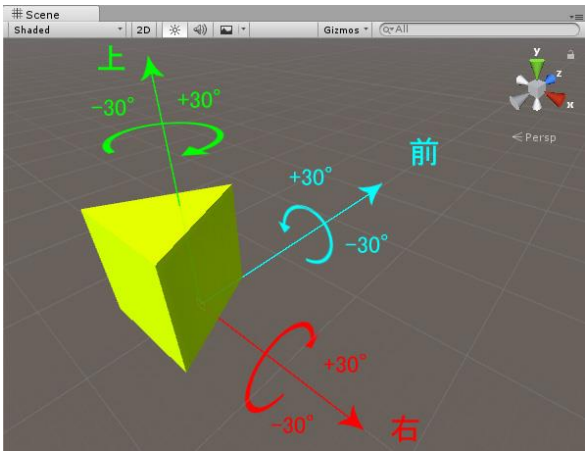
カメラの方向は、「カメラ座標」と「注視点」と「上方向」の情報があれば制御できますが、XYZ 軸のそれぞれの軸でカメラを回転する事で方向を変える事もできます。

飛行機などの姿勢制御で Yaw（ヨー）、Pitch（ピッチ）、Roll（ロール）を使用しますが、正確には回転する方向の事で Y, X, Z 軸で回転するという意味ではありません。

しかしながら考え方として分かりやすいので Yaw（ヨー）、Pitch（ピッチ）、Roll（ロール）でイメージしても良いです。








```
カメラの視点、垂直回転角度、水平回転角度、ひねり回転角度を設定する
int SetCameraPositionAndAngle(カメラ位置、pitch 回転、yaw 回転、roll 回転);

VECTOR camPos; //  カメラの位置
VECTOR camRol; //  カメラの回転
                    .x; // pitch 回転(X 軸回転)のラジアン値
                    .y; // yaw 回転(Y 軸回転)のラジアン値
                    .z; // roll 回転(Z 軸回転)のラジアン値
```

「3軸での回転」と「視点・注視点での回転」では、目的は同じでも座標やベクトルなどの管理方法が異なります。
特に、3軸回転はあくまでも XYZ 軸での回転となる為、任意軸での回転の場合には対応していません。
最初の段階は、ゲームを想定した「視点：プレイヤー」「注視点：ターゲット（敵）」の方が考えやすいと思います。

■総仕上げ！FPS カメラ（一人称視点カメラ）の実装

カメラ移動の集大成として FPS の自分視点カメラを作成します。
「←」「→」キーで左右に回転、「W」「A」「S」「D」キーで向いている方向に前後左右移動するカメラを設定しましょう。

1）カメラの仕様の確認

必要な変数

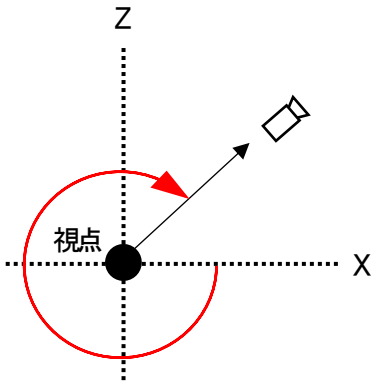
```
VECTOR  camPos;          // 視点（camPos.x, camPos.z）
VECTOR  targetPos;       // 注視点(target.x, target.z)
VECTOR  cameraUpVec;     // カメラの上方向

VECTOR  camRol;          // Y 軸の回転角 = camRol.y(ラジアン)
float   camLength;       // 注視点までの距離(単位ベクトル可)
```

使用する関数

SetCameraPositionAndTargetAndUpVec(カメラ位置, 注視点, 上方向)を使用する

2）計算方法

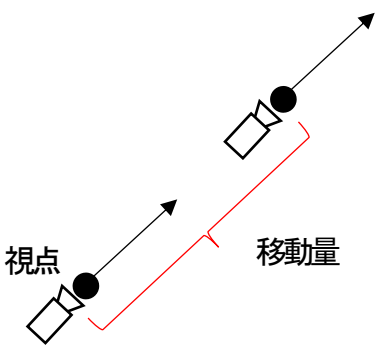
①視点を回転させる。Y 軸を 3 時の向きから時計回りに回転した角度をラジアンで管理する



camRol.y = Y 軸の回転後の角度

右方向 ： camRol.y += 回転量
左方向 ： camRol.y −= 回転量

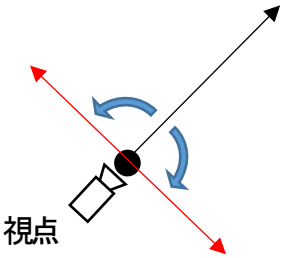
②向いている方法に移動させる（前後移動）。



移動後 X = camPos.x + cosf（回転値）＊ 移動量
移動後 Z = camPos.z − sinf（回転値）＊ 移動量

※回転後の角度の方向に X、Z 軸での増減分で移動する。
2D ゲームの斜め移動と原理は同じです。
※回転角度は時計回りに＋になりますが、sin 値に対応している Z 軸は±が逆になっている為、回転後の座標を計算する場合は Z の値を−にする必要があります。

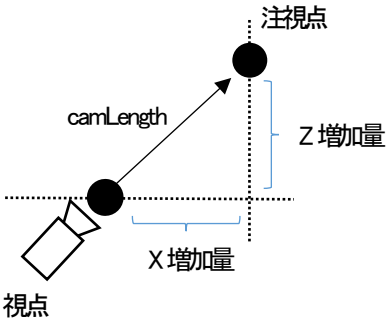
③向いている方法に移動させる（左右移動）。



左右移動の場合は、向いている方向から 90 度回転した方向になります。

移動後 X = camPos.x + cosf（回転値±90 度）* 移動量
移動後 Z = camPos.z − sinf（回転値±90 度）* 移動量
※右移動の場合は+90 度
※左移動の場合は−90 度

④視点を基準に向いている方法の注視点を計算する。



注視点（target）の座標を求める

注視点 X = 視点 X + X 増加量
注視点 Z = 視点 Z − Z 増加量

※回転角度は時計回りに＋になりますが、sin 値に対応している Z 軸は±が逆になっている為、回転後の座標を計算する場合は Z の値をーにする必要があります。

一人称視点で、前後左右移動＋左右の視点移動ができれば OK です。
下記の様なゲームを作る基本ができました。

更に上下にカメラの視点を変えるなどの処理がありますが、回転行列の演習の後に取り組んでいきます。

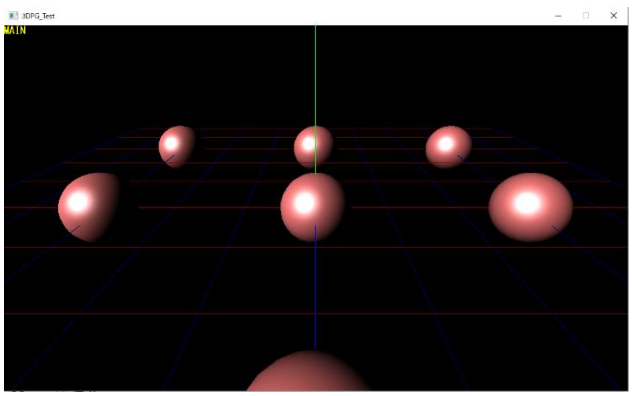
■視野角の無いカメラ(正射影)の設定　Orthogonal Projection

正射影は、オブジェクトを平面的に見るもので、遠近感がないので遠くのものでも近くのもので同じ大きさに見えます。
CAD などの設計において、図形を正確に表示する必要がある場合などに使用されます。

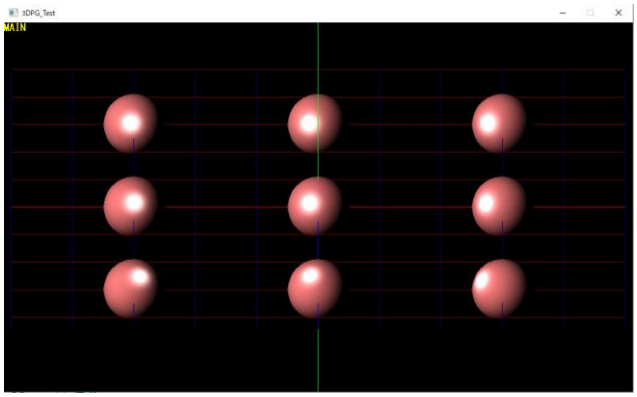
正射影カメラを設定する（遠いものでも近いものでも同じ大きさに見える）
int SetupCamera_Ortho(縦サイズ);

縦サイズ・・・ float Size;　※画面垂直方向の表示範囲

```
void Camera::Init()
{
    float ySize = 600.0f;
    SetupCamera_Ortho(ySize);   // 縦ｻｲｽﾞ 600 で正投影に描画する。
}
```



Ortho オフ



Ortho オン

こんな感じのパースのかかっていない 3D 表現のゲームで活用できますね。