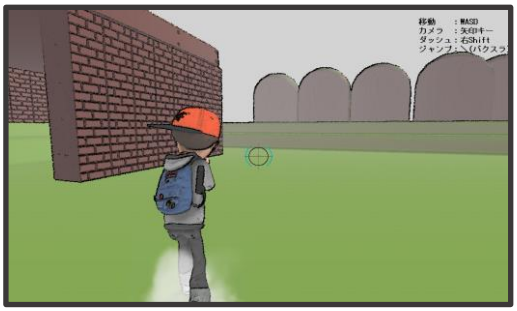
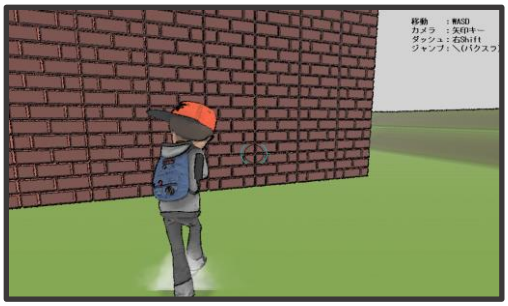
カメラの設定

カメラは３Ｄワールドのどの場所を映し出すかを決めるための
最も重要な設定になります。

キャラクターが動いたり、向きを変えたりすると、
画面もそれに応じて、写す内容を変えるゲームが多いかと思います。





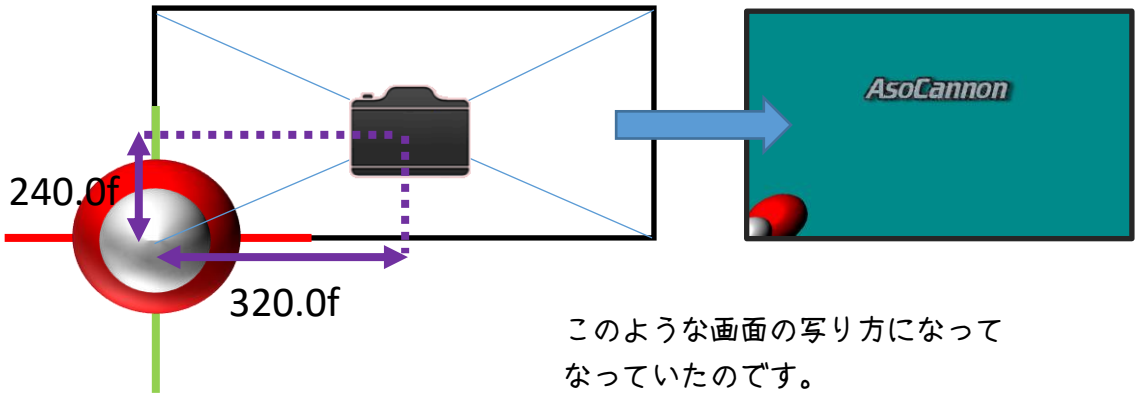
右の画面は、前進して、ちょっと左を向いた後の画面。
これは、キャラクターが移動していると同時にカメラも移動したり、
回転して向きを変えているため。

頻繁にカメラの位置を移動させたり、角度を変えたり、
しっかり制御する必要がありますので、カメラクラスを作成していきます。

ちなみに、
球体が { 0.0f, 0.0f, 0.0f } に設定されているのに
左下に表示されている理由としては、DxLibでは、カメラの初期位置が

x = 320.0f, y = 240.0f, z = (画面のサイズによって変化)

向きが真正面となっておりますので、



このような画面の写り方になって
なっていたのです。

それではカメラクラスを作成して、カメラ位置を設定していきましょう。

DxLibでカメラの位置を設定する方法はいくつかありますが、
今回は、最も直感的な下記の関数を使用していきます。

```
int SetCameraPositionAndAngle(
    VECTOR Position, float VRotate, float HRotate, float TRotate);
```

VECTOR Position ： カメラの位置
float VRotate ： 垂直回転角度(単位：ラジアン)
float HRotate ： 水平回転角度(単位：ラジアン)
float TRotate ： 捻り回転角度(単位：ラジアン)

必要な要素は、カメラの位置(x, y, z)、カメラの角度(x, y, z)になりますので、
逆算して、VECTOR構造体を２つ、メンバ変数に定義していきます。

Camera.h

```
#pragma once
#include <DxLib.h>

class Camera
{

public:

    // コンストラクタ
    Camera(void);

    // デストラクタ
    ~Camera();

    // 初期処理(基本的に最初の１回だけ実装)
    void Init(void);

    // 更新処理(毎フレーム実行)
    void Update(void);

    // カメラ設定(毎フレーム実行)
    void SetBeforeDraw(void);
```

```
    // 描画処理(毎フレーム実行)
    void Draw(void);

    // 解放処理(基本的に最後の１回だけ実装)
    void Release(void);

private:

    // カメラの位置
    VECTOR pos_;

    // カメラの角度
    VECTOR angles_;

};
```

３Ｄゲームにおいて、ほぼ全てのシーンでカメラを使う形になりますので、
Sceneクラス１つ１つにカメラ機能を付けるのではなく、
全てのSceneを統括する、SceneManagerにカメラ機能で実体を作っていきます。

SceneManager.h

```
    ～ 省略 ～

private:

    ～ 省略 ～

    // カメラ
    Camera* camera_;            ※前方宣言を忘れずに
```

宣言が終わりましたら、SceneManager.cppの方に組み込んでください。
インスタンスの生成、Init、Update、Draw、Release。

今回新しく登場した概念なのですが、

```
// カメラ設定(毎フレーム実行)
void SetBeforeDraw(void);
```

Drawの前に、必ずカメラ設定を行いましょう、という意味で、
上記関数を追加しています。
この関数の中で、カメラの位置や角度を設定していきます。

なぜかといいますと、
ダブルバッファリングを行うために、背面スクリーンに描画対象領域を
設定しているかと思いますが、

```
// 描画先グラフィック領域の指定
//（３Ｄ描画で使用するカメラの設定などがリセットされる）
SetDrawScreen(DX_SCREEN_BACK);
```

この関数、『 カメラの位置や角度がリセットされます 』。

せっかくカメラの設定をプログラムしても、設定する場所によっては、
効果がかき消されてしまうため、なかなか３Ｄモデルが画面に映らない、
原因がわからないトラブルになりやすいです。

そういった事故をできるだけ無くすために、

SceneManager.cpp

```
void SceneManager::Draw(void)
{
    // 描画先グラフィック領域の指定
    //（３Ｄ描画で使用するカメラの設定などがリセットされる）
    SetDrawScreen(DX_SCREEN_BACK);

    // 画面を初期化
    ClearDrawScreen();

    // カメラ設定
    camera_->SetBeforeDraw();

    // 描画
    scene_->Draw();

    // カメラデバッグ等
    camera_->Draw();

    // 暗転・明転
    fader_->Draw();
}
```

この処理順番を崩さないようにしていきたいと思います。

それではいよいよ、カメラクラスにカメラ設定を実装していきます。

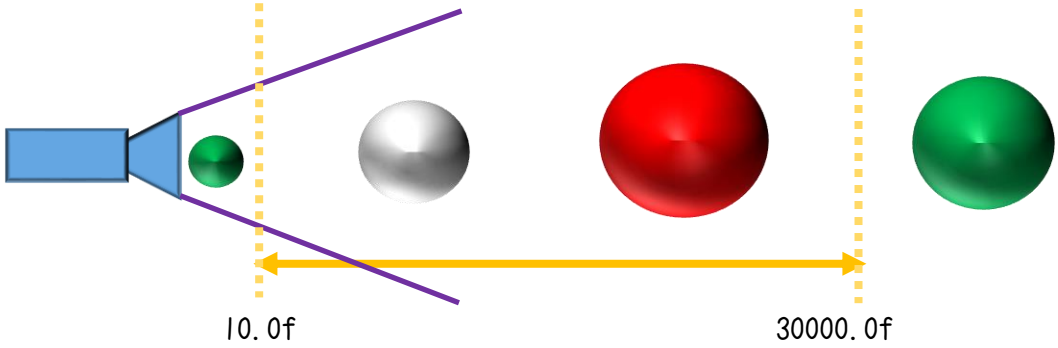
Camera.cpp

```
void Camera::Init(void)
{
    // カメラの位置
    pos_ = { 0.0f, 500.0f, -500.0f };

    // カメラの角度
    angles_ = { 40.0f * DX_PI_F / 180.0f, 0.0f, 0.0f };
}
```

```
void Camera::SetBeforeDraw(void)
{

    // クリップ距離を設定する(SetDrawScreenでリセットされる)
    SetCameraNearFar(10.0f, 30000.0f);

    // カメラの設定(位置と角度による制御)
    SetCameraPositionAndAngle(
        pos_,
        angles_.x,
        angles_.y,
        angles_.z
    );

}
```

ここで急に出てきたクリップ距離ですが、
カメラで写す奥行きが無限だと、処理も無限になりますので、
キリがありません。そこで、映し出す奥行きの最小と最大を設定します。

```
    // クリップ距離を設定する(SetDrawScreenでリセットされる)
    SetCameraNearFar(10.0f, 30000.0f);
```



上図の例ですと、緑の球体は写りません。
無限という言葉を使ってしまいましたので、Farを制限すれば良い、と
思われがちですが、距離が近ければ近いほど、
精密な計算を行う必要があります。(近くでハッキリ大きく見えてしまうため)
ですので、意外にNearの設定も大切になってきますのでご注意ください。
許容できる範囲で、いくらかは設定するようにしてください。(0.0fにはしない)

このカメラ設定もリセットされてしまいますので、このタイミグで設定します。

あと、角度の設定についても、しばらく皆さんを悩ませる種になるかと思います。

```
// カメラの角度
angles_ = { 40.0f * DX_PI_F / 180.0f, 0.0f, 0.0f };
```

角度の単位は、度数(デグリー)ではなく、弧度(ラジアン)である必要があります。
Unityなどのゲームエンジンは別ですが、プログラム上で角度を使用する場合、
ラジアンの方が都合が良いですので、C++を扱う際には、角度 = ラジアンと
覚えて貰って良いかと思います。

度数(DEG)	0	45	90	135	180	225	270	315	360
弧度(RAD)	0.00	0.79	1.57	2.36	3.14	3.93	4.71	5.50	6.28

度数は人間的にわかりやすいですが、弧度はなかなかどのくらいの角度なのか、
イメージするのが難しいかもしれません。
そんな場合は、度数から弧度に変換するようにしましょう。

【デグリー／ラジアンの変換式】

度数(デグリー)から弧度(ラジアン)へ変換

度数 × π ÷ １８０度 = 弧度

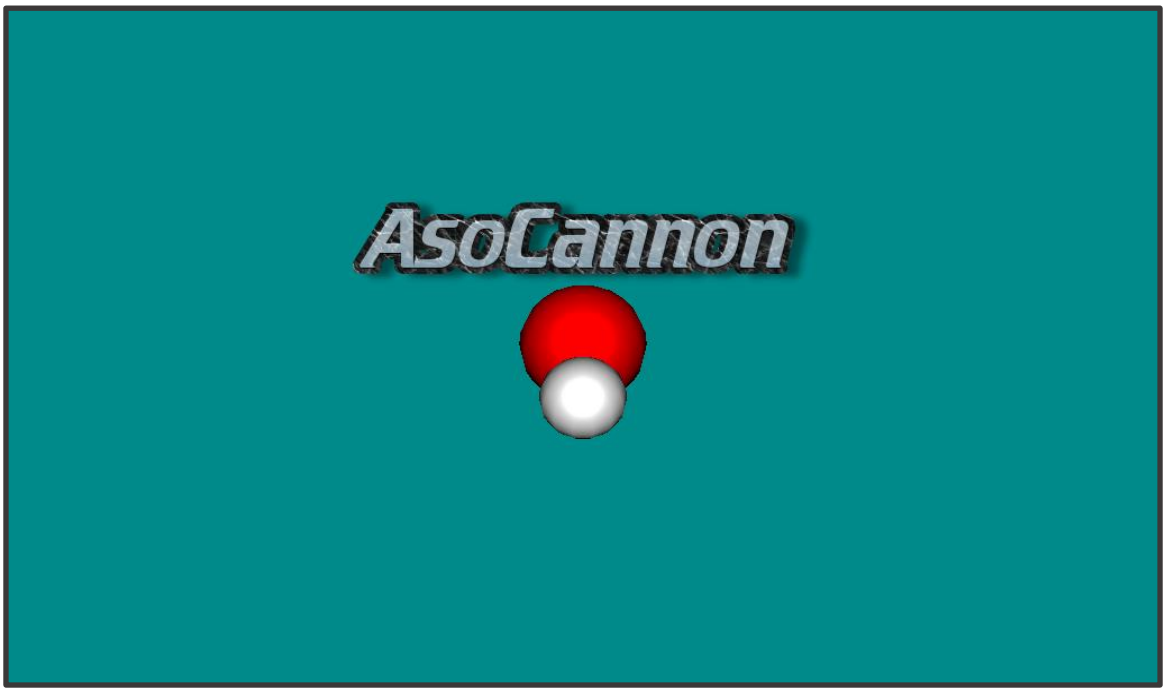
40度の場合だと、
40.0f * DX_PI_F / 180.0f = 0.698...f

弧度(ラジアン)から度数(デグリー)へ変換

弧度 × １８０度 ÷ π = 度数

0.698...ラジアンの場合だと、
0.698...f * 180.0f / DX_PI_F = 40度

※DX_PI_Fは、DxLibで定義されている円周率(3.141592...)の定数です



やっとそれっぽく描画されたのではないでしょうか。
カメラと球体の位置関係は、現在、このようになっています。