3D アクションゲームプログラミング

○評価要件

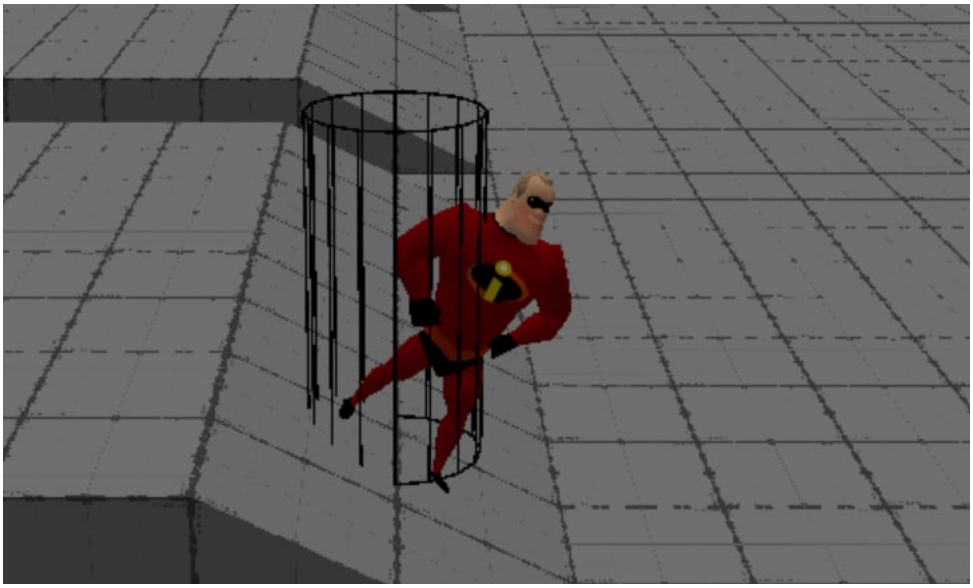
☑キャラクターを地面の向きに沿うように回転させる
☑滑らかに回転するように補完処理をする。

○概要

今回はレイキャストを利用して地面の法線ベクトルを算出し、キャラクターの姿勢を地面に沿うように回転させます。

以前、弾丸処理の実装の際にベクトルから行列を作成する方法で学習しました。
その方法でも実現できるのですが、キャラクターの回転制御は XYZ 軸のオイラー角で制御しています。

今回は法線ベクトルから X 軸、Z 軸のオイラー角を算出し、姿勢制御を行います。
また、算出した角度を単純に設定すると地面の傾斜が変わったときに急に回転してしまうため不自然なので、滑らかに回転するように補完処理を実装しましょう。

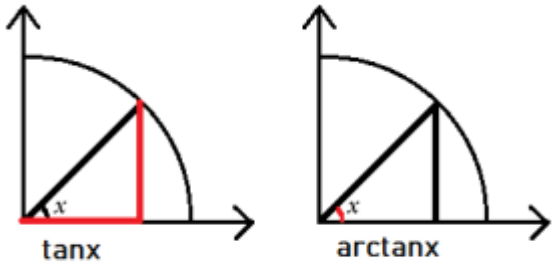


○ベクトルから角度を算出する

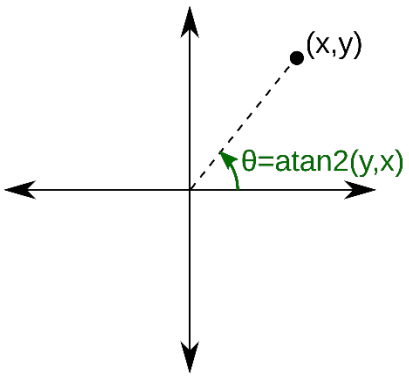
今まで、回転制御は内積と外積を使って算出してきました。
回転角を求める方法はアークタンジェント（atan2）関数を使用することでも算出できます。

タンジェント（tan）とはサイン（sin）、コサイン（cos）と同じ三角関数の仲間の一つです。
三角関数は直角三角形から角度や辺の長さなどを算出することができます。

タンジェントとは直角三角形において、底辺と対辺の２辺の比を表しており、
アークタンジェントはこの２辺の比に対しての角度を表しています。



そして、プログラムでは atan2(y,x)という関数があり、2D ベクトルから角度を算出できます。



この関数を利用して回転角度を算出します。
今はキャラクターの Y 軸の回転を内積と外積を利用して回転させていましたが、XZ 平面で指定方向に回転させるには下記の方法でも回転させることができます。

```
angle.y = atan2f(vec.x, vec.z);
```

Y 軸回転は既に出来ているので、上記の計算をしなくても大丈夫です。
今回、Y 軸回転制御は今のままで、アークタンジェントを使って X 軸と Z 軸の回転角を算出して地面に沿うような姿勢に回転させましょう。

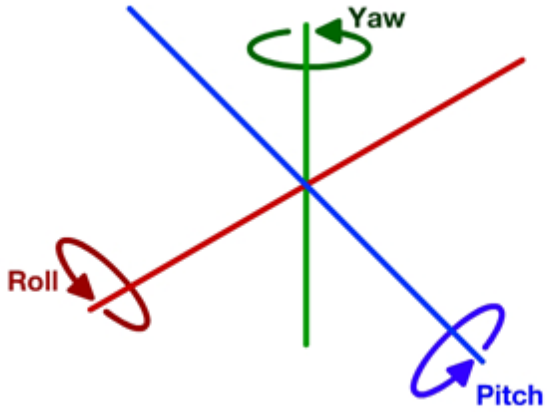
○行列の回転順について

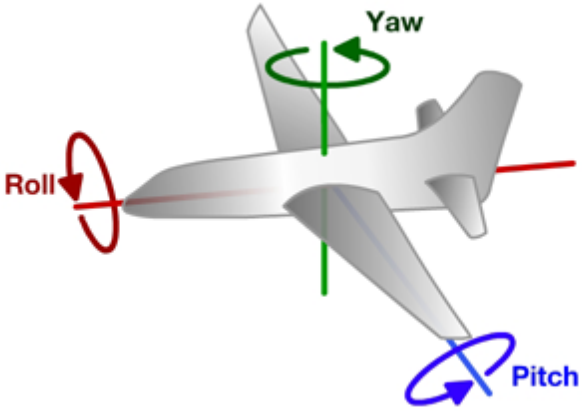
現在、キャラクターは XYZ 軸のオイラー角で回転制御していますが、この値は下記の関数を使用して回転行列に変換しています。

```
DirectX::XMMATRIX R = DirectX::XMMatrixRotationRollPitchYaw(angle.x, angle.y, angle.z);
```

この関数の Roll（ロール）、Pitch（ピッチ）、Yaw（ヨー）には以下の意味があり、回転順を表しています。

Roll（ロール）	Z 軸回転
Pitch（ピッチ）	X 軸回転
Yaw（ヨー）	Y 軸回転





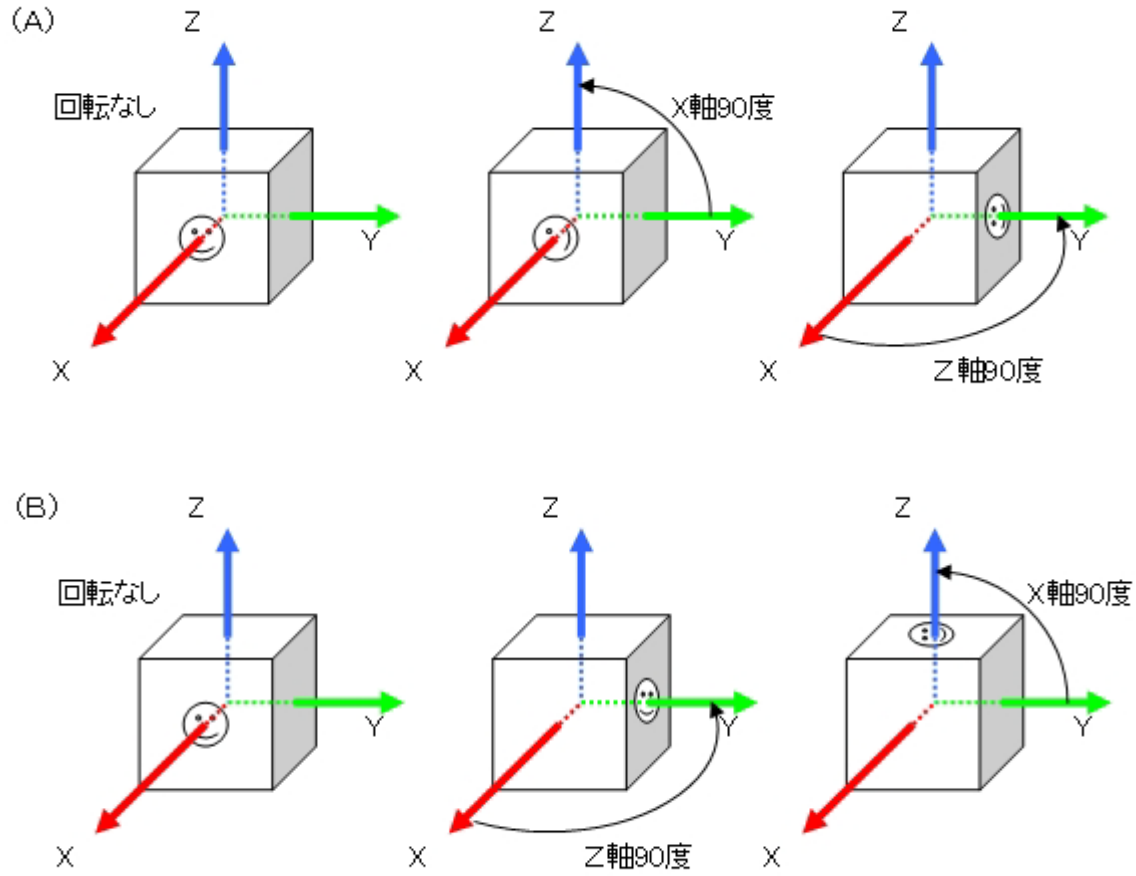
XMMatrixRotationRollPitchYaw()の名前から ZXY 回転ということがわかります。
これをプログラムにすると以下のようになります。

```
DirectX::XMMATRIX X = DirectX::XMMatrixRotationX(angle.x);
DirectX::XMMATRIX Y = DirectX::XMMatrixRotationY(angle.y);
DirectX::XMMATRIX Z = DirectX::XMMatrixRotationZ(angle.z);
DirectX::XMMATRIX R = Z * X * Y;
```

この計算を 1 つの関数で行っているのが XMMatrixRotationRollPitchYaw()です。

ここで考えて欲しいのは X、Y、Z の回転行列を乗算する順番です。
これは、まず Y 軸を回転させ、次に X 軸を回転させ、最後に Z 軸を回転させています。
順番が逆に感じるかもしれませんが、数学では右から順番に掛けていくルールだからです。

通常の掛け算は掛ける順番が違っても答えは同じなのですが、行列の掛け算は掛ける順番が違うと結果が変わるのです。



今まで、キャラクターは Y 軸の回転しかしなかったので回転順序を気にする必要はなかったのですが、今回は X 軸、Y 軸、Z 軸のすべてを制御することになります。
複数の軸を回転させると回転順によって結果が変わるので、今回やりたいことに対して正しい回転順序について考えましょう。

ヒントとしては地形に沿わせるために X 軸、Z 軸の回転を確定させ、Y 軸を回転することで正しい回転になるはずです。

○キャラクターの姿勢が壁に沿うように回転

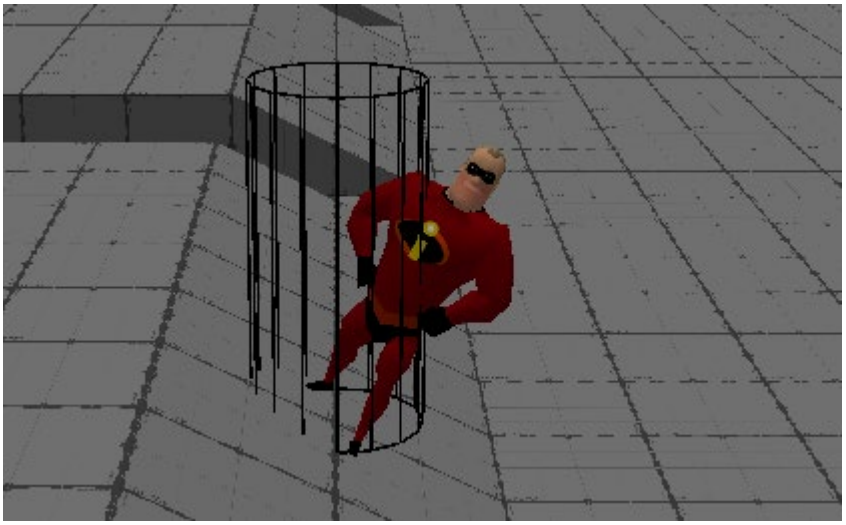
回転角の算出と回転順についてわかったところでプログラムを実装しましょう。

Character.cpp

```
---省略---

// 行列更新処理
void Character::UpdateTransform()
{
    ---省略---

    // 回転行列を作成
    DirectX::XMMATRIX R = DirectX::XMMatrixRotationRollPitchYaw(angle.x, angle.y, angle.z);
```

```
    DirectX::XMMATRIX X = DirectX::XMMatrixRotationX(angle.x);
    DirectX::XMMATRIX Y = DirectX::XMMatrixRotationY(angle.y);
    DirectX::XMMATRIX Z = DirectX::XMMatrixRotationZ(angle.z);
    DirectX::XMMATRIX R = Y * X * Z;

    ---省略---
}

---省略---

// 垂直移動更新処理
void Character::UpdateVerticalMove(float elapsedTime)
{
    // 垂直方向の移動量
    ---省略---

    // キャラクターのY軸方向となる法線ベクトル
    DirectX::XMFLOAT3 normal = { 0, 1, 0 };

    // 落下中
    if (my < 0.0f)
    {
        ---省略---

        // レイキャストによる地面判定
        HitResult hit;
        if (Stage::Instance().RayCast(start, end, hit))
        {
            // 法線ベクトル取得
            normal = hit.normal;

            ---省略---
        }
        ---省略---
    }
    // 上昇中
    else if (my > 0.0f)
    {
        ---省略---
    }

    // 地面の向きに沿うようにXZ軸回転
    {
        // Y軸が法線ベクトル方向に向くオイラー角回転を算出する
        angle.x = atan2f(normal.z, normal.y);
        angle.z = -atan2f(normal.x, normal.y);
    }
}
```

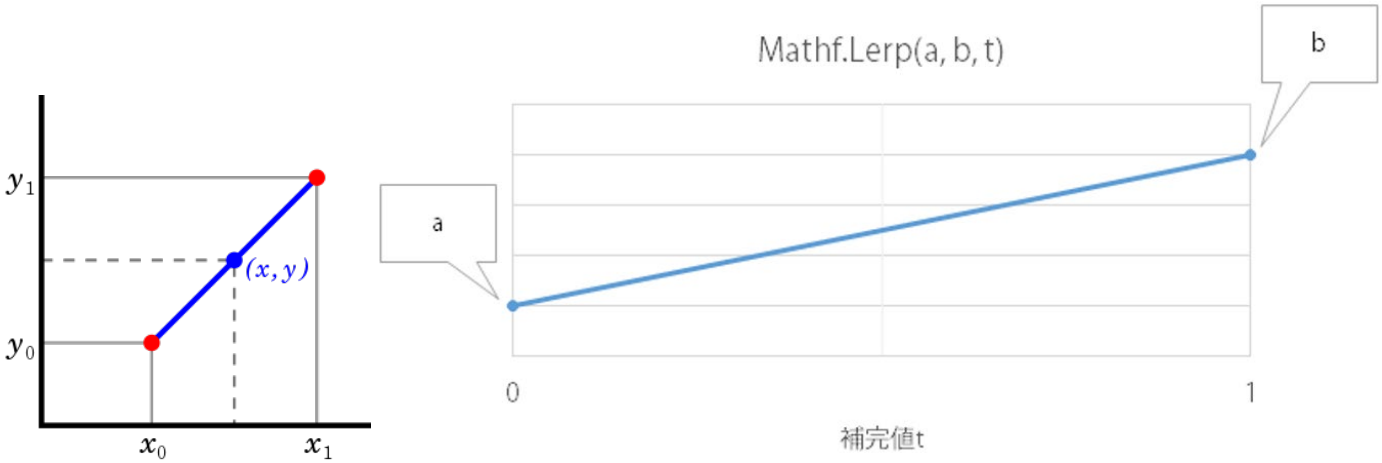
実装できたら実行確認してみましょう。
キャラクターを移動操作して地面に沿うように姿勢を回転できていれば OK です。



○回転補完

現状では回転の補完処理を行っていないため、地面の傾斜が変わると姿勢が一瞬で変わってしまいます。
Y 軸の回転は内積によるテクニックで滑らかな回転制御を行いましたが、今回はもっと簡易的な補完処理で実装してみましょう。

補完処理の計算で一番単純な方法として「線形補完」という方法があります。
線形補完とは、次のように２点（x0, y0）と（x1, y1）を直線で結んだときに間にある値を計算方法で、２点間を 0.0～1.0 の係数で補完値を求めます。



線形補完は大変便利でゲームプログラミングにおいて様々な所で使うので関数化しておきましょう。
Mathf.cpp と Mathf.h を作成し、下記プログラムコードを記述しましょう。

Mathf.h

```
#pragma once
```

```
// 浮動小数算術
class Mathf
{
public:
    // 線形補完
    static float Lerp(float a, float b, float t);
};
```

Mathf.cpp

```
#include "Mathf.h"

// 線形補完
float Mathf::Lerp(float a, float b, float t)
{
    return a * (1.0f - t) + (b * t);
}
```

線形補完を使って X、Z 軸を滑らかに回転させましょう

Character.cpp

```
---省略---

// 垂直移動更新処理
void Character::UpdateVerticalMove(float elapsedTime)
{
    ---省略---

    // 地面の向きに沿うようにXZ軸回転
    {
        // Y軸が法線ベクトル方向に向くオイラー角回転を算出する
        angle.x = atan2f(normal.z, normal.y);
        angle.z = -atan2f(normal.x, normal.y);
        float ax = atan2f(normal.z, normal.y);
        float az = -atan2f(normal.x, normal.y);

        // 線形補完で滑らかに回転する
        angle.x = Mathf::Lerp(angle.x, ax, 0.2f);
        angle.z = Mathf::Lerp(angle.z, az, 0.2f);
    }
}
```

実装出来たら実行確認をしてみましょう。
キャラクターを移動操作して傾斜が変わる場所で滑らかに回転できていれば OK です。

お疲れ様でした。