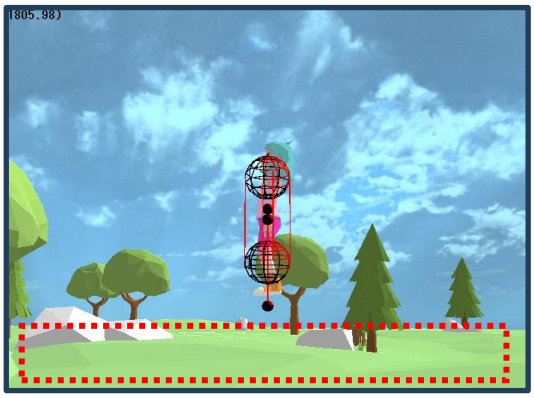
カメラの衝突判定

カメラがステージのモデルにめり込むことで、
普段、見えてはいけないものが見えてしまったり、
ポリゴンのバックカリングにより、モデルの裏側が描画されないため、
ステージが透明になったり、オブジェクトが宙に浮いているように見えます。



それらを防ぐために、カメラにも色々を工夫が必要です。
カメラとプレイヤーとの間に障害物があれば、その障害物を半透明にしたりする
別の技術もありますが、今回は、めり込ませないための衝突判定を行います。

現在の設計では、Colliderを付けるためには、
ActorBaseの継承が必要になりますので、カメラもActorBaseを継承させ、
Colliderを付けれるようにしていきいます。

①CameraのActor化

Camera.h

```
#pragma once
#include <DxLib.h>
#include "../Common/Quaternion.h"
#include "../Object/Actor/ActorBase.h"
class Transform;

class Camera : public ActorBase
{

    ～ 省略 ～

    // デストラクタ
```

```
~Camera(void) override;

// 初期化
void Init(void);        ⇒削除

// 更新
void Update(void) override;

～ 省略 ～

// 解放
void Release(void) override;

protected:

    // リソースロード
    void InitLoad(void) override {}

    // 大きさ、回転、座標の初期化
    void InitTransform(void) override {}

    // 衝突判定の初期化
    void InitCollider(void) override {}

    // アニメーションの初期化
    void InitAnimation(void) override {}

    // 初期化後の個別処理
    void InitPost(void) override;

private:

    // カメラの位置
    VECTOR pos_;            ⇒削除 transform_.posに置き換え

    // カメラ角度
    Quaternion rot_;        ⇒削除 transform_.quaRotに置き換え

    // カメラの上方向
    VECTOR cameraUp_;       ⇒削除 transform_.quaRot.GetUp()に置き換え
```

Camera.cpp

```
void Camera::InitPost(void)
{
    ChangeMode(MODE::FIXED_POINT);
}
```

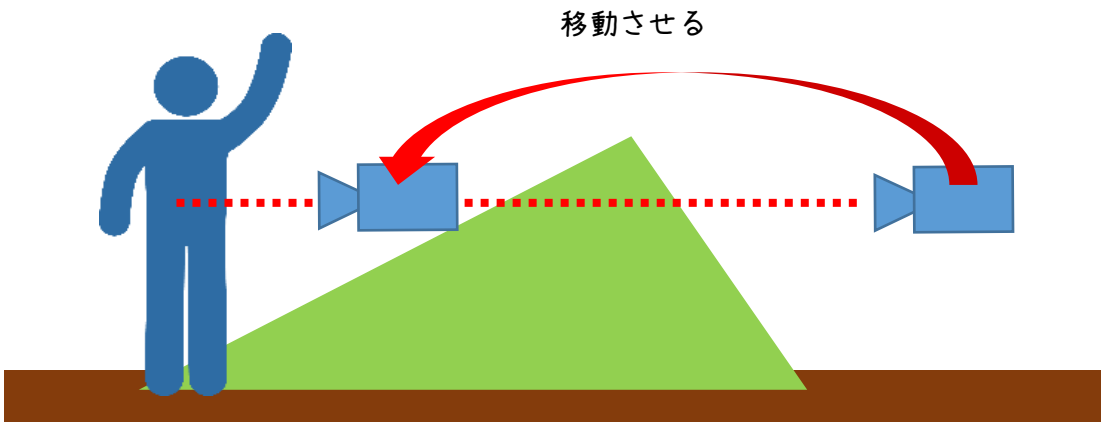
解説

カメラクラスにActorBaseを継承させるだけではなく、
座標や回転をTransformに管理させる必要がありますので、
メンバ変数の置き換えを行います。

カメラ角度は、完全にクォータニオンに置き換える方法もありますが、
3Dアクションゲームは、キャラクターの移動方向を計算する時などで、
X軸とY軸の回転を明確に分けたいことがありますので、
オイラー角制御のangles_は残しておき、
カメラ位置、注視点計算の際に、angles_からtransform_.quaRotに
クォータニオン変換するようにしています。

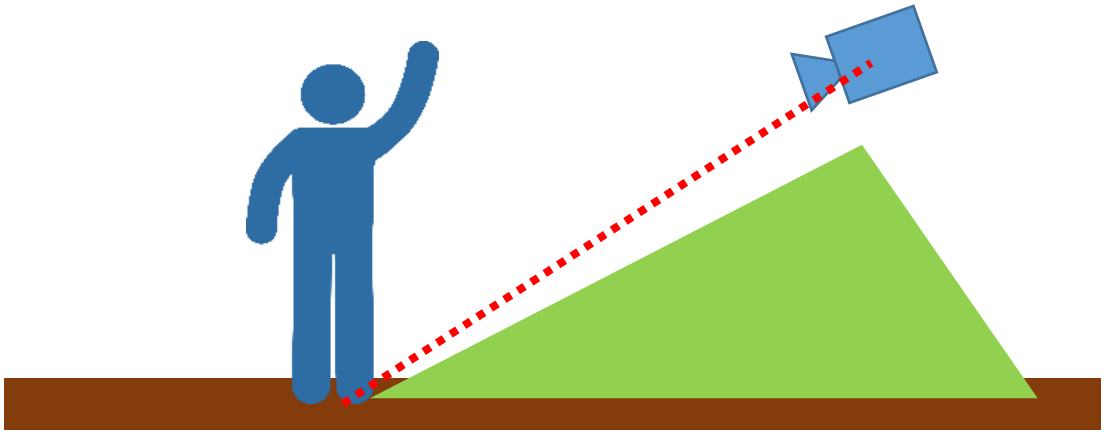
②プレイヤー ⇔ カメラ間の線分衝突

プレイヤーとカメラとの間で、線分の衝突判定を行い、
もし、衝突していたら、プレイヤーに最も近い衝突地点へカメラを移動させ、
めり込みや透明を発生させないようにします。



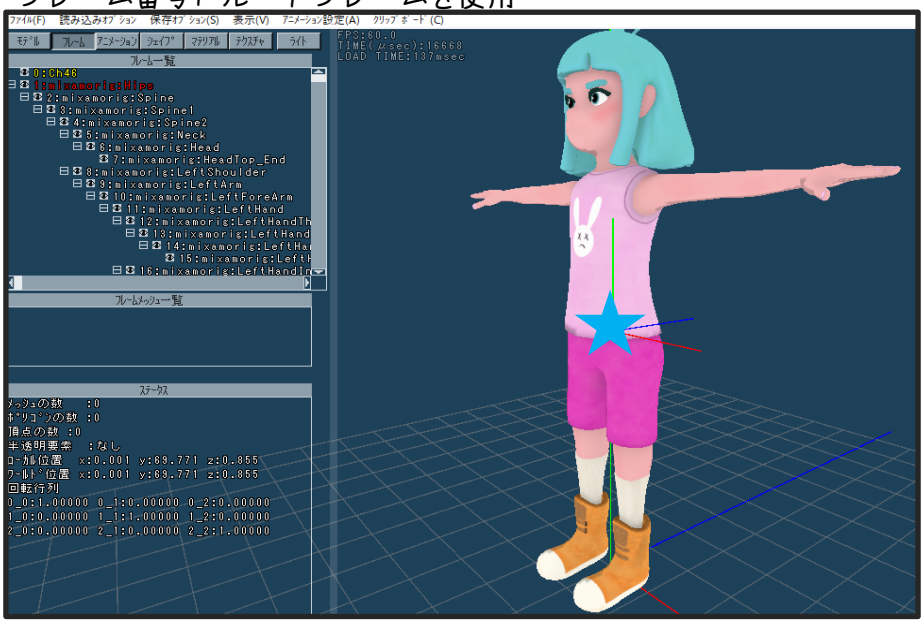
注意点①

カメラの位置から素直にキャラクター座標との間に線分を引いてしまうと、
下図のような位置関係になることが多く、
足元の地形に衝突しまい、プレイヤーに最接近してしまうことが
よくあります。

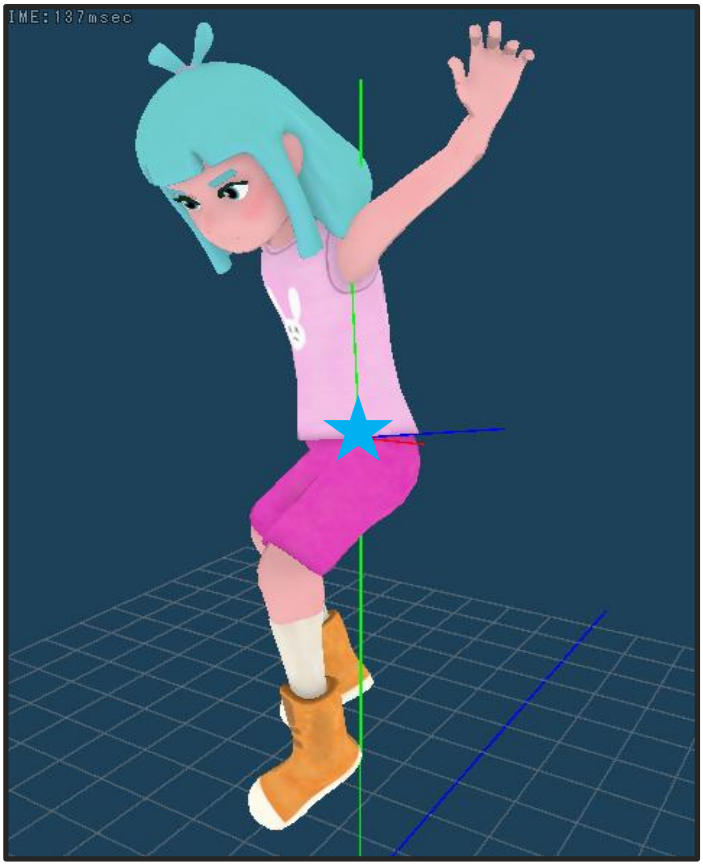


そういったトラブルを軽減するため、プレイヤーの腰あたりに
線分の座標を設定していきたいと思いますが、
プレイヤーのアニメーションによって、移動値が含まれているケースを
想定すると、また制御がややこしくなりますので、
腰あたりのフレーム座標を取得して、極力手間を無くしていきます。

フレーム番号1 ルートフレームを使用



フレーム座標は、アニメーションを考慮した上での座標を返してくれます。



注意点②

現在の線分コライダは、特定の2つのTransformに依存した形状に
対応していません。

この線分には対応していない

こういった線分には対応可

新しい線分コライダを拡張したり、
新しい線分コライダクラスを作っても良いですが、
今回は、カメラ自身の線分コライダは作成せず、自力で線分を作って、
ステージのモデルコライダを、カメラに登録するようにします。

Camera.h

```
private:

    // 衝突時の押し戻し量
    static constexpr float COLLISION_BACK_DIS = 2.0f;

    ～ 省略 ～

    // 衝突判定
    void Collision(void);
```

Camera.cpp

```
void Camera::SetBeforeDrawFollow(void)
{

    // カメラ操作(回転)
    ProcessRot(true);

    // 追従対象との相対位置を同期
    SyncFollow();

    // 衝突判定
    Collision();

}


void Camera::Collision(void)
{

    // プレイヤーのルートフレーム
    VECTOR start = MV1GetFramePosition(followTransform_->modelId, 1);
```

```
for (const auto& hitCol : hitColliders_)
{

    // モデル以外は処理を飛ばす
    if (hitCol->GetShape() != ColliderBase::SHAPE::MODEL) continue;

    // 派生クラスへキャスト
    const ColliderModel* colliderModel =
        dynamic_cast<const ColliderModel*>(hitCol);

    if (colliderModel == nullptr) continue;

    // 線分で衝突判定
    auto hits = MV1CollCheck_LineDim(
        colliderModel->GetFollow()->modelId,
        -1,
        transform_.pos,
        start
    );

    // 追従対象に一番近い衝突点を探す
    bool isCollision = false;
    MV1_COLL_RESULT_POLY hitPoly;
    double minDist = DBL_MAX;
    for (int i = 0; i < hits.HitNum; i++)
    {

        const auto& hit = hits.Dim[i];

        // 除外フレームは無視する
        ○○○

        // 衝突判定
        isCollision = true;

        // 距離判定
        ○○○
        ○○○
        {
            // 追従対象に一番近い衝突点を優先
            minDist = dist;
```

```
            hitPoly = hit;
        }

    }

    // 検出した地面ポリゴン情報の後始末
    MV1CollResultPolyDimTerminate(hits);

    if (!isCollision)
    {
        // 衝突していなければ次のコライダへ
        continue;
    }

    // カメラ位置から注視点への方向
    VECTOR dirToTarget = ○○○

    // 衝突点の少し手前にカメラを置く
    transform_.pos =
        VAdd(hitPoly.HitPosition, VScale(dirToTarget, COLLISION_BACK_DIS));

  }

}
```

【要件①】

線分とモデルの衝突判定を完成させること。

- 除外フレームとは衝突判定はおこなわない

- プレイヤーの腰座標から、衝突座標までの距離を測り、
  最もプレイヤーの腰座標に近いMV1_COLL_RESULT_POLY構造体を取得すること

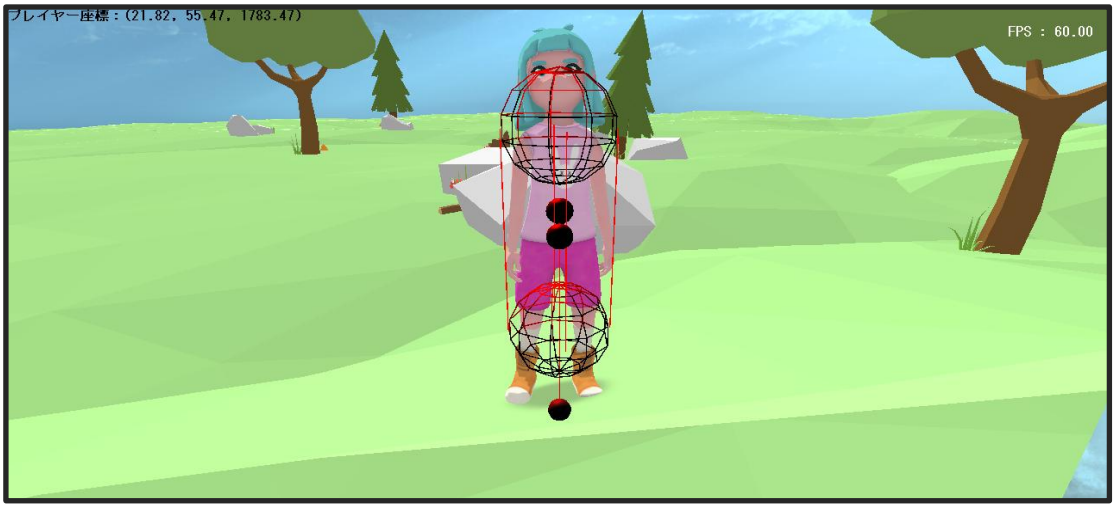
- カメラ位置から注視点に向かった単位ベクトルを取得し、
  最接近衝突地点から、注視点側にカメラを移動させること
  (衝突地点だと、ポリゴンとカメラが近すぎて、
    NEARより近くなり、何も描画されなくなる可能性をケアするため)

【目標②】

プレイヤーを丘の近くに移動させ、
わざとカメラを丘にめり込ませるように回転させると、
カメラがプレイヤーに近づき、
カメラがステージモデルにめり込まないようになること。

※ 但し、まだ完璧ではないので、多少の透明ポリゴンは見えてしまう

↓ カメラのY軸回転



③カメラの球体衝突

線分衝突だけだと、カメラのNEAR範囲よりも近いモデルが描画されず、
透明ポリゴンが、結構な頻度でチラチラ見えてしまいます。

押し戻し量がを増やせば良いのですが、
線分だけだと、衝突の範囲が狭く、画面端側のケアが難しいです。
そこで、新しいコライダ、球体コライダを作成して、
カメラに付けて、ステージのモデルコライダと衝突させて、対応していきます。

ColliderBase.h

```
public:

    // 衝突種別
    enum class TAG
    {
        STAGE,
        PLAYER,
        CAMERA,
    };
```

Camera.h

```
class ColliderSphere;

class Camera : public ActorBase
{
```

```
public:

    // 衝突判定種別
    enum class COLLIDER_TYPE
    {
        SPHERE,
        MAX,
    };

protected:

    void InitCollider(void) override;            中括弧削除

private:

    // 衝突時の押し戻し試行回数
    static constexpr int CNT_TRY_COLLISION_CAMERA = 30;

    // 衝突時の押し戻し量
    static constexpr float COLLISION_BACK_DIS = 2.0f;

    // 衝突判定用球体半径
    static constexpr float COL_CAPSULE_SPHERE = 50.0f;
```

Camera.cpp

```
#include "../Object/Collider/ColliderSphere.h"

void Camera::InitCollider(void)
{

    // 主に地面との衝突で使用する球体コライダ
    ColliderSphere* colliderSphere = new ColliderSphere(
        ColliderBase::TAG::CAMERA,
        &transform_,
        AsoUtility::VECTOR_ZERO,
        COL_CAPSULE_SPHERE
    );
    ownColliders_.emplace(
        static_cast<int>(COLLIDER_TYPE::SPHERE), colliderSphere);
```

```
}

void Camera::Collision(void)
{

    ～ 省略 ～

        // 衝突点の少し手前にカメラを置く
        transform_.pos =
            VAdd(hitPoly.HitPosition,
                VScale(dirToTarget, COLLISION_BACK_DIS));


        // カメラ位置の球体コライダ
        int typeSphere = static_cast<int>(COLLIDER_TYPE::SPHERE);

        // 球体コライダが無ければ処理を抜ける
        if (ownColliders_.count(typeSphere) == 0) continue;

        // 球体コライダ情報
        ColliderSphere* colliderSphere =
            dynamic_cast<ColliderSphere*>(ownColliders_.at(typeSphere));
        if (colliderSphere == nullptr) return;

        // 衝突補正処理
        int sphereCnt = 0;
        while (sphereCnt < CNT_TRY_COLLISION_CAMERA)
        {

            // 球体と三角形の当たり判定
            ○○○

            // 衝突していたら法線方向に押し戻し
            transform_.pos = ○○○

            sphereCnt++;

        }

    }
```

```
    }
```

ColliderSphere.h

```
#pragma once
#include <DxLib.h>
#include "ColliderBase.h"
class Transform;

class ColliderSphere : public ColliderBase
{

public:

    // コンストラクタ
    ColliderSphere(
        TAG tag, const Transform* follow, const VECTOR& localPos, float radius);

    // デストラクタ
    ~ColliderSphere(void);

    // 親Transformからの相対位置を取得
    const VECTOR& GetLocalPos(void) const;

    // 親Transformからの相対位置をセット
    void SetLocalPos(const VECTOR& localPos);

    // ワールド座標を取得
    VECTOR GetPos(void) const;

    // 半径
    float GetRadius(void) const;
    void SetRadius(float radius);

protected:

    // デバッグ用描画
    void DrawDebug(int color) override;

private:
```

```
    // 親Transformからの相対位置(下側)
    VECTOR localPos_;

    // 半径
    float radius_;

};
```

【要件②】

球体コライダによる、カメラとステージモデルの衝突処理を
完成させること。

- DxLibの関数の中から、三角形と球体の衝突判定を行う関数を探し出し、
  処理の中に組み込むこと

- 球体と三角形が衝突した場合、
  三角形の法線方向に少しずつカメラを移動させ、
  押し戻し処理を行うこと

- これまで作成したColliderクラスを参考に
  ColliderSphereクラスを完成させること

【目標②】

線分よりもふっくらとした衝突判定となり、
ステージの裏側に入って透明ポリゴンが見えてしまう問題が、
かなり改善されていること。