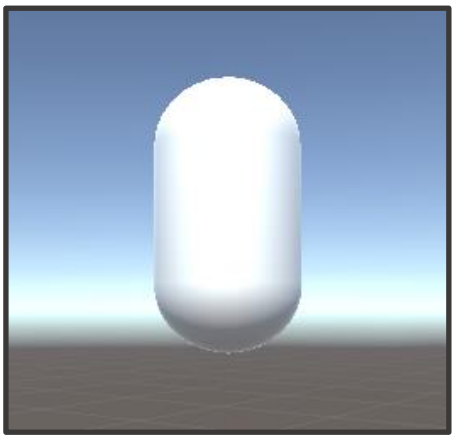
カプセルによる衝突判定

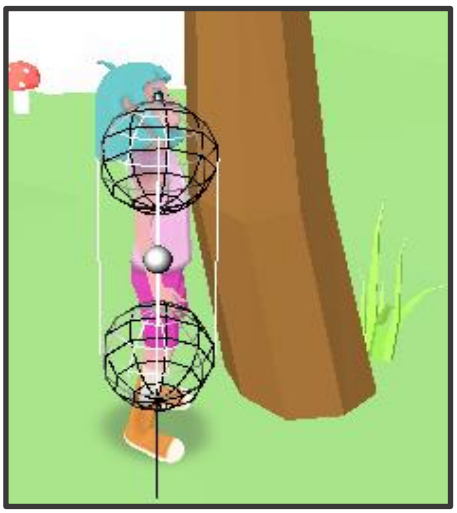
地面との衝突判定が実装できましたので、
次は、ステージや建物、キャラクターとの衝突判定をとっていきます。

地面との衝突判定を処理を分ける理由としては、
線分とモデルの衝突判定では、衝突ポリゴンは１つに絞ることができますが、
球体や、今回紹介するカプセルだと、複数のポリゴンと衝突してしまうため、
制御が複雑になるからです。

人型キャラクターの衝突判定には、よくカプセル型が使用されます。



カプセルは、複雑な形状にも見えますが、
衝突計算を行う上で、とてもシンプルな
形状をしており、左図だと、
上下に球体があって、それを繋ぐと
カプセル形状になります。



人型に近い形状に調整しやすく、
ある程度高低差のある地形でも自然に
動きやすくなります。

今回は、カプセルを使用して、
衝突判定をとっていきます。

Player.h

```
public:

    // 衝突用カプセルの取得
    const Capsule& GetCapsule(void) const;

private:

    std::unique_ptr<Capsule> capsule_;

    void CollisionCapsule(void);
```

Player.cpp

```
void Player::Init(void)
{

        ～ 省略 ～

        // カプセルコライダ
        capsule_ = std::make_unique<Capsule>(transform_);
        capsule_->SetLocalPosTop({ 0.0f, 110.0f, 0.0f });
        capsule_->SetLocalPosDown({ 0.0f, 30.0f, 0.0f });
        capsule_->SetRadius(20.0f);

}

const Capsule& Player::GetCapsule(void) const
{
    return *capsule_;
}

void Player::DrawDebug(void)
{

        ～ 省略 ～

        // カプセルコライダ
```

```
        capsule_->Draw();

}

void Player::Collision(void)
{

    // 現在座標を起点に移動後座標を決める
    movedPos_ = VAdd(transform_.pos, movePow_);

    // 衝突(カプセル)
    CollisionCapsule();

    // 衝突(重力)
    CollisionGravity();

    // 移動
    transform_.pos = movedPos_;

}

void Player::CollisionCapsule(void)
{

    // カプセルを移動させる
    Transform trans = Transform(transform_);
    trans.pos = movedPos_;
    trans.Update();
    Capsule cap = Capsule(*capsule_, trans);

    // カプセルとの衝突判定
    for (const auto c : colliders_)
    {

        auto hits = MV1CollCheck_Capsule(
            c.lock()->modelId_, -1,
            cap.GetPosTop(), cap.GetPosDown(), cap.GetRadius());

        // 衝突した複数のポリゴンと衝突回避するまで、
        // プレイヤーの位置を移動させる
        for (int i = 0; i < hits.HitNum; i++)
```

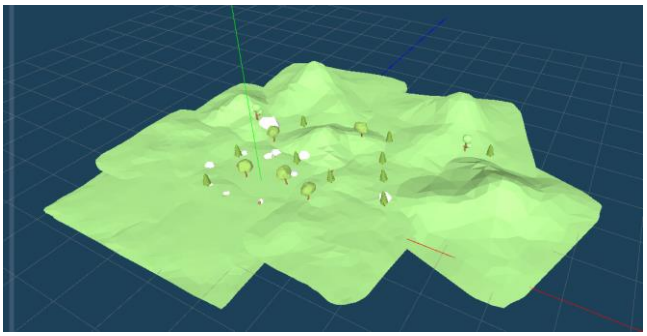
```
        {

            auto hit = hits.Dim[i];

            // 地面と異なり、衝突回避位置が不明なため、何度か移動させる
            // この時、移動させる方向は、移動前座標に向いた方向であったり、
            // 衝突したポリゴンの法線方向だったりする
            for (int tryCnt = 0; tryCnt < 10; tryCnt++)
            {

                // 再度、モデル全体と衝突検出するには、効率が悪過ぎるので、
                // 最初の衝突判定で検出した衝突ポリゴン1枚と衝突判定を取る
                int pHit = HitCheck_Capsule_Triangle(
                    cap.GetPosTop(), cap.GetPosDown(), cap.GetRadius(),
                    hit.Position[0], hit.Position[1], hit.Position[2]);

                if (pHit)
                {
                    // 法線の方向にちょっとだけ移動させる
                    movedPos_ = VAdd(movedPos_, VScale(hit.Normal, 1.0f));
                    // カプセルも一緒に移動させる
                    trans.pos = movedPos_;
                    trans.Update();
                    continue;
                }

                break;

            }

        }

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

    }

}
```

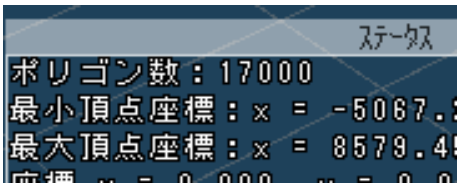
衝突判定の効率化

衝突判定は必須の機能ですが、処理負荷もそこそこ高いです。

```
auto hits = MV1CollCheck_Capsule(
    c.lock()->modelId_, -1,
    cap.GetPosTop(), cap.GetPosDown(), cap.GetRadius());
```

このコードだと、モデル内の全フレームの全ポリゴンと衝突判定を
行なおうとするため、17,000ポリゴンの処理が必要になります。





一方、下記のコードは、1枚のポリゴンとの衝突判定を行うため、

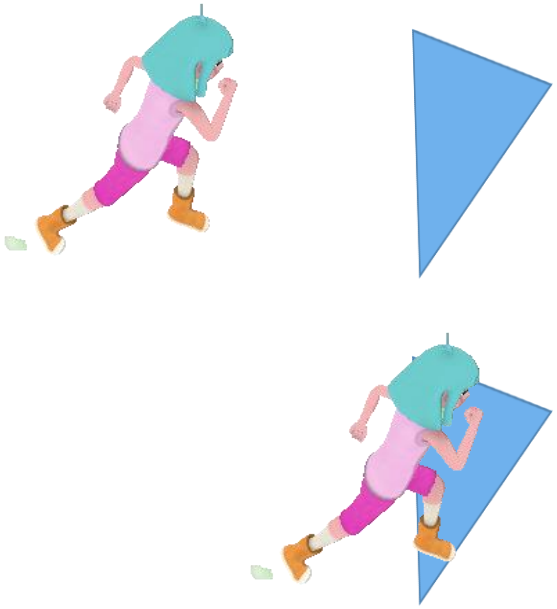
```
int pHit = HitCheck_Capsule_Triangle(
    cap.GetPosTop(), cap.GetPosDown(), cap.GetRadius(),
    hit.Position[0], hit.Position[1], hit.Position[2]);
```

単純に前者の17,000分の1の処理速度で実行できます。

モデル全てではなく、フレームを限定したり、
ポリゴンを限定した方が、お得です。

押し戻し(衝突回避)の考え方

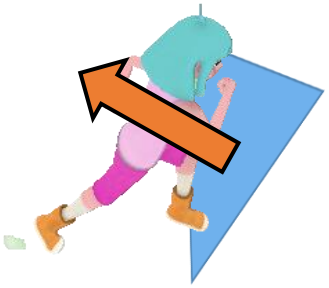
キャラクターと衝突する一枚のポリゴンがあったとして、



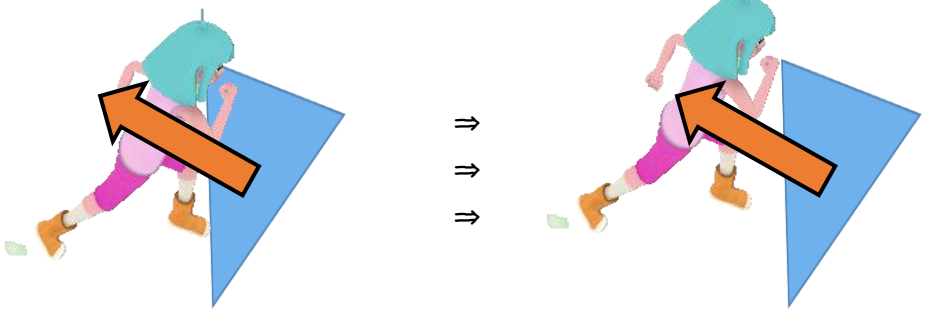
衝突！

どの位置にキャラクター座標を戻せば衝突が回避できるのかは、
明確にはわかりません。

そこで、ポリゴンの法線方向に、ちょっとだけ移動させます。



移動後、衝突チェックを再度行っても、まだ、衝突していますので、
衝突しなくなるまで、法線方向に移動させ続けます。



すると、いつかは衝突回避するはずなので、このポリゴンはクリアとなり、
もし、複数のポリゴンと衝突していたら、次のポリゴンとの衝突回避に
移ります。

これが３Ｄの一般的な衝突後の押し戻し処理になります。