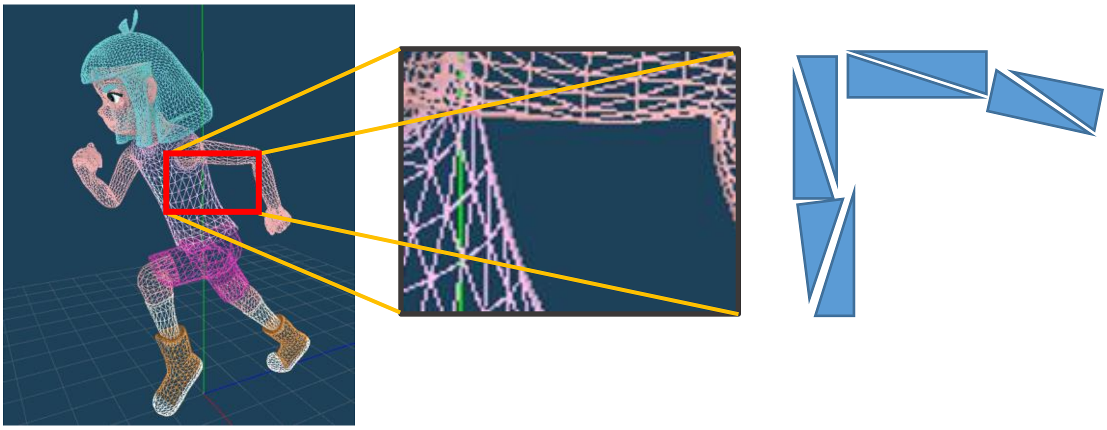
【衝突判定３D】三角形と球体

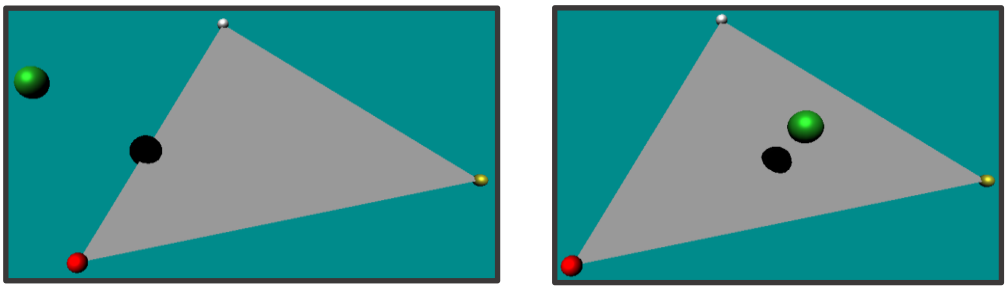
DxLibにはある程度の衝突判定機能が実装されていますが、
どうしても、自分の力で実装したい方に向けて基礎や考え方を解説します。

３Dポリゴンは、三角形の集まりになりますので、
三角形と球体の当たり判定を取り続ければ、モデルとの衝突判定が取れます。



但し、三角形の数が大量にありますので、最適化をしないと処理が遅くなってしまいます。それはまた別の資料で補足します。

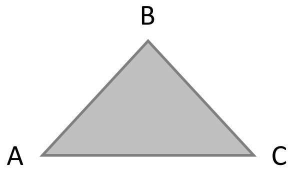
三角形と球体の当たり判定は、
球体の中心座標と、三角形面上の最近接点(最も近い座標)を計算して、
その最近接点と球体の中心座標との距離を測り、球体の半径より短ければ、
衝突している、という判定になります。



黒い球体が最近接点になりますが、
左図は、三角形の外側に球体の中心座標が位置しており、
右図は、三角形の内側に球体の中心座標が位置しています。

外側、内側というが、１つの大切な考え方になります。

外側の種類が全部で６種類あり、個別に判定していきます。



外側の種類

頂点Aに近い　頂点Bに近い

頂点Cに近い

辺ABに近い　辺BCに近い

辺ACに近い

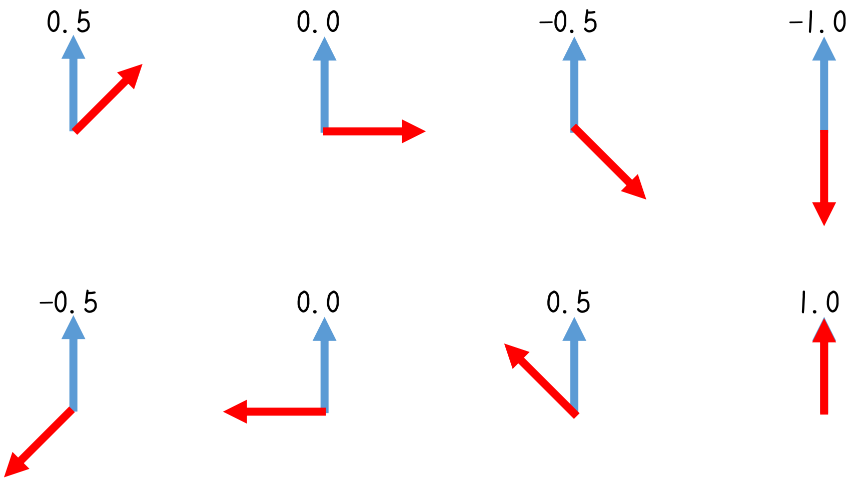
これから使う計算式の予備知識

P　：　球体の中心座標

AB　：　頂点AからBへのベクトル　　AC　：　頂点AからCへのベクトル

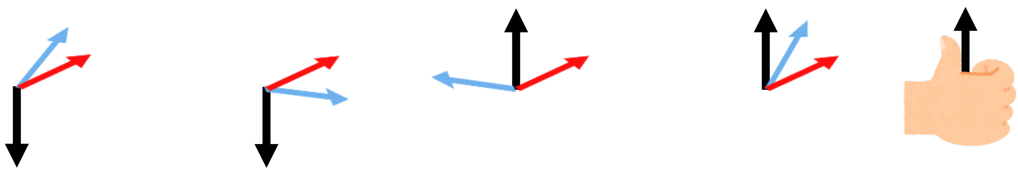
AP　：　頂点AからPへのベクトル

内積の結果(単位ベクトル同士の比較)



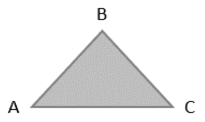
上図の通り、２つのベクトルの方向を正負で判定できる。

外積の結果

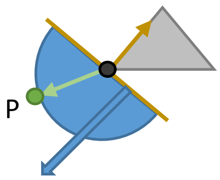


２つのベクトルの右ねじ方向の垂線(直角な方向)が取得できる。

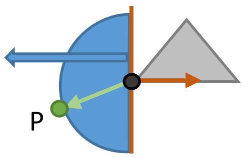
①最近接点が頂点Ａ



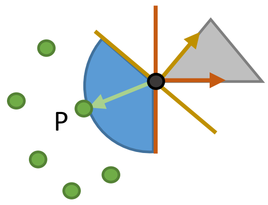
頂点ABCは左図の通り、



ベクトルABとAPの内積を見た時、
結果が0以下の場合、
点Pは、左図の青い範囲の方向に位置することがわかります。

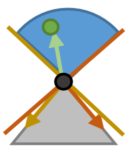


同じように、
ベクトルACとAPの内積を見た時、
結果が0以下の場合、
点Pは、左図の青い範囲の方向に位置することがわかります。



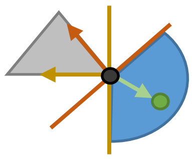
ということは、
ABとAPの内積結果と、ACとAPの内積結果が
両方、0以下だった場合、最近接点はＡとなる。

②最近接点が頂点Ｂ



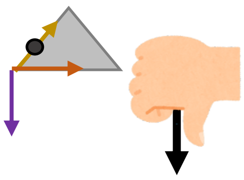
頂点Aと同じように、
BAとBPの内積結果と、BCとBPの内積結果が
両方、0以下だった場合、最近接点はＢとなる。

③最近接点が頂点Ｃ

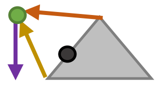


頂点A、頂点Bと同じように、
CAとCPの内積結果と、CBとCPの内積結果が
両方、0以下だった場合、最近接点はＣとなる。

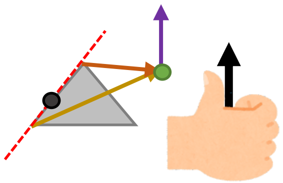
④最近接点が辺ＡＢ



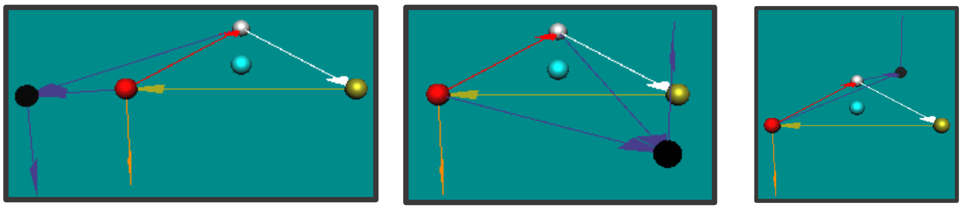
ベクトルABとACの外積を見た時、
直交するベクトルが算出できる。
外積AC×ABだと、逆の直交ベクトルになるので、
注意してください。
ここでは、AB×ACです。

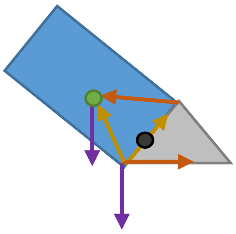


ベクトルAPとBPの外積を見た時、
直交するベクトルが算出できる。

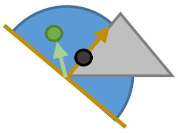


最近接点が辺AB以外の場合、
図のように辺ABの直交を境に、ベクトルが反転する。

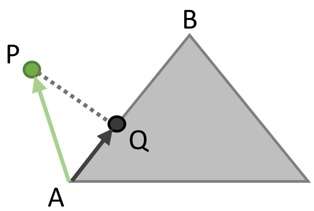




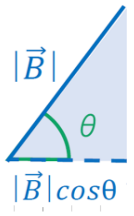
算出した２種類の外積AB×ACとAP×BPの内積結果を見て、
0以上であれば、同じ方向を向いているので、
点Pは、左図の青い範囲の方向に位置することが
わかりますので、最近接点は辺ＡＢとなる。



念のため、
AB・AP(内積)が0以上、AB・BP(内積)が0以下の条件を
加えた方が無難ではある。



最近接点が辺ＡＢと判定されてたら、
辺ＡＢ上の最近接点を求める必要があるので、
最近接点をＱとした場合、内積が表す射影を用いて、
AQを求める。



$$|\vec{B}|\cos\theta = \frac{\vec{A}\cdot\vec{B}}{|\vec{A}|}$$

AQの長さは、
　AB・AP ÷ ABの長さ
　(ABの長さは、３平方の定理)
ベクトルAQは、
　ABの単位ベクトル × AQの長さ
点Ｑは、
　頂点Ａ + ベクトルAQ
となります。

⑤最近接点が辺ＢＣ

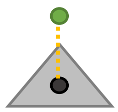
BC×AB・BP×CP が0以上であれば、最近接点が辺ＢＣ。
最近接点 = 頂点Ｂ + (BCの単位ベクトル × (BP・BC ÷ 頂点BCの長さ))

⑥最近接点が辺ＣＡ

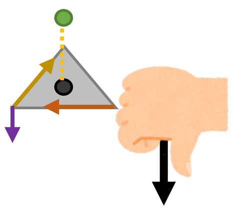
CA×BC・CP×AP が0以上であれば、最近接点が辺ＣＡ。
最近接点 = 頂点Ｃ + (CAの単位ベクトル × (CP・CA ÷ 頂点CAの長さ))

⑦最近接点が三角形ＡＢＣの面上

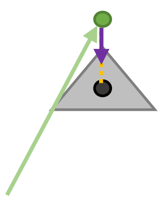
①～⑥のいずれの条件にも該当しない場合、
最近接点は、三角形ＡＢＣの面上に位置することになる。

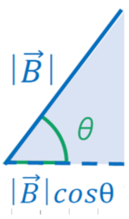


辺上の最近接点を求めたのと同じように、
内積の射影を使って、面上の座標を算出する。



外積AB×CAを正規化して、三角形の直交ベクトルＮを求める。



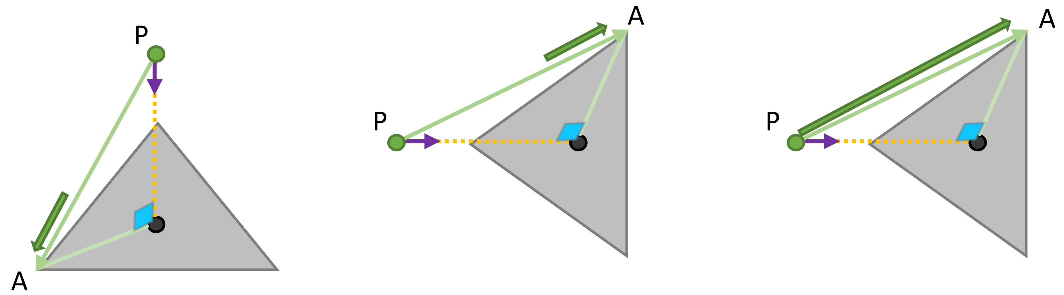


$$|\vec{B}|\cos\theta = \frac{\vec{A}\cdot\vec{B}}{|\vec{A}|}$$

長さが欲しい方が、
公式のAベクトルになりますので、
三角形の直交ベクトルＮ(紫)とします。
正規化されていますので、長さは1となっており、
割り算が省略されます。

よって、Ｎ・AP(内積)により、点Ｐから三角形の面上までの長さLENが
求められますが、射影で使用する向きが本来の使い方とは逆になっているため、
頂点Ｐ ＋ (直行ベクトル×-LEN)とすると、三角形面上の最近接点が求められます。

(本来の射影ベクトル)APではなく、ベクトルPAを使用するとイメージしやすい。



今回の解説は、できるだけわかりやすいように単純な数学式を使用しています。
しかし、計算量が多いため、実用的ではないというデメリットがあります。

数学ではなく、ゲーム数学は、ここからできるだけ計算量を少なくするために
最適化したり、数式を崩していく必要があります。

今回のベクトルPAもAPを反転させるためには、掛け算をXYZ、
合計３回行う必要がありますが、内積を求めた後の実数(float)だったら、
掛け算１回で済みますので、PAを使わない方が計算が早いということになります。

今回の式の中で、こういったことを、かなりの箇所でチューニングことができます。

最終的には、コストが重い平方根を一切無くして、
外積の変わりにラグランジュの公式により、

$$(A \times B) \cdot (C \times D) = (A \cdot C)(B \cdot D) - (A \cdot D)(B \cdot C)$$

内積計算のみにしてあげたり、内積判定を逆転させて、
内積箇所を減らしたりします。

また、衝突判定自体の高速化については、別紙で解説致します。

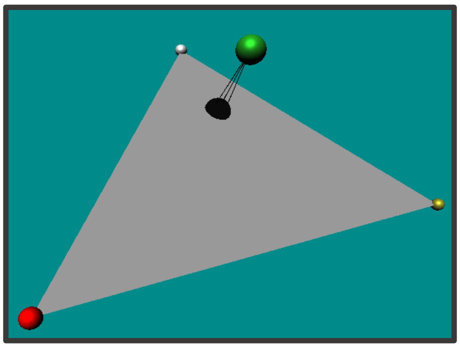
⑧例外チェックを行う

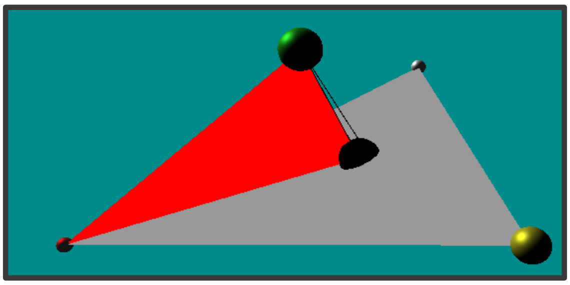
⑦の計算上で、三角形の法線を求める箇所がありますが、
仮に一直線上に頂点が並んだなど、法線ベクトルが上手く取れない
場合があります。

通常の三角形　　　　頂点が一直線上に並んでいる

三角面上の最近接座標が取れませんので、点Pに最も近い
頂点ABCのいずれかを最近接座標とするのが良いでしょう。

数学は、イメージするのが難しかったかもしれませんが、
ゲーム数学の場合は、自分で三角形を描画したり、外積で求めたベクトルを
矢印で描画したりして、数式の結果をグラフィカルに確認することができます。





数式の結果を１つ１つ丁寧に画面に表示していくと、
イメージがしやすくなり、今まで理解しにくかったことが理解しやすくなり
ますので、ぜひ試してみてください。

```
VECTOR AsoUtility::GetClosestPosTriangleBeta(
    const VECTOR& tPos1, const VECTOR& tPos2, const VECTOR& tPos3,
    const VECTOR& sPos1)
{

    VECTOR ab = VSub(tPos2, tPos1);
    VECTOR ac = VSub(tPos3, tPos1);
    VECTOR ba = VSub(tPos1, tPos2);
    VECTOR bc = VSub(tPos3, tPos2);
    VECTOR ca = VSub(tPos1, tPos3);
    VECTOR cb = VSub(tPos2, tPos3);

    // 最近接点　頂点Ａ
    VECTOR ap = VSub(sPos1, tPos1);
    float dotABAP = VDot(ab, ap);
    float dotACAP = VDot(ac, ap);
    if (dotABAP <= 0.0f && dotACAP <= 0.0f)
    {
        return tPos1;
    }

    // 最近接点　頂点Ｂ
    VECTOR bp = VSub(sPos1, tPos2);
    float dotBABP = VDot(ba, bp);
    float dotBCBP = VDot(bc, bp);
    if (dotBABP <= 0.0f && dotBCBP <= 0.0f)
    {
        return tPos2;
    }

    // 最近接点　頂点Ｃ
    VECTOR cp = VSub(sPos1, tPos3);
    float dotCACP = VDot(ca, cp);
    float dotCBCP = VDot(cb, cp);
    if (dotCACP <= 0.0f && dotCBCP <= 0.0f)
    {
        return tPos3;
    }
```

```
// 最近接点　辺ＡＢ
VECTOR crossABCA = VCross(ab, ca);
VECTOR crossAPBP = VCross(ap, bp);
float dotAP = VDot(crossABCA, crossAPBP);
if (dotAP >= 0.0f)
{
    // 頂点Aと点Pの射影座標
    float dotAPAB = VDot(ap, ab);
    float disAB = Distance(tPos1, tPos2);
    return VAdd(tPos1, VScale(VNorm(ab), dotAPAB / disAB));
}

// 最近接点　辺ＢＣ
VECTOR crossBCAB = VCross(bc, ab);
VECTOR crossBPCP = VCross(bp, cp);
float dotBP = VDot(crossBCAB, crossBPCP);
if (dotBP >= 0.0f)
{
    // 頂点Bと点Pの射影座標
    float dotBPBC = VDot(bp, bc);
    float disBC = Distance(tPos2, tPos3);
    return VAdd(tPos2, VScale(VNorm(bc), dotBPBC / disBC));
}

// 最近接点　辺ＣＡ
VECTOR crossCABC = VCross(ca, bc);
VECTOR crossCPAP = VCross(cp, ap);
float dotCP = VDot(crossCABC, crossCPAP);
if (dotCP >= 0.0f)
{
    // 頂点Cと点Pの射影座標
    float dotCPCA = VDot(cp, ca);
    float disCA = Distance(tPos3, tPos1);
    return VAdd(tPos3, VScale(VNorm(ca), dotCPCA / disCA));
}

// 最近接点　面ＡＢＣ
VECTOR tCrossN = VNorm(crossABCA);
if (tCrossN.x == -1.0f && tCrossN.y == -1.0f && tCrossN.z == -1.0f)
{
```

```
        // 一直線上に頂点が並び、法線ベクトルが取れない
        // この場合は、３頂点のうち、最も近い頂点を返す
        float aPow = SqrMagnitudeF(ap);
        float bPow = SqrMagnitudeF(bp);
        float cPow = SqrMagnitudeF(cp);
        if (aPow <= bPow)
        {
            if (aPow <= cPow) { return tPos1; }
            else { return tPos3; }
        }
        else
        {
            if (bPow <= cPow) { return tPos2; }
            else { return tPos3; }
        }

    }

    // 三角形の法線とベクトルAPを使用して面上の射影を落とす
    float d = VDot(tCrossN, ap);

    // 球体の中心座標から射影の長さ分、三角形の法線方向にベクトルを伸ばすと、
    // 三角形面上の最近接点が求められる
    return VAdd(sPos1, VScale(tCrossN, -d));

}
```

```
bool AsoUtility::IsHitTriangleSphere(
    const VECTOR& tPos1, const VECTOR& tPos2, const VECTOR& tPos3,
    const VECTOR& sPos1, float radius)
{

    // 三角形面上の最近接点を求める
    VECTOR pos = GetClosestPosTriangleBeta(tPos1, tPos2, tPos3, sPos1);

    // 最近接点と球体の中心点との長さ(２乗)を求める
    float disPow = SqrMagnitudeF(VSub(sPos1, pos));

    // 半径の２乗と比較する
    if (disPow <= radius * radius)
    {
        return true;
    }

    return false;

}
```

ちなみに、DxLibで使用されている三角形上の最近接点を求める関数は、
以下の通りです。

```
// 点に一番近い三角形上の座標を得る
extern VECTOR    Get_Triangle_Point_MinPosition(
   VECTOR Point, VECTOR TrianglePos1, VECTOR TrianglePos2, VECTOR TrianglePos3 )
{
   VECTOR Line12, Line23, Line31, Line1P, Line2P, Line3P, Result ;
   float Dot1P2, Dot1P3, Dot2P1, Dot2P3, Dot2PH, Dot3P1, Dot3P2, Dot3PH;
   float OPA, OPB, OPC, Div, t, v, w ;

   VectorSub( &Line12, &TrianglePos2, &TrianglePos1 ) ;
   VectorSub( &Line31, &TrianglePos1, &TrianglePos3 ) ;
   VectorSub( &Line1P, &Point,        &TrianglePos1 ) ;
   Dot1P2 = VectorInnerProduct( &Line12, &Line1P ) ;
   Dot1P3 = VectorInnerProduct( &Line31, &Line1P ) ;
   if( Dot1P2 <= 0.0f && Dot1P3 >= 0.0f ) return TrianglePos1 ;
```

```
VectorSub( &Line23, &TrianglePos3, &TrianglePos2 ) ;
VectorSub( &Line2P, &Point,        &TrianglePos2 ) ;
Dot2P1 = VectorInnerProduct( &Line12, &Line2P ) ;
Dot2P3 = VectorInnerProduct( &Line23, &Line2P ) ;
if( Dot2P1 >= 0.0f && Dot2P3 <= 0.0f ) return TrianglePos2 ;

Dot2PH = VectorInnerProduct( &Line31, &Line2P ) ;
// ↓ラグランジュ恒等式
OPC = Dot1P2 * -Dot2PH - Dot2P1 * -Dot1P3 ;
if( OPC <= 0.0f && Dot1P2 >= 0.0f && Dot2P1 <= 0.0f )
{
   t = Dot1P2 / ( Dot1P2 - Dot2P1 ) ;
   Result.x = TrianglePos1.x + Line12.x * t ;
   Result.y = TrianglePos1.y + Line12.y * t ;
   Result.z = TrianglePos1.z + Line12.z * t ;
   return Result ;
}

VectorSub( &Line3P, &Point,        &TrianglePos3 ) ;
Dot3P1 = VectorInnerProduct( &Line31, &Line3P ) ;
Dot3P2 = VectorInnerProduct( &Line23, &Line3P ) ;
if( Dot3P1 <= 0.0f && Dot3P2 >= 0.0f ) return TrianglePos3 ;

Dot3PH = VectorInnerProduct( &Line12, &Line3P ) ;
// ↓ラグランジュ恒等式
OPB = Dot3PH * -Dot1P3 - Dot1P2 * -Dot3P1 ;
if( OPB <= 0.0f && Dot1P3 <= 0.0f && Dot3P1 >= 0.0f )
{
   t = Dot3P1 / ( Dot3P1 - Dot1P3 ) ;
   Result.x = TrianglePos3.x + Line31.x * t ;
   Result.y = TrianglePos3.y + Line31.y * t ;
   Result.z = TrianglePos3.z + Line31.z * t ;
   return Result ;
}

// ↓ラグランジュ恒等式
OPA = Dot2P1 * -Dot3P1 - Dot3PH * -Dot2PH ;
if( OPA <= 0.0f && ( -Dot2PH - Dot2P1 ) >= 0.0f
   && ( Dot3PH + Dot3P1 ) >= 0.0f )
{
```

```
        t = (-Dot2PH - Dot2P1) / ((-Dot2PH - Dot2P1) + (Dot3PH + Dot3P1));
        Result.x = TrianglePos2.x + Line23.x * t ;
        Result.y = TrianglePos2.y + Line23.y * t ;
        Result.z = TrianglePos2.z + Line23.z * t ;
        return Result ;
    }

    Div = 1.0f / ( OPA + OPB + OPC ) ;
    v = OPB * Div ;
    w = OPC * Div ;
    Result.x = TrianglePos1.x + Line12.x * v - Line31.x * w ;
    Result.y = TrianglePos1.y + Line12.y * v - Line31.y * w ;
    Result.z = TrianglePos1.z + Line12.z * v - Line31.z * w ;
    return Result ;
}
```