```
// Copyright (C) Square Enix Co., Ltd. All Rights Reserved.
// The Pseudo Code of the Ray-Phong Tessellation Intersection Test
Binary PhongTessellation::Intersect( Ray &ray, Real &depth, Vector3D &barycentric_coord ) const
{

  Real param;
  Real A, B, C, D, E, F;
  Real la1, lb1, lc1, la2, lb2, lc2;
  Real u, v, w;
  Real g, h;
  Real c0, c1, c2;
  Real discriminant, sqrt_discriminant;

  const Vector3D &T1 = ray.PlaneNormal1;
  const Vector3D &T2 = ray.PlaneNormal2;

  const Vector3D positions[3] = { GlobalGeometry::Vertices[VertexIDs[0]].Position, GlobalGeometry::Vertices[VertexIDs[1]].Position, GlobalGeometry::Vertices[VertexIDs[2]].Position };
  const Vector3D normals  [3] = { GlobalGeometry::Vertices[VertexIDs[0]].Normal  , GlobalGeometry::Vertices[VertexIDs[1]].Normal  , GlobalGeometry::Vertices[VertexIDs[2]].Normal   };

  const Vector3D e01 = positions[1] - positions[0];
  const Vector3D e12 = positions[2] - positions[1];
  const Vector3D e20 = positions[0] - positions[2];

  const Vector3D C1 = ( normals[1] * normals[1].Dot( e01 ) - normals[0] * normals[0].Dot( e01 ) ) * Alpha;
  const Vector3D C2 = ( normals[2] * normals[2].Dot( e12 ) - normals[1] * normals[1].Dot( e12 ) ) * Alpha;
  const Vector3D C3 = ( normals[0] * normals[0].Dot( e20 ) - normals[2] * normals[2].Dot( e20 ) ) * Alpha;

  // The Pencil of M and N is Given as Mx + N

  //           a d e
  // Matrix M = d b f
  //           e f c

  //           l o p
  // Matrix N = o m q
  //           p q n

  const Real a = -T1.Dot( C3 );
  const Real l = -T2.Dot( C3 );
  const Real b = -T1.Dot( C2 );
  const Real m = -T2.Dot( C2 );
  const Real c = T1.Dot( positions[2] ) + ray.PlaneConstant1;
  const Real n = T2.Dot( positions[2] ) + ray.PlaneConstant2;
  const Real d = T1.Dot( C1 - C2 - C3 ) * 0.5;
  const Real o = T2.Dot( C1 - C2 - C3 ) * 0.5;
  const Real e = T1.Dot( C3 + e20 ) * 0.5;
  const Real p = T2.Dot( C3 + e20 ) * 0.5;
  const Real f = T1.Dot( C2 - e12 ) * 0.5;
  const Real q = T2.Dot( C2 - e12 ) * 0.5;

  // auu+bvv+c+2duv+2eu+2fv
  // luu+mvv+n+2ouv+2pu+2qv

  // Cubic Equation
  const Real coeffs[4] =
  {
    ( l*m*n + 2.0 * o*p*q )                                  - ( l*q*q + m*p*p + n*o*o ),
    ( a*m*n + l*b*n + l*m*c + 2.0 * (d*p*q + o*e*q + o*p*f) ) - ( a*q*q + b*p*p + c*o*o + 2.0 * (l*f*q + m*e*p + n*d*o) ),
    ( a*b*n + a*m*c + l*b*c + 2.0 * (o*e*f + d*e*q + d*p*f) ) - ( l*f*f + m*e*e + n*d*d + 2.0 * (a*f*q + b*e*p + c*d*o) ),
    ( a*b*c + 2.0 * d*e*f )                                  - ( a*f*f + b*e*e + c*d*d )
  };

  // Sovle Cubic
  Real    xs[3];
  Integer num = FunctionSolver::Cubic(coeffs, xs);
  if( 0 == num ) { return false; };

  Real   x;
  Real   determinant = Infinity;
  Binary flag        = false;
  for(Integer i = 0; i < num; ++i)
  {
    A = a * xs[i] + l;
    B = b * xs[i] + m;
    D = d * xs[i] + o;
    if ( determinant > (D*D-A*B) ) { determinant = D*D-A*B; x = xs[i]; }
  }

  const Integer domain = ray.Direction.Domain();
  {
    A = a * x + l;
    B = b * x + m;
    D = d * x + o;

    // Avoid an Ellipsoid whose Area is Zero
    if( 0.0 < determinant )
    {
      C = c * x + n;
      E = e * x + p;
      F = f * x + q;

      // Choose A or B, whichever Suitable
```

```
if( Abs(A) < Abs(B))
{
  // Factorizing by Using an Irreducible Equation
  // ( x y 1 ) S ( x y 1 )' = ( a1 x + b1 y + c1 )( a2 x + b2 y + c2 )
  // ( x y 1 ) S ( x y 1 )' = ( a1 x +    y + c1 )( a2 x +    y + c2 )
  A /= B; C /= B; D /= B; E /= B; F /= B; // B = 1.0;
  Real sqrtA = Sqrt( D*D - A );
  Real sqrtC = Sqrt( F*F - C );
  la1 = D + sqrtA;
  la2 = D - sqrtA;
  lc1 = F + sqrtC;
  lc2 = F - sqrtC;

  //if(Delta64 < Abs( (2.0*E) - (la1*lc2 + la2*lc1)) )
  if( Abs( (2.0*E) - (la1*lc1 + la2*lc2)) < Abs( (2.0*E) - (la1*lc2 + la2*lc1)) )
  {
    Swap( lc1, lc2 );
  }
  for(Integer loop = 0; loop < 2; ++loop)
  {
    g = (0 == loop) ? -la1 : -la2;
    h = (0 == loop) ? -lc1 : -lc2;

    // Solve the Quadratic Equation f(y) = c0 u u + c1 u + c2 = 0
    c0 = a + (2.0 * d + b * g) * g;
    c1 = 2.0 * ( (d + b * g) * h + e + f * g );
    c2 = (b * h + 2.0 * f) * h + c;

    if(0.0 < c0 && (0.0 < c1 && 0.0 < c2 || 0.0 > c2 && 0.0 > c0 + c1 + c2)) { continue; }
    if(0.0 > c0 && (0.0 > c1 && 0.0 > c2 || 0.0 < c2 && 0.0 < c0 + c1 + c2)) { continue; }

    if( 0.00000000001 > Abs(c0) )
    {
      if( 0.00000000001 < Abs(c1))
      {
        u = -c2 / c1; v = g * u + h; w = 1.0 - (u + v);
        if(0.0 <= u && 0.0 <= v && 0.0 <= w)
        {
          param = (EvaluatePhongTessellation( u, v, w, positions, normals, domain ) - ray.Position[domain]) / ray.Direction[domain];
          if( Epsilon < param && param < ray.Depth - Epsilon && param < depth )
          {
            barycentric_coord   = Vector3D(u, v, w);
            depth = param;
            flag  = true;
          }
        }
      }
    }
    else
    {
      Real tmp0, tmp1;
      discriminant = (c1 * c1) - 4.0 * (c0 * c2);
      if( 0.0 <= discriminant )
      {
        FunctionSolver::Quadratic( c0, c1, c2, tmp0, tmp1 );

        // with Line 1
        u = tmp0; v = g * u + h; w = 1.0 - (u + v);
        if(0.0 <= u && 0.0 <= v && 0.0 <= w)
        {
          param = (EvaluatePhongTessellation( u, v, w, positions, normals, domain ) - ray.Position[domain]) / ray.Direction[domain];
          if( Epsilon < param && param < ray.Depth - Epsilon && param < depth )
          {
            barycentric_coord   = Vector3D(u, v, w);
            depth = param;
            flag  = true;
          }
        }

        // with Line 2
        u = tmp1; v = g * u + h; w = 1.0 - (u + v);
        if(0.0 <= u && 0.0 <= v && 0.0 <= w)
        {
          param = (EvaluatePhongTessellation( u, v, w, positions, normals, domain ) - ray.Position[domain]) / ray.Direction[domain];
          if( Epsilon < param && param < ray.Depth - Epsilon && param < depth  )
          {
            barycentric_coord   = Vector3D(u, v, w);
            depth = param;
            flag  = true;
          }
        }
      }
    }
  }
}
else
{
  // Factorizing by Using an Irreducible Equation
  // ( x y 1 ) S ( x y 1 )' = ( a1 x + b1 y + c1 )( a2 x + b2 y + c2 )
  // ( x y 1 ) S ( x y 1 )' = (    x + b1 y + c1 )(    x + b2 y + c2 )
  B /= A; C /= A; D /= A; E /= A; F /= A; // A = 1.0;
  Real sqrtB = Sqrt( D*D - B );
```

```
        Real sqrtC = Sqrt( E*E - C );
        lb1 = D + sqrtB; // lb = D +- SqrtB
        lb2 = D - sqrtB;
        lc1 = E + sqrtC; // lc = E +- SqrtC
        lc2 = E - sqrtC;
        //if(Delta64 < Abs( (2.0*F) - (lb1*lc2 + lb2*lc1)) )
        if(Abs( (2.0*F) - (lb1*lc1 + lb2*lc2)) < Abs( (2.0*F) - (lb1*lc2 + lb2*lc1)) )
        {
          Swap( lc1, lc2 );
        }
        for(Integer loop = 0; loop < 2; ++loop)
        {
          g = (0 == loop) ? -lb1 : -lb2;
          h = (0 == loop) ? -lc1 : -lc2;

          // Solve the Quadratic Equation f(y) = c0 v v + c1 v + c2 = 0
          c0 = b + (2.0 * d + a * g) * g;
          c1 = 2.0 * ( (d + a * g) * h + f + e * g );
          c2 = (a * h + 2.0 * e) * h + c;

          if(0.0 < c0 && (0.0 < c1 && 0.0 < c2 || 0.0 > c2 && 0.0 > c0 + c1 + c2)) { continue; }
          if(0.0 > c0 && (0.0 > c1 && 0.0 > c2 || 0.0 < c2 && 0.0 < c0 + c1 + c2)) { continue; }

          if( 0.00000000001 > Abs(c0) )
          {
            if( 0.00000000001 < Abs(c1) )
            {
              v = -c2 / c1; u = g * v + h; w = 1.0 - (u + v);
              if( 0.0 <= u && 0.0 <= v && 0.0 <= w )
              {
                param = (EvaluatePhongTessellation( u, v, w, positions, normals, domain ) - ray.Position )[domain] / ray.Direction[domain];
                if( Epsilon < param && param < ray.Depth - Epsilon && param < depth  )
                {
                  barycentric_coord = Vector3D(u, v, w);
                  depth             = param;
                  flag              = true;
                }
              }
            }
          }
          else
          {
            Real tmp0, tmp1;
            discriminant = (c1 * c1) - 4.0 * (c0 * c2);
            if( 0.0 <= discriminant )
            {
              FunctionSolver::Quadratic( c0, c1, c2, tmp0, tmp1 );

              // with Line 1
              v = tmp0; u = g * v + h; w = 1.0 - (u + v);
              if( 0.0 <= u && 0.0 <= v && 0.0 <= w )
              {
                param = (EvaluatePhongTessellation ( u, v, w, positions, normals, domain ) - ray.Position )[domain] / ray.Direction[domain];
                if( Epsilon < param && param < ray.Depth - Epsilon && param < depth  )
                {
                  barycentric_coord = Vector3D(u, v, w);
                  depth             = param;
                  flag              = true;
                }
              }
              // with Line 2
              v = tmp1; u = g * v + h; w = 1.0 - (u + v);
              if( 0.0 <= u && 0.0 <= v && 0.0 <= w )
              {
                param = (EvaluatePhongTessellation ( u, v, w, positions, normals, domain ) - ray.Position )[domain] / ray.Direction[domain];
                if( Epsilon < param && param < ray.Depth - Epsilon && param < depth  )
                {
                  barycentric_coord = Vector3D(u, v, w);
                  depth             = param;
                  flag              = true;
                }
              }
            }
          }
        }
      }
    }
  }
  return flag;
}
```