Computational Geometry

◊ II.2

Point in Polyhedron Testing Using Spherical Polygons

Paulo Cezar Pinto Carvalho and Paulo Roma Cavalcanti
IMPA, Instituto de Matematica Pura e Aplicada
UFRJ, Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil
pcezar,roma@visgraf.impa.br

◊ Introduction ◊

This gem presents a method based on spherical polygons to determine if a given point is inside or outside a three-dimensional polyhedron, given by its face list. This approach extends a well-known 2D technique (Haines 1994) to 3D.

In two dimensions, one can decide whether a point p is inside a simple polygon P by computing the signed angle, around p, determined by each side of P. If p is not on the boundary of P, the sum S of all such signed angles is necessarily -2π, 0 or 2π. If $S = 0$, p is exterior to P. Otherwise, p is interior to P. Usually, this method is considered to be inferior to the one which is based on counting the number of intersections with P of a ray through p. However, it deserves attention for its elegance and simplicity.

Below, it is shown how to extend the signed angle method to the 3D problem. It is assumed that P is a simple polyhedron, given by its face list, in which the faces are consistently oriented.

◊ Method of Solution ◊

First observe that the measure of the signed angle corresponding to an edge is (in the 2D case) the measure of the directed arc obtained by projecting that edge onto the unit circle whose center is the point p being tested. The arc is positive if its orientation is counterclockwise and negative otherwise. The corresponding operation in three dimensions is to project each face of the polyhedron onto a unit sphere of center p and compute the signed area of the spherical polygon thus determined. The sign is positive if the spherical polygon has counterclockwise orientation and negative otherwise (Figure 1). In analogy to the 2D case, the following holds:

Theorem 1 *: The sum S of the signed areas of the projections of all faces of the simple polyhedron P onto the unit sphere of center p is necessarily* 0, 4π *or* -4π. *If* $S = 0$,

IBM ISBN 0-12-543455-3
Macintosh ISBN 0-12-543457-X

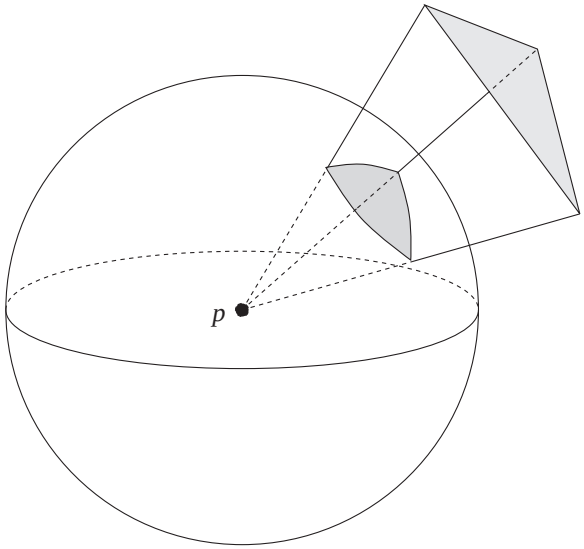

Figure 1. Projecting faces in 3D.

then p is exterior to P. Otherwise, p is interior to P.

Proof: The projections of all edges of P partition the surface of the unit sphere into a finite family $Q = \{Q_1, Q_2, ..., Q_m\}$ of spherical polygons. The projection of a face of P is a finite union of elements of Q. Each element of Q may appear in the projection of several faces and in each case its area may contribute positively or negatively to the total signed area S, depending on the face orientation. Hence, S can be expressed as $S = \sum_{i=1}^{m} \alpha_i * \text{area}(Q_i)$, where α_i is the *net contribution* of Q_i. Assume that p is interior to P and that the faces of P have counterclockwise orientation. Let us compute the contribution α_i of a spherical polygon Q_i to S. Consider a ray defined by p and an arbitrary point interior to Q_i. This ray may cross several faces of P. The first crossing goes from the inside to the outside of P (Figure 2). As a consequence, the orientation of the projection of the first face crossed is counterclockwise and the area of Q_i is counted positively. If there is another crossing, then it goes from the outside to the inside and so the corresponding face projects onto a clockwise spherical polygon. Since p is interior, the total number of crossings is odd and the ray goes from the inside to the outside once more often than it goes the other way. So, the net contribution of Q_i is positive and $\alpha_i = 1$. Since this happens for every Q_i, S is equal to the area of the sphere, which is given by 4π.

If the faces of P have clockwise orientation, then $S = -4\pi$. Finally, if p is exterior, the number of crossings is even, and the total contribution of each spherical polygon is zero, regardless of the orientation of the faces of P. Thus, in this case $S = 0$.

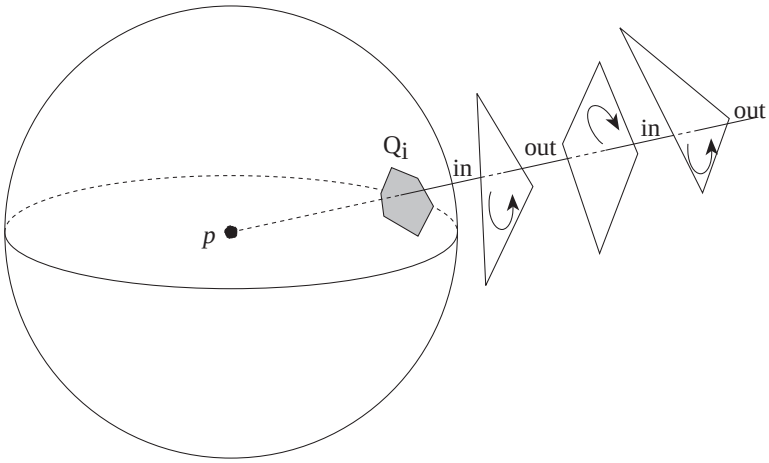

Figure 2. Crossing faces.

Computing Signed Areas of Spherical Projections of Polygons

To use the previous theorem to locate a point p with respect to a polyhedron P, it is necessary to find the signed area of the projection of each face F of P onto the unit sphere of center p. It is possible to project each vertex onto the sphere and employ the routine presented in a previous gem (Miller 1994) to compute [1] the signed area of the spherical polygon thus obtained. A more practical approach avoids the projection of faces onto the sphere. Presented below, it is based on the classical formula of Girard for the area of a spherical triangle.

According to Girard's formula (Coxeter 1961,Lines 1965,Bian 1992), the area of a spherical triangle on the unit sphere is given by $S = A + B + C - 2\pi$, where A, B and C are the (spherical) angles at each vertex, and 2π is the spherical excess. This is readily extended for spherical polygons by adjusting the excess. If a spherical $n-$gon is triangulated into $n-2$ spherical triangles, its area may be expressed by

$$S = (\sum_i A_i) - 2(n-2)\pi. \tag{1}$$

The spherical angle at a given vertex A is the angle α determined by the tangents to two great circles corresponding to the sides that cross at A. But α is also the angle formed by the planes containing those two great circles. Thus, α can be determined from the normal vectors to each of these planes, which can be computed without actually projecting the vertices onto the sphere (Figure 3).

[1] A correction to his implementation concludes this work.

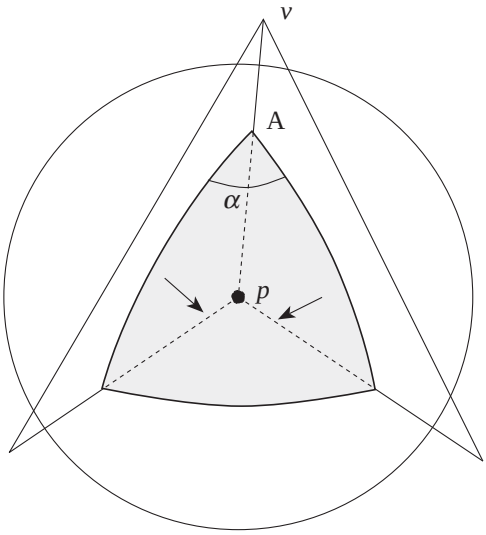

Figure 3. Computing spherical angles.

Note, however, that the corresponding angle A of the spherical polygon may be either α or $2\pi - \alpha$, depending on whether A is the projection of a convex or concave angle. This can be ascertained by computing the vector product of the two corresponding face edges and comparing the resulting vector with the normal vector to the plane. If they have the same orientation, the angle is convex and $A = \alpha$; otherwise, $A = 2\pi - \alpha$.

This procedure is executed for each vertex of the polygon, and Equation (1) then gives the area of the spherical polygon. Finally, it is necessary to find the sign to be attributed to this area. It suffices to compute the inner product of the face normal vector and the vector pv that joins the center p of the sphere to an arbitrary vertex v of the face. If this product is positive, the projection of F is counterclockwise and its signed area is positive. Otherwise, the area takes a negative value.

Code Revision for Computing the Area of a Spherical Polygon

The published routine (*op. cit.*, page 136) used to compute the area of a spherical polygon does not work in every case. The error lies in the statement

```
if (Lam2 < Lam1) Excess = - Excess;
```

appearing as the penultimate line of the final *if* statement. The method fails when the polygon crosses the 0° meridian (the case is analogous to crossing the international date line). It should be replaced by

```
    double Lam;
    Lam = (Lam2 - Lam1 > 0) ? Lam2 - Lam1 : Lam2 - Lam1 + 4*HalfPi;
    if (Lam > 2*HalfPi) Excess= -Excess;
```

With this revision in place the routine may be used to find the correct orientation of the projected face and hence the correct signed area.

♢ ANSI C Code ♢

The code given below reads the face list of a polyhedron (description of each face, consisting of the number of vertices and the coordinates of each vertex) and tests an arbitrary point for inclusion in the polyhedron. To keep it short, the code does not test whether a given point is too close to a polygon plane. In practice, should this happen, the code should check for proximity to the polygon and return *point on the boundary*.

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>

#ifndef max
#define max(a,b) ((a)>(b)?(a):(b))
#define min(a,b) ((a)<(b)?(a):(b))
#endif
#define PI 3.141592653589793324
#define GeoZeroVec(v) ((v).x = (v).y = (v).z = 0.0)
#define GeoMultVec(a,b,c) \
  do {(c).x = a*(b).x; (c).y = a*(b).y; (c).z = a*(b).z; } while (0)
#define Geo_Vet(a,b,c) \
  do {(c).x = (b).x-(a).x; (c).y = (b).y-(a).y; (c).z = (b).z-(a).z;} while (0)

typedef double Rdouble;
typedef float  Rfloat;
typedef struct _GeoPoint { Rfloat x, y, z; } GeoPoint;

/*========================  Geometrical Procedures  ====================== */

Rdouble GeoDotProd ( GeoPoint *vec0, GeoPoint *vec1 )
{
 return ( vec0->x * vec1->x + vec0->y * vec1->y + vec0->z * vec1->z );
}

void GeoCrossProd ( GeoPoint *in0, GeoPoint *in1, GeoPoint *out )
{
 out->x = (in0->y * in1->z) - (in0->z * in1->y);
 out->y = (in0->z * in1->x) - (in0->x * in1->z);
 out->z = (in0->x * in1->y) - (in0->y * in1->x);
}

Rdouble GeoTripleProd ( GeoPoint *vec0, GeoPoint *vec1, GeoPoint *vec2 )
```

```
{
 GeoPoint tmp;

 GeoCrossProd ( vec0, vec1, &tmp );
 return ( GeoDotProd( &tmp, vec2 ) );
}

Rdouble GeoVecLen ( GeoPoint *vec )
{
 return sqrt ( GeoDotProd ( vec, vec ) );
}

int GeoPolyNormal ( int n_verts, GeoPoint *verts, GeoPoint *n )
{
 int      i;
 Rfloat   n_size;
 GeoPoint v0, v1, p;

 GeoZeroVec ( *n );
 Geo_Vet ( verts[0], verts[1], v0 );
 for ( i = 2; i < n_verts; i++ )
     {
      Geo_Vet ( verts[0], verts[i], v1 );
      GeoCrossProd ( &v0, &v1, &p );
      n->x += p.x; n->y += p.y; n->z += p.z;
      v0 = v1;
     }

 n_size = GeoVecLen ( n );
 if ( n_size > 0.0 )
    {
     GeoMultVec ( 1/n_size, *n, *n );
     return 1;
    }
 else
     return 0;
}

/*=========================  geo_solid_angle  =========================*/
/*
  Calculates the solid angle given by the spherical projection of
  a 3-D plane polygon
*/

Rdouble geo_solid_angle (
        int      n_vert,  /* number of vertices */
        GeoPoint *verts,  /* vertex coordinates list */
        GeoPoint *p )     /* point to be tested */
{
 int      i;
 Rdouble  area = 0.0, ang, s, l1, l2;
 GeoPoint p1, p2, r1, a, b, n1, n2;
 GeoPoint plane;
```

```
 if ( n_vert < 3 ) return 0.0;

 GeoPolyNormal ( n_vert, verts, &plane );

 /*
    WARNING: at this point, a practical implementation should check
    whether p is too close to the polygon plane. If it is, then
    there are two possibilities:
      a) if the projection of p onto the plane is outside the
         polygon, then area zero should be returned;
      b) otherwise, p is on the polyhedron boundary.
 */

 p2 = verts[n_vert-1];  /* last vertex */
 p1 = verts[0];         /* first vertex */
 Geo_Vet ( p1, p2, a ); /* a = p2 - p1 */

 for ( i = 0; i < n_vert; i++ )
     {
      Geo_Vet(*p, p1, r1);
      p2 = verts[(i+1)%n_vert];
      Geo_Vet ( p1, p2, b );
      GeoCrossProd ( &a, &r1, &n1 );
      GeoCrossProd ( &r1, &b, &n2 );

      l1 = GeoVecLen ( &n1 );
      l2 = GeoVecLen ( &n2 );
      s  = GeoDotProd ( &n1, &n2 ) / ( l1 * l2 );
      ang = acos ( max(-1.0,min(1.0,s)) );
      s = GeoTripleProd( &b, &a, &plane );
      area += s > 0.0 ? PI - ang : PI + ang;

      GeoMultVec ( -1.0, b, a );
      p1 = p2;
     }

 area -= PI*(n_vert-2);

 return ( GeoDotProd ( &plane, &r1 ) > 0.0 ) ? -area : area;
}

/* ======================  main ========================== */

int main ( void )
{
 FILE     *f;
 char     s[32];
 int      nv, j;
 GeoPoint verts[100], p;
 Rdouble  Area =0.0;

 fprintf ( stdout, "\nFile Name: " );
```

```
 gets ( s );
 if ( (f = fopen ( s, "r" )) == NULL )
    {
     fprintf ( stdout, "Can not open the Polyhedron file \n" );
     exit ( 1 );
    }
 fprintf ( stdout, "\nPoint to be tested: " );
 fscanf( stdin, "%f %f %f", &p.x, &p.y, &p.z );

 while ( fscanf ( f, "%d", &nv ) == 1 )
       {
        for ( j = 0; j < nv; j++ )
            if ( fscanf ( f, "%f %f %f",
                  &verts[j].x, &verts[j].y, &verts[j].z ) != 3 )
               {
                fprintf ( stdout, "Invalid Polyhedron file \n" );
                exit ( 2 );
               }

        Area += geo_solid_angle ( nv, verts, &p );
       }

 fprintf ( stdout, "\n  Area = %12.4lf spherical radians.\n", Area);
 fprintf ( stdout, "\n  The point is %s",
         ( (Area > 2*PI) || (Area < -2*PI) )? "inside" : "outside" );
 fprintf ( stdout, "the given polyhedron \n" );
 return 1;
}
```

♢ Bibliography ♢

(Bian 1992) Buming Bian. Hemispherical projection of a triangle. In David Kirk, editor, *Graphics Gems III*, chapter 6.8, page 316. Academic Press, Boston, 1992.

(Coxeter 1961) H.S.M. Coxeter. *Introduction to Geometry.* John Wiley and Sons, New York, 1961.

(Haines 1994) Eric Haines. Point in polygon strategies. In Paul Heckbert, editor, *Graphics Gems IV*, pages 24–46. Academic Press, Boston, 1994.

(Lines 1965) L. Lines. *Solid Geometry.* Dover Publications, New York, 1965.

(Miller 1994) Robert D. Miller. Computing the area of a spherical polygon. In Paul Heckbert, editor, *Graphics Gems IV*, pages 132–137. Academic Press, Boston, 1994.