Chapter 1

GEOMETRIC UNCERTAINTY IN SOLID MODELING

Pierre J. Malraison
PlanetCAD, Inc.,
Boulder Colorado, USA
pierre.malraison@planetcad.com

Bill Denker
PlanetCAD, Inc.,
Boulder Colorado, USA
bill_denker@yahoo.com

Abstract In solid modeling geometric uncertainty arises in several settings. For systems which use analytic solutions when possible it is necessary to determine the type of a surface so that the correct analytic solver will be used. In passing from 3-D space data to 2-D parametric data, the algorithms used are vulnerable to geometric uncertainty – examples and solutions from a project in a commercial solid modeler [ACIS, 2001] are presented. For any solid modeler, there is a fundamental uncertainty imposed by the use of exact logic for topology together with necessarily inexact logic for geometry.

Keywords: solid modeling, data translation, healing, parametric surfaces

1. Introduction

This paper covers three types of geometric uncertainties arising in the context of geometric modeling. First, uncertainty as to the type of a surface, e.g., sphere or very un-eccentric ellipsoid. Second, uncertainty about approximations arising from the mapping between parameter space and three-space. Third, an epistemological uncertainty caused by the fact that in a computer algorithms invloving topology demand

exact answers but the underlying geometric mechanisms only operate within a given tolerance.

2. "Pure" Geometry

The classical example of geometric uncertainty in this area is the quadratic formula. In theory (or for an infinite precision computer) it provides an exact solution to finding the roots of a quadratic. In practice it becomes numerically unstable close to singular points. This section presents two other examples of this sort of phenomenon occurring in geometric modelling.

2.1 Quadrics

[Levin, 1976] describes how to draw and intersect quadric surfaces using the discriminant of the matrix for the quadric. [Hakala, et. al. 1980] discuss the geometric uncertainties arising from an actual implementation.

The technique is to represent a quadric surface as a 4×4 matrix Q. If x is a vector in homogeneous cooridnates, the quadric surface is defined by $Q\ x\ Q^T\ =\ 0$. Levin's algorithm then looks at the matrix $P + \lambda Q$ having the simplest form to draw the curves on $P \cap Q$.

The uncertainty is: how to decide if Q is *almost* a sphere; do we treat it as a sphere or an ellipsoid? In [Hakala, et. al. 1980] the issue was avoided by restricting attention to a fixed set of quadrics which did not include ellipsoids.

2.2 When is a cone too flat?

One algorithm for intersecting a cone with a plane looks at the ellipses formed by intersecting a bounding box with a cone. If the cone angle is near 90^o the radius of those ellipses can be very large. The definition of "nearness" is arbitrary - in the software we use, the test was that the cosine of the cone angle should be less than .1. This translates to about 85^o.

Figure 1: Cones

For very small values of the cosine of the cone angle it is better to treat the cone as a plane. The threshhold used for this is quite tight: $1.0e^{-10}$. This is an arbitrary choice but has proven to be adquate in practice.

2.3 Conic Sections

There are several different ways to represent ellipses, parabolas and hyperbolas, and none is stable for all conic-section curves. The classification of a conic can be quite significant, because the geometric properties and algorithms can be very different for the three types. For example, an invariant of the common representation $Ax^2+Bxy+Cy^2+Dx+Ey+F = 0$ is the *discriminant* $B^2 - 4AC$. This value is negative, zero or positive for ellipses, parabolas or hyperbolas, respectively. In the corresponding cases, the *eccentricity* is less than, equal to, or greater than 1. Analagous situations arise in the rational quadratic representation, and even if the conic were to be represented using a 3D conic and a plane. In all cases, the curve is a parabola if one real number "equals" another. The value of a floating-point tolerance will depend on the representation and the application, and choosing an appropriate value can be very difficult. Wilson [Wilson, 1987] provides a more detailed discussion.

3. Surface Inversion

This section presents examples of problems caused by geometric uncertainty that arose during the development of a geometric algorithm within a commercial solid modeler (ACIS [ACIS, 2001]). Some of the problems were expected, and some were not.

The task was to invert curves on a surface: given a parametric surface $S(u, v)$ and a three-space curve $C(t)$ lying on (or very near) the surface, create the preimage $P(t)$ of the curve in the parameter space of the surface; i.e., $S(P(t)) = C(t)$, within a given tolerance. The curve $P(t)$, whose range consists of uv positions in the domain of $S(t)$, is termed a *pcurve*.

Surface inversion is central to this algorithm: given a position xyz on the surface, find a parameter position uv such that $S(u, v) = xyz$. The xyz position may not be exactly on the surface, due to noisy data, but it should be close. In the usual case, inverting a position requires finding a good uv guess if none is provided, and performing a two-dimensional Newton-Raphson iteration to set to zero the distance from the given position to the surface normal vector.

Vectors, as well as positions, must be inverted. Having inverted a curve position $C(t)$ onto the surface, so that $S(u, v)$ corresponds to $C(t)$, the derivative of uv with respect to t is required. This amounts to writing a vector (C_t) as a linear combination of two basis vectors (S_u and S_v), i.e., find u_t and v_t such that $C_t = u_t S_u + v_t S_v$. In the usual case, inverting a vector requires solving a two-by-two linear system. In the current algorithm, the parameter-space derivative uv_t is used to find the coefficients of the two-dimensional B-spline for the pcurve.

Each Newton step for inverting a position is the same problem as inverting a vector, but there are differences in the requirements that necessitate different code for the two operations. The main difference is that a Newton step is actually used to change a uv guess value, and large steps probably represent a problem in the progress of the algorithm, but a large vector in uv space is perfectly acceptable for finding B-spline coefficients, as long as it's accurate. For this reason, system routines were sufficient for position inversions, but a special routine had to be written to invert a vector.

3.1 Expected Complications

3.1.1 Poles. A surface is singular when the surface parametric normal vanishes, i.e., $S_u \times S_v$ is the zero vector. The most common reason for this is that one of the surface derivatives goes to zero. Singularities include the poles of spheres, apices of cones, and the degenerate edge of a three-sided patch represented as a four-sided tensor-product spline.

As an example, consider a sphere, where u is the longitude, $0 <= u < 2\pi$, and v is the latitude, $-\pi/2 <= v <= \pi/2$. Then at the north pole, S_u is the degenerate derivative, and S_v runs into the pole. When inverting an xyz point corresponding to the pole, any parameter position $(u, \pi/2)$ will map to that position, and the system inversion routines will indeed return any value. For our purposes, however, the value of u is significant: the pcurve should be well behaved in parameter space. For example, an incorrect u value can cause what should be an isoparametric curve to take drastic turns near the pole. Analogously, the vector-inversion method must return the correct uv_t, so that the derivative of the pcurve is correct in parameter space.

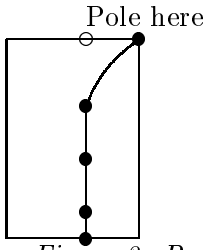

Figure 2: Parametic pole

The basic solution to this is quite simple, and almost always works: just make u the same as that of an adjacent point. The corresponding solution for vector inversion is to set the degenerate component of the parameter derivative to zero. This makes the pcurve go straight into the pole along an isoparameter line, which in real-world models is almost always the case.

3.1.2 Seams. Periodic surfaces add another complication. The sphere again provides a good example: at any point along the prime meridian, $S(0, v) == S(2\pi, v)$, and an inversion algorithm could return either 0 or 2π as the u value.

There are two cases here: the curve runs either along the seam, or across it. If the curve is the seam curve, our algorithm deals with the ambiguity by always returning the low edge of the period; at this level, there is no information to decide otherwise. If the curve runs across the seam, then we may assume that a point on the seam is either the start or end of the curve. In this case, we check whether its direction corresponds to increasing or decreasing u on the surface.

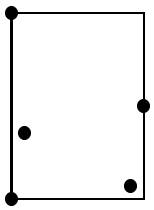

Figure 3: Curve running along a seam

3.1.3 Incorrect Convergence. It is also possible for inversion routines to converge to an incorrect solution. Because the position to be inverted does not necessarily lie on the surface, the convergence criterion is actually that xyz is on the surface normal vector, within tolerance. Therefore, the inversion algorithms can converge to a local distance maximum instead of a minimum. This was observed, for example, on a large, thin torus. A special method was written to determine

whether a uv parameter position is the proper inversion of an xyz position. It checks both whether the uv is a solution, and whether it appears to be the correct solution.

3.2 Unexpected Complications

Even the expected complications such as periodicity and singularities became problematic when dealing with real-world data.

3.2.1 Behavior Near Poles. In addition to dealing with singularities, points that are very close to true singularities also need special handling. If one derivative is extremely small compared to the other, then Newton iteration steps can be meaningless. As an example, consider inverting a position very near the pole of a sphere. The standard two-dimensional Newton step can, because of numerical noise, give large steps in u. These large steps have almost no effect on the surface position, but they can prevent the loop from settling into a solution. This behavior arises when the three-space step is closely aligned with the non-degenerate surface derivative. In that case, a tiny component in the direction of the degenerate component, which could easily be nothing more than numerical noise, can cause a macroscopic, and meaningless, change in parameter value.

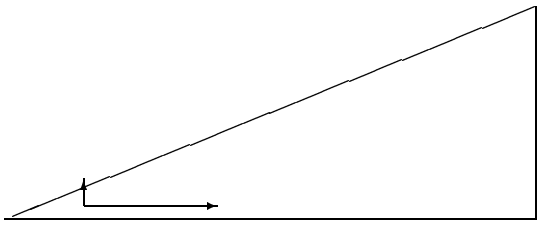

Figure 4: Parametric Degeneracy

3.2.2 Detecting Poles. Even just deciding whether a uv parameter position corresponds to a singular point has caused problems. There are two reasons for this. First, ACIS [ACIS, 2001] allows surface singularities only on domain boundaries. Because of this, system methods first check whether the parameter position corresponds to a surface parametric boundary, and if not, report that the point is nonsingular. But imported data has been known to contain internal singularities, for example where the surface passes through a pole and emerges, flipped, on the other side.

Another reason why it can be difficult to determine whether a point is singular is the age-old question, how small is a "zero" vector? Legitimate derivatives have been encountered with magnitudes as small as 10^{-7} and

as large as 10^7. This would indicate that tests for degeneracy should be relative, not absolute. The basic criterion that is appropriate for this application is the ratio of the magnitudes of the derivatives: if S_u is, say, 10,000 times as big as S_v, then this parameter position is presumably unstable and special care should be taken. Unfortunately, even that is not enough in practice: counterexamples arise in which absolute tests are required in addition to the relative tests.

For these reasons, more general methods for determining singularity had to be written.

3.2.3 Pcurve Position and Direction at Poles. The new pcurve algorithm was occasionally observed to create pcurves with very many small segments, and not particularly good accuracy, as they came into poles. The reason turned out to be that the curves were "curving" into the pole, in parameter space. As described under expected complications, this is very rare in practice. New methods were required to find the correct value – and derivative – of the degenerate parameter at the pole.

In any event, a special method was written to decide whether a parameter-space derivative is accurate. It simply checks the result in three dimensions: it compares the given three-space vector with the composition $u_t S_u + v_t S_v$. As soon as the check routine is satisfied, the result is returned. If it is not able to calculate any result that will satisfy the check routine, this fact is communicated to the calling routines, which are then able to use other information to find an acceptable value to be used in the pcurve spline.

3.2.4 Seams. As with poles, what exactly does it mean to be "on" a seam? Periodicity checks might invoke system methods to determine whether a value is within an interval. For a parameter value t in an interval $[a, b]$, system routines will return $a <= t <= b$, with the interval perhaps even expanded by some epsilon. For our application however, being within the base period of a periodic entity generally means $a <= t < b$. For reasons such as these, local versions of some system utility algorithms had to be written.

One numerical problem arose from a very tiny curve. The curve crossed a seam in a generally perpendicular direction, and was longer than the system tolerance, but so short that every point on it was within tolerance of the seam. This somewhat pathological example necessitated a change in the algorithm for determining whether a curve represents a surface seam.

3.2.5 Bad Guesses. Newton-type iterations are famous for having extremely fast convergence, if the guess is good, but also for the possibility of pathologically bad behavior if the guess is not close enough. The caller may or may not provide a guess, and if it does, the guess could be too far away. This could be because the caller simply uses a previous value as a guess, which would usually represent a good guess, but not always. For example, if the curve corresponds to an isoparametric longitude line on a sphere, steps can be very big.

A special method was written to check whether or not to use a given guess. (If no guess is given, the system inversion routines will find one themselves. That procedure is reliable, but slower, because it has less information.) It first checks the proximity of the points: if they are very close already, it returns TRUE. It then checks a trial Newton step, and if that is too large, it returns FALSE. After that, it looks at curvatures to see whether a Newton step could converge to an incorrect solution, such as a relative maximum.

3.2.6 Incorrect Convergence. Convergence to an incorrect solution could also be expected, but the exact nature of the problems and the manner of dealing with them were not obvious during the design phase, only when encountered.

Two factors conspired to make it difficult to determine whether a given uv is the correct solution: the input curves can be a substantial distance from their surfaces, and surfaces can be highly curved (such as the large, thin torus). These two possibilities make it impossible to determine a single tolerance value for the distance between $S(u, v)$ and xyz: the distance between the input curve and its surface can actually be greater than the diameter of a surface.

The convergence-checking method must take this into account. It actually uses two different tolerance values: one to check whether the given uv is a solution at all, and one to check whether it is the correct solution. For two tolerance values called `good_tol` and `bad_tol`, the algorithm is:

- IF $S(u, v)$ and xyz are themselves are within `good_tol`, return TRUE;
- IF xyz is not on surface normal to within `good_tol`, return FALSE;

  // This is a solution; check whether it's the correct one.

  // Check the surface curvature:
- IF xyz is at or beyond the center of curvature of the surface, return FALSE; // wrong convergence;

- IF the points are farther apart than `bad_tol`, return FALSE; // wrong convergence;

The two tolerance values must be determined by the caller, and could depend on the geometric situation.

4. Topology and Geometry

This is the central quandry of building boundary representations on a computer: *Topology is from Mars, Geometry is from Venus.* Topological relationships are logical – two faces either are or are not adjacent. Geometric relationships are numerical – point a is the same as point b if the distance between them is within some tolerance. This problem can be managed within the context of a single modeler but becomes more pronounced when dealing with data created externally. For example, translating a CATIA model which was created with a tolerance of 10^{-3} into ACIS [ACIS, 2001] with a tolerance of 10^{-6}.

There are two approaches to solving this problem: healing and tolerant topology. Several companies have addressed one or both of these approaches, typically with proprietary algorithms. We present here an overview.

4.1 Healing

Because of tolerance differences edges or faces which are supposed to be adjacent appear not to be. Most healing software breaks the process into two pieces: an analysis phase which finds the parts of a solid which need to be healed and a healing phase which does the healing.

Healing attempts to correct a wide range of problems. Here's a list selected from the Theorem Solutions web site [Theorem, 2001]:

- Remove a Sliver Face, with a maximum width of less than a defined value.
- Heal a whole body i.e. automatically modify edge curves and surfaces to close gaps between the edges / surfaces and vertices / curves.
- Remove a Small Edge if the length is less than a defined value.
- Remove a spike. This merges two adjacent edges of a face where the angle between the two edges and the maximum width between the two edges is small (e.g. removal of spike).
- Split edges and redefine loops etc. at a face waist position i.e. if two edges approach and are pinched other than at a mutual vertex

position within a given tolerance, split the edges if necessary at that point.

- Sew a group of unconnected faces with a given sewing tolerance.

4.2 Tolerant topology

To continue the analogy of our quotation, this approach moves topology to Venus. In other words the notion of tolerant topology is introduced, typically at the vertex and edge level.

The basic approach is to add a tolerance to the topological object:

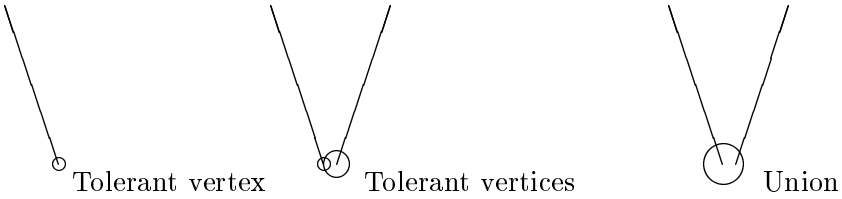

Figure 5: Tolerant topology

When two tolerant objects interact they inherit a larger tolerance. From a theoretical point of view this would seem to lead inevitably to all objects being reduced to a single tolerant vertex. In practice, tolerant vertices tend to be isolated to fairly local parts of a solid model, for example when doing variable radius blends with small radii.

References

Spatial Corporation, a Dassault Systemes, S.A. company *ACIS 3D Geometric Modeler (ACIS),* http://www.spatial.com/products/Toolkit/toolkit.htm,

D.G. Hakala, R. C. Hillyard, P. Malraison, B. E. Nourse, *Natural quadrics in mechanical design,* Proc - AUTOFACT West, Vol. 1, CAD/CAM 8, Anaheim, Calif, USA, November 17-20, 1980 Publ by SME, Dearborn, Mich, USA, 1980, 363–378

J. Levin, *A parametric algorithm for drawing pictures of solid objects composed of quadric surfaces,* Communications of the ACM, Vol. 19, No. 10, 1976, 555–563

Theorem Solutions, Ltd. *CADhealer information,* http://www.theorem.co.uk/docs/cadheinfo.htm

P. Wilson, *Conic Representations for shape descriptions,* IEEE Computer Grahics and Applications, Vol. 7, No. 4, 1987, 23–30