Knowledge-guided Computation for Robust CAD

Les A. Piegl

University of South Florida, lap@piegl.com

ABSTRACT

Robust computations have been haunting CAD system builders for decades. The common belief among researchers is that the source of the problem lies in rounded arithmetic and that new forms of computations may be necessary to ensure consistency and accuracy in commercial systems. This paper argues that, although floating point arithmetic is not perfect, the problem is not with the tool, but rather how it is used. The first part of the paper presents several techniques to increase the reliability of computations within the realm of the old-fashioned floating point computation. Many of these techniques have been applied in industrial systems, however, few of them have been publicized. The second part focuses on a knowledge-guided system mostly appropriate in a NURBS-based environment. It will be argued that the more knowledge is available about the entities to be computed on, the more intelligent decisions can be made on how to proceed with the computation to achieve a high level of reliability.

Keywords: robustness, floating point computation, knowledge-based systems.

1. INTRODUCTION

During the late sixties and early seventies emphasis for system building was on capabilities and speed. Lots of fancy mathematical tools were designed, microseconds were counted and bits and bytes of data had to be squeezed into a small memory pack. As time went on, the systems got bigger while the speed got faster and the storage got larger. Unfortunately, reliability and robustness did not get recognized until the late eighties and early nineties [4,7,9-12]. Research went on to conclude that the problem was with floating point arithmetic and when replaced with some new form of calculation, issues that had been causing problems would simply disappear in thin air. Robustness problems have their sources in several places including:

- imprecise arithmetic (rounding and error accumulation);
- careless design of algorithms (improper selection and use of techniques);
- careless implementation of algorithms (careless code design and execution);
- lack of knowledge about the data (blind computation);
- data inconsistencies (lost in translation); and
- geometric uncertainties (dirty and tricky data).

The literature focuses almost exclusively on the first problem and offers a variety of alternatives: infinite precision [10], changing the order of calculations [4], epsilon geometry [17], hidden variables [14], minimum feature size [19], symbolic reasoning [9], interval arithmetic [13,15], exact arithmetic [8,12,16,21-23], perturbation methods [5,9,17], logical modeling [18], backward error analysis [3], local robustness [20], tolerance-based geometries [6,24], lazy rational arithmetic [2] and fuzzy logic [1]. While these methods are legitimate alternatives, they address only a small facet of the robustness problem, and if used, they pose the same or similar problems once one gets into the thick of the system development. Throwing away the shovel may not be a good idea, especially if the ditch digger is not invented yet. Improving one's skill of shoveling is a better idea, and even if later the ditch digger becomes available, the shovel still comes handy (for the so called CAD beautification).

2. MOTIVATION: THE GOOD, THE BAD AND THE UGLY

The motivation for research and writing this paper came from three different sources:

The Obvious (the Good): imprecise arithmetic and error accumulation. The problem is obvious and has been around for quite some time. It is good because it is recognizable and there is a wealth of information about the nature and the source of the problem.

The Annoying (the Bad): object recognition and classification. For example, objects that are stored as B-rep, must often be recognized. A simple case is a NURBS curve that was created as a circle, however, stored as a NURBS only. When the curve is to be offset, it has to be recognized as a circle again. This is bad because it requires extra calculations, it may miss due to improper tolerances or high degree NURBS representation, and even if the recognition works, the original circle data is approximate only.
The Fatal (the Ugly): data migration and inconsistencies across platforms. A simple example is classification based on, say, line-circle intersection. System A may register a touch case whereas system B finds the entities intersecting. This is fatal because when the system slips on this tiny banana it may produce an entirely different result (think of a collaborative design environment, with several systems, where the sub-results simply do not match up).

The guiding principles for this research are two-fold: (1) remain within the realm of floating point calculations and use common sense techniques to maximize the robustness, and (2) introduce knowledge at each possible level so that intelligent decisions can be made to guide the computation. The remainder of the paper is organized as follows. Section 3 presents several methods for better floating point geometry, whereas Section 4 introduces some basic principles of a knowledge-guided system. A few conclusions are offered at the end of the paper.

3. METHODS

System builders have been tinkering with techniques to improve the accuracy of algorithms for quite some time. The techniques are mostly common sense and therefore have not found their ways into academic literature. However, since the ultimate goal is robustness, it really does not matter whether it is arrived at by rigorous derivations or by plain simple peasant thinking. This section presents several techniques ranging from the obvious to the not so obvious.

3.1 Anchoring Computations

It is a given fact the rounded arithmetic produces error, and there is absolutely nothing wrong with this as long as the error is within a certain bound. The real problem is error accumulation, i.e. an erroneous entity is passed down to produce a new result with a much larger error. The basic principle is to anchor the calculation to the original data, i.e. use the original data even if intermediate results are more natural. A beautiful example is De Casteljau's corner cutting algorithm which found its way into just about every textbook. Let `<P[0],…,P[n]>` denote the control points of a Bezier curve of degree `n`, and let `B[k][r]` denote the binomial coefficients of degree `r`, scaled by `pow(2,-r)`. Figure 1 (left) shows the popular algorithm to split the curve and to compute the midpoint on it.

```
for(i=0 to n) P[i][0]=P[i];
for(r=1 to n)
  for(i=0 to n-r)
    P[i][r]=0.5*(P[i][r-1]+P[i+1][r-1]);
  endfor
endfor
```

```
for(r=1 to n)
  for(i=0 to n-r)
    P[i][r]=0.0;
    for(j=i to i+r) P[i][r]+=B[j-i][r]*P[j];
  endfor
endfor
```

Fig. 1: De Casteljau's algorithm: (left) naïve, (right) robust.

The algorithm on the left uses the intermediate control points as the corner cutting proceeds, whereas the one on the right computes all intermediate results from the original data. The error accumulation during repeated midpoint subdivision using blind rounding is illustrated in Figure 2.

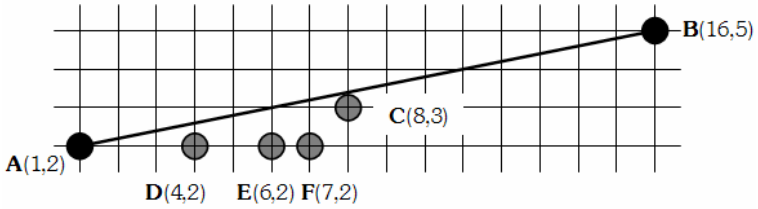

Fig. 2. Error accumulation during midpoint subdivision.

This figure clearly illustrates that blind rounding produces large errors even after a few operations. That is, had *knowledge* about the calculations been available, the point **D** would have been set to be (4,3), instead of (4,2), producing the correct result of **F**(7,3). Using the lines **AF** and **FB** for another subdivision would propagate the error to

the next level of polygons. The algorithm on the right computes all intermediate points from the original data using pre-computed entities, and hence the error at all levels is bounded round-off error, well within engineering precision. Recursive algorithms, like the one on the left in Figure 1, are most popular in research and are very tempting to implement them, however, they are not suitable for high precision computations. Anchoring the computation at the original data can avoid error accumulation, however, it does not help much if the original data was already subject to unacceptably large error.

3.2 An Eye on Numbers

Popular numerical analysis textbooks provide complete code for implementing various methods such as matrix inversion or root finding. While the core of these algorithms is correct, they cannot be assumed to produce the correct or accurate result for all data sets. These algorithms must be adjusted to account for imprecise arithmetic. Several corrections on numbers controllable by programmers can and should be employed:

- **Setting values:** known entities must be set not computed even if the algorithm includes them in the general code. Examples are curve end points and end values of know vectors.
- **Adjusting values:** computed entities may have to be readjusted regardless of their values. Examples are parameters and knot values after scaling, or numbers like `0.49999999999999999` should be set to `0.5`. Experienced programmers know where loss of accuracy can creep in, can check the values to see if there was a loss of digit or two, and the values can be adjusted for the next round of computation.
- **Pre-computing values:** frequently used entities can (should) be pre-computed and stored. An example is binomial coefficients up to a certain degree (scaled or not scaled).

Experience shows that with careful adjustments one can increase robustness quite a bit. The good news is that the adjustment is easy, the not so good news is that it requires very careful coding and an experienced programmer.

3.3 Geometric Transformations

Common transformations are normally done by matrix multiplication: form a matrix with the vertices of the entity and multiply it with the transformation matrix to get the new position of the transformed object. Again, this is simple mathematics that may not work well is practice. It not only provides an inaccurate transformed object, the geometry of the new object may not be preserved as well, e.g. right angles may not be right angles after rotation. Robustness requires a different way of thinking, e.g. the simple algebra may not be the right choice. The pseudo-code below illustrates an idea of robust transformation.

```
Get the bounding box of the object
Compute the local coordinates with respect to the local frame
Transform the local reference frame and adjust it if necessary
Reconstruct the object in the transformed reference frame
```

This method is of course not free from round-off errors, however, it produces the correct topology no matter how many times the object is transformed (the local coordinates do not change). It also serves both visualization as well as manufacturing purposes. For highly interactive visualization only the bounding box needs to be dragged and when the final position is reached, the object is reconstructed. For manufacturing the object can be positioned and locally reconstructed to guarantee topological consistency.

For repeated transformation one can do at least two things: (1) keep a transformation tree to recall the object if reverse transformations or an undo operation is needed, and (2) always compute all intermediate objects from the original one by way of transformation concatenation.

3.4 Consistent Geometric Tolerances

CAD systems are full of difficult geometric arrangements such as touch cases, overlapping objects, parallel lines, perpendicular planes, vertices incident on several planes, etc. These arrangements are checked against various tolerances, depending on how the particular configuration is expressed mathematically. For example, parallel lines are checked by cross products, perpendicular lines use dot products, incidence needs point coincidence, etc. While the various tolerances can be adjusted for particular applications, their interplay and cross-platform consistency may not be as straightforward. It is argued in this paper that *one* tolerance should be used and the conditions should be changed from the mathematical representation to the engineering construction. Some examples are shown in Figure 3. The left figure demonstrates the condition for parallelism using the offset (engineering) construction of parallel lines, i.e. it is

much easier, and more natural, to construct a parallel line via end point offsetting than via the direction vector. The middle figure explains how to measure perpendicularity using point coincidence tolerance, i.e. the end point of the line is dropped perpendicularly onto the other line and the foot point is checked against the intersection point. The right part of the figure is the familiar point incidence test. All three conditions are expressed using one tolerance requiring distance measure only. This makes comparisons simple in complex arrangements, helps with cross-platform compatibility and is natural to the engineers' working styles.

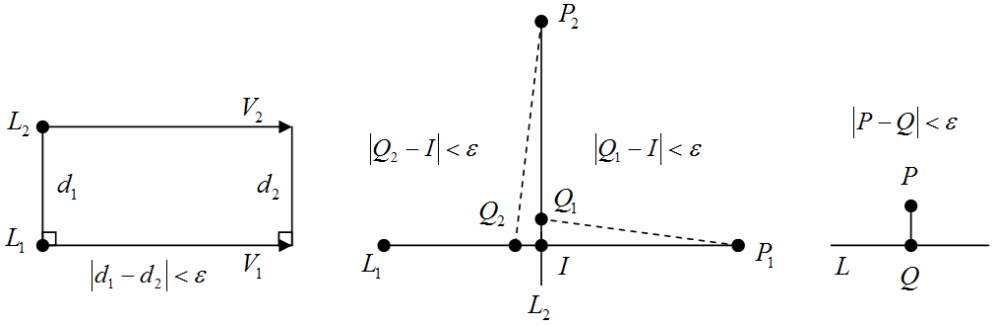

Fig. 3. Conditions for parallelism, perpendicularity and incidence.

In a later section it will be argued that conditions like the cross product condition for line parallelism is not only non-practical, it is downright incorrect from a constructional standpoint.

3.5 Proper Algorithm Design

A typical mistake of algorithm development is to read in some data and start computation on it without any kind of pre-processing. CAD data files contain all kinds of redundancies as well as incorrect data, resulting from a range of activities including careless coding, system inconsistencies and translational problems during data migration. Figure 4 illustrates the flow of a typical CAD algorithms intended to account for various pitfalls. A few notes are in order:

- **Error checking:** not everything can and should be checked. Entities entering the system the first time must be checked as well as those that may be subject to error.
- **Validation:** all entities generated outside the system must be validated, e.g. NURBS curves read from an IGES file should be validated against the system in use.
- **Cleaning:** dirty data must be cleaned. Dirty means different things in different context, e.g. knots with multiplicity greater than the degree or edges that are of no use.
- **Purging:** excess data or redundant data must be removed, unused space must be returned to the operating system, and data containing unessential elements must be reduced.
- **Computation:** no computation should ever be done on totally unknown data, i.e. the algorithm must gather intelligence on the data before deciding on how to compute. The output should be created with the input of another program in mind.

Problems of robustness go deep into algorithm design and no matter how good the underlying methods are if the implementation is done in a substandard manner, robustness will never be achieved!

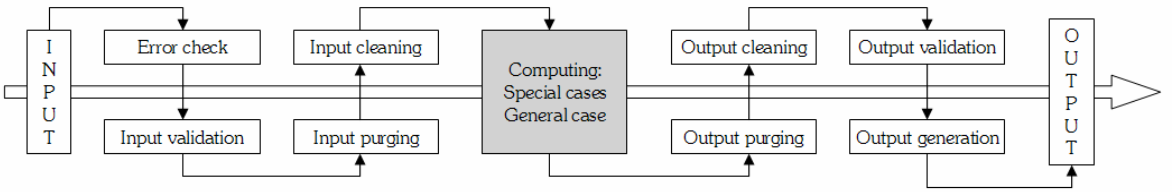

Fig. 4. Major components of robust algorithm design.

3.6 Situational Computing

Singularities are part of CAD methods, e.g. engineers deliberately create situations that are not for the textbook algorithms. These situations cannot be removed by, say, perturbation methods because they are part of the design. On the other hand, the general methods do not work well or do not work at all on these special cases. The only way these

cases can be handled is by: (1) identifying the special situation by either computation or by taking the information from a knowledge base, and (2) choosing the appropriate technique for each case. An example is surface-surface intersection. When two cylinders of equal radii intersect so that their axes intersect as well, the intersection curve consists of two ellipses. The general method would almost certainly miss the point where the ellipses intersect [13] (and of course would produce the intersection curve as a general curve). This is, of course, unacceptable, however, the engineering solution is not interval arithmetic [13], but rather identifying the case and writing special code that handles it precisely. It is well known in software engineering that designing large software systems requires decomposition. It is not quite as well known that each algorithm, even the simpler one, requires the same decomposition to achieve what we call *situational computing*, i.e. for each distinct situation a different kind of technique is appropriate. Figure 5 illustrates the concept using the example of line-line intersection.

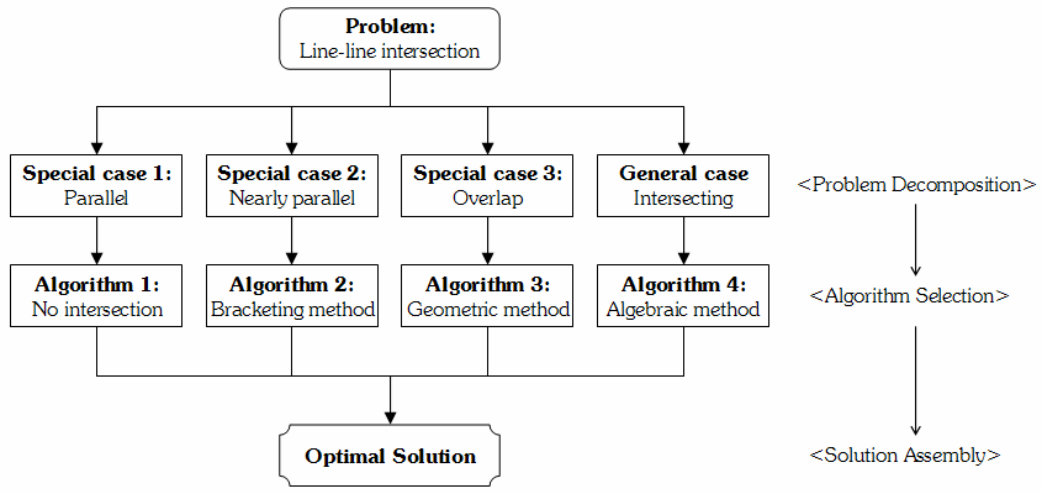

Fig. 5. Situational computing: line-line intersection.

In general, the method involves decomposition, algorithm selection and solution assembly. The decomposition is done in order to guarantee the optimal solution, where optimal means:

- most accurate, e.g. special quadric intersectors are more accurate than general ones;
- fastest, e.g. special case intersectors are real-time whereas general ones may take some time;
- most reliable, e.g. the overlap case must be handled geometrically for best reliability; and
- easy to maintain, i.e. special code can be well designed and tested requiring little or no adjustments.

For each problem there can be a myriad of situations so the question is what situations are of importance:

- special cases, e.g. line-circle touch;
- difficult cases, e.g. intersecting nearly parallel lines;
- inefficient cases, e.g. large and noisy point sets;
- singular cases, e.g. interesting overlapping curves;
- inaccurate cases, e.g. approximating unevenly spaced points;
- most frequently used cases, i.e. these cases require the most amount of optimization; and
- general case, i.e. the textbook example.

Situational computing has its pros and cons. On the positive side one has accuracy, reliability, speed and better maintenance. The negatives are longer development time, much larger code side and that the situations must be recognized which requires time and expertise. Also, special cases may have a combinatorial explosion in some applications like constraint solving.

3.7 Work Space

In classical geometry objects are infinite and occupy a universal space. In contrast, engineering objects are finite and are in a certain space, e.g. the room of the machine shop. The knowledge-guided system under proposal makes the following assumptions:

- all objects are finite, e.g. the plane cannot be defined by a point and a normal, it must be defined by a closed polygon and a normal;
- all objects are in work spaces;
- all computations are valid with respect to one or more works spaces only; and
- all relationships are valid in a work space.

Figure 6 shows an important example: two lines occupy two different work spaces. In the smaller work space they are declared parallel, whereas in the larger one they are not. This example clearly illustrates that the geometric definition of parallelism (the magnitude of the cross product is less than a tolerance) is completely useless. By this definition the two lines are parallel in *any* space, however, from a constructional point of view they are not. For example, to check if the fender of the car is parallel to the front end, one measures the distances at the left and right ends to see if the discrepancy is within tolerance. Also, building parallel objects is much easier in a smaller space than in a larger one, i.e. keeping the lines parallel in a short distance is easier than for a longer distance.

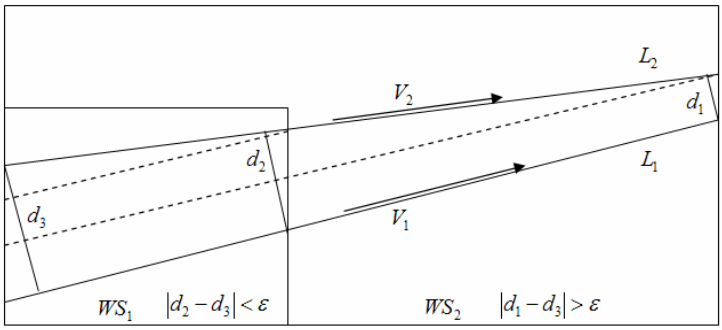

Fig. 6. Classification of lines with respect to two work spaces.

This example clearly illustrates that CAD cannot simply be handled with classical methods without modifications. The parallelism relationship in the above example changes from work space to work space and hence the outputs of algorithms may not be the same. Therefore, the proposed system associates each object with at least one work space: each relationship is taken from the appropriate work space and computations are performed in each work space separately.

4. KNOWLEDGE-GUIDED COMPUTATION

To assist in improving the robustness of CAD systems, this paper proposes a knowledge-guided system. The system uses information about the entities, their constructions and the existing relationships among them. The gathered knowledge is used to (1) reason about the objects, (2) understand their relationships, (3) establish important cases for situational computing, (4) reconstruct objects if necessary, and (5) pass the information to other systems. The knowledge is represented in two parts, entities and relationships, as follows:

E = <ID; type; origin; definition; work space>
R = <list of entities; relationship; parameters; work space>

Some examples are listed below:

```
E_1 (point) = <ID; near; sampling; x, y, z; WS1>
E_2 (line)  = <ID; unknown; input; Ps, Pe; WS1>
E_3 (curve) = <ID; spline; interpolation; control points, knots; WS2>

R_1 = <line1, line2; parallel; 0.000001; WS1>
R_2 = <point, line; incidence; 0.00001; WS1>
R_3 = <curve1, curve2; offset; distance, tolerance; WS2>
```

The initial knowledge may come from the following sources:

- construction, e.g. a curve is generated to interpolate a set of points;

- geometric queries, e.g. are two lines parallel?; and
- logical inference, e.g. using the rule of associativity for parallelism.

The first two sources are straightforward, however, the result of the logical inference must be verified numerically using the accepted condition for a given property. The inference must also consider that objects may lie in different work spaces and the traditional logical rules may or may not be applied.

4.1 Building Knowledge Bases

Knowledge bases require entities, stored relationships, parameters and work spaces. Each system must decide on what kind and how many of these items to process. The NURBS-based module this system was designed for has, as a minimum, the following relationships:

- **Position:** incident, coincident, collinear, co-planar, co-circular, overlapping, etc.
- **Geometry:** parallel, perpendicular, tangential, etc.
- **Construction:** interpolation, approximation, offset, styling, advanced surfacing, etc.

A sample of various entities, their definitions and possible relationships, are listed in Table 1 below.

Entity	Type	Origin	Definition	Relationships
Point	None, Through Near	Unknown, Input, Definition, Intersection Sampling	Point Coordinates	Incident, Collinear, Co-planar, Co-circular, Interpolating, Approximating
Line	None, Through Near	Unknown, Input, Definition, Intersection Fitting	Start and End Points	Parallel, Perpendicular, Incident, Overlap, Tangent, Approximating
Arc	Circle, Conic	Unknown, Input, Definition, Intersection Fitting	Center, Radii, Sweep Angles	Incident, Tangent, Co-circular, Approximating
Curve	Line, Circle Conic, Spline	Unknown, Input, Definition, Intersection Fitting, Offset, Styling	Control Points and Knots	Incident, Tangent, Overlap, Interpolating, Approximating, Offset, Styling
Plane	None, Through Near	Unknown, Input, Definition, Fitting	Closed Polygon and Normal Vector	Parallel, Perpendicular, Incident, Overlap, Tangent, Approximating
Quadric	Plane, Cone Cylinder, Sphere	Unknown, Input, Definition, Fitting	Center, Radii, Angles, Axes, Heights	Incident, Tangent, Co-spherical, Approximating
Surface	Plane, Sphere Quadric, Spline	Unknown, Input, Definition, Fitting, Offset, Styling, Advanced surfacing	Control Points and Knots	Incident, Tangent, Overlap, Interpolating, Approximating, Offset, Styling, Advanced Construction
Volume	Block, Sphere Quadric, Spline	Unknown, Input, Definition, Fitting, Styling, Construction	Control Points and Knots	Incident, Tangent, Offset, Interpolating, Approximating, Styling, Advanced Construction

Tab. 1. Some basic entities and their relationships.

The implementation of entities is fairly straightforward; the type and the origin can be enumerated and tagged as data types. The entity class or structure can then be built from the components of a unique ID, the entity type, the origin and all the numerical data necessary to define the entity. The entities built this way provide part of the knowledge base, e.g. points may be known as "near" points coming from "sampling", i.e. these points are most probably part of an approximation procedure. What is left is to record the relevant relationships, e.g. if a curve is used to approximate the "near" points, the point and the curve are in an "approximating" relationship. Of the many possibilities to record such relationships, a minimal model was chosen for its simplicity and expandability. Using the line entity, Figure 7 illustrates the relationship knowledge base for one line and all other entities that are of importance. The following notes are in order:

- Each line that is to be recorded has a header with the unique line ID and a reference to the work space where the line resides and where the relationships are valid.
- All entities with recorded relationships are listed in separate lists with the relevant parameters. For example, all lines that overlap the head line are given in a list whose nodes contain references to the lines, the ID's of the lines and the tolerances used to check for overlap. If a curve tangency is recorded, it is necessary to store a reference to the curve, the curve's ID, the tolerance, the parameter where the tangency occurs, and the curve's current parameter range (in case it will be reparametrised later).

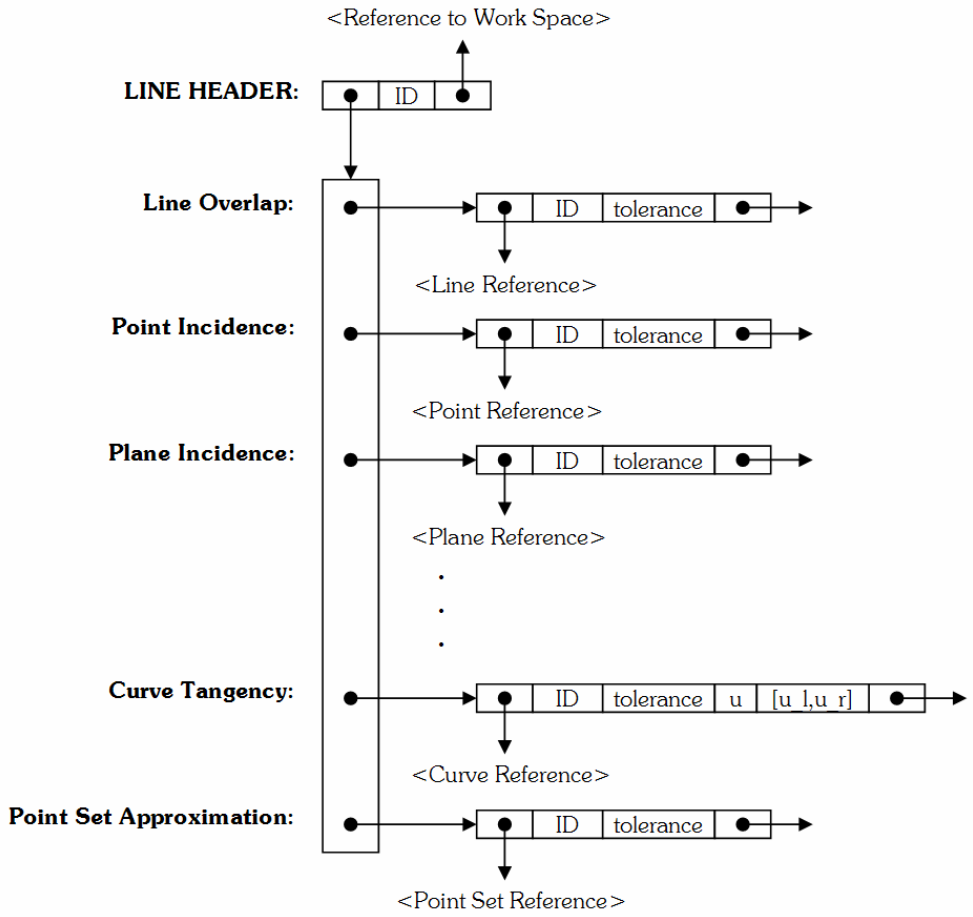

Fig. 7. Knowledge base of line entity.

Theoretically, all relationships may have a list associated with the given line. However, for all practical purposes only the most needed, or most error prone, relationships need to be stored. For example, if a fairly uniform data set is given, then the computation of the curve interpolating the data is fairly robust and the relationship between the curve and the point set may not need to be stored. However, if the parametrization of the data plays a crucial role, then curve-point set relationship needs to be recorded, along with the parametrization function, in case the interpolation must be redone in the receiving system.

The structure of the complete knowledge base is shown in Figure 8. Each entity has a reference to a list where all important entities are stored whose relationships are to be recorded. For example, in Figure 8 there are three lines whose relationships are important. These lines are part of a list where references, ID's and work spaces are accessible. Then they all become headers for another list where the references, the ID's and the parameters of the recorded relationships are stored, as illustrated in Figure 7. The advantages of such an arrangement are: (1) it is very easy to build, (2) adding or deleting entities as well as relationships is straightforward, and (3) it requires a very small space as

only references of large data are stored along with a few parameters. The disadvantage is that the relationships are predefined and so if a new one is to be stored, then the system has to be recompiled.

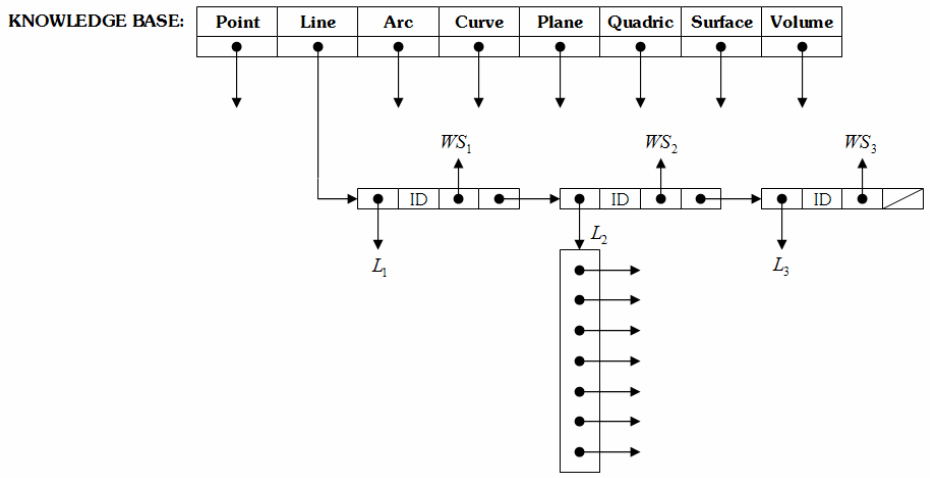

Fig. 8. Overall structure of the knowledge base.

4.2 Using the Knowledge Base

There are surprisingly many ways the knowledge base can be used in a CAD system. Below a few examples are given of the most obvious applications:

- **Geometric queries.** Queries such as find "all offset curves given a base curve", or "find all planes perpendicular to a given line", can be answered by simply traversing the lists of the "offset" and "perpendicular" relationships. Absolutely no computation is required!
- **Geometric interrogations.** Often times one needs to know as much as possible about entities. For example, given a curve, one needs to know its type, its origin, its defining data, and how the curve was arrived at. Or, given an offset curve, it is useful to know what is the base curve, the offset distance, and what tolerance was used to approximate the offset.
- **Situational computing.** In order to find the best algorithm, one needs to know how the entities are situated with respect to the work space as well as with respect to one another. For example, for the line-line intersector the knowledge base records if they are parallel or overlapping (one can also create a relationship called "nearly parallel").
- **Consistency checking.** Consistency becomes an important issue in communication between various modules, ranging from algorithm to algorithm to system to system. For example, incidence may be measured against different tolerances in different systems. With the proposed system incidence becomes more than just the matter of tolerancing, it is also the matter of knowledge, or information bestowed upon the entities. Relationships such as incidences and parametrizations can be used consistently if information is made available in each system.
- **Data migration.** Moving data around with only their numerical values has been causing serious problems for decades. For example, lines that are declared parallel in one system may be classified intersecting in another, producing different results. With the proper knowledge tagged to the data, migration can be done safely ensuring that the same result is produced in any system the data is used.
- **Model repair.** Even if knowledge is passed around, models may have to be repaired by regenerating parts of them due to differences in local requirements. For example, surfaces fit to a point cloud are trimmed and their intersection curves are computed via approximating a set of intersection points. When the model is passed to another system, a gap between two surfaces may be detected due to the different tolerance requirements. If only the B-rep is available, the engineer is out of luck and the model has to be patched up. However, passing the knowledge base along, one has access to the underlying surfaces, the point set as well as the tolerance used. This allows the engineer to reconstruct the intersection using more points and/or a tighter tolerance.

4.3 Knowledge Update

A critical question is how to update the knowledge base, or is it possible to update the system at all? Here is an example. Two lines have been recorded as parallel with respect to a tolerance. A new process finds the information in the knowledge base, however, the tolerance used is not sufficiently tight. If it checks the lines with respect to a tighter tolerance, there are two possibilities: (1) the lines are still parallel, and (2) they are not. The question is what to do with this new piece of knowledge? There are a few possibilities:

- leave the knowledge as is and process the line locally based on local requirements;
- overwrite the knowledge with the new information; or
- generate a new entity (line) with respect to the new requirements.

Changing the data in a knowledge base causes relationships established before not to be valid any more, and hence it should not be done. For example, process A recorded no intersection between the lines, whereas process B will have one based on a different tolerance used. This example underlines the important of consistent tolerancing discussed above, i.e. all geometric relationships (parallelism, perpendicularity, etc.) are measured with respect to *one* global tolerance. Now, this may have to be defined differently from application to application, however, in each application the tolerance should be global to avoid inconsistencies. Knowledge update is provably the most delicate part of knowledge-guided computation requiring deeper investigations, beyond the limitation of this paper.

4.4 Knowledge Deduction

Based on the laws of logic and that of geometric algebra, knowledge can be deduced from existing data. For example, if line 1 is parallel to line 2, and if line 2 is parallel to line 3, then by the associativity rule line 1 is parallel to line 3. In theory it is correct, however, before the new piece of knowledge is created, one has to verify that this new relationship is correct with respect to the tolerance and work space requirements.

4.5 Life Span of Knowledge

How long should a knowledge base, used for robust computation, be kept? Ideally, it should be kept as long as the data is used. As a minimum, it should be kept until all computations have been completed. However, it is recommended to keep it for the life of the data, i.e. it becomes part of the legacy system.

5. CONCLUSIONS

This paper presented an attempt to improve the efficiency and reliability of computations used in CAD systems based on the NURBS representation. A knowledge-guided system has been proposed to assist the programmer as well as the processes to make intelligent decisions and to perform corrections to account for round off errors. It has been argued that the problem of robustness cannot be pinned down to a simple matter of better arithmetic. The overall problem is very complex requiring careful work at all levels: basic arithmetic, careful choice of techniques as well as all aspects of system design. Most of the problems are inherited from the early systems and it might take another software system generation until robustness is achieved to a desired level. The non-compatible CAD systems fighting for market share and the legacy data requiring the use of outdated and non-supported systems add another level of complexity to the already chaotic situation. Standardization is a possibility, however, when one has just as many standards as CAD systems, standards may actually make things worse rather than better. While the jury is still out, the best one can do is to use intelligence (knowledge) to improve reliability as much as possible.

Acknowledgements. The author is indebted to Professors Christoph Hoffmann and Weiyin Ma and to Dr. Wayne Tiller for their many valuable comments on the first draft of this paper. This work was supported by the US National Science Foundation under grant No. DMI-0200385, awarded to the University of South Florida.

6. REFERENCES

[1] Baker, S. M., Towards a topology for computational geometry, *Computer-Aided Design*, Vol. 27, No. 4, 1995, pp 311-318.

[2] Benouamer, M., Michelucci, D. and Peroche, B., Error free boundary evaluation based on a lazy rational arithmetic: a detailed implementation, *Computer-Aided Design*, Vol. 26, No. 6, 1994, pp 403-416.

[3] Desaulniers, H. and Stewart, N. F., Robustness of numerical methods in geometric computation when problem data is uncertain, *Computer-Aided Design*, Vol. 25, No. 9, 1993, pp 539-545.

[4] Dobkin, D. and Silver, D., Applied computational geometry: towards robust solutions of basic problems, *Journal of Computer and System Sciences*, Vol. 40, 1990, pp 70-87.

[5] Edelsbrunner, H. and Mucke, E. P., Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms, *Proc. ACM Symposium on Computational Geometry*, June 1988, pp 118-133.
[6] Fang, S., Bruderlin, B. and Zhu, X., Robustness in solid modeling: a tolerance-based intuitionistic approach, *Computer-Aided Design*, Vol. 25, No. 9, 1993, pp 567-576.
[7] Forrest, A. R., Geometric computing environments: computational geometry meets software engineering, *Proc. NATO Advanced Study Institute on TFCG and CAD*, Italy, July 1987.
[8] Fortune, S., Polyhedral modeling with exact arithmetic, *Proc. Symposium on Computational Geometry*, 1993, pp 163-172.
[9] Hoffmann, C. M., Hopcroft, J. and Karasick, M., Towards implementing robust geometric computations, *Proc. Symposium on Computational Geometry*, 1988, pp 106-117.
[10] Hoffmann, C. M., The problems of accuracy and robustness in geometric computation, *IEEE Computer*, Vol. 22, No. 3, 1989, pp 31-42.
[11] Hoffmann, C. M., *Geometric and Solid Modeling: An Introduction*, Morgan Kaufmann, 1989.
[12] Hoffmann, C. M., Robustness in geometric computations, *Journal of Computing and Information Science in Engineering*, Vol. 1, No. 2, 2001, pp 143-156.
[13] Hu, C.-Y., N. M. Patrikalakis and Ye, X., Robust interval solid modeling Part I: representation, *Computer-Aided Design*, Vol. 28, No. 10, 1996, pp 807-817.
[14] Milenkovic, V., Robust polygon modeling, *Computer-Aided Design*, Vol. 25, No. 9, 1993, pp 546-566.
[15] Mudur, S. and Koparkar, P., Interval methods for processing geometric objects, *IEEE Computer Graphics and Applications*, Vol. 1, February, 1984, pp 7-17.
[16] Ottmann, T., Thiemt, G. and Ullrich, C., Numerical stability of geometric algorithms, *Proc. ACM Symposium on Computational Geometry*, June 1987, pp 119-125.
[17] Salesin, D. Stolfi, J. and Guibas, L., Epsilon geometry: building robust algorithms from imprecise calculations, *Proc. ACM Symposium on Computational Geometry*, 1989, pp 208-217.
[18] Schreck, P., Robustness in CAD geometric constructions, *Proc. 5th International Conference IV 2001*, London, UK, 2001, pp 111-116.
[19] Segal, M. and Sequin, C. H., Consistent calculations for solids modeling, *Proc. ACM Symposium on Computational Geometry*, 1985, pp 29-38.
[20] Stewart, A. J., Local robustness and its application to polyhedral intersection, *J. Computational Geometry and Applications*, Vol. 4, No. 1, 1994, pp 87-118.
[21] Sugihara, K. and Iri, M., A solid modeling system free from topological inconsistency, *Journal of Information Processing*, Vol. 12, 1989, pp 380-393.
[22] Yu, J., *Exact arithmetic solid modeling*, PhD Thesis, Purdue University, 1992.
[23] Yap, C.-K., Towards exact geometric computation, *Computational Geometry*, Vol. 7, 1997, pp 3-23.
[24] Wallner, J., Krasauskas, R. and Pottmann, H., Error propagation in geometric constructions, *Computer-Aided Design*, Vol. 32, No. 11, 2000, pp 631-641.