*Computer-Aided Design and Applications*
© 2009 CAD Solutions, LLC
http://www.cadanda.com

# Fault-tolerant Computing in a Knowledge-guided NURBS Environment

Les A. Piegl[1], Khairan Rajab[1], Volha Smarodzinava[1] and Kimon P. Valavanis[2]

[1]University of South Florida, {lpiegl,khairanr,olyagrove}@gmail.com
[2]University of Denver, kvalavan@du.edu

**ABSTRACT**

This paper presents a framework and a methodology to increase the robustness of CAD systems based on the *de facto* standard of NURBS. Contrary to popular belief, the solution to robustness does not lie in either the mathematical tool used in computations or in the form of arithmetic chosen, but rather, it depends on how much and what types of information (knowledge) is available and how cleverly the system makes use of this information. In other words, the issue of robustness has a lot more to do with software than with theory, although some basic theories are also important. This paper argues that the problem is related, by a large margin, to (1) the maze of data formats and representations, (2) standards that are not fully implemented and followed, and (3) data bases that contain models with cracks, overlaps, etc, that cannot be healed properly. A three-tier architecture is proposed consisting of a lower tier (algorithms), a middle tier (data preparation) and an upper tier (knowledge acquisition). This three-tier approach improves the reliability of CAD techniques beyond expectations and when combined with *n*-version programming techniques, it allows the design of a complete fault-tolerant computing model. The main pieces of this model are failure analysis, knowledge-base update and a dynamically updated algorithm vault.

**Keywords:** knowledge engineering, intelligent computing, robustness, NURBS.
**DOI:** 10.3722/cadaps.2009.809-823

## 1. THE MAZE OF DATA FORMATS

One of the fundamental reasons that account for the lack of robustness across CAD platforms is the large variety of data formats that are in use today, Table 1 and Figure 1. CAD vendors guard their proprietary data formats with iron fists and would expect the rest of the design community to adhere to their data formats. Supply chain companies that use several design packages, have the hopeless task of converting one data format into the other in order to complete a complicated model that is often the result of 100 or more contributors. Converting different data formats is inherently a risky process, however, on closer examination it turns out to be an impossible task.

The first major bottleneck is representation inconsistency, e.g. the same object is stored in different formats in two CAD systems. A simple example is the circle that is stored as a degree two rational curve is one system and degree 5 in another. Going from degree two to five can be made precisely (with the disadvantage of introducing multiple knots into the fifth degree circle which defies the very purpose of having it in the first place), however, reducing the degree from five to two is an imprecise task that introduces inaccuracies responsible for failures downstream.

| | |
|---|---|
| ART | ArtCAM |
| ASC | BRL-CAD geometry file |
| ASM | Solidedge assembly |
| CCM | CopyCAD model |
| CAD | CADst |
| CATx | CATIA (drawing, part, assembly and manufacturing document) |
| DWG | AutoCAD and Open Design Alliance applications |
| DGN | MicroStation design file |
| DGK | Delcam geometry |
| FM | FeatureCAM part file |
| GRB | T-FLEX CAD file |
| ICD | IronCAD 2D file |
| IGES | Initial Graphics Exchange Specification |
| ISFF | Integraph standard file format for MicroStation |
| PRT | Unigraphics,Pro/ENGINEER |
| SKP | SketchUp model |
| RWS | Rhino work session |
| SLDPRT | SolidWorks part model |
| STEP | Standard for the Exchange of Product model data |

Tab. 1: A selection of CAD file formats.

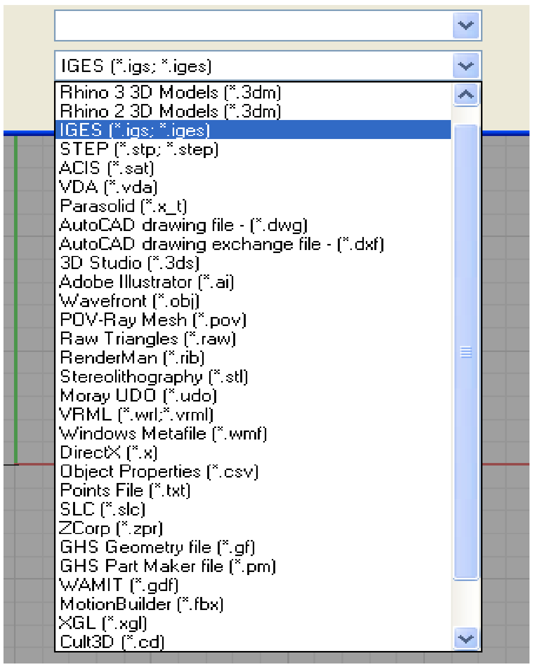

Fig. 1: File formats to save Rhino models.

The second major problem is topological inconsistency. Table 2 below summarizes how three CAD systems represent the cylinder [19]. The inconsistency is quite evident giving rise to difficulties when converting one format into another. For example, one system, Pro/Engineer, generates the cylindrical surface as two open patches, whereas the other, IDEAS, produces a closed surface with a seam. Since open and closed surfaces are handled inherently differently at the software level, e.g. intersection curves crossing the seam must be split, topology differences may give rise to geometry problems in addition to confusing the systems with the number of vertices, edges and faces.

| | Unigraphics | IDEAS | Pro/Engineer |
|---|---|---|---|
| Number of faces | 3 | 3 | 4 |
| Number of edges | 2 | 3 | 6 |
| Number of vertices | 2 | 2 | 4 |
| Curve type | Circle | B-spline | Circle |
| Surface type | Cylinder | B-spline | Cylinder |

Tab. 2: A variety of cylinder representations.

The third major problem lies with the levels of accuracy by which entities are defined in different systems. The most obvious discrepancy comes from approximating entities, such as offset and intersection curves, up to different tolerances. While this is a computational issue, the generated curves must be converted to satisfy internal requirements. A more annoying problem is due to the use of different data formats, e.g. circle conversion from degree five to degree two requires approximation introducing two major flaws: (1) the inherently precise circle is now inaccurate, and (2) the four times continuously differentiable degree five circle has become either a non-rational approximation or a discontinuous curve in homogeneous space.

Finally, different data formats can give rise to data growth or even explosion. A simple example illustrates the problem. Assume that one of the cross-section curves is a degree five circle that needs to be approximated in order to form a lofted surface. If the tolerance requirement is high, e.g. $10^{-7}$, then the approximation produces over 250 control points out of the original 6 [14]! This explosion of control points for one curve is multiplied by the lofting process that needs to make all section curves compatible, giving rise to tens or hundreds of thousands of control points, 95+ percent of which is theoretically unnecessary.

## 2. STANDARDS DO NOT WORK

The original idea behind standards, or neutral data formats, has been to allow data exchange between systems while maintaining proprietary data formats. The process of data exchange would work as shown in Figure 2. The model in system A is converted to a neutral format via a writer, e.g. its own IGES writer. The IGES file would then be read into system B via its IGES reader and converted into system B's internal data representation.

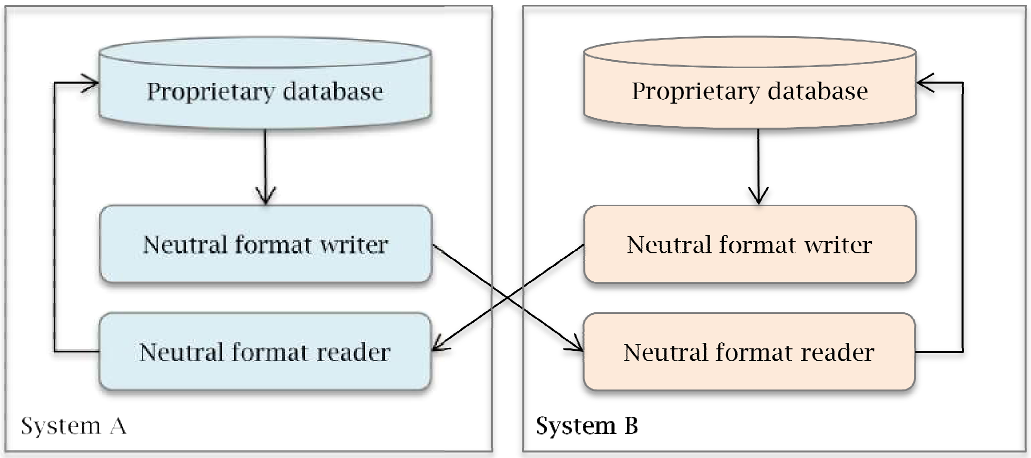

Fig. 2: Data exchange between two CAD systems.

It is like translating a Russian novel to Korean by first translating the Russian version to English, and then the English version to Korean. While in theory it seems feasible, but because translations almost always introduce inaccuracies and loss of information, the basic idea is doomed to fail. Indeed, Figures 3 to 7 illustrate what happens to a simple design if it is passed around via IGES or STEP.

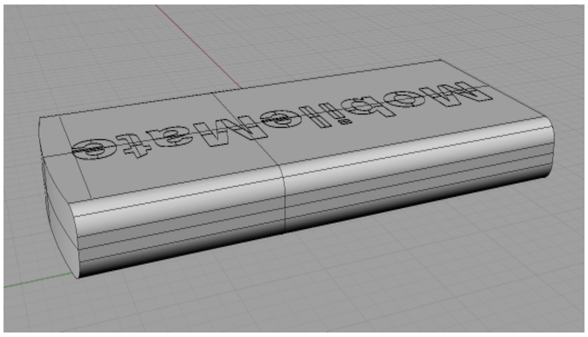

Fig. 3: Original Rhino design.

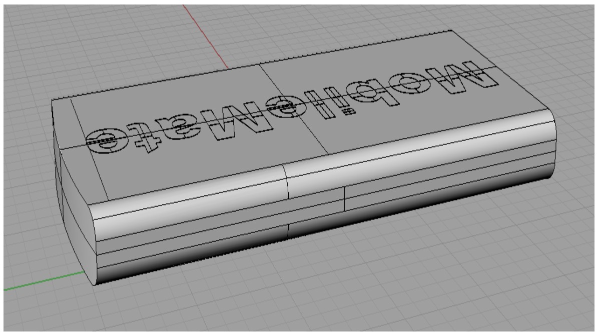

Fig. 4: Design exported to CATIA IGES.

The first conversion is shown in Figure 4 where the original design, Figure 3, is converted to CATIA IGES. The topology is altered, which is a minor problem, however, closer examination reveals that there are large cracks between the surfaces on the left end, affecting at least three surface patches, Figure 5. Exporting to STEP, Figure 6, or Unigraphics, Figure 7, produces similar results by altering the original topology.

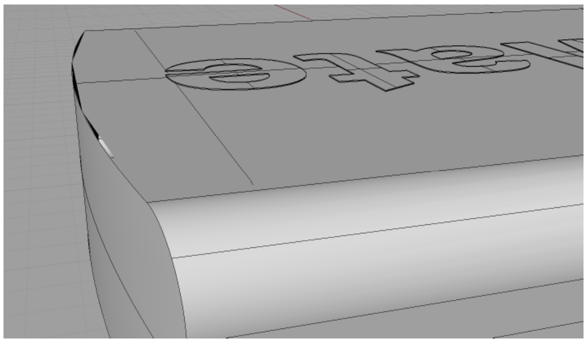

Fig. 5: Cracks in CATIA IGES.

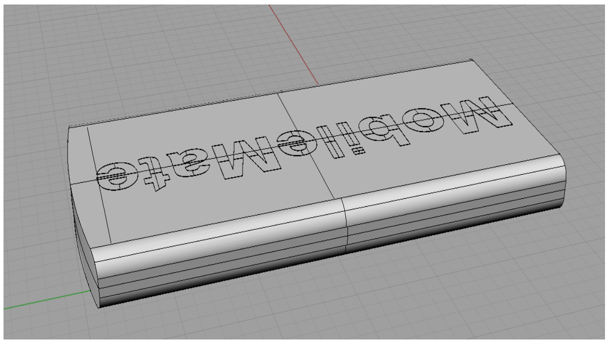

Fig. 6: Design exported to STEP.

There are several fundamental problems with standards. First and foremost, it is assumed that the original model is fully convertible to the chosen standard format, i.e. the Russian novel is translatable to English without any loss of meanings, which is of course not correct. Similarly, it is assumed that the standard data is then translatable into the format of the receiving system, e.g. the English version of the Russian novel is seamlessly translatable to Korean. The two-step process produces inaccuracies or loss of information at least in one step (standard to the receiving system), but more often than not, trouble surfaces at either end of the translator.

The second source of errors comes from the lack of version control, e.g. one system implements IGES version 5.3 whereas the other has only version 5.2. Writing good translators is a difficult task, in addition to being remotely related to productivity and hence receives lower priority.

Thirdly, systems hardly ever implement translators in their entirety, i.e. only subsets, the ones that are deemed most important, receive full implementation. The result of this is "selective standardization", i.e. the scope of data exchangeability of all vendors do not overlap, Figure 8, and hence full conversion becomes a dream. It is like creating a color photo on two different printers that have non-overlapping color spaces. While the prints may look similar, they are not entirely the same, showing the effect of approximate color mapping.

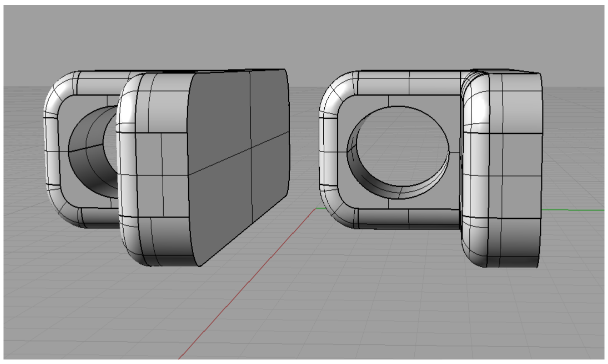
Fig. 7: Design exported to Unigraphics.

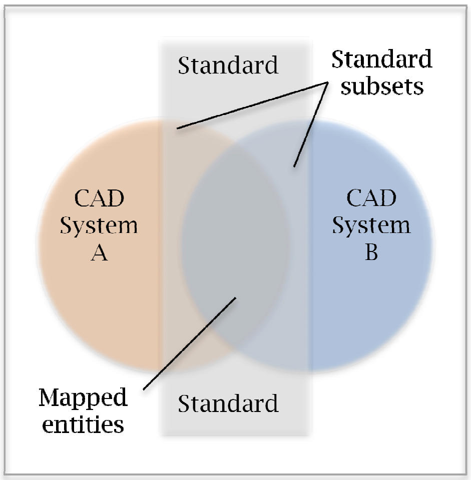

Fig. 8: Partial implementation of standards.

Yet another annoying problem with standards is due to the way various entities are tagged. For example, a curve may be tagged as a NURBS curve/surface (IGES 126/128) or a spline curve/surface (IGES 112/114). The reason for this discrepancy lies in the preferences by which companies implement standards, and that standards allow different mathematical representations for the same geometry, e.g. cylinder versus NURBS surface.

## 3. DATA HEALING MAY NOT WORK EITHER

If data formats are confusing and incompatible, and translation to neutral file formats does not always work, then how one can exchange data with CAD systems? One obvious choice is repair, i.e. if a flawless model is not obtainable from, say, an IGES file, then get the best there is and fix it up to internal requirements [2,3,6].

Fixing CAD models entails the following: (1) condition for validity, (2) meaningful tolerances, and (3) model checking and repair. Conditions for validity can be set up to guarantee correct models at least theoretically. For example, one may impose the following conditions:

- Each edge belongs to two faces.
- Each vertex is incident at a single cycle of edges and faces.
- Faces intersect only at common edges and vertices.

These are fair conditions, however, they work only with two-manifold objects.

The issue of tolerances is a difficult one [17]. At first glance it may seem that one gives a small tolerance to check the above conditions, and the object is easily repaired after some modifications. Unfortunately, there is a maze of tolerances defined in the ISO and ASME standards [1,8]. There is a single dimensional tolerance and a large set of geometric tolerances:

- Form (straightness, flatness, circularity and cylindricity).
- Operation (parallelism, perpendicularity and angularity).
- Location (position and concentricity).
- Runout (circular and total).
- Profile (line and surface)
- Symmetry

The problem is at least two-fold: (1) setting the right kind of tolerances, and (2) making changes to ensure validity. It is easy to see that ensuring validity in one place may create an invalid condition in another. For example, correcting a parallel condition as per ISO parallelism standard may destroy an incidence relationship between a surface and an edge. Correcting this new problem may introduce another, etc. At the end the only option there may be to correct some conditions and fill the gaps that are left behind. An example is shown in Figure 9 where the gaps created in Figure 5 are filled using the patch operator of Rhino.

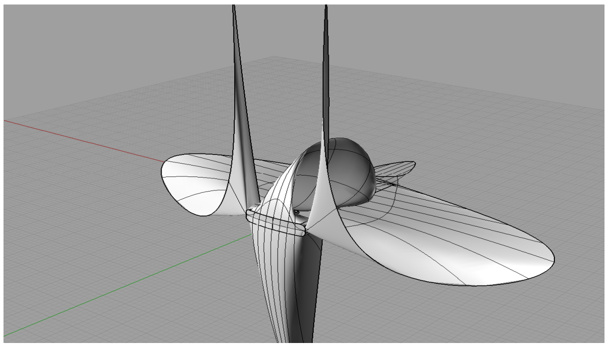

Fig. 9: The patch operator is applied in Rhino.

Patching up CAD models based on NURBS is a bad idea especially if the crack is long and tiny. There are two available approaches here. The first is to generate a small patch that fills the gap. This does not work because continuity conditions require the use of existing cross-boundary derivatives of neighboring (large) surfaces, which create nonsense patches, as shown in Figure 9. The second approach is to extend the existing surfaces to fill the gap. While this is a better approach, the original tensor product surfaces may not be extended in the tensor product sense and the extension is not unique.

Yet another issue with data healing is the fact that conditions can be contradictory. A simple example illustrates the problem. Assume that we are given a line $L$ and two points $P$ and $Q$. The following distance conditions hold:

$$d(L,P) < \varepsilon \quad d(L,Q) > \varepsilon \quad d(P,Q) < \varepsilon$$

where $\varepsilon$ is the given tolerance to repair the object. Based on the order in which the repair is made the following outcomes are possible:

$$P \in L \quad P \equiv Q \quad Q \equiv P \Rightarrow P,Q \in L$$

That is, data repair is ambiguous, depending on the order in which corrections are made.

In summary, the question remains: if data formats are complicated, standards do not work and models cannot be repaired, how would one produce models in a collaborative, multi-cultural environment? The solution is simple: a new generation of Russian novelists will emerge who study English from early on so that they can express themselves in perfect English (write novels, that is). Then a new generation of Korean readers will also emerge who go through schooling with English language education and hence can read English novels. This way no translation is needed and no information ever gets lost. The sections below outline a proposal for a new standard, called knowledge-guided NURBS, and a new computing paradigm that ensures a more robust way of computing in a NURBS modeling environment.

## 4. KNOWLEDGE-GUIDED NURBS

Since NURBS have become *de facto* standards of representing objects in virtually all applications [11], and because raw numerical computations do not produce robust solutions, it comes natural to endow these powerful entities with sufficient knowledge [4,5,7,9,16,18] so that:

- design knowledge can be embedded directly into the geometric model;
- design intent can be captured and incorporated into the model's knowledge base;
- design replay can be supported through a design-and-edit paradigm; and
- robustness can be enhanced by providing knowledge support to computational algorithms that operate on knowledge types instead of simple data types.

Figure 10 shows the overall architecture of the proposed knowledge-guided NURBS system [13].

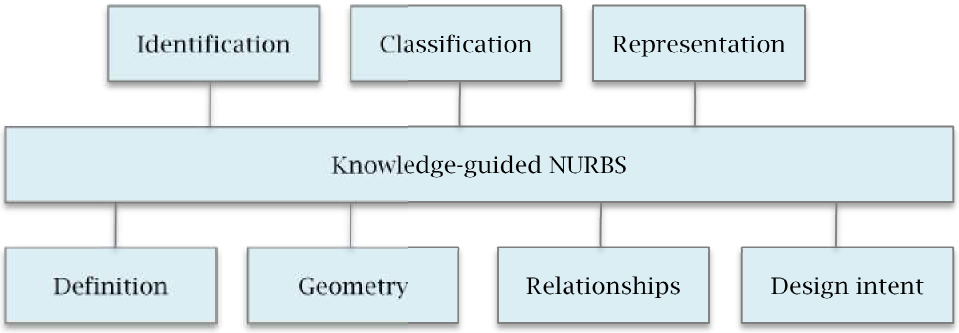

Fig. 10: Architecture of knowledge-guided NURBS.

**Identification.** For proper naming convention and for version control, identification becomes an important issue. It contains a name string, with a prefix, version number and an ID, an indicator whether the entity is rational or not, and the date of initial creation.

**Classification.** Classification follows the *what-how-why* paradigm and contains the following components:

- Object type, or *what* exactly it is in terms of NURBS. Examples: line, spline, plane, lofted surface, etc.
- Object's origin, or *how* it was created in terms of NURBS modeling tools. Examples: import, definition, sketch, offset, etc.
- Object's destination, or *why* it was created and what it may be used for. Examples: intersection, styling, fitting, etc.

**Representation.** NURBS are parametric entities whose representations are not unique. For many processes, in particular, numerical processes, it is critical to know the internal details of parametric forms and their potential danger zones. The following entities are stored:

- Parametrization factor, an indicator on how far the parametrization is from being uniform. It is defined by the maximum and the minimum derivative magnitudes.
- Level of continuity: geometric, e.g. $G^1$, parametric, e.g. $C^2$, as well as approximate, e.g. $G^{\epsilon}$.
- Irregularities, such as cusps, large weights, multiple knots, coincident control points, negative weights, etc.

**Definition.** This is the classical data structure that defines a NURBS entity. It contains control points, knots, degree(s) and various indexes. The control points are always weighted, i.e. 4-D points, and the knots are clamped, i.e. end knots are repeated with multiplicities equal to degree plus one. It is also required that internal knots cannot have multiplicities greater than the degree(s).

**Geometry.** Storing geometry related information has two main purposes: (1) avoiding the repeated computations of important entities such as arc length, and (2) enforcing design intent, e.g. the curve length is computed to be 1.2754 inches and it is enforced across all receiving systems. The following geometric characteristics are stored:

- Curves: arc length, global minimum and maximum curvatures, dimensionality (2, 2.5 and 3), and openness indicator.

- Surfaces: area, perimeter, global minimum and maximum Gaussian and mean curvatures, and openness indicators in both directions.
- Volumes: volume, area of covering surfaces, perimeter, and openness flags in all three parametric directions.

**Relationship maps.** Ontology [10] and relationship maps [16] are the two common forms of representing knowledge. Since working with NURBS is highly creative and generates highly coupled relationships among objects, the more complicated relationship map structure was selected. It is a sophisticated graph structure that contains references to the following entities:

- Nodes and sub-nodes, e.g. *curve-point* node and *curve-point-sampling* sub-node. The main node represents a relationship between two entities, e.g. curve and points, and the sub-node indicates the operation, e.g. sampling or interpolation, that provided the glue for this relationship.
- References to objects participating in the relationship, e.g. reference to the array of points or to the curve structure, entity IDs or other information that may be useful.
- Parameters that are used to perform the operations that define the relationship among entities. These are the parameters that become crucial for design replay, i.e. to regenerate the design from scratch. For example, the *curve-point-fitting* relationship needs to store the degree of the fitting, the knots, the tolerance, if approximation was required, and a reference to the function that performed the fitting operation.
- Conditions that must be invariant across platforms in order to guarantee compatibility and reproducibility. Such conditions are incidence, parallelism, circularity, etc. These conditions are ensured by consistently used tolerances and utility functions, e.g. a function that checks for parallelism based on a well-chosen condition.
- Other parameters such as the quality of a relationship, e.g. how parallel two lines are, and the level of participation, e.g. what percentage of the parallel lines map to one another, may be considered.

**Design intent.** Although the intent of the designer is probably the most difficult to quantify [9,16], there are a number of indicators that are useful for analysis, editing and replay:

- Alternatives represent design alternatives as well as alternative capabilities provided by different functions. To handle alternatives, one needs to store (1) the name of the function that produced the current design, (2) alternative capabilities, and (3) justification why the current function was chosen.
- Functions producing a design need to be clearly specified so that it can be replayed and edited. The necessary details are name, location, version and documentation.
- Decisions are most important to supply as the same mistakes tend to be made over and over again. Relevant details are the decision itself, justification, and possible deliberations leading to the decision.
- History files allow design by learning and to avoid redoing what has proved to be infeasible. Historical data that should be provided are the date of initial creation, the designer's contact information, all major updates, and references to documentation.

The ultimate goal of the knowledge-guided NURBS system is to generate and to retain as much knowledge as possible to support robustness, as described in the sections that follow, to assist with decision making and to allow design editing and replay thereby avoiding costly and hopeless patching up CAD models.

## 5. KNOWLEDGE ACQUISITION

Knowledge, loosely defined, is three interrelated items:

- raw numerical data;
- information about the data; and
- the ability to use the information to get new data and/or information.

That is, the numbers 213-555-1212 represent *raw data*. Knowing that these numbers represent a phone number is *information*. Knowing how to use this phone number to get new data or additional information is *knowledge*. In general, knowledge acquisition can be a very challenging task, however, in a NURBS-based modeling environment knowledge can be obtained with reasonable effort. The proposed methods are shown in Figure 11.

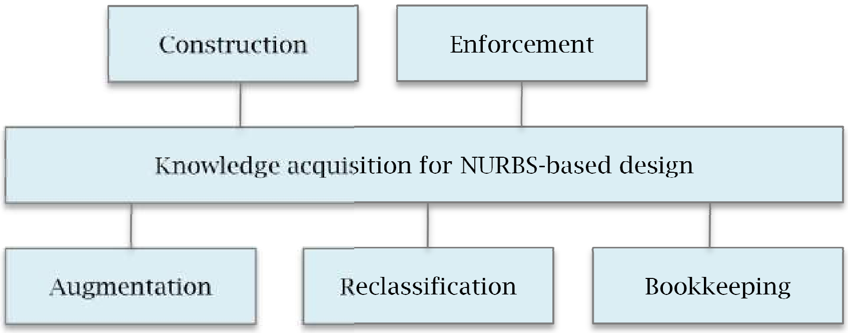

Fig. 11: Knowledge acquisition for NURBS.

**Construction.** Entity construction is by far the most common way of acquiring knowledge. Once the object is designed, its type, origin and destination are saved, and all relevant parameters are entered into the relationship graph. Alternatives and decisions are also recorded.

**Enforcement.** Design intent needs to be enforced to (1) satisfy constraints, e.g. styling with length or curvature constraints, and (2) to eliminate discrepancies between systems, e.g. one reports a parallel case whereas the other shows a non-parallel situation. Enforcing design intent requires an update on the knowledge base, plus (1) possible object editing, (2) a new design, (3) changing the condition and/or the tolerance, or (4) noting the update but leaving the design as is.

**Augmentation.** In case of a discrepancy between the receiving and the sending systems, augmenting the knowledge base without changing either the design or the conditions or the parameters may be the safest course of action. Changing a parallel case to non-parallel because of local requirements, may create chaos in downstream calculations, however, augmenting the knowledge base with new application experiences may turn out to be quite useful. More precisely, relationship pairs may receive contradictory classifications, however, these may in fact contribute to a better decision-making process.

**Reclassification.** When parts enter the supply chain systems used by a variety of contractors, they are often not classified the same way due to discrepancies in tolerances and computational algorithms. Should the parts be needed for a variety of processing, the offending relationships may be reclassified as needed. This reclassification requires (1) updating the knowledge base with full regression, (2) changing the parameters, (3) changing the functions that determined the new classifications, and (4) strict and precise bookkeeping in the history file.

**Bookkeeping.** When constructing an entity, a new relationship is obtained, e.g. intersecting two curves a *curve-curve* relationship is obtained with a *curve-curve-intersection* sub-domain. While this is recorded, additional bookkeeping is needed to record other auxiliary relationships, i.e. *point-curve* with *point-curve-incidence* sub-domain. The usefulness of this is understood by the following simple example. Assume that the intersection point on the two curves is found to not lie on either one of the curves due to different tolerance requirements. Since the point is not linked to any of these curves, there is no way to rectify the problem as the point is considered a floating entity unrelated to the curves in question.

In summary, knowledge guided NURBS with the capability of knowledge acquisition is the basis for a three-tier computational architecture, which in turn will be part of a fault-tolerant computing paradigm based on fault analysis, knowledge acquisition and *n*-version programming.

## 6. THREE-TIER COMPUTING ARCHITECTURE

Numerical algorithms, such as projecting points to free-form entities, has taught CAD developers a simple lesson, raw numerical computations simply do not work [12]. The past few decades has seen various "patches" added to numerical algorithms to compensate for the failures reported by users. Although these patches may eliminate the immediate symptoms, they do not solve the actual problem: raw numerical algorithms are simply inadequate to solve an inherently complex problem, i.e. they were invented to perform number crunching, not problem solving. Problem solving is a highly creative process that involves, as a minimum, the following: (1) a tool chest, e.g. Newton iteration, (2) experimentation, (3) risk taking, e.g. making lots of mistakes, (4) learning, primarily from mistakes, and (5) building knowledge bases. A three-tier architecture is proposed in Figure 12 that forms the bases of a more complex system aimed at fault-tolerant computing, outlined in the next section. From bottom-up the components are explained below.

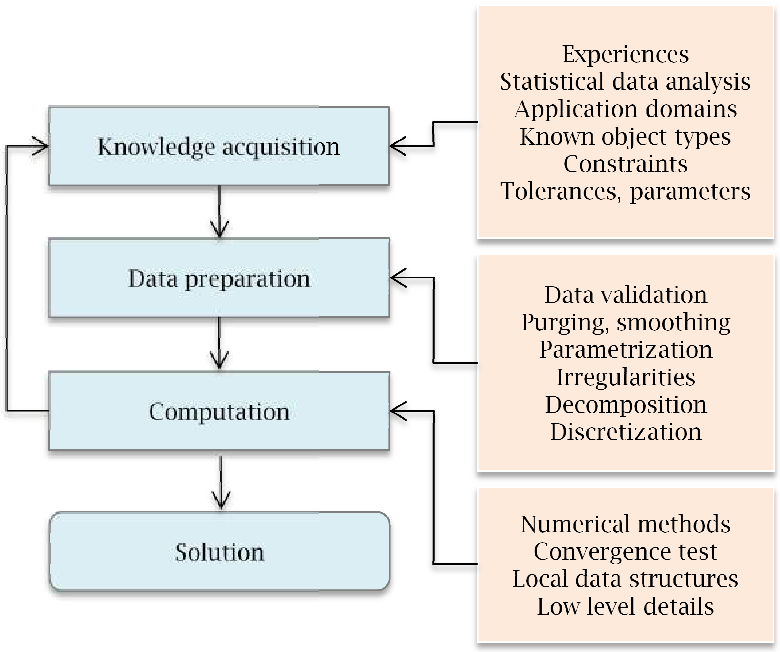

Fig. 12: Three-tier computing architecture.

**Computation.** This is the usual, and almost always, the only layer of available method. The responsibilities at this level involve:

- All low level details, e.g. program interface or error handling.
- Local data types and data structures.
- Various constraint satisfactions, e.g. convergence tests.
- Numerical methods, e.g. iterations or system of equation solvers.

Traditional textbook algorithms, which tend to form the basis of most CAD systems (with some patches added), have no choice but returning an error if the convergence test fails or the solution is not within the required tolerance. Additional layers of capabilities are required to make sure that this is not the end of the road.

*Output:* a valid solution, or invalid solution or failure for analysis and knowledge base update.

**Data Preparation.** It is quite amusing to see that operators of cars carefully select the right kind of gasoline and oil to make sure the car gets the right "input", yet users of algorithms tend to believe that they can throw just about any garbage data at them. Algorithms are like fine cars: they require the input in the right format and amount, i.e. data preparation is one of the most important parts of a successful computing experience. While various applications may require different forms of preparations, the following techniques are quite universal:

- Data validation is the most critical part of trouble free computing. Experience has shown that the majority of the problems come from invalid input data. Examples are invalid NURBS definition, coincident data points, etc.
- Data smoothing is a technique that eliminates unnecessary noise, or introduces some. A typical problem is to try to fit a curve to noisy data when the tolerance is smaller than the noise. The resultant curve becomes so wiggly that it is completely useless. On the other hand, smooth data may be a curse in some cases, e.g. symmetrical arrangements may cause some algorithms to fail. Jittering or making the input entities fuzzy can solve the problem without altering the original method.
- Data purging or thinning is another important preparatory step when there is more data than needed. Excess data not only slows down the system but it may introduce inaccuracies by propagating the error through a large number of entities.
- Improper parametrization is the mother of all failures in NURBS systems. The parametrization is deemed improper if it is highly non-uniform or its level of continuity is lower than expected. Lots of problems stem from the discrepancy of parametrizations in Euclidean and homogeneous spaces, i.e. rational curves and surfaces have two lives: they are stored and computed in 4-D, however, they are viewed and manufactured in 3-D. It is absolutely paramount that NURBS are created with smooth parametrizations and if this is not the case, then all entities are reparametrized before any computation begins.
- Irregularities, such as cusps, zero curvatures or large weights, can cause fatal errors in almost all numerical algorithms. These irregularities must either be removed via redesign or reparametrization, or handled via special code.
- Decomposition techniques have been used successfully in a NURBS-Bezier-NURBS based paradigm. Since Bezier entities are simple polynomials with no internal knots, most of the methods, e.g. degree elevation, are very efficient on the Bezier. The method works as follows: (1) the NURBS is decomposed, via knot refinement, into its Bezier constituents, (2) the required computation is performed on the Bezier pieces, and (3) the results are re-composed, via knot removal, into a required new NURBS.
- Discretization is almost always required when various quantities, such as arc length, are computed. In the world of computing everything is approximate, e.g. the arc length is the length of an approximating polygon. The level of dicretization is determined by what knowledge is available or it is based on tolerances or other constraints such as density.

Data preparation is the bulk of the problem in any NURBS based system. It can be significantly more complicated than the actual numerical algorithm and can consume considerably more time. However, this is the price tag for reliability and robustness.

*Output:* smooth NURBS pieces with no irregularities, or discretized geometry, or proper start point, etc.

**Knowledge acquisition.** The success of computation depends on how well the data is prepared, which in turn requires knowledge on just how this preparation is to be performed. That is, without adequate knowledge, computations become nothing more than guessing games. What distinguishes human computing from machine computing is that humans can see, hear, and learn by experimenting and making lots of mistakes. It may come as a surprise but NURBS systems can be "humanized", i.e. the system can be endowed with a wealth of information (knowledge) that is used to prepare the data so adequately that low-level algorithms hardly ever fail. Based on experience, the following types of knowledge can be acquired:

- Experiences, in particular, with similar objects. For example, if the task is font design using quadratic NURBS, and the font is the letter "S", then decomposition should preserve the corners, the weights should not exceed 1.5 and the parametrization along the smooth segments should be quasi uniform. This in turn should call for weight-based reparametrization and multiple knots at the corner locations.

- Data analysis via running statistical tools on large cases. For example, random data points situated along a long and narrow rectangular domain give rise to poor least squares planes. This bit of knowledge has been acquired by statistics on a variety of data sets.
- Application domains almost always have characteristics that require special attention. For example, tessellation capabilities are very sensitive to the accuracy of the trimming curves, Newton iteration during point projection is sensitive to parametrization and to the choice of the start point [15], etc. On the other hand, computing the global maximum derivative magnitude may require a high level of knot refinement in some applications, e.g. data exchange, but allow a loose bound in many others, e.g. computing intermediate results.
- Knowing the type of object to be computed on can have a profound effect on both reliability as well as accuracy [15]. A significant number of operators, e.g. point distance calculators or intersection routine, have special code for known data types. For example, there is a world of difference between intersecting a line and a circle, and a line and a NURBS curve. Knowing that the NURBS curve is in fact a circle, would make the accuracy and the reliability jump by orders of magnitude.
- Constraints can be easily gathered either from prior experiences or from relationship maps. For example, knowing the type of curve in question, the number of iterations for a Newton-type process can be restricted. If the process is not completed within the required iterations, it will not converge and hence proper action is called for. Knowing that entities have special relationships, e.g. parallelism, also avoids unnecessary and completely error prone operations.
- Assisting data preparation with the right parameters and tolerances is probably the simplest and the most hands-on service the knowledge base can provide. Based on the parametrization factor, curvature analysis, or simply on various experiences, the myriad of parameters can be adjusted so that the input data is well prepared for successful computation.

Traditional computing uses a lot of tune-ups on the basic algorithms. This is contrary to how humans operate: if the gardener is not skilled using a shovel, shoveling will not be improved by tuning up the tool, but rather, improvement is needed at the skill level. Similarly, successful computing requires a high level of skill to prepare the data and to use low-level tools. That is, the module that needs to be dynamically updated is the knowledge base and its communication capability with the rest of the system.

*Output:* reference to special code for special data types, proper parameters and tolerances, convergence and other constraints, etc

## 7. FAULT-TOLERANT COMPUTING

In a NURBS-based system faults may mean two different things: (1) an implementation bug as in any other software system, or (2) the departure from the expected behavior, e.g. non-convergence, inaccurate result or incorrect result because of the application of improper tolerances. In this section the focus will be on the second type of faults, i.e. faults that are the cause of imprecision and the discrepancy between the neat analytical and the dirty numerical solutions. Figure 13 shows the architecture of a system that is tolerant to this kind of faults. The major components are an algorithm vault, failure analysis and knowledge acquisition and update.

The idea of applying an algorithm vault is based on *n*-version programming used in safety critical systems (allowed to fail once in every 114,000 years!). The vault contains several algorithms that are designed by satisfying the following constraints:

- All algorithms are functionally equivalent, e.g. they have the same input/output lists and provide the same solution to a given problem.
- Each algorithm is the work of independent developers, i.e. the chance of making the same mistake more than once is minimized.
- Each algorithm is based on a different solution strategy, e.g. for solving a system of equations one would choose, Gaussian elimination with or without pivoting, LU-decomposition or singular value decomposition.

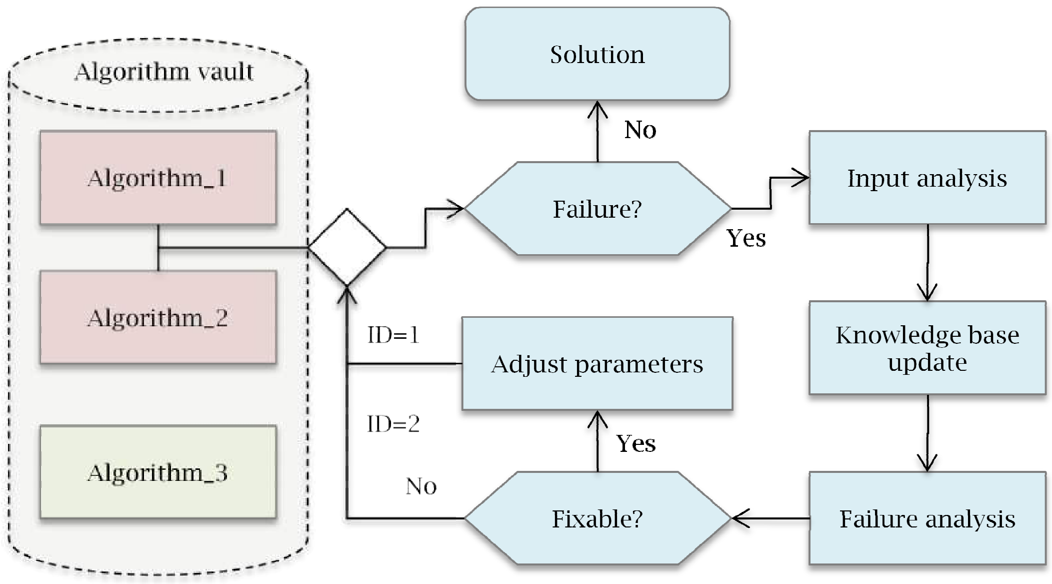

Fig. 13: Architecture of fault-tolerant computing.

The advantage of using a vault is that a large repertoire of methods allow more efficient and much more reliable computing. The major disadvantage is the vast amount of development it takes to build such capability over a long period of time. The other important issue is that robustness is not increased quickly; it comes over time as more and more information is acquired and better methods are introduced. Let us trace the workflow as shown in Figure 12.

Assume that *Algorithm_1* is called to solve a problem. If it works fine, the solution is presented. However, if it fails, the following steps are performed:

- The input data is analyzed to see what can be learned from this failure. Note that data preparation, as described in the previous section, has already been done.
- The knowledge base is updated based on the input condition, e.g. Newton iteration fails when the parametrization factor is greater than 0.8, the tolerance requirement is below $10^{-6}$ and the iteration limit is 5.
- Next, the failure is analyzed, i.e. can certain parameters be changed in order to generate the expected output? For example, if Newton does not converge within 5 iterations in the example above, one can provide a better start point and/or allow the program to iterate longer than 5 times. That is, instead of generating an error, the system should make an attempt, based on available knowledge, to generate the required output.
- If the problem is fixable, then after some parameter adjustments, e.g. iteration limit, the same *Algorithm_1* is attempted again (ID=1).
- If the failure is not avoidable or after several attempts *Algorithm_1* still does not deliver the required solution, a new method, *Algorihm_2* is taken from the vault (ID=2) and the process is repeated.

There are two main questions here: (1) just how efficient is this system, and (2) will the system ever stop? The halting problem is easy to solve by putting a limit on how many times an algorithm is allowed to execute. After all attempts are exhausted, the system will stop and an error is returned.

The efficiency issue is a bit more involved. First, in today's computing world, with ever-faster processors, plenty of memory and special purpose hardware, it is quite acceptable that several versions of the same algorithm are executed multiple times. Second, the wait time between fixing an error, rerunning the same code and producing the new part can be days or even weeks. The proposed system does all this in the background within a fraction of a second. Third, as knowledge accumulates as a result of early failures, systems become smarter and smarter, reducing the time of trial an error and hence improving productivity even further. Finally, as new types of algorithms become available, as a result of development in a cross-disciplinary environment, they are placed in the vault (*Algorithm_3* in the dashed box) to further enhance reliability and robustness. For example, data in bioengineering, based on CT scans, and data in mechanical design can be quite different. Traditional fitting algorithms, developed for the MCAD world, tend to fail miserably requiring the development of a new breed of methods, the crux of which are data smoothing and segmentation. These new techniques, in turn, enhance the quality of the results obtained decades ago using more traditional algorithms. That is, the vault is a dynamic storage of capabilities; new capabilities are added and some of the older ones are retired. Yet, the user of such a system does not need to know what is the latest version of an algorithm or how the problem is solved internally.

## 8. CONCLUSIONS

This paper presented a framework and a global architecture of an intelligent and semi-autonomous system of computing in a NURBS-based modeling environment. The central pieces of the system are knowledge acquisition, data preparation and a dynamically maintained algorithm vault.

The future of CAD and modeling systems is on a divergent path: data formats are increasing, standards are not fully implemented and/or adhered to, and parts cannot be fully repaired as a result of various (tolerance) requirements across supply chain companies. The time has come to select one data format, one standard and avoid translators as well as repair.

## 9. ACKNOWLEDGEMENTS

This research was supported by the National Science Foundation under grant No. DMI-0758231, awarded to the University of South Florida. All opinions, findings, conclusions and recommendations expressed in this paper are those of the authors and do not necessarily reflect the National Science Foundation or the University of South Florida.

## 10. REFERENCES

[1] ASME Standard, dimensioning and tolerancing, ASME Y14.5M-1994, ASME, New York.
[2] CAD Doctor, http://www.elysiuminc.com/fr_index.html?.proinfo/caddoctorinfo.html, Elysium Inc.
[3] CADFix: A product for CAD data exchange, repair, and upgrade, International TechneGroup, Inc.
[4] Contero, M.; Company, P.; Vila, C.; Aleixos, N.: Product data quality and collaborative engineering, IEEE Computer Graphics and Applications, 22(3), 2002, 32-42.
[5] Fenves, S. J.: A core product model for representing design information, NISTIR6736, Gaithersburg, MD, 2001.
[6] Gu, H.; Chase, T. R.; Cheney, D. C.; Bailey, T. T.; Johnson, D.: Identifying, correcting, and avoiding errors in CAD models which affect interoperability, Transaction of the ASME, 1, 2001, 156-166.
[7] Hicks, B. J.; Culley, S. J.; Allen, R. D.; Mullineux, G.: A framework for the requirements of capturing, storing and reusing information and knowledge in engineering design, International Journal of Information Management, 22, 2002, 263-280.
[8] ISO 1101: Geometric tolerancing – tolerancing of form, orientation, location and run-out – generalities, definitions, symbols, indications on drawings, 1983.
[9] Iyer, G. R.; Mills, J. J.: Design intent in 2D CAD: definition and survey, Computer-Aided Design and Applications, 3(1-4), 2006, 259-267.
[10] Lee, J.; Suh, H.; Han, S. H.: Ontology-based knowledge framework for product development, Computer-Aided Design and Applications, 2(1-4), 2005, 635-643.
[11] Piegl, L. A.; Tiller, W.: The NURBS Book, Springer-Verlag, NY, 1997.
[12] Piegl, L. A.: Knowledge-guided computation for robust CAD, Computer-Aided Design and Applications, 2(5), 2005, 685-695.
[13] Piegl, L. A.: Knowledge-guided NURBS: principles and architecture, Computer-Aided Design and Applications, 3(6), 2006, 719-729.
[14] Piegl, L. A.; Rajab, K.; Smarodzinava, V.: High-fidelity conversion of NURBS curves for data exchange, Computer-Aided Design and Applications, 4(5), 2007, 719-729.

[15] Piegl, L. A.; Rajab, K.; Smarodzinava, V.; Valavanis, K. P.: Point-distance computations: a knowledge-guided approach, Computer-Aided Design and Applications, 5(6), 2008, 855-866.

[16] Regli, W. C.; Hu, X.; Atwood, M.; Sun, W.: A survey of design rationale systems: approaches, representations, capture and retrieval, Engineering with Computers, 21, 2005, 395-405.

[17] Shah, J. J.; Ameta, G.; Shen, Z.; Davidson, J.: Navigating the tolerance analysis maze, Computer-Aided Design and Applications, 4(5), 2007, 705-718.

[18] Studer, R.; Benjamins, V. R.; Fensel, D.: Knowwledge Engineering: Principle and Methods, Data Knowledge Engineering, 25, 1998, 161-197.

[19] You, C. F.; Chan, T. H.: Assurance of product data, Computer-Aided Design and Applications, 3(1-4), 2006, 221-230.