Knowledge-guided NURBS: Principles and Architecture

Les A. Piegl[1]

[1]Univesity of South Florida, lap@piegl.com

ABSTRACT

This paper outlines the fundamentals of a knowledge-guided modeling system based on NURBS. NURBS have become standard tools in shape representation in almost all fields where shape information is processed, however, the representation has been largely numerical with no sufficient information attached to aid the design and production processes. The purpose of this research and development effort is two-fold: (1) transform the raw numerical data into a knowledge base, incorporating relevant design parameters and decisions for later use, and (2) increase the robustness and productivity of NURBS-based systems by proliferating knowledge from low level functions to the entire system, and from one system to any other receiving system.

Keywords: Design knowledge, Design intent, Knowledge-guided systems, NURBS.

1. INTRODUCTION

The B-spline technology, both the rational as well as the non-rational B-spline, commonly referred to as NURBS [10], has become the *de facto* standard in shape representation, design and processing. Once a part design is finished, it can be saved in popular formats such as those of the CAD vendors or some standard data formats such as IGES or STEP. Unfortunately, the saved data is largely numerical carrying no information about the design process parameters or the reasoning behind the decision of the designer. As the design process becomes more and more knowledge intensive, possibly involving numerous designers working in geographically diverse locations, it is becoming a necessity to record as much information about the design process as possible. This applies not only at the higher level, as witnessed in, the by now mature field of, knowledge-based engineering, but at any level where information loss can cause problems somewhere in the design pipeline, e.g. sending design parts from one system to the other.

This paper is a step in the direction toward addressing two major challenges [12]:

- organizing and managing knowledge (in a NURBS-based kernel [10]); and
- building a useful knowledge-based system.

The current paper is the first in a series of papers that report on the progress of building a knowledge-guided NURBS modeling kernel. Our goals are as follows:

- embed design knowledge into the geometric model;
- capture design intent and incorporate it into the model's knowledge base;
- support design replay, i.e. provide a mechanism for design reproducibility;
- provide more robust computations supported by knowledge; and
- introduce capabilities for knowledge acquisition, deduction and mining.

The literature on knowledge-guided systems is fairly sparse, spanning fields from artificial intelligence to cognitive science. To start with, the survey paper [12] is highly recommended, along with the references therein. Additional references worth perusing are in the areas of design rationale [9,12], knowledge capture [3], behavioral modeling [13], knowledge representation [14], design intent [4-6], design knowledge [7] and cognitive factors [8]. Other alternative schemes are ISO STEP [11], the Core Product Model [2] and the Open Assembly Model [1].

The paper is organized as follows. Section 2 is on data healing to support the relevance of using knowledge. In Section 3 the general usage of knowledge in knowledge-guided NURBS systems is outlined, followed by the presentation of relevant details about our system in Sections 4 and 5. Section 6 is on knowledge acquisition and Section 7 illustrates how the knowledge base is built and used. Some conclusions are offered at the end of the paper.

2. DATA HEALING

CAD/CAM systems produce models that satisfy certain requirements, either embedded into the system used to produce the model, or given by the contractors. Although many modeling methods are subject to numerical failures, the problems can be handled with a few tweaks and patches. Unfortunately, these fixes loose their effects once the part begins its journey in the world of supply chain companies (it is not unusual to see anywhere from 100 to 200 companies producing parts for such a simple thing as a laptop computer). The part, entering the various CAD systems, may be received with varying levels of enthusiasm, ranging from minor misfits to major glitches. To illustrate the magnitude of the problem, let us take a simple example of trimming surfaces covering point clouds in a reverse engineering application.

Surface trimming is an operation known as surface-surface intersection. To obtain the intersection curve, one computes a dense set of intersection points, followed by fitting a curve to these points up to a certain tolerance. Now, the points do not lie on the surfaces and the curve only approximates these points as well. This is OK as long as the curve is “close enough” for manufacturing applications. What does it mean “close enough?” The answer is simple: we do not know. What is close enough for one application, e.g. placing the fender to the front end of the car, may be too far for another, e.g. modeling the wind shield to avoid glare. So here comes the problem: once the part, made in system A, is read into system B, things may not match up, i.e. the trimmed model may not be water tight. Even though the topological data structure does show the intersection curve as an edge between the two surfaces, the gap between the surfaces and the intersection curve contradicts the data structure. To remedy this problem the CAD doctor is called in and the data healing process begins. While there is no good solution to this problem, the CAD doctor prepares a remedy to fill the gap with a small surface including or excluding the (incorrect) trimming curve. The small patch surface indeed fills the gap and makes the model water tight, however, it opens up another can of worms in the areas of parametrization, smoothness, shape fidelity, etc. Unfortunately, one cannot squeeze a small surface between two large ones with reasonable level of continuity as the cross-partial derivatives of the two large surfaces would push the tiny surface way out of proportion (the surface may loop around).

There are three significant problems at hand in the receiving system:

- there is no (or not enough) information on *what* the curve is;
- there is no information on *how* it was generated; and
- there is no indication as to *why* or what it was made for.

In a knowledge-guided system the above information is readily available and the problem is solved as follows:

- the system sees that the curve is a *spline* curve, created via *approximation* and used to represent the *intersection* curve;
- the intersection curve is in *relationship* with the two surfaces whose references and definitions are also available;
- the *parameters* of obtaining the intersection curve, i.e. the number of sampling points, their locations and the tolerance used, are available;
- obtaining all this information, the initial design can be *replayed* and the new intersection curve is computed satisfying system B’s requirements;
- everything that needs to be modified is done and the knowledge base is *updated*.

The price to be paid for this design replay is that both the sending and the receiving system must have the ability to process both geometric as well as knowledge information. The best way this can be done is that both systems have the same capabilities in the form of the same knowledge-guided kernel. Translating knowledge information from one output to a proprietary input may have the same data loss as translating geometric information from one representation to the other.

History has taught us that patching up CAD models does not pay very well, in fact, it can cause an avalanche. Assume that the model in the previous example is now passed on to system C. If we get lucky, then the requirements in system C’s applications are such that the model remains water tight. However, if not, which is the case quite often, then yet another call to the CAD doctor is required to put another, perhaps even smaller, patch in two gaps (between the patch surface and the two original surfaces). One can only imagine what happens when this part reaches the 107th company in the supply chain!

3. KNOWLEDGE BASE FOR NURBS SYSTEMS

A knowledge base is used in a NURBS-based modeling system primarily in two different case scenarios: (1) building an initial model and its knowledge-base, and (2) using the model and the data base for various design tasks. Fig. 1 illustrates how the data bases are built.

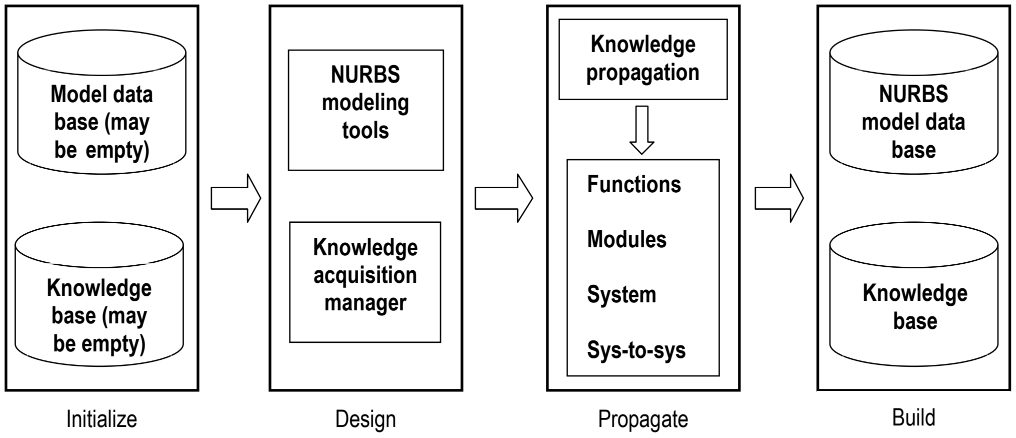

Fig. 1. Building an initial knowledge base for NURBS modeling.

The data bases are built in four steps. The process starts with the two data bases, the model base and the knowledge base, which may be, and often are, empty. As the design proceeds, the modeled parts are recoded and knowledge about the design intent is acquired. All this information is then propagated to the rest of the system to aid additional design tasks. Upon completing the job, or part thereof, a model data base and a knowledge base are built.

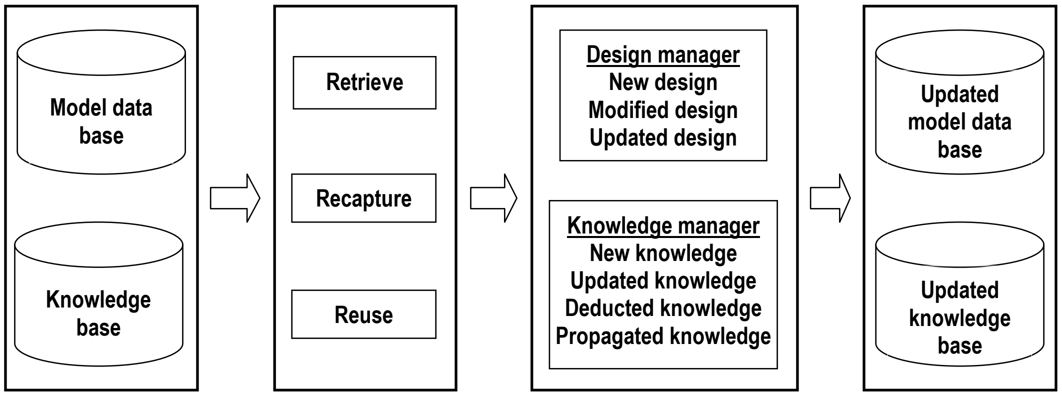

Fig. 2. Using knowledge bases in NURBS modeling.

Fig. 2 illustrates the process of how to make (good) use of all the information generated during the first phase shown in Fig. 1. The retrieve-recapture-reuse mechanism is employed to pass information to the design and to the knowledge managers. The design manager may use this information to generate a new design, modify an existing design or just simply make a small update. The knowledge manager, working in parallel to the design manager as well as independently, takes advantage of the recaptured information along with the output of the design manager to generate new knowledge, or update existing entries in the data base, or deduce knowledge from existing ones, and finally to propagate all this to the rest of the system. The end result is the updated model and knowledge bases. This cycle can, of course, continue indefinitely as long as the modeling system is in use. Details on all the major elements are given in subsequent sections of this paper.

4. KNOWLEDGE-GUIDED NURBS (KGN)

A major challenge in knowledge-guided systems is how much information to retain and/or generate. Too little information may not allow the system to replay the design process, on the other hand, too much information may create problems with storing, using and maintaining all the relevant as well as irrelevant knowledge. The major components of our knowledge-guided NURBS system, developed after two decades of experience with NURBS systems, are shown in Fig. 3.

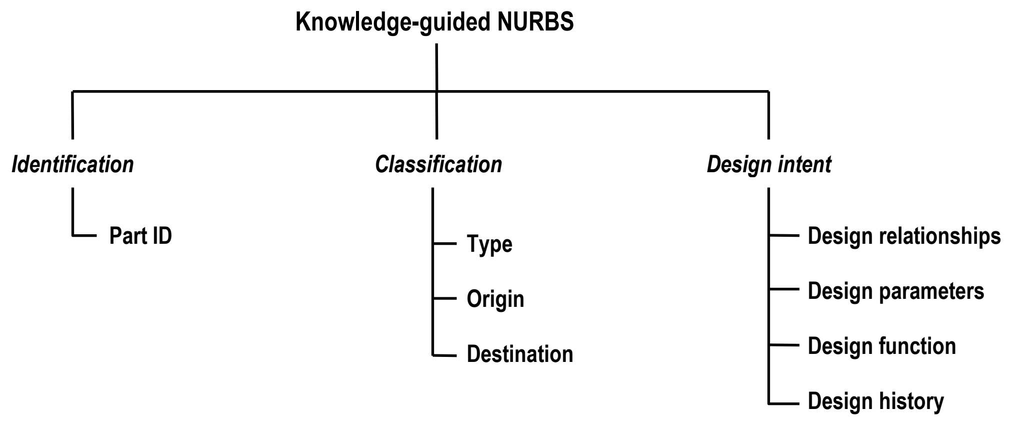

Fig. 3. Structure of knowledge-guided NURBS.

4.1 Identification

Each entity in the system has a unique ID for purposes of identification, saving and rebuilding the data bases. The ID can be as simple as a number, e.g. "`2007`", however, to make it more meaningful, the recommended identification is in the form of "`KGN_V2.1_2007_CUR_0008`." That is, it uses a prefix (`KGN`), a version number (`V2.1`), the year of the release (`2007`), the entity type (`CUR`) and the entity ID (`0008`).

4.2 Classification: The Three Treasures

Knowledge-related disciplines have considered the what-how-why or the syntax-semantics-pragmatics paradigm for design rationale or capturing design intent. In the knowledge-guided NURBS system we use the following trilogy:

- **Type** represents the type of the object, or *what* exactly it is in terms of the NURBS modeling scheme. Example types are `point`, `line`, `curve`, `surface`, etc.
- **Origin** tells the system where the part comes from, or *how* it was created. Example origins are `input`, `definition`, `interpolation`, etc.
- **Destination** refers to the application where the object will be used, or *why* or *what* was it made *for*. Examples are `offsetting`, `styling`, `lofting`, etc.

Below is an example using the process of offsetting and representing the offset curve as a NURBS curve:

Type	`spline curve`
Origin	`approximation`
Destination	`offset`

That is, the base curve was sampled, each point was offset, and the offset points were *approximated* to get the *offset curve* represented as a *spline curve*. In order to be able to replay this simple design session, the information above is not sufficient. The sections below provide the missing pieces of the puzzle.

5. DESIGN INTENT

The key element of the knowledge-guided system is to capture the design intent inherent in every design process for later use or for complete reproduction of the design. The most important parts of capturing intent are the relationships and the parameters used to define those relationships.

5.1 Relationships

Fig. 4 illustrates the five elements used to define what is meant by a relationship in KGN: name, objects, conditions, quality and participation.

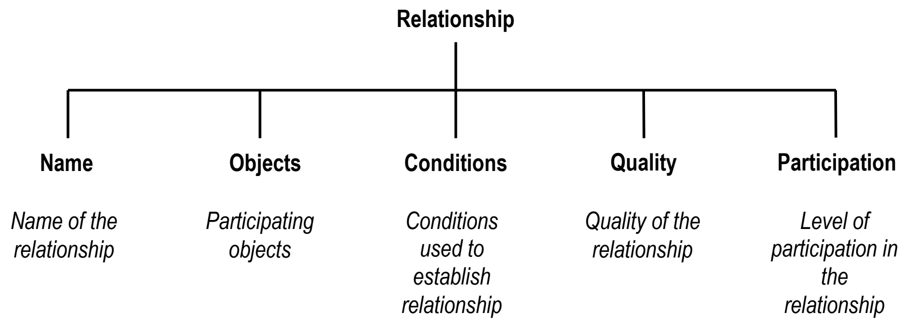
Relationship
Name
Name of the relationship
Objects
Participating objects
Conditions
Conditions used to establish relationship
Quality
Quality of the relationship
Participation
Level of participation in the relationship

Fig. 4. Elements of relationships used in KGN.

5.1.1 Name

The name of the relationship incorporates both the name as well as the participating objects. An example is parallelism with various entities involved:

`Line_to_line_parallel`	lines are parallel
`Line_to_plane_parallel`	lines and planes are parallel
`Curve_to_curve_parallel`	two or more curves are parallel

5.1.2 Objects

For each relationship a list of objects are given that take part in that relationship. Some examples are below.

```
Line_to_line_parallel  = { l1,l2; … }
Line_to_plane_parallel = { l1, l2, l3, pl; … }
```

5.1.3 Conditions

Each relationship requires certain conditions to be satisfied. There are conditions for parallelism, perpendicularity, incidence, etc. Fig. 5 below shows three cases of parallel lines.

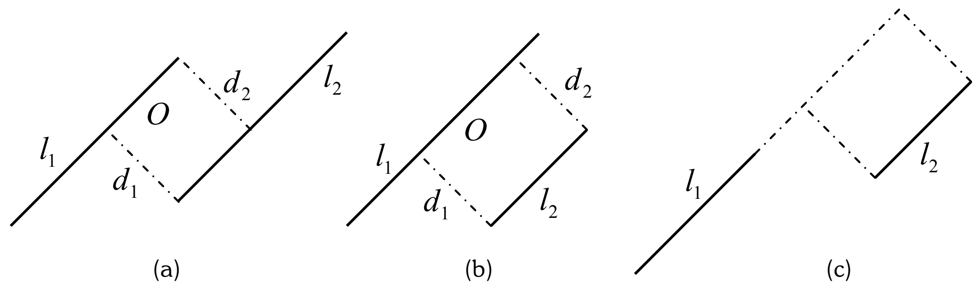

Fig. 5. Cases of parallel line segments.

First of all, all entities in KGN are finite. Second, the traditional definitions, e.g. cross-product magnitude is less than a tolerance, are no longer appropriate. That is, parallelism between two lines is defined as follows:

- the projections of the line segments onto each other must overlap (Fig. 5a-b); and
- $\delta = |d_1 - d_2| < \varepsilon$, where d_1 and d_2 are the distances of the projections of the end points, and ε is the parallelism tolerance.

5.1.4 Quality

It makes perfect sense to ask the question: "just how good is the relationship?" That is, in the case of parallel lines, it is a legitimate question to ask "just how parallel are they?" Referring to the definition above, we can introduce the following quality measure:

$$q = \frac{|\varepsilon - \delta|}{\varepsilon} \tag{1}$$

This quantity lets us know how far we are, in percentages, from the theoretically parallel lines. That is if $\varepsilon \approx \delta$ the quality of the parallelism is near zero percent. On the other hand, $\delta \approx 0.0$ produces a nearly perfect parallelism. Put it in another way, there is no such thing in KGN as parallel lines. However, there are lines that are *weakly parallel*, *parallel* or *strongly parallel*. The classification may change from application to application, or from system to system. A useful application of the quality measure is in error accumulation, e.g. if one of the two weakly parallel lines is offset, it may produce a third line that is no longer parallel to one of them.

5.1.5 Participation

The level of participation indicates to what degree the entities are participating in the relationship. Referring to Fig. 5 above, it is clear that the two lines have different degrees of participation. For example, one line in Fig. 5(b) is participating with its entire length, whereas the other is only partially utilized. The level of participation for the parallel line case is defined as

$$p = \min\left\{\frac{O}{L_1}, \frac{O}{L_2}\right\} \tag{2}$$

where O is the overlap region of the projections and L_1 and L_2 are the total length of the lines. A legitimate question is: why is it important? Fig. 6 illustrates the reason.

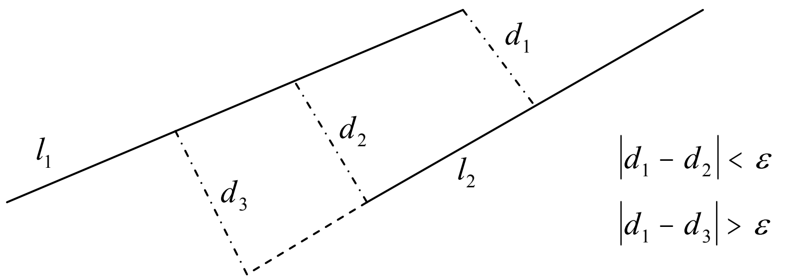

Fig. 6. Increasing the level of participation for parallel lines.

That is, as the level of participation increases, i.e. the length of the lines are getting longer and longer, the quality of the relationship may decrease, e.g. short line segments that are strongly parallel may become weakly parallel or even non-parallel when extended. In the world of NURBS more is not necessarily better. The more entities are participating in a relationship, the more error can creep in and the worse the quality may be to the point where the relationship may no longer exist.

5.2 Design Parameters

Each design task is individually specific and requires its own set of parameters. To reproduce the design one has to store these parameters either in the form of a master parameter table or, what KGN elected, in the form of individual parameters sets attached to the relationships. Fig. 7 illustrates how the parameters are represented for the relationship called "`curve_to_point`."

The following explanation is in order:

- The relationship is called "curve to point" to reflect a directional dependence, i.e. from a curve we produced a set of points.
- There are sub-relationships within "curve to point" such as sampling or incidence and the actual parameters are listed for each "refined" relationship.

- The sampling process requires the reference to the points `P_ref` (the ones that are in relationship with the curve), the reference to the parameters `u_ref`, the parameter bounds of the curve `ul, ur` (in case it gets rescaled), and the quality `q` and the participation `p` (both are 100% for sampling).
- The incidence relationship needs the reference to the points, the tolerance used, the quality of the incidence (just how close these points are to the curve), and the level of participation (how many of the points are participating).

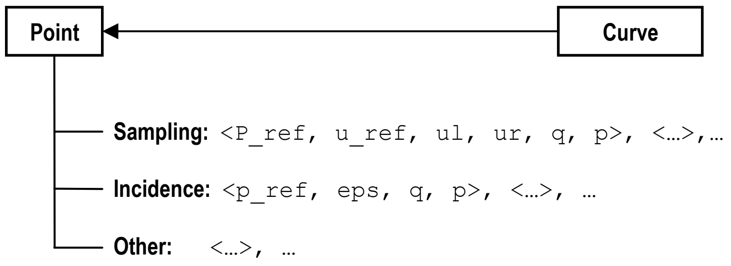
Point
Curve
Sampling: <P_ref, u_ref, ul, ur, q, p>, <…>,…
Incidence: <p_ref, eps, q, p>, <…>, …
Other: <…>, …

Fig. 7. Curve to point relationship.

Once the points are sampled, additional information is stored into the entity type as follows:

Type	`through point`
Origin	`sampling`
Destination	`polygonal approximation`

To see that relationships may not be symmetrical, the reader is referred to Fig. 8 that depicts the “point to curve” relationship.

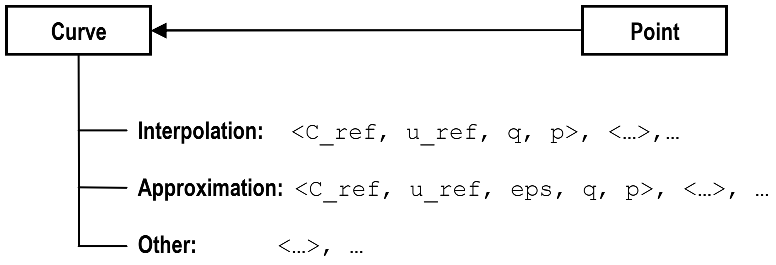
Curve
Point
Interpolation: <C_ref, u_ref, q, p>, <…>,…
Approximation: <C_ref, u_ref, eps, q, p>, <…>, …
Other: <…>, …

Fig. 8. Point to curve relationship.

The structure is quite similar to the “curve to point” except the sub-cases are not the same and of course the parameters are different. Please note that `u_ref` is optional for both the interpolation and the approximation. Once the curve is computed, some additional information is stored as follows:

Type	`spline curve`
Origin	`approximation`
Destination	`intersection curve`

5.3 Design functions

In order to reproduce the design, the functions that used to make the part need to be recorded. As a minimum, the following information is necessary:

- the name of the functions used, e.g. `KGN_asc_spnloft.c;`

- the location of the functions, e.g. `c:\KGnurbs\src;`
- version and release date, e.g. `V2.1-2007`; and
- reference to the (on-line or off-line) documentation, e.g. `c:\KGnurbs\doc\index.htm.`

5.4 Design history

In order to keep the amount of information under control, design history should be kept short. Relevant detailes that are worth recording are:

- the date of the initial creation;
- name and contact details of the designer;
- all major updates, modifications and error fixes;
- contact details of all individuals who made changes since the initial installation; and
- reference to the file that documents all changes if the changes are substantial and require detailed and elaborate documentation.

5.5 Packaging

There is a lot of information to be stored in order to serve the objectives of a knowledge-guided system. The question is where exactly should all this information go, i.e. how to package the bits and pieces of information? KGN elected to partition the information space as shown in Fig. 9.

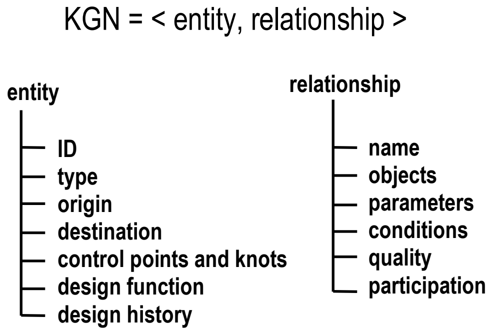

Fig. 9. Partitioning the knowledge space for KGN.

There is an obvious discrepancy between the conceptual presentation of knowledge-guided NURBS and how information gets stored at the software level. The requirement of conceptual modeling and software design are not necessarily the same, and therefore creating images of the system in both the conceptual and software spaces are equally important.

6. KNOWLEDGE ACQUISITION

In this section we examine where and how knowledge enters the KGN system. The five main sources of knowledge are construction, deduction, enforcement, reclassification and double bookkeeping.

6.1 Knowledge by Construction

This is by far the most common way to create a new piece of knowledge. Once an object has been constructed, e.g. an approximating curve is computed to a set of points, it gets an ID, a type (`spline curve`), origin (`approximation`) and destination (`unknown`) are specified, and the relationship with the point set along with the parameters of the approximation procedure are recorded. The name of the design function is saved and the history field is marked as "initial construction." If any change is required, e.g. a tighter approximation, only the information that has changed, e.g. the approximation tolerance and the curve, is updated.

6.2 Knowledge by Deduction

KGN recognizes two ways of knowledge deduction: (1) logical inference and (2) relationship query. Logical inference is a two step process:

- use logical reasoning to infer a new relationship; and

- test if this (theoretical) relationship is valid under the system's requirements.

More formally, deduction works like this:

$$\{\kappa_1,...,\kappa_n\} \xrightarrow{\text{inference}} \kappa_{n+1}(?) \quad (3)$$

where κ_i represents the i-th piece of knowledge and κ_{n+1} is the new piece of knowledge. A simple example is as follows:

$$\{l_1 \,||\, l_2, l_2 \,||\, l_3\} \xrightarrow{\text{inference}} l_1 \,||\, l_3(?) \quad (4)$$

which is the well known transitive relationship of parallelism. If such a (theoretical) relationship is found, the next step is to test if indeed the two lines are parallel using the conditions shown in Figs. 5 and 6.

Relationship query operates on the objects, instead of on the knowledge elements. It can be formulated as follows:

$$\{\Omega_1,...,\Omega_n\} \xrightarrow{\text{query}} \kappa \quad (5)$$

That is, a query is going to be made on some objects to see if they satisfy a certain relationship. Here is an example:

$$\{P_1,...,P_n\} \xrightarrow{\text{co-planar}} \kappa_{co-planarity} \quad (6)$$

That is, given a set of points, check if they are co-planar. If *yes*, link the plane and the points together with the relationship entity called "co-planar."

6.3 Knowledge by Enforcing Intent

One of the major uses of design intent is to resolve discrepancies between the numerical results and the designer's actual intention. Assume that on checking if two lines are parallel, it turns out that the numerical condition is not met, however, the knowledge base flags the entities as parallel, i.e. the designer intended them to be parallel. The discrepancy may be eliminated in a number of ways:

- **Repair** the entities until the condition is met, e.g. modify one or both of the lines till the tolerance condition is met.
- **Redesign** the part that has the offending parts so as to meet the requirements.
- **Modify the condition** without altering the entities, e.g. the lines, or the part that has the entities.

If there is a discrepancy between the design intent and the numerical solution, there has to be a way to fix the problem, although it may not be simple. One method is to follow the design intent by *assigning* the new entity instead of *computing* it, or computing it flowed by subsequent adjustment based on the original intent.

6.4 Knowledge by Reclassification

When entities are passed between systems, due to differing requirements, some of the relationships may no longer be valid. In this case new relationships must be established (if allowed) leading to new pieces of information. An example follows using parallelism between lines:

System A: $l_1 \,||\, l_2$ with respect to ε_1

System B: $l_1 \,||\, l_2$ with respect to $\varepsilon_2 < \varepsilon_1$

Now, if the two lines in system B are no longer parallel, then this knowledge must be erased and a new piece of knowledge may be generated, i.e. $l_1 \rho l_2$ with respect to ε_2. Possibilities for ρ in case of line segments are: `parallel`, `perpendicular`, `nearly parallel`, `overlapping` and `general`. That is, an initially parallel relationship may be reclassified to "nearly parallel" to satisfy the new requirement (and perhaps to aid robustness in, say, line-line intersection). One minor wrinkle with reclassification is that the change of knowledge has to be propagated to the entire knowledge base and if contradictions are found, they need to be properly addressed.

6.5 Double Bookkeeping

Assume that two curves are intersected to get one intersection point. This constitutes a "curve to point" relationship which is called "intersection" and the point is listed under this category. At a later time the designer points at the point and asks the question: what do we know about this point? Although the point is involved in a "point to curve" relationship with the two curves, it is only implicit and some browsing is required to find that out. To make data base browsing, knowledge mining, that is, more efficient, an explicit relationship is entered into the data base, called "incidence", i.e. the intersection relationship created a new piece of knowledge, called incidence, via double bookkeeping. Strictly speaking, all relationships would require double bookkeeping, e.g. point sampling would require an extra incidence, however, it is worthwhile to record relationships both ways if and only if they are needed for design reuse or for knowledge mining.

7. WORKING WITH KNOWLEDGE

During the design process knowledge is constantly generated and used at the same time. Design decisions are recorded into the knowledge base, on the other hand, decisions are also made based on what type of knowledge is already available. There are two processes at work: (1) design and record knowledge, and (2) make use of knowledge.

The design and record session first *identifies* the participating entities. Then their *types*, *origins* and *destinations* are established. Then *relationships* with their relevant *parameters* are recorded. Finally, the knowledge is propagated to:

- the current design function;
- other functions that may be used by the current routine;
- various modules and to the entire system; and
- other systems upon completion of the design task.

Propagation within the system is done by the mechanism of access control, scoping and references. The system to system propagation requires the knowledge base to be exported to a file and to be rebuilt in the receiving system.

Making use of knowledge is done by a simple three step process:

- retrieve all relevant knowledge along with all the participating entities;
- recapture all relationships and the stored parameters; and
- reuse the pair `<entities, relationship>` to *re-create*, *update*, *maintain* or *document* the design, or to increase the *robustness* of various processes with all the available knowledge instead of applying blind computation.

8. CONCLUSIONS

The principles and some architectural issues of a knowledge-guided NURBS system have been presented in this paper. The motivation for developing such a system lies in the concept of intelligent computing that supports the capturing of design intent, design reuse, robust computing via intelligent choice of methods and tolerances, and in-system and cross-platform compatibility. Although the amount of knowledge may seem overwhelming, and indeed it can be if the system is used improperly, the knowledge base can, and should, be built in a flexible manner. In the extreme, all knowledge can be recorded or none at all. In between, it is the user's decision to record only relevant relationships with some parameters, or only a few difficult relationships, e.g. overlapping, with a full range of parameters. Knowledge can also be recorded incrementally starting with a small knowledge base to support a few design tasks, and ending up with the full-blown data base used to support legacy systems.

9. ACKNOWLEDGEMENTS

The author is indebted to Professors Tony Woo and Nick Sapidis 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. All opinions, findings, conclusions and recommendations expressed in this paper are those of the author and do not necessarily reflect the views of the National Science Foundation or the University of South Florida.

10. REFERENCES

[1] Baysal, M. M., Roy, U., Sundarsan, R., Sriram, R. D. and Lyons, K., The open assembly model for the exchange of assembly and tolerance information: overview and example, Proc. DETC 2004, Salt Lake City, UT, 2004,

[2] Fenves, S. J., A core product model for representing design information, NISTIR6736, Gaithersburg, MD, 2001.

[3] Hicks, B. J., Culley, S. J., Allen, R. D. and Mullineux, G., A framework for the requirements of capturing, storing and reusing information and knowledge in engineering design, *International Journal of Information Management*, Vol. 22, 2002, pp 263-280.

[4] Ishino, Y. and Jin, Y., Estimate design intent: a multiple genetic programming and multivariate analysis based approach, *Advanced Engineering Informatics*, Vol. 16, 2002, pp 107-125.

[5] Iyer, G. R. and Mills, J. J., Design intent in 2-D CAD: definition and survey, *Computer-Aided Design and Applications*, Vol. 3, Nos. 1-4, 2006, pp 259-267.

[6] Iyer, G. R., Mills, J. J., Barber, S., Devarajan, V. and Maitra, S., Using a context-based inference approach to capture design intent from legacy CAD, *Computer-Aided Design and Applications*, Vol. 3, Nos. 1-4, 2006, pp 259-267.

[7] Kitamura, Y., Kashiwase, M., Fuse, M. and Mizoguchi, R., Deployment of an ontology framework of functional design knowledge, *Advanced Engineering Informatics*, Vol. 18, 2004, pp 115-127.

[8] Lang, S. Y. T., Dickinson, J. and Buchal, R. O., Cognitive factors in distributed design, *Computers in Industry*, Vol. 48, 2002, pp 89-98.

[9] McMahon, C. and Browne, J., *CAD/CAM: Principles, Practice and Manufacturing Management*, 2nd Edition, Harlow, London, UK, 1998.

[10] Piegl, L. and Tiller, W., *The NURBS Book*, 2nd Edition, Springer-Verlag, New York, NY, 1997.

[11] Pratt, M. J. and Anderson, W. D., A shape modeling applications programming interface for the STEP standard, *Computer-Aided Design*, Vol. 33, 2001, pp 531-543.

[12] Regli, W. C., Hu, X., Atwood, M. and Sun, W., A survey of design rationale systems: approaches, representations, capture and retrieval, *Engineering with Computers*, Vol. 16, 2002, pp 209-235.

[13] Xu, X. W. and Galloway, R., Using behavioral modeling technology to capture designer's intent, *Computers in Human Behaviour*, Vol. 21, 2005, pp 395-405.

[14] Yap, A. Y., Ngwenyama, O. and Osei-Bryson, K.-M., Leveraging knowledge representation, usage, and interpretation to help re-engineer the product development life cycle: visual computing and the tacit dimension of product development, *Computers in Industry*, Vol. 51, 2003, pp 89-110.