2011

Knowledge Guided Non-Uniform Rational B-Spline (NURBS) for Supporting Design Intent in Computer Aided Design (CAD) Modeling

Khairan Rajab
University of South Florida, khairanr@gmail.com

Knowledge Guided Non-Uniform Rational B-Spline (NURBS) for Supporting Design Intent in Computer Aided Design (CAD) Modeling

by

Khairan D. Rajab

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Date of Approval:
June 9, 2011

Keywords: NURBS Model exchange, CAD robustness, CAD incompatibility, Design intent exchange, Data migration

Acknowledgements

First and foremost, my highest gratitude goes to my advisor, Dr. Les A. Piegl. To work with and be mentored by a renowned expert in the area of NURBS has truly been an honor and privilege. He is a generous and conscientious mentor who was always there when I needed advice and guidance.

I thank the members of my committee: Dr. Susanna Lai-Yuen, Dr. Arthur Shapiro, Dr. Dewey Rundus, and Dr. Miguel Labrador. I appreciate their valuable time and comments. They have been inspiration in many ways. I would also like to thank Dr. Steve Permuth for taking the time to serve as the Committee Chair.

My appreciation is also extended to the seminar members who listened to my research presentations on numerous occasions. Thanks to Dr. Daniel Simkens and to Olya Grove for the lively and rewarding discussions about my research ideas.

I have thanked many yet also have left many out. Thanks to all whom I left out but were available to help and encourage. Special thanks to Dr. Yousef AL-Marshad for his relentless encouragement throughout most of my studies as an undergraduate and graduate student. Thanks to Dr. Fahd Rajab for the many comments he provided at the time of writing this dissertation.

Dedication

I most certainly thank and dedicate this work to my family for all their love and unconditional support. Without them my graduation would have been impossible.

Table of Contents

List of Tables

List of Figures

Abstract

For many years, incompatible computer-aided design (CAD) packages that are based on Non-uniform Rational B-Spline (NURBS) technology carried out the exchange of models and data through either neutral file formats (IGES or STEP) or proprietary formats that have been accepted as quasi industry standards. Although it is the only available solution at the current time, the exchange process most often produces unsatisfactory results. Models that are impeccable in the original modeling system usually end up with gaps or intersections between surfaces on another incompatible system. Issues such as loss of information, change of data accuracy, inconsistent tolerance, and misinterpretation of the original design intent are a few examples of problems associated with migrating models between different CAD systems. While these issues and drawbacks are well known and cost the industry billions of dollars every year, a solution to eradicate problems from their sources has not been developed. Meanwhile, researchers along with the industries concerned with these issues have been trying to resolve such problems by finding means to repair the migrated models either manually or by using specialized software.

Designing in recent years is becoming more knowledge intensive and it is essential for NURBS to take its share of the ever increasing use of knowledge. NURBS are very powerful modeling tools and have become the de facto standard in modeling. If we stretch their strength and make them knowledge driven, benefits beyond current expectations can be achieved easily. This dissertation introduces knowledge guided NURBS with theoretical and practical foundations for supporting design intent capturing, retrieval, and exchange among dissimilar CAD systems. It shows that if NURBS entities are tagged with some knowledge, we can achieve seamless data exchange, increase robustness, and have more reliable computations, all of which are ultimate objectives many researchers in the field of CAD have been trying to accomplish for decades. Establishing relationships between a NURBS entity and its origin and destinations can aid with seamless CAD model migration. The type of the NURBS entity and the awareness of any irregularities can lead to more intelligent decisions on how to proceed with many computations to increase robustness and achieve a high level of reliability.

As a result, instead of having models that are hardly modifiable because of migrating raw numerical data in isolation, the knowledge driven migration process will produce models that are editable and preserve design intent. We have addressed the issues not only theoretically but also by developing a prototype system that can serve as a test bed. The developed system shows that a click of a button can regenerate a migrated model instead of repairing it, avoiding delay and corrective processes that only limit the effective use of such models.

Chapter 1---Introduction

1.1 Motivation and Problem Formulation

Most current Computer-Aided Design (CAD) systems primarily focus on designing and manufacturing capabilities. They accommodate only geometric and limited production data, ignoring one of the most important factors in the designing process, namely the *design intent*. Some commercial CAD systems retain the design process history. However, when a CAD model is converted from one system to another, using natural format files, the modeling history, the knowledge used during modeling process, and the design intent are all lost and cannot be recovered. If CAD systems can encode and support knowledge and design intent at the lowest level, along with the geometries of each Non-uniform Rational B-Spline (NURBS) entity during the modeling process, a wide range of benefits can easily be achieved. Benefits can be as important as enhancing robustness that has been hindering CAD system builders for many years to overcoming the more serious issues usually stemming from data migration and inconsistencies across incompatible CAD systems, which cost industries and governments billions of dollars each year [1].

To put things in perspective for the reader, a simple example will illustrate why it is important to support design intent in CAD modeling. Figures 1.1 and 1.2 are renderings of a complete *intended* design of a flash drive. This design was generated with Rhino, the NURBS modeling system for Windows.

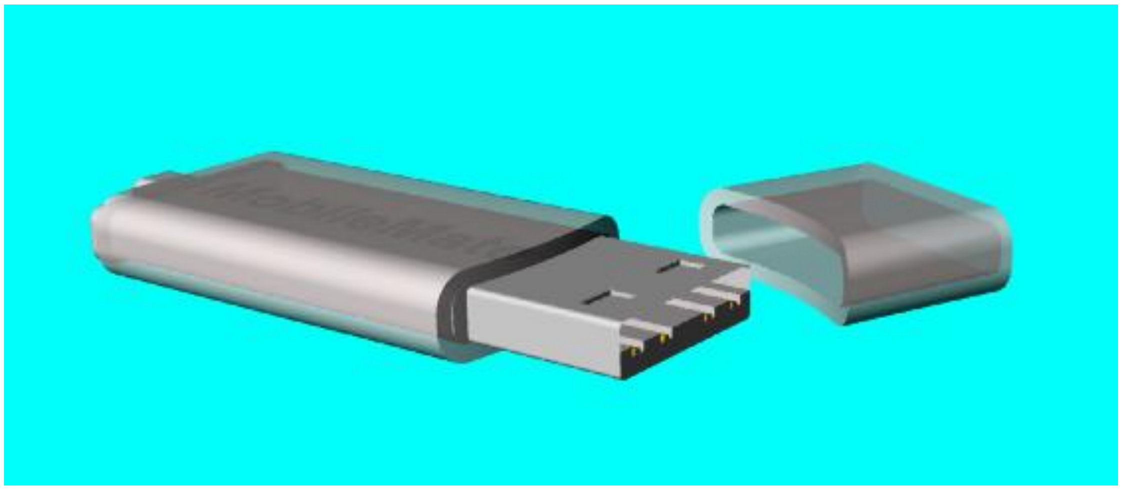


Figure 1.1. USB drive designed with Rhino modeling system

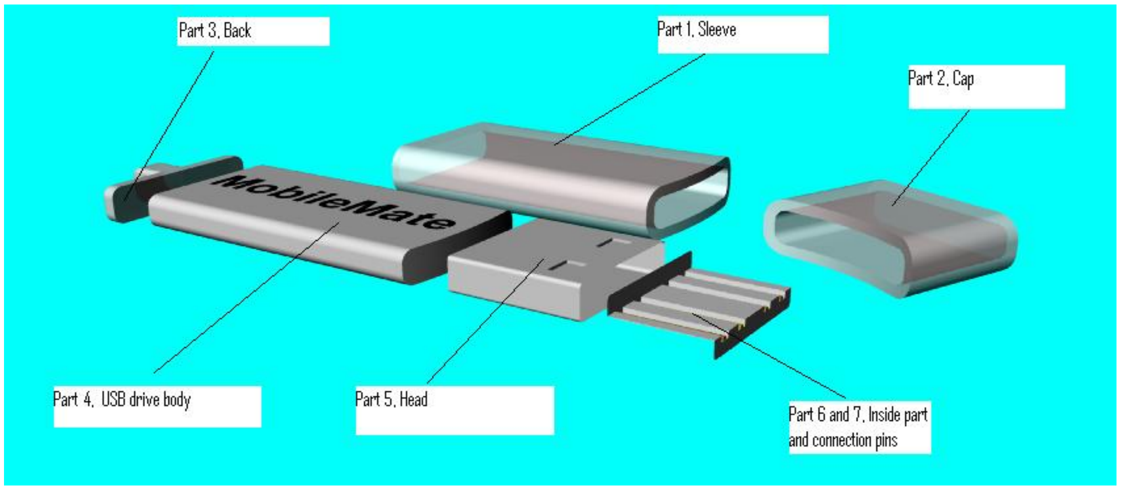


Figure 1.2. Parts of the USB

To be able to send or open the model in another system, e.g., CATIA, the designer will have to export the intended model to a neutral format. Usually designers use one of the two standards: Initial Graphics Exchange Specification (IGES) or Standard for the Exchange of Product model data (STEP). Other non-standards or de facto standards, such as STL and DFX, can be used as well. The receiving system, CATIA in this case, will have to reverse the mapping process by remapping the migrated entities from the neutral file (in IGES or STEP format) to its local proprietary format. Figure 1.3 shows schematically how the process works between two CAD systems.

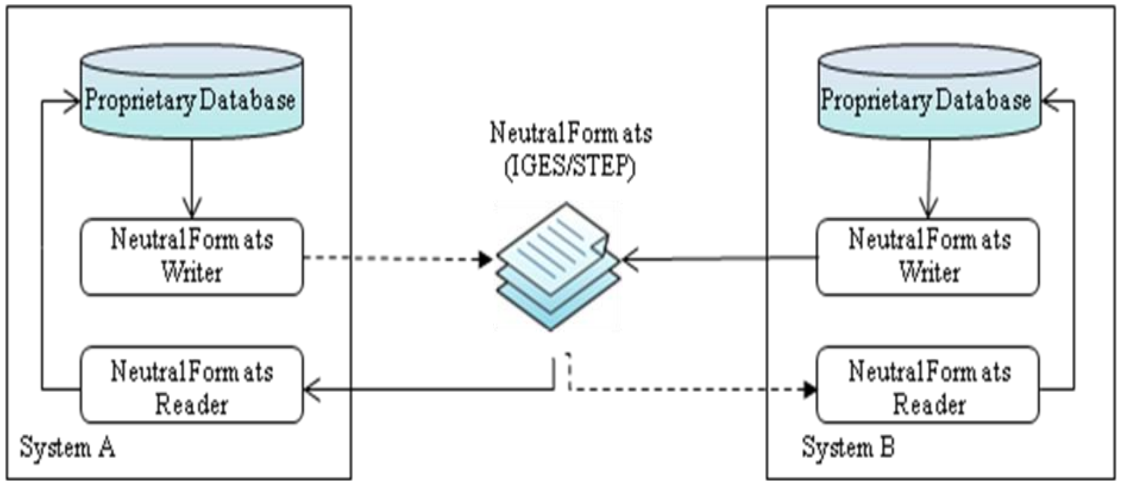


Figure 1.3. Process of data exchange

While this seems feasible and should not introduce inaccuracies, the processes of data exchange are afflicted by several problems, such as information loss (how entities are created), total loss of design intent (symmetry, equality, parallelism, perpendicularity, concentricity, etc.), and change of data (analytic surfaces such as cones and planes changed into NURBS surfaces or spline surfaces).

The body of the flash drive without the sleeve, shown in Figure 1.4 (Part 4 in Figure 1.2), is constructed from 10 surfaces. Five of the original 10 surfaces topologically have changed after converting the surfaces into CATIA's implementation of IGES. Cracks on the left side of the top surface are introduced. Recall that model rendering is a visualization of the geometries of the model that are usually sent to downstream applications such as Rapid Prototyping, CAE, and CAM packages. If this model is to be sent to a CAM for machining, surfaces have to be continuous or tangent to one another. CAE systems (FEA/CFD) will also be affected by the gaps and cracks since an accurate tessellated model to simulate stress, strains, and temperature differences cannot be achieved. Therefore, neither horizontal migration (CAD to CAD) nor vertical migration (CAD to Downstream Applications) can be subjected to these issues.

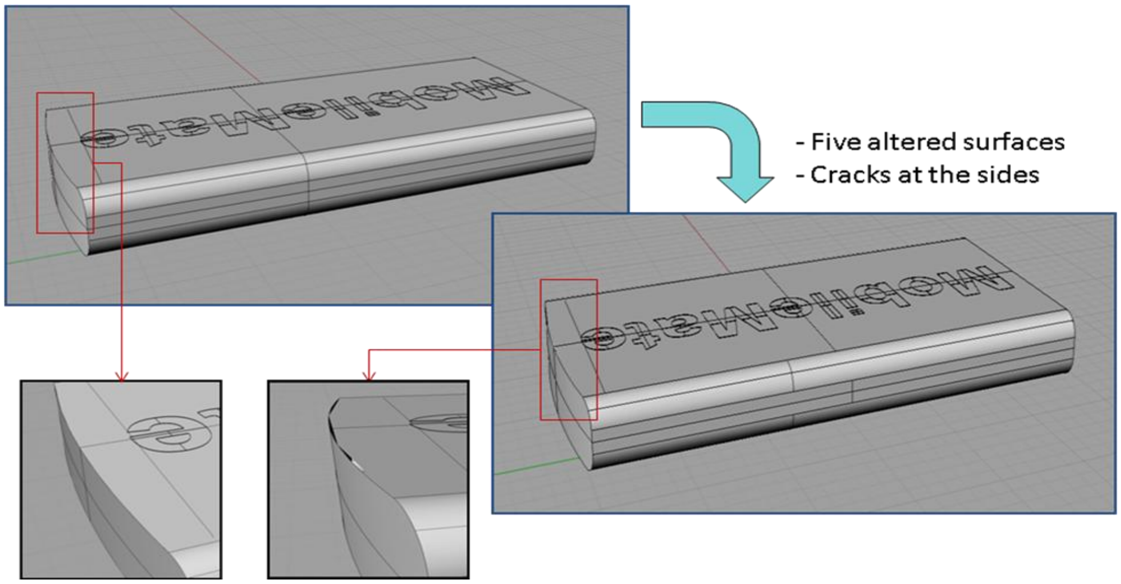


Figure 1.4. Model converted to CATIA implementation of IGES

To remedy problems in a CAD model, such as cracks, gaps, overlaps, self-intersecting surfaces, and change in topologies, CAD designers at the receiving ends will have to fix, patch, or repair the models manually or use specialized software [2,3]. However, patching up and fixing CAD models based on NURBS will create another dimension to the problem [4]. For example, to correct the gaps and cracks that appear in Figure 1.4, three possible approaches can be used.

The first is to generate a small patch that fills the gap. Many CAD systems will provide a patch operator. However, this does not work if the objective is to insert a tiny patch between two large surfaces. Bulged, rounded, or sometimes unpredictable patches will be created because continuity conditions require the use of existing cross-boundary derivatives of the two surfaces [4,5]. Figure 1.5 shows the result from the patch operator in Rhino. The produced patch disfigures the model to point of being hardly recognizable.

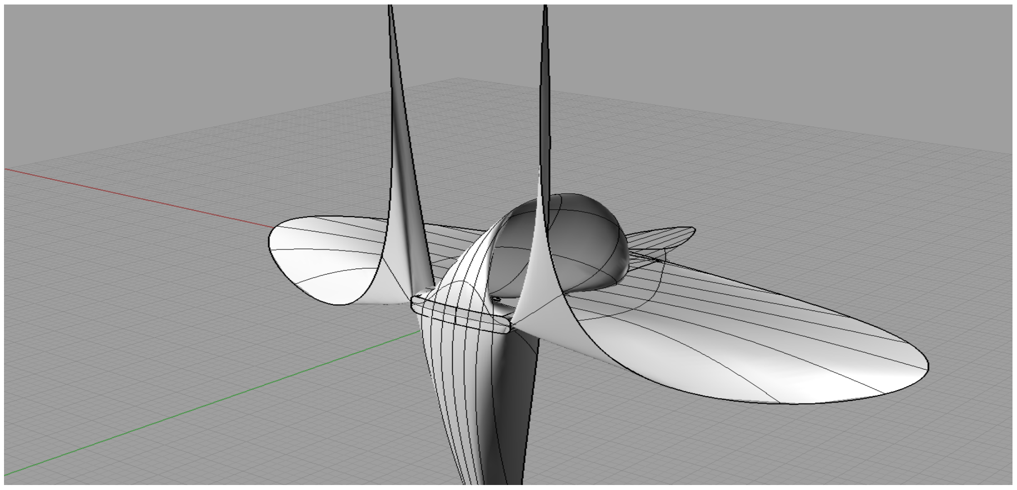


Figure 1.5. Patch operator in Rhino

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 [5]. The third approach is to extract the boundary curves of the two offending surfaces manually and rule another long and thin surface between the two surfaces as in Figure 1.6. In this case it is feasible since we have two boundary curves of the same length and direction. If the boundaries are of different lengths and directions, then the designer must creatively come up with a patch that will fit. The designer also needs to account for continuities among the three surfaces to ensure smoothness.

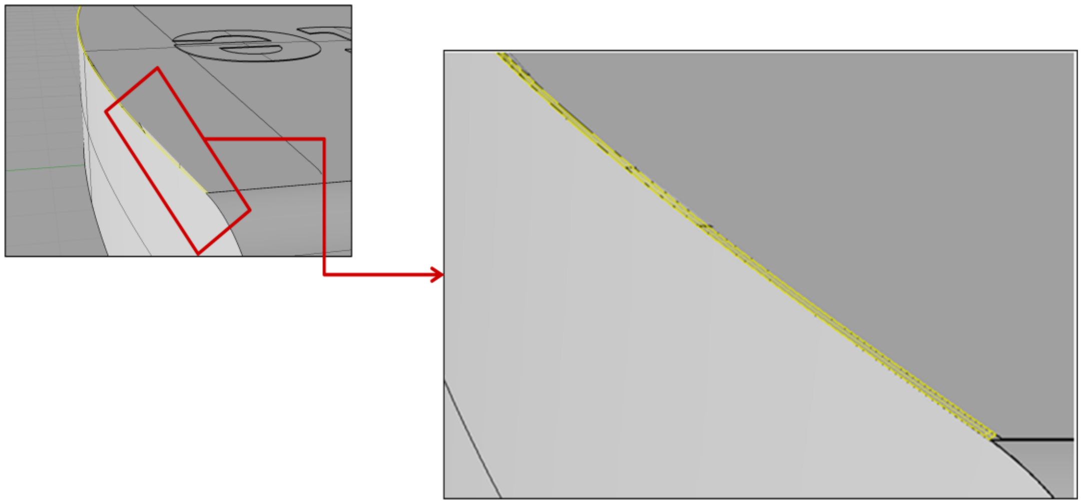


Figure 1.6. Ruled surface (thin patch)

In the best-case scenario, one can ignore the fact that this model might go to a third and fourth system, resulting in more repairs. The design intent, however, has never made it from the first conversion of the model.

If CAD systems can preserve the designer's intent and endow sufficient knowledge in each NURBS entity, the receiving systems can *regenerate* the whole model or parts of the model without having to repair it. Systems can also *verify* that the design is the intended design. Other advantages that can be achieved include but are not limited to: *evaluating* the chosen design, *mining and browsing* the whole and partial designing process, *tradeoffs* between *alternative* designs and priorities [6], *justification* of why one choice is considered over another, and *documentation* that can be valuable for both the producers (designers) and consumers (manufacturers).

1.1.1 Why isn't Design Intent Supported Yet?

If design intent in the modeling process of an artifact has such potential values, and there is widespread agreement on the need to support it [6-8], then why is it not in widespread use? There are a number of reasons why CAD systems in existence do not support the exchange of design intent.

One difficulty, despite a good deal of research, is the capture and supporting of design intent in general. Intrusive recording of design intent can be time consuming and expensive [9]. Designers will be reluctant to spend the time to document, among many other parameters, the type or *what* NURBS entity they are creating, the origin or *how* it was created, and the destination or *where* it is going to be used. If deadlines are an issue, designers will tend to resist spending time on documenting and recording their intentions especially if the difference between a project that meets its deadlines and is completed versus one that did not meet deadlines results in cancellation [10]. In their survey [8], Tang et al. stated that 90% of the participants agree on the important of design intent, however, 60% do not care to document it because of time constraints. Capturing designer intent for a NURBS model should be non-intrusive but should be modifiable if the designer finds the need to change it.

Another issue affecting the likelihood of designers to record their intent is that designers do not gain immediate benefits from the extra efforts put into recording their intent [10]. This provides little incentive to take the extra time and effort to record the design intent intrusively as stated previously. It is common for companies to outsource the modeling of an artifact for reasons such as the developing company's superior knowledge of the development of that particular object. When it is time to deliver the model, it could be in a neutral format (IGES or STEP), and the model could suffer from many issues as stated before. It is the receiver of the CAD model who bears the burden of not supporting and exchanging the design intent. Supported design intent in NURBS modeling should provide incentives for the designers (not only for the receivers) in order to make sure that their intent is recorded correctly. Immediate benefits include, for example, "what-if scenarios" that can be supported by design intent.

Another reason for not supporting design intent is the lack of standardized design intent. The survey in [8] stated that 40% of the designers do not document design intent because of non-standardization. Design intent is not easy to standardize in general since it is difficult to quantify. Geometries are easy to standardize since they are well-defined. Design intent in a CAD environment has to be well-defined and structured in order to make it standard. If we can find concise yet expandable structured design intent to support, the idea to standardize it can emerge to the surface.   Fortunately, structuring design intent in NURBS modeling can be achieved if NURBS entities are tagged with some extra knowledge.

A final reason is the "push" for standardization. The first version of IGES as a standard to exchange and transfer designs among dissimilar CAD systems was released in 1981. However, it was not widely used until years later (1988) when the U.S. Department of Defense (DoD) required that all engineering drawings, circuit diagrams, etc. be delivered in electronic form, and specifically in the IGES format [11]. By pushing these requirements on subcontractors, all Computer-Aided technologies (CAx) vendors who want to market their product to DoD subcontractors were pushed to support IGES format files. If key players in the market ask for design intent to be supported and standardized, then CAx vendors will follow. With the advances in technology, CAD models will only get more and more complicated and knowledge intensive; with it, the need to support design intent will only grow.

Literatures relevant to the topic agree on the need to support design intent; however, how to support and exchange it among the incompatible CAD systems is not a trivial problem to solve [7]. Chapters 3, 4, and 5 discuss the solution. They provide the needed theory as well as a prototype system to illustrate that with the right approach, a *well-defined* and *structured* design intent can be captured automatically to be exchanged among CAD systems environments and lay the ground for possible future standardization of design intent.

1.2 Prior Works

Research in the area of design intent, knowledge-guided systems, and interoperability among systems has been active for the past two decades. Although thorough research on relevant topics can be performed, it would be tedious to cover all the work related to the problem of interest in details. Therefore, we will primarily focus on the most significant topics and contributions. Section 1.2.1 discusses research in the area of design intent. Section 1.2.2 details the works on CAD models repair. Section 1.2.3 provides a literature review on attempts to exchange design intent. Finally section 1.2.4 summarizes the research and history of CAD models' exchange including both standards and non-standards formats.

1.2.1 Design Intent

Research in design intent covers a wide range of fields and topics resulting in a large body of papers and approaches. Researchers from software engineering, civil engineering, artificial intelligence, mechanical design, knowledge-based engineering, design manufacturing, and cognitive science are among the research communities that relate to this concept. Research is quite diverse in both depth and involvement. Some pursue this concept at a theoretical level while some others create prototype systems that are able to capture and retrieve designers' intent. A number of authors from different fields try to formalize definitions to terms that are used inconsistently and interchangeably, e.g., design intent, design rationale, and design history, while others try to demystify the distinctions and overlaps among these terms [12]. Below are selective definitions from different fields that researchers in the area of design intent are interested.

Gurber, from the Knowledge-based and Systems Engineering [13] argues that design rationale and intent can be constructed form design history. Garcia [14] states that "design rationale can be viewed as the design history – the sequence of events that occurred while performing the design." Conklin and Yakemovic from Human-Computer Interactions [9] define design rationale as the reason for constructing an artifact in certain way.

From the Artificial Intelligence standpoint, Sim and Dufy [15] describe design intent as "the reasoning and argument that lead to the final decision of how the design intent is achieved." Ullman [16] views design history as "a record of the rationale behind design decisions and of the intent of designers." Louridas and Loucopoulos from Software Engineering [17] argue that "design rationale aims at capturing the why behind the how in design, i.e., on externalizing and documenting the reasons behind design decisions and artifact features."

Iyer and Mills [12] define this concept most clearly and comprehensively in their survey on design intent. They state that "design intent is application, domain and context dependent knowledge that describes design space, represents design alternatives and processes history, justifies design solutions and decisions and determines the characteristics of features and entities and the relationships among them" [12].

While these definitions span many fields, have merits and overlaps, the focus of this section is to give an overview of the most influential contributions on *general* design intent and rationale.

1.2.1.1 Existing Design Intent System

There are numerous design intent tools that have been developed since the early 1980's, but only a handful have made it into practical use in industry. The first notable work on design intent is by the design theorists Rittel and Kunz in 1970 [18]. They developed Issue-Based Information System (IBIS) tool to structure the intent as argumentation. IBIS is only an argumentative notation, not software. However, it was very influential in the early developments of design intent and rationale systems; several prototypes have been proposed and developed as variations of IBIS notations. IBIS works by creating "issue-maps." The map consists of an issue (design question) specifying the main problem with resolutions answering the design question and arguments supporting or objecting to these positions, as in Figure 1.7. It supports 9 relations (Generalized, Specialized, Questions, etc.).

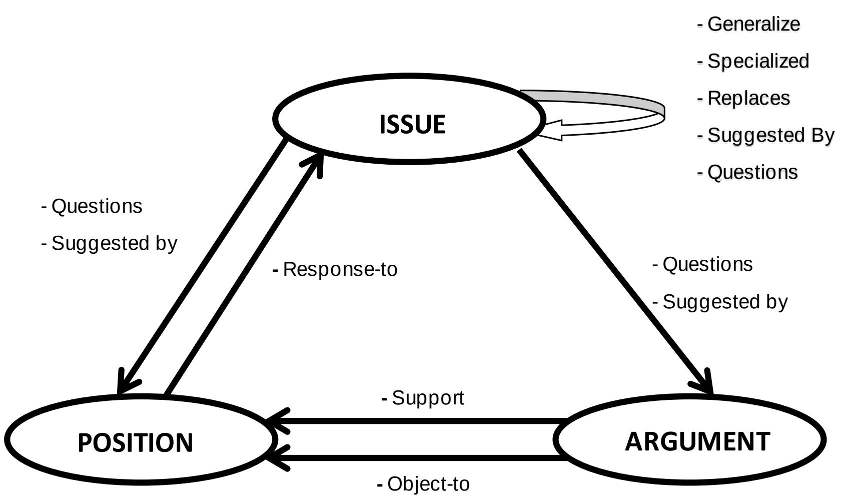


Figure 1.7. Relationship of issue, position, and argument

Two immediate variations of IBIS are: gIBIS (for graphical IBIS), which has been used in an influential hypertext implementation, and itIBIS (for indented text IBIS), which uses only indented text for the representation of IBIS trees [19]. gIBIS and itIBIS are the earliest case study of the use of design intent and rationale in a real industrial design setting where they used it at NCR. Two extensions of IBIS are Procedural Hierarchy of Issues (PHI) [20] and Potts and Bruns [21]. Decision Representation Language (DRL) [22] extends the Potts and Bruns model. As indicated by its name, DRL is an expressive language that focuses more on the representation of decision making and its rationale than on design rationale. Drawbacks for these four systems include, but are not limited to, user-intervention knowledge capture and inability to retrieve captured knowledge.

REpresentation and MAintenance of Process Knowledge (REMAP) [23] is another system that is based on IBIS. It is a more complex system that enables users to retrieve knowledge by a set of queries. Knowledge capture capability still requires user intervention with REMAP. Other systems that are implemented based on the IBIS methodology include Potts and Bruns [24], IBIS-Style browser [25], KBDS-IBIS [26] for chemical plants, and DRAMA [27]. These have automotive capture of knowledge and intent, but no knowledge retrieval methods are supported. CRACK [28], VIEWPOINTS [28], and JANUS [29] are used specifically for kitchen design. They follow a hybrid approach to retrieve knowledge. Hyper-Object Substrate (HOS) [30] permits triggered intent and knowledge retrieval. It allows the capture of data informally during the design process and converts it into a useable form. PHIDIAS [30], which is based on PHI, is also a hyperemia system. It represents design intent and rationale as nodes and links. Both HOS and PHIDIAS capture knowledge automatically.

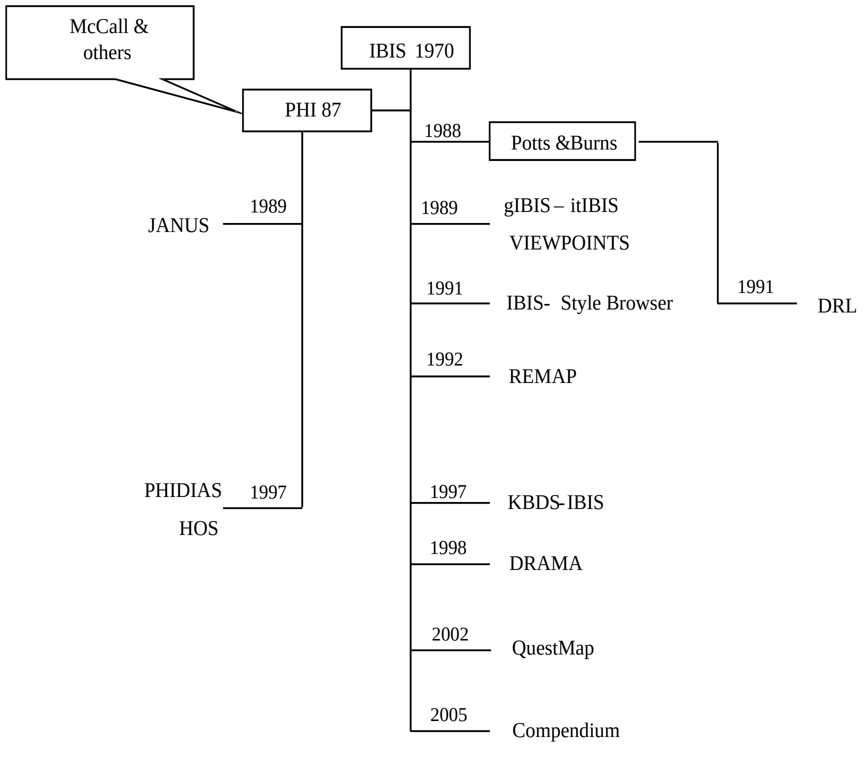


Figure 1.8. Chronological order of IBIS-based approaches

QuestMap [31] by Soft Bicycle Co. and Compendium [31] by Verizon Research Lab are the two of the latest systems that are based on the IBIS methodology. In 2005 Verizon research lab released Compendium 2.0 as open source tools that is intended to replace QuestMap. Both help users visualize the connections between people, ideas, and information at multiple levels of discussions and debates. Figure 8 shows the approaches that are based on IBIS on a chronological order. Figure 1.8 gives the chronological order of IBIS-based approaches mentioned above.

Other systems are built on different notation, such as Questions, Options, and Criteria (QOC) [32]. QOC focuses on alternative-features of an artifact as the main objective of a discussion. A QOC process starts by establishing Questions about an artifact followed by Options that address the questions. The options are evaluated and selected based on specific Criteria that describe the goals to be fulfilled. Similar to some previous approaches, QOC lacks the abilities to capture and retrieve knowledge and intent without designers' intervention.

Design History Tool [33] for mechanical design records the design history from the specification and constraints to the detailed designing process. One appealing feature of this system is the playback facility that permits the user to retrieve information about the design. However, this system depends heavily on the design constraints that are based on the designer's knowledge of the domain. AIDEMS [34] asks the user to create product specifications at the beginning of the design. The specifications are then refined at each level of the designing process. Thus, intent emerges during the design process and theoretically captured during that time. Design Rationale Authoring and Retrieval System (DRARS) and TED [35] use variations of QOC. In DRARS, it is the user's responsibility to give descriptive names to the system's object (claims, questions, and answers).

Functional Representation (FR) [36] is an approach for a formal representation of the artifact's functioning. It supports knowledge generation, simulation, design verification, and case-based design. The focus of FR is to automate the design intent support. Another approach that automates the design intent support is Generative Design Rationale [37], which proposes reasoning ontology and processes for use by computers. The idea of actively generating documentation is pursued in Active Design Documents (ADD) [38]. ADD allows the user to verify the knowledge required for justifying each decision. However, it only provides a read-only interface to navigate through the decisions and reasoning associated with an artifact. MultiADD [39] is an extension to ADD that is proposed to deal with group design.

Some of the new tools are more geared towards a specific field than being generic. View and Beyond (V&B) is a collection of methods proposed by Clements et al. [40] to document software architecture design. They argue the importance of documenting why the architecture is the way it is and the justification of decisions. Architecture Rationalization Method (ARM) [41] uses qualitative and quantitative rationale in design reasoning. It selects an optimal solution by eliminating inferior design alternatives, using the quantification of the design options. Tyree and Akerman [42] provide a template that captures certain types of information as design rationale. Software Engineering Using RATionale (SEURAT) [43] is based on DRL. SEURAT ensures that the system building is complete and consistent by providing insight into the reasons behind the choices made during the design process and implementation of software systems.

The need for capturing design intent and rationale has been recognized in software architecture design and has led to several efforts to standardize it. The IEEE 1471-2000 standard [44] provides a definition of design rationale and discusses its importance. Even though this is a step in the right direction, the real work still needs to be done. The standardization needs further elaborations on the nature of design intents and how they might be captured, retrieved, and represented.

1.2.2 CAD Models Repair

There have been various attempts to repair poor quality exchanged CAD models, both in research institutions and commercially. The most relevant attempt to our work is the approach by Jeongsam and Han [45]. They propose to correct a CAD model based on the design history. They claim that their system Q-Raider can repair six types of CAD model: tiny faces, narrow regions, non-tangent faces, narrow steps, sharp face angles, and narrow spaces. Even though this is called history-based repair, the corrected model can collapse if it is reconstructed solely on the design history. They have to couple the repair with other parameters, constraints, and boundary representation (B-Rep), which make the repair process very delicate.

Other notable approaches are based the B-Rep of a model. STEP application protocol AP203 [46] allows the transfer of B-Rep. This makes it appealing to use to repair transferred models. However, correcting CAD models based on B-Rep suffers from drawbacks, such as no consideration of the designer intent. B-Rep's lack of concern with designer intent is a consequence of the model being a “dumb” solid that has no parameters, constraints, reason for a particular choice, or the engineering data that are big part of the design intent. Repairing the model based on B-Rep while ignoring the design intent could lead to failure of the geometric structure of the transferred model. This also will create another problem if the model is to be reused or transferred back “upstream” to the creating CAD system. Another drawback is the computation and complex data structure that require large memory to repair even simple models with B-Rep. Literature reveals three classes of B-Rep approaches: exact B-Rep, faceted B-Rep, and boundary curve-based approach.

Exact B-Rep approaches include the work of Hoffman et al [47]. They correct errors in a CAD model by analyzing the topological and geometric elements of the boundary representation (B-Rep) of 3D shapes. They proposed the architecture for a master model to correct problems arising from different geometric tolerances between different downstream applications. Gu et al. [48] correct topological errors in CAD models by storing and using a model object tree that represents the topological entities and use the tree to repair the model. Their focus is on correcting the topological errors rather than on geometrical errors since errors in the topology are catastrophic [48].

Faceted B-Rep approach is faster than the exact B-Rep approach that is used in [47,48]. It approximates the B-Rep model with a set of polyhedrons. One of the early works on faceted B-Rep is by Rock et al. [49]. They present efficient data structures for computing exact matches between polygon edges in order to reconstruct the topology of the model. Makela et al. [50] use local techniques to fill cracks in the model surface by triangles. Turk et al. [51] remove error resulting from overlapped polygons by clipping them. Barequet et al. [52] describe a globally consistent approach for identifying and filling holes. However, when a large number of cracks is involved, the technique can result in a very large number of polygons to describe the model. In [53], Barequet et al. describe a hashing algorithm that corrects errors in a model by stitching the small gaps between polygons.

The boundary curve-based approach is restricted to 3D shapes that are constructed from surfaces. Works based on this approach used the idea of simplifying the original object's mesh and then reconstructing a smooth surface on top of it. The original idea of simplification is to produce a smaller model to aid the analysis and display of surfaces [54, 55]. Steinbrenner et al. [56] correct gaps and overlaps between surfaces through edge-splitting and merging of the boundary curves.

Commercially, some companies find this problem lucrative and implement commercial software packages to correct errors in CAD models. CAD Doctor by Elysium Inc.[2], CADfix [3] by ITI, and TransMagic [57] are a few examples. Many of the commercial software in the area are based on B-Rep approach, which suffers from the drawbacks stated earlier.

1.2.3 Design Intent Exchange

One of the earliest attempts to exchange parts of the design intent was by Hoffmann et al. [58]. The authors suggested a method of exchanging CAD models in terms of their construction history using their EREP (Editable REPresentation), which is a neutral format for expressing form features, parametrization and constraints. Although EREP presents the foundation for the exchange of parametric information, there are a number of key technical problems with this approach that have been discussed in a number of papers [59].

Another early attempt to exchange more than the geometries of CAD models is ENGEN (Enabling Next GENeration design) [60] by PDES Inc. It concentrates on the exchange of geometric constraints, and demonstrates the exchange of constrained 2D profile data. CHAPS (Construction History And ParametricS) [61] is another recent project by the same company. It mainly concentrates on using new draft ISO 10303 capabilities to transfer construction history or procedural models without explicit parameters or constraints. Another approach that follows the exchange of procedural models is proposed in [62]. It uses the journal file created by CAD systems, which should contain a record of every action of the system user. The exchange of procedural models provides some level of flexibility to edit the model in the receiving system but omits certain information relating to design intent that may be important in maintaining product functionality after a modification.

Bettig et al. [63] also attempt to exchange constraints for parametric modeling. They proposed a standard set of geometric constraints for parametric modeling and data exchange. They defined explicit constraints for the relationships between all the geometric entity data types specified in ISO 10303-42[64], STEP geometric, and topological representation of CAD models.

Rappoport [65] tackles the problem by focusing more on the consistency of pure geometry between the original and the exchanged models. While geometrically this is satisfactory, design intent and characteristics of the original design usually get lost by his approach.

Kim et al. [66] suggested a theoretical approach to extract the design intent information using Parts 55 [67], Part 108 [68], and Part 111 [69] of STEP. These are recently published Parts of ISO 10303 that are not yet widely used in current translators and commercial CAD systems. A drawback of this method is that explicit design constraints are big part of extracting the design intent in their approach. For example, some widely used CAD systems do not allow access to constraint information used in the model [61]. Without access to these constraints, the whole process may halt. Pratt et al. [70,71] also used these new Parts of ISO 10303 as a solution to exchange the parametric data of CAD models. Both [61] and [71] call for standardizing data exchange of CAD models with design intent.

1.2.4 CAD Model's Data Exchange

In the early 1970's only a few CAD systems were implemented, and only few organizations could justify the purchase of a CAD system or machine, which had an average price tag of $125,000. With only a few CAD vendors in existence and only a few users applying CAD technologies, it was very common for organizations and manufacturers to agree on the use of a single CAD system. Others, like Boeing and General Electric, faced with the need to migrate and exchange data among the different CAD systems they were using and implemented their own in-house CAD and translators. The translations had to be direct, as in Figure 1.9(a), which shows the increasing complexity with the addition of more incompatible CAD systems.

By the end of the 1970s, the need for efficient means to exchange modeling data with incompatible CAD system was evident. Advances in technologies made the CAD system a valuable developing tool for organizations and manufacturers. However, most organizations find it difficult to enforce the use of a common CAD tool within their organization. It is much harder to enforce it across multiple supply chains and among joint venture partners. At the same time, CAD vendors guarded their proprietary data format in order to not lose the competitive advantages. These needs and concerns have driven the development of a standard for a neutral format as illustrated in Figure 1.9 (b). It relives the manufacturers from developing and maintaining several system-specific translators as in Figure 1.9 (a) and grants the CAD vendors the choice not to disclose their data format.

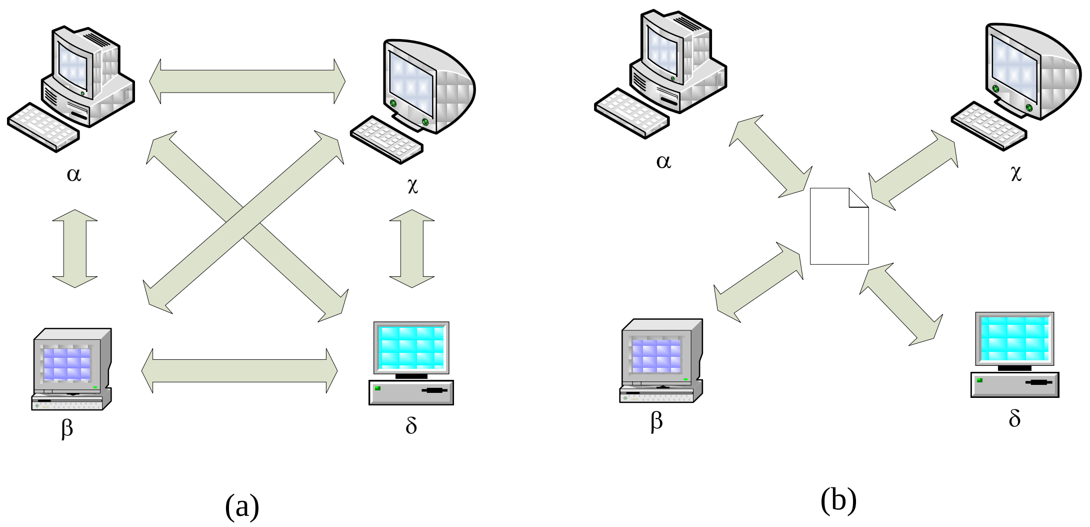


Figure 1.9. Ways to exchange CAD data

1.2.4.1 Efforts for Standardization

In the early 1970's most of the efforts were focused on the foundational description data, which could be exchanged and shared among CAD systems. The

Computer-Aided Manufacturing - International Inc. (CAM-I) organization contributed greatly to the formal description of B-Rep data [72]. However, it did not provide an exchange mechanism. CAM-I also investigated the mathematical representation of standard geometry and topology and then submitted the results to ANSI committee Y14.26 (Computer Aided Preparation of Product Definition Data) for standardization.

The U.S. Air Force also made a significant contribution to the standardization of product data exchange. The funded Integrated Computer Aided Manufacturing (ICAM) program developed new formal manufacturing technologies such as IDEF0 (ICAM DEFinition) for activities modeling, IDEF1 for information modeling, and IDEF1X for data modeling [73]. Contractors were required to use these new engineering methodologies as if they were standards.

The Product Definition Data Interface (PDDI) program, also funded by the U.S. Air Force, contributed to the advancements of the tools and methodologies that were key components in subsequent standards [74], most notably ISO 10303 (STEP). PDDI was a research exercise that focused on developing a mechanism to allow the complete exchange of product model data directly among computer applications. It also contributed to standardization of data exchange by developing a modeling language that later would be used in developing EXPRESS [75], an exchange file format used in STEP to separate the exchanged data from its definition.

International efforts, largely European, also contributed greatly to the process of standardization for CAD model exchanges. The earliest effort was in 1977 when the European Association of Aerospace Industries (AECMA) developed an intermediate format to exchange shape representation between different systems. The format was very basic and only supported the exchange of limited basic surface types. When more complex surfaces representations were developed in subsequent years, the format did not perform as needed causing it to be abandoned [74]. In 1984, the United Kingdom submitted the "AECMA Report of the Geometry Data Exchange Study Group" [76] to the International Organization for Standardization (ISO) as an effort to develop an international product model data standard.

The West Germany Organization of the Automotive Industry (VDA for "Verband der Automobilindustrie") standardized VDA-FS [77] in 1982. VDA-FS was developed to address the exchange of free form surfaces and curves that were needed by the automotive industry during that time. The FS, which stands for "Flächen Schnittstelle," translates to "surface translation format."

The French standard, Standard for Exchange and Transfer (SET) [78], was developed at Aerospatiale in 1983 to address the difficulties using IGES.

Like AECMA, both VDA-FS and SET were submitted to ISO in the same year (1984) to contribute toward the international efforts for standardization of product model data. The international standard "STandard for the Exchange of Product model data" (STEP) was released a decade later in (1994).

1.2.4.2 IGES: Initial Graphic Exchange Specification [79]

IGES is a national (ANSI) standard for CAD data exchange. The origin of IGES has a history associated with it. It was the result of combined efforts form CAD vendors, interests of users, and the standards committee.

In 1979, The Society of Manufacturing Engineers held a two-day meeting. On the first day of the meeting, a CAD user from General Electric (GE) asked the CAD vendors panel about possibility of working together to enable a common neutral exchange mechanism [74]. This proposition was very threatening to the CAD vendors' panel, which included ComputerVision, Applicon and Gerber, since sharing their proprietary data formats and the databases structures would put them in less competitive advantage. On the other hand, there were large and important Navy contracts, and every CAD vendor wanted to look responsive to customer requirements. In response to the suggestion, the vendors' panel stated that they would share their database structures if Boeing and GE would supply the CAD translators they had already developed for their organizations' uses.

By the end of the first day of the meeting, Air Force ICAM agreed to begin the effort to implement a common translator with a $50,000 contract. The National Institute of Standards and Technology (known at the time as National Bureau of Standards) agreed to support the process. The two big CAD vendors, Applicon and ComputerVision, agreed to disclose their internal databases. GE offered its neutral database and its translator. Boeing offered the structure of its Computer Integrated Information Network (CIIN) database and its existing translator.

The next meeting day, the name, Initial Graphic Exchange Specification, was chosen. It was carefully selected to avoid any suggestion of a database standard that would compete with the proprietary databases and to avoid offending ANSI committee Y14.26, that had worked for years towards Y14.26 standard [79]. In January 1980, the first draft was submitted to the ANSI Y14.26 committee for standardization that released version one of IGES in 1981 after attaching the work on B-Rep that was funded by CAM-I.

As stated, IGES was released in 1981; however, it was not widely used until years later (1988) when the DoD required that all engineering drawings, circuit diagrams, etc. be delivered in electronically, and specifically in the IGES format [14]. These mandatory requirements forced all CAx venders who wanted to market their product to DoD subcontractors to support IGES file formats. IGES 2.0 was released in 1982, IGES 3.0 in 1987, IGES 4.0 in 1988, IGES 5.0 in mid 1991 [80], and IGES 5.3, the last version, in 1996.

1.2.4.2.1 Shortcomings of IGES

IGES is a good and a practical initial solution for CAD data exchange. However, it suffers from many drawbacks. A thorough report [81] by PDDI defines IGES's shortcomings. The report details the problem of representing and capturing the same information in several ways. For example, a curve may be tagged as a NURBS curve (IGES 126) or a spline curve (IGES 112). The reason for this discrepancy lies in the preferences by which companies implement standards. Over the years CAD vendors have abused the IGES file format by making variants or "flavors" to suit their specific needs. This annoying problem of "flavoring" [80] made proper interpretation of the exchanged entity be based on the particular flavor of the pre- and post-processors. One look at the Rhinoceros IGES export options dialog box in Figure 1.10 illustrates the magnitude of the problem. The figure shows a portion of the 60 flavors of IGES that

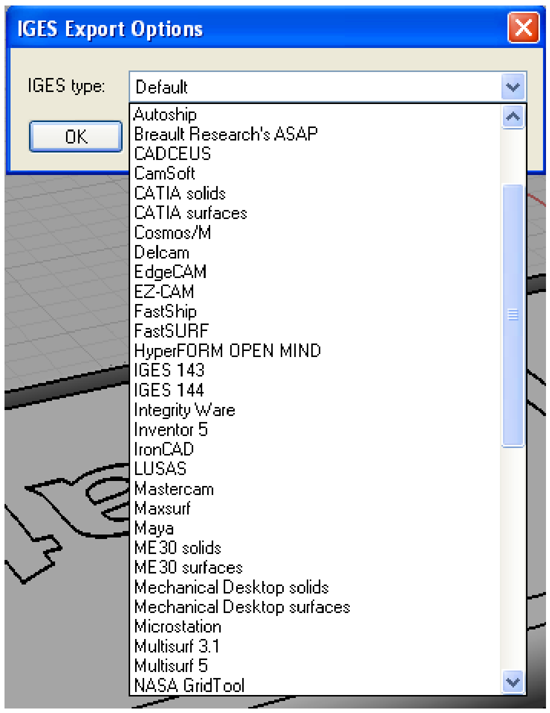


Figure 1.10. Flavoring of IGES in Rhino

allows user to specify, and the list will just grow in time. Flavoring a standard is oxymoronic as it defies the very purpose of having it in the first place.

Peter Wilson [81] also reported the problem of partial implementation of IGES, as illustrated in Figure 1.11. Vendors selected and implemented only portions of the standards that are deemed most important. This "subsetting," which allows for full exchange between two systems, seems impossible without agreement on what to exchange. To overcome the unpredictability of translations, with the help of the Department of Defense Computer-Aided Acquisition and Lifecycle Support program, the standard committee formalized the subsets of IGES entities. However, it was insufficient since accurate exchange required additional instructions and information.

Loss of modeling information is another drawback of IGES when migrating a model from one CAD system to another dissimilar CAD system with different capabilities. Other drawbacks of IGES include upward compatibility, file size, processing time, and inability to capture product information [80, 81]. The shortcomings and evolutions of IGES justified a shift toward a better standard, STEP.

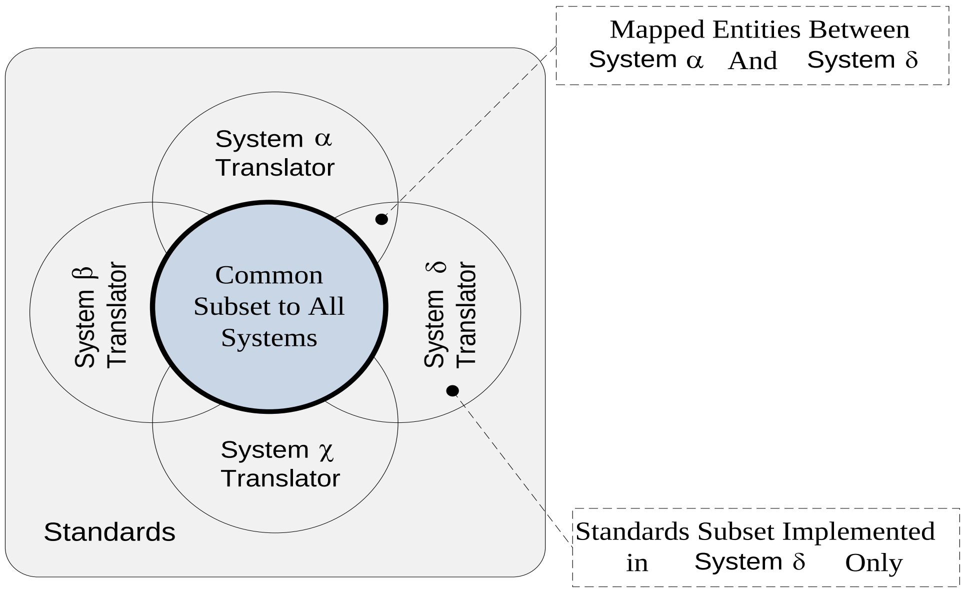

Mapped Entities Between System α And System δ
System α Translator
System β Translator
Common Subset to All Systems
System δ Translator
System χ Translator
Standards
Standards Subset Implemented in System δ Only

Figure 1.11. Partial implementations of standards (subsetting)

1.2.4.3 STEP: *ST*andard for the *E*xchange of *P*roduct Model Data [82]

STEP (ISO 10303) is an international standard developed by ISO in 1994 to replace IGES, SET, and VDA-FS. It is meant to avoid all their drawbacks, not only by exchanging data between dissimilar CAx but also by providing mechanisms and rules to describe the product data throughout the phases of its life cycle. The information generated about a product during its life cycle can include its design, manufacture, utilization, maintenance, quality control testing, inspection and product support functions, and even disposal. The nature of this description makes STEP the largest ISO standard, which has to be implemented as parts that are published separately and over an extended period of time.

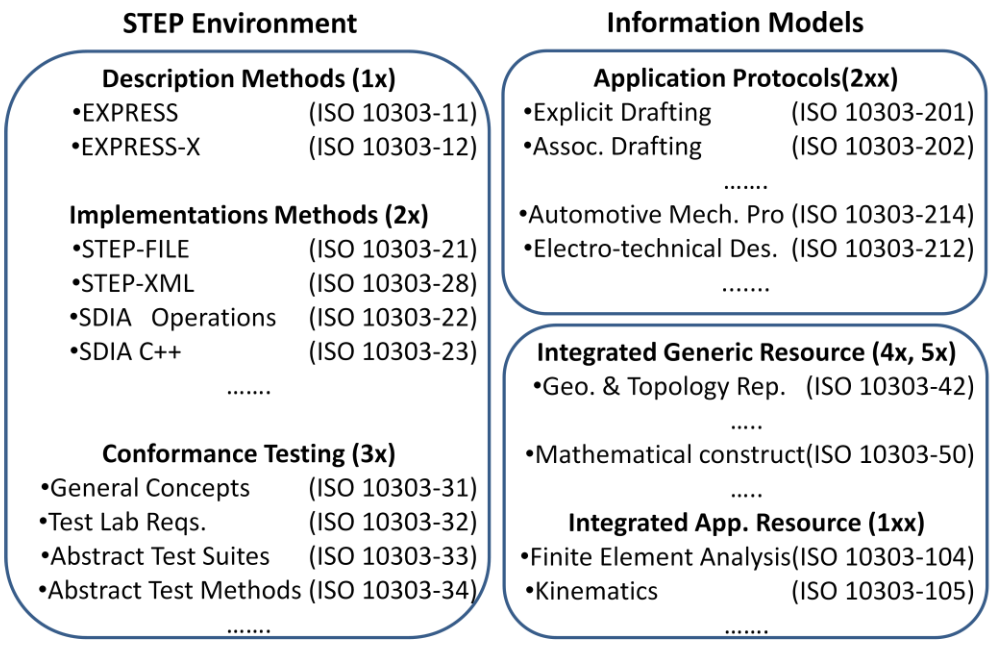

STEP Environment
Information Models
Description Methods (1x)
•EXPRESS (ISO 10303-11)
•EXPRESS-X (ISO 10303-12)
Implementations Methods (2x)
•STEP-FILE (ISO 10303-21)
•STEP-XML (ISO 10303-28)
•SDIA Operations (ISO 10303-22)
•SDIA C++ (ISO 10303-23)
....
Conformance Testing (3x)
•General Concepts (ISO 10303-31)
•Test Lab Reqs. (ISO 10303-32)
•Abstract Test Suites (ISO 10303-33)
•Abstract Test Methods (ISO 10303-34)
....
Application Protocols(2xx)
•Explicit Drafting (ISO 10303-201)
•Assoc. Drafting (ISO 10303-202)
....
•Automotive Mech. Pro (ISO 10303-214)
•Electro-technical Des. (ISO 10303-212)
....
Integrated Generic Resource (4x, 5x)
•Geo. & Topology Rep. (ISO 10303-42)
.....
•Mathematical construct(ISO 10303-50)
.....
Integrated App. Resource (1xx)
•Finite Element Analysis(ISO 10303-104)
•Kinematics (ISO 10303-105)
....

Figure 1.12. STEP: general structure

The basic parts, such as testing procedures, file formats, programming interfaces, and industry-specific information, are complete and published. The new parts that are added to the standard after the initial release of STEP are published as ISO 10303-xxx, where (xxx) represents the parts numbers. The number of the published parts is in the hundreds; however, they can be grouped in three main clusters: STEP environment parts, integrated data models parts, and top parts. Figure 1.12 shows the general structure of STEP.

STEP environment parts include Parts 1x for description methods of data, Parts 2x for implementation methods data, and Parts 3x that are used for conformance testing methodologies. They form the infrastructure of STEP and have been separated from industry-specific Parts.

STEP uses EXPRESS [75] as its description method. EXPRESS is a formal data modeling language defined in standard ISO 10303-11 to specify the representation of product information. It uses schema in which various data types can be defined together with structural constraints and algorithmic rules. The use of a formal description method enables consistency of representation and facilitates development of implementation. The encoding methods and implementations of the EXPRESS schema are defined in Parts 21, 22, and 28.

Conformance testing methodology and framework are the testing procedures and setups that are used to guide the testing processes of the implementations of a new STEP preprocessor and postprocessor. This is to ensure that products comply with the standard both syntactically and structurally.

STEP top parts group includes Parts 2xx for Application Protocols (APs), Parts 3xx for Abstract Test Suites (ATS) for APs, and Parts 4xx for Application Modules (AM) for the APs. The APs form the bulk of the standard, and are the basis for STEP product data exchange. A single AP usually covers a particular application or industry domain. For example, AP 214 focuses on automotive mechanical design processes and AP 236 is for furniture product data and project data in the mechanical field, AP 215 and AP 218 is for ship arrangements and structure, and AP 239 for product life cycle. The idea of AP helps in controlling STEP from forming a single large standard.

To support the conformance requirements, each AP uses ATS which contains the set of abstract test cases for that particular AP. This is to guarantee the compliance to the standard and to ensure quality of service to service providers, equipment manufacturers, and equipment suppliers. STEP Parts 4xx for AMs define common building blocks to create modular AP within the standard.

The last group, integrated data models, includes Parts 4x, 5x, and 1xx for the Integrated Resources (IR), and Parts 5xx for Application Integrated Constructs (AIC). Every AP defines a top data module to be used for data exchange by following specifications in AMs. The AMs are higher-level modules that are constructed by choosing generic objects defined in lower level data modules (4x, 5x, 1xx, 5xx). The common generic data models are the basis for interoperability between APs for different kinds of industries and life cycle stages. If specializations are needed for the particular application domain of the AP, they can be chosen and added to the AP. AICs define collections of common definitions that can be shared between Application Protocols.

1.2.4.4 Quasi Industry Standards

Some privately developed formats are widely used and are regarded as quasi industry standards. Drawing Exchange Format (DXF) is developed by Autodesk for enabling data interoperability between their produce, AutoCAD, and other CAx systems. Autodesk for many years did not publish specifications of DXF; however, recently they published the specifications to further the use of their formats. JT, developed by Siemens PLM Software, is not only used for CAD data exchange but also for product visualization, collaboration, and Product and Manufacturing Information (PMI). In 2009, JT specification has been accepted for publication as an ISO Publicly Available Specification.

1.3 Contributions

In this research, we investigate how knowledge-guided NURBS [83,84] can be used to support design intent of CAD models. This work contributed the following:

- Augmentation of geometric models with design knowledge for use during the design process.
- Support for design intent at the lowest level of geometry construction and incorporate it into the model's knowledge base.
- Support for knowledge acquisitions, i.e., provide various ways to gather knowledge about NURBS entities during and after the construction of these entities.
- Provision of mechanisms to mine the knowledge-base to assist a wide range of processes that are deemed important for CAD designers.
- Provision of the ability to reproduce (or adjust) CAD models instead of patching them up through a *design replay manager*.
- Enhancement of the robustness of CAD systems by providing knowledge support to computational algorithms that deal with knowledge types instead of data types.

The goal, in summary, is to show that with the appropriate approach, well defined and structured design intent can be identified and supported in a NURBS environment to provide a wide range of benefits. These benefits are expressed through a prototype system that supports design intent, addresses the issues of robustness, and enables seamless data transfer. This is an important step in the direction toward showing both the CAx standards committees and the systems implementers that the costs and effort can be justified to modify current systems' kernels to incorporate knowledge and design intents.

1.4 Organization of the Dissertation

The remainder of this dissertation presents our approach in using knowledge-guided NURBS for supporting design intent in CAD modeling. Chapter 2 presents NURBS fundamentals and model representation and storage. Chapter 3 answers fundamental questions, such as:

- What is knowledge?
- What information needs to be retained?
- What is design intent?
- How to support design intent?

Chapter 4 reports on the results from previous chapters to form the optimum knowledge to store. It details works on knowledge-guided NURBS (KGN) specifically on:

- How to build a knowledge base for KGN.
- Type of all relationships between entities (point, curves, quadrics …etc.).
- Features and functionalities that are important and must be supported in order to make a system a truly knowledge guided system. These features are organized in the four managers: knowledge acquisition, knowledge mining, design replay, and export/import managers.

- How to use a knowledge base for knowledge acquisition, design replay to reconstruct the whole design, knowledge mining to learn and support design intent, and export or import of knowledge.
- Robustness issues when using knowledge in CAD modeling.

Chapter 5 describes the system architecture and initial design of the prototype including the implementation and user interface issues. It presents case studies with results using a test object to demonstrate the effectiveness of using KGN to support design intent. Chapter 6 concludes the dissertation with summary of the research and important conclusions. Some potential and important goals for future work are also presented at the end of this chapter.

Chapter 2 --- NURBS Fundamentals

The main focus of this chapter is on NURBS, which are mathematical representations for both analytic (e.g., conics) and freeform shapes. The acronym NURBS stands for *Non-Uniform Rational B-Spline* and refers to the following.

- *Non-Uniform* refers to the parametrization of the curve which allows the use of the needed multiple knots to represent Bezier curves, among other important advantages that will be discussed in this chapter.
- *Rational* refers to the shapes' mathematical representation. It allows for both free form shapes and *exact* conic (such as parabolic curves, circles, etc.) representation in NURBS. Non-Rational B-Spline can only approximate the parabola.
- *B-Spline* are the piecewise polynomial curves that have a parametric representation. The letter B in B-Spline refers to Basis.

This chapter starts with a very brief overview of various curves and surfaces representations in section 2.1. Section 2.2 discusses the fundamentals of NURBS such as the properties and the mathematical representation of both curves and surfaces. NURBS modeling techniques are described in section 2.3. Representation and storage in a NURBS-based model is discussed in section 2.4. Data export to other CAD systems that use NURBS as their modeling kernel is discussed in Section 2.5.

2.1 Mathematical Representations of Curves and Surfaces

Curves and surfaces can mathematically be represented explicitly, implicitly, or parametrically.

2.1.1 Explicit

An explicit representation is the most basic definition of curves in two dimensions. It has the form $y = f(x)$. For example, the cubic polynomial $y = ax^3 + bx^2 + cx + d$, where a, b, c, d are constants, defines a cubic curve.

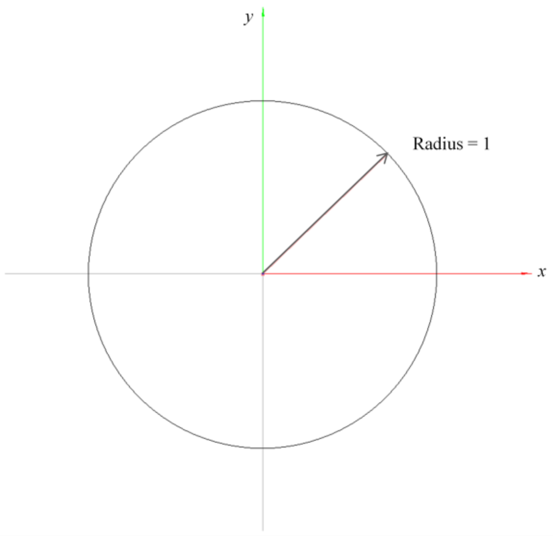


Figure 2.1. Unit circle created by cubic polynomial

Even though this form is used very often to plot graphs of functions, it is not very appealing for CAD, computer aided graphics, and geometry design for many reasons. Explicit curves and surfaces are axis dependent and single valued in y, meaning that they cannot represent closed, self loops, vertical, and multiple-valued curves. Many curves of practical importance such as circles, ellipses and the other conic sections are not permitted in this form without creating multiple cases. For example, to create the circle in Figure 2.1, it is required to divide the computations into two segments, with $y = +\sqrt{r^2 + x^2}$ for the upper half and $y = -\sqrt{r^2 + x^2}$ for the lower half. These limitations made this form unsuitable for geometric modeling since segmentations will create cases, which are a nuisance in computer programs.

2.1.2 Implicit

Implicit representations for planar curves have the form $f(x, y) = 0$ and $f(x, y, z) = 0$ for surfaces. The equations describe implicit relationships between the x and y coordinates of all the points on the curve, and between the x, y, and z coordinates for surfaces. Implicit forms can represent larger classes of curves and surfaces when compared to the explicit form. Difficulties of multiple values and vertical tangents inherent in the explicit form can be avoided using the implicit form. For example, a circle of unit radius with its center at the origin is defined implicitly by $f(x, y) = x^2 + y^2 - 1 = 0$ (Figure 2.1). Overcoming these difficulties makes these type of curves and surfaces useful in some applications, typically for low-degree cases such as degree one or two. However, the equations in the implicit representation for curves and surfaces are single non-linear equations. Calculating the x and y values for such complex higher-degree equations (curves or surfaces) is not efficient computationally since it results in solving the entire non-linear equation. Moreover, they are still axis dependent, which can cripple coordinate transformations such as translation and rotation that are required for graphical display and used intensively in CAD. Computation of points on a curve or a surface in the implicit representation is also difficult and adds to the limitations of this representation. An alternative way to avoid these limitations and difficulties is the parametric forms of curves and surfaces discussed below. A detailed comparison between implicit and parametric forms is described in [5].

2.1.3 Parametric

Parametric forms are the standard representations of curves and surfaces in CAD systems. Curves and surfaces in the parametric form use auxiliary parameters to represent the position of a point on a curve or surface. The concept of associating parameters to points on a curve or surface is very important since it can be used by many tools in CAx. A parametric curve in the xy plane is defined by $C(u) = (x(u), y(u))$. A parametric surface uses two parameters (u and v) to define it by $S(u,v) = (x(u,v), y(u,v), z(u,v))$. Every coordinate of a point on the curve is represented separately as a function of the parameter u and by the two parameters u and v on the surface. Parametrization of curves and surfaces (i.e., numbering the points along the curve) can be achieved by methods such as *uniform* and *chord-length* parametrization. The uniform parametrization evenly distributes the parameter along the curve. The chord-length approach assigns parameters based on the chord-length, or shortest linear distance on the curve. Other parametrizations such as arc-length and centripetal are also presented in literature

Parametric curves and surfaces have many properties that make them useful for geometric modeling. Dependencies on parameters rather than on the axis allow for easy transformations such as translation and rotation that are difficult in the non-parametric forms (explicit or implicit). Problems that arise in representing closed or multiple-valued curves and surfaces in explicit forms are also avoided. The same circle in Figure 2.1 represented by explicit and implicit methods can be represented parametrically by an angle parameter $u \in [0, \pi]$ by $x(u) = \cos(u)$ and $y(u) = \sin(u)$.

Furthermore, the parametric method can be easily used to describe free form shapes and three-dimensional curves. Due to these advantages, parametric representations of curves and surfaces are the most commonly used forms with the new generations of CAD systems.

2.2 NURBS Basics

NURBS, a class of parametric curves and surfaces, has been the *de facto* standard in the geometric design since the introduction of major CAD systems three decades ago. One of the greatest advantages of using NURBS in modeling is the ability to use one common mathematical form for both standard analytical objects such as a circle as seen in Figure 2.2 and free form shapes as seen in Figure 2.3. They are generalizations of (parametric) non-rational B-Spline and non-rational and rational Bezier curves and surfaces.

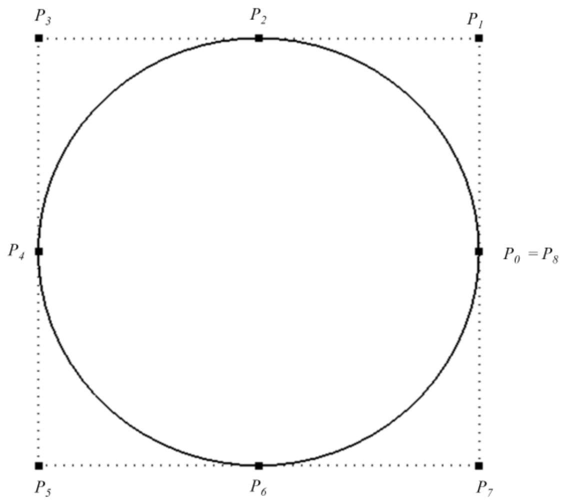


Figure 2.2. Closed NURBS curve (degree 2 circle)

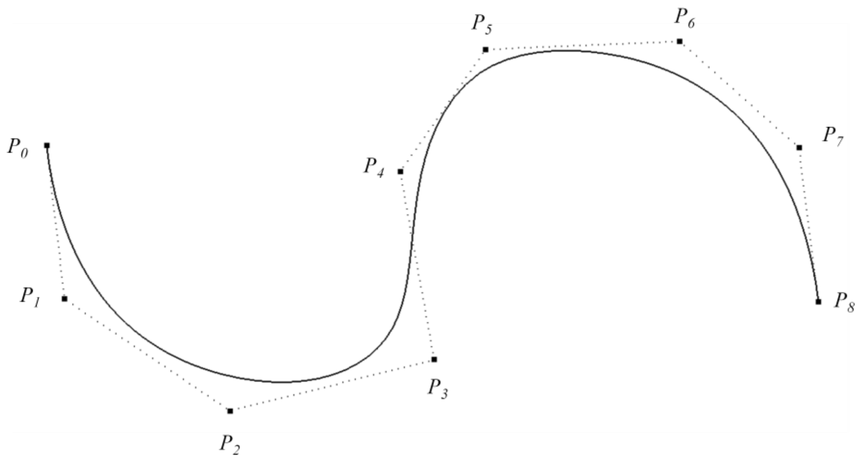


Figure 2.3. Free form NURBS curve

NURBS representations are more popular among developers of CAD systems compared with other parametric and non-parametric curves and surfaces representations. This is because of the unique properties of NURBS representation. Desirable theoretical properties, such as projective invariance, convex hull property, local scheme, controlled continuity, and visually understandable geometric characteristics, are some of NURBS' properties. From the programming standpoint, NURBS are very efficient in computation and can easily be processed by a computer and stable algorithms to achieve both accuracy and speed. They can easily represent geometrical shapes in a compact form that can be handled efficiently by simple programs. They are stable to floating points' errors and require little memory to represent any kind of curves or surface. These properties allowed NURBS to serve and perform well in a wide range of fields like designing, manufacturing, prototyping, visualization, and virtual reality. NURBS properties also made them very popular among new fields such as bio-engineering and nano-manufacturing since they can allow for easy human interactions.

NURBS are defined by a set of control points, basis (or blending) functions, knots, degrees, and weights that are associated with every control point. These five components are evaluated mathematically at a range of parameters to produce NURBS curves and surfaces.

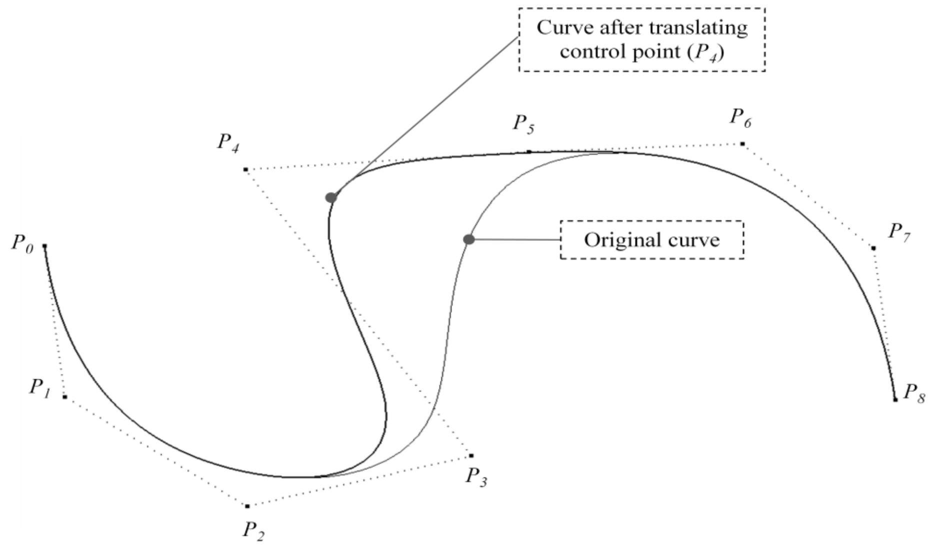


Figure 2.4. Modifying NURBS curve via control points

The set of control points are used to characterize the general shape of the curve or surface. Moving one of the control points is one of easiest ways to change the shape of a NURBS curve or surface. CAD systems usually allow users to modify NURBS entities using this option. Figure 2.4 shows the cubic curve (in Figure 2.3) after moving control point P_4. The desirable locality property of NURBS will limit the effect of a single control point to the area of the curve in the vicinity of the point. In this particular example, the effect will span 4 knots.

Basis functions, computed over the knots, are used to determine how much each point will influence the curve or surface smoothness and continuity. Figure 2.5 shows how we can use multiple knots (non-uniform knot vector) to control the continuity of NURBS. A full-multiplicity knot, when the number of duplicated knots is the same as the degree, can create a kink or sharp corner on the curve by interpolating the control point. Control point P_4 is shown in Figure 2.5.

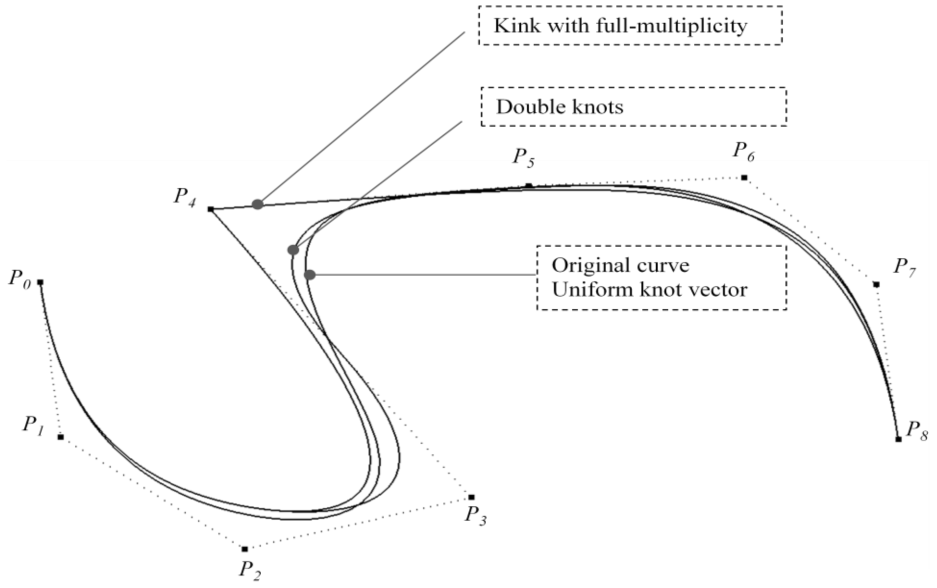


Figure 2.5. Effects of multiple knots

Degree is also one of the ways to control the shape of the NURBS curves and surfaces. Figure 2.6 shows the effect of using various degrees (2,3,4,5 and 6) with the same control points and knot vector. NURBS lines and polylines are usually degree 1 where the curve will follow the control polygon (dashed line in Figure 2.6). By looking at Figure 2.6, one can generally conclude that the lower the degree, the closer a NURBS curve follows its control polygon. More importantly, however, the degree of the curve determines the smoothness of the joins between spans. A linear curve (degree 1) has positional continuity at the join. A quadric curve (degree 2) provides tangent continuity, whereas the cubic (degree 3) curve, which is sufficient in many modeling tasks, gives curvature continuity, which incorporates the positional and tangential continuities.

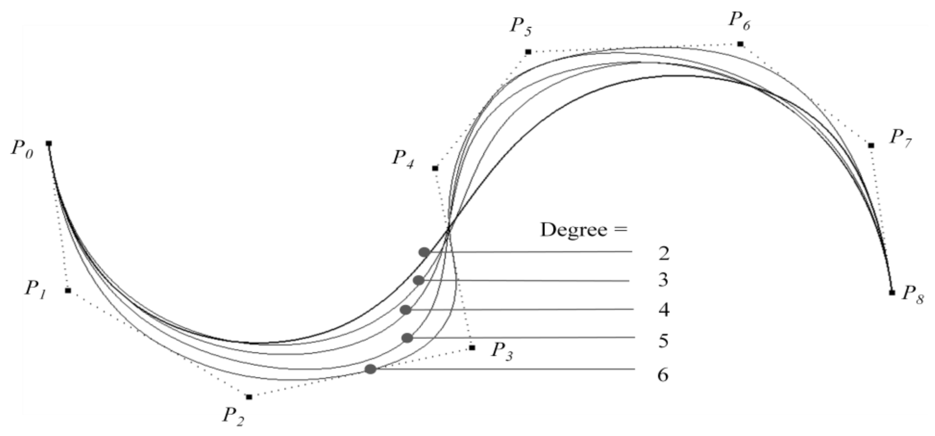


Figure 2.6. NURBS curve with different degrees, using the same control points

Weights are also used to influence how much a particular control point affects the curve or surface. Generally, the higher the weight compared to other weights, the closer a NURBS curve is to its control point as illustrated in Figure 2.7.

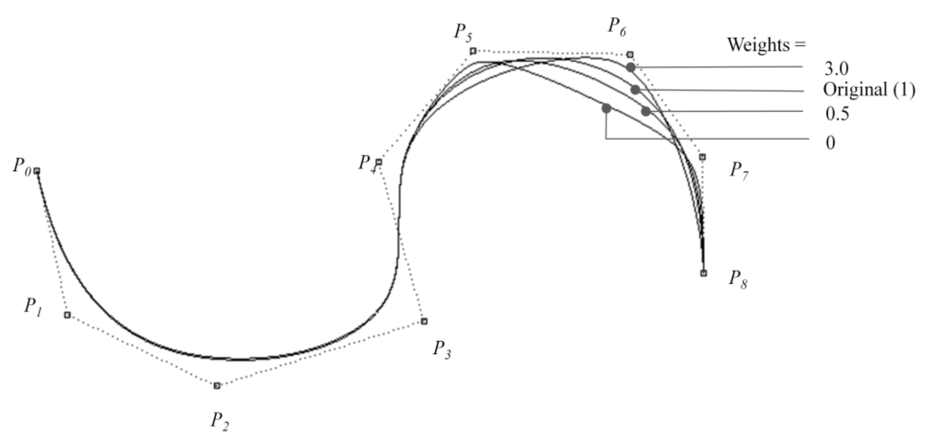


Figure 2.7. Cubic NURBS curve, with w6 varying

2.2.1 NURBS Curve

A NURBS curve $C(u)$ is vector-valued tensor-product function in the follow form [5]:

$$C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i \boldsymbol{P}_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i} \quad \text{for} \quad a \le u \le b, \tag{2.1}$$

Where $\boldsymbol{P}_i$ are the control points that form the control polygon, w_i are values of weighting functions associated with $\boldsymbol{P}_i$, n is the highest index of $(n+1)$ control points, and ith $N_{i,p}$ are the pth degree B-Spline non-rational basis functions given by the Cox-de Boor recurrence relation

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \le u < u_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.2a}$$

and

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u), \tag{2.2b}$$

defined over the non-periodic knot vector U for degree p,

$$U = \left\{ \underbrace{a,\ldots,a}_{p+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{b,\ldots,b}_{p+1} \right\} \quad \text{subject to } x_i \le x_{i+1} \tag{2.3}$$

There are some rules that govern how NURBS curves are defined. The degree p of a NURBS has to be a positive whole number and cannot be less than one (i.e., lines and polylines). The number of control points $n+1$ must always be greater than the degree. For example, a degree one NURBS curve (line) must have at least two control points, and a cubic NURBS curve must have at least 4 control points. The highest index m of $m+1$knots is associated with the degree and the number of control points in a NURBS curve by $m = n + p + 1$. For example, the cubic NURBS curve in Figures 2.3, 2.4 and 2.5 all have 13 knots in their knot vectors; $m+1 = (8+3+1)+1 = 13$ (note that this is the number of the knots, not the values).

2.2.1.1 Knot Vectors

There are more rules that have to be addressed in regard to knot vectors. The knot vector as stated in Equation 2.3 is subjected to $x_i \le x_{i+1}$, meaning it must be monotonically increasing, with successive values in the knot vector that must be arranged in ascending order. For example, the knot vector that is used for the curve in Figure 2.3

is $U = \{0,0,0,0,1/6\,,2/6\,,3/6\,,4/6\,,5/6\,,1,1,1,1\}$. Any different arrangement of these values is not acceptable. Typically, knot vectors are normalized as the one above. But they can also be chosen as an array of integers ranging from zero to some maximum number. The magnitude of the values of the knot vector has no effect on the NURBS curve. It is the relative ratios of the differences of the values to each other that affect the curve, and thus a knot vector like $U = \{10,10,10,10,20,30,40,50,60,70,70,70,70\}$ has the same effect as the normalized one.

Knot vectors can also be classified as *clamped* or *unclamped*. A clamped knot vector is the one that has first and last values of the vector repeated $p+1$ times. For example, $U = \{0,0,0,0,1/6\,,2/6\,,3/6\,,4/6\,,5/6\,,1,1,1,1\}$ used for the cubic NURBS curves in Figure 2.3 and 2.4 is clamped. The multiplicities at the beginning and end of the knot vector force the NURBS curve to pass through the first and last control points. Unclamped knot vectors are simply the ones that do not satisfy the multiplicities at the beginning and at the end $p+1$ times.

Knot vectors can also be divided into *uniform* and *non-uniform*, that is, within each group of the clamped and unclamped knot vectors discussed earlier. A uniform knot vector has equally spaced values, excluding the multiplicities at the beginning and at the end. The knot vector $U = \{0,0,0,0,1/6\,,2/6\,,3/6\,,4/6\,,5/6\,,1,1,1,1\}$ is a uniform knot vector. A non-uniform knot vector, on the other hand, is the one that has values that are not equally spaced. The knot vector that is used to create the kink in Figure 2.5, $U = \{0,0,0,0,1/6\,,2/6\,,2/6\,,2/6\,,5/6\,,1,1,1,1\}$, is an example of non-uniform knot vector since the value 2/6 is repeated 3 times. Another example is
$U = \{0,0,0,0,1/20\,,3/20\,,7/20\,,17/20\,,19/20\,,1,1,1,1\}$ used in Figure 2.7.

The differences in spacing between the values of the knot vector must be significant; otherwise, a non-uniform knot vector will behave much like a uniform one. Multiplicity of knots tends to have more drastic effects on the curve, such as those illustrated in Figure 2.5.

2.2.1.2 Basis Functions

The basis (or blending) functions is a set of linearly independent functions, that can represent every function in a given function space as long as the functions are in linear combination. The letter "B" in NURBS refers to "basis". If we let

$$R_{i,p}(u) = \frac{N_{i,p}(u)w_i}{\sum_{j=0}^{n} N_{j,p}(u)w_j} \quad \text{for} \quad a \le u \le b\,, \tag{2.4}$$

then we can write Equation 2.1 in the form

$$C(u) = \sum_{i=0}^{n} R_{i,p}(u)\,\mathbf{P}_i \tag{2.5}$$

The { $R_{i,p}(u)$ } are the rational basis functions. All the desirable geometric characteristics of NURBS curves and surfaces are drawn from properties (non-negativities, partition of unity, local support, etc.) that Equation 2.4 provides. The basis functions are defined over a knot vector and hence the shapes of the basis functions are determined entirely by the relative spacing between the knots in the knot vector. Figures 2.8-2.10 show the basis functions used for the three different curves in Figure 2.5.

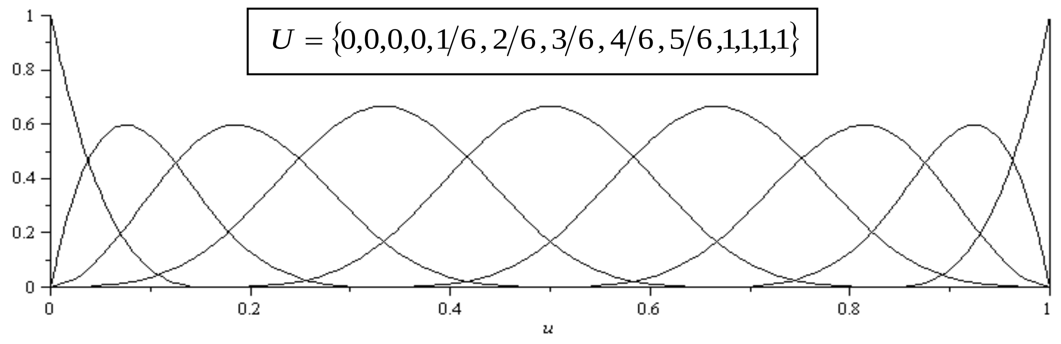


Figure 2.8. Third-degree basis functions, uniform knots.

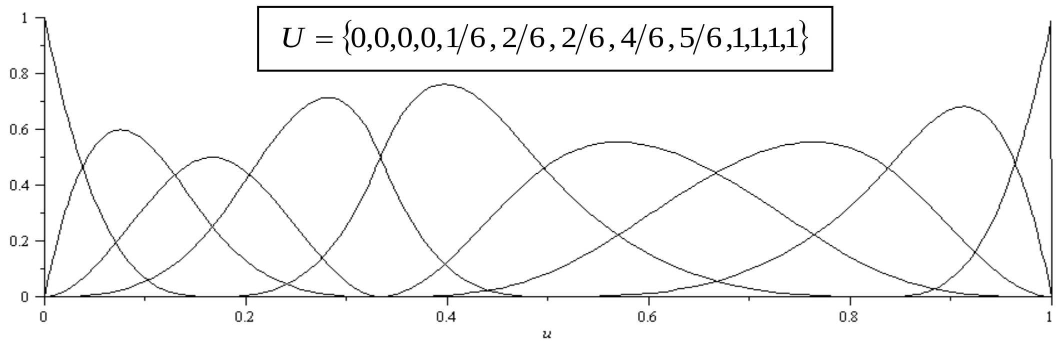


Figure 2.9. Third-degree basis functions, multiple internal knots

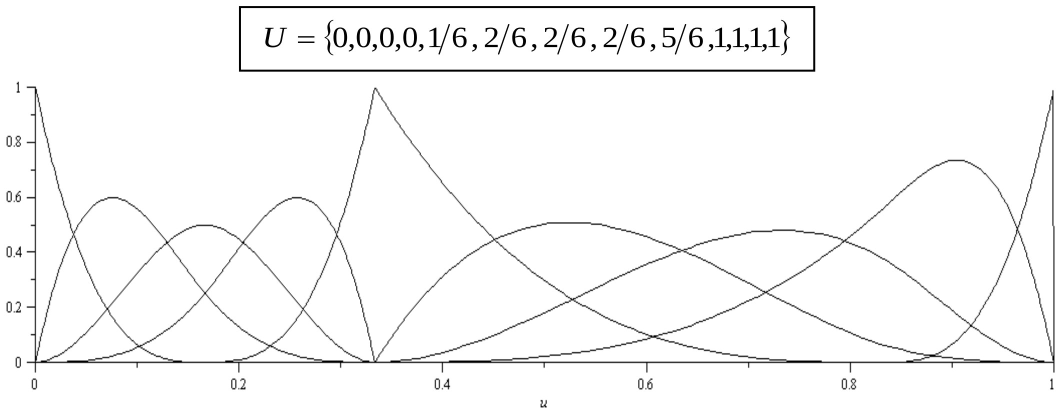


Figure 2.10: Third-degree basis functions, full multiplicity in internal knots

A B-spline blending function has compact support. The support of this function depends on the knot sequence and always covers an interval containing several knots, containing *p+2* knots if the curve is degree *p*.

It is sometimes useful to represent a *p*th degree NURBS curve in homogeneous form. This can be accomplished easily by using the following equation:

$$C^w(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i^w \tag{2.6}$$

where $P_i^w = (w_i x_i, w_i y_i, w_i z_i, w_i)$ is the weighted control points.

2.2.2 NURBS Surface

NURBS surfaces mathematically work like their curve counterpart, except they have two parametric directions (u, v) instead of one (u) as in Figure 2.11.

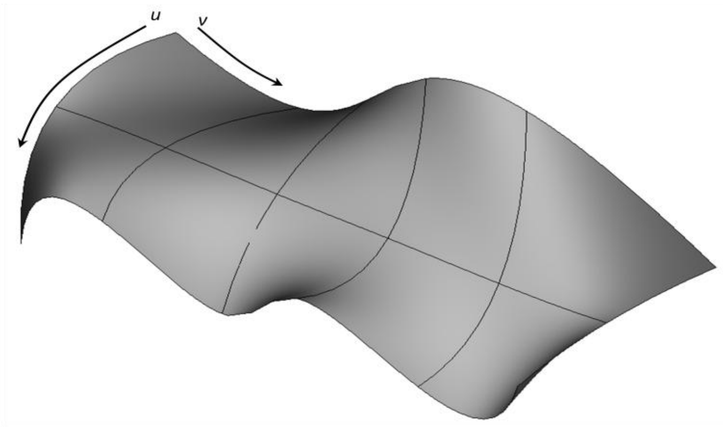


Figure 2.11. NURBS surface

A NURBS surface of degree p in the u direction and degree q in the v direction is defined by

$$S(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) w_{i,j} \boldsymbol{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) w_{i,j}} \quad 0 \le u, v \le 1, \tag{2.7}$$

where $\boldsymbol{P}_{i,j}$ are the control points that form a bidirectional control net, $w_{i,j}$ are the weights associated with each of the control points, and $N_{i,p}(u), N_{j,q}(v)$ are the non-rational B-Spline basis functions (Equations 2.2a and 2.2b) defined on the non-periodic knot vectors U and V for degrees p and q that are denoted by

$$U = \left\{ \underbrace{0,...,0}_{p+1}, u_{p+1},..., u_{r-p-1}, \underbrace{1,...,1}_{p+1} \right\} \quad u_i \le u_{i+1}, \tag{2.8a}$$

$$V = \left\{ \underbrace{0,...,0}_{q+1}, v_{q+1},..., u_{s-q-1}, \underbrace{1,...,1}_{q+1} \right\} \quad v_i \le v_{i+1}, \tag{2.8b}$$

with the highest indexes of knots (r and s) hold the relationships $r = n + p + 1$ for the U knot vector, and $s = m + q + 1$ for the V knot vector.

All the rules that are stated for the NURBS curve apply to a surface's two directions. Different surface directions can have different degrees. For example, the *U*-direction can have degree 2 while the *V*-direction can have degree 4. As with curves, we can introduce the piecewise rational basis function for the surface as follows:

$$R_{i,j}(u,v) = \frac{N_{i,p}(u) N_{j,q}(v) w_{i,j}}{\sum_{k=0}^{n} \sum_{l=0}^{m} N_{k,p}(u) N_{l,q}(v) w_{k,l}} \quad 0 \le u, v \le 1, \tag{2.9}$$

so that Equation 2.7 can be written as

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} R_{i,j}(u,v) \boldsymbol{P}_{i,j} \tag{2.10}$$

A NURBS surface can also be efficiently represented in homogeneous coordinates as in Equation (2.11)

$$S^{w}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) \boldsymbol{P}_{i,j}^{w} \tag{2.11}$$

where $\boldsymbol{P}_{i,j}^{w} = \left(w_{i,j} x_{i,j}, w_{i,j} y_{i,j}, w_{i,j} z_{i,j}, w_{i,j} \right).$

2.2.3 NURBS with Higher Dimensions

NURBS can also be used for higher dimensional representations. However, their visualization is more difficult and precarious [85]. For each additional dimension of the NURBS, for example a trivariate, an additional summation of the Cox-de Boor recursion formulas is performed. Equation 2.12 defines the trivariate (i.e., volume) representation in a homogeneous form

$$V(u,v,t) = \sum_{i=0}^{n} \sum_{j=0}^{m} \sum_{k=0}^{l} N_{i,p}(u) N_{j,q}(v) N_{k,d}(t) \boldsymbol{P}_{i,j,k}^{w} \tag{2.12}$$

Comprehensive information and explanations about the intricacies of NURBS and the various types of curves and surfaces can be found in literature [5,86,87].

2.3 Modeling Techniques

Creating complex 2D or 3D models using specialized software such as AutoCAD or SolidWorks usually starts by constructing basic geometric entities such as circles, lines, or curved surfaces in order that an idea can be realized from a designer's mind. These models can be represented and constructed in a number of ways.

A model can be constructed using polygonal modeling techniques where vertices are connected by line segments to form what is called a polygonal mesh or net. Polygonal models are flexible and can be rendered quickly. The main disadvantage is that polygons are planar; hence, curves and curved surfaces used to structure models cannot be accurately represented. They can only be approximated using many polygons as illustrated in Figure 2.12. Another problem is that polygon objects are more or less fixed. In other words, the shape of the object cannot be changed easily.

Another modeling technique that is used is to combine basic geometric primitives, like cones, cubes, and cylinders, to construct more complex and larger models. Typically, geometry can be joined through Boolean operations such as union, subtraction, and intersection. Like polygonal techniques, using primitives are quick and precise but cannot accurately represent complex curved surfaces.

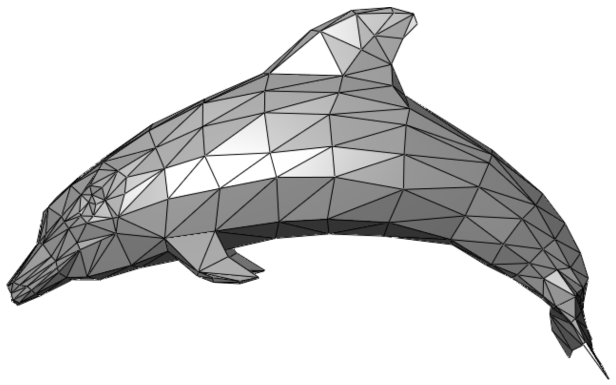


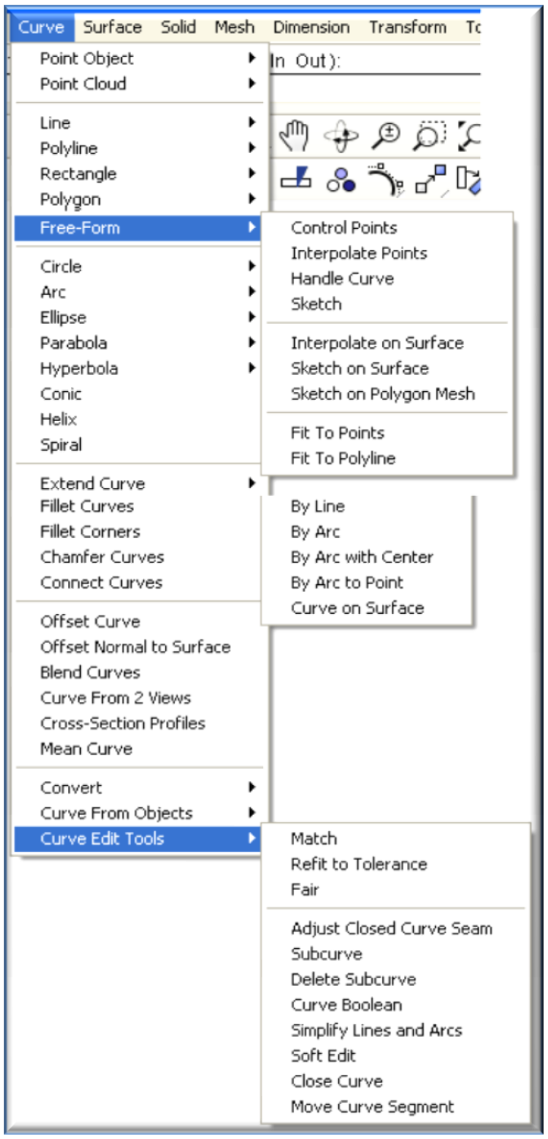
Figure 2.12. Polygonal (triangles) mesh of a dolphin

The subdivision of surfaces modeling technique is also used to create complex models. It combines some of the features of polygonal and NURBS modeling techniques in terms of flexibility and ease of use.

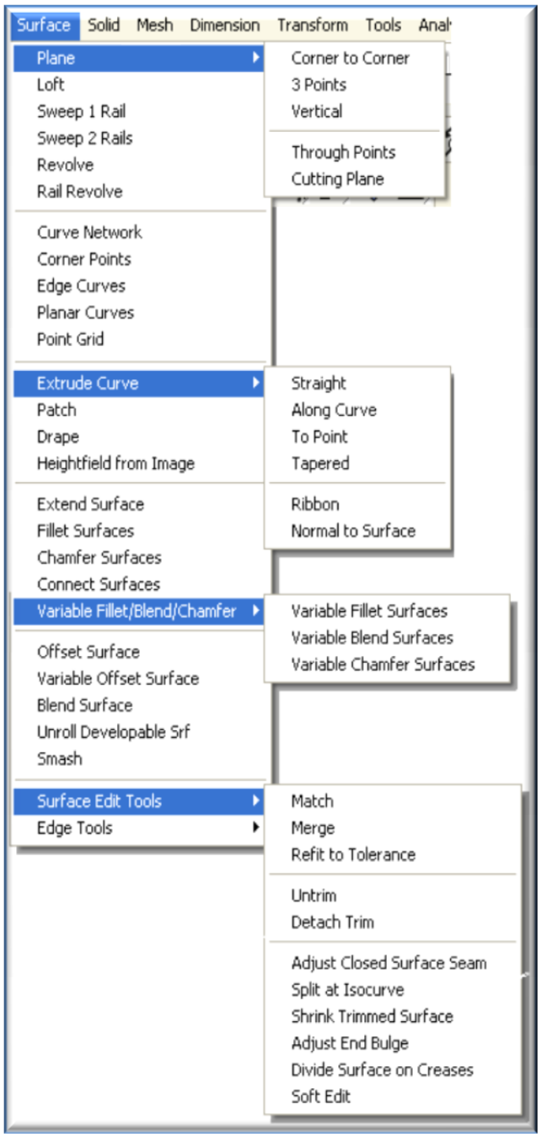
NURBS modeling techniques, which is the focus of the reminder of this section, is similar to polygonal mesh except that individual polygons are truly curved surfaces, not an approximation. Because of their accuracy and flexibility, NURBS models can be used in several applications such as illustration, animation, and manufacturing.

2.3.1 NURBS Modeling Techniques

NURBS modeling allows for truly smooth surfaces, not approximations using small flat surfaces. This makes them suitable for various modeling needs, including organic modeling. A complex NURBS model usually consists of many curves and surfaces joined together to form a complete model. NURBS modelers, such as Maya and Rhino, which use NURBS as the modeling kernel, usually provide designers with sets of modeling techniques. Such systems allow users to carry out the modeling process without mathematical experience although the the underlying implementation could include complex mathematical functions. Figure 2.13 (a) and (b) shows a portion of the wide varieties of curves and surfaces generations' techniques used in Rhino 4.0. However, whatever software a designer uses, there are a set of underlying basic NURBS modeling techniques that are used in creating NURBS curves and surfaces.



(a)



(b)

Figure 2.13. Rhino's curves and surfaces modeling techniques

The first set of techniques used in NURBS modeling is to use fundamental geometric tools such as reparametrization, degree elevation and reduction, knots insertion, deletion, and refinement. These fundamentals can be used for both NURBS curves and surfaces to enable extra modeling techniques and simplify the modeling process of complex models. Commercial CAD modelers usually allow the user to edit such parameters. However, these techniques are mostly used by the systems internally. For example, joining two NURBS curves with different degrees to form a single NURBS curve, or creating a surface by lofting a set of curves with different knot vectors and degrees might require degree elevation and knots insertion. Changing the direction of a curve or surface without changing the geometry can easily be achieved by reparametrization of the curve or surfaces. Figure 2.14 shows the process of skinning (lofting) three curves with different degrees and directions.

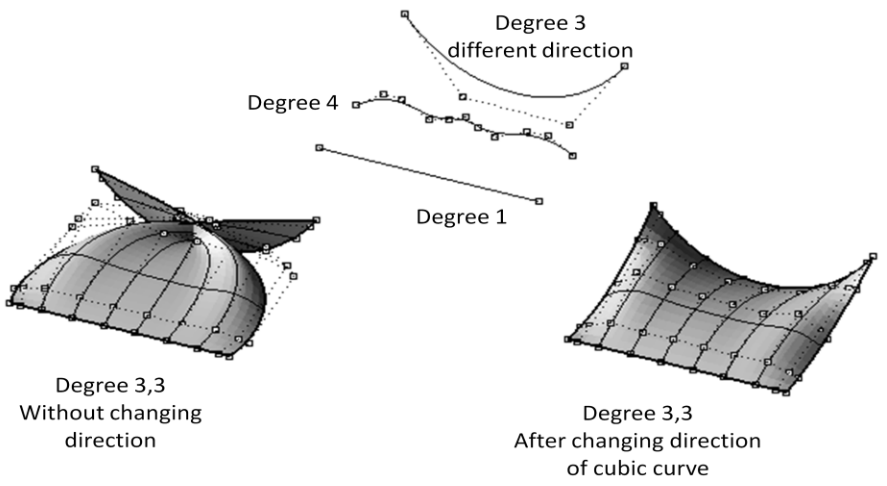


Figure 2.14. Degree elevation and reduction, knot insertion and removal, and reparametrization needed internally to perform surface lofting

The other set of NURBS modeling techniques involves the generation of NURBS curves and surfaces. Curves and surfaces can be created from a set of control points supplied to the system or by interpolating (or approximating with some tolerance) a set of data points. However, these are not the most convenient ways to model many NURBS entities, particularly surfaces. For example, an iso-parametric curve can be extracted from a surface; circles can be constructed from a center and a radius; and the special-case NURBS surfaces such as tours, spheres, or cylinders, illustrated in Figure 2.15, can be constructed from radiuses and some more parameters other than the control points.

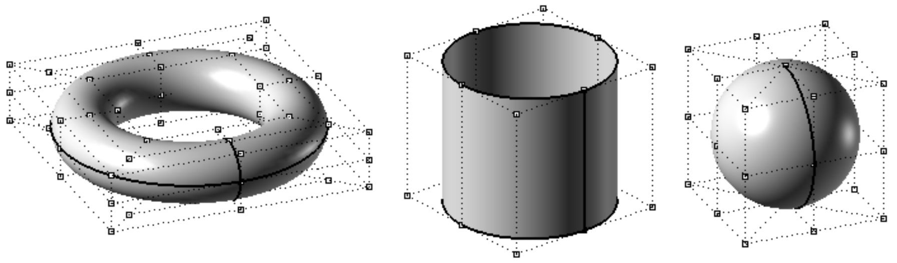


Figure 2.15. Special-case NURBS surfaces

NURBS surfaces are usually defined by a curve or set of curves. Beside the skinned surface and special-case surfaces illustrated above, there are many NURBS modeling techniques used to create wide varieties of surfaces: extrusion, full revolution and Gordon surfaces. Extrusion or swept surfaces are constructed by given a spine (sometimes called trajectory) and a cross section NURBS curves, as in Figure 2.16. Even though the basic idea is the same, extruding a curve to generate a surface can be done in many ways, such as to a point or by a straight line. See the Extrude option in Figure 2.13(b).

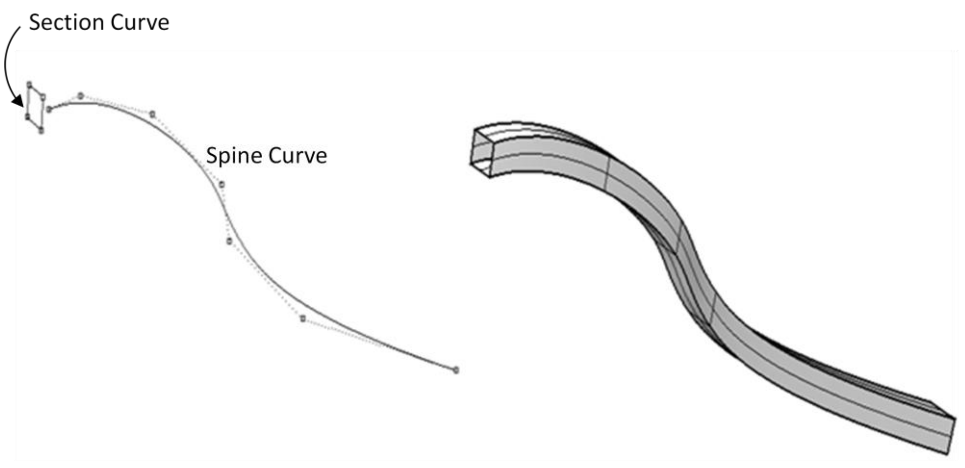


Figure 2.16. Extrusion (swept) NURBS surface

Surfaces of revolution are constructed by given a circle (full revolution) or circular arc (degree of revolution) and a cross section NURBS curve, Figure 2.17.

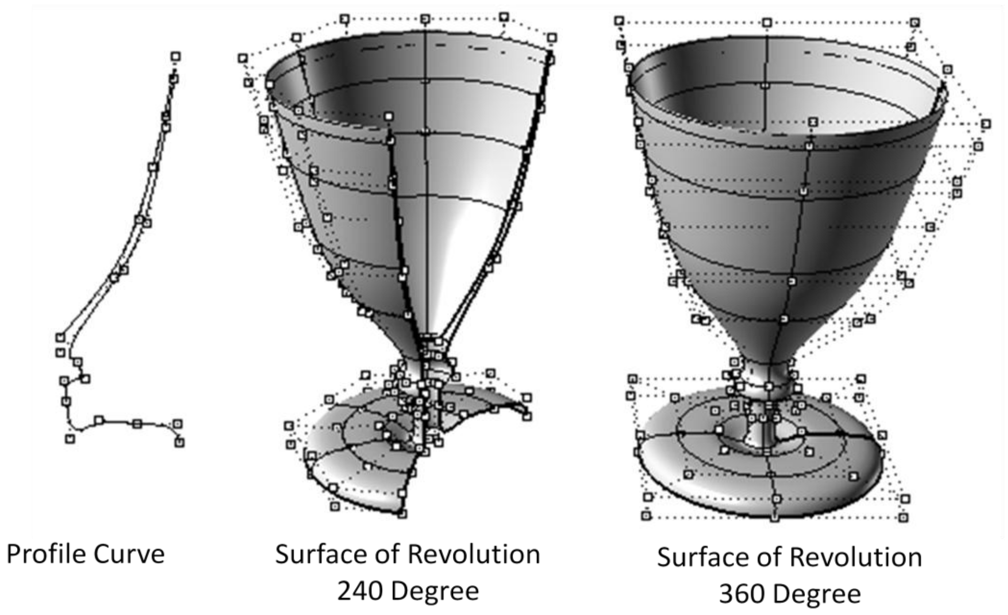

Profile Curve
Surface of Revolution 240 Degree
Surface of Revolution 360 Degree

Figure 2.17. Surface of revolution

Gordon surfaces are created by interpolating two bidirectional sets of curves network, as in Figure 2.18.

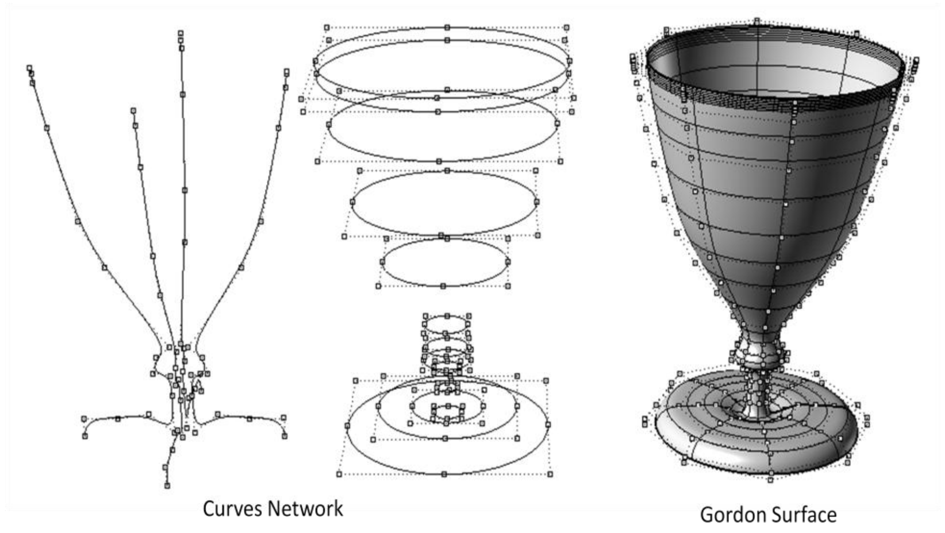


Figure 2.18. Gordon surface, interpolating of bidirectional curve network

Other types and techniques include but are not limited to swung surface, ruled surface, and Coons patches. Swung surface is a generalization of surface of revolution. Ruled surfaces are constructed from two rail curves, and Coons patches are bicubic blended surfaces constructed from 4 border curves [5].

One can notice the simplicity of constructing the class in Figure 2.16 by designing one curve compared to the curves network used to construct the similar class in Figure 2.17. The above NURBS modeling techniques are some of the basic ones used. There is a wide range of modeling techniques in NURBS allowing for design with simplicities and modeling with more possibilities.

2.4 NURBS Model Representation

Once created, every NURBS surface is exactly the same as every other NURBS surface; it is defined by control net, degrees, and knot vectors. The inconsistencies are within different CAD systems. For example, one system, CATIA, generates the cylindrical surface as two open patches, while the other, Rhino, produces a closed surface with a seam. See Figure 2.19.

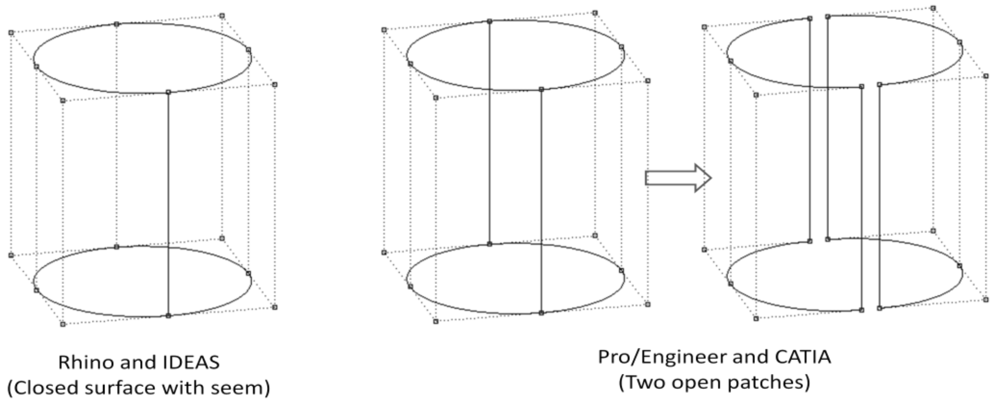


Figure 2.19. Different representations of a cylinder

The inconsistency is going to result in difficulties when converting one format into another. This is because open and closed surfaces are handled differently at the software level. For example, intersection curves crossing the seam must be split, topology differences may result in geometry problems in addition to confusing the systems with the number of vertices, edges, and faces that are also represented differently in different CAD systems.

Another example is the circle that is represented as a degree two, degree four, or degree five rational-curve in different systems. See Figure 2.20. Degree elevation, e.g., going from degree two to five, can be made precisely. However, there are two disadvantages.

One, the process only changes the form of the curve. It keeps the geometry as well as the parametrization intact. A NURBS curve is a parametric entity whose geometry is closely tight to its parametrization. A poorly parametrized NURBS curve can cause discontinuities in subsequent surface constructions and operations, such as when used to generate ruled and lofted surfaces or when used for offsetting based on point sampling. After degree elevation, the curve has the structure of degree five; however, its behavior is degree two only.

Second, a circle is an important entity in modeling and many systems prefer to represent it as a degree five NURBS curve to avoid multiple knots and to have adequate continuity for subsequent operations. The conversion process from degree two to five introduces knots with multiplicity five and discontinuities in the homogeneous space, which contradicts the purpose of having it in the first place. The process can also increase the data drastically when a set of curves are merged together for subsequent (surface) constructions.

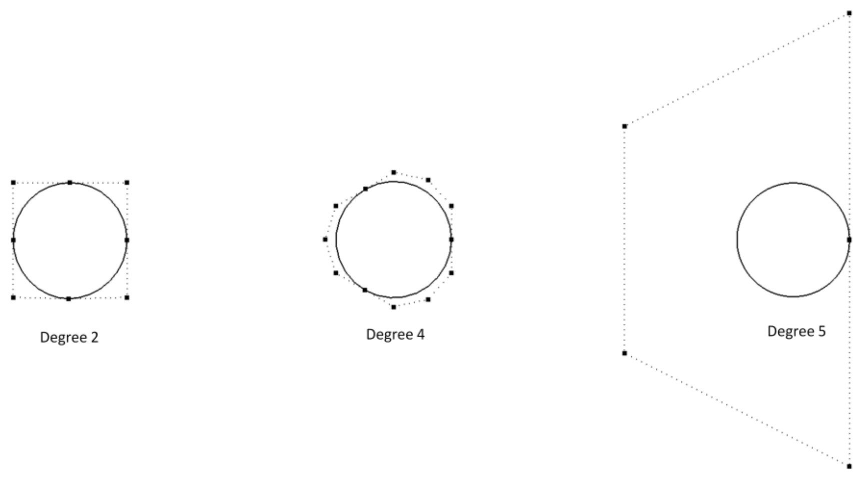


Figure 2.20. Different circle representations in NURBS

Degree reduction, for example, going from degree five to two, is an imprecise process, and the original precise circle representation is now inaccurate. The converted degree five circle, which is four times continuously differentiable, becomes either a non-rational approximation or a discontinuous curve in homogeneous space after degree reduction.

NURBS models are usually constructed from a set of NURBS curves and surfaces. However, the representation of the same model could vary based on the systems that are used during the initial and intermediate construction of the model.

2.5 NURBS Data Export to Other CAD Systems and Storage

Every CAD system has its own methods of describing geometry, resulting in a large variety of data formats. See Table 2.1. CAD vendors secure and protect their data formats since sharing their proprietary data formats and the database structures would put them at a less competitive advantage. Moreover, new CAD systems are constantly emerging with new data formats, and some old vendors go out of business. Supply chain companies that use CAD systems are faced with the task of continually exporting data formats from one system to other CAD systems in order to complete a complicated model that could be the product of tens of contributors.

When exchanging data that represents NURBS curves and surfaces to and from systems that use NURBS as their modeling kernel, three possible scenarios can occur. A NURBS entity can be translated precisely both mathematically and geometrically with only floating point round off errors. It can also be geometrically precise, but the

Table 2.1. A selection of CAD file formats

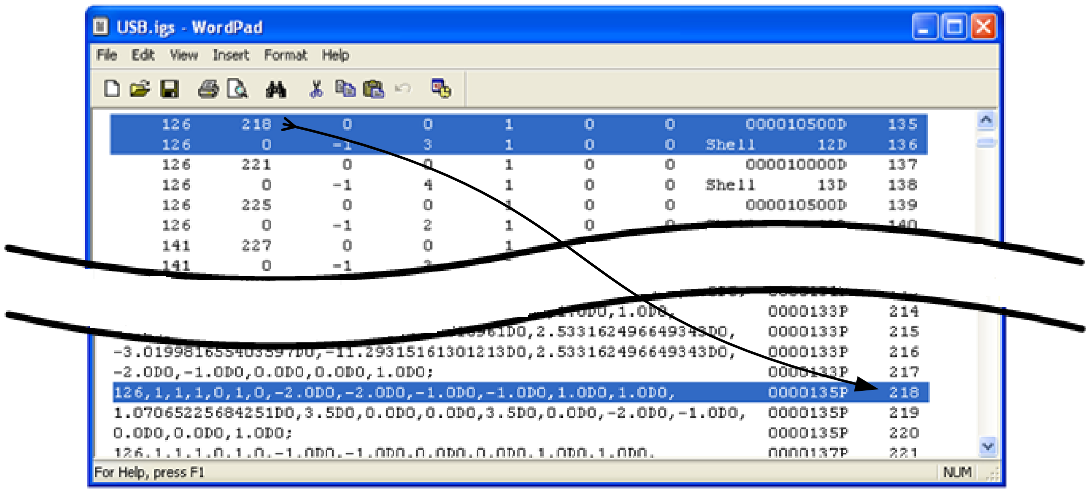
ART	ArtCAM
ASC	BRL-CAD
ASM	Solidedge assembly
CCM	CopyCAD model
CAD	CADst
CAT	CATIA
DWG	AutoCAD
DGN	MicroStation design file
DGK	Delcam geometry
FM	FeatureCAM part file
GRB	T-FLEX CAD file
ICD	IronCAD 2D file
IGES	IGES file
ISFF	Integraph format for MicroStation
PRT	Unigraphics,Pro/ENGINEER
SKP	SketchUp model
RWS	Rhino work session
SLDPRT	SolidWorks part model
STEP	STEP file

parametrization is changed: for example, if a circle is parameterized by trigonometric functions in one system while parameterized based on chord length in another system. Notice that surfaces that use circles in their constructions can be different geometrically. Finally, the NURBS entity can be translated approximately with some specified tolerance: for example, if there is a degree restriction in one system and the other system allows for higher degrees. Converting a higher degree NURBS entity to a lower degree can only be achieved by approximation.

Exchanging different data formats is a risky process since it is afflicted by many issues, including information loss or misinterpretation and change of data accuracies, as illustrated in the motivation section of Chapter 1. However, it is the only option used in the industry so far, and most often it is achieved through the use of standard formats, i.e., IGES and STEP.

2.5.1 NURBS within Standards

An IGES file must structurally be composed of five sections: *S*tart for sender comments, *G*lobal section for general file characteristics, *D*irectory entry for entity index and common attributes, *P*arameter data for entity data, and *T*erminate section to indicate the end of the file. Each section consists of records that are 80 ASCII characters long with characters in column 73 (S, G, D, P, or T) to indicate what section this record belongs to. The start section is set up by the person creating the IGES file to contain useful information to the receiver of the file. The global section contains parameters needed for the file translation. The directory section is generated by the IGES pre-processor and contains entry for each IGES entity in the file. IGES entities are labeled with numbers to indicate their type. For example, the repeated 126 shown in Figure 2.21 is for NURBS curve entity. The second column value indicates the line number in the next section (Parameter Data) where specific parameters such as knot vectors and control points are associated with each IGES entity.



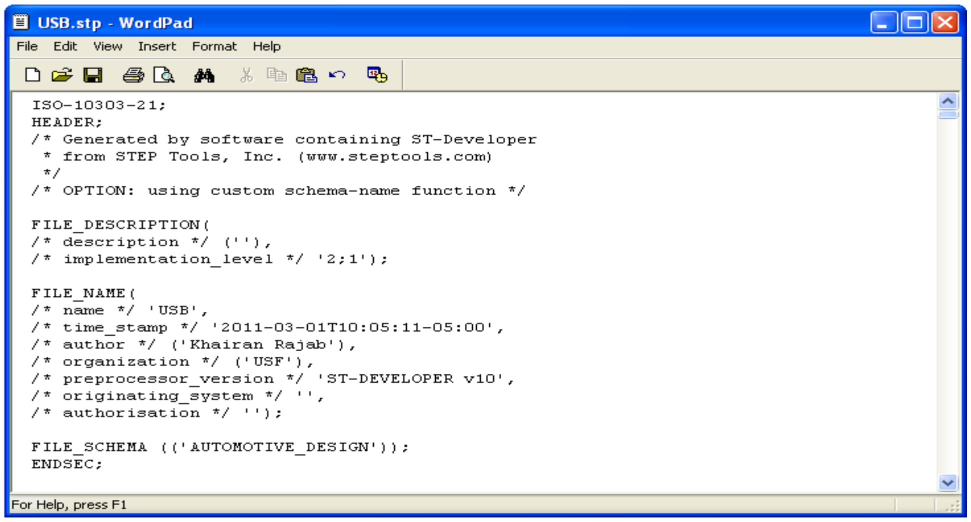
```
     126     218       0       0       1       0       0        000010500D    135
     126       0      -1       3       1       0       0 Shell        12D    136
     126     221       0       0       1       0       0        000010000D    137
     126       0      -1       4       1       0       0 Shell        13D    138
     126     225       0       0       1       0       0        000010500D    139
     126       0      -1       2       1       0                              140
     141     227       0       0       1
     141       0      -1

                                        .0D0,1.0D0,                0000133P    214
                                   961D0,2.53316249664934 3D0,     0000133P    215
-3.01998165540355 97D0,-11.293151613012 13D0,2.533162496649343D0,  0000133P    216
-2.0D0,-1.0D0,0.0D0,0.0D0,1.0D0;                                   0000133P    217
126,1,1,1,0,1,0,-2.0D0,-2.0D0,-1.0D0,-1.0D0,1.0D0,1.0D0,           0000135P    218
1.0706522568425 1D0,3.5D0,0.0D0,0.0D0,3.5D0,0.0D0,-2.0D0,-1.0D0,   0000135P    219
0.0D0,0.0D0,1.0D0;                                                 0000135P    220
126,1,1,1,0,1,0,-1.0D0,-1.0D0,0.0D0,0.0D0,1.0D0,1.0D0,             0000137P    221
```

Figure 2.21. Snapshot of IGES file

For storage and representation of a piece of geometry, the amount of data required is much smaller than the amount of information required by many other representations, including the common faceted approximations. IGES requires degrees and number of control points, Euclidean control points, knot vector, parameter values, and other useful information such as tagging a curve or a surface as a special type. For example, line, conic arc, and circular arc used to define a NURBS curve; and plane, torus, surface of revolution, and cylinder are used to define special surfaces.

STEP files can be implemented by ASCII structure defined in STEP-File (ISO 10303-21), by Extensible Markup Language (XML) format defined in STEP-XML (ISO 10303-28), or through an abstract Application Programming Interface (API) that is defined in Standard Data Access Interface or simply SDAI (ISO 10303-22). STEP-File implementation is the most widely used due to its ASCII nature. See Figure 2.22. Almost every translator will have the option to import or export to STEP-File, which ends with file extensions ".stp" or ".step".



```
ISO-10303-21;
HEADER;
/* Generated by software containing ST-Developer
 * from STEP Tools, Inc. (www.steptools.com)
 */
/* OPTION: using custom schema-name function */

FILE_DESCRIPTION(
/* description */ (''),
/* implementation_level */ '2;1');

FILE_NAME(
/* name */ 'USB',
/* time_stamp */ '2011-03-01T10:05:11-05:00',
/* author */ ('Khairan Rajab'),
/* organization */ ('USF'),
/* preprocessor_version */ 'ST-DEVELOPER v10',
/* originating_system */ '',
/* authorisation */ '');

FILE_SCHEMA (('AUTOMOTIVE_DESIGN'));
ENDSEC;
```

Figure 2.22. STEP file (USB in Figure 1.1)

NURBS curves and surfaces are defined in Part 42, which focuses on basic geometry and topology of a product model. The same data and information that are used in IGES to define NURBS curves and surfaces are also used with STEP definition of NURBS, with little extra information such as a flag for self-intersecting.

Chapter 3 --- Knowledge Guided Design Intent Systems

Knowledge guided design intent systems are, in general, much more complex than CAD systems that primarily focus on the design and manufacture capabilities. Such systems go several steps beyond the basic CAD systems by not only providing the basic capabilities of traditional CAD system but also by dealing with knowledge and design intent capturing, archival, management, and retrieval without hindering the designing processes. The added features make such systems fairly complex. However, they can enable benefits and capabilities ranging from object classification to knowledge mining and design replay that are discussed in Chapter 4.

To be able to add these extra features to CAD systems, a set of questions about knowledge and design intent has to be addressed. Section 3.1 deals with fundamental questions such as the meaning of knowledge in the general sense and within the context of NURBS environment. It also discusses knowledge acquisitions and representations in a CAD setting. Section 3.2 narrows the general discussion of design intent from the Prior Work section in Chapter 1 to structure a well-defined design intent within NURBS. It also discusses the mechanisms by which we can support the presence of design intent in NURBS. The last section of this chapter, section 3.3, discusses the type and amount of information that needs to be retained in such systems. Investigating the above questions should set the stage for Chapter 4, in which the broad definition of the knowledge guided design intent systems can be focused to form a knowledge guided NURBS system.

3.1 What is Knowledge?

Data, information and knowledge are often considered synonyms of one another, and the distinction between them is certainly fuzzy [88]. However, researchers differentiate and describe the relationship between them. Miller et al. [89] detail how data can be structured or unstructured and can represent a measure such as a quantity. For example, a string of numbers on a piece of paper (e.g., 8139740000) is considered data since it represents raw and non-interpreted facts. Information, on the other hand, has been described as the "describing a fact" where the fact is an occurrence of a measure or inference of some quantity or quality [90,91]. Knowing that these numbers represent a phone number in the US is an example of information. The more information that is inferred, derived, or deducted from the telephone number is what constitutes knowledge: for example, knowing that a number with a particular area code was issued in the state of Florida or using a number to get an address.

Within the NURBS-based modeling environment, most current CAD systems produce raw numerical data at the end of a designing task, that is, control points, degree, and knots of a curve or a surface. A knowledge guided design intent system goes beyond the basic CAD systems by gathering and managing information such as the type of curve that was created: for example, a circle with a given radius and center, how it was created, function used in creation, alternative algorithms, relationships (if any) with other NURBS entities, and much more other information that is part of the designing process. Such information is usually not accounted for in traditional CAD systems and is largely lost either at the low level of object design or during model transfer, causing topological inaccuracies and threatening robust computations down-streams. Using the gathered data and the important information about NURBS entities to get more information—i.e. knowledge—is what makes the system a knowledge guided NURBS system. In the designing process, the gathered data and information will be stored in a design base and a knowledge base. The knowledge base and design base can enable knowledge mining to learn about the design, document the design, aid design replay to reconstruct offending parts of the design. Figure 3.1 shows how data, information and knowledge can be integrated into knowledge guided NURBS-based system.

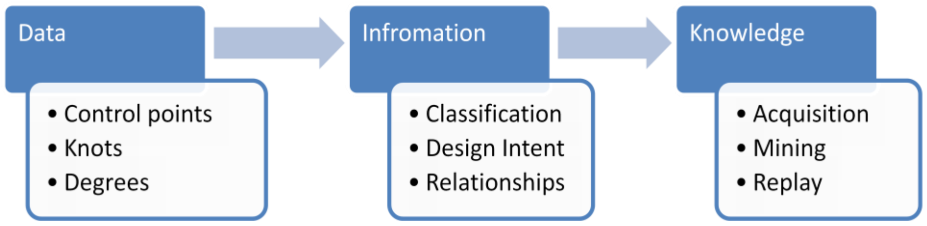


Figure 3.1. Data, information, knowledge within NURBS-based systems

From the literature definitions of data, information and knowledge and from Figure 3.1, one can conclude: knowledge is built or constructed on top of both data and information that are acquired at earlier stages or during the designing process of a model; the presence of information implies the abilities to access data; and the presence of knowledge implies the ability to access both data and information.

3.1.1 Knowledge Acquisition and Representation

The process of acquiring knowledge can be achieved in three main ways: user-intervention, automatic, or a hybrid approach. The designing process is highly creative. If knowledge and information is to be gathered with great user intervention, the designer might get reluctant to record the information needed to preserve the designing intent. Moreover, the knowledge base might have gaps that will render it useless if the knowledge acquisition is left to the designer. Thus, in a CAD environment, knowledge acquisition should be fully automated with flexibility to allow for knowledge management such as modifications, deletions, retrieval, and maintenance to the acquired knowledge.

Acquired knowledge can be represented based on the purpose and intended use of it. Literature, however, usually classifies it under three broad classes of representations: formal, informal, and semi-formal.

Lee [91] describes the formal representation as a full written documentation. He states that it is overly costly and can limit the creative thought and development. Hicks et al. [90] sub-classify the formal representation into three categories: textual, pictorial, and verbal. They state that textual representation of knowledge is usually structured numeric, alphabetic, or symbolic, or a combination of these. Pictorial representation is any visual image, such as three-dimensional engineering drawing. Verbal representation is the documented descriptions and explanations of design intent and decisions of the design team.

Informal representation is in the form of memos, emails, meetings, videotapes, etc. This type of representation can be difficult to combine and present into single coherent form [92]. The level of details of the documentation varies based on a designer's judgment.

Semi-formal representation is a compromise between the formal and the informal approaches. It provides some computation power but is still understandable by the human providing the information.

3.2 What is Design Intent?

Design intent has various meanings in different applications and fields. The Prior Works section in Chapter 1 gives a detailed discussion on design intent in the general sense and from the perspective of many fields. The section states that the definition formulated by Iyer and Mills survey [12] on design intent is adequately comprehensive. Repeated here for convenience: they state that "design intent is application, domain and context dependent knowledge that describes design space, represents design alternatives and processes history, justifies design solutions and decisions and determines the characteristics of features and entities and the relationships among them."

According to general connotations of the term, this definition is difficult to quantify and structure. Fortunately, in a NURBS-based modeling environment, it is fairly clear what design intent is: all relevant design information that can be structured to support design replay. See Figure 3.2. In other words, it is all the information we need to enable the regeneration of a CAD model or design on many dissimilar CAD systems as if it was created on each one them. To this end, the capabilities outlined in Figure 3.2 and presented in the subsections that follow are supported.

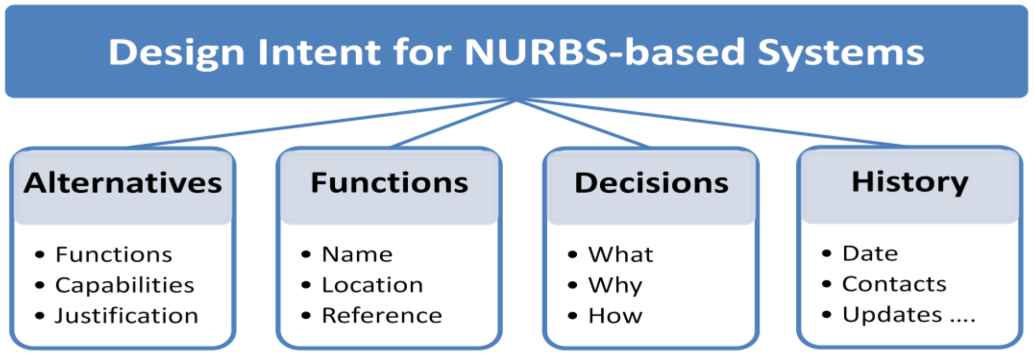


Figure 3.2. Design intent in NURBS modeling

3.2.1 Design Alternatives

Modeling systems usually provide various alternatives to perform the same design tasks. If the design is not up to standard in the receiving system, it is most useful to know what kind of modeling tool was used in the first place and alternatives in order to make the proper decision. At a minimum, knowledge guided design intent system will support the following:

- Name of the function(s) that produced the current design.
- Alternative capabilities and their short descriptions.
- Justification as to why the current function was chosen.

3.2.2 Design Functions

All design tasks in a NURBS system are performed by some function. In order to support the replay of the designer's intent (design session), functional information becomes a must. The following functional information will be provided:

- Name of the function(s) that produced the final result, e.g. `KGN_surruled.c`.

- Location of the function, e.g. `c:\KGnurbs\src`.
- Version and the release date, e.g. `V2.1_2011.`
- Reference to documentation (on-line or offline), e.g. `c:\KGnurbs\doc\surruled.htm`.

3.2.3 Design Decisions

Designers tend to make mistake during the designing process and then repeat the same mistake at later stage of the designing process. They also undo and redo each other's work because there is no information on what led to one's decision and what the relevant circumstances were. For example, one designer might create a kink on a surface because it is stylish and was part of the original concept of the design while another designer might remove it because of the machining process. The second decision should be recorded to avoid redoing the same mistake. To assist in proper reconstruction of the design and to avoid mistakes caused by repetition, knowledge guided design intent system will provide at least the following:

- Recording of the decisions that resulted in the current design, i.e., *what*.
- The reasons and justification that led to decisions that were made, i.e., *why*.
- Deliberations that lead to the decisions, i.e., *how*.

3.2.4 Design History

Designers need history for many reasons, most importantly for learning from prior designs and avoiding repetitive mistakes and works on what had been proven to be infeasible. Design history can be detailed; however, the amount of information should be under control and the design history should be kept short. The following capabilities are the minimal information:

- The date of the initial design's creation.
- The name and contact details of all people involved in the process.
- All major updates, error fixes, and other maintenance information.
- Reference to any history file, if any. The history file will contain a complete profile of the design, incorporating all of the above: alternatives, decisions, functions, maintenance and a time stamp.

3.3 What Information Needs to be Retained?

One of the major challenges in knowledge guided design intent systems is what and how much information needs to be generated and retained. Too little information may not allow the system to replay the design process. On the other hand, too much information may devastate the system and create problems with storing, usage, and maintaining all the relevant as well as irrelevant knowledge.

One of the main objectives of knowledge guided design intent systems is to preserve the designer intent when migrating models from one system to another dissimilar system. All information and data that are part of design intent as described above must be retained—i.e., design alternatives, functions, decisions, and history.

Other objectives such as reliability and robustness are also supported by retaining necessary information. For example, finding the distance between a set of points and a curve or surface can be made robust if we know the object type and discontinuities (if any) and some other information that might help in choosing the right algorithms and computations. Knowing the type of curve is a line can aid in robustness by projecting all the points without any Newton-type iterations is an example of such a case. Anomalies such as cusps and boundary points can be accounted for if we know where at the curve or surface they exist.

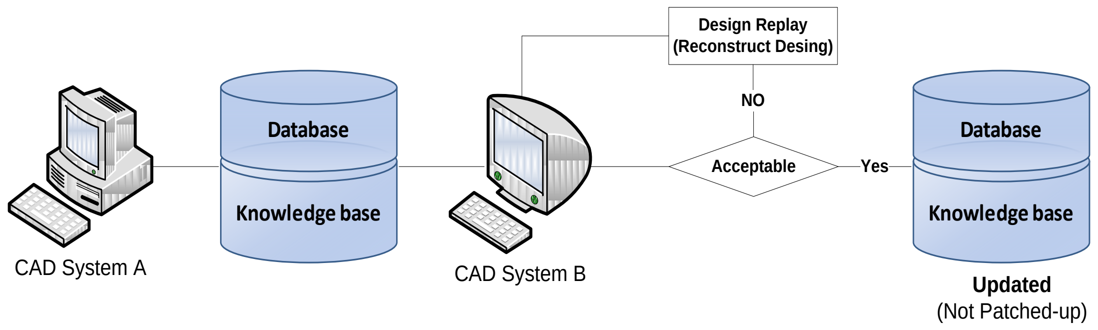


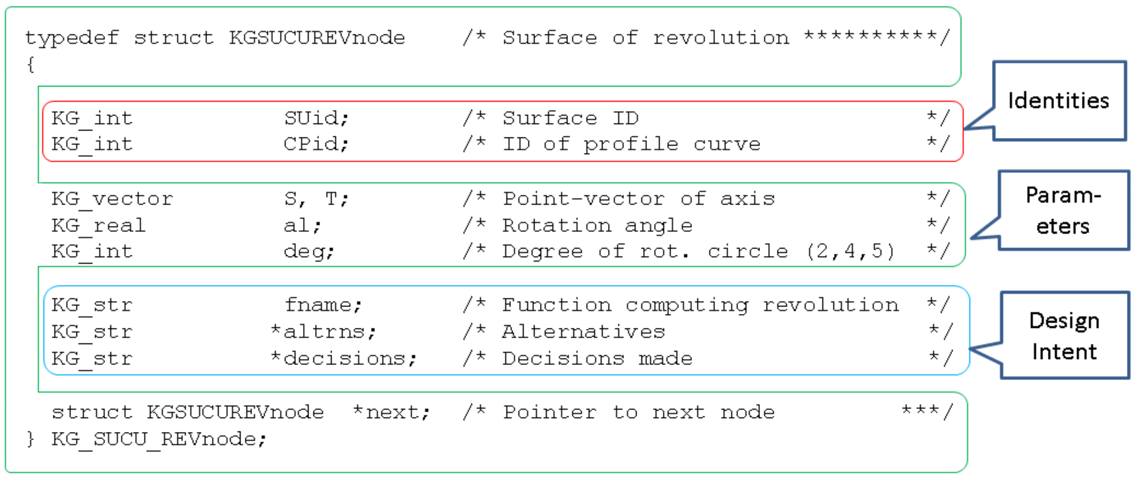
Figure 3.3. Design-and-edit in knowledge guided system

Current CAD systems do not preserve the design intent and work in a design-and-repair behavior, where the final model might be repaired and patched up manually or using specialized software as described in the motivation section in Chapter 1. Knowledge guided design intent systems work on a *design-and-edit* approach. If the migrated design is not acceptable at the other end, the system will reconstruct the whole design based on local requirements through a *design replay manager*. See Figure 3.3. In order to accomplish this goal, all necessary information that enables design replay (design regenerate in particular) must be gathered and stored in a design base and knowledge base.

An example is a surface of revolution constructed form a profile curve. If the surface is not acceptable in the receiving system, the design scenario must be repeated.

For simplicity at this point, we focus our concern on the information needed for reconstructing the design, not accounting for objectives such as the information needed to aid the robustness and reliability. A design replay process has to be provided with the following:

- *Identities* of the participating objects. These can be attained by proper *naming* and *identification*.
- *Design intent* will provide the receiving system with what *functions* were used to design the entities, what *alternatives* were considered, and what *decisions* were made.
- *Relationship maps* are complicated data structure. They are made of nodes that hold objects *references*, *relationships*, and their *parameters*. For example, when the surface of revolution is computed, the surface and the profile curve enter into a *Surface-Curve-Revolution* relationship and all the parameters needed to recreate the design are saved.



```
typedef struct KGSUCUREVnode     /* Surface of revolution **********/
{
  KG_int            SUid;        /* Surface ID                      */
  KG_int            CPid;        /* ID of profile curve             */
  KG_vector         S, T;        /* Point-vector of axis            */
  KG_real           al;          /* Rotation angle                  */
  KG_int            deg;         /* Degree of rot. circle (2,4,5)   */
  KG_str            fname;       /* Function computing revolution   */
  KG_str           *altrns;      /* Alternatives                    */
  KG_str           *decisions;   /* Decisions made                  */
  struct KGSUCUREVnode  *next;   /* Pointer to next node         ***/
} KG_SUCU_REVnode;
```

Figure 3.4. Surface-curve information for design replay of surface of revolution

Figure 3.4 shows the C code used in the system for a *surface-curve node* that hold information needed to reconstruct a surface of revolution. Recall that having information implies the presence of data, and hence the control points, knot vectors, and degree are all available to be accessed and retrieved. The design intent and as the information used for the relationship map are implemented in different parts of the system based on where suitable and hence only parts of them are encoded in the nodes.

Using this fundamental data and information, the designer at the receiving system should repeat the designing process with ease using a user friendly interface. The system can determine what objects are involved in the design and find their IDs and references. The system should retrieve design alternatives, functions, and decisions to perform the redesign so as not to repeat previous mistakes. Finally, the system must access the modeling kernel, pass all required parameters, and reconstruct the surface based on the receiving system specifications—e.g., tolerance.

The next chapter, Chapter 4, provides a complete definition of all knowledge and information needed to achieve all the objectives of the knowledge guided design intent system. It details information needed for knowledge acquisition, knowledge mining, as well as detailed information on design replay.

Chapter 4 --- Knowledge Guided NURBS

Previous chapters set the stage for what follows by discussing knowledge guided design intent systems and the motivations to embrace them. This chapter discusses the system in more detail. The global architecture of a knowledge guided design intent system within NURBS environment is shown in Figure 4.1

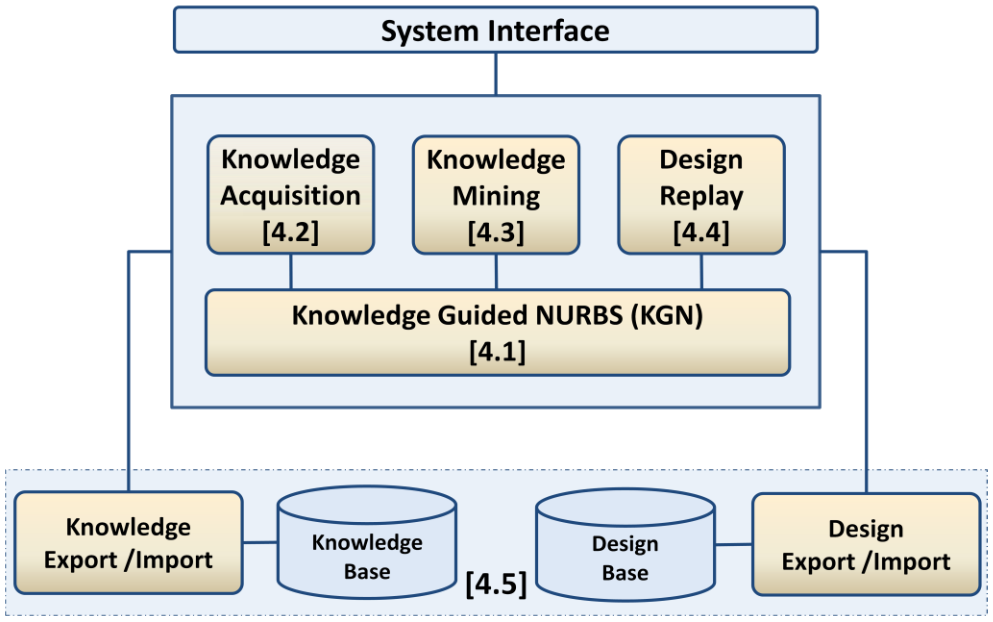


Figure 4.1. Knowledge guided NURBS system global architecture. Chapter sections shown in brackets

The system is built on top of a knowledge guided NURBS kernel. Access to modeling functions, memory stacks, and knowledge stacks are enabled via access links (routines). Knowledge acquisition manager, knowledge mining manager, design replay manager, and export/import managers are the other main components of the system. As with any CAD modeling system, the users interface with the system's functionalities via a user-friendly interface.

Section 4.1 defines knowledge guided NURBS (KGN) kernel as the heart of the whole system. Section 4.2 describes the ways that KGN acquires knowledge through the knowledge acquisition manager. The knowledge mining manager is discussed in section 4.3 as a mean of browsing the knowledge and design bases for many reasons, such as assisting with design replay as well as learning about the generated models. Design replay and reconstruction capability of KGN models is the subject of section 4.4. Finally, sections 4.5 details exported/imported managers and how design and knowledge bases are built within KGN environment.

4.1 Knowledge Guided NURBS

The core of the system is a KGN kernel. Like any other NURBS-based kernel, it is required to provide the modeling capabilities and routines that enable designers to realize ideas and concepts into a geometric model. However, it surpasses the traditional kernels by incorporating a wealth of information to individual entities as well as to the whole model. KGN is capable of processing enough information and knowledge to support our three main objectives: seamless model migration, robustness, and reliability. The precise definition of KGN is a compromise between data storage and design and manufacturing requirements. Figure 4.2 gives the major components of KGN kernel along with the corresponding chapter's section that covers each of them. Except for design intent, which has been discussed in depth previously in Section 3.2, this section covers identification, classification, definition, representation, geometry, and relationship maps. These constituents are the keys to achieving our objectives, and they are encoded at the lowest level of the designing process making it part of the kernel.

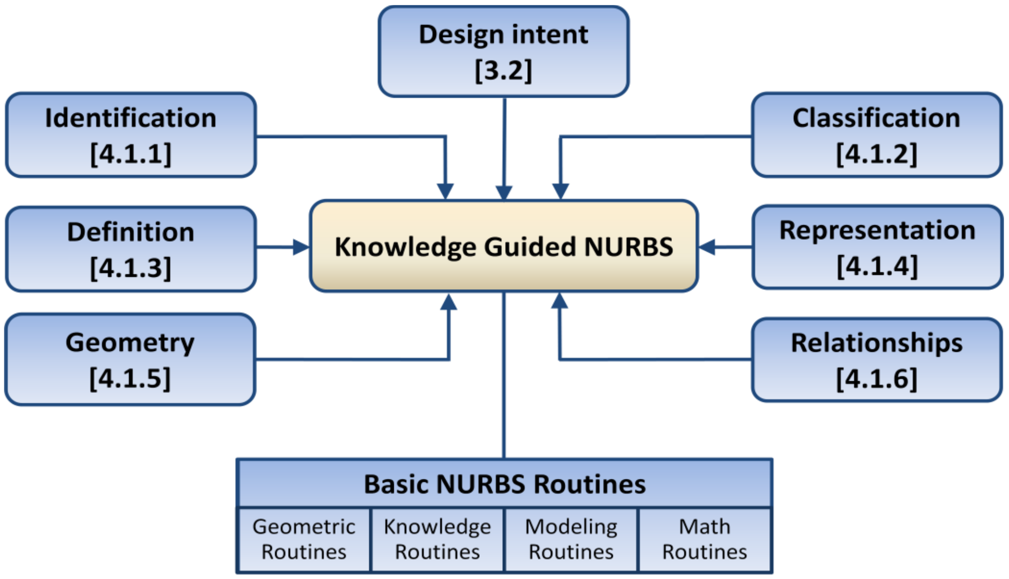


Figure 4.2. Knowledge guided NURBS constituents. Chapter sections shown in brackets

4.1.1 Identification

Literature reveals that the naming of entities is causing many problems when migrating CAD models between the inherently incompatible CAx systems. The papers [93-98] proposed a set of approaches that can be classified into two main approaches: (1) naming based on topological ID’s such as faces, edges, and vertices and (2) naming based on geometric coordinates of the entities. Both approaches have weaknesses, such as ambiguity and naming collisions. Some commercial CAD systems—e.g., SolidWorks—

minimize the problem by naming entities based on type and the 3D coordinates of the referenced entities. Proper naming and identification is important for consistency within the KGN system and many issues have to be considered.

Like database tables that have primary keys, entities in KGN must have unique IDs for proper database building and management. However, for a proper persistent naming, a proper migration, and an error-free knowledge bases rebuilding, a simple indexing mechanism that merely gives entities a simple unique number is not sufficient. Simple indexing, for practical purposes, can preserve the integrity of the design base or the database within one system. But CAD models are always going from one system to another, raising problems such as naming collisions and ambiguities. Names of objects in KGN have to be meaningful and should contain enough information to eliminate issues associated with persistent naming in CAx. The first issue that KGN naming should address is to distinguish its entities names from the many NURBS kernels that are in the market today with inconsistent naming. KGN uses a prefix `(KGN)` before every entity goes to the data and knowledge bases in order to avoid clashes with other CAD systems or kernels.

The second is that the same kernel or CAx system gets updated with new versions over the course of the years, hence there is no version control. KGN incorporates the version number into the name—e.g., (V2.1)—to gain a better version controlled naming. This will assist with upward and downward versions' compatibilities of the naming. It will also assist with design replay, particularly when to redesign an entity if a newer version or update (ideally) fixes coding bugs that were encountered in older versions of the system, particularly with the designing functions. The year of the release—e.g., 2011—is also added to the name to recognize the name and versions years later if the system ever gets to be a legacy.

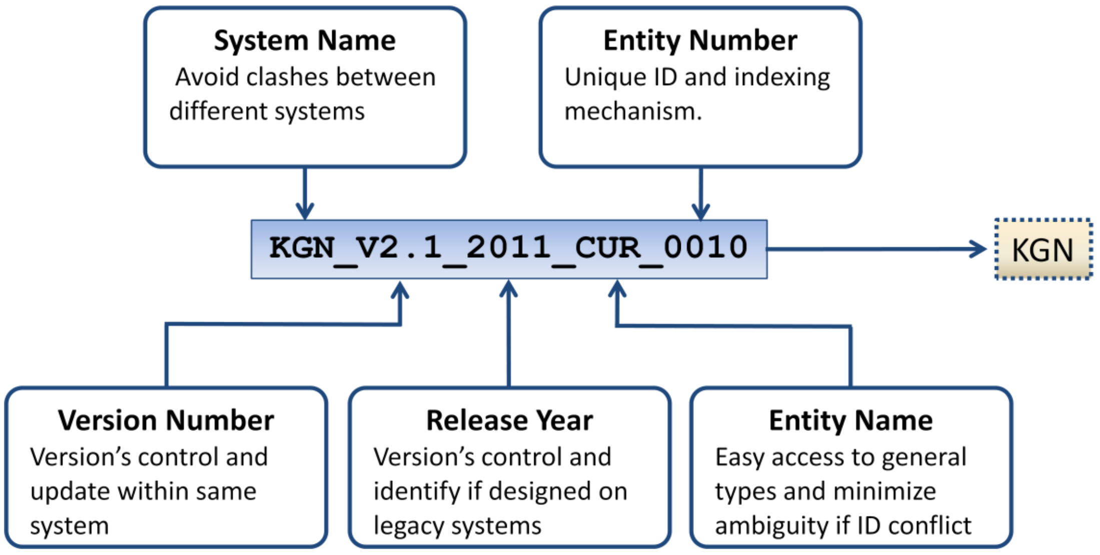

System Name
Avoid clashes between different systems
Entity Number
Unique ID and indexing mechanism.
KGN_V2.1_2011_CUR_0010
KGN
Version Number
Version's control and update within same system
Release Year
Version's control and identify if designed on legacy systems
Entity Name
Easy access to general types and minimize ambiguity if ID conflict

Figure 4.3. Unique part ID in KGN: avoid associated naming problems

Another identifier that KGN adds to its entities' names is the object general type—e.g., curve, surface, and volume. For implementation and consistency purposes, the first three letters of each type is proven to be good choice. For example, a curve object name will be tagged with `(CUR)`, a surface `(SUR)`, and volume `(VOL)`. The last part of the name is the index or number ID—e.g., `(0001)`. It is the only part that must be unique for every name. The system must not allow any duplicate names. If, for any reason, there is an ambiguity or a collision between two or more names, the other parts of the name are used for validation. An example of a curve object full name used in KGN is shown in Figure 4.3 with an explanation of each part.

4.1.2 Classification

Classification of NURBS objects follows the *what-how-why* paradigm that corresponds to *type-origin-destination*. See Figure 4.4.

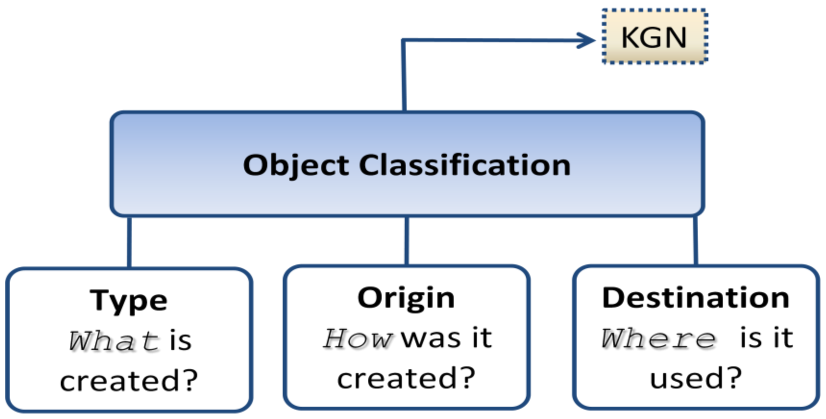


Figure 4.4. Classification in KGN: what-how-why paradigm

4.1.2.1 Type

This represents the exact type of the object, or *what* exactly it is in terms of NURBS. The type is not the general type used in identification mentioned before. It is more specific. For example, a NURBS curve can be: line, circle, ellipse, hyperbola, biarc, conic section, etc. NURBS surfaces can be special cases, such as a cone or cylinder, torus, sphere, etc. Other advanced constructed NURBS surfaces types can be a lofted surface, ruled surface, surface of revolution, Gordon, etc. KGN tagged every object created with the proper type. Although KGN should provide all possible types, an *unknown* type case is added to the list of available types to cover any type that is not accounted for during the implementation of the kernel.

An immediate benefit of classifying NURBS objects based on types is that certain object types are handled by special algorithms. A significant number of operators, such as intersection routine or point distance calculators, have special codes for known data types. For example, there is a big difference between intersecting a line and a circle, and a line and a NURBS curve. Knowing that the NURBS is a circle would provide the system with enough information to produce a more accurate result than simply using general code such as general purpose intersection code for line/circle intersection. Many types of surfaces can be considered separately, such as ruled surfaces, lofted surfaces, and surfaces of revolution. Special case NURBS surfaces such as planes, spheres, cones, cylinders, and torii can also be handled separately by the system with special case routines if accuracy and reliability becomes an issue.

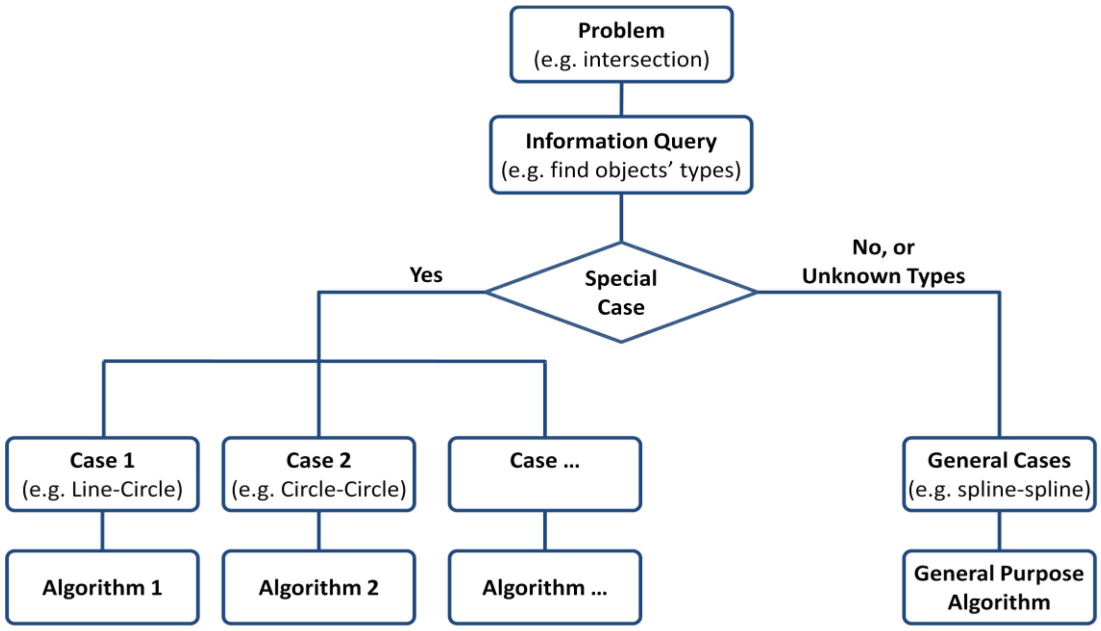


Figure 4.5. Using objects' type (information) to locate a suitable algorithm

It is clear that coding will increase in both size and complexity since the kernel has to consider calling different routines for different cases, as illustrated in Figure 4.5. However, this is a much more efficient and more reliable computing system than having to deal with the inaccurate results that could result in a wait time of days or even weeks to fix errors. The system does all this in the background within a fraction of a second.

4.1.2.2 Origin

This tells KGN where exactly the part comes from, or *how* it was created in terms of NURBS modeling tools. Every NURBS object originated somewhere in the system. The most basic origin is when the designer defines a NURBS entity by some components—e.g., creating a curve by plotting control points on the modeling canvas or creating a circle by its center and radius.

Other origins of NURBS entities involve other NURBS objects. For example, a set of points can originate from sampling a NURBS curve, or a curve can be created from approximating a set of points. Examples of possible origins used in KGN include intersection, sketching, fitting, offsetting, transformation, and general advance construction. To account for NURBS objects that are created on different systems or of unknown origins, KGN adds import and unknown origins to the list of the possible origins of an entity.

4.1.2.3 Destination

This is the application where the object will be used, or *why* and *what-for* it was created. Examples of destination include intersection, styling, and fitting. The destination of an object is more involved than the type and origin described above. This is because the system cannot determine the destination in advance—i.e., during the initial creation of an object. Another issue is that an object can have multiple destinations. For example, a curve can be used in the construction of a ruled surface and then used as a base curve for sampling.

A simple solution is to ask the designer to record the destination with a pop up window that has a selection list of possible destination. However, this choice is not practical and can be a bothersome to the designing and creativity process of the designer. The system deals with this issue by marking the destination as unknown, and then updates the destination once the system determines it.

KGN does not account for multiple destinations at this junction of information gathering since it can be easily inferred from different parts of the knowledge base and the relationship map. Relationship query, discussed in Section 4.2.2 as part of *knowledge mining*, can provide the designer and the system with all objects involved with a particular object, including destination object(s).

4.1.3 Definition

Control points, knots, and degrees are necessary to fully define NURBS curves, surfaces, and volumes; this includes special case NURBS such as lines, conics, planes, and quadrics. However, for the purpose of design replay, which is a major objective of the system, additional definition information has to be gathered and stored in the database. For example, if the NURBS entity at hand is a degree 2 circle that is generated in one system, and another system needs to regenerate (not convert but actually redesign) the circle as a degree 5 circle, the system needs the center point, radius, and normal vector to define the original circle. Having these defining components can assure that the receiving system produces another NURBS curve (circle) whose topology is identical to the original curve but with different characteristics—i.e., control points, knots, and degree that are suitable for this receiving system. Table 4.1 shows what KGN stores as defining components for different NURBS entities.

Table 4.1. KGN basic entities and definitions

Entity	**Defining Components**
Point	▪ Point coordinates
Line	▪ Start point ▪ End point
Conics (circle, parabola, ellipse, hyperbola)	▪ Center ▪ Radii ▪ Unit Axes
Plane	▪ Co-planar points, or ▪ Closed polygon ▪ Normal vector
Quadric (sphere, ellipsoid, cone, torus)	▪ Center ▪ Radii ▪ Unit axes
Curve, **Surface,** **Volume**	▪ Control points ▪ Knot vector(s) ▪ Degree(s)

Additional information such as color, material types, cost of processing, etc. can also be included.

4.1.4 Representation

A NURBS is a parametric entity whose parametric representation is not unique. How well it is represented parametrically has a profound effect on many subsequent operations, such as point sampling, numerical processes, and surface constructions. Every single construction is valid only with respect to a given parametrization and must be recorded in the knowledge base to ensure a reliable and robust system. Figure 4.6

shows the representation parts recorded in KGN to aid with a wide range of numerical processes.

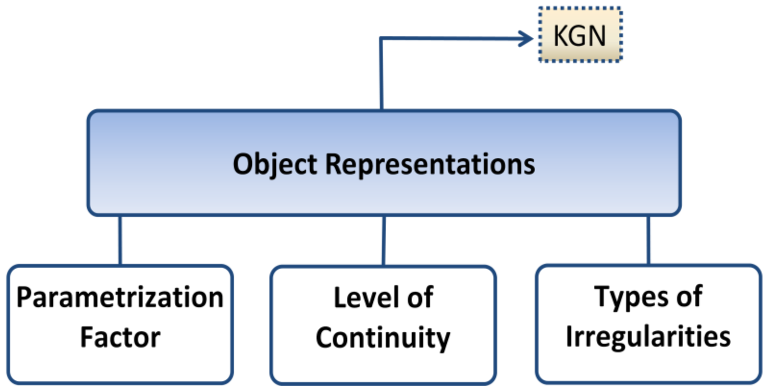


Figure 4.6. Representations in KGN, increase robustness and reliability

4.1.4.1 Parametrization Factor

Parameters in NURBS usually are classified broadly as either uniform or non-uniform. But a closer look can reveal that we could have degrees of non-uniformity—e.g., some parameters are almost uniform. The variations and types of parametrization have a profound effect on many important applications. For example, curve/surface decomposition and choosing the start value for point projection using Newton iterations [99]. In the case of decomposition, knowing the type of parameters becomes critical in finding the proper decomposition, that is, the iterative process can be anchored to not "run-off" if the parametrization is highly non-uniform. For Newton's method, it is important to start with good initial value; good start points are easy to guess if the curve/surface is parametrized uniformly.

Parametrization factor is a measure of how far a parametrization is from being uniform. Choosing a good method to measure the parametrization factor has been discussed in detail in [99]. Briefly, a parametric curve is the path traced out, say by a particle, while the parameter sweeps out the parametric domain. If the particle moves along the path of the curve/surface with constant speed, the parameters are spaced uniformly. On the other hand, if the speed of the particle changes, then the entity has a non-uniform parameters. Since speed is measured by the magnitude of the first derivatives, the parametrization factor (*pf*) can be measured as follows:

$$pf = \frac{D_{\max} - D_{\min}}{D_{\max}} \quad D_{\max}, D_{\min} \text{ are min-max derivatives}$$

The formula above is the standard measure of variation in terms of a range. If the curve is parametrized uniformly— $D_{\max} = D_{\min}$ and $pf = 0$ — there are no variations between the values. On the other hand, if the entity has a vanishing derivative (the particle comes to a stop), the parametrization factor is 1. Simply put, the smaller the factor, the less variation between the values and the more uniform the parametrization. Table 4.2 gives a good distribution of the sub-classification of parameters we have found to be practical based on testing numerous different cases of NURBS curves and surfaces.

Table 4.2. Parametrization factor

Pf	**Parametrization**
$0.0 \le pf \le 0.2$	(Nearly) uniform
$0.2 < pf \le 0.4$	Nearly non-uniform
$0.4 < pf \le 0.7$	Non-uniform
$0.7 < pf \le 1.0$	Highly non-uniform

The parametrization factor plays as an indicator when it is imperative to reparametrize entities before any computation begins, such as when it is highly non-uniform ($0.7 < pf \leq 1.0$) and may cause computation problems. Therefore, KGN computes and records the parametrization factor for every parametric entity created. Doing this, KGN can increase robustness and reliability of routines, where speed and guaranteed convergence is highly dependent on the type of parametrization, such as Newton's method.

4.1.4.2 Level of Continuity

NURBS curves and surfaces have continuity or smoothness associated with them internally, that is, between the curves' segments and the surfaces' patches, or at joint points and edges. There are two forms of continuities: geometric (i.e., physical) and parametric (i.e., mathematical). These two types are further subdivided into various levels of continuities—e.g., positional (G^0), tangential (G^1), curvature (G^2), or only an approximate geometric continuity. Generally, the increase of the order for both geometric and parametric indicates an increasing measure of smoothness and motion. Many numerical algorithms require first or second-degree parametric continuity, other processes are achieved by geometric continuity only, and yet some constructions can only guarantee continuity in an approximate manner. For example, filling an n-sided hole with a set of rectangular patches guarantees only approximate continuity, which then must be recorded in order to avoid failure in subsequent numerical processes. A continuity level greater than C^2 is unnecessary for most 3D computer modeling; however, KGN records continuities up to level five.

4.1.4.3 Irregularities

Significant work has been done on fairing and smoothing, that is, removing unwanted irregularities. The robustness of any CAD system can be improved greatly by appropriately dealing with irregularities that not only create unacceptable designs, but also may crash the system. Typical irregularities that hinder numerical code from working properly are cusps, poles, seams, zero curvatures, zero normals, large or negative weights, collapsing derivatives, and multiple knots. For example, a numerical process to find the distance between a point and a curve using Newton's method can fail to converge if the curve has a discontinuity, for example, a cusp or the closest point (projection) is at the point of discontinuity. The Newton method tends to jump over the discontinuity or hovers around it, entering into an infinite loop.

KGN records all forms of irregularities that are known to cause failures or may result in decreased system performance. The awareness of their presence in a NURBS entity can increase the system robustness and reliably through special code segments that account for these irregularities or initiate redesign or reparametrization processes to remove them.

4.1.5 Geometry

Curves and surfaces in NURBS models have a lot of geometry that can be tagged for downstream applications for robustness and reliability. For example, curves and surface curvature can play important roles in a robotics path, NC machining, and shape analysis. It is highly recommended to slow down the robot or the cutter of an NC machine in high curvature areas. In shape analysis applications, the curvature can play an important task in the process of sampling or discretization. High curvature areas must be sampled with more density, whereas low curvature segments can use fewer sampling points.

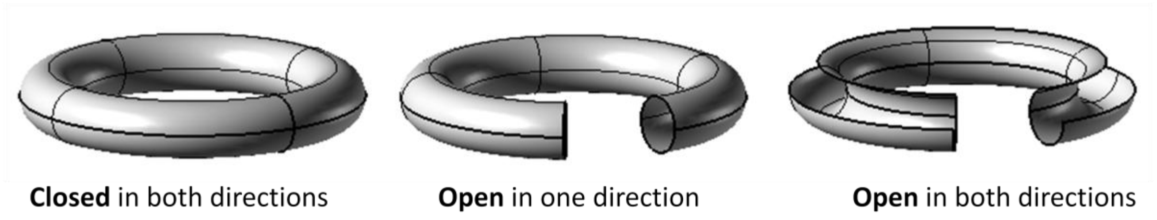

Closed in both directions
Open in one direction
Open in both directions

Figure 4.7. Geometry: example of open and closed NURBS surface (torus)

Important information that can be used by many processes and applications is knowing if the curve or surface is closed or open. A curve can be either closed or open while a surface can be open in both directions like a grid, closed in both like a torus, or open in one and closed in the other like a tube as shown in Figure 4.7. Table 4.3 list some of the important geometries that KGN record for NURBS entities.

Table 4.3. NURBS geometries in KGN

NURBS	Geometry Components
Curve	▪ Length ▪ Curvatures (min, max) ▪ Openness
Surface	▪ Area ▪ Perimeter ▪ Curvatures (min, max, mean) ▪ Openness (U and V-directions)
Volume	▪ Volume ▪ Perimeter ▪ Openness (U, V, and W- directions)

4.1.6 Relationship Maps

Organizing and managing a large amounts of related but disjointed information is a great challenge that KGN must address. Knowledge has to be structured in a way that permits easy access for knowledge mining and design replay as well as the ability to perform modifications that might cause full backward regression to the whole knowledge base. There are a number of ways to represent and structure knowledge in computer and information science. However, the most suitable form for NURBS-based modeling is a *relationship map*, which is a sophisticated data structure recording important relationships.

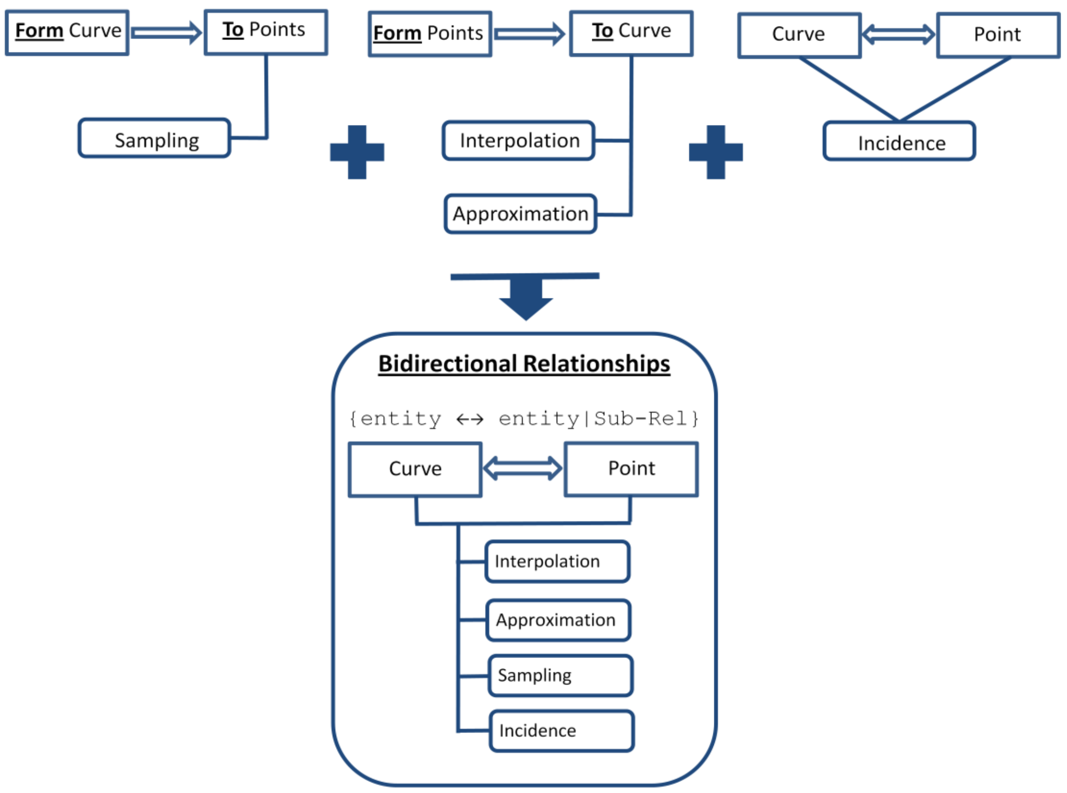


Figure 4.8. Bidirectional relationship map with sub-relationships

In order to create a structured knowledge map, it is necessary to identify specific pieces in the designing process and use them to imply the knowledge they are representing. During any construction operation in a NURBS system—e.g., point sampling or interpolation—entities enter into directional, and usually two-way, relationships. For example, the upper part of Figure 4.8 shows that when a NURBS curve is sampled, the points and the curve enter into a Curve → Points relationship, which may be called sampling.

On the other hand, when points are used to generate an interpolating curve, the points and the curve enter into a Curve ← Points relationship that may be called interpolation. The direction arrow represents the sequence of the operation: e.g., first we had points, then we generated the curve, and the name of the sub-relationship represents the operation used to obtain the design, e.g. interpolation.

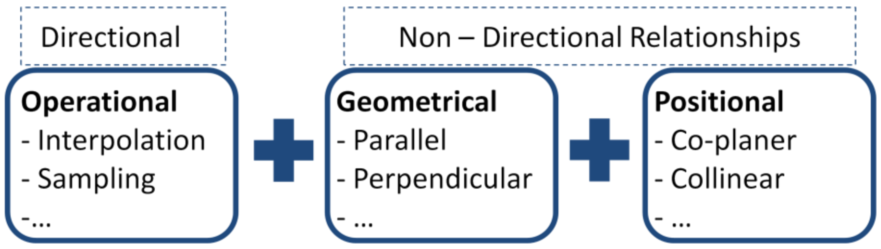
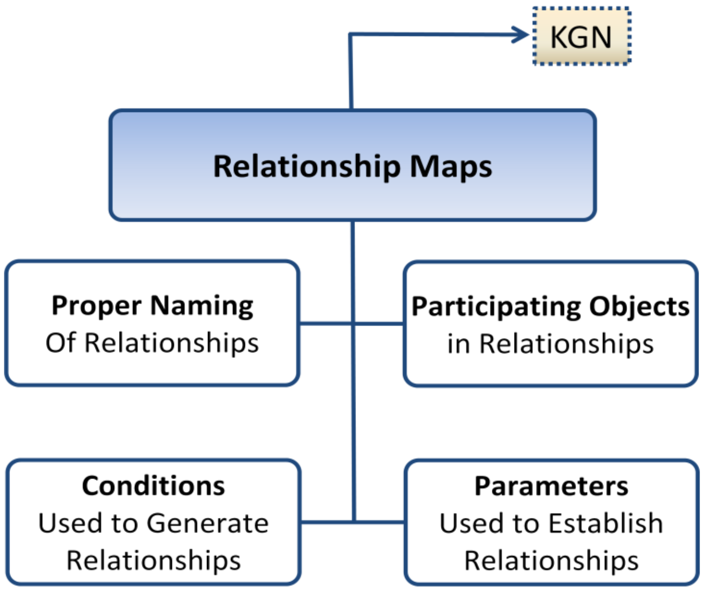


Figure 4.9. Types of sub-relationships in naming

The two-way directional relationship explained above is a good way to illustrate how we can establish relationships between different NURBS entities if the association is based on construction operations. However, other types of relationships can complicate the implementation, storage, and management of the data structure. For example, *geometrical* relationships, such as if two lines are parallel or perpendicular, have no sense of direction. *Positional* relationships, such as if points are co-planer or collinear, also lack the directional relationship. See Figure 4.9. To overcome these difficulties we use a *bidirectional* approach for the relationships as: `Rel = {entity ←→ entity | relationship: …}`, as illustrated in the lower part of Figure 4.8.

Proper relationship naming, participating objects, parameters, and conditions used to establish relationships, as outlined in Figure 4.10, are the most relevant pieces of information that will be needed to design a relationship map for KGN. These key pieces of the relationship maps are most crucial with design replay. They also play an important role in elevating the system robustness and reliabilities


KGN
Relationship Maps
Proper Naming Of Relationships
Participating Objects in Relationships
Conditions Used to Generate Relationships
Parameters Used to Establish Relationships
Figure 4.10. Factors to consider for relationship map implementation in KGN

4.1.6.1 Proper Naming

The naming of the relationship should reflect the types of the objects in relation as well as the type of the relationship and sub-relationship. If the relationship is a construction relationship, then the operation type and the direction of the relationship can be inferred from the relationship name. For example: `Rel = {curve ←→ points | sampling: …}` indicates that a curve was sampled and the result is a set of points. On the other hand, `Rel = {curve ←→ points | fitting: …}` indicates that a set of points was interpolated or approximated and the result is a curve.

4.1.6.2 Participating Objects

All objects that participate in the relationship should be recorded and should be accessible. For example, a fitting operation should have access to the generated curve and to the set of points that was used to generate the curve. Example: `Rel = {curve ←→ points | fitting: C, P;…}`. Figure 4.11 shows the general structure of the relationship map along with curve-point relationship. It also illustrates that the IDs can be used to access the objects from the design base to get all the needed information—e.g., information needed for design replay.

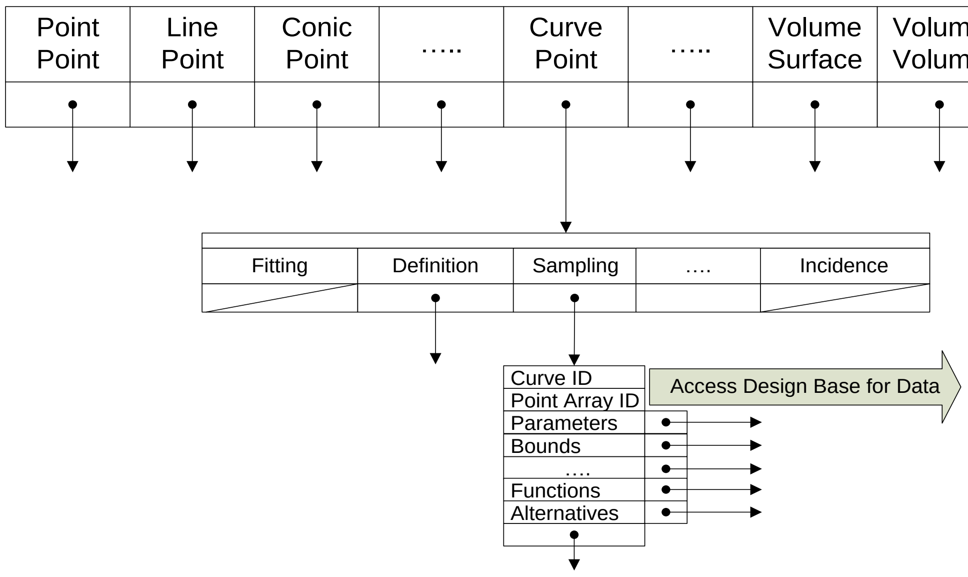


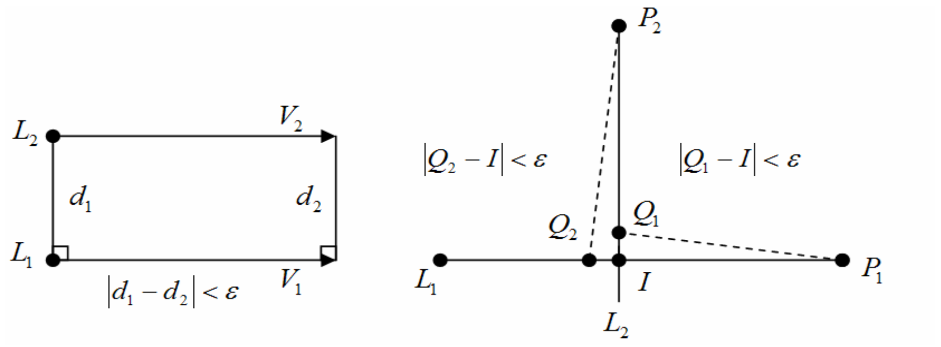
Figure 4.11. General structure of relationship map. Curve-point relationship

4.1.6.3 Parameters

Different CAD systems can use different sets of modeling parameters, which can cause problems when migrating a model from one system to another. To be able to perform design replay and model editing, it is imperative to record every one of them. An example: `Rel = {curve ←→ points | fitting: C, P; t0,…,tk; u0,…,um; deg; tol;…}`. That is, the system has access to the points and the curve (participating objects), the parameters, the knots, degree, and the tolerance used. For this specific example, if the tolerance is zero, then the fitting operation is interpolation to the points; otherwise, it indicates an approximation operation. Moreover, each operation has its own set of parameters and different bookkeeping methods and structures. For example, if we have: `Rel = {curve ←→ points | incidence: C, P; tol; function}`, then only the access to the curve and point, the tolerance, and the function used to test for incidence need to be recorded. A global or a master structure to fit all types of parameters can be done but will be a waste of memory space.

4.1.6.4 Conditions

The conditions that are applied to generate relationships must be the same across all functions and platforms. For example, the reason why two lines in two different systems are classified differently—e.g., parallel in one and intersecting in another—is not just because of different tolerances, but also because of different conditions [100]. For example, one system can use the cross-product condition of two lines (vectors) to test parallelism and use dot-product to test for perpendicularity, while another system may use the distance conditions to test for the two cases as illustrated in Figure 4.12. The two testing conditions can give different results. Therefore, when migrating models between the two systems, KGN supplies a set of conditions that might be used to determine a relationship.



L_2
V_2
d_1
d_2
L_1
V_1
$|d_1 - d_2| < \varepsilon$
P_2
$|Q_2 - I| < \varepsilon$
$|Q_1 - I| < \varepsilon$
Q_2
Q_1
L_1
I
P_1
L_2

Figure 4.12. Using distance condition. Test for parallelism (left) and perpendicularity (right)

4.2 Knowledge Acquisition Manager

This section explains how knowledge is acquired and fed into the KGN system. The five main sources of knowledge are construction, enforcement, reclassification and double bookkeeping. See Figure 4.13.

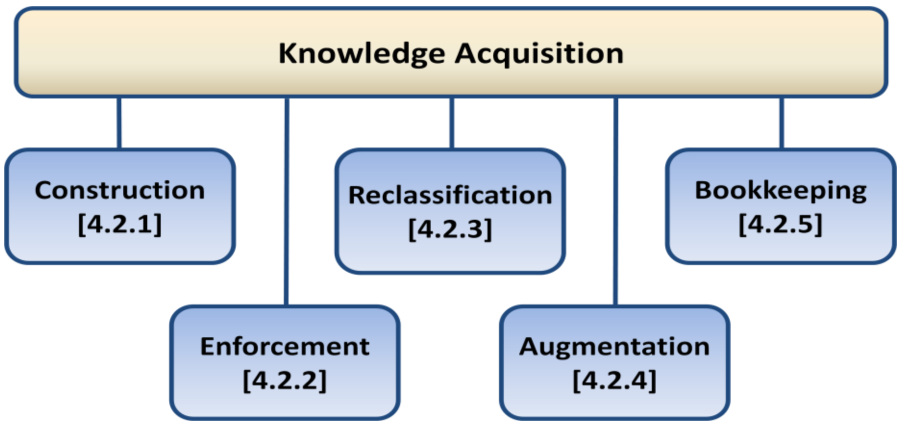


Figure 4.13. Knowledge acquisitions for knowledge guided NURBS systems

4.2.1 Construction

The greatest amount of knowledge entered the system occurs when constructing entities from other components: e.g., a curve from control points, and extruded surface from a curve, etc. New pieces of knowledge are introduced with every incremental designing step. Once the construction has been completed and all relevant relationships with their parameters are entered into the relationship graph, the types, origins, destinations, design intent, etc. are all saved. All this is done automatically without user intervention. If the designer requires making any kind of change, the knowledge base is sufficiently populated with information to learn about the design and to make the right decision. Any kind of alterations as well as documentations and design maintenance are possible with the knowledge acquired at this stage of the designing process.

4.2.2 Design Intent Enforcement

We have discussed design intent as a mean of enabling design replay—i.e., the ability to support and capture intent within KGN. However, there is yet another side of design intent that can be used in knowledge-based systems, called intent enforcement. When a model is migrated between incompatible systems, many times there is a discrepancy between the resultant model and the intended output—e.g., the user intended objects to touch; however, due to the tolerance(s) incompatibility, a system might return no touching objects. The original intent can be enforced through design replay, modifying conditions, or sometimes doing the repair manually if the designer prefers. The enforcements usually result in new pieces of knowledge that have to be recorded in the knowledge base.

In some cases, the discrepancy between the intended design and the resultant design is subtle, and changing the offending part may cause more damage to subsequent processes, such as manufacturing. In such cases, the offending part may be left as is; however, the designer might enter new pieces of knowledge, such as by flagging them with the intended outcome.

4.2.3 Reclassification

When entities enter the supply chain of companies that use different incompatible CAD systems, and the resultant model is not the intended model, the first approach to rectify the discrepancies is to *enforce* the design intent, as stated above. However, it could be the case that enforcing the intent is not the best choice. For example, the change can propagate to other relationships causing more serious contradictions. Figure 4.14 (a) shows that System A can have two lines that are perpendicular and one is tangent to a circle based on its local tolerances; Figure 4.14 (b) demonstrates that System B finds the two lines are not perpendicular anymore based on its local (tighter) tolerances; Figure 4.14 (c) explains that if we enforce the perpendicularity on System B, the tangency relationship might not hold any more, causing contradictions. In a worst case scenario, the system can have a regression that affects the entire knowledge base. In such cases, a second option is to *re-classified* objects relationships upon entering the receiving system. See Figure 4.14 (d). However, how can reclassification participate in knowledge acquisition?

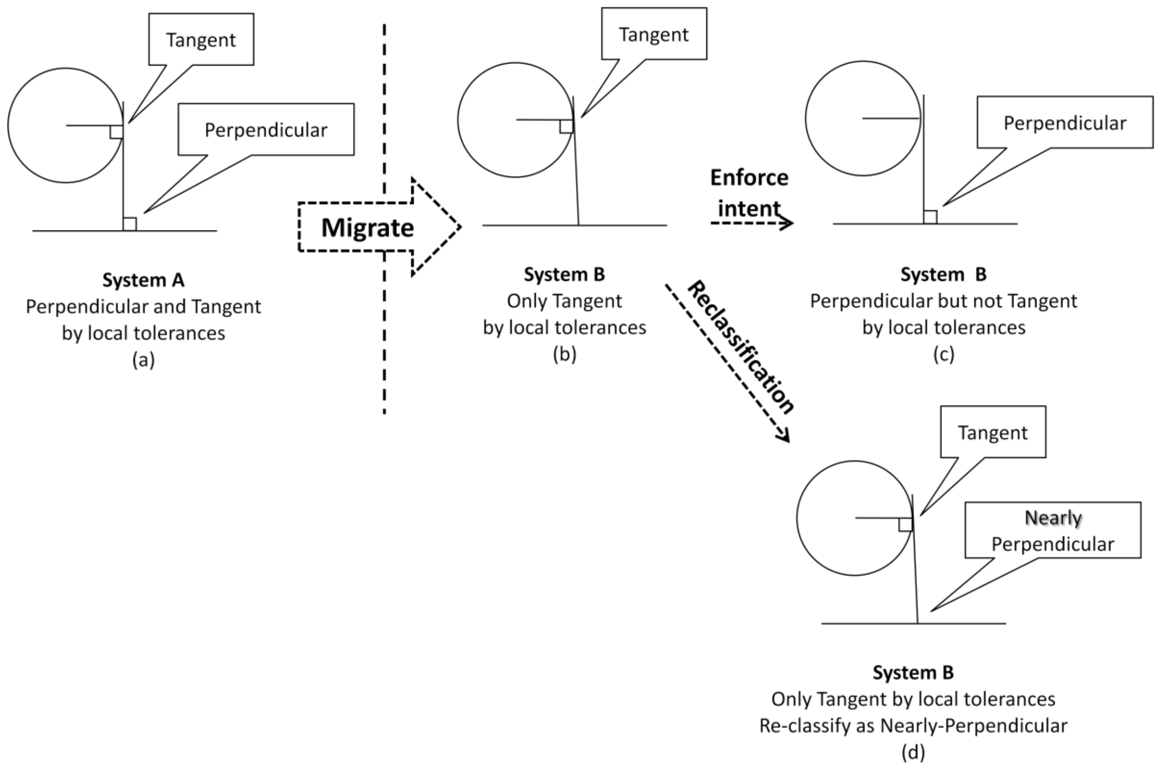


Figure 4.14. Contradictions caused by enforcing intent

Reclassification can cause the old stored knowledge to be erased or modified and may generate a new piece of knowledge. In both cases—either modification or generation of new knowledge—the system acquires a new form of information and must record it. The relationship graph has to be updated with the new acquired "non-parallel" relationship as well as the local tolerance used to test for parallel cases. This change must also be reflected in the history file so that the designer at the next system is aware of the situation and can make proper decision based on the new circumstances.

Tagging objects as non-parallel while they are almost parallel is a bit extreme, and many designers might get reluctant to change or re-classify the relationships. KGN introduces intermediate cases such as nearly-parallel, nearly-perpendicular, etc. in order for the designer to re-classify relationships more appropriately for different situations.

4.2.4 Design Augmentation

At the receiving end of a CAD design chain, inconsistencies must be removed by either design intent enforcement or reclassification. If both intent enforcement and re-classifications are not preferable choices by designers, KGN provides design augmentation as a third choice. Augmentation, as anther knowledge acquisitions capability, introduces new pieces of knowledge that will be recorded in the knowledge base.

If we take the same example used above, in which two lines that are parallel in one system and non-parallel in another due to differences in tolerances, it is sometimes the case that the receiving system finds neither a need to enforce the parallelism intent nor a need to re-classify the relationship. The geometries of the lines can be taken as they are and be used without drastic effects on subsequent operations. In this case, the receiving system will augment the knowledge base with: "lines are parallel in the sending system", e.g., System A, "but not on this system", e.g., System B, "and the results are obtained by taking the geometries as they are" without altering the old relationship graph and the databases. When the design goes to a third system, e.g., System C, it should understand that the geometries are those generated on a system other than the sending system (i.e., System A); however, numerical results were used to make the decision. System C can acknowledge this fact and act accordingly since System B augmented the knowledge base with the decisions taking at the time.

4.2.5 Bookkeeping

A final form of knowledge acquisition is due to the fact that most operations can produce more than one relationship. This is because of inquiries about all entities involved in creating them. For example, when two curves intersect, the curves generate a *curve-curve-intersection* relationship. However, NURBS generates an intersection point that is incident on both curves. It is clear that the system should record the original relationship (*curve-curve-intersection*) but what about the generated extra piece of knowledge (*curve-point-incident*): is it of any benefit? KGN finds this extra knowledge to be important and can aid with many operations; therefore, it keeps record of these new pieces of knowledge by double bookkeeping. For example, if another system (System B) finds that the intersection point is not positioned on either of the two curves because of different tolerance requirements, the reason can be identified quickly by realizing that an intersection routine generated the point; and the tolerance used on this system (System B) is not tight enough. If the systems do not keep record of this extra piece of knowledge and do not link the point to any of these curves, the point will be considered a floating or general entity, and there is no means by which we can rectify the intersection appropriately on System B.

4.3 Knowledge Mining

Once the knowledge and design bases are populated with design data and information, a knowledge guided system has to be able to mine them in order to make good uses of this wealth of information. This section explains how the acquired knowledge can be mined to assist with a range of activities as well as the capabilities that a knowledge-mining manager supports. The section starts by discussing the benefits of knowledge mining and later details the ways by which knowledge can be mined.

4.3.1 General Usage of Knowledge Mining

Knowledge mining in KGN can assist as a minimum with the activities outlined in Figure 4.15.

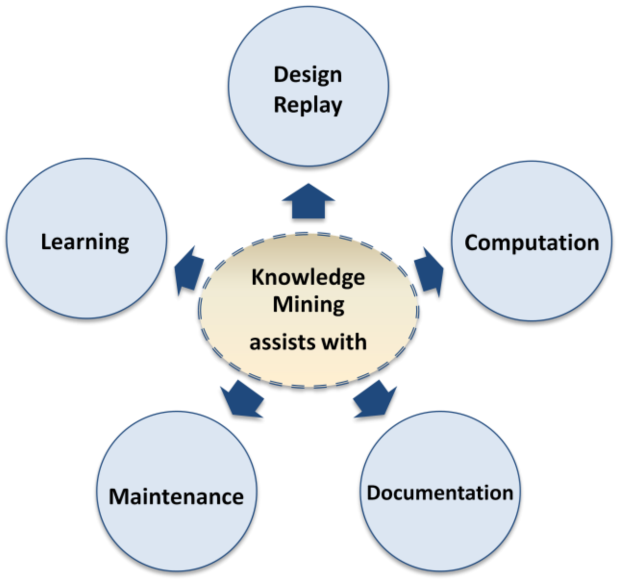


Figure 4.15. Knowledge mining usages

4.3.1.1 Design Replay

A very important application of knowledge guided NURBS systems is to overcome issues stemming from cross-platform incompatibility, such as the flawed model migration between systems. This is the duty of the *design replay manager*, explained in the next section of this chapter. However, it cannot achieve its task without the critical support of knowledge mining. When a design or part of it has to be reconstructed—e.g., the intersection curve between two surfaces is found to not lie on the surfaces—having as much knowledge about the participating entities is a must, and this knowledge must be retrievable.

4.3.1.2 Computations

When many operations are performed in KGN, the knowledge base is mined to find useful information that leads to more robust and reliable computations. A simple example is to find the distance between set of points and a NURBS entity. The type of the entity—e.g., line or circular arc—and the awareness of the presence of any irregularities—e.g., loops and cusps—are used to speed up and increase the reliability of computations. Point-distance computation routines treat lines and circular arcs differently than a curve with irregularities. Another example is offsetting a curve. The computation routines can be more accurate if the system distinguishes that we are dealing with a circular arc. The awareness of irregularities assists in avoiding the Newton's type computations pitfall and leads to a faster convergence. Blind computation should not be the case with the available knowledge that KGN provides and the knowledge mining is a key player in achieving that.

4.3.1.3 Design Documentation

Although the knowledge base is the main form of documentation in KGN, participating companies may want a different form of documentation. The designing process is documented at the generating system: e.g., types, destination, design intent, decisions, etc. The receiving system can use knowledge mining manager to gain access to information such as entities, relationships, and parameters to form extra documentations for the design.

4.3.1.4 Design Maintenance

Designing systems and models need to be updated as time goes on. Certain parameters and measures—e.g., tolerances—that were sufficient at the initial designing process may go out of date, and maintenance (or update) may be required. Knowledge mining provides the ability to browse the knowledge base to find all entities and the parameters that can be affected by such updates, that is before the system applies the changes to the model via design replay.

4.3.1.5 Learning

Mining the design and knowledge base to learn about a design can be done for many reasons. Example-based design is a powerful method of accomplishing great new designs and can be achieved easily in KGN by mining a similar model to learn how it was created. Mining the knowledge base to find information on design alternatives, functions, decisions, and history is a way to educate designers about past designs and why it was design the way it is. Moreover, the structure of the knowledge base can be mined sequentially allowing the simulation of the designing process as a learning tool about a design.

4.3.2 Knowledge Mining Manager

A knowledge base is usually structured in such a way that it allows for information access and retrieval. Theoretically, every piece of stored data and information in the knowledge and data bases can be browsed, and in many different ways. However, it would be confusing, misleading, and overwhelming if a knowledge guided system allows users access to about everything in the bases. It is the mining manager's responsibility to present the information in the right order and form for each design task. Figure 4.16 outlines the types of mining/browsing capabilities that are supported.

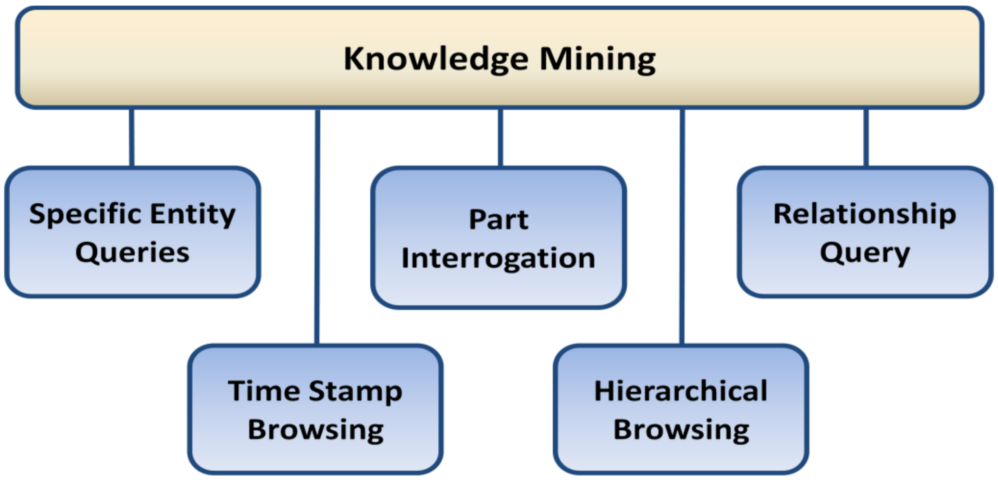


Figure 4.16. Knowledge mining for knowledge guided NURBS systems

4.3.2.1 Specific Entity Queries

This type of mining gathers information about specific entities and their relationships. The queries are submitted in a question form and the response can be saved in the relationships map if the designer chooses to. The following are some examples referring to line and plane entities:

Are line and plane parallel? The response could be one of the following:

• line and plane are parallel	(parallel relationship)
• line is on the plane	(coplanar relationship)
• line is in a general position	(general relationship)
• line is degenerate case	(degenerate relationship)

Are the line and plane perpendicular? The response could be one of the following:

• line and plane are perpendicular	(perpendicular relationship)
• line is in a general position	(general relationship)
• line is degenerate case	(degenerate relationship)

4.3.2.2 Part Interrogation

This type of mining reveals all the stored knowledge about one specific entity. It starts with brief information about the entity. If the designer requires more information or “interrogation” of the object in question, the system mines deeper and deeper until all relevant information is displayed. An example: given an offset curve, what do we know about this curve? After a quick response on identification, date of creation, and classification (type, origin, and destination), more mining may be needed to find all relevant information, such as the base curve, the offset distance, and the tolerance was used to approximate the offset, alternatives, decisions, etc.

4.3.2.3 Relationship Query

This type of mining is concerned with the relationship and can come in the following three subtypes:

- Find all the entities that are in immediate relationships. It returns the objects participate in the creation of the entity in question, and all the objects use it in their creation—i.e., created-from and used-in objects. This is useful for finding the actual origin and destinations (the entities themselves, not a general origin and destination).
- The designer can *recursively* find all distance relationships in a similar way by finding the connected components in an undirected graph. It will return all the entities that could have any type of connection with the object in question, immediate, as in the first type mentioned above, or through other entities. This type of relationship query is very useful on many levels. It can be used to judge the ripple effect or magnitude of any change or update to the design. In other words, how much of the whole design can be affected if the designer *replays* the entity in question. It can also be used to show breaking points of the model if the model needs to be broken into smaller sub-models without breaking a chain of relationships.
- Find all entities under one type of relationship. For example, find all overlapping entities in a part or in the entire design. Parameters and participating entities are sought along with historical updates.

4.3.2.4 Time Stamp Browsing

Mining based on time is a way of finding out what has been changed since a certain date. The system tags every entity with its creation date. If the designer performs some changes, the system will update the knowledge base with the new date by either changing the creation date to the new date (default) or by reflecting the modification date in the history file. When browsing the design based on the date, the designer can see all the changes, updates, or new designs that have been introduced since the date in question.

4.3.2.5 Hierarchical Browsing

This form of mining is for learning about the design as well as for design replay. It can aid with design animation to learn about the whole designing process of a model once a model is complete. The designer can step through the designing stages one at a time with the ability to pause, play back, and fast forward animation. It starts with the complete design and traverses the database in the order the various elements were generated.

An internal use of this type of mining is by the design replay manager. As with the animation, it mines part of the whole model in the order the entities were generated. The design replay, however, applies changes to the mined model by regenerating the entities—e.g., with different tolerances—as if they are designed on the current system the model is on.

4.4 Design Replay

One of the main objectives of knowledge guided systems is to deal with cross-platform incompatibility. This is the main task of the design replay manager. The idea is simple: regenerate the model from all the components that created it in the first place but perhaps with some different parameters and tolerances. Even though this sounds very simple, the process is very complicated and only is made possible by what knowledge acquisition and mining managers have provided. Figure 4.17 illustrates the interaction of the three managers and how design replay is the core of the editable system.

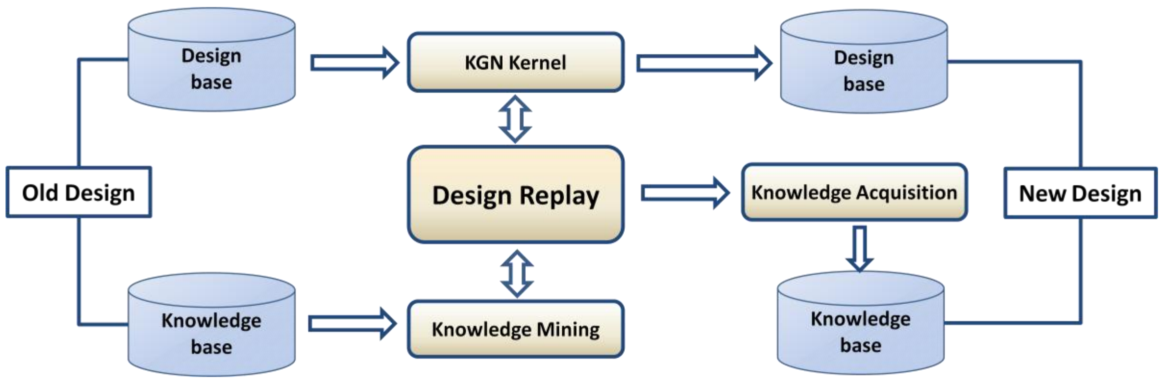


Figure 4.17. Design replay interaction with other managers

Successful design replay requires the following capabilities.

- Hierarchical knowledge base browsing; this is a part of the knowledge mining manager that we discussed in Section 4.3.2. It returns data, information, and all parameters in the same order the design occurred.
- Retrieve design functions and alternatives as well as previous decisions to redesign the model but avoid repeating past mistakes. The mining manager will handle this task too.

- Access to a compatible modeling kernel and the design user interface. That is, the receiving end must have the same capabilities as the sending end. This is when the entities' proper naming with version control can be utilized to overcome upward and downward compatibilities.
- Update of knowledge and design bases. This is a very involved process and needs well-structured databases to allow for changes that can cause ripple effects (regression update) requiring a lot of updates of several entities and relationships. Moreover, it is sometimes the case that these updates may produce inconsistencies, as discussed in design intent enforcement as part of knowledge acquisition manager. In the worst case (if the inconsistencies cannot be removed) the entire design may have to be redone with a new set of requirements. A process can be performed with a very simple button click that hides all involved complexities form the designer.

An example is generating a ruled surface by the following steps: first, two curves interpolated two set of points; and second the curves are used to create the surface. If the ruled surface is not acceptable in the receiving system, any or all of the steps can be repeated with new parameters. Whether individual step or the entire process is repeated is the decision of the application engineer who may consider all available knowledge. The design replay invokes hierarchal browsing to find all the data and information involved in the initial constructions of the curves and the ruled surface. It also requests the mining manager to supply it with the functions, alternatives, and previous decisions taken. Finally, it accesses the modeling kernel (KGN) and invokes the designing functions or alternative functions with new parameters which are suitable for the current system. The process is done in such a way that the intended design will be recreated with almost no user interventions.

4.5 Export/Import Managers

Data structure, memory management, and input/output files processing are the main tasks of this manager. The export part converts the data structure (in-memory database) for both the model and relationship map into a KGN file, while the import manager's task is to take a KGN file, and then allocate and populate a dynamic data structure with the data and information it contains. The cycle of the the process is illustrated in Figure 4.18.

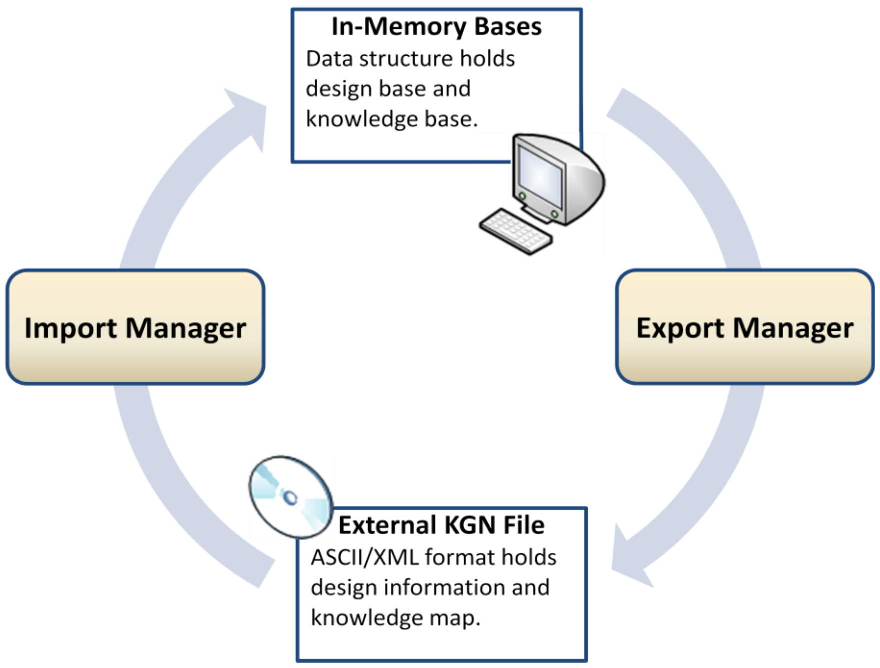

In-Memory Bases
Data structure holds design base and knowledge base.
Import Manager
Export Manager
External KGN File
ASCII/XML format holds design information and knowledge map.

Figure 4.18. General view of import/export managers' functionality

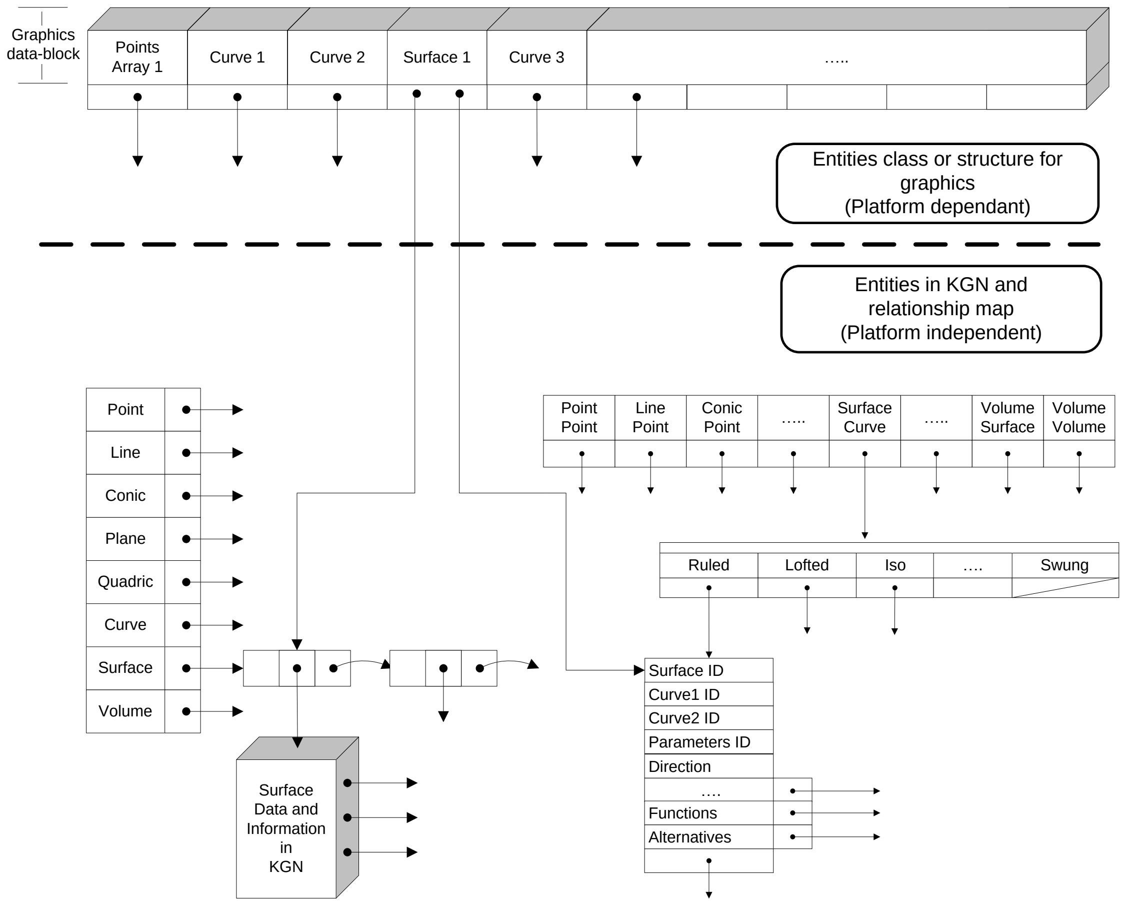


Figure 4.19. Import/export manager accounts for three types of information: system specific (upper); KGN entities information(lower, left), and knowledge tree (lower, right)

When exporting or importing, the system accounts for three types of information that are placed in disjointed locations (structures), illustrated in Figure 8.19. First, the system specific data that is common to all entities is accounted for. It consists of a set of flags and variables as well as some references to the objects in relation with particular shape. They are dependent on the system implementation and the type of graphic engine. In other words, it is the data that is targeted to support the interface. Second, KGN specific data and information, such as control points, knot vectors, classification (type, origin, and destination), representation, etc., is accounted for. The export and import managers account for every type in the KGN system differently. Third, the knowledge tree that holds all recorded relationships in a stack like structure is accounted for. We have recognized 84 sub-relationships. As with the KGN specific data, the export and import managers have to account for every sub-relationship data in a different manner.

4.5.1 Data Structure and Memory Management

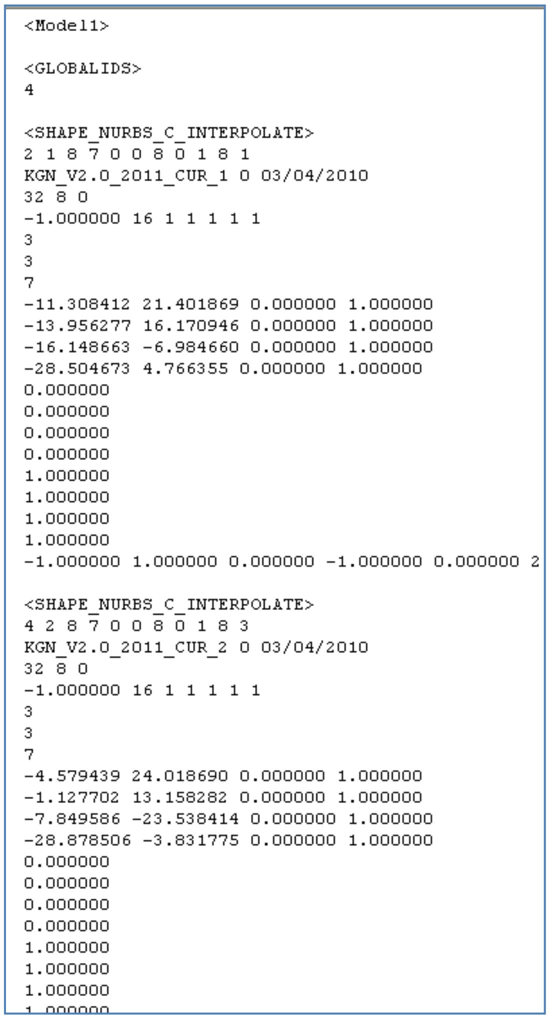
KGN system on the surface is similar to all CAD systems, highly interactive and has to deal with a lot of graphic objects that might be translated, edited, and deleted. The frequent data structure access is highly coupled with two processes: (1) numerical and geometrical operations performed by the KGN routines and (2) the frequent repaint of the model on the screen performed by a graphics engine such as OpenGL.

The data structure for both the design base and knowledge base along with memory management for the NURBS routines are integrated in KGN as part of the kernel. KGN deals with the fact that all memory allocation is dynamic, which is much harder to manage since the system has to deal with unfulfilled memory requests, memory leaks, etc. Improper allocation, using, or deallocation can cause serious errors and cause the system to crash. Therefore, KGN employs very strict memory rules through a set of memory routines. The export manager does not allocate any memory directly. It utilizes these special memory routines.

The data structure that is targeted for the graphic engine is separated and is part of the interface design. A poorly designed data structure can play significant roles in degrading the application performance, and hence must be considered with great attention. Most CAD systems are multi-threaded with four viewing ports, memory management routines have to deal with concurrency issues. For example, one viewing port might try to access the data structure for the repainting purpose, while another active port might want to update the data structure with some new values. The protection of memory manipulation has to account for any critical sections.

4.5.2 KGN Files Structure

The export manager organizes a KGN file as a heap file where records are inserted and stored in their chronological sequence. The file starts with a header holding some global information about the model and the file itself. Each record starts with a header tag bearing the type of the entity in order to prepare a node for it in the data structure, followed by some system specific data for that particular shape, and finally all the data and information about the entity. The record is then terminated with a tag showing the end of the record. Breaking a file into individual records assists with fault tolerance importing by isolating and ignoring bad records in case there were errors (corrupted data) in the file. KGN outputs the design data and information in two formats: ASCII and XML as shown in Figure 4.20.



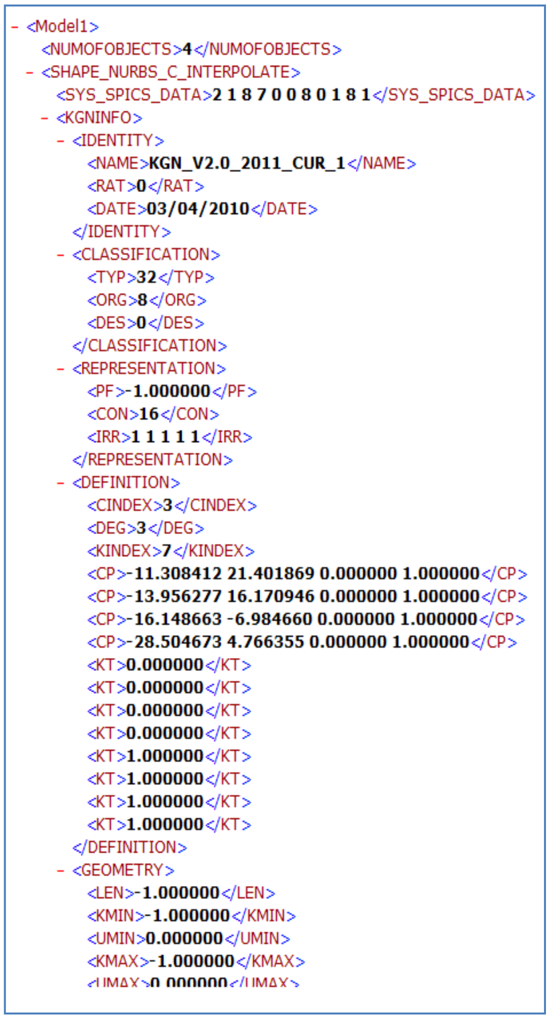
```
<Model1>

<GLOBALIDS>
4

<SHAPE_NURBS_C_INTERPOLATE>
2 1 8 7 0 0 8 0 1 8 1
KGN_V2.0_2011_CUR_1 0 03/04/2010
32 8 0
-1.000000 16 1 1 1 1 1
3
3
7
-11.308412 21.401869 0.000000 1.000000
-13.956277 16.170946 0.000000 1.000000
-16.148663 -6.984660 0.000000 1.000000
-28.504673 4.766355 0.000000 1.000000
0.000000
0.000000
0.000000
0.000000
1.000000
1.000000
1.000000
1.000000
-1.000000 1.000000 0.000000 -1.000000 0.000000 2

<SHAPE_NURBS_C_INTERPOLATE>
4 2 8 7 0 0 8 0 1 8 3
KGN_V2.0_2011_CUR_2 0 03/04/2010
32 8 0
-1.000000 16 1 1 1 1 1
3
3
7
-4.579439 24.018690 0.000000 1.000000
-1.127702 13.158282 0.000000 1.000000
-7.849586 -23.538414 0.000000 1.000000
-28.878506 -3.831775 0.000000 1.000000
0.000000
0.000000
0.000000
0.000000
1.000000
1.000000
1.000000
1.000000
```

(a)



```
- <Model1>
    <NUMOFOBJECTS>4</NUMOFOBJECTS>
  - <SHAPE_NURBS_C_INTERPOLATE>
      <SYS_SPICS_DATA>2 1 8 7 0 0 8 0 1 8 1</SYS_SPICS_DATA>
    - <KGNINFO>
      - <IDENTITY>
          <NAME>KGN_V2.0_2011_CUR_1</NAME>
          <RAT>0</RAT>
          <DATE>03/04/2010</DATE>
        </IDENTITY>
      - <CLASSIFICATION>
          <TYP>32</TYP>
          <ORG>8</ORG>
          <DES>0</DES>
        </CLASSIFICATION>
      - <REPRESENTATION>
          <PF>-1.000000</PF>
          <CON>16</CON>
          <IRR>1 1 1 1 1</IRR>
        </REPRESENTATION>
      - <DEFINITION>
          <CINDEX>3</CINDEX>
          <DEG>3</DEG>
          <KINDEX>7</KINDEX>
          <CP>-11.308412 21.401869 0.000000 1.000000</CP>
          <CP>-13.956277 16.170946 0.000000 1.000000</CP>
          <CP>-16.148663 -6.984660 0.000000 1.000000</CP>
          <CP>-28.504673 4.766355 0.000000 1.000000</CP>
          <KT>0.000000</KT>
          <KT>0.000000</KT>
          <KT>0.000000</KT>
          <KT>0.000000</KT>
          <KT>1.000000</KT>
          <KT>1.000000</KT>
          <KT>1.000000</KT>
          <KT>1.000000</KT>
        </DEFINITION>
      - <GEOMETRY>
          <LEN>-1.000000</LEN>
          <KMIN>-1.000000</KMIN>
          <UMIN>0.000000</UMIN>
          <KMAX>-1.000000</KMAX>
          <UMAX>0.000000</UMAX>
```

(b)

Figure 4.20. Output-file formats generated by export manager

Chapter 5 --- Prototype System and Case Studies

To validate the theoretical aspects of this research, we developed a prototype knowledge guided NURBS system. This prototype system is intended to support our objectives of seamless data exchange, robustness, and reliability. Moreover, it is meant to eliminate any uncertainty as to whether the proposed methods are actually implementable and can accomplish the objectives we claim to achieve.

Chapter 4 presented the tasks of knowledge acquisition, knowledge mining, design replay, and export/import managers. This chapter illustrates how these tasks can be realized in a real CAD system, a prototype system in this case. Some of the system’s internal functionalities are highlighted and presented in visible forms to show their advantages. For example, the relationships map is made visible as a hierarchical tree to represent the internal structure.

Section 5.1 addresses the general design requirements along with the developing environments used in coding the prototype. This section also touches on some of the system interface issues that developers have to account for. Sections 5.2 and 5.3 discuss the implementations and support of both knowledge acquisition and mining managers. We introduce some examples to illustrate the functionalities of the managers in a real software setting.

The last section, Section 5.4, gives case studies to verify the suitability of our approach as an alternative to the currently used flawed migration process of CAD models. It gives different testing scenarios and mainly evaluates the design replay manager.

5.1 KGN Software Architecture

The system is developed on Microsoft Visual Studio 2005. KGN kernel and the four managers are all written in ANSI C to ensure portability of the code. The interface is written using C++, Windows API (Win32), and OpenGL 3.0. Figure 5.1 shows a screenshot of the system in its current state along with a camera model, which was used as a test model. It consists of four viewports: three orthogonal viewports (top, front, and right ports) and one perspective (top-right corner). They are OpenGL viewports that serve as a painting and modeling canvases. The frame, buttons, menu, and status bar are implemented with Windows API. The left window, also implemented with Windows API, holds references to the entities in a knowledge tree. It represents the relationships as parents and the related NURBS entities as children and grandchildren. Note that the *screenshots are in color*. This is because the system is intended to be a test bed, and the color coding in some situations is used to demonstrate the internal functionalities of the system. If the snapshots are reproduced (printed) in black and white, it can be difficult to distinguish between the colors since they will just be different shades of black.

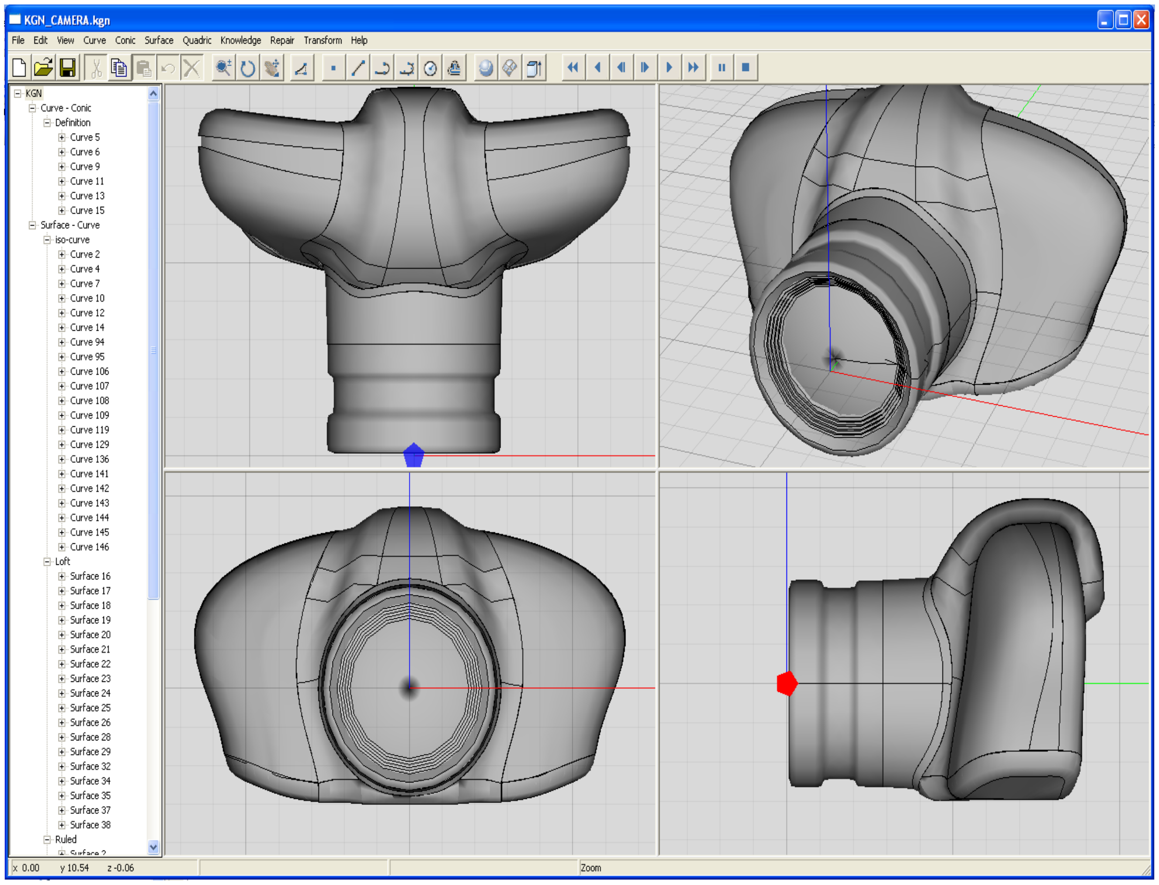


Figure 5.1. KGN prototype system

5.1.1 User Interface

Although developing a user interface is not an objective of this dissertation, a considerable amount of time had to be invested in its design and implementation in order to expose the benefits of our proposed approach in a practical setting. The KGN interface as we would like it to be resembles most CAD systems. It is primarily a graphical user interface (GUI) that is based on the use of pointing devices, a mouse in this case. It has to be rapid and provide reversible actions, informative feedback, simple error handling, ease of use, and consistency. Like any modeling system, the central idea of the KGN prototype is visibility of the objects and actions. Therefore, the GUI should support functionalities present in basic CAD systems such as the following:

- Construction of objects, such as points, curves, surfaces, volume, etc. However, the backend functionality—e.g., computing the control points and knot vector for interpolating a set of points—must be performed using the KGN kernel.
- Rendering of created objects through a graphical package. OpenGL was used because of the following: it provides good support for NURBS; it is an open standard API; and it is available on most modern operating systems, including but not limited to Windows, Mac OS X, and GNU/Linux.
- Picking, translating, rotating, and scaling of objects. Picking is performed mainly on the GUI. Translating, rotating, and scaling of NURBS objects are operations that are performed at the backend by the KGN kernel.
- Navigation of the modeling space: i.e., camera rotation, zooming, and panning. These functions are all independent of the kernel and primarily supported by the interface.

5.1.2 User Interface Issues

Since the four viewports and knowledge tree window all need to be active at the same time, the system needs to be multithreaded. For example, if we are to translate a NURBS curve, all viewports have to reflect the change simultaneously. This brings to the fore some issues that the system has to address: for example, updating the data structure for both the knowledge base and design base, which are shared among the four threads and the repeated repaint processes of the four viewports.

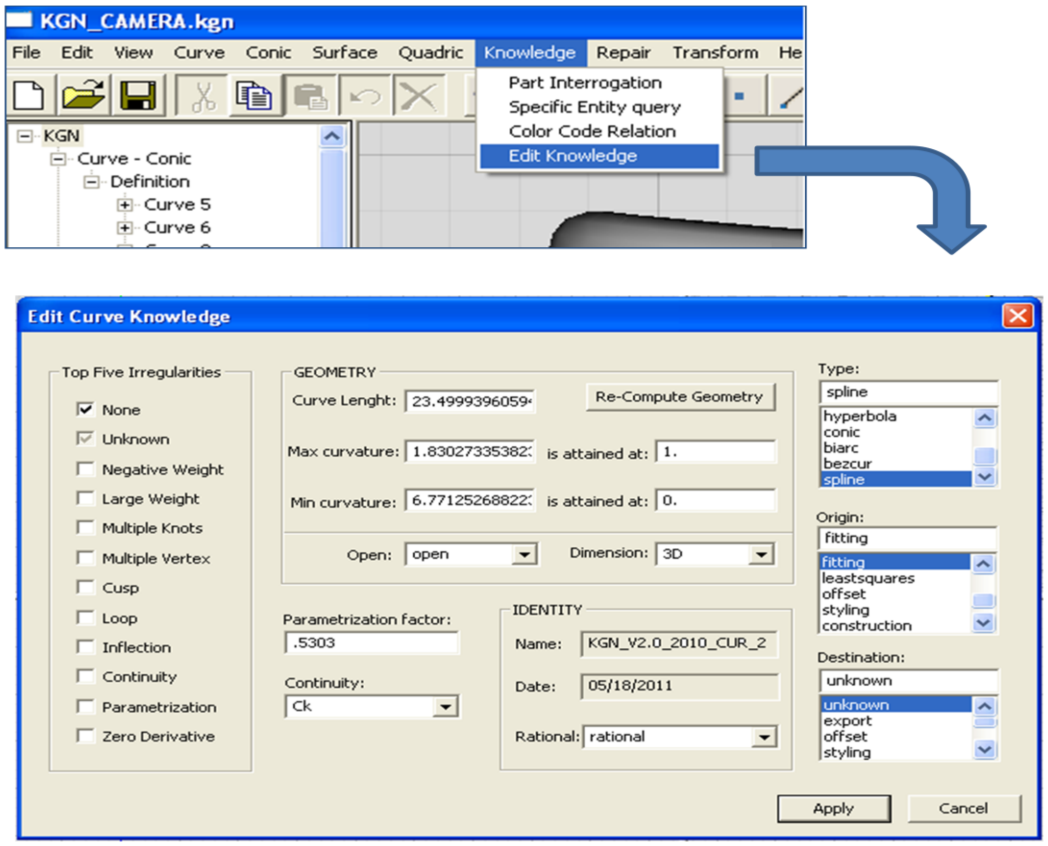
The system has to protect the critical sections in almost every activity that can result in a data structure update. For instance, when a user translates a NURBS curve, the system computes new control points and updates the data structure accordingly. If the system does not protect the critical sections (the control points array in this case), one viewport (i.e., thread) might access previous coordinates while another might access the updated position. This can result in different positions of the curve on two different viewports. A more severe situation arises if one thread triggers a delete to a curve with all its allocated memory while another thread tries to access the deallocated memory assuming it is still available. The system will cause memory access violation and force the operating system to halt it.

The repaint of the viewports is another concern. With a big model that consists of hundreds of surfaces and curves, every repaint of the four viewports will be computationally intensive and can degrade the system's performance. For example, if we rotate the scene in one viewport, the other three viewports should not be repainted. Since the system is multithreaded, it is not difficult to keep track of the active viewport and it only triggers a repaint command when it is most necessary.

Implementation details of the GUI (such as mapping between OpenGL coordinates and window's (screen) coordinates), the use of double buffering to eliminate any flickering, the coupling of the viewports and the window frames when resizing the frame, and the release of all memory and resources that are used by the GUI during a designing session were considered and tested thoroughly.

5.1.3 Knowledge Base Edit and Update

The amount of information contained in the knowledge and design bases can be overwhelming. Although every piece of stored data and information has its purpose and contributes to the KGN system proposed objectives, it is necessary not to confuse the users, especially beginners, with what to modify and what to keep unchanged. At the same time, updating the knowledge base is a very delicate issue. Several related portions of the knowledge base might have to be changed in order to maintain its coherence and integrity. Therefore, KGN permits only high level functionalities to the user while hiding all the low level details.



Top Five Irregularities

- [x] None
- [x] Unknown
- [ ] Negative Weight
- [ ] Large Weight
- [ ] Multiple Knots
- [ ] Multiple Vertex
- [ ] Cusp
- [ ] Loop
- [ ] Inflection
- [ ] Continuity
- [ ] Parametrization
- [ ] Zero Derivative

GEOMETRY

Curve Lenght: 23.4999396059

Max curvature: 1.83027335382 is attained at: 1.

Min curvature: 6.77125268822 is attained at: 0.

Open: open Dimension: 3D

Parametrization factor: .5303

Continuity: Ck

IDENTITY

Name: KGN_V2.0_2010_CUR_2

Date: 05/18/2011

Rational: rational

Type: spline

hyperbola
conic
biarc
bezcur
[✓ spline]

Origin: fitting

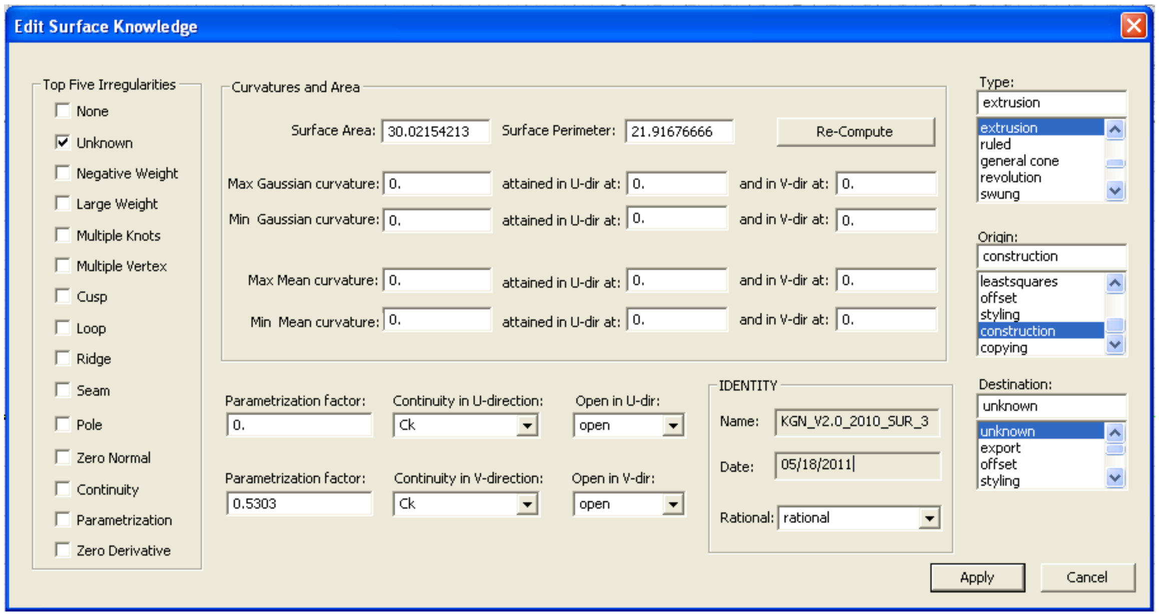
[✓ fitting]
leastsquares
offset
styling
construction

Destination: unknown

[✓ unknown]
export
offset
styling

Figure 5.2. Knowledge edit for NURBS curve in KGN

Figure 5.2 and Figure 5.3 show a curve and a surface knowledge pieces that can be modified in KGN. Some fields are not intended to be changeable and only can be updated by the system such as the entities' names. As previously stated, the names have to be unique. Authorizing the user to modify the names might create unexpected results. Note that the changes caused by the two popup windows (Figure 5.2 and Figure 5.3) do not cause all participating objects in a relationship to be updated. These types of changes that might cause a series of changes are usually caused by, for example, a change of tolerance and usually are implemented through design replay manager.



Top Five Irregularities
- ☐ None
- ☑ Unknown
- ☐ Negative Weight
- ☐ Large Weight
- ☐ Multiple Knots
- ☐ Multiple Vertex
- ☐ Cusp
- ☐ Loop
- ☐ Ridge
- ☐ Seam
- ☐ Pole
- ☐ Zero Normal
- ☐ Continuity
- ☐ Parametrization
- ☐ Zero Derivative

Curvatures and Area

Surface Area: 30.02154213 Surface Perimeter: 21.91676666

Max Gaussian curvature:	0.	attained in U-dir at:	0.	and in V-dir at:	0.
Min Gaussian curvature:	0.	attained in U-dir at:	0.	and in V-dir at:	0.
Max Mean curvature:	0.	attained in U-dir at:	0.	and in V-dir at:	0.
Min Mean curvature:	0.	attained in U-dir at:	0.	and in V-dir at:	0.

Parametrization factor: 0. Continuity in U-direction: Ck Open in U-dir: open

Parametrization factor: 0.5303 Continuity in V-direction: Ck Open in V-dir: open

IDENTITY

Name: KGN_V2.0_2010_SUR_3

Date: 05/18/2011

Rational: rational

Type: extrusion (extrusion, ruled, general cone, revolution, swung)

Origin: construction (leastsquares, offset, styling, construction, copying)

Destination: unknown (unknown, export, offset, styling)

Figure 5.3. Knowledge edit for NURBS surface in KGN

5.2 Knowledge Acquisition Manager

This manager is responsible for gathering information about the entities and populating the design base and knowledge base with the relevant information.

Construction is by far the most involved implementation process in knowledge acquisition. Both the interface and the KGN kernel need to support various NURBS entities to illustrate the diverse knowledge that is acquired with different NURBS entities. For example, knowledge gathered for curves are different than those for surfaces. Knowledge acquisition by enforcing design intent, reclassification, augmentation, and bookkeeping are all supported as part of this prototype.

5.2.1 Construction

A great amount of information enters the system at the initial construction of NURBS entities. Figure 5.4 shows some of the provided construction capabilities that KGN prototype currently support. Although these supported construction techniques are sufficient to illustrate how knowledge can be acquired through construction, a full blown KGN system should provide much more construction techniques. The KGN prototype system automatically updates the relationship map and tags the entities with identification, classification, definition, representation, and geometry, as appropriate. Some of the supported construction capabilities include the following:

- Points. A single point and a set of points can be constructed in KGN. A single point can be considered as a degenerate NURBS curve where the curve and the point can enter into a definition relationship.
- Lines. Given two points, a NURBS curve of degree one (line) is constructed. The system records a line-curve definition relationship that also stores the two points coordinates.

- Curves. Given three data points or more, an approximating or interpolating curve is constructed. The system records points-curve fitting relationship. Providing the control points and knot vector through a file is also supported as an imported curve.
- Iso-curves. Given a surface, an iso-curve can be extracted. The system records surface-curve iso-curve relationships.
- Surfaces. Given a curve, two curves, or a set of curves, the prototype creates an extruded surface, ruled surface, or lofted surface respectively. The system will update the relationship map with the appropriate type of relationship.

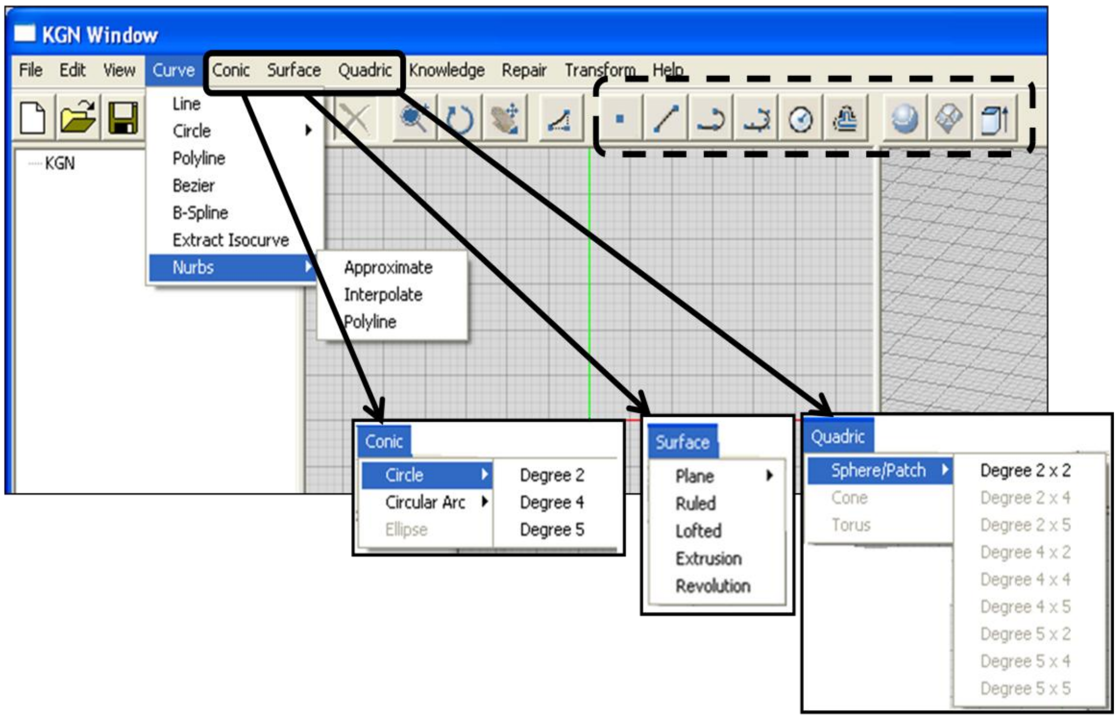
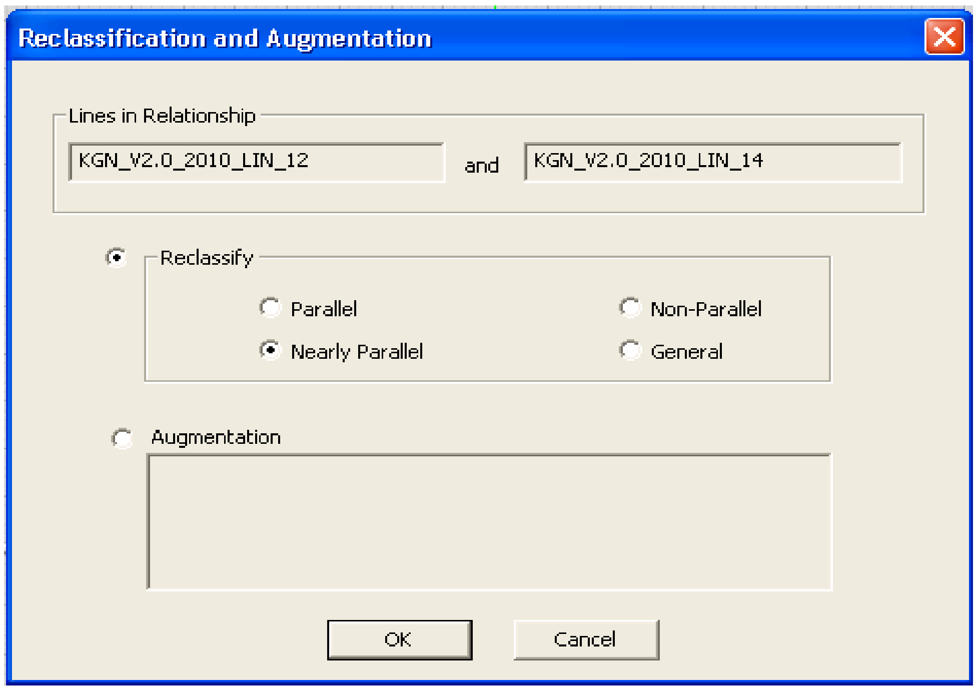


Figure 5.4. Object constructions in KGN prototype system

As an acquisition method, the constructions create new pieces of knowledge without user intervention. All relevant relationships with their parameters are entered into the relationship graph. The types, origins, destinations, and design intent are all saved once an entity is constructed.

5.2.2 Reclassification and Augmentation

Figure 5.5 shows an example of reclassifying the relationship between two lines. The window also gives the user the choice to augment the knowledge base with new information. The system allows either reclassification or augmentation through the utilization of radio buttons. This is because augmentation is only a second option when it is not preferable to reclassify the relationships


Reclassification and Augmentation

Lines in Relationship

KGN_V2.0_2010_LIN_12 and KGN_V2.0_2010_LIN_14

[✓ Reclassify]

Parallel
[✓ Nearly Parallel]
Non-Parallel
General

Augmentation

Figure 5.5. Reclassification of lines' relationships or augmentation

5.2.3 Bookkeeping

Figure 5.6 and Figure 5.7 give an example of bookkeeping that is performed by the prototype system as a means of acquiring new knowledge. There are two coincident curves in Figure 5.6: one is selected (yellow) while the other is exhibiting a z-buffer fighting[1]. The relationships map tree does not show any relationship between the two curves. Figure 5.7 shows a lofted surface that is created from the four curves. The construction of the lofted surface creates a surface-curve-lofting relationship, which is expected. Bookkeeping detects that there are two curves that are in a coincident relationship that create yet another relationship: curve-curve-coincident. The relationship map in Figure 5.7 is marked to indicate the new piece of knowledge that was captured with this type of acquisition.

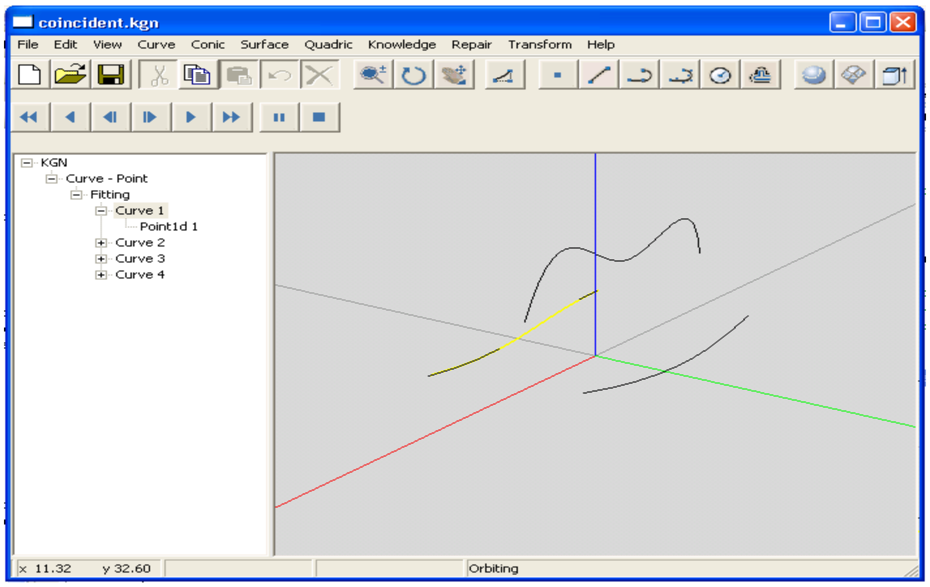


Figure 5.6. Two coincident curves (yellow and black fragments)

[1] Z-buffer fighting is a rendering phenomenon when two entities occupy essentially the same space with neither in front, resulting in rendering overlapping fragments from each curve. The fragments' visibility is dependent on the camera position

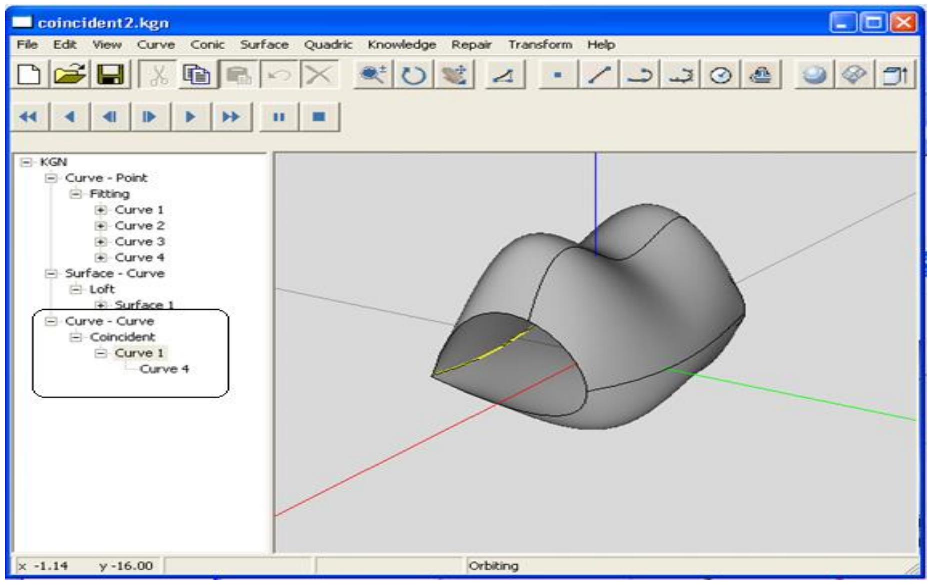


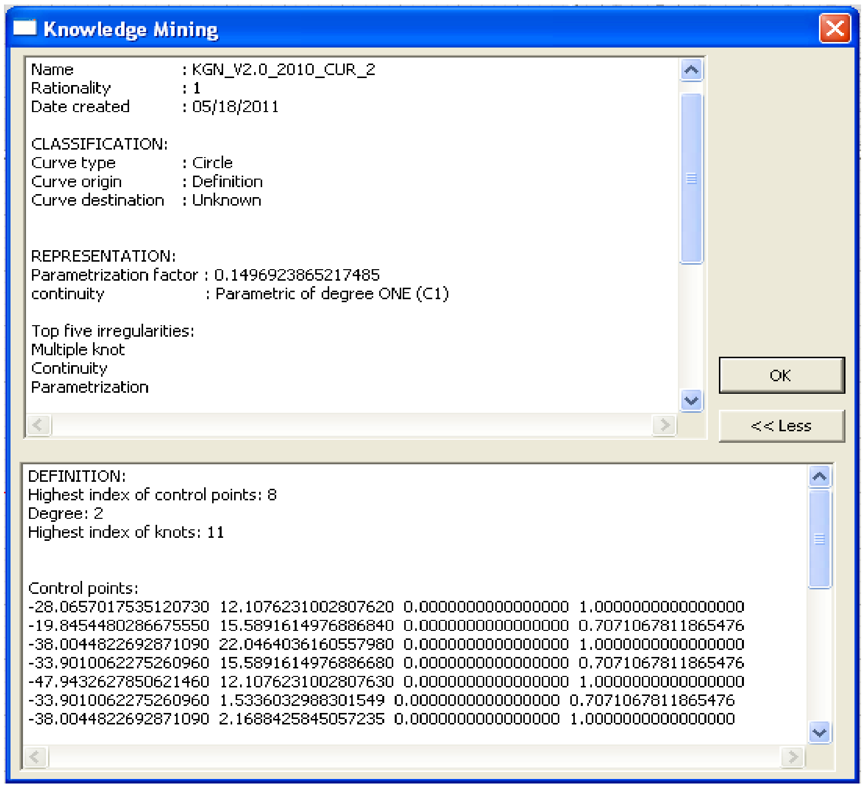
Figure 5.7. Bookkeeping creates coincident relationship (marked with a box)

5.3 Knowledge Mining Manager

Once the knowledge and design bases are populated with design data and information through the knowledge acquisition manager, the system can trigger mining activities to assist with a range of tasks. The following subsections explain the supported capabilities by this manager in the implemented KGN prototype system.

5.3.1 Part Interrogation

In this form of mining, the prototype system allows for finding all recorded information about a particular entity. Figure 5.8 shows the interrogation process of the NURBS curve *KGN_V2.0_2010_CUR_2*. At first, the system presents the most basic information about the entity in question. For a curve, it displays the identification information (name and date of construction) and the classification (type, origin, and destination). The destination in the example curve is still unknown and will be updated when the curve is used in further modeling. The time of creation is recorded and is used for *time stamp browsing*.



```
Name                : KGN_V2.0_2010_CUR_2
Rationality         : 1
Date created        : 05/18/2011

CLASSIFICATION:
Curve type          : Circle
Curve origin        : Definition
Curve destination   : Unknown


REPRESENTATION:
Parametrization factor : 0.1496923865217485
continuity             : Parametric of degree ONE (C1)

Top five irregularities:
Multiple knot
Continuity
Parametrization
```

```
DEFINITION:
Highest index of control points: 8
Degree: 2
Highest index of knots: 11


Control points:
-28.0657017535120730  12.1076231002807620  0.0000000000000000  1.0000000000000000
-19.8454480286675550  15.5891614976886840  0.0000000000000000  0.7071067811865476
-38.0044822692871090  22.0464036160557980  0.0000000000000000  1.0000000000000000
-33.9010062275260960  15.5891614976886680  0.0000000000000000  0.7071067811865476
-47.9432627850621460  12.1076231002807630  0.0000000000000000  1.0000000000000000
-33.9010062275260960  1.5336032988301549  0.0000000000000000  0.7071067811865476
-38.0044822692871090  2.1688425845057235  0.0000000000000000  1.0000000000000000
```

Figure 5.8. Specific part interrogation, NURBS curve (circle degree 2)

The interrogation process can continue until all recorded information about the entity in question is revealed. The parametrization factor, continuity, top five irregularities, and other representation's constituents are presented at the second stage of the interrogation. Finally, the system displays the definition components such as the control points and knot vector along with their highest indexes. The figure shows the interrogation process at its final stage by displaying all associated information to the curve in question.

5.3.2 Relationship Query

This type of mining is performed on the relationship map and is illustrated in two ways. Figure 5.9 uses color coding to visually illustrate the backend of the system. This simple example shows that the curve in question (yellow) is generated from an array of data points (blue) and is used in the construction of the two surfaces (red). Although it is useful to visually identify the entity in relationship, the main focus of this mining is targeted for the system's internal operations, such as supplying design replay manager with the appropriate sequence of relationships as a requirement to reconstruct a model from its components and relationships

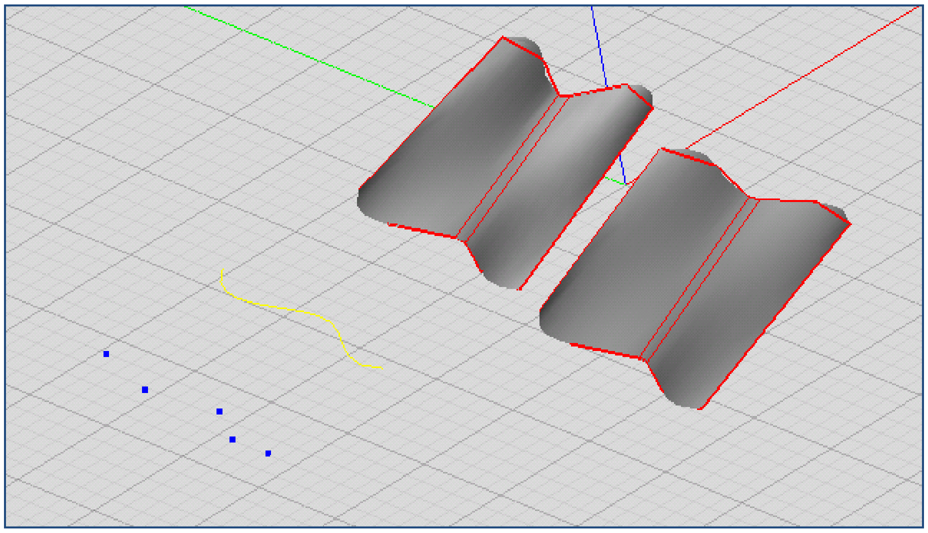


Figure 5.9. Relationships map in KGN: curve in question (yellow) created from points (blue) and is used in construction of 2 surfaces (red boundary)

The knowledge tree (relationship map) in Figure 5.10 is another example of how relationships can be mined and viewed. A user can click on any of the tree's nodes (entities' names) and the system will color code the actual entities in the relationship

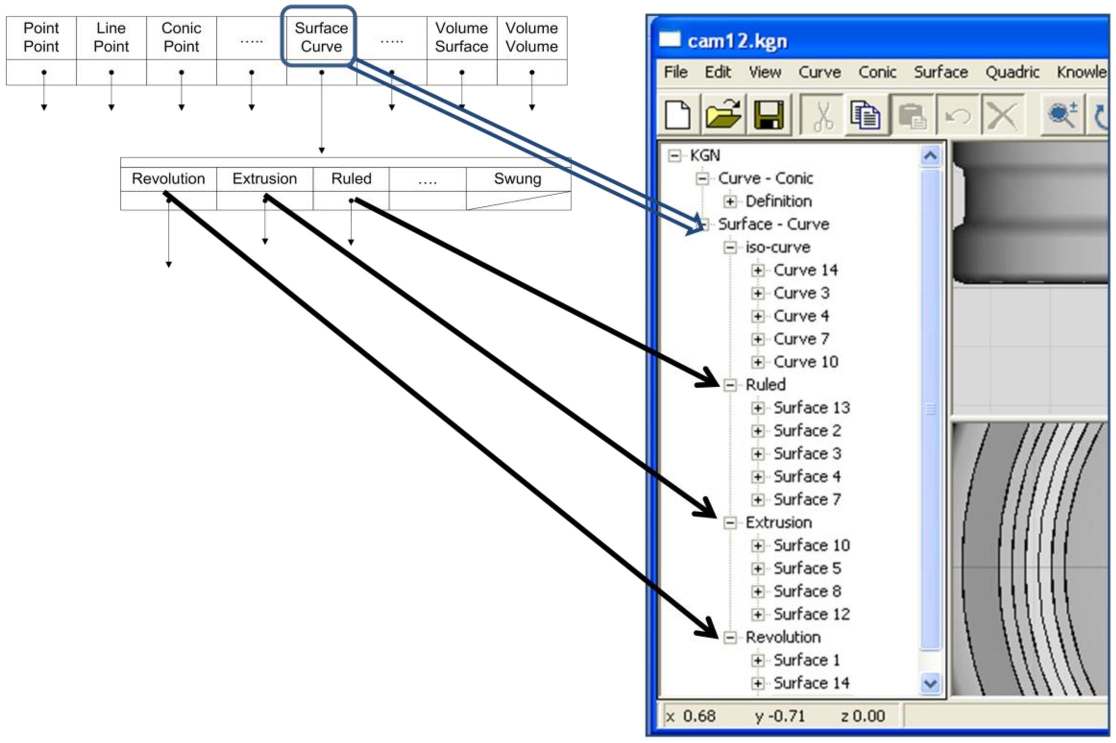


Figure 5.10. Relationships map in KGN represented as a tree

5.3.3 Hierarchical Browsing

An important task of this type of mining is to provide support for design replay manager. The system can internally mine part or the whole of the model in the order to ensure that the entities were generated. It also used for educating a new user about the design through an animated traversal or browsing of the generated model. Figures 5.11 through Figure 5.15 show how the system can hierarchically browse the design base. The white curves shown in Figure 5.11 are the hidden curves used in constructions of surfaces. It is essential to keep a record of them in the database for design replay, as illustrated in Section 5.4. The pause, back play, and fast forward buttons are used to browse the design base. Note that the knowledge trees in the figures indicate that this is actual browsing, not construction where the tree gets updated with new relationships.

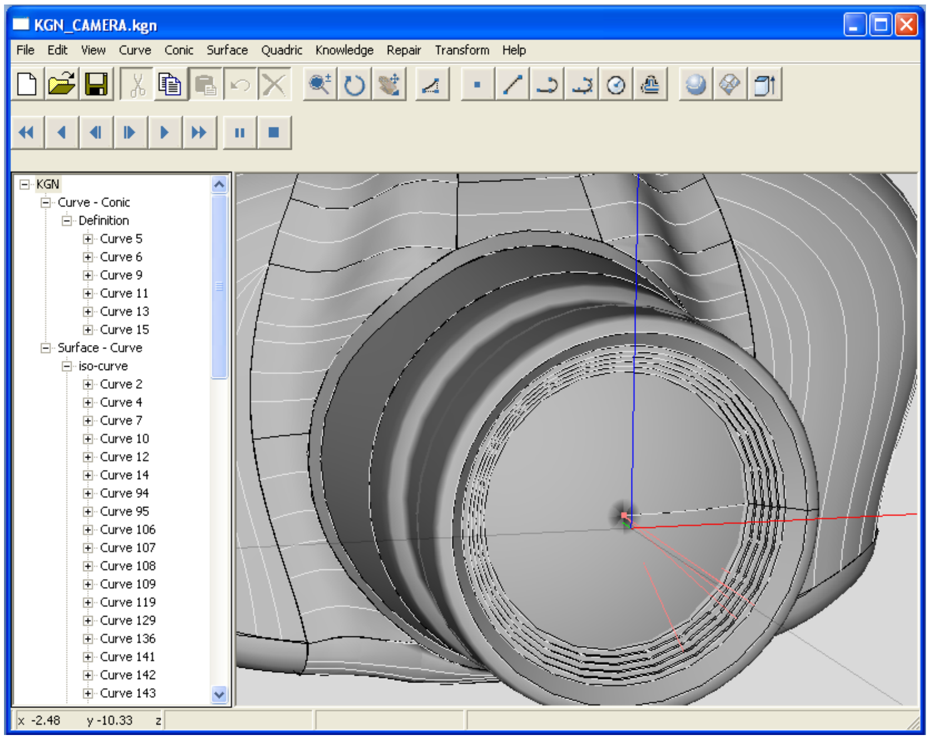


Figure 5.11. Hierarchal browsing (a): the completed model. White indicates the entity is not part of the final model but is kept in the design base for design replay

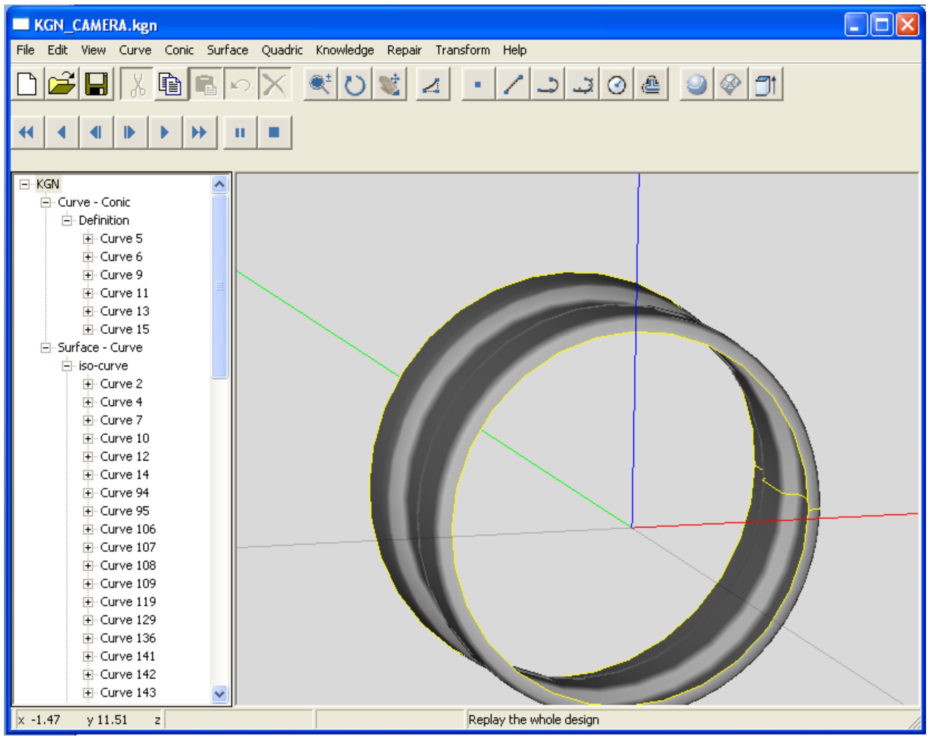


Figure 5.12. Hierarchal browsing (b): after three hierarchal steps, a surface of revolution (yellow boundary) is last displayed.

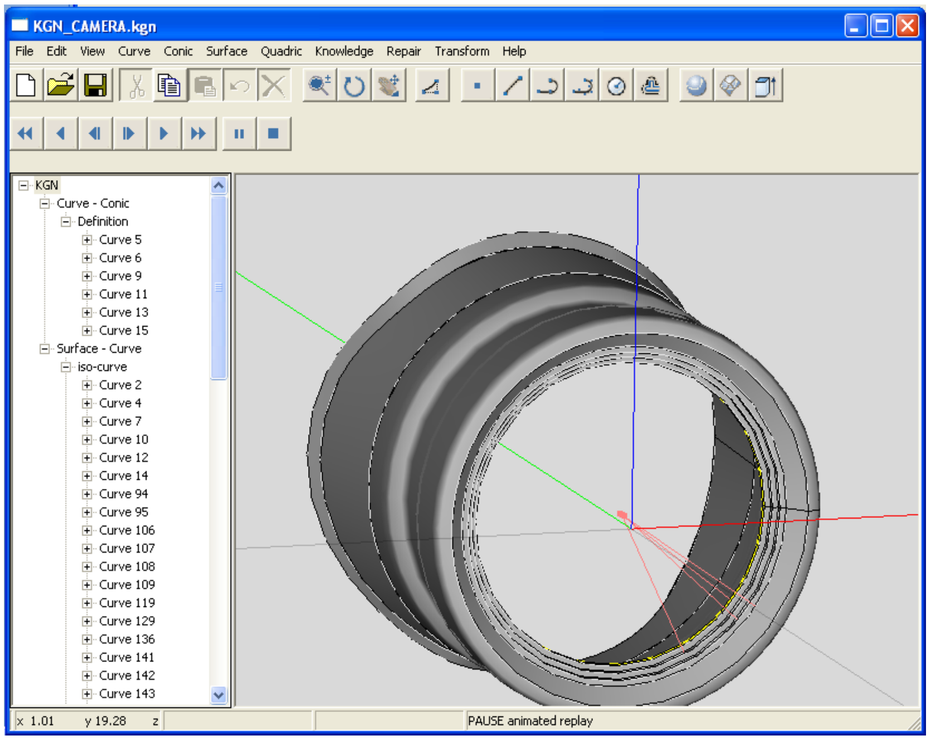


Figure 5.13. Hierarchal browsing (c): after 17 hierarchal steps

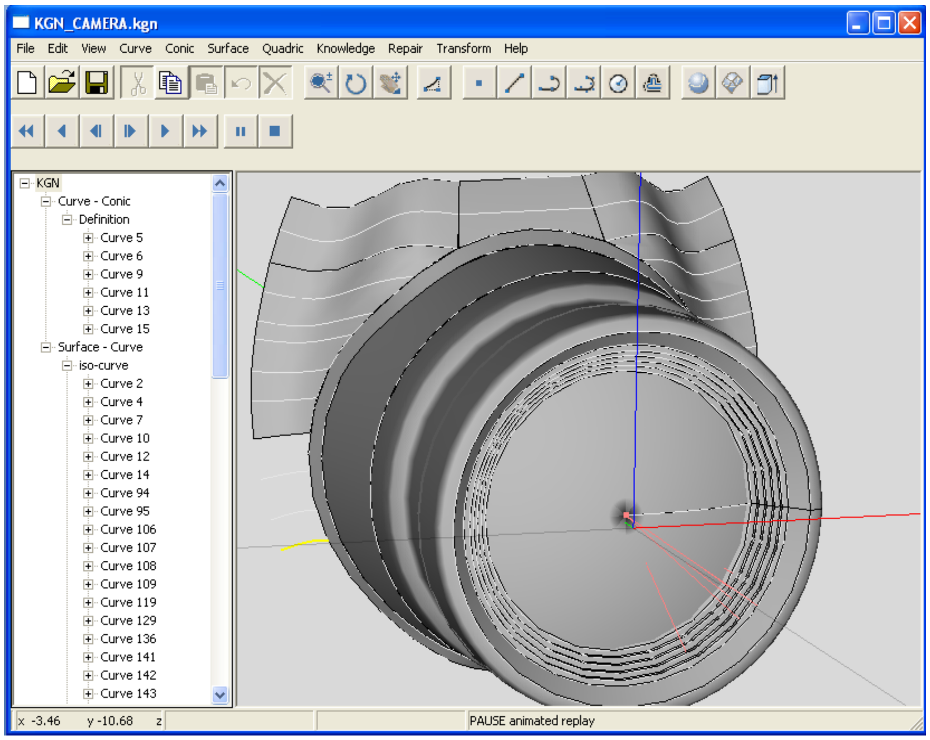
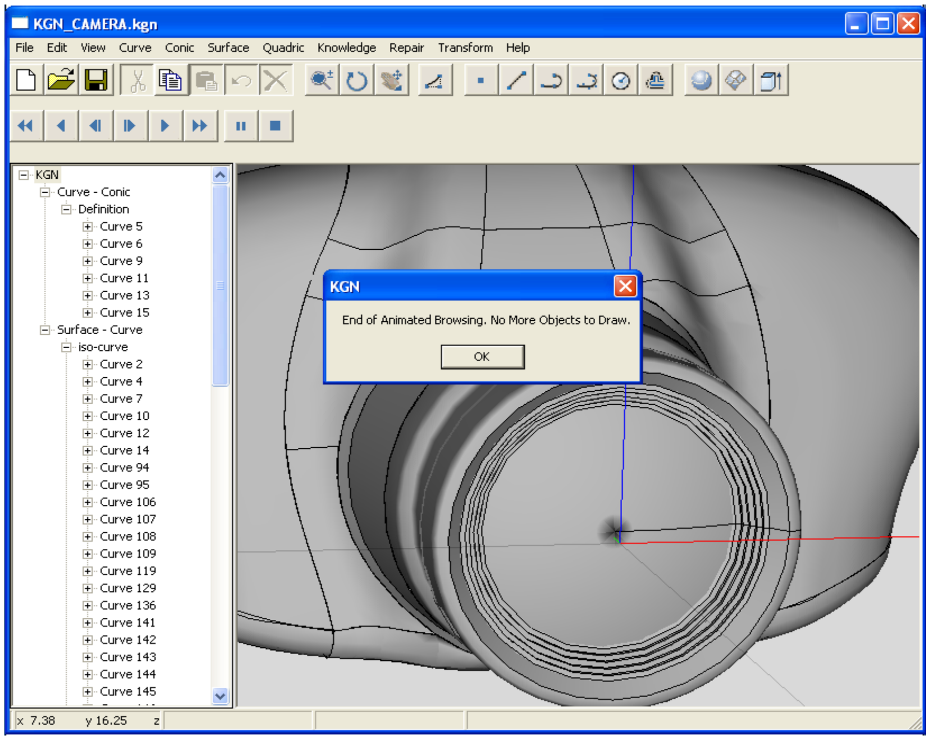


Figure 5.14. Hierarchal browsing (d): after 37 hierarchal steps.



End of Animated Browsing. No More Objects to Draw.

Figure 5.15. Hierarchal browsing (e): no more entities to browse and draw.

5.4 Design Replay and Case Studies

For design replay, we designed a camera as our testing model. See Figure 5.16. The model was chosen so that we can show different NURBS curves and surfaces and the effects of migrating them. For instance, the model consists of the following types of surfaces: extruded, ruled, lofted, and surface of revolution. It also contains circles, free-form curves, and polylines. The model is constructed on our system as close replica of a camera model that is available on Rhino examples folder. This way we can compare the two models after migration between systems using IGES and STEP for the Rhino model and using design replay via the KGN prototype system. To test design replay on a KGN system, we introduce gaps, intersections, and flaying edges manually, and then use the design replay manager to regenerate the model. Different circle degrees are also presented to illustrate how KGN can be used to regenerate the design as if we had three systems that use dissimilar circle representations. To augment the undesirable effects caused by traditional model's migration, surfaces are distorted excessively in some of the testing scenarios, a case that can be irreparable on non-knowledge driven systems.

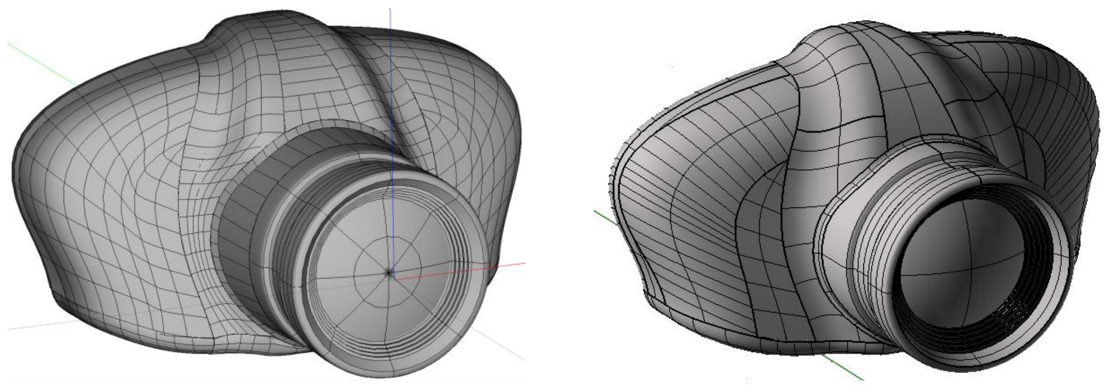


Figure 5.16. Camera model by KGN prototype system (left) and by Rhino (right)

5.4.1 Case Study 1: Model Exchange by Standards

Figure 5.17 shows two very extreme cases of unacceptable migrations of the camera model. In the right side of the figure, the Rhino IGES model was imported to HOOPS default settings. The model seems as if it was constructed from large polygons rather than smooth NURBS surfaces. The model at the lower part of the figure is due to the fact that IGES flavoring can result in ignoring trimmed surfaces. In both cases, the model is beyond repair and a user who needs this model for further applications such as machining, must get a better converted version.

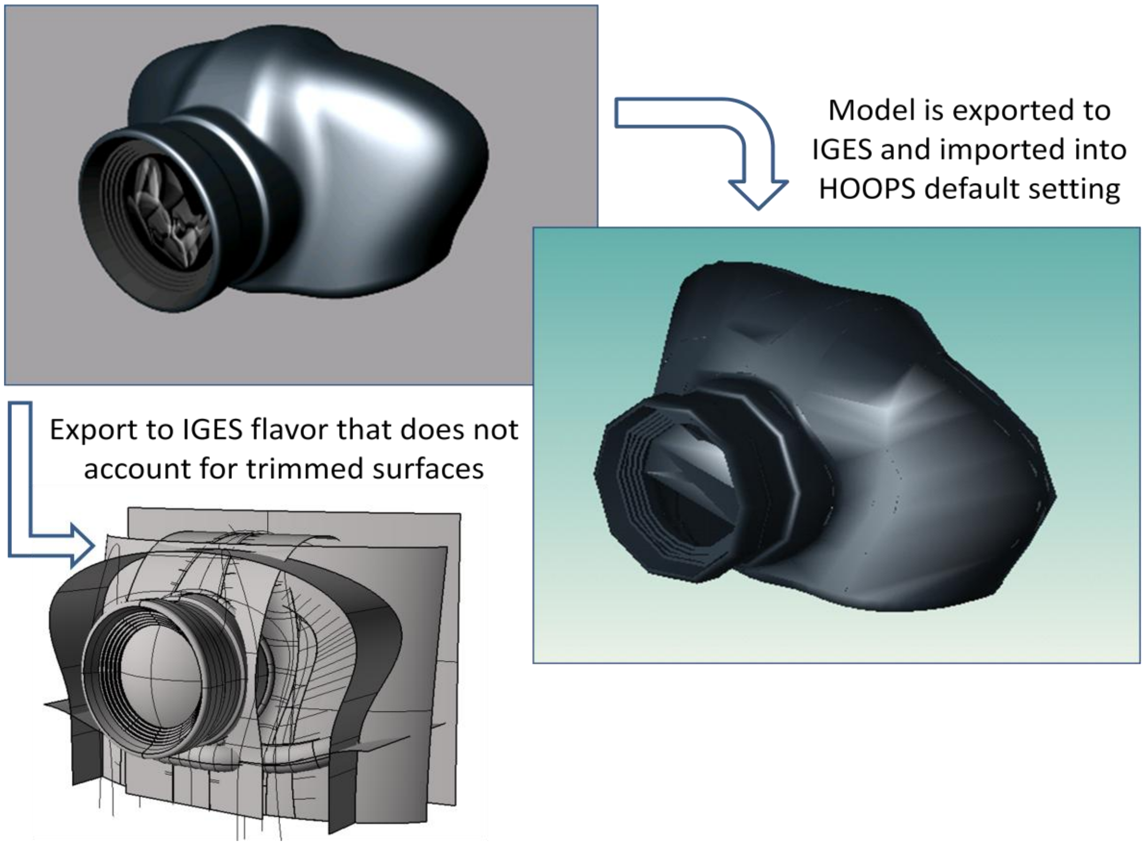

Model is exported to
IGES and imported into
HOOPS default setting
Export to IGES flavor that does not
account for trimmed surfaces

Figure 5.17. Extreme cases of data-driven migration

With better luck, a migrated model can seem acceptable when compared to the extreme cases above. Figure 5.18 shows the original Rhino model (top-left) and three snapshots of the camera after being exchanged using SolidWorks' flavor of IGES. Although it looks flawless, a closer look shows that the model suffers from some cracks, gaps, and boundaries intersections, indicated by red arrows in the figure.

To patch the gaps, a designer needs to create boundary curves around the crack and then fits a tiny surface between them. Generating the boundary curves is not always a straightforward process as can be seen in Figure 5.19 (a). The designer needs to extract boundary curves of neighboring surfaces (yellow) and then split those curves at the appropriate parameters in order to get the desired curves' lengths. Using the extracted boundary curves, Figure 5.19 (b) shows a generated patch that fills the gap.

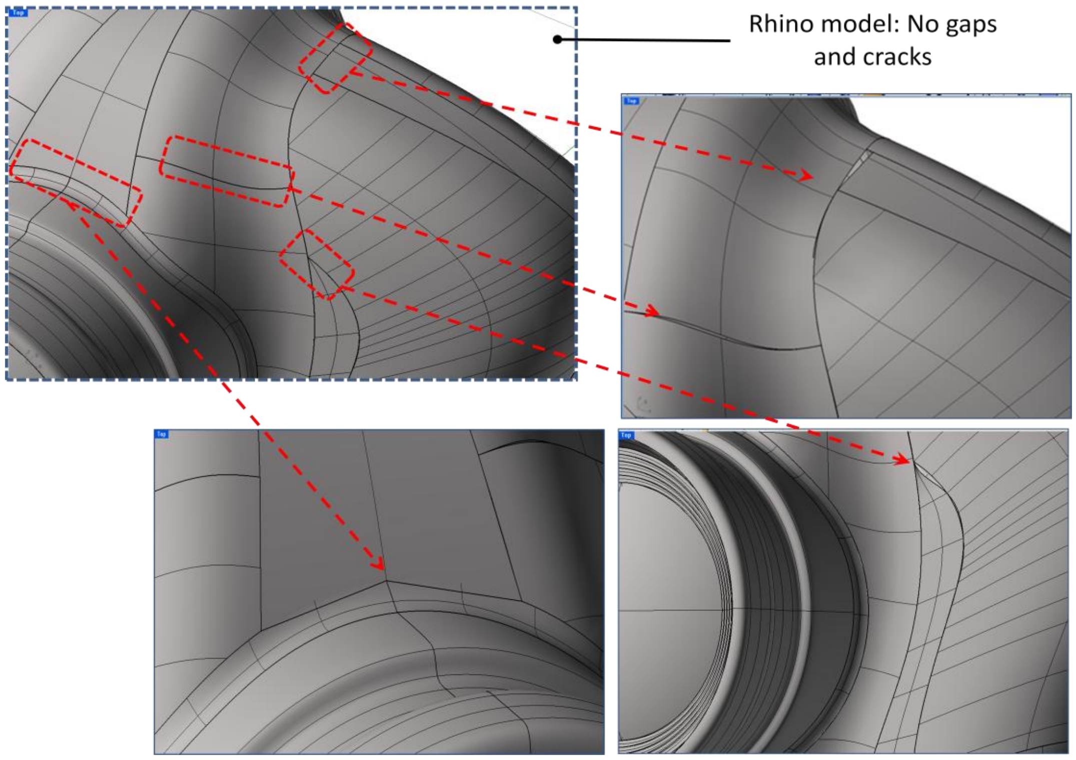


Figure 5.18. Original model (top left), gaps and intersections (indicated with arrows)

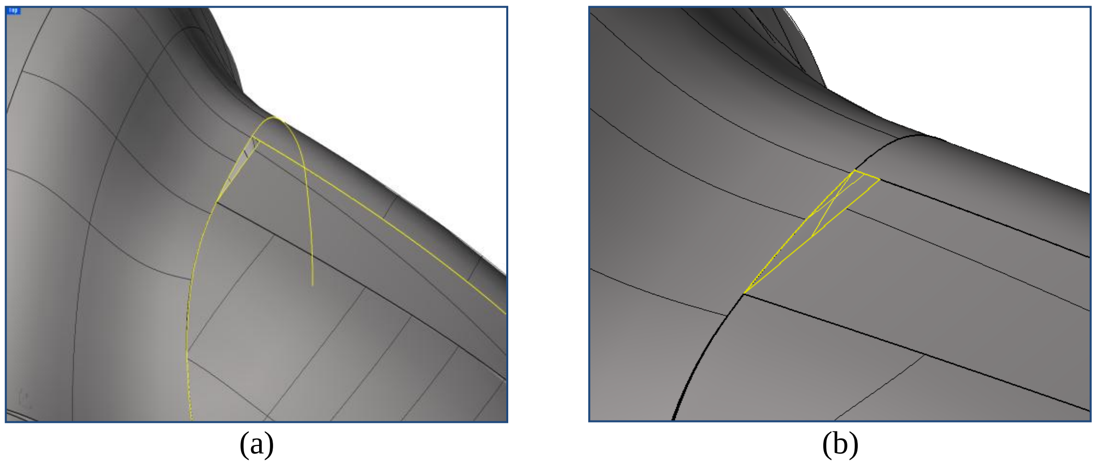


Figure 5.19. Patching the gaps manually

5.4.2 Case Study 2: Flawed Model in KGN Prototype

Figure 5.20 shows the camera with some severe cracks (pointed out with red dashed boxes). The prototype system can reconstruct a flawed design in two ways, a designer can replay a single entity in order to fix it or select to replay (reconstruct) the whole design with a single command, i.e. one button's click. In both cases, the knowledge mining manager will be utilized to provide all required relationships and needed parameters. The knowledge acquisition manager might also bring into play the gathering of new knowledge pieces through *design intent enforcement* that was discussed in Section 4.2.2.

Figure 5.21 and Figure 5.22 show how a designer can fix faulted entities one at a time. The prototype system use *red* color to point out to the user what entities have been reconstructed at a design replay session. Figure 5.21 shows the lens as a picked entity to be fixed while Figure 5.22 shows another choice. The two figures are intended to illustrate the possibility of fixing individual entities without having to redesign the whole model. Figure 5.23 shows the model after fixing only the five damaged surfaces one by one, leaving the rest of the model unchanged.

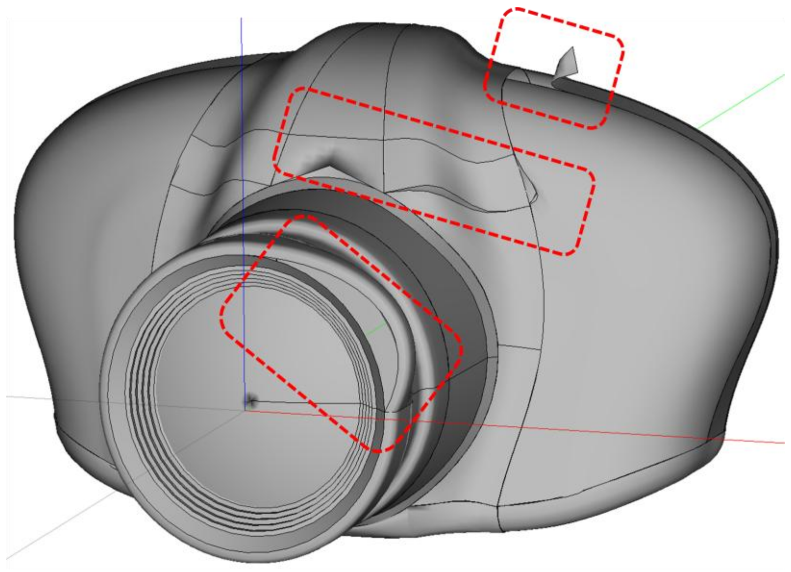


Figure 5.20. Camera on KGN prototype (a): five damaged surfaces

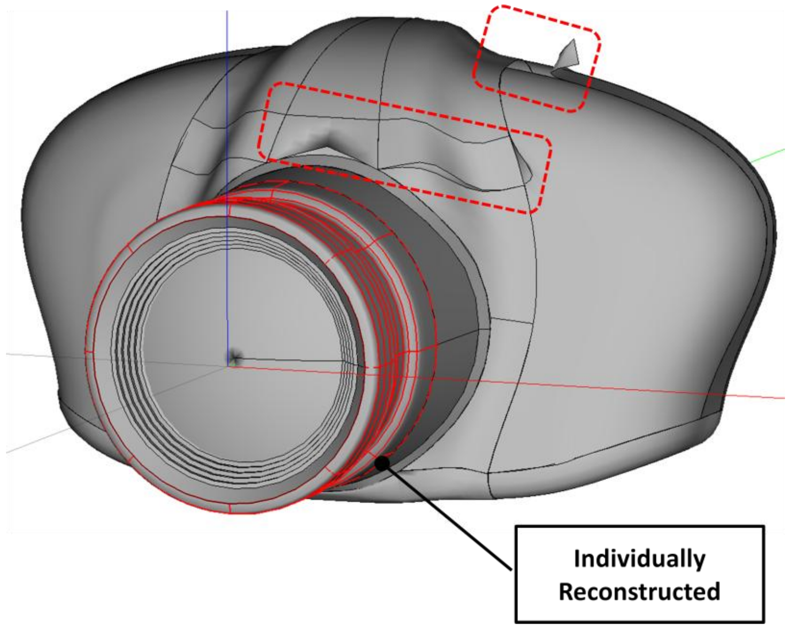


Figure 5.21. Camera on KGN prototype (b): lens individually replayed

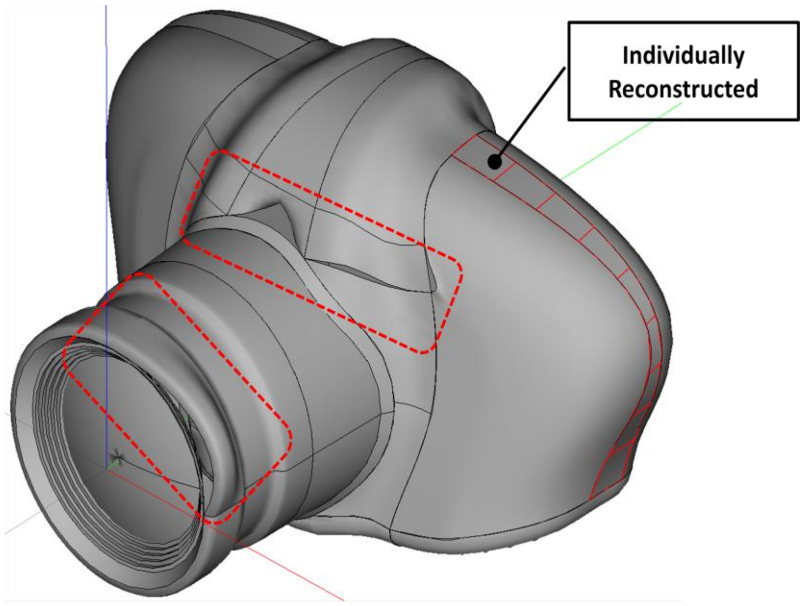


Figure 5.22. Camera on KGN prototype (c): indicated surface individually replayed

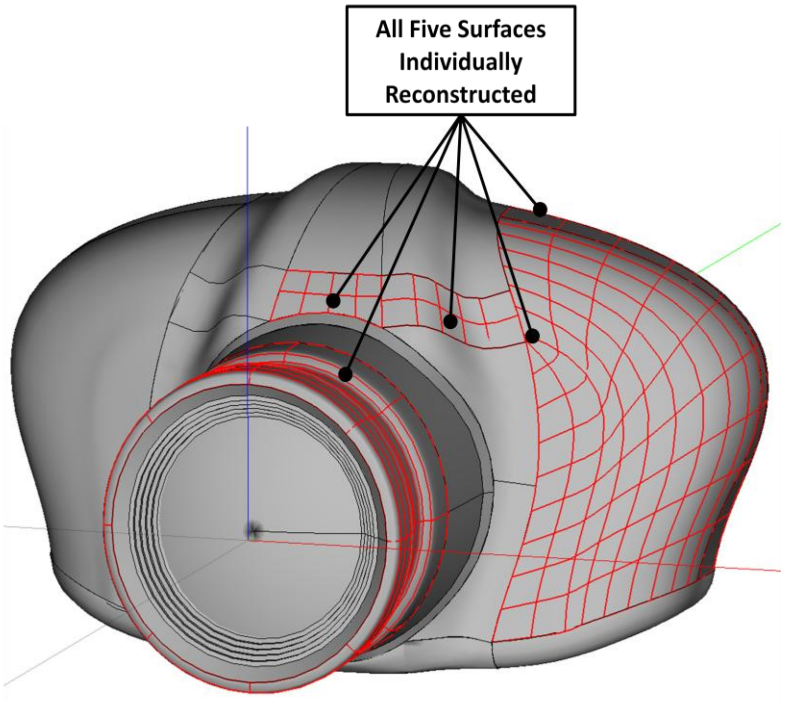


Figure 5.23. Camera on KGN prototype (d): five surfaces individually replayed

Figure 5.24 show an example of using the second way where the design replay reconstructs the whole model.

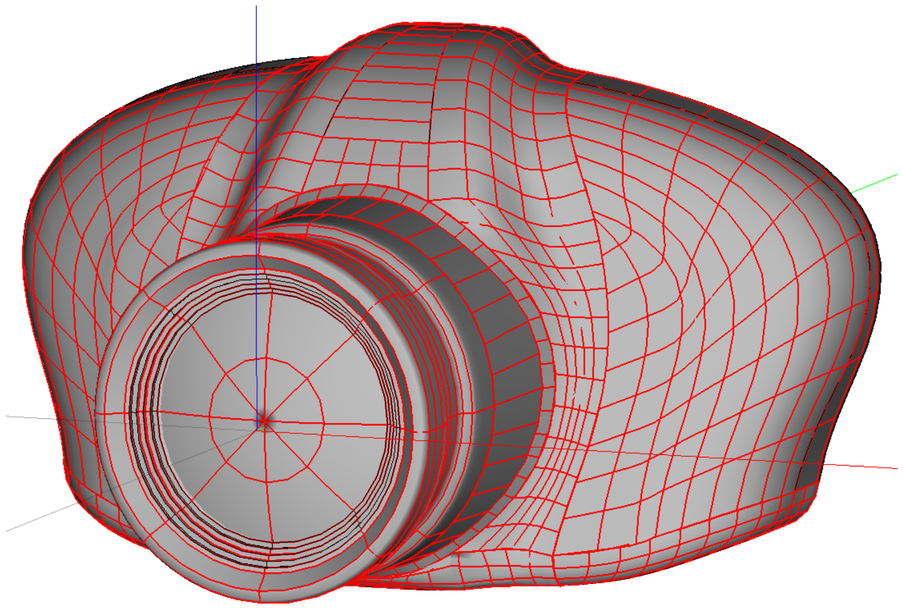


Figure 5.24. Camera on KGN prototype: all model replayed

A third capability can be added easily to incorporate the "*what-if*" scenarios. For example, a designer can inquire "*what* will happen to the model *if* we alter the lens's profile?" The design replay manager can reconstruct the model based on the new introduced modified situation. Depending on the relationship map, a whole design can be fixed with minimal user intervention or the designers might end up doing extensive manual reconstructions of the model. The what-if capability can give a designer an idea beforehand of whether a model can be reconstructed automatically or requires some manually adjusting.

5.4.3 Case Study 3: Different Circle Representations with KGN

Figure 5.25, Figure 5.26, and Figure 5.27 show design replay reconstructing the surfaces and curves based on how each system might represent a circle. The control points are made visible for the camera lens to show the differences after design replay, although other entities that are in relationships with the lens's curves and surfaces are reconstructed as well. In the three figures, we assume that the model is generated on a system that uses a degree 2 circle (Figure 5.25). Figure 5.26 shows a camera model that is migrated to another system that is using a degree 4 circle. The surfaces and curves that are colored in red are altered based on the conversion to degree 4.

The system has gathered enough information in the knowledge base and design base about the model, allowing it to be regenerated with specifications that are required on the second system. In other words, the surfaces and curves are not approximations but are regenerated entities as if they were not migrated but rather modeled on the second system. The designer's intent is preserved and the model is flawlessly replayed. Figure 5.27 shows a second migration to a system that might be using a degree 5 circle. The view was zoomed out so that the control points will be visible for proper illustration.

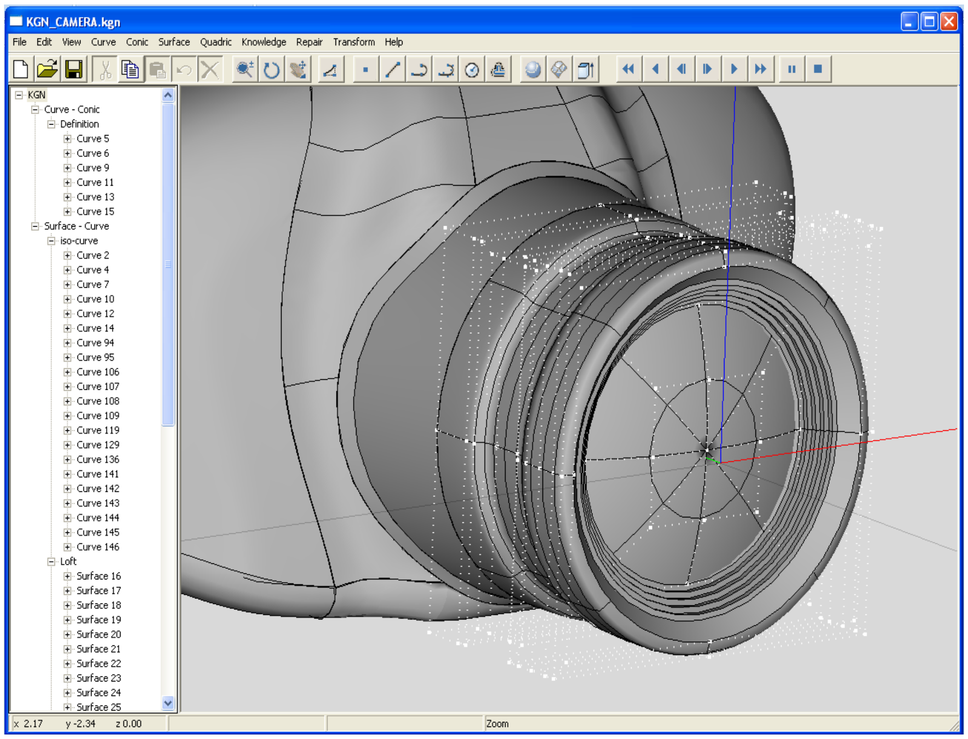


Figure 5.25. Design replay: systems using degree 2 circle (assume original designing system)

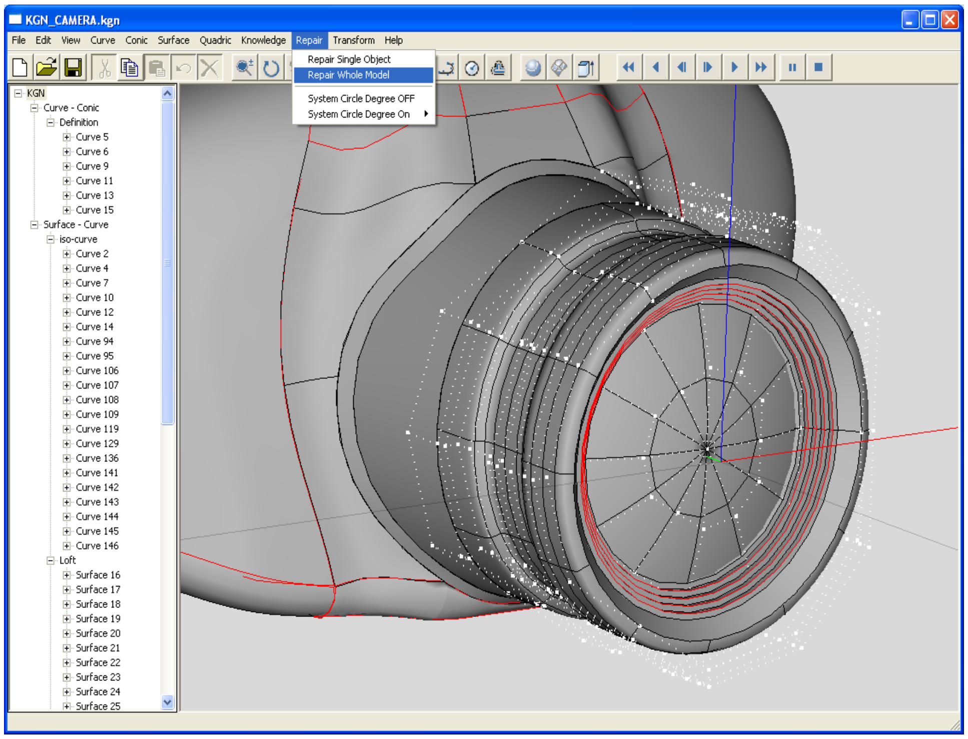


Figure 5.26. Design replay: systems using degree 4 circle (assume migrated model)

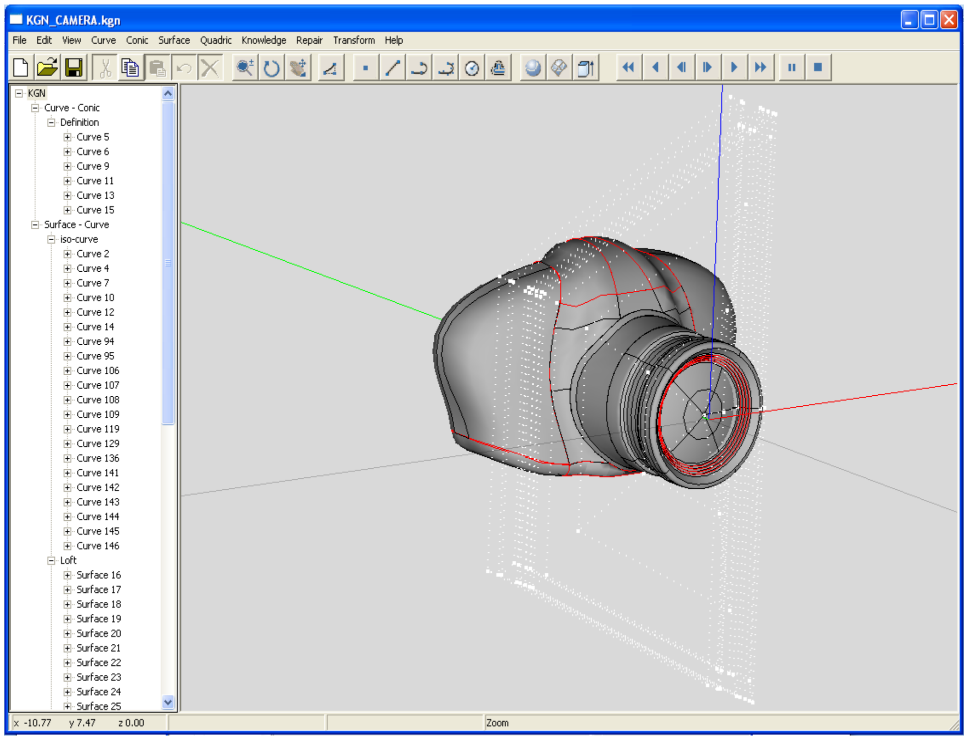


Figure 5.27. Design replay: systems using degree 5 circle (assume migrated model)

Chapter 6 --- Conclusions and Future Work

6.1 Conclusions

In this dissertation we introduced knowledge guided NURBS along with theoretical and practical foundations for supporting design intent capturing, retrieval, and exchange among dissimilar CAD systems. For many years, CAx systems have been exchanging raw numerical data, a process that most often results in countless wasted hours, money, and manpower to correct and repair CAD models that do not migrate as intended. We have shown that preserving the design intent along with endowing NURBS entities with sufficient knowledge are the keys to achieving seamless data exchange, increased robustness, and reliable computations in CAD environments that are based on NURBS.

We defined design intent within NURBS environments as the designing functions, alternatives, decisions, reasons for considering one decision over another, as well as including consideration of the designing history. Chapter 3 showed how this definition of design intent can be structured and incorporated into a knowledge guided NURBS system so that it can be captured and exchanged without user intervention. Chapter 4 details what and how much information needs to be generated and retained, beside design intent, in order to achieve our objectives of seamless model migration as well as more robust and reliable computations. We have established that every NURBS entity should maintain, at a minimum, basic information on identification, classification, definition, representation, and geometry. Moreover, the chapter details how NURBS entities are usually not created in isolation. For example, curves are created from points, and surfaces are created from curves. These relationships are most important in reconstructing models that are not migrated as intended. A relationship map, referred to as knowledge tree, is used to chain NURBS entities together, based on origins and destinations along with specific information for different types of relationships.

After identifying all necessary information, the chapter discussed four knowledge managers that make a system truly a knowledge guided system. A knowledge acquisition manager exhibits the various ways information can enter the system. A knowledge mining manager describes how the acquired knowledge can be browsed to aid with many tasks. It can be used to aid with knowledge traversal for internal processes, such as design replay or it can be used by the users to learn about a model. Export/import managers will interface between two KGN systems. An export manager writes the knowledge and design bases to a file while the import manager will read a KGN file into a running system. The design replay manager makes use of the gathered information and the mining capabilities provided by other managers to regenerate models based on new local requirements, such as tolerance.

All the theoretical aspects discussed in chapter 3 and 4 are supported through a prototype system in Chapter 5. The chapter gives many examples to illustrate the benefits of the system. For model migration we presented a NURBS camera model as a case study. The model was tested on different migration scenarios that resemble what it would go through when exchanged between dissimilar CAD systems. The prototype system and the evaluation results showed that there is no doubt that our methods are implementable and workable as proposed. The prototype KGN system shows that a click of a button can regenerate a migrated model instead of repairing it, avoiding delay and corrective processes that only limit the effective use of such models.

6.2 Future Work

Exploring the potential of knowledge presence should not stop at the stage of merely utilizing it to achieve seamless migration of CAD models and aid their robustness; rather, it should be extended to investigate other possibilities for future usage. An immediate extension to the system is that it can incorporate knowledge farming. An extra manager, namely knowledge farming manager as seen in Figure 6.1, can be added to add additional potency of the knowledge. The process involved with this manager can be very challenging, that is, when compared to the other managers we detailed in Chapter 4. First, the process requires adding two additional difficult capabilities to the knowledge mining manager. It needs to systematically browse for various forms of patterns of behavior and presents pieces of knowledge that may be used in a logical inference. The relationship query in the mining manager has to be augmented with a new capability that determines if sets of objects satisfy a certain relationship.

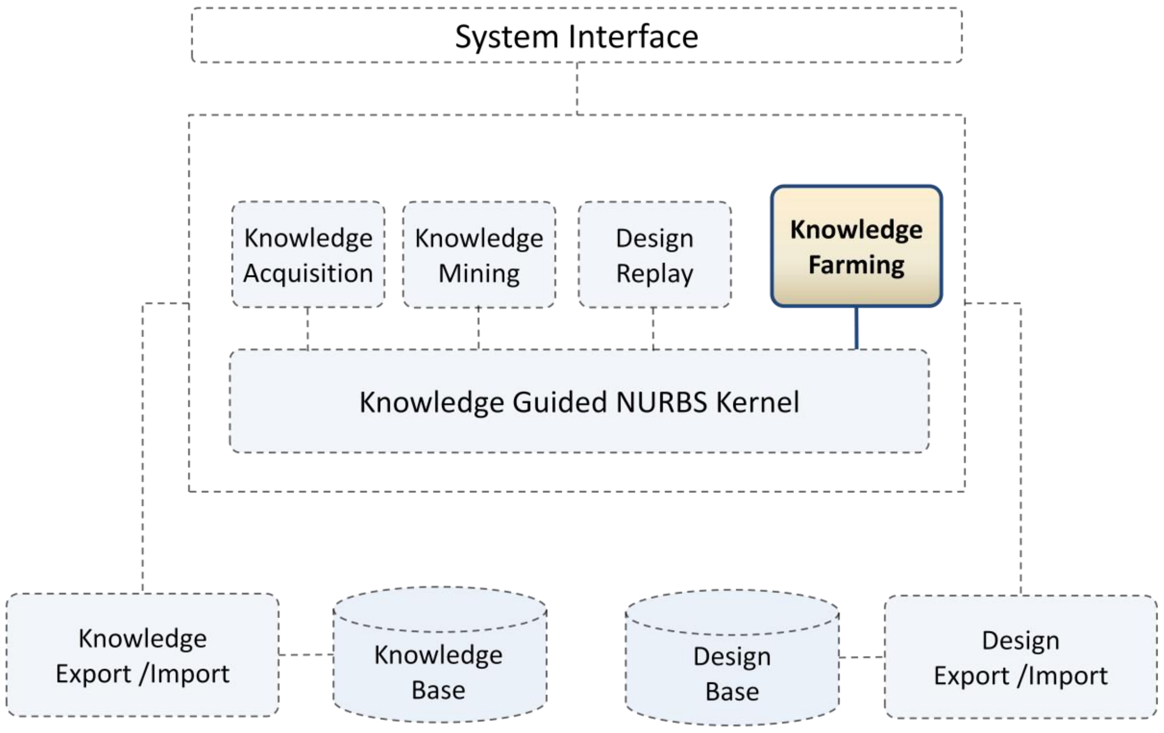


Figure 6.1. Augmenting the system with knowledge farming manager

The knowledge farming manager can then perform two basic operations: knowledge seeding and knowledge harvesting. See Figure 6.2.

6.2.1 Knowledge Seeding

Knowledge farming can use two kinds of seeds: one for the design base and one for the knowledge base. Design base seeds can be made of relationship definitions, e.g. incidence, whereas knowledge base seeds can be existing relationships, e.g., several entities are parallel. These seeds are planted into the data and knowledge bases to see what new knowledge they produce when harvested with the right kind of tools.

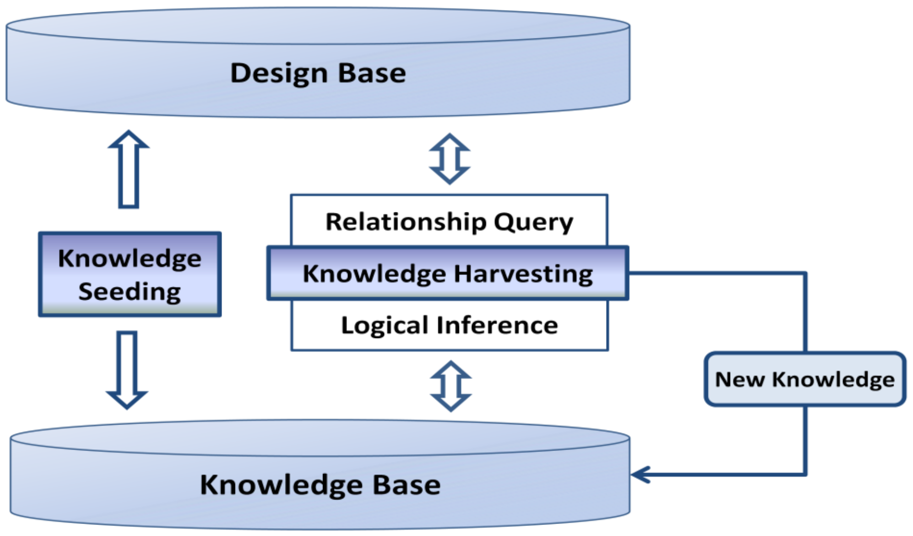


Figure 6.2. Proposed knowledge farming in KGN

6.2.2 Knowledge Harvesting

The design and knowledge bases are in fact full of useful knowledge when harvested with the right kind of tools. KGN can harvest knowledge in two ways: (1) *logical inference* or (2) *relationship query*.

Using logical inference with laws of geometric algebra, new pieces of knowledge can be harvested from existing knowledge. For example, if line 1 and line 2 are perpendicular, and line 2 and line 3 are perpendicular, then line 1 and line 3 should be parallel. Once this information is harvested, it can become most useful when line 1 and line 3 are used as edges in a new design model. Before accepting a new knowledge candidate, the system verifies it against a set of conditions and requirements. If the verification process accepts the new knowledge, it enters the knowledge base, otherwise, it gets discarded.

Relationship query, as a harvesting tool, can prevent many serious numerical failures that are caused by all sorts of relationships unbeknownst to the designer. For example, point sets can be *collinear*, lines can be *parallel*, two surfaces can be *overlapping*, etc. The countless unknown relationships between entities can be harvested using proper definitions as seeds. The core of this type of harvesting is the spatial relationship. Similar to results from the logical inference tool, the new knowledge candidate must be verified before it enters the knowledge base.

Although knowledge farming seems to be a straightforward task, there are numerous theoretical and algorithmic challenges, such as the following:

- Proper spatial data structure for fast relationship query.
- Efficient algorithms for searching for candidates in relationship query—e.g., avoiding a brute-force pair-wise search for overlapping entities.
- Efficient relationship conditions—e.g., what is a stable and efficient method to check if a point cloud is co-planar or co-cylindrical.
- Identifying patterns of relationships—e.g., perpendicular + parallel → perpendicular.

References

1. Tassey, G.: Interoperability Cost Analysis of the U.S. Automotive Supply Chain, Technical report, National Institute of Standards and Technology, March 1999.

2. CAD Doctor: http://www.elysiuminc.com/, Elysium Inc., (Last accessed in May, 2011).

3. CADFix: http://www.transcendata.com/products/cadfix/, International TechneGroup, Inc., (Last accessed in May, 2011).

4. Piegl, L. A.; Rajab, K.; Smarodzinava, V.; Valavanis, K. P.: Fault-tolerant Computing in a Knowledge-guided NURBS Environment, Computer-Aided Design and Applications, 6(6), 2009, 809-823.

5. Piegl, L. A.; Tiller, W.: The NURBS Book, Springer-Verlag, NY, 1997.

6. Bass, L.; Clements, P.; Kazman, R.: Software Architecture in Practice, Addison-Wesley, Boston, 2003.

7. Tyree, J.; Akerman, A.: Architecture decisions: demystifying architecture, IEEE Software 22 (2), 2005, 19–27.

8. Tang, A.; Babar, M.; Gorton, I.; Han, J.: A Survey on Architecture Design Rationale, Journal of Systems and Software, 79(12), December, 2006, 1792-1804.

9. Conklin, E.J.; Yakemovic, K.C.B., A process-oriented approach to design rationale, Human-Computer Interactions, 6(3-4), 1991, 357-391.

10. Grudin, J.: Evaluating Opportunities for Design Capture, in Design Rationale Concepts, Techniques, and Use, T. Moran and J. Carroll, (Eds.), Lawrence Erlbaum Associates, Mahwah, NJ, 1995, 453-470.

11. Product Data Exchange Technologies Success Story Booklet, IPO Winter Meeting, 1997.

12. 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.

13. Gruber, T. R.; Russell, D. M.: Design Knowledge and Design Rationale: Framework for Representing, Capture, and Use. Technical Report, Knowledge Systems Laboratory, Stanford University, California, USA, 1991.

14. Garcia, A.; Howard, H.; Stefik; M.: Active Design Documents: A New Approach for Supporting Documentation in Preliminary Routine Design, Tech. Report 82, Stanford University, Center for Integrated Facility Engineering, Stanford, CA, 1993.

15. Sim, S.; Dufy, A.: A new perspective to design intent and design rationale, In Artificial Intelligence in Design Workshop Notes for Representing and Using Design Rationale, 15-18 August, 1994, 4-12.

16. Ullman, D. G.; Issues Critical to the Development of Design History, Design Rationale, and Design Intent Systems, Proceedings of Design Theory and Methodology - DTM ‘94, DE, 68, 1994, 249-258.

17. Louridas, P.; Loucopoulos, P.: A Generic Model for Reflective Design, ACM Transactions on Software Engineering and Methodology, 9(2), April 2000, 199-237.

18. Kunz, W.; Rittel, H. W. J.: Issues as Elements of Information Systems, Technical Report, Institute of Urban and Regional Development, University of California, 1970.

19. Conklin, J.; Begeman, M.: gIBIS: a hypertext tool for exploratory policy discussion, Proceedings of the ACM Conference on Computer-Supported Cooperative Work, 1988, 140–152.

20. McCall, R. J.: PHI: A conceptual foundation for design hypermedia, Design Studies, 12(1), 1991, 30–41.

21. Potts, C.; Burns, G.: Recording the reasons for design decisions, 10th International Conference on Software Engineering (ICSE '1988), 1988, 418-427.

22. Lee, J.: Extending the Potts and Bruns model for recording design rationale, Proceedings of the 13th International Conference on Software Engineering (ICSE '13), IEEE Computer Society Press, Los Alamitos, CA, 1991, 114-125.

23. Ramesh, B.; Dhar, V.: Supporting systems development by capturing deliberations during requirements engineering, IEEE Trans Software Engineering, 18(6), 1992, 498– 510

24. Potts, C.; Bruns, G.: Recording the Reasons for Design Decisions, Proceedings International Conference on Software Engineering, IEEE CS Press, 1988, 418-427.

25. Lubars M. D.: Representing design dependencies in an issue-based style, IEEE Software, 6(4), 1991, 81–89

26. Banares-Alcantara, R.; King, J. M. P.: Design support systems for process engineering III – design rationale as a requirement for effective support, Computers and Chemical Engineering, 21(3), 1997, 263–276

27. Brice, A.; Johns, B.: Improving process design by improving the design process, QSL-9002A-WP-001, QuantiSci, October 1998.

28. Fischer, G.; McCall, R.; Morch, A.: Design environments for constructive and argumentative design, In Proceedings of the SIGCHI conference on Human factors in computing systems: Wings for the mind (CHI '89), K. Bice and C. Lewis (Eds.), ACM, New York, NY, USA, 1998, 269-275

29. Fischer, G.; McCall, R.: JANUS: Integrating hypertext with a knowledge-based design environment, Proceedings of the second annual ACM conference on Hypertext (HYPERTEXT '89), ACM, New York, NY, USA, 1989, 105–117

30. Shipman III, F. M.; McCall, R. J.: Integrating different perspectives on design rationale: Supporting the emergence of design rationale from design communication, Artificial Intelligent for Engineering Design, Analysis, and Manufacturing, 11(2), 1997, 141–154.

31. Compendium, http://compendium.open.ac.uk/institute/index.htm (Last accessed in May, 2011).

32. MacLean, A.; Young, R.; Belloti, V.; Moran, T.: Questions, options, and criteria: Elements of design space analysis, Human-Computer Interaction, 6(3–4), 1991, 201–250

33. Chen, A.; McGinnis, B.; Ullman, D.; Dietterich, T.: Design History Knowledge Representation and Its Basic Computer Implementation, The 2nd International Conference on Design Theory and Methodology, ASME, Chicago, IL, 1990, 175-185.

34. Thompson, J.; Lu, S.: Design Evolution Management: A Methodology for Representing and Utilizing Design Rationale, The 2nd International Conference on Design Theory and Methodology, ASME, Chicago, IL, 1990, 185-191.

35. de la Garza, J. M.; Ramakrishnan, S.: A tool for designers to record design rationale of a constructed project. 10th International Conference on Applications of Artificial Intelligence in Engineering, Southampton, U.K. Computational Mechanics Publications, 1995, 533–540

36. Chandrasekaran B.; Goel, A. K.; Iwasaki, Y.: Functional representation as design rationale, IEEE Computer, 26(1), January 1993, 48–56.

37. Gruber, T. R.; Russell, D. M.: Derivation and use of design rationale information as expressed by designers. Technical Report, KSL-92-64, Knowledge Systems Laboratory, Stanford University, Stanford, CA, July 1992.

38. Garcia, A. C. B.; Howard, H. C.: Acquiring design knowledge through design decision justification. Artificial Intelligent Engineering, Design, Analysis, and Manufacturing, 6(1) January 1992, 59–71.

39. Garcia, A. C. B.; Vivacqua, A. S.: MultiADD: Multiagent Active Design Documents, Artificial Intelligence & Knowledge Management, Papers from the 1997 AAAI Workshop - Bradley Whitehall, Chair - Providence, RI, USA, July 1997.

40. Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; Stafford, J.: Documenting Software Architectures: Views and Beyond, first ed. Addison Wesley, 2002.

41. Tang, A.; Han, J.: Architecture rationalization: a methodology for architecture verifiability, traceability and completeness, Proceedings of the 12th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS ’05), IEEE, USA, 2005, 135–144.

42. Tyree, J.; Akerman, A.: Architecture decisions: demystifying architecture, IEEE Software 22 (2), 2005, 19–27.

43. Burge, J.: Software engineering using design RATionale, Ph.D. thesis, Worcester Polytechnic Institute, 2005.

44. IEEE, 2000. IEEE Recommended Practice for Architecture Description of Software-Intensive System (IEEE Std 1471-2000).

45. Jeongsam, Y.; Han, S.: Repairing CAD model errors based on the design history, Computer Aided Design, 38(6),2006, 627–640.

46. ISO 10303-203. Industrial automation systems and integration Product data representation and exchange: Application protocol: Configuration controlled design of mechanical parts and assemblies. Geneva (Switzerland): International Organization for Standardization; 1994.

47. Hoffman, C. M.; Robert, J. A.: CAD and the product master model. Computer Aided Design, 30(11), 1998, 905–918.

48. Gu, H.; Chase, T.R.; Cheney, D.C.; Bailey, T.; Johnson, D.: Identifying, correcting, and avoiding errors in computer-aided design models which affect interoperability, Journal of Computing and Information Science in Engineering , 1(2), 2001, 156-166.

49. Rock, S. J.; Wozny, M. J.: Generating topological information from a ’bucket of facets’, presented in Solid Freeform Fabrication Symposium, Austin, Texas USA, August 1992, 251–259.

50. Makela, I.; Dolenc, A.: Some efficient procedures for correcting triangulated models, presented in Solid Freeform Fabrication Symposium, Austin, Texas USA, August 1993, 126–134.

51. Turk, G.; Levoy, M.: Zippered polygon meshes from range images, In Proceedings of ACM SIGGRAPH, 1994, 311–318.

52. Barequet, G.; Sharir, M.: Filling gaps in the boundary of a polyhedron, Computer Aided Geometric Design, 12(2) , 1995, 207–229.

53. Barequet G.: Using geometric hashing to repair CAD objects, IEEE Computer Science Engineering 1997, 22–28.

54. Klein, R.; Liebich, G.; Strasser, W.: Mesh reduction with error control, Visualization 96 Proceedings. Oct-Nov. 1996, 311-318.

55. Kalvin, A. D.; Taylor, R. H.: Superfaces: polygonal mesh simplification with bounded error, Computer Graphics and Applications, IEEE, 16(3), May 1996, 64-77.

56. Steinbrenner, J. P.; Wynman, N. J.; Chawner, J. R.: Procedural CAD model edge tolerance negotiation for surface meshing. Engineering with Computer, 17(3), 2001, 315–325.

57. TransMagic: http://www.transmagic.com, (Last accessed in May, 2011).

58. Hoffmann, C. M.; Juan, R.: Erep: an editable, high-level representation for geometric design and analysis, Working Conference on Geometric and Product Modeling, North Holland, 1993, 129-164.

59. Chen, X.; Hoffmann, C. M.: On editability of feature-based design, Computer-Aided Design, 27(12), December 1995, 905-914.

60. Shih, C-H.; Anderson, W. D.: A design/constraint model to capture design intent, Proceedings of the fourth ACM symposium on Solid modeling and applications, 1997 ACM solid modeling symposium. NewYork: ACM Press, 1997, 255-264.

61. Stiteler, M.: Construction History and Parametrics: Improving affordability through intelligent CAD data exchange: CHAPS program final report, Advanced Technology Institute, USA, 2004.

62. Seo, T-S.; Lee, Y.; Cheon, S-U.; Han, S.; Patil, L.; Dutta, D.: Sharing CAD models based on feature ontology of commands history, International Journal of CAD/CAM, 5(1) , 2005, 38-47.

63. Bettig, B.; Shah, J. J.: Derivation of a standard set of geometric constraints for parametric modeling and data exchange, Computer Aided Design, 33, 2001, 17-33.

64. ISO 10303-42. Industrial automation systems and integration — Product data representation and exchange: Integrated generic resource: Geometric and topological representation. 3rd ed. Geneva (Switzerland): International Organization for Standardization; 2003.

65. Rappoport A.: architecture for universal CAD data exchange. In., Proc. 2003 ACM solid modeling symposium, New York, ACM Press, 2003, 266–269.

66. Kim, J.; Michael J.; Pratt, R. G.; Iyer; Ram D. S.: Standardized data exchange of CAD models with design intent, Computer-Aided Design, 40(7), July 2008, 760-777.

67. ISO 10303-55. Industrial automation systems and integration — Product data representation and exchange: Integrated generic resource: Procedural and hybrid representation. Geneva (Switzerland): International Organization for Standardization; 2005.

68. ISO 10303-108. Industrial automation systems and integration—Product data representation and exchange: Integrated application resource: Parameterization and constraints for explicit geometric product models. Geneva (Switzerland): International Organization for Standardization; 2005.

69. ISO 10303-111. Industrial automation systems and integration—Product data representation and exchange: Integrated application resource: Elements for the procedural modeling of solid shapes. Geneva (Switzerland): International Organization for Standardization; 2007.

70. Pratt, M.J.: Extension of ISO 10303, the STEP standard, for the exchange of procedural shape models, Paper presented in the Proceedings of International Conference on Shape Modeling and Applications 2004, Genoa, Italy, June 7–9, 2004, 317-326.

71. Pratt, M.J.; Anderson, B. D.; Ranger, T.: Towards the standardized exchange of parameterized feature-based CAD models, Computer-Aided Design, 37, 2005, 1251–1265.

72. Anderl, R.; Schmitt, M.: State Of The Art of Interfaces for the Exchange of Product Model Data in Industrial Applications, Institut fur Rechneranwendung in Planung und Konstruktion (RPK, Uni Karlsruhe), December 1989.

73. Mayer, R. J.; Painter, Michael K.; deWitte, Paula S.: IDEF Family of Methods for Concurrent Engineering and Business Re-Engineering Applications, Knowledge Based Systems, Inc., 1993.

74. Goldstein, B. L.; Kemmerer, S. J.; Parks, C.H.: A brief history of early product data exchange standards-NISTIR 6221, 1998.

75. ISO 10303-11: Industrial automation systems and integration - Product data representation and exchange - Description methods: The EXPRESS language reference manual, 1994.

76. ISO TC 184/SC4 N4, Contribution of the United Kingdom, AECMA: Report of Geometry Data Exchange Study Group, June 6, 1984.

77. ISO TC 184/SC4 N3, "Contribution of West Germany, VDAFS,", June 8, 1984.

78. ISO TC 184/SC4 N10: Contribution of France, SET: Standard for Exchange and Transfer, September 25, 1984.

79. Smith, B. K.; Nagel, R. N.; Wellington, J.: IGES-Initial Graphics Exchange Specification, delivered at Autofact, 1981.

80. Bloor, M. S.; Owen, J.: CAD/CAM product-data exchange: the next step, Computer-Aided Design, 23(4), May 1991, 237-243.

81. Wilson, P. R.: A View of PDES, General Electric Company, January 7, 1987.

82. ISO 10303-1:1994 Industrial automation systems and integration Product data representation and exchange - Overview and Fundamental Principles, International Standard, ISO TC184/SC4, 1994.

83. Piegl, L. A.: Knowledge-Guided Computation for Robust CAD, Computer-Aided Design and Applications, 2(5), 2005, 685-695.

84. Piegl, L. A.: Knowledge-Guided NURBS: Principles and Architecture, Computer-Aided Design and Applications, 3(6), 2006, 719-729.

85. Samuelcik M.: Visualization of Trivariate NURBS Volumes, In Journal of Applied Mathematics, 2(1), 2009,143-150.

86. Rogers, D. F.: An Introduction to NURBS: With Historical Perspective, Academic Press, 2001.

87. Piegl, L. A.: On NURBS: A Survey, IEEE Computer Graphics and Applications, 11(1), Jan 01, 1991, 55-71

88. Hicks, J.: Management information systems: A user perspective, MN, USA, West Publishing Company, 1993.

89. Miller, L. L.; Honavar, V.; Barta, T.: Warehousing structured and unstructured data for data mining, The American Society for Information Science, presented at Annual ASIS Meeting, Washington, DC, November 1-6, 1997.

90. 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(4), August 2002, 263-280.

91. Lee, J.: Design Rationale Systems: Understanding the Issues, IEEE Expert, 12(3), 1997, 78-85.

92. Wall, R. A.: Finding and Using Product Information: From Trade Catalogues to Computer Systems, Gower, UK, 1986.

93. Kripac J.: A mechanism for persistently naming topological entities in history-based parametric solid models, Computer Aided Design, 29(2), 1997, 113–122.

94. Wu J.; Zhang T.; Zhang X.; Zbou J.: A face based mechanism for naming, recording and retrieving topological entities. Computer Aided Design, (33) 2001, 687–698.

95 Mun D. H., Han S. H.: An approach to persistent naming and naming mapping based on OSI and IGM for parametric CAD model exchange, Proceedings of the fifth Japan–Korea CAD/CAM workshop on digital engineering workshop (DEWS), February 2005, 24–25.

96. Capoyleas, V., Chen, X., Hoffmann, CM.: Generic naming in generative, constraint-based design, Computer Aided Design, (28) 1996,17–26.

97. Rappoport A.: An architecture for universal CAD data exchange, Proceedings of the solid modeling 03, June 2003, Seattle, Washington, 2003, 266-269.

98. Marcheix, D.; Pierra G.: A survey of the persistent naming problem, Proceedings of the seventh ACM symposium on solid modeling and applications, June 17–21, Saarbrucken, Germany, 2002, 13-22.

99. 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.

100. Piegl, L A.: Knowledge-guided Computation for Robust CAD, Computer-Aided Design and Applications, 2(5), 2005,685-695.