Parametric Modeling

Christoph M. Hoffmann [*]
Department of Computer Sciences
Purdue University

Robert Joan-Arinyo [†]
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Parametric solid modeling is a key technology to define and manipulate solid models through high-level, parameterized steps. These steps can be modified by users and instantiated to specific parameter values and constraint configurations. More than that, the design paradigm supported allows the shape designer to define entire families of shapes, not just specific instances. We review the core techniques of parametric modeling, describe new trends, and sketch a number of open problems that must be resolved to take full advantage of the potential of parametric modeling.

1. Introduction

A parametric solid can be defined as a solid whose actual shape is a function of a given set of parameters and constraints upon them. The objective of *parametric solid modeling*, hereafter also referred to as *parametrics*, is to represent, manipulate, and reason in a computer about the three-dimensional shape of parameterized solid objects.

Prior to the development of parametrics, designers of solid models created a particular shape. Once created, editing and altering the shape was not specifically supported. To accomplish that, the designer had to import the shape and modify it by additional design steps. In contrast, parametric design focuses on the steps creating a shape and parameterizes them. This allows the designer to define an entire class of shapes that later on can be simply instantiated. The added flexibility can be exploited in many ways, and constitutes an important advance in solid modeling and its applications in, e.g., product design.

This overview of parametric solid models covers the two main components, constraints in Section 4 and in features in Section 5. While constraints comprise a well-defined set of tools and techniques, features are a more loosely-knit vocabulary. Feature semantics evolves with applications that seek to conceptualize, in a high-level vocabulary, major design steps and components. The multiplicity of application requirements and agendas makes features a less precisely cast subject that continues to be debated.

We also explore trends we perceive in parametric modeling. Those trends bifurcate into issues especially of interest to academics and issues of immediate interest to industry.

[*]Supported in part by ARO Contract 39136-MA and by NSF Grant CCR 99-02025.
[†]While on leave in the Department of Computer Sciences, Purdue University. Partially supported by the CICYT Grant TIC98-0586-C03-03 and by NSF Grant CCR 99-02025.

There is overlap, of course, and we will work out key aspects in Section 6. Naturally, the trends throw up open problems, to be described more fully in Section 7.

2. Parametric Models

The foundations of solid modeling were laid by the pioneering work of Requicha and Voelcker in the late 1970s for constructive solid geometry (CSG), in which solid shapes are composed from instantiated primitives using set-theoretic operations. Their careful investigation of the topological and geometric foundations of the representation of rigid solids applies to rigid solid models in general, including the boundary representation models that arose around the same time from the work by Braid and Eastman. A survey of the state of the art in 1982 is found in [1].

The framework that originates with the CSG work captures models that have a geometric and a topological structure. The geometric structure relates to the actual shape of the solid surface, and the topology to adjacency and connectedness of the solid interior and its boundary surface. Such models can be characterized as semi-algebraic point sets, and we refer to them as *specific* solid models.

In contrast, a parametric solid model is more than a specific solid because it includes a metastructure from which specific solid models can be derived as instances. Thus, it is more appropriate to think of a parametric solid as a class of specific solid models, and so very different representational schemes have been proposed for them. See for example [2]. The representational proposals divide into procedural representations and mathematical ones. In procedural representations a specific solid shape is constructed by elaborating a sequence of construction steps. In mathematical representations an attempt is made to characterize variational classes of solids by postulating properties such as, e.g., that all members of the class have the same topological genus. To add to the diversity, note that the procedural representations may include nonprocedural substeps. For instance, a cross section to be extruded may be defined by a set of geometric constraints, with more than one solution, and the selection of a particular solution may depend on the constraint solver employed.

The procedural approach is unsatisfactory to some because it does not explicitly characterize a class of solids that can be derived from a common procedural representation. However, the mathematical approach is unsatisfactory as well to some because it has difficulty capturing properties accepted in practice. Those properties are based on a veiled intuition grounded in application requirements or in the particulars of an evaluation mechanism. At this point in time, there is no satisfactory definition of the term *variational class* of solids that has broad acceptance. The field thus moves through territory whose foundations are not fully understood, propelled by technological advances that arise from needs of applications. In view of this incomplete state of knowledge, we offer the following working definition for parametric models:

> A *parametric solid model* is an information structure that permits deriving specific solid models, in the sense of Requicha and Voelcker, using a deterministic algorithm. Moreover, the specific shape derived depends on parameters that are explicit in the information structure and must be valuated for obtaining a specific solid shape.

Our commitment to the procedural school, apparent in this definition, reflects the current state of technology and practice. Note also that we understand a parametric solid model to comprise all specific solid shapes that are derivable from the representation. Some authors have called this a *variational family*, [3].

The bulk of tools incorporated into parametric models and their evaluation are geometric constraints and feature operations. Variant modeling is a precursor to this concept and has closer ties to specifics of the model representation or creation. We explain those concepts in separate sections. In addition, operations such as deformations of solid shapes have been considered, but are found predominantly in experimental solid modeling systems. We do not discuss them further.

3. Variant Modeling

If the objective is to shift from an instance design to a generic one, a simple technique is to prepare a variant design. Using the representation as a symbolic system, parameters can be identified and valuated in different ways to generate variant designs. For example, consider the CSG expression BLO(W, H, D) that evaluates to a block, in standard position, of width W, height H and depth D. Understanding the quantities W, H, and D as parameters, we can instantiate many blocks. This paradigm can be broadened by parameterizing complex expressions built from parametric solid primitives and embedding the expressions into a programming language that permits computing parameter values procedurally. Clearly, we can parameterize the transformation expressions that place the primitives in relation to each other, form conditional branches that may or may not evaluate component shapes based on specific parameter valuations, and abstract a design by encapsulating dependent parameters and exposing independent ones. We call this design approach, first demonstrated by the PADL-2 system, [4], *variant design*.

A slightly different variant design approach [5], implemented by Joan-Arinyo at the Universitat Politècnica de Catalunya in 1993, derives parameters from a symbolic abstraction of design gestures. Ultimately, a program is derived that generates design instances based on pre-defined parameters observed from a visual design gesture. For example, in ducting and pipe design, we may work with a repertoire of standard shapes, to be parameterized in a predefined way and placed sequentially in a way the user defines. Here, the design system can derive the design structure from the user interface gestures and create the variant design.

The variant methodology is especially well suited for applications that deal with those product families that are composed of standard basic shapes with simple parameterization. Moreover, the variation in net shape should be small. See for example [6]. Variant designs survive in libraries of standard parts. For example, there are libraries of fasteners, brackets, and so on, that are essentially derived from a few variant designs and indexed by a catalogue. Some limitations of variant design are explained in, e.g., [7].

Recent developments aim to improve the methodology by providing full support for retrieval of an existing design specification for the purpose of adapting it to design a new but similar product, [8–11]

4. Constraint-Based Modeling

Variant design depends on a fixed script that has been defined manually. Although the script could be very complex, as in the case of embedding a design language into a general programming language, design as programming is less desirable than giving the designer visual tools and deriving from a visual design process a flexible and intuitive parametric design. With the arrival of the geometric constraint solving the technology was at hand to do that. Finally, it was possible to prepare a rough sketch and, by adding specific dimensional and relational constraints, transform it into a precise drawing. Coupled with operations such as extrusions and cuts, it became possible to create designs intuitively and with ease. Furthermore, by valuating the dimensional constraints differently, variants could be obtained automatically by means of a general purpose *constraint solver*.

4.1. Constraints

A constraint specifies a relation on or between entities in a model that must be maintained. The following classes of constraints arise naturally:

- Geometric relationships such as concentricity, perpendicularity, etc., as well as metric dimensional constraints such as distance or angle.
- Equational constraints that express relations between dimensional parameters and/or technological variables such as torque.
- Semantic constraints that define validity conditions on a shape.
- Topological relations between entities in a model, such as incidence or connectivity.

To date, constraint systems of varying competence deal with some or all of these types of constraints. We distinguish between variational and parametric constraint problems.

> A *parametric* geometric constraint problem is one in which a sequence of steps can be identified or derived that solves the problem. In each step, a single geometric element is placed in relation to elements already placed.
>
> In contrast, a *variational* geometric constraint problem includes steps in which several geometric entities must be placed simultaneously in relation to each other.

For planar geometric constraint problems competent and efficient solvers are readily available. For spatial constraint problems the technology is not nearly as to mature, likely owing to a greater intrinsic difficulty of spatial problems. This difference is manifest in design systems available today.

4.2. Modeling with Constraints

When defining a model initially, sketches are prepared and annotated with constraints. *Sketching* can be done with a mouse or more specialized devices. *Constraining* the sketch often is through menu dialogues. The sketches are then converted into precise shapes, by solving the constraints. Finally, the solid shape is defined from the sketches, using operations such cuts or protrusions generated from revolving or extruding cross sections.

Most systems allow interleaving sketching and constraining. Figure 1 left shows the sketch of a constraint problem input to the constraint solver. Here the arc should be tangent to the adjacent segments, and the two other segments should be perpendicular. Output of the constraint solver is shown on Figure 1 right.

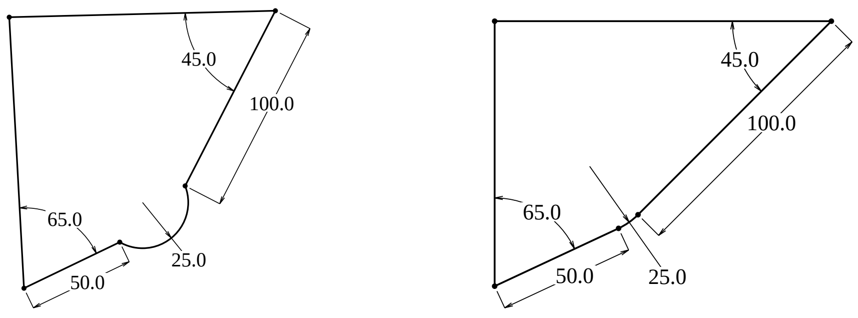

Figure 1. Sketch of a constraint problem and a solution generated by the solver.

When a sketch is *solved*, the underlying constraint solver expects in some cases a well-constrained problem, that is, one in which no additional constraints can be added without creating redundancies. Overconstrained sketches are usually rejected. In the case of underconstrained sketches the system will infer additional constraints that, when added, make the problem well-constrained.

When an already defined model is edited, the user changes some parameters or constraint values in the simplest case. The system constructs the new instance automatically solving for the changed values. More complex editing may change the parametric model itself, adding or deleting features, or changing the definition of some of them.

4.3. Solving Geometric and Equational Constraints

Some constraint problems permit a sequential solution, in which the geometric elements are placed one-by-one, in accordance with the constraints. Such problems correspond to triangular, nonlinear equation systems. For planar cross section definitions, only a few modelers restrict to sequential problems. Most systems allow variational constraint problems in 2D, and therefore free the designer from the burden of having to understand whether the constraint schema is a constructive, sequential one. Note that sequential problems may also entail multiple solutions. For instance, assume that we are given two fixed circles and seek a common tangent of them. Then we would have to select one of up to four possible tangents.

A variational constraint problem is equivalent to a nonlinear system of equations. Moreover, a mathematically well-constrained problem will have more than one solution in general. There are general algorithms to convert a nontriangular system of nonlinear equations into a triangular system, [12]. Therefore, the distinction between parametric and variational constraint solving is artificial in theory. However, triangularization of systems of nonlinear equations is not tractable even for problems with a relatively small number of variables and equations.

Ideally, differentiating between the possible solutions and selecting the appropriate one would be accomplished by adding other, nongeometric constraints. Unfortunately, to-date no convincing approach to this problem has been discovered, and solvers rely on proprietary, sometimes rather complicated heuristics to select a solution that hopefully matches the intent of the designer. Simple “metaconstraints” can be entertained that might assist solution selection. For example, when designing a cross section, we might require that the bounding contour is not self-intersecting. Unfortunately, such simple rules cannot be efficiently implemented; [13].

Owing to the difficulty of variational constraint solving in three-space, spatial constraint problems are typically sequential. This imposes limitations on the designer that manifest themselves very clearly when designing mechanical assemblies.

Many approaches to solve geometric constraint problems have been reported in the literature. They can be categorized roughly as equational, constructive and degree of freedom analysis. We give a brief sketch of these techniques. For a thorough review see, e.g., reference [13].

4.3.1. Equational Methods

An equational solver translates the geometric constraint problem into a system of algebraic equations which are then solved using a collection of techniques.

The Numerical Approach

A numerical solver applies iterative techniques to solve the equation system. Such solvers can be quite general, and many constraint solvers switch to numerical methods as an alternative to another method. However, most numerical methods have trouble handling overconstrained and underconstrained problems. Only overconstrained problems which consistently define an object may be solved using this techniques.

Early systems such as those reported in [14, 15] used *relaxation* methods to solve the system of equations. Relaxation methods work by perturbing the values assigned to variables in such a way that the total error is minimized. The main problem is that convergence is slow.

A widely used numerical technique is the Newton-Raphson iterative method. Its main drawback is that the iteration requires a good initial value. If, as is usual, the initial values are taken from a rough sketch defined by the user, the sketch must almost satisfy all the constraints. Nonlinear systems have an exponential number of solutions and the Newton-Raphson iteration will find only the solution closest to the initial guess. Since the approach is unable to find alternative solutions, it is inappropriate when the initial sketch leads the solver to a solution which does not fit the users needs. Solvers in [16–19] are based on Newton iteration.

Hel-Or *et al.*, [20], report on a paradigm called *relaxed parametric design*, to guide the solver in the selection of a solution amongst a set of candidates, which satisfy all the constraints. The designer may provide soft constraints weighted by a user-defined certainty. Soft constraints are represented by a measurement and a tolerance and do not have to be satisfied exactly. A probabilistic constraint schema is used and an estimate of the model is computed using the Kalman filter technique developed in control theory.

Kin *et al*, [21], solve geometric constraint problems using an extended Boltzmann machine, an artificial neural network. An energy function associated with the constraint network is defined to include terms of higher order than quadratic with respect to the binary states of the units that constitute the network. The extended Boltzman machine minimizes the polynomial energy function.

Recently developed methods in numerical continuation known as *homotopy* methods, are able to compute all solutions to polynomial systems [22–24]. The solution of a system of nonlinear equations by numerical continuation is motivated by the idea that small changes in the parameters of the system usually produce small changes in the solutions.

The Symbolic Approach
The symbolic approach translates the system of equations into another set of polynomials with the same roots. The resulting system is solved with symbolic algebraic methods, such as Buchberger's Gröbner Bases, [25], or the Wu-Ritt method, [26]. Both methods can solve general nonlinear systems of algebraic equations, but they require exponential running times. The transformed system is triangular, so the problem of simultaneously solving n polynomials with n variables is reduced to repeated univariate polynomial solving. The approach finds in principle all solutions. Solvers in references [27] and [28], use Buchberger's algorithm.

Propagation Methods
The method generates an undirected graph whose nodes are the variables and constants in the system of equations and whose edges represent equations relating these variables and constants. The propagation method attempts to direct the graph edges so that every equation can be solved incrementally. The technique thus tries to discover a sequential strategy for solving the constraint system.

Various propagation techniques have been reported in the literature, [29–31], but none of them guarantees a solution when one exists, and most fail when a cyclic dependence is found. Propagation is sometimes used in conjunction with a numerical technique. For example, in [14, 15], when the propagation of degrees of freedom fails, a relaxation method is used. For a review of these methods, see [32].

4.3.2. Constructive Methods

Constraint solvers based on a constructive approach take advantage of the fact that many geometric constraint problems can be seen as engineering drawings which are usually solvable by ruler, compass and protractor. The two main approaches commonly classified as constructive are the rule-based and the graph-based approach.

Rule-Based Approach
In a rule-based approach, constraints are expressed by predicates, and geometric construction operations by functional expressions. These constructive solvers compute a symbolic solution of the constraint problem using a rewriting system to find a sequence of geometric operations that build the object which satisfies all the constraints. If the constraints consistently describe the position and orientation of the object, then the constraint problem can be solved.

The earliest rule-constructive solvers did not consider the problem of nonunique solutions, [33, 34]. However, later approaches, [35–40], compute all possible solutions when constraint problems are well-defined.

Hoffmann and Joan-Arinyo, in [41], combine a rule-constructive solver with an equational solver based on graphs. When no more constructive rules apply, a bigraph is used to analyze the structure of the system of equations. Using matching theory techniques, a set of equations is isolated and solved in a general purpose equational solver. Joan-Arinyo and Soto generalized this approach in [42].

Graph-Based Approach

Graph-constructive solvers derive a sequence of construction steps using graph analysis techniques. DCM, a commercial constraint solver described in [43], uses this method: a graph is broken up into a set of subgraphs such that an algebraic solution for each class of the resulting subgraphs exists. Then, the subgraphs are positioned applying rigid body transformations to all geometric elements that belong to the subgraph.

Fudos and Hoffmann in [44] report on a graph-constructive approach to solve systems of geometric constraints capable of efficiently handling well-constrained, overconstrained, and underconstrained configurations.

Although this approach is faster and more methodical than the rule-constructive approach, the graph analysis algorithm needs to be modified when new types of constraints have to be considered.

4.4. Degrees of Freedom Analysis

In this approach, the notion of degrees of freedom is associated to primitive geometric objects and constraints. Any geometric object (point, line, circle, etc.) has a number of degrees of freedom in its embedding space. Constraints (coincidence, distance, angle, etc.) reduce the degrees of freedom of an object.

Kramer, [45], solves geometric constraint problems by symbolically reasoning about the geometric entities themselves using a technique called *degrees of freedom analysis*. In this approach, the configuration variables of a geometric object are defined as the minimum number of real-valued parameters required to specify the object in space unambiguously. The configuration variables parameterize an object's translational, rotational and dimensional degrees of freedom with one variable required for each degree of freedom. A constraint solver for three dimensional constraints is described in [45], in which constraints on rigid bodies are satisfied incrementally by a sequence of rigid-body motions. A plan of measurements and actions is devised to satisfy each constraint incrementally, thus monotonically decreasing the system's remaining degrees of freedom. This plan is used to solve, in a maximally decoupled form, the equations resulting from an algebraic representation of the problem. Kramer's solver is restricted to kinematic loops of length 4. For more complex interdependence his solver has to resort to numerical methods.

Using a graph-based technique, Hsu derives a plan of evaluation by examining and updating the degrees of freedom and dependencies between objects, [46]. First the method generates a connected subgraph and a dependency graph. Then the dependency graph is solved by a hybrid solver which generates the solution in the form of direct constructions and iterative constructions.

A flow-based method for decomposing the graph of a geometric constraint problem is described by Hoffmann *et al.* in [47]. The method fully generalizes the degree-of-freedom approach. The method iterates to obtain a decomposition of the system of equations underlying the constraint graph, into small subsystems.

5. Feature-Based Modeling

Features have become an integral part of parametric modeling. Features provide a higher level vocabulary for specifying operations to create shapes by providing parametrized geometry, attributes and geometric constraints. Moreover, parameters, attributes and constraints can be encapsulated.

In a good design, features capture explicit engineering attributes and relationships for product definition and provide essential information for various design tasks and performance analyses. In manufacturing, features can be linked to manufacturing knowledge, thereby facilitating manufacturing and process planning. Features also provide a framework for organizing design and manufacturing information in a data repository for reuse in new product design, [48].

5.1. Features and the Feature Model

Features have been defined in a number of different ways in the literature. A good definition that captures the current trends in features development is due to Shah, [49], who defines a feature as a generic shape with which engineers associate certain properties or attributes and knowledge useful in reasoning about a product.

In order to be useful, a feature should embody at least three different concepts: Generic shape, behavior, and engineering significance, [48]. The generic shape is parametrically defined as a boundary representation, a CSG tree or another geometric representation, including procedural representations.

Behavior and engineering significance are defined by means of attributes and domain-specific rules. Attributes can be classified into several groups. Geometric attributes refer to the feature's shape and examples are dimension attributes, default and feasible values for parameters, tolerances, location parameters and so on. Technological attributes give information useful to downstream applications, such as material properties, heat treatments, tool and fixture information, etc. Some attributes can take the form of rules to define the behavior of the feature. The rules state what conditions should or must be imposed on a feature within a given process in order to perform a particular activity. Attachment validation and symbolic or skeletal representation derivations are examples of such rules.

A feature model is a data structure that represents a part or an assembly in terms of its constituent features. Feature models are created by organizing the constituent features in a suitable structure that expresses the required relationships between the various features.

There is a continuing debate on what a precise definition of feature should be. In part, the debate is fueled by conflicting part and assembly conceptualizations arising from different categories of design, analysis, and manufacture. For example, the burner casing of a jet engine may have a set of features relevant to thermal analysis, yet a different set of features may be relevant to structural analysis. A third set of features may be important to the casting process by which the casing is manufactured, and a fourth set of features may be important to analyzing tolerances in the context of the assembly with other engine parts. These different categories can be considered views, and each view of the product will focus on its particular set of features.

This divergence of feature sets, on the same product, would be less onerous were it not that the design process of the geometric shape forces the designer to distinguish a particular set of features for the purpose of geometry creation. This set of design features often is not useful to the manufacturing engineer or the performance analyst. Owing to limited technology, switching between different feature sets during product design and manufacture is difficult or not supported by many CAD systems, leading to privileged views and continuing interest in developing techniques to switch effectively between different views without losing the flexibility of parametric design.

5.2. A Brief Feature Taxonomy

We distinguish between geometric features and nongeometric features. Geometric features are closely related to the geometry of a model and can be further differentiated into

- Form features: portions of nominal geometry defining a feature's shape.
- Tolerance features: Deviation from nominal definitions of shape, size or location.
- Assembly features: Grouping of various features to define assembly relations such as mating conditions, position and orientation, kinematic relations, etc.

Viewed from an application perspective, geometric features can also be classified into design features, manufacturing features, process planning features, etc.

Nongeometric features are generally related to technological information. Examples of this type of features are:

- Functional features: Sets of features related to a specific function like design intent, parameters related to function and performance, etc.
- Material features: Material composition, treatment, surface finish, etc.

5.3. Feature Model Construction

Three basic techniques for constructing parametric feature models have been identified, [49, 48]: Interactive feature definition, automatic feature recognition and design by features.

5.3.1. Interactive Feature Definition

In the interactive feature definition technique the user interacts with a model that has already been defined, possibly using another design methodology. Interactively, the user selects on this displayed model entities to be grouped into a feature. A feature so defined can then be annotated with attributes such as surface finish and tolerances. In some cases, the feature can be parameterized by defining parameters and constraints on the entities.

Groupings and annotations are easily implemented. Moreover, the entire model need not be featurized, only those features need to be defined that are of particular use in the application the user has in mind. Feature validation is usually the task of the user.

If many features have to be defined, this process is error-prone and tedious. Moreover, the persistence of annotation is usually not guaranteed, although technology exists to make persistent annotations of parametric models. The definitional task can be assisted by feature recognition coupled with a feature library that contains generic feature definitions and can propose completions of a partially defined feature.

A more complicated issue arises when the original model does not have any parametric information and the user adds parameters to the features defined on the model. Re-evaluation of the model with changed parameter values is needed in this case, and there are commercial modelers that can accomplish this task within certain limits. The problem increases in difficulty if the original model is parametric, and we wish to preserve the original parameterization structure in addition to editing the model from user-defined features and their parameters. This problem is technically related to reconciling different views when editing, and requiring that different views can edit from within their perspective.

5.3.2. Automatic Feature Recognition

In automatic feature recognition, a previously defined geometric model is processed algorithmically to detect features defined in terms of rules or subgraphs or other kind of generic feature knowledge. This approach has received considerable attention in the literature. However, several research challenges still need to be addressed and solved in order to ground the approach in a robust and solid framework. See [50] for a review of automatic feature recognition systems.

A major difficulty common to all known approaches to automated feature recognition is the recognition of intersecting features. When several features intersect, their topology can change dramatically. Since most of the proposed feature recognition techniques seek to identify among the model's edges and faces groups that exhibit a specific topological and geometric character, the fragmentation entailed by intersecting features can foil the recognition algorithm. Moreover, it can lead to interpretation ambiguities that must be resolved by adopting certain heuristics. Work has been reported that attempts to recognize features not only individually but also feature interrelations like containment and intersection; see, e.g., [51]. Unfortunately, the computational complexity explodes even for very simple objects.

Most of the existing feature recognizers work in batch mode. They accept as input a completed design and produce as output a feature model. For the reasons explained before, the feature model built represents one of several possible interpretations. Even a small change in the initial model can force feature recognizers to discard the previous work and start an expensive geometric reasoning process from scratch. Since batch operation is undesirable in interactive environments, some efforts have been devoted to incremental feature recognition, [52].

Another drawback arises from the fact that automated feature recognition techniques process final functional forms. Therefore, it is not adequate for certain types of downstream applications. For example, in process planning in the automotive industry, the intermediate geometry must also be identified, [53]. Since intermediate shapes, sometimes also called *in-process* shapes, cannot reliably be reconstructed from the final net shape, important information has been lost that reduces the applicability of automated feature recognition.

5.3.3. Design by Features

Design by features is the most widely used technique in feature-based modeling. Here the model is built directly by the user by instancing generic feature definitions, which are used as templates, and locating them in space or attaching them on existing features. The features can be instantiated from a library of either standard generic features or from application-oriented, user-defined features, [54].

Feature-based systems usually provide standard generic feature libraries as collections of predefined features such as slots, pockets, and holes, and operations for defining sketched features where geometry is created by sweeping a planar cross section or lofting between two or more planar cross sections. Standard operations provided by systems allow the user to create complex shape designs that could not be built using standard features only.

A plausible design process of a pocket with a bridge across the bottom is illustrated in Figure 2. The designer first creates a pocket of maximum depth, and then adds the rib as a protrusion with fillets at the edges.

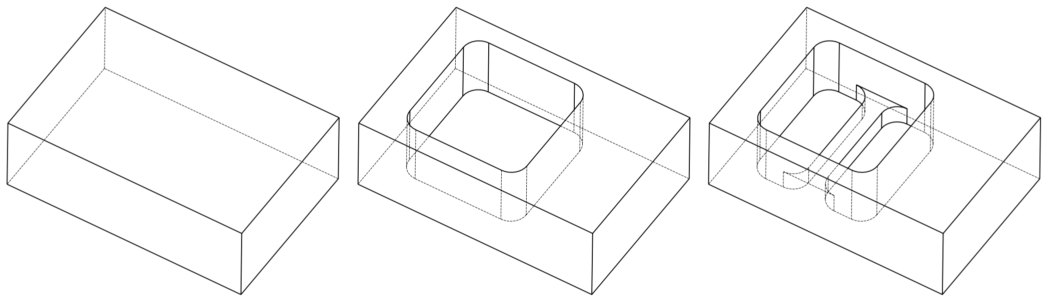

Figure 2. Design sequence: make a cut, then add a protrusion.

The design by features technique provides a set of operations to edit the model. Broadly speaking, these operations are: Inserting or deleting an entire feature, changing feature attributes, modifying dimension values that define the feature or place it in the model, changing the set of constraints associated with the feature, and changing the feature shape definition. Procedural steps common to these editing operations are given in [55].

5.4. Feature Representation

The shape of a feature may be expressed in terms of construction steps that produce the geometry corresponding to the feature or in terms of an enumeration of geometric and topological entities and relations along with dimension parameters, [56]. The first approach is a procedural approach while the second is a declarative approach.

5.4.1. Procedural Representation

In the procedural approach, generic features are predefined in terms of a collection of procedures which may include methods for managing a feature as a whole, like instancing, copying and deleting a feature, and methods for specific operations on a given feature like generating the geometry, deriving values for parameters, and validating feature operations. The procedures may be encoded in either a general or special-purpose programming language.

In procedural feature representation, feature definitions are explicitly expressed in terms of a computation. So, a feature is always instantiated from a given function with a given set of parameters whose values the user sets. If the given parameter values are changed, the entire procedure must be run again to compute the new instance.

5.4.2. Declarative Representation

The declarative approach, features and their properties are first described in a declaratively; i.e, the definition declares what the feature is rather than how it is built. Then, a general algorithm constructs the actual detailed feature model. One of the main tools of the declarative approach is the use of constraints to define the particulars of features.

In [57], the authors proposed the Erep declarative framework that achieves a clean separation between definition and construction. In this framework, geometry and properties are represented in a neutral form while the actual construction is performed by algorithms committed to a clear, underlying semantics. The framework naturally provides tools for representing constraints and attributes of features.

In declarative feature representation, constraints play a central role because they provide a natural way to describe spatial relations between geometric entities within a feature and between features. Furthermore, constraints provide a mechanism to define relationships between geometric and technological parameters. Therefore, all the constraint solving machinery can be applied effectively.

5.5. Features and Constraints

Increasingly, solid modeling systems use both features and constraints in the design interface. Typically, feature-based design systems deploy a design paradigm in which the designer may use a set of predefined features and operations for defining *sketched features*. The geometry of sketched features is created by sweeping a planar cross section or lofting between two or more planar cross sections.

Cross sections are defined as sketches with declarative constraints. A variational constraint solver instantiates the cross section based on the sketch and the constraints, and places it with respect to the geometry constructed so far from prior features. Parameters and constraints then define the sweep extent.

6. Trends

Trends in parametric solid modeling are fostered mainly by two different requirements: integration with product data management and downstream applications, and support for concurrent distributed design environments.

To fulfill these requirements, solid modeling should provide an efficient and direct communication between engineering processes which, in turn, requires advanced modeling tools and methods to provide users with facilities to capture geometry sufficiently enriched with engineering semantics. These requirements affect parametrics, and therefore features and constraints, in a number of ways.

6.1. Feature Libraries

Devising a universal set of features would improve the interoperability between different applications in an integrated environment. But seeking to devise such a set of features would lead to an unmanageable number of features. For this reason, research has begun to investigate generic mechanisms to give the user the option of building custom feature libraries that might satisfy specific application needs. This approach has been advocated in the following work.

Shah *et al.* reported on the ASU shell in [58], a testbed for rapid prototyping of feature based applications. The library of generic features is organized in the form of a list of properties. Each feature has a feature type identifier, a name, a list of generic, compatible features, and a solid representation. A CSG tree provide the solid representation for form features. Recent developments of the test-bed are reported in [59].

Laakko and Mäntylä, [60], describe an extension of the programming language Common Lisp to represent features procedurally. The feature models are organized as a structure of Lisp frames. Such frames model two different types of features: *features classes* which are templates that store generic information, and *feature instances* that store specific information belonging to individual features. Feature classes are organized by a taxonomy and use the inheritance mechanism of Common Lisp. A feature model is a list of feature instances.

De Kraker *et al.* in [61] and Dohmen *et al.* in [62] report on the specification of a feature language developed at Delft University of Technology. Features are specified using predefined types in the object-oriented, imperative programming language LOOKS. Therefore, the feature library is a library of LOOKS procedures and defining a new feature means to write new code for it.

Hoffmann and Joan-Arinyo in [57] proposed the Erep framework for expressing form features and constraints. The representation of a part design is a generative feature description. It is textual and neutral, in the sense that it is not committed to a particular core solid modeling system. In [56], the Erep was extended with a procedural mechanism for generating and deploying user-defined features through standard graphic operations provided by the underlying modeling system.

6.2. Multiple Views

In an integrated, concurrent and distributed design environment, the data in the product model is contributed by different applications. Eeach application has its own *view* of the data. For example, from a machining point of view, the feature structure shown in Figure 3 is one of many possible interpretation for the design view in Figure 2. Therefore, a persistent association between data contributed by each application must be establish and maintained. Creating and maintaining such a persistent association is a key problem.

It is natural to argue that in concurrent engineering, a modification required by a specific application should be made in the view of that application. Moreover, all modifications introduced in one view should be propagated automatically to all other views. Some work based on this assumption has approached the problem in a setting far too general. In the absence of specifics, such work has not proposed credible mechanisms to address the view consistency problem. The work by Bronsvoort *et al.* is a notable proposal in that respect, [63, 64, 61, 62]. The work addresses formally the problem of different form feature views editing a common net shape. Briefly, the net shape is modeled by a cell complex where the cells are subdivided as necessary such that every feature of an application view is composed of entire cells. That property permits to edit shape mechanically from any feature view and achieves consistency across all views.

A different approach is developed by Hoffmann and Joan-Arinyo in [65, 66]. The ap-

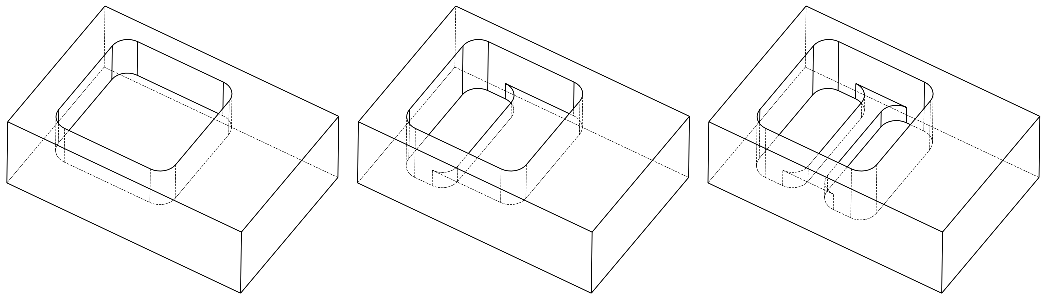

Figure 3. Machining view sequence: make a shallow pocket, then deepen the pockets to the left and right of the bridge.

proach is based on the concept of the *master model*, an object-oriented repository that provides essential mechanisms for maintaining the integrity and consistency of the deposited information structures. The clients of the master model are the modeling system and domain-specific applications. An analysis of the needs of different views allows a simpler solution of consistent edits that uses persistent information associations.

6.3. Semantic Features

The concept of semantic feature was developed in response to the need of capturing engineering information that connect form with functional intent. Such features also support integration with downstream applications. Semantic features provide tools for using features consistently.

Procedural semantics for attaching features to a model in generative constrained-based modeling systems have been defined by Chen and Hoffmann in [67]. The work considers generated features that are based on a planar profile swept into a three dimensional shape. This work was extended in [68] where a set of techniques needed for editing generative feature-based models is discussed, including persistent naming techniques. Persistent naming is needed to re-evaluate edited features.

Effectively maintaining the validity of a feature, attached to a model, entails developing mechanisms to detect invalid situations, mechanisms to properly report improper feature use, and mechanisms to provide the user with good choices to recover validity. In the work reported in [69–71] and [62], the validity of a feature is specified as a set of geometric, topological and functional constraints. Whenever a modeling operation is completed, a validity check is performed. If a violation of some validity criterion is detected, the operation branches into a reaction loop where a validity recovery process is triggered. A valid state is achieved again, either by reestablishing the previous valid state or through a dialog with the user.

6.4. Persistent Naming

As mentioned before, applications of solid modeling in manufacturing and other application arenas seek to enrich the semantic content of a shape model. Geometric constraints and parameters can be considered semantic information pertaining to shape, sometimes in an implicit form. Other information, such as annotations or feature definitions relevant to different views, is needed as well. When a parametric model is edited, it is crucial to preserve and re-attach semantic information. The technology to do this has been called *persistent naming*.

The problem of persistent naming arises as follows. Consider a particular shape instance on which we annotate a selected face with some material property information. This is done easily for the instance data structure, usually a boundary representation. However, the parametric model itself need not have any faces, so it is not clear how to record the information such that it persists under a change of parameters or constraints. More than that, different model instances of the same parametric model may have zero, one, or several faces that might correspond to the original annotated face. This problem has been considered in [68, 72, 3] and [73], and several approaches have been proposed in the literature.

Ultimately, a solution to the persistent naming problem is a semantic interpretation of the instantiation of a parametric model. There remain basic questions on what constitutes a good semantics that may never be resolved by the community. Therefore, the persistent naming solutions proposed in the literature can be considered to be specific proposals for such a semantics. Because of this larger context, work on persistent naming should be considered evolutionary.

We postulate that a successful semantics for parametric shape design has to allow construction algorithms that exhibit two key properties:

1. The parametric instantiation algorithm has to be *continuous*.

2. The parametric instantiation algorithm has to be *persistent*.

Continuity requires that a geometric configuration, derived by instantiating for a set of parameter values $(p_1, ..., p_k)$ should change by a small amount when altering the values of the parameters by a small amount. Persistence means that after changing the parameter values and reinstantiating the configuration, a return to the original parameter values should result in the original configuration being recovered. Clearly, those are minimal requirements any user of would expect from parametric design.

Geometric software is often hard-pressed to make good on continuity which is easily violated when the configuration passes a degenerate configuration. Such degenerate configurations are not easily recognized because parameter changes are typically discrete. Cinderella [74] is an example of geometry software whose design pays particular attention to achieving continuity. Cinderella allows sequential constructions of geometric configurations of points, lines and conic sections. The user can then drag elements and the system updates the configuration in a continuous way. It accomplishes continuity by tracking solutions through complex configurations. Since the paths for changing the configuration avoid singularities by randomization, but do not remember the chosen paths, Cinderella does not exhibit persistence.

Many constraint solvers exhibit persistence without continuity, including the solvers we have designed. Here, persistence is accomplished by finding coordinate-independent characterizations of constraint solutions so that a solution can be designated by a short certificate that is derived from the initial or the current configuration.

7. Open Problems

Parametric design has achieved great accomplishments. However, to take full advantage of its potential, a number of open problems need to be solved. We outline some of the key problems.

7.1. Constraint Solving

Two-dimensional constraint solving has been studied extensively, and although there is no single best technique, successful approaches have been developed that are both efficient and sufficiently general. Geometric constraint solving in three dimensions has been much less explored, except for the purely numerical techniques, whose drawbacks have been discussed in Section 4.

The problem in three dimensions grows in difficulty, not only because the number of variables is larger, but also due to the fact that some of the results valid in two dimensions do not extend to three dimensions. Some recent developments in placing points, lines and spheres with fixed radius in three dimensional space with reasonable computational cost and reliability are reported by Hoffmann and Vermeer, [75, 76] and by Durand [22]. Currently, active research seeks better techniques. A general problem decomposition algorithm has been given in [47].

7.2. Features

A key obstacle is the lack of techniques to support multiple views effectively. Mapping algorithms are needed that can connect different feature schemata with each other and allow designers to edit in different views. This would greatly expand the flexibility of parametric design, and permit designers to increase the semantic content of parametric models.

7.3. Semantics of Parametric Design

Advantages of parametrics lie in the ability of the system to allow easy subsequent edits of the design by changing input parameters that were initially specified when the design was created. To date, all such changes are described in terms of the procedures that actually perform the change. For simple objects, those procedures provide highly productive tools.

When the ranges of parameter values widen or the complexity of the designed objects increases, today's algorithms no longer guarantee that the parametric models are valid and unambiguous, and the results of modeling operations are not always predictable.

As an example, consider the solid shown in Figure 4 left. It was constructed as follows: First, a rectangle was drawn and extruded into a block. On the front face of the block, a circle was drawn as a profile of a slot across the top of the block. Then the left edge of the slot was visually identified for rounding. The result is shown in Figure 4 left. This design is now edited by altering the position of the circular slot profile. The correct result is show in Figure 4 middle, but some systems may construct instead the shape in Figure 4 right, clearly an error.

The problem is how to describe in the generic design the edge to be rounded. An abstract definition of shape change under constraint changes would be needed, as a step towards a rigorous semantic definition of generic design and constraint-based editing.

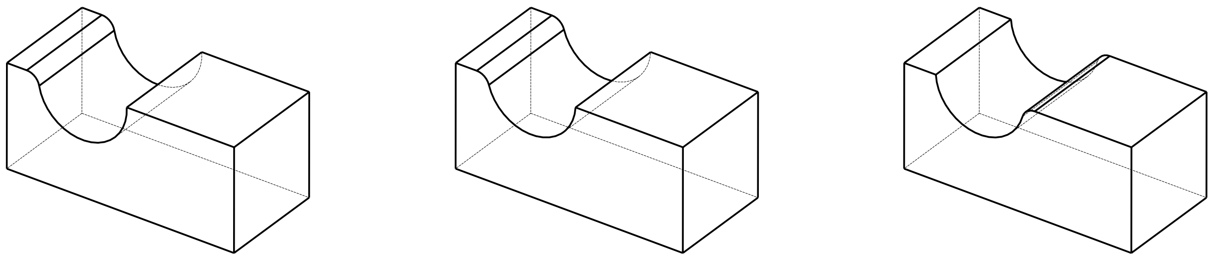

Figure 4. A block with a slot and a round on an edge.

7.4. Assembly-Centric Design

Shape design has traditionally been conceptualized as part-centric design, and CAD systems are very good at designing detailed shapes of individual parts. However, in many applications the interactions of the parts create the primary view of a mechanism, and engineering design often begins from this vantage point. In part-centric design, the focus of the design activity is on the geometry creation, typically of the net shape, and annotations of the shape with features relevant to various views.

To create an assembly-centric parametric design process, it is minimally required to interrelate the parameters and constraints of the various parts of the mechanism to each other. As with views, this raises the question of updating all parts of an assembly when changing a parameter of a single component part. Here, design methodology might guide how to conceptualize key parameters and their functional relevance. Assembly-centric design would find immediate application in areas such as tolerancing and process planning, and in the functional design of abstract mechanisms.

Assembly-centric design would span two levels. On the basic level, parts are interrelated to each other in the assembly, and design parameters would be correlated between parts or derived from assembly-level parameters in an algorithmic way.

A more advanced conceptualization of assembly design would give the designer the capability to differentiate subassemblies and re-use them as parametric elements of the overall design. For instance, we might consider the sensor assembly of an automobile cooling system such a subassembly. Depending on dashboard configurations, this subassembly could be connected to different electric leads, in one combination only lighting up a light indicating operating range temperature and a warning light if the temperature is too high. In another combination the leads from the subassembly could drive a temperature gauge in the dashboard; e.g., [77, 78].

Finally, specific assembly-level parameters could result in the instantiation of differently configured subassemblies that would be responsive to different functional characteristics and ranges.

REFERENCES

1. A.A.G. Requicha and H.B. Voelcker. Solid modeling. A historical summary and contemporary assessment. *IEEE Computer Graphics and Applications*, 2(2):9–24, March 1982.

2. A. Rapoport. Geometric modeling: a new fundamental framework and its practical implications. In C. Hoffmann and J. Rossignac, editors, *Third Symposium on Solid Modeling and Applications*, pages 31–41, Salt Lake City, Utah, May 17–19 1995. ACM Press.
3. V. Shapiro and D.L. Vossler. What is a parametric family of solids? In C. Hoffmann and J. Rossignac, editors, *Third Symposium on Solid Modeling and Applications*, pages 43–54, Salt Lake City, UT, May 1995. ACM Press.
4. C.M. Brown. PADL-2: A technical summary. *IEEE Computer Graphics and Applications*, 2(2):69–84, March 1982.
5. R. Juan, Ll. Solano, E. Torres, and J. Vega. Sistema dmi modelado geométrico de tuberías. In *III Congreso Español de Informatica Grafica*, Granada, Spain, 1993.
6. J.R. Wilkes and R. Leonard. Variant design as a method of automating the design process. *Computer-Aided Engineering Journal*, pages 97–102, June 1988.
7. C.A. McMahon, K. Lehane, J.S. Sims Williams, and G. Webber. Observations on the application and development of parametric-programming techniques. *Computer Aided Design*, 24(10):541–546, October 1992.
8. P.T.J. Andrews, T.M.M. Shahin, and S. Sivaloganathan. Design reuse in CAD environment - Four case studies. *Computers & Industrial Engineering*, 37(1):105–109, 1999.
9. J.E. Fowler. Variant design for mechanical artifacts: A state-of-the-art survey. *Engineering with Computers*, 12(1):1–15, 1996.
10. M. Mitchell, T. Jiao, and J. Jiao. A variant approach to product definition by recognizing fucntional requirements patterns. *Computers & Industrial Engineering*, 33(3-4):629–633, 1997.
11. K.A. Olsen and P. Sætre. Describing products as executable programs: Variant specification in a customer-oriented environment. *International Journal of Production Economics*, 56-57(20):495–502, September 1998.
12. B. Buchberger, G. Collins, and B. Kutzler. Algebraic methods for geometric reasoning. *Anual Review of Computer Science*, 3:85–120, 1988.
13. I. Fudos. *Constraint Solving for Computer Aided Design.* PhD thesis, Purdue University, Department of Computer Sciences, 1995.
14. A.H. Borning. The programming language aspects of ThingLab, a constrained oriented simulation laboratory. *ACM Trans. on Prog. Lang. and Systems*, 3(4):353–387, October 1981.
15. I. Sutherland. Sketchpad, a man-machine graphical communication system. In *Proc. of the Spring Joint Comp. Conference*, pages 329–345. IFIPS, 1963.
16. A. Heydon and G. Nelson. The Juno-2 constraint-based drawing editor. Research Report 131a, Digital Systems Research Center, December 1994.
17. R.C. Hillyard and I.C. Braid. Characterizing non-ideal shapes in terms of dimensions and tolerances. In *ACM Computer Graphics*, pages 234–238, 1978.
18. R. Light and D. Gossard. Modification of geometric models through variational geometry. *Computer Aided Design*, 14:209–214, July 1982.
19. G. Nelson. Juno, a constraint-based graphics system. *SIGGRAPH*, pages 235–243, San Francisco, July 22–26 1985.
20. Y. Hel-Or, A. Rapoport, and M.Werman. Relaxed parametric design with probabilis-

tic constraints. In J. Rossignac, J. Turner, and G. Allen, editors, *Second Symposium on Solid Modeling and Applications*, pages 261–270, Montreal, Canada, May 19-21 1993. ACM Press.

21. N. Kin, Y. Takai, and T.L. Kunii. A connectionist approach to geometrical constraint-solving. In B. Falcidieno and T. Kunii, editors, *Modeling in Computer Graphics*. Springer Verlag, 1993.
22. C. Durand. *Symbolic and Numerical Techniques for Constraint Solving*. PhD thesis, Computer Science, Purdue University, December 1998.
23. H. Lamure and D. Michelucci. Solving geometric constraints by homotopy. In C. Hoffmann and J. Rossignac, editors, *Third Symposium on Solid Modeling and Applications*, pages 263–269, Salt Lake City, Utah USA, May 17-19 1995. ACM Press.
24. C.W. Wampler, A.P. Morgan, and A.J. Sommese. Numerical continuation methods for solving systems arising in kinematics. *Journal of Mechanical Design*, 112:69–68, 1986.
25. B. Buchberger. *Multidimensional Systems Theory*, chapter Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory, pages 184–232. D. Reidel Publishing Theory, 1985.
26. S.-C. Chou. An introduction to Wu's method for mechanical theorem proving in geometry. *Journal of Automated Reasoning*, 4:237–267, 1988.
27. S.A. Buchanan and A. de Penington. Constraint definition system: a computer-algebra based approach to solving geometric-constraint problems. *Computer-Aided Design*, 25(12):741–750, December 1993.
28. K. Kondo. Algebraic method for manipulation of dimensional relationships in geometric models. *Computer Aided Design*, 24(3):141–147, March 1992.
29. B. Freeman-Benson, J. Maloney, and A. Borning. An incremental constraint solver. *Communications of the ACM*, 33(1):54–63, 1990.
30. M. Sannella. The SkyBlue constraint solver. Technical Report 92-07-02, University of Washington, Dep of Computer Science and Engineering, 1993.
31. R.S. Latham and A.E. Middleditch. Connectivity analysis: a tool for processing geometric constraints. *Computer Aided Design*, 28(11):917–928, November 1996.
32. W. Leler. *Constraint Programming Languages: Their Specification and Generation*. Addison Wesley, 1988.
33. B. Aldelfeld. Variation of geometric based on a geometric-reasoning method. *Computer-Aided Design*, 20(3):117–126, April 1988.
34. G. Sunde. A CAD system with declarative specification of shape. *Eurographics Workshop on Intelligent CAD Systems*, pages 90–105, April 1987.
35. W. Bouma, I. Fudos, C. Hoffmann, J. Cai, and R. Paige. Geometric constraint solver. *Computer Aided Design*, 27(6):487–501, June 1995.
36. B.D. Brüderlin. Using geometric rewrite rules for solving geometric problems symbolically. In *Theoretical Computer Science 116*, pages 291–303. Elsevier Science Publishers B.V., 1993.
37. R. Joan-Arinyo and A. Soto. A ruler-and-compass geometric constraint solver. In M.J. Pratt, R.D. Sriram, and M.J. Wozny, editors, *Product Modeling for Computer Integrated Design and Manufacture*, pages 384 – 393. Chapman and Hall, London, 1997.

38. W. Sohrt. Interaction with constraints in three-dimensional modeling. Master's thesis, Dept of Computer Science, The University of Utah, March 1991.
39. A. Verroust, F. Schonek, and D. Roller. Rule-oriented method for parameterized computer-aided design. *Computer Aided Design*, 24(10):531–540, October 1992.
40. Y. Yamaguchi and F. Kimura. A constraint modeling system for variational geometry. In J.U. Turner M.J. Wozny and K. Preiss, editors, *Geometric Modeling for Product Engineering*, pages 221–233. Elsevier North Holland, 1990.
41. C.M. Hoffmann and R. Joan-Arinyo. Symbolic constraints in constructive geometric constraint solving. *Journal of Symbolic Computation*, 23:287–300, 1997.
42. R. Joan-Arinyo and A. Soto-Riera. Combining constructive and equational geometric constraint solving techniques. *ACM Transactions on Graphics*, 18(1):35–55, January 1999.
43. J.C. Owen. Algebraic solution for geometry from dimensional constraints. In R. Rossignac and J. Turner, editors, *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 397–407, Austin, TX, June 5-7 1991. ACM Press.
44. I. Fudos and C.M. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, 16(2):179–216, April 1997.
45. G. Kramer. *Solving Geometric Constraints Systems*. MIT Press, 1992.
46. C.-Y. Hsu. *Graph-Based Approach for Solving Geometric Constraint Problems*. PhD thesis, Department of Computer Science. The University of Utah, June 1996.
47. C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Geometric constraint decomposition. In B. Brüderlin and D. Roller, editors, *Geometric Constraint Solving and Applications*, pages 171–195. Springer, Berlin, 1998.
48. J.J. Shah and M. Mäntylä. *Parametric and feature-based CAD/CAM*. John Wiley and Sons, Inc, New York, 1995.
49. J.J. Shah. Conceptual development of form features and feature models. *Research in Engineering Design*, 2:93–108, 1991.
50. V. Allada and S. Anand. Feature-based modelling approaches for integrated manufacturing: state-of-the-art survey and future research directions. *International Journal of Computer Integrated Manufacturing*, 8(6):411–440, 1995.
51. H. Sakurai and C.-W Chin. Definition and recognition of volume features for process planning. In J.J. Shah, M. Mäntylä, and D.S. Nau, editors, *Advances in Feature Based Manufacturing*, Manufacturing Research and Technology, 20, chapter 4, pages 65–80. Elsevier Science B.V., 1994.
52. J. Han and A.A.G. Requicha. Incremental recognition of machining features. In *Proceedings of the ASME Computers in Engineering Conference*, pages 587–598, Minneapolis, MN, 1994.
53. V. Hetem. Communication: computer aided engineering in the next milennium. *Computer-Aided Design*, 32(5-6):389–394, May-June 2000.
54. O.W. Salomons, F.J.A.M. van Houten, and H.J.J. Kals. Review of research in feature based design. *Journal of Manufacturing Systems*, 12(2):113–132, 1993.
55. X. Chen. *Representation, evaluation and editing of feature-based and constraint-based design*. PhD thesis, Department of Computer Sciences, Purdue University, May 1995.
56. C.M. Hoffmann and R. Joan-Arinyo. On user-defined features. *Computer-Aided Design*, 30(5):321–332, April 1998.

57. C.M. Hoffmann and R. Juan. Erep – An editable high-level representation for geometric design and analysis. In P. Wilson, M. Wozny, and M. Pratt, editors, *Geometric Modeling for Product Realization*, pages 129–164. North Holland, 1993.
58. J.J. Shah and M.T. Rogers. A testbed for rapid prototyping of feature based applications. In J.J. Shah, M. Mäntylä, and D.S. Nau, editors, *Advances in Feature Based Manufacturing*, Manufacturing Research and Technology, 20, chapter 18, pages 423–453. Elsevier Science B.V., 1994.
59. J. Shah, H. Dedhia, V. Pherwani, and S. Solkhan. Dynamic interfacing of applications to geometric modeling services via modeler neutral protocol. *Computer Aided Design*, 29(12):811–824, December 1997.
60. T. Laakko and M. Mäntylä. Incremental constraint modelling in a feature modelling system. *Computer Graphics Forum*, 15(3):367–376, 1996.
61. K.J. de Kraker, M. Dohmen, and W.F. Bronsvoort. Feature validation and conversion. In D. Roller and P. Brunet, editors, *CAD Systems Development*. Springer Verlag, Heidelberg, 1997.
62. M. Dohmen, K.J. de Kraker, and W.F. Bronsvoort. Feature validation in a multiple-view modeling system. In *16th ASME International Computers in Engineering Conference*, Irvin, NY, USA, 19-22 August 1996. ASME.
63. W.F. Bronsvoort and F.W. Jansen. Multi-view feature modelling for design and assembly. In J.J. Shah, M. Mäntylä, and D.S. Nau, editors, *Advances in Feature Based Manufacturing*, Manufacturing Research and Technology, 20, chapter 14, pages 315–330. Elsevier Science B.V., 1994.
64. K.J. de Kraker, M. Dohem, and W.F. Bronsvoort. Multiple-way feature conversion. Opening a view. In M. Pratt, R.D. Siriram, and M.J. Wozny, editors, *Product Modeling for Computer Integrated Design and Manufacture*, pages 203–212, London, UK, 1997. Chapman and Hall.
65. C.M. Hoffmann and R. Joan-Arinyo. CAD and the product master model. *Computer-Aided Design*, 30(11):905–918, September 1998.
66. C.M. Hoffmann and R. Joan-Arinyo. Distributed maintenance of multiple product views. *Computer-Aided Design*, 32(7):421–431, June 2000.
67. X. Chen and C.M. Hoffmann. Towards feature attachment. *Computer Aided Design*, 27(9):695–702, 1995.
68. X. Chen and C.M. Hoffmann. On editability of feature-based design. *Computer Aided Design*, 27(12):905–914, 1995.
69. R. Bidarra and JW.F. Bronsvoort. Validity maintenance of semantic feature models. In W.F. Bronsvoort and D.C. Anderson, editors, *Fifth Symposium on Solid Modeling and Applications*, pages 85–96, Ann Arbor, MI, June 9-11 1999. ACM Press.
70. J. de Kraker, M. Dohmen, and W. Bronsvoort. Maintaining multiple views in feature modeling. In C. Hoffmann and W. Bronsvoort, editors, *Fourth Symposium on Solid Modeling and Applications*, pages 123–130, Atlanta, GA, May 14-16 1997. ACM Press.
71. M. Dohmen. *Constrained-based feature validation.* PhD thesis, Delf University of Technology, 1997.
72. C.M. Hoffmann. On the semantics of generative geometry representations. In *Nineteenth ASME Design Automation Conference*, pages 411–420, New York, NY, 1993. ASME Press.

73. S. Raghothama and V. Shapiro. Consistent updates in dual representation systems. *Computer Aided Design*, 32(8/9):463–477, July/August 2000.
74. J. Richter-Gebert and U. H. Kortenkamp. *The Interactive Geometry Software Cinderella*. Springer, New York, 1999.
75. C.M. Hoffmann and P.J. Vermeer. Geometric constraint solving in R^2 and R^3. In D.-Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry*, pages 266–298. World Scientific Publishing, 1995.
76. C.M. Hoffmann and P.J. Vermeer. A spatial constraint problem. In J.P. Merlet and B. Ravani, editors, *Computational Kinematics'95*, pages 83–92. Kluwer Academic Publ., 1995.
77. ISATP'99. *Proceedings of the 1999 Third IEEE International Symposium on Assembly and Task Planning*, Porto, Portugal, 1999.
78. D.E. Whitney. The potential for assembly modeling in product development and manufacturing. Technical report, MIT, 1996.