Constraint-Based CAD[1]

Christoph M. Hoffmann
Department of Computer Science
Purdue University
West Lafayette, IN 47907

Abstract

CAD systems have become parametric, basing shape design on constraints and design feature operations. We review the development of constraint-based parametric CAD, explaining some of the foundational issues as well as giving an outlook on possible future directions of development.

Keywords: CAD, geometric constraints, design features, parametric CAD, shape editing, semantics.

1. Introduction and Historical Development

Computer-Aided Design (CAD) began in the mid 1970's with two competing approaches.

The Constructive Solid Geometry (CSG) approach by Requicha and Voelcker, then at the University of Rochester, conceptualized a CAD model as an expression; [1]. The operations of expressions were rigid-body motions and regularized Boolean set operations (union, intersection and difference), and whose operands were instantiated simple primitives. Initially the primitives were block, cylinder, cone and sphere, with the subsequent addition of the torus as primitive. The primitives were parameterized, for example the cylinder by diameter and height.

The Boundary Representation (Brep) approach by Braid, then in Cambridge, England, conceptualized a CAD model as a quilt of surface patches; [2]. The patches were joined at edges and vertices and enclosing a solid shape by a 2-manifold without boundary. The patches could be, in the simplest case, polygons. Building complex shapes could be done by regularized Boolean operations on two Brep solids, suitably positioned by rigid-body motions.

Both approaches sought to represent solids. Early on, a key application considered was modeling objects for discrete manufacturing, hence early papers restricted to shapes that were manufacturable and excluding solids such as the one shown in Figure 1.

[1] Work supported in part by NSF grants DMS-013098, DCNS-0216131, DHER-0227828, DSC-0325227, DCMS-0443148, and by an IBM Faculty Scholar award.

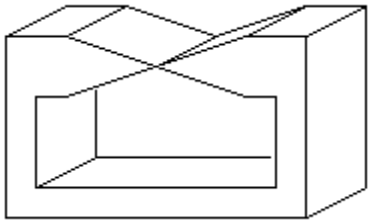

Figure 1: Nonmanifold solid considered not manufacturable

Other considerations included extending the allowed operations on solids, such as rounding, chamfering, shelling, and so on.

Some extending operations, such as rounding and filleting, are easily stipulated and straightforwardly implemented for Brep solids, but they are not so easy to deal with for CSG solids. Uniform radius rounding and filleting was considered by Rossignac [3] who approximated the surfaces that arise in this context by CSG primitives.

The semantics of representing, manipulating and reasoning about solids was also considered early on. Tech memo 28 by Requicha [4] set forth a rigorous semantics for CSG. Comparable attempts at providing a rigorous semantics for Breps fell short in that it would have required exact arithmetic and representational convention for parametric patches that even to-date are not used. Nevertheless, the greater flexibility of Breps in accommodating new operations and ultimately offering greater (practical) flexibility for shape representation persuaded applications to go with Breps and live with the semantic shortcomings of that representation. Inroads into converting between the two representations were made, in particular by Shapiro and Vossler [5]. They give rise to profound mathematical issues when converting Brep to CSG. The opposite conversion, CSG to Brep, was accomplished early-on and is a solved problem (subject to numerical issues); [6].

Summarizing, the two initial approaches to CAD differed in practical expressiveness and in the ease with which to prove rigorously properties of the solids so represented. Applications employed the approaches to model specific solids, with fixed dimensions, i.e., to create instance models; [7]. Although over time Brep became the dominant representation for CAD models, the CSG methodology of creating the bulk of the shape by CSG operations remained a core of the user-interface for many years, even in the commercial Brep systems.

In principle it is possible using CSG to create parameterized models. Two tools are available. The PADL language with which to create CSG models can be embedded into a general purpose programming language, thus allowing users to write programs to create CSG models from input parameter values. Moreover, the primitives are parameterized by specific dimensions, and thus can be resized if those dimensions are used as design parameters. However, CAD models remained primarily instance designs until the 1990s and the arrival of Parametric Technology's Pro/Engineer (ProE) CAD system.

ProE started as another Brep system but the innovation was a sketch-based graphical user interface (GUI) that allowed constraint and dimensional annotations to the sketches. Sketches were then automatically instantiated by solving geometric constraints. Furthermore, the traditional CSG operations (union, intersection, difference) were replaced by operations such as extrude, revolve, protrude, and cut, using the sketches to define profiles with which to carry out these operations. Extrude and revolve are easily recognized as creating primitive shapes, but

protrude and cut are not immediately seen as CSG operations. However, as Chen showed, they can be mapped to CSG operations, thereby giving a firm semantic basis to those operations; [8].

The key innovation of ProE which, by market dynamics, was forced on the other CAD vendors, has been the passage from an instance design to a design family. That is, it has become possible to alter dimensions and constraints of some or all sketches and operations, with the system automatically creating new, related shapes accordingly. We can thus conceptualize this editing process as selecting instances from a parametric family of shapes. This fact was soon recognized, but a generally accepted semantics of the term *parametric family* remains extant. Several proposal have been made, e.g., [9], but vendors differ in the details of the implemented operational semantics of the term, exacerbating CAD interoperability.

2. Constraints in CAD

The sketch interface of modern CAD systems allows the user to put down a rough sketch, usually composed of lines and circular arcs, and annotate the sketch with dimensions and geometric constraints. An example of such a sketch is shown in Figure 2.

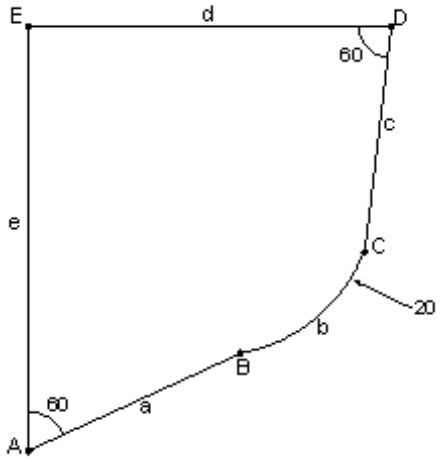

Figure 2: Input sketch with constraints to be solved; arc tangencies required but not annotated

Aside from the dimensional constraints of distance, angle, and arc radius, geometric constraints of tangency at the arc ends are imposed as well as perpendicularity of the segments at the upper left corner. Sketches with constraints can be automatically instantiated, provided a solution exists. The result is, in this case, a profile that can be used for a protrusion or a cut. Using sketches, we can build linearly a sequence of increasingly more detailed shapes, culminating in the final shape model. Subsequent model edits can then alter dimensional values, geometric constraints, or, if necessary, change the entire constraint schema.

2.1. Graph-Based Constraint Solving

CAD applications of constraint based sketching have inspired a large section of the literature on geometric constraint solving; see, e.g., [10]. Of the various techniques to solve geometric constraints, some of them quite old, we will concentrate in the following on graph-based solvers, a dominant solution strategy.

Assume that the sketch is in the Euclidean plane. It then consists of individual sketch elements, points, lines, and circles, and constraints upon them. The constraints may be dimensional,

stipulating specific distances or angles, or they may be geometric, stipulating tangency, perpendicularity, and so on. Not all constraints are explicitly given but may be inferred. Most commonly, incidences are inferred, but sometimes so are tangency and whether segments are vertical or horizontal.

Sometimes a distinction is drawn between parametric solvers and variational solvers. A *parametric solver* is one that solves constraint problems in an explicitly defined sequence. At each step in the sequence, a single geometric element is placed in correct relationship to the elements already localized. In a *variational solver* the problem is not necessarily solved sequentially and there could be steps in the solution that require placing several geometric elements simultaneously. Most solvers of 2D constraint problems are variational.

2.2. Solver Phases

Constrained sketches are solved in two phases. In the first phase, the problem is translated into a graph whose vertices are the geometric elements in the sketch, and whose edges are the constraints upon them. Figure 3 shows the graph constructed from the problem in Figure 2.

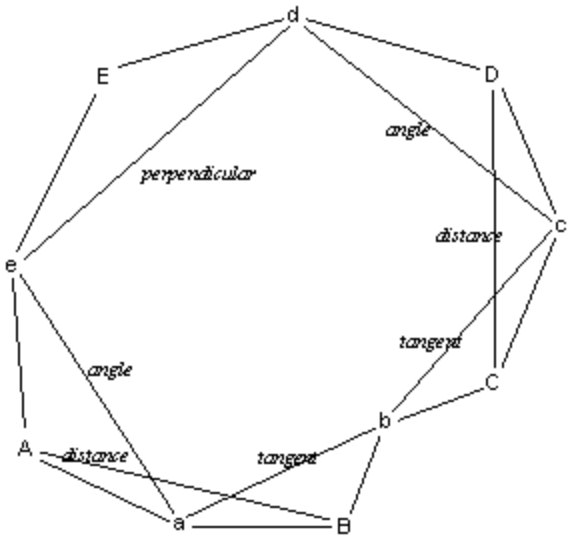

Figure 3: Constraint graph of phase 1; unlabeled edges represent incidences

A graph algorithm analyzes the problem and formulates a solution plan from it. The second phase of the solver then computes actual coordinate values for the geometric elements that satisfy the constraints. The analysis of the first phase usually does not account for specific values of dimensional constraints. For example, assigning lengths to the three sides of a triangle, phase 1 will not check whether the values satisfy the triangle inequality. The first phase is thus a generalized degree-of-freedom analysis.

In bottom-up solvers [11], phase 1 isolates small subproblems that can be solved separately and determines how to put them together recursively into larger parts of the sketch, observing the given constraints. Similarly, in top-down solvers [12], phase 1 dissects the graph recursively into subgraphs that correspond to subproblems that can be combined observing the constraints. Both decomposition directions have their strengths and weaknesses. Generally speaking, a top-down solver naturally recognizes underconstrained problems, whereas a bottom-up solver recognizes overconstrained problems naturally.

If the solver plan calls for solving the subgraph induced by constraint graph vertices[2] *A*, *a* and *B*, phase 2 would be asked to construct a line segment of a specific length. Likewise, consider that the three subgraphs $G_1 = \{e, A, a, B, b\}$, $G_2 = \{b, C, c, D, d\}$, and $G_3 = \{d, E, e\}$ have been solved. Then their combination would require, in phase 2, assembling three geometric structures that pair-wise share the elements *e*, *b*, and *d*. This amounts to finding suitable rigid-body transformations aligning the shared shape elements and can be solved in a simple manner.

2.3. Well-Constrained Sketches

A geometric constraint problem can be underconstrained, overconstrained, or well-constrained; [13]. Loosely speaking, an underconstrained problem has an infinity of solutions and an overconstrained problem has no solution. The exact definition of these terms involves reformulating the problem as a set of algebraic equations and characterizing the dimension of the algebraic set so defined. That is, after formulating all constraints as a system of algebraic equations, and fixing the position and orientation of the sketch with respect to a global coordinate system, a sketch is well-constrained if the system has a finite number of solutions, and it is underconstrained if there is an infinity of solutions. Finally the problem is inconsistently overconstrained if there are no solutions.

This definition of terms would be appropriate for phase 2 of the solver that constructs actual solutions. However, there are approximate counter parts to the three situations that can be determined in phase 1 and have been called structurally underconstrained, structurally overconstrained, and structurally well-constrained. The difference is that a structurally well-constrained problem need not be well-constrained due to particular value configurations of the dimensional constraints or due to unrecognized interdependencies among the constraints that could arise from geometry theorems. For precise definitions see [14].

A problem may be well-constrained, and yet the solver may be unable to find a solution. This situation arises when the constraint pattern is complicated in a technical sense and the solver is unable to determine a solution; [15,16]. This *solver competency* problem arises from the fact that the natural complexity of solving geometric constraints is exponential, thus there cannot be efficient solvers that find a solution of every well-constrained problem in a reasonable amount of time. Such difficult problems are simple to find using classical constructions in geometry that express algebraic relationships. It is possible to express arithmetic relationships among lengths using only circular arcs and line segments, and the resulting problems are often beyond the capabilities of geometric constraint solvers, even when the relations so expressed are simple.

2.4. Root Selection

It is well-known, but often not appreciated, that a well-constrained problem has multiple solutions. Which solution is the one that an applications user intended is usually settled by deducing heuristically topological properties of the input sketch and seeking to preserve them. Practically speaking, this strategy is satisfactory for initial problem formulations. However, when the sketch is edited, it is not at all clear that the previous set of topological properties is appropriate. As example, consider the sketch of Figure 2. A key topological property is that the center of the arc *b* lies in the interior of the profile. But reconsider the problem with different angles, and require that the two angles are 20 degrees instead of 45 and 68 degrees, respectively.

---

[2] Note that the constraint graph vertex *a* represents the line *a* of the sketch

Now one would want a solution in which the arc *b* is centered outside the profile. See also Figure 4.

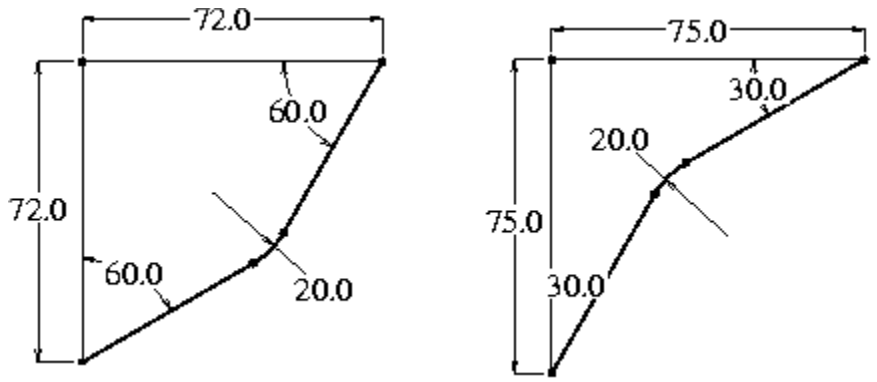

Figure 4: Solutions for different constraint values with different topological properties

The selection of the intended solution of a constrained sketch is known as the *root-selection* problem. In many situations, navigating the solver from an unintended solution to the intended one is a user interface problem that is quite difficult. There have been attempts at devising good user interface dialogues to do this, e.g., [17], but they require that the user understands how the solver works, and most users do not.

2.5. Extensions to the Constraints and Shape Elements

2D constraint solvers have been extended to handle simple parametric curves, and/or conic sections; [18,19,20,21]. Higher order algebraic curves have not been considered. So extending the geometric coverage complicates in most cases only phase 2 of the solver. Briefly, these elements introduce algebraic equations that are more complicated to solve, and solvers often resort to numerical solutions in such situations. This tends to interfere with the root selection problem since in most cases the numerically determined solution cannot be selected based on topological properties of the sketch.

More complicated constraints can also be introduced, including parallel curves (this would lead to higher algebraic degrees for conics as well) minimum distance constraints between curves, and algebraic relations among dimensional constraints. In the latter case it is natural to consider coupling the geometric constraint problem with an equation system in which variables are designated as dimensional constraints. Here the first solver phase has to interleave equation solving and constraint solving in phase 2 so that, at each point, progress can be made; [22].

Applications would also advocate considering soft constraints that are prioritized as to importance, or inequalities that should be respected, or semantic constraints that define the validity of shapes, such as obtaining a contour that does not self-intersect. Typically such additional constraints lead to unsolved problems or to highly complex and potentially inefficient algorithms.

2.6. Spatial Constraint Problems

The simplest 3D constraint problems involve only points and planes. Both points and planes are determined by three coordinates. As they are duals of each other, the algebraic problems they generate are often highly symmetric. Sequential problems involving points and planes are simple

to formulate and solve. Nontrivial simultaneous problems include forward solutions to the Stewart platform and are algebraically much more demanding. Unlike in the planar case, there appear to be no subclasses of 3D problems that are both simple to solve and practical in applications; [23].

When adding lines to the repertoire of 3D solvers, additional difficulties ensue; [24]. They begin with the representation of lines. A line in 3-space has 4 degrees of freedom, yet the various coordinate representations in use usually require six coordinates and two constraint equations upon them. Among the difficult sequential problems we name the problem of finding a line at prescribed distance from four fixed points in space, leading to an algebraic equation system with up to 12 real solutions; [25,26]. Simultaneous problems introduce a combinatorial explosion in the number of essential configurations, even without considering the complex algebraic equation systems entailed by them; [27].

For modeling assemblies, it is clearly appropriate to consider 3D geometric constraint problems. However, since the technology to solve spatial constraint systems is not very mature applications are resorting to coupled 2D constraint problems that are variational in the 2D components and sequential in the spatial arrangement of them. 3D constraint solving and assembly modeling probably evolve in tandem. At this point it appears that each side is stuck because of limitations of the other side.

3. Feature-Based Design

The term *feature* has a lengthy history in CAD and many different definitions have been given. In manufacturing applications, a compelling definition seeks to relate the term to a particular view of the product design; [28,29,30]. This leads to separate definitions of *design feature*, by which is meant an idiosyncratic shape or operation used to design product shape, *machining feature*, an idiosyncratic shape of significance to machining processes used to manufacture a product, *assembly feature*, an idiosyncratic shape of significance to assembling parts, and so on. In the following we restrict to design feature operations used by CAD systems to define shape.

Recall that the shape operations in CSG were Boolean operations. Early on, additional operations were coined, such as rounding and filleting, creating draft angles, chamfering, and other such localized modifications. Sketch-based design operations replaced CSG operations primarily with protrusions and cuts, but soon added global shape operations such as sweeping and shelling. Finally, common operations such as creating ribs or circular holes were stereotyped in the user interface to reduce the amount of parameters the user would be asked to specify. Group operations defining patterns are also used.

3.1. Semantics of Cut and Protrusion

The operations cut and protrusion require a current shape model plus a contour to be used as cross section for the cut or for the protrusion. The cross section must be positioned with respect to the current shape, done usually by selecting a reference plane or surface of the current shape to define the sketching plane. The cut or the protrusion is then done either *blind*, or else to an extent that is related to the current shape. In a *blind* cut or extrusion, we must specify a length for extruding cuts or protrusions, or an angle for revolving cuts and protrusions. These operations have a straightforward meaning easily expressed by CSG operations.

Alternatively, the extent of the cut or the protrusion can be defined by attributes that determine it from the current shape. Thus, a cut *through-next* considers a cut that removes only one contiguous amount of material, whereas *through-all* removes as much material as can be removed by cutting across the entire shape model. Analogously, an extrusion *to-next* adds material that must be contiguous. Such operations can be mapped algorithmically to the construction of an operand that then is used, with the current shape, in a Boolean operation to create the required result; [31].

Semantically, mapping these operations to CSG gives them a firm semantic footing from which to judge correctness of an implementation. However, the attributes *through-next* and *to-next* entail a number of problems that must be clarified to make the operation precise in all situations, and typically no such complete specification is given. Therefore, different CAD systems often differ in what such an operation means. As example, consider Figure 5.

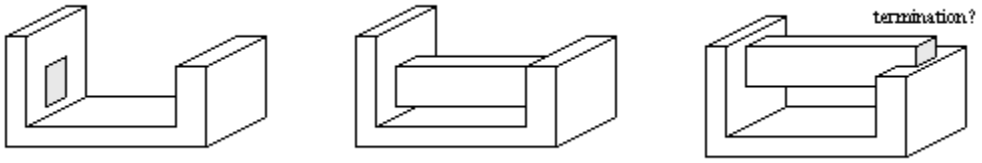

Figure 5: Extrusion *to-next* with uncertain meaning

The operation is initiated by drawing a rectangular contour on a face of the current shape (left in the figure). The *to-next* extrusion adds a rectangular bridge between the two sides, as shown in the center. But if the rectangular contour is shifted upwards, part of the bridge may miss the other side. In this situation is uncertain whether the bridge should extend to the plane defined by the inner face of the right side, or to the plane defined by the outer face, or whether the operation should simply fail.

Note that the example of Figure 5 allows the user simply to indicate where to end the extrusion. The main point of the example is that even in the simple polyhedral case there arise semantic uncertainties. The situation becomes much more complicated for curved faces where there may not be any clearly recognizable termination surface.

3.2. Fillets and Rounds

Adding fillets and rounds to edges of a shape is a common operation. Many papers on the subject have considered the mathematics of surfaces that are needed to smoothly connect two (curved) faces, or multiple faces meeting in a common vertex. The surface constructed achieving the fillet or round is commonly called *blending surface*.

The most natural interpretation of the edge rounding or filleting operation is to consider identifying one or more edges, specifying the radius of the blending surface and then expect a blending surface that is from a suitably bent tube of fixed radius. The cross sections should be approximately circular, in a direction perpendicular to the axis or *spine* of the tube. More complex surfaces are also possible, for example variable-radius blending surfaces where the radius variation is suitably defined.

The description already makes it clear that the mathematical specification of the surface requires many parameters and conventions that are rarely spelled out unambiguously. Customarily,

constant-radius blending surfaces are obtained by approximating the envelope of a rolling ball of constant radius that moves along the edge touching both adjacent surfaces. The definition of constant-radius blends as envelope of a rolling ball of constant radius is precise, but it runs into some local and global problems that are not readily resolved. The local problems include how to end the vertex or edge blend. In the case of edge blends, the problem may require filling gaps (for concave edges) or extending surfaces for edges ending at nonconvex vertices.

The global problems of blending are less often discussed; [32]. They include interactions of different blending surfaces. Consider Figure 6. A circular hole is close to a circular peg. We wish to fillet the base of the peg and round the top edge of the hole. The dashed curves indicate where the fillet and round would end on the top face of the block given a suitable radius for the blend. Clearly, both blends can be constructed individually, but they cannot coexist as specified since this would create a gap in the area where the dashed curves overlap.

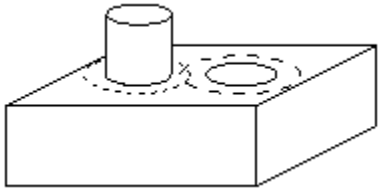

Figure 6: Overlapping blends that would create gaps in the solid surface

In the example of Figure 6, the gap must be closed. How this is done exactly depends on whether the blends are inserted simultaneously, possibly arguing for a symmetric solution, or serially, possibly arguing that the later blend adapt shape to the already existing one.

A different example is shown in Figure 7, reconstructed from [32]. Here the issue is how to combine two interacting blends at a vertex. The situation is complicated by the fact that two of the surfaces become tangential at the vertex. These examples show clearly that a complete semantics of blending solid shapes, as opposed to blending two or more surface patches in isolation, remains an open problem fraught with difficulties that arise from the interactions due to the global geometry of the solid.

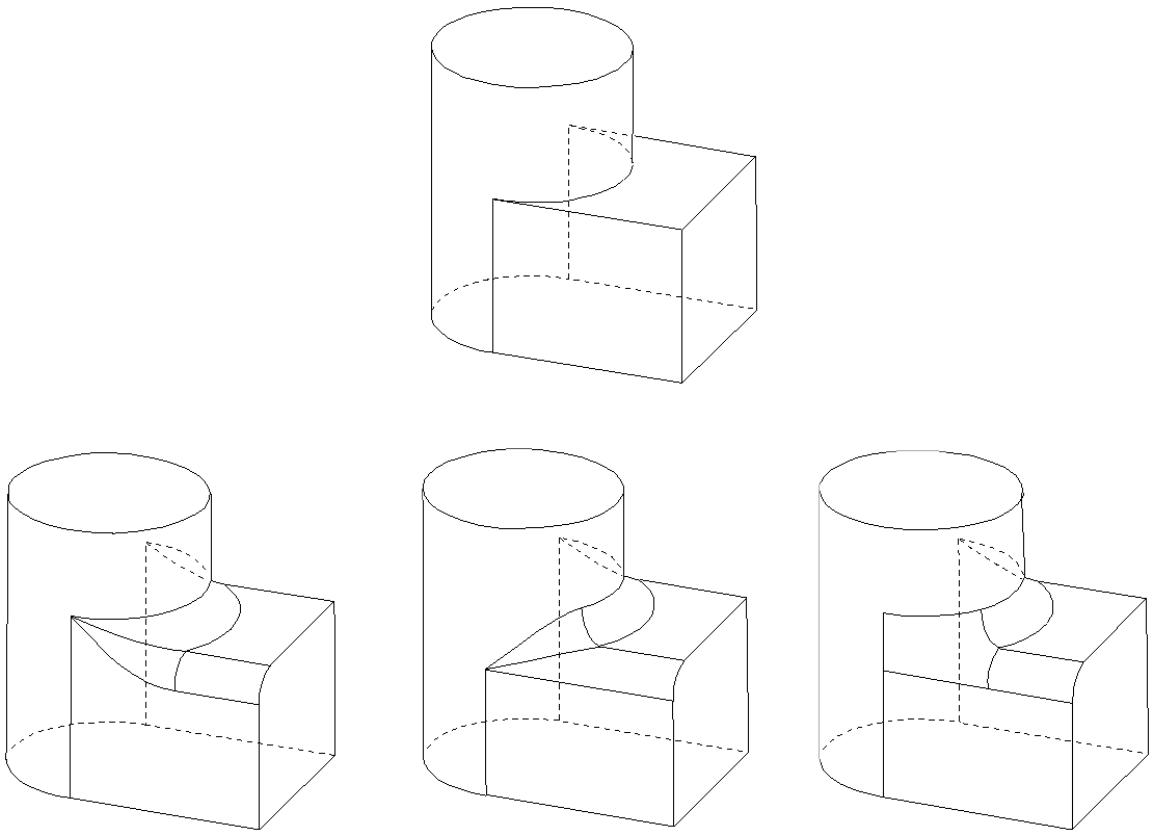

Figure 7: Different joins of a fillet and round at a vertex; see [32]

3.3. Persistent Naming

A CAD shape model is constructed step-by-step using design features and constraints, as discussed before. Editing the model consists of selecting a design feature and deleting it, or changing its parameters (including potentially the cross section). When committing the edit, the system then updates the shape and re-executes the subsequent feature operations automatically. Consider some of the complications this entails.

In the construction of a blend one identifies visually one or more edges of the current shape and then selects and valuates attributes describing the blending surface to be constructed. Since an edge so identified is present only on the current shape instance, the operation of editing the shape may have to reinterpret the user selection on a different shape that arises from modifying one of the earlier design features. That is, we have to find the edge on the changed shape. This requires a description of the selected edge that is independent of the current shape instance. A similar statement can be made about selected vertices and faces. The identification problem has been called the *persistent naming* problem; [33].

Persistent naming is exacerbated by the fact that some selected items arise from the interaction of different design features which, in some shape instances, do not interact at all. That is, the selected edge may be absent after an editing operation and reappear again after another editing operation, or it may be cut into several segments. Other complications include faces that are merged or split by an editing operation. We therefore speak of a parametric family of designs, each design constructed from a sequence of design features that may differ in some of the

attributes or parameters. A particular algorithm for persistent naming, e.g., [34,35,36], is a procedural semantics of the term *parametric family*.

Many examples can be given that show that to-date persistent naming schemes by vendors and researchers can behave counter-intuitive. For example, the sequence dependence of feature operations may lead to unexpected results; e.g., [37,9]. An example is shown in Figure 8.

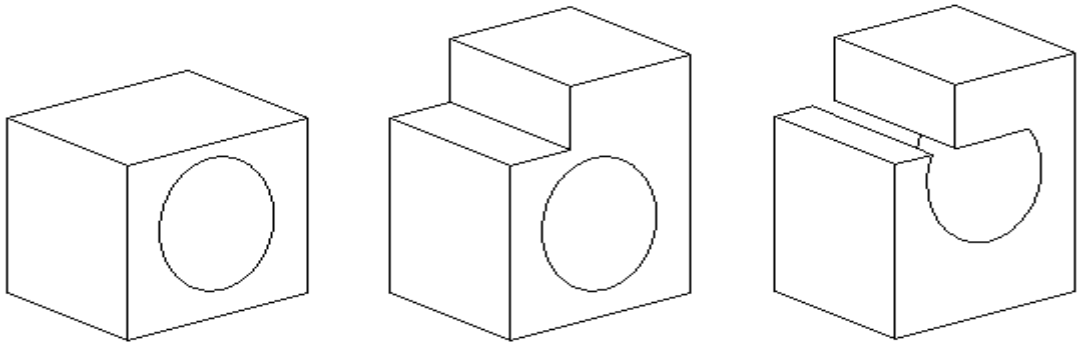

Figure 8: History dependence can lead to unintuitive edits

The design begins with a block into which a hole is cut. An extrusion is later added by drawing a profile on the top face and extruding it vertically. Next, the edit operation repositions the hole. Since this feature is made prior to the extrusion, the hole now cuts the top face into two parts, resulting in the extrusion sketch partially extending into what now is the hole interior. The top extrusion is now added as shown, a logical outcome of the edit but hardly a reasonable one.

In [9], Shapiro and Vossler start a foundational investigation of what constitutes a parametric family of solids, an attempt to find standards by which to judge persistent naming algorithms. They point out that, based on a boundary representation, two solids could be part of the same family if a correspondence is established between the two Breps. For CSG solids, the same question yields instead that the two solids should have trees that correspond. Clearly, the two notions only overlap, neither properly contained in the other. Thus, the familiar instance representations of solids give no clear guidance.

A third approach to finding two instance solids ``related'' is to consider a cell decomposition of two instances, by half spaces that are induced by the respective boundaries. Note that some solids require additional half spaces so that the solid can be described as the disjoint union of certain cells of a spatial subdivision; [5]. In [38], Raghothama and Shapiro postulate that the mapping of the cell complexes should be a continuous map that preserves some orientation criterion of each cell, and map cells consistent with the mapping of the adjacent cells. This framework is derived from the very reasonable postulate that a small change of parameter values should entail a small change in the solid's boundary topology. The framework allows many of the desirable editing operations and rejects many of the undesirable ones. But the conditions do not fit exactly.

4. Outlook

Today, CAD is intimately linked with applications in discrete manufacturing. It is often heard from applications experts that CAD systems are very good at creating geometry, but not very good at creating designs. As we have seen, creating geometry with ease is not necessarily

synonymous with creating and editing geometry in a manner that is both rigorous and intuitive. Thus the remark from applications is rooted in a different situation, of integrating CAD models with conceptual design and with analyses and manufacturing processes.

Since the beginnings of using CAD in manufacturing, in the 1970s and 1980s, much has changed. In the early days, research emphasis was clearly on geometry creation and early pioneers were keen on putting this process on sound footing; [41]. The appetite of practice for sophisticated shape creation operations, as well as the switch to boundary representations as dominant underlying data structure, soon outstripped the semantic. Indeed, further semantic complications arise from the well-known problem of robustness in geometric computations and proposals how to overcome them in a practical way are sparse to-date; [42, 43].

Today, it is inconceivable that large product systems, such as airplanes, cars, and ships, would be designed without the use of CAD systems that create the detailed geometry. CAD is thus a bona-fide part of the product design and manufacturing process. However, as soon as electronic representations of shape have become available, the applications sought to integrate them with other sectors of design and manufacture. In particular, the finite element analyses that are associated with evaluating product performance and manufacturing processes need geometric models, albeit in different representations and from different vantage points. Thus, one speaks of *views* of the model, where each view emphasizes a different conceptualization of the shape model; e.g., [39, 40]. Translating between different views ought to be automated but is not at this point. It requires a deeper understanding of shape than we have at this time and which has been amply illustrated in this paper. It would seem, then, that the applications' comments, about creating designs, relate in particular to this difficulty of switching between views and editing, with ease, a shape -- no matter which view is used.

CAD interoperability is another subject of intense interest to end users. Despite market consolidation that led to the demise of several CAD systems, there are lower-tier suppliers who need to service different primes with different CAD systems, thus placing an extra burden on the suppliers. We have argued in the past that the approach of constraint-based, feature-based design ameliorates the CAD model exchange, [44], but two obstacles can be identified. The first obstacle is the competitive stance of CAD vendors who want to lock in customers with proprietary CAD models and operations on them. The second obstacle is the absence of exact mathematical semantics of the operations which largely preclude exchanging design histories and recreating congruent geometry models in other systems. As we have explained, this second obstacle relates to mathematical difficulties that are not yet overcome. Nevertheless, there have been serious efforts to accomplish CAD system interoperability precisely in this way, by executing the design history and reinterpreting the parameters of the operations that created the shape; [45].

References

1. H. Voelcker and A. Requicha. Geometrical modeling of mechanical parts and processes. *IEEE Computer*, 12:48-57, 1977.

2. Ian Braid. The synthesis of solids bounded by many faces. *Comm. ACM*, 18:209-216, 1975.

3. J. Rossignac. *Blending and Offseting Solid Models*. PhD thesis, Univ. of Rochester, Dept. of ME, 1985.

4. A. Requicha. Mathematical models of rigid solids. Technical Report PAP Tech Memo 28, University of Rochester, 1977.

5. V. Shapiro and D. Vossler. Separation for boundary to CSG conversion. *ACM Transactions on Graphics*, 12:35-55, 1993.

6. A. Requicha and H. Voelcker. Boolean operations in solid modeling: Boundary evaluation and merging algorithms. *Proc. IEEE*, 73:30-44, 1985.

7. C. M. Hoffmann. *Geometric and Solid Modeling*. Morgan Kaufmann, 1989.

8. Xiangping Chen. *Representation, Evaluation and Editing of feature-Based and Constraint-Based Design*. PhD thesis, Purdue University, Dept. of CS, 1995.

9. V. Shapiro and D. Vossler. What is a parametric family of solids? In *Proc 3rd ACM Symp on Solid Modeling*, 1995.

10. C. M. Hoffmann and R. Joan-Arinyo. Parametric modeling. In G. Farin, J. Hoschek, and M.-S. Kim, editors, *Handbook of CAGD*, Chapter 21. Elsevier, 2002.

11. W. Bouma, I. Fudos, C. M. Hoffmann, J. Cai, and R. Paige. A geometric constraint solver. *CAD*, 27:487-501, 1995.

12. J. Owen. Algebraic solution for geometry from dimensional constraints. In *ACM Symp. Found. of Solid Modeling*, pages 397-407, Austin, Tex, 1991.

13. Ioannis Fudos. *Constraint Solving for Computer-Aided Design*. PhD thesis, Purdue University, Dept. of CS, 1995.

14. I. Fudos and C. M. Hoffmann. Correctness proof of a geometric constraint solver. *Intl. J. of Computational Geometry and Applications*, 6:405-420, 1996.

15. C. M. Hoffmann, A. Lomonosov, and M. Sitharam. Decompostion Plans for Geometric Constraint Systems, Part I: Performance Measurements for CAD. *Journal of Symbolic Computation*, 31:367-408, 2001.

16. C. M. Hoffmann, A. Lomonosov, and M. Sitharam. Decompostion Plans for Geometric Constraint Problems, Part II: New Algorithms. *Journal of Symbolic Computation*, 31:409-427, 2001.

17. M. Sitharam. Combinatorial approaches to geometric constraint solving: Problems, progress and directions. In D. Dutta, R. Janardan, and M. Smid, editors, *Computer aided design and manufacturing*, AMS/DIMACS Volume, 2004.

18. C. M. Hoffmann and J. Peters. Tschirnhaus cubics analyzed for a constraint solver. In M. Dahlen, T. Lyche, and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*. Vanderbilt Press, 1995.

19. I. Fudos and C. M. Hoffmann. Constraint-based parametric conics for CAD. *Computer Aided Design*, 28:91-100, 1996.

20. C. M. Hoffmann and C.-S. Chiang. Variable-radius circles of cluster merging in geometric constraints. Part I: Translational clusters. *CAD*, 34:787-797, October 2002.

21. C. M. Hoffmann and C.-S. Chiang. Variable-radius circles of cluster merging in geometric constraints. Part II: Rotational clusters. *CAD*, 34:799-805, October 2002.

22. C. M. Hoffmann and R. Joan-Arinyo. Symbolic constraints in constructive geometric constraint solving. *Journal of Symbolic Computation*, 23:287-300, 1997.

23. C. M. Hoffmann and P. Vermeer. Geometric constraint solving in R^2 and R^3. In D. Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry*. World Scientific Publishing, 1994. 2nd edition.

24. C. Durand. *Symbolic and Numerical Techniques for Constraint Solving*. PhD thesis, Computer Science, Purdue University, December 1998.

25. C. M. Hoffmann and B. Yuan. *There are 12 common tangents to four spheres*, 2000. http://www.cs.purdue.edu/homes/cmh/distribution/SphereTangents.htm.

26. I. Macdonald, J. Pach, and T. Theobald. Common tangents to four unit balls. *Discr. Comp. Geometry*, 26:1-17, 2001.

27. X.-S. Gao, C. M. Hoffmann, and W.-Q. Yang. Solving spatial basic geometric constraint configurations with locus intersection. In *Proc. 7th ACM Symp. on Solid Modeling and Applications*. ACM Press, 2002.

28. J. Shah, D. Hsiao, and J. Leonard. A systematic approach for design-manufacturing feature mapping. In P. Wilson, M. Wozny, and M. Pratt, editors, *Geometric Modeling for Product Realization*, pages 205-222. North Holland, 1992.

29. H. Dedhia, V. Pherwani, and J. Shah. Dynamic interfacing of applications to geometric modelers via neutral protocol. *CAD*, 29:811-824, 1997.

30. J. Shah. Designing with parametric cad: Classification and comparison of construction techniques. In F. Kimura, editor, *Geometric Modeling: Theoretical and Computational Basis Towards Advanced CAD Applications*, pages 53-68. Kluwer, 2001.

31. X. Chen and C. M. Hoffmann. Towards feature attachment. *CAD*, 27: 695-702, 1995.

32. I. Braid. Non-local blending of boundary models. *CAD*, 29:89-100, 1997.

33. X. Chen and C. M. Hoffmann. On editability of feature-based design. *CAD*, 27: 905-914, 1995.

34. J. Kripac. *Topological ID system - A Mechanism for Persistently Naming Topological Entities in History-based Parametric Solid Models*. PhD thesis, Czech Technical University, Prague, 1993.

35. V. Capoyleas, X. Chen, and C. M. Hoffmann. Generic naming in generative, constraint-based design. *CAD*, 28: 17-26, 1996.

36. S. Raghotama and V. Shapiro. Boundary representation deformation in parametric solid modeling. *ACM Trans on Graphics*, 17:259-286, 1998.

37. C. M. Hoffmann. On the semantics of generative geometry representations. In *Proc. 19th ASME Design Automation Conference*, pages 411-420, 1993. Vol. 2.

38. S. Raghotama and V. Shapiro. Consistent updates in dual representation systems. *CAD*, 32:463-477, 2000.

39. 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.

40. J. de Kraker, M. Dohmen, and W. F. Bronsvoort. Maintaining multiple views in feature modeling. In C. M. Hoffmann and W. F. Bronsvoort, editors, 4th *Symposium on Solid Modeling and Applications*, pages 123-130, Atlanta, GA, 1997.

41. A. Requicha and H. Voelcker. Solid modeling – a historical summary and contemporary assessment. *IEEE Comp. Graphics and Applic.*, 2:9-24, 1982.

42. C. M. Hoffmann. Robustness in geometric computation. *JCISE*, 1:143-156, 2001.

43. C. M. Hoffmann and N. Stewart. Accuracy and semantics in shape-interrogation applications. *Geometric Models*, Vol. 67, forthcoming; 2005.

44. 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, 1992.

45. S. Spitz and A. Rappoport. Integrated feature-based and geometric CAD data exchange. *Proc. ACM Symp. Solid Modelingand Applic.*, G. Elber, N. Patrikalakis, P. Brunet, eds., 183-190, Eurographics, 2004.