Persistent Naming Through Persistent Entities

Rafael Bidarra and Willem F. Bronsvoort
Faculty of Information Technology and Systems
Delft University of Technology
Mekelweg 4, NL-2628 CD Delft, The Netherlands
Email: (Bidarra/Bronsvoort)@cs.tudelft.nl

Abstract

Current parametric modeling systems suffer from the persistent naming problem, which is responsible for the unpredictable, sometimes stunning, behavior of such systems when re-evaluating a model, even after simple editing operations.
This paper claims that the problem is an inherent difficulty of history-based parametric modeling, and that it is of little use to insist on developing more and more persistent naming schemes which end up solving only a fraction of the problem. Instead, it is argued that the rationale behind such schemes should itself be revised. Alternative approaches to define a parametric model based on persistent parametric entities can, in fact, eliminate the use of references to non-persistent geometric model entities, which is the cause of the problem.
One such approach is described here, which is able to take full advantage of parametric solid modeling. It provides persistent entities in the parametric definition domain, which can be safely and consistently referred to. A number of examples illustrate how user specification of modeling operations can be performed through interaction with a declarative feature model.

Keywords: *parametric modeling, feature modeling, boundary representation, persistent naming*

1. Introduction

Most current modeling systems are parametric, history-based feature modeling systems using a dual representation for solid objects: on the one hand a parametric definition of the object, on the other hand a boundary representation (b-rep) of the resulting shape; see Figure 1. The parametric definition can consist of a large variety of information, usually organized in a graph structure. Typically, the graph represents relations involving instances of parameterized features, constraints, set operations, auxiliary geometric entities, etc. Among these relations, the most important is usually the *order of creation* of feature instances, defining the so-called *model history*.

Creating a new feature in the model, by providing a set of input values for its parameters, appends a new node to the model history, yielding a new parametric definition of the object. Similarly, feature instances can be modified by specifying new values for their parameters, or be deleted from the model. This is done by modifying, or deleting, the respective feature node in the model history. Whenever a new parametric definition is obtained, it can be input to the boundary evaluator, which generates the corresponding new boundary representation. This can be realized by, for example, sequentially re-executing all operations in the modified model history.

The ability to easily generate variants of a feature model has been pointed out as the most important reason for the popularity of current parametric modeling systems [9]. An important characteristic of these systems, highlighted by the loop in Figure 1, is that they allow the use of topologic entities (e.g. edges and faces) of the b-rep in

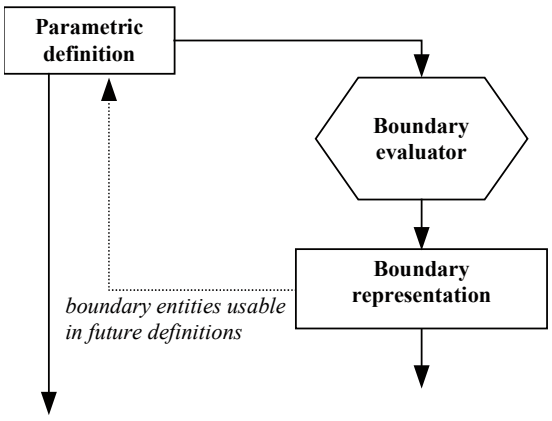
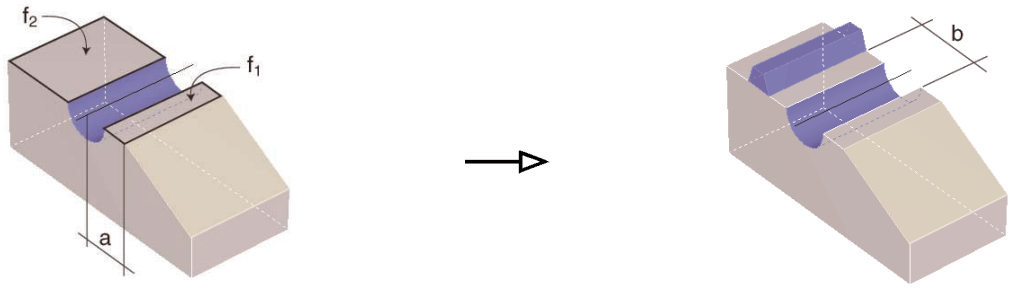


Figure 1. Dual representation in current parametric modeling systems

0-7695-1674-2/02 $17.00 © 2002 IEEE
IEEE COMPUTER SOCIETY



(a) an attach face is split by the round slot

(b) a resulting edge is used for positioning the rib

Figure 2. Using boundary entities in parametric definitions

the specification of subsequent parametric definitions. Consequently, each node in the model history typically contains references to topologic entities of the intermediate b-rep which resulted from the previous modeling step. As an example of this, the model in Figure 2.a shows a cylindrical through slot, attached to the top face of the model, and positioned at a distance *a* from the indicated edge, causing the top face of the model to be split into faces f_1 and f_2. In the next modeling step, a rib is attached to face f_2, at a distance *b* from an edge of f_1, as indicated in Figure 2.b.

Obviously, references to boundary entities in parametric definitions should always be unambiguous, so that these entities can be unequivocally identified, every time the boundary evaluator is called to create a new b-rep. However, a general property of b-reps is that topologic entities may be split, as the top face in Figure 2.a, merged or deleted because of modeling operations. As a result, parametric definitions referring to such entities can become ambiguous or invalid, and therefore unsuitable for the boundary evaluator. This is usually called the *persistent naming problem*.

This problem has been classified as one of the most serious difficulties plaguing current parametric solid modeling systems. As Ragothama and Shapiro [8] put it, "the new solid modeling systems no longer guarantee that the parametric models are valid and unambiguous, and the results of modeling operations are not always predictable".

This paper claims that this problem is an inherent difficulty of history-based parametric modeling, and that it is of little use to insist on developing more and more persistent naming schemes which end up solving only a fraction of the problem. Instead, it revises the rationale that led to such schemes, and presents an alternative approach to parametrically define a model, which does not involve any references to changeable boundary model entities, but instead to persistent entities in the parametric definition only.

The paper first briefly surveys current approaches to the persistent naming problem (Section 2). Next, an alternative approach to the definition of parametric models is presented, which effectively solves the problem (Section 3). This approach is further elaborated, and its use is illustrated with examples of parametric definitions referring to faces (Section 4) and to edges (Section 5). These examples have been kept relatively simple, to best illustrate the basic ideas, but more complex models are also correctly handled. Finally, some conclusions are given (Section 6).

2. Current approaches

Persistent naming is the process of tracking and identifying topologic entities as a model evolves. Currently, all commercial parametric modeling systems use a b-rep, and have their own scheme for maintaining persistent names, based on their own taxonomy of situations, heuristics and matching rules. Little is known of their details, which are proprietary information, let alone of the underlying theory, if any.

The main goal of such schemes is to guarantee that the boundary evaluator produces a valid boundary, in a deterministic way. In practice, although they seem to work correctly in some foreseen situations, all these schemes exhibit unpredictable results after certain modeling operations, which are not even the same for all systems. Because there are no standards for either defining persistent names or consistently using them in parametric definitions, "different systems employ incompatible, ad hoc, and often internally inconsistent semantics for processing parametric models" [9].

A typical situation is illustrated in Figure 3, with a very simple operation on the model of Figure 2.b. The position

IEEE COMPUTER SOCIETY

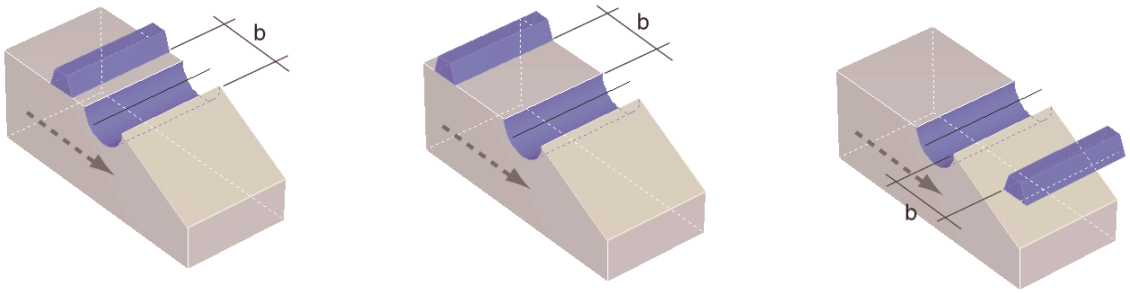


Figure 3. Disparity of boundary evaluation results

of the round slot is modified, making the parameter value *a* equal to the radius of the slot, by which face f_1 in Figure 2.a collapses into an edge. According to the positioning constraints specified, the user would most likely expect the model in Figure 3.a as the result of this parameter modification. However, some current modeling systems produce the model in Figure 3.b, and other systems the model in Figure 3.c, where the rib is flipped across the slot and becomes disconnected from the object.

Because the slot edge used to position the rib (Figure 2.b) has been merged with another edge, it is possible to guess what the rationale is behind these disparate results: according to its particular naming scheme, a system may select the merged edge (Figure 3.a), another edge of the slot (Figure 3.b), or the merged edge but the other side of it (Figure 3.c), to position the rib.

Several schemes have been proposed to alleviate the persistent naming problem [4, 3, 2, 5], so that the boundary re-evaluation can at least be consistently executed. See ref. [6] for a recent survey on the persistent naming problem, containing several more references. All these schemes use auxiliary data structures to keep track of how faces and edges evolve, but don't really solve the problem, as explained in ref. [8], where the persistent naming problem was first formalized.

Raghothama and Shapiro [8] introduced *boundary representation deformations* in their approach to the problem, which they related to the question of how to define families of objects in parametric modelling. The main issue in the latter is to determine which objects belong to a family, defined by a prototype model, and which objects do not. Basically they argue that as long as a continuous boundary deformation is possible from the prototype model to the new instance, the latter is considered to be a member of the family. However, as they also admit, continuous deformations seem to be too restrictive, and operations like splitting and merging of model entities would have to be dealt with, in order to allow topological changes such as the elimination of holes, but this has not been elaborated.

It can be concluded that the current approaches to persistent naming have not completely solved the problem. This is caused by a wrong starting point: they all try to *keep track of b-rep entities that are not persistent,* and this is *impossible* in a truly generic way. It seems, therefore, more effective to develop a new approach, with a better starting point.

3. An alternative approach

All approaches mentioned in the previous section are in one way or another tied to the generic scheme of Figure 1. Their basic reasoning goes as follows:

1. intermediate boundary entities *must* be available for use in parametric definitions;
2. however, such entities have a transient existence in b-reps;
3. hence, we need to give them persistent names.

As a result of the closed loop in Figure 1, parametric definitions are dependent on (the entities of) a previously generated b-rep, while at the same time they determine a new resulting b-rep. Since this interdependence between parametric definitions and boundary representations is the main cause of the persistent naming problem, one may legitimately wonder whether it is possible to eliminate the loop in Figure 1. In other words, wouldn't it be possible to drop requirement 1 above, and still provide powerful parametric definition capabilities?

This question is affirmatively answered in this section, which presents an alternative approach to the *parametric definition of feature models*. Its basic idea can be summarized as follows:

1. provide persistent entities in the parametric definition domain (instead of in the b-rep domain);
2. consistently refer to these entities, whenever needed, in any parametric definition.

IEEE COMPUTER SOCIETY

A similar idea has been hinted before by Pratt [7], but, to the best of our knowledge, it has never been elaborated. Clearly, the lion's share of this approach lies in task 1 above. Once we have persistent entities at our disposal, using them in a systematic way is straightforward, provided that they are generic and intuitive enough to express our intent. The next subsections introduce three categories of such entities that effectively support this approach: *feature*, *reference* and *constraint* classes. A parametric feature model basically consists of interrelated instances of these classes, and thus contains persistent entities only.

3.1. Feature classes

The notion of *feature class*, extending that used in current parametric feature modeling systems, is particularly useful in this approach. A feature class can be defined as a structured description of all properties of a given feature type, defining a template for all its instances.

Feature class descriptions build on a parameterized volumetric shape, accounting for a bounded region of space, called the feature's *shape extent*. This parameterized shape allows us to think of a feature class as a parametric family of objects.

On the boundary of the shape extent, one can distinguish each boundary entity (e.g. face or edge) with a unique, generic name. Such entities are usually called *feature elements*. For example, Figure 4.a depicts the shape of a cylindrical hole class, indicating some of its feature element names. As will be shown in the following sections, such feature elements, unlike b-rep entities, are never split, merged or deleted, even though their geometric representation may be.

In addition to the boundary entities just mentioned, other uniquely named feature elements linked to the feature shape may be specified as auxiliary geometric entities in a feature class, so-called datums or *references*. Examples of these are the axis line reference or a longitudinal symmetry plane reference of a cylindrical hole; see Figure 4.b.

Internally, a feature class encapsulates a set of constraints which geometrically and/or algebraically relate its shape parameters and its feature elements. All these feature constraints and feature elements are members of the feature class, and are therefore automatically instantiated with each new feature instance. Furthermore, a feature class associates to its shape the notion of *feature nature*, indicating whether its feature instances represent material added to or removed from the model (respectively *additive* and *subtractive* natures).

Typically, the feature elements and a number of feature shape parameters are made public in a feature class, and belong therefore to the class interface. In addition, each feature class may specify in its interface a number of positioning and orientation schemes, which relate some of its

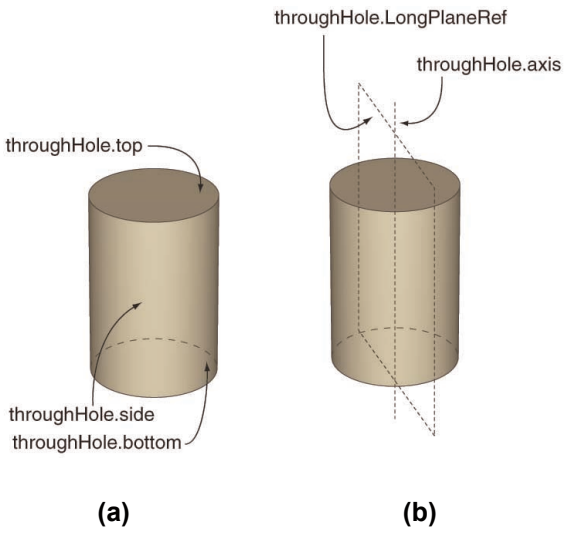


Figure 4. Feature elements of a cylindrical hole

feature elements to external entities. The latter can be chosen out of, for example, the feature elements of features already present in the model. For example, the top and bottom faces of a through hole's cylindrical shape may be used to attach it to, say, the top and bottom faces of a block, respectively.

The main advantage of using feature elements for attaching and positioning features relative to each other in a model, is that each of these elements is indeed *persistent*, and therefore its unique name is also *persistent*. In fact, assuming no two feature instances have equal names, the combination

```
<feature instance name>.<feature element
    name>
```

univocally determines one and only one entity, which can always be found in the parametric definition, as long as that feature instance has not been explicitly removed.

The next subsection explains that besides feature elements, also other persistent entities may be useful in parametric definitions for attaching, positioning or orientating features.

3.2. Reference classes

The use of references was introduced above with the goal of providing auxiliary datums for a feature shape, within the scope of feature classes. This notion can now be taken one step further, by observing that standalone reference instances can also play a very useful role in creating and editing features in a model. For example, instead of positioning each hole in a row relative to some common feature element in the model, it may be much more convenient to first place a reference plane, at a given distance of

IEEE COMPUTER SOCIETY

that common feature element, followed by the creation of the hole instances, each one positioned relative to this reference plane. Subsequent displacement of these holes can then be easily achieved in one step by modifying that reference's position. In other words, this use of references allows positioning of other features relative to entities in the parametric definition that do not even correspond to any face or edge in the boundary representation of an object.

In order to make possible the creation and use of such standalone reference instances in parametric definitions, a number of *reference classes* should be available to the user of the modeling system. Basic examples of these are a plane reference class and a line reference class, from which other classes can easily be derived or composed. Each reference class should provide its own positioning and orientation scheme, which fixes the degrees of freedom of a reference relative to other feature or reference elements in the parametric model. Again, referring to such entities is not only unambiguous, but it is also always valid, as long as those entities remain in the parametric definition.

Similarly as for feature instances, each reference instance is assigned a unique name when it is created in a parametric definition.

3.3. Constraint classes

The third and last building block for sound parametric definitions in this approach regards *constraint classes*.

Various categories of constraints exist (e.g. algebraic, geometric and topologic constraints), each one with its own solving methods. We concentrate here on geometric constraints, because typically this category is most directly involved in current persistent naming problems, as illustrated in the example of Figure 3.

Geometric constraints were already implicitly referred to in the previous subsections, in connection with the positioning and orientation schemes of features and references. For that purpose, they were members of the respective feature or reference class, hence automatically created with each of its instances.

However, additional geometric constraint instances can be profitably used in parametric definitions in order to express one's intent in a model. In conformity with the goal of the new approach, the only condition for the use of such constraint instances is that they should establish their relations among entities in the parametric definitions, i.e. feature element instances and/or reference instances, instead of among boundary entities in the b-rep of the object. Simple examples of geometric constraint classes are *distance-face-face* – imposing a given distance between two parallel, planar entities –, and *distance-line-face* – imposing a given distance between a reference line and a parallel, planar entity.

4. References to feature faces

In this section it will be illustrated how our approach works with a few examples of parametric definitions involving feature elements. For this, one should keep in mind that *each face in the b-rep of a feature model is always a geometric representation of (possibly a part of) a feature element of type 'face'* (or *feature face*, for short). In more informal terms, if one would "stamp" each b-rep face with the name of the feature face from which it originates, no b-rep face would remain unvisited. This property is very important for user interaction purposes, at the moment of picking some *b-rep face* on the visualized model, e.g. in order to attach or position a feature: it is always possible to find out which corresponding *feature face* should be stored in the parametric definition of the operation.

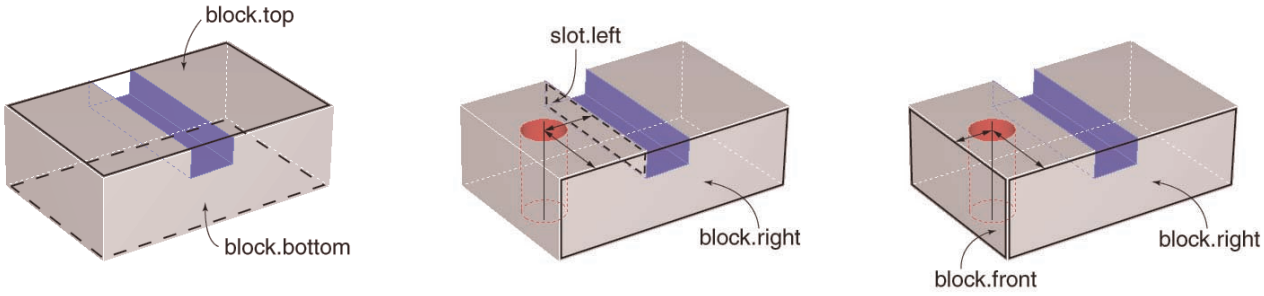
For example, if one wishes to attach a through hole instance to the model of Figure 5.a, as shown in Figure 5.b, all that is needed is to let it refer to the top and bottom feature faces of the block instance as its attach faces. Additionally, one will have to choose two other feature elements for fixing the through hole's position, e.g. the block's right face and the slot's left face, as shown in Figure 5.b.

All entities referred to in this operation are persistent feature elements, and thus always available in the parametric definition of the object. Moreover, the relations established with this operation unambiguously define a dependency between the new feature (the through hole) and the features which it refers to (the block and the slot). As a result, it is always possible to determine within a parametric definition which entities depend on a given entity, so that removal of the latter may be disallowed as long as those dependencies are not cleared away. In the example of Figure 5.b, removing the slot of the model is not possible as long as it has the through hole as a dependent feature instance. Fortunately, this dependency can easily be removed, by modifying the parametric definition of the through hole, for example, repositioning it relative to the block's front face; see Figure 5.c.

5. References to boundary edges

The example in the previous section deals with the use of feature faces for attaching and positioning features. However, some feature classes, e.g. chamfers and blends, are intuitively associated with edges, rather than with faces, on a b-rep. An apparent problem with this is that, in contrast to the correspondence observed in Section 4 between b-rep faces and feature faces, not all edges in the b-rep of a feature model are a geometric representation of some *feature* element of type 'edge' (or *feature edge*, for short).

IEEE COMPUTER SOCIETY



(a) attach faces **(b) positioning faces** **(c) alternative positioning faces**

Figure 5. Referring to feature faces to attach a through hole

For example, in the model with two crossing slots of Figure 6, the highlighted edges are not representing feature edges of either slot, but result instead from the intersection of their side faces. Such edges of a b-rep are here called *intersection edges*, to distinguish them from those representing feature edges.

In any case, for *each edge* in a manifold b-rep holds that it *has always two and only two adjacent b-rep faces*. In turn, because each of these two b-rep faces is always representing some feature face, the problem of referring to a b-rep edge can again be transferred to a problem in the parametric definition domain, involving the corresponding feature faces. These two faces can therefore be used to identify that edge, although they will not always be sufficient, as will be shown later in this section.

A few examples are now given that illustrate how this method of referring to edges is put into practice in various situations. In the first example, a chamfer is created on an edge of a slot; see Figure 7. To specify this in the parametric definition of the chamfer, the two feature faces –slot.right and block.top – corresponding to the b-rep faces adjacent to that edge, are stored as the attach faces for the chamfer. In addition, two other feature faces should be provided that bound the start and end of the chamfer, in this case block.left and block.right.

The fact that two feature faces, instead of two b-rep faces, are used to identify an edge does not necessarily mean that only a complete feature edge can be referred to, as was the case in the chamfer of Figure 7. For example, in Figure 8 another chamfer is attached to a b-rep edge that is a part of a block edge. Although the feature edge adjacent to the feature faces block.top and block.right corresponds to two b-rep edges (see Figure 8.a), the start and end bounds of the chamfer in Figure 8.b – block.front and slot.left – univocally determine the desired result.

A third example shows that this method also applies for referring to intersection edges. In Figure 9, a chamfer on an intersection edge is specified, by identifying its two adjacent b-rep faces, determining the corresponding feature faces –slot1.left and slot2.right–, as shown in Figure 9.a. The start and end of this chamfer are specified as indicated in Figure 9.b.

All examples given so far deal with planar feature faces, which always intersect along a *single* rectilinear edge. Dealing with non-planar feature faces, in contrast,

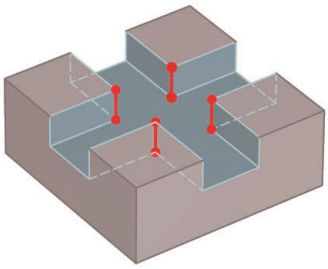
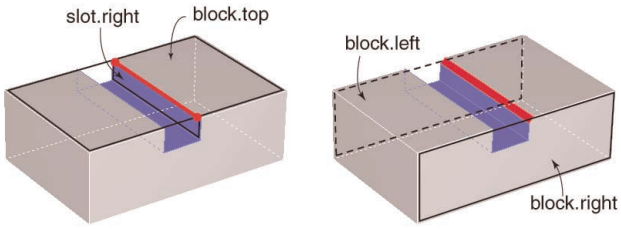


Figure 6. Intersection edges on a b-rep model



(a) **(b)**

Figure 7. Creating a chamfer on a feature edge

0-7695-1674-2/02 $17.00 © 2002 IEEE
IEEE COMPUTER SOCIETY

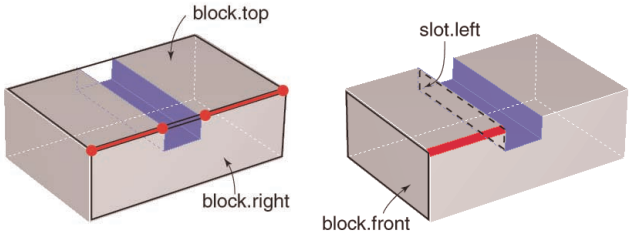


Figure 8. Creating a chamfer on part of a feature edge

requires the ability to handle *multiple* intersection edges between feature faces. A number of examples will demonstrate how the approach successfully applies to such cases as well.

The basic idea is that whenever two feature faces intersect along multiple edges, the explicit use of references should identify the particular edge to be chosen. In the following examples, distinguishing such multiple intersections is achieved through the use of auxiliary planar references, *defined within the feature classes with shapes involving non-planar faces*; see Figure 4 in Subsection 3.1 for an example.

In the first example, a blend has to be made on one of the side edges of a round slot; see Figure 10.a. Because the two feature faces defining that edge –cylinder.top and roundslot.side– intersect twice, the longitudinal plane reference of the round slot is used to identify the edge to be blended. In Figure 10.b, the one in its right half space has been chosen.

The second example consists of blending one of the edges highlighted in Figure 11.a. Because the two cylindrical feature faces –cylinder.side and roundslot.side– intersect twice, a transversal plane reference of the round slot is used to identify the edge to be blended; see Figure 11.b.

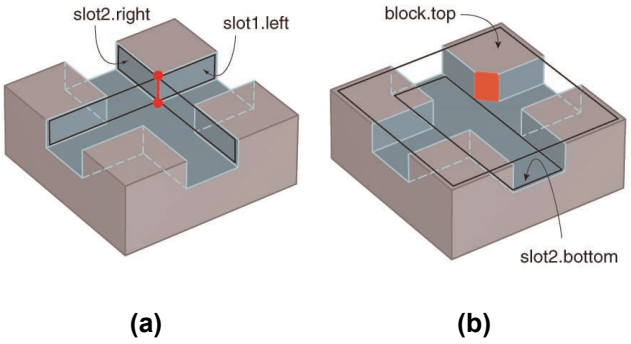


Figure 9. Creating a chamfer on an intersection edge

An advantage of this approach is that the disambiguating role of these references only needs to be invoked if the two attach feature faces actually have multiple edge intersections. For example, displacing the round slot in Figure 11.b to the left, as shown in Figure 11.c, yields one single intersection edge. Hence there is no longer ambiguity for attaching the blend feature to the edge defined by the cylinder.side and the roundslot.side faces. In this case, even though the transversal plane reference of the round slot still keeps its information, it does not need to be taken into account. However, if later on the round slot is moved back to, say, its original position of Figure 11.b, the reference's information would again be used to select the correct edge for the blend feature.

If the round slot were originally located as shown in Figure 11.c, and the edge blend subsequently applied, an essentially different situation would occur, because that edge can be univocally determined by its two adjacent faces, without recurring to references of any features. However, if later on that round slot is displaced to the position shown in Figure 11.b, so that two intersection edges occur, the user will have to explicitly identify which of them (or possibly both) should be blended.

6. Conclusions

An alternative approach has been presented to solve the well-known persistent naming problem in parametric feature modeling systems. Usually a dual representation is used in such systems: a parametric definition and a b-rep

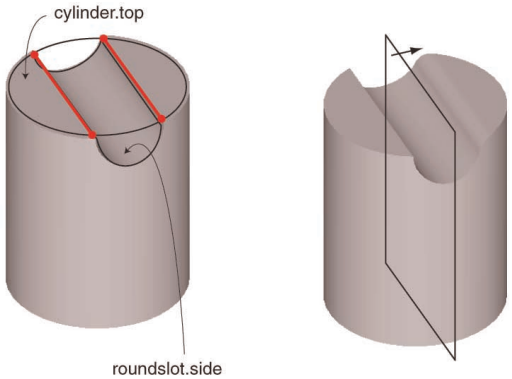


Figure 10. Identifying an edge for blending

IEEE COMPUTER SOCIETY

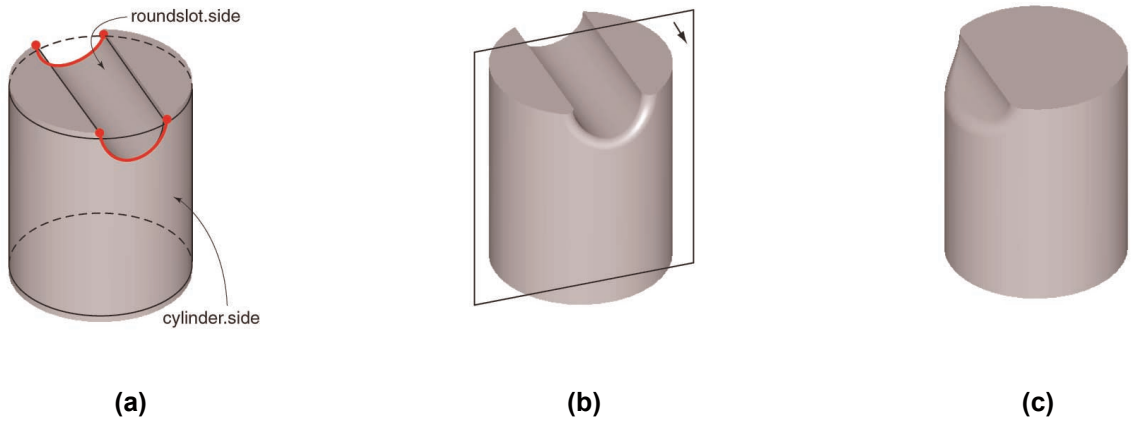


Figure 11. Identifying an edge for blending at feature instantiation stage

of the resulting shape. It was shown that the problem occurs because in the parametric definition of an object, non-persistent entities in intermediate b-reps are referenced to attach and position additional features in the model.

Previous attempts to solve the problem tried to keep track of all non-persistent entities with auxiliary data structures, but this has so far not been successful, and it is questionable whether it will ever be.

A completely different approach was therefore proposed, in which all references are made to entities in the parametric definition, instead of in the b-rep. Such entities are persistent and so, obviously, their names too. One might say that in this way the persistent naming problem in its original formulation was in fact avoided.

The approach has been illustrated with several examples resulting from an implementation in our prototype semantic feature modeling system [1]. However, it may be used in any parametric modeling system, as long as there is a clear separation between the parametric definition and the representation of the resulting shape, and all references are made to entities in the parametric domain.

Acknowledgements

We thank Klaas Jan de Kraker for valuable comments on the manuscript, and Eelco van den Berg for his assistance in producing most figures.

References

[1] Bidarra, R. and Bronsvoort, W.F. (2000) Semantic feature modelling. *Computer Aided-Design* **32**(3): 201–225

[2] Capoyleas, V., Chen, X. and Hoffmann, C.M. (1996) Generic naming in generative, constraint-based design. *Computer-Aided Design* **28**(1): 17–26

[3] Chen, X. and Hoffmann, C.M. (1995) On editability of feature-based design. *Computer-Aided Design* **27**(12): 905–914

[4] Kripac, J. (1995) A mechanism for persistently naming topological entities in history-based parametric solid models. In: *Proceedings Solid Modeling '95 – Third Symposium on Solid Modeling and Applications, 17–19 May, Salt Lake City, UT, USA,* Hoffmann, C.M. and Rossignac, J.R. (Eds.), ACM Press, New York, pp. 21–30. Also in: *Computer-Aided Design* **29**(2): 113–122

[5] Lequette, R. (1997) Considerations on topological naming. In: Product Modeling for Computer Integrated Design and Manufacturing – Proceedings TC5/ WG5.2 International Workshop on Geometric Modeling in Computer Aided Design, 19–23 May 1996, Airlie, VA, USA, Pratt, M., Sriram, R.D. and Wozny, M.J. (Eds.), Chapman & Hall, London, pp. 394–403

[6] Marcheix, D. and Pierra, G. (2002) A survey of the persistent naming problem. In: *Proceedings Solid Modeling '02 – Seventh Symposium on Solid Modeling and Applications, 17-21 June, Saarbrücken, Germany,* Lee, K. and Patrikalakis, N.M. (Eds.)

[7] Pratt, M.J. (1988) Synthesis of an optimal approach to form feature modelling. In: *Proceedings of the 1988 ASME Computers in Engineering Conference, August, San Francisco, CA, USA,* ASME, New York, Vol. 1, pp. 263–274

[8] Raghothama, S. and Shapiro, V. (1998) Boundary representation deformation in parametric solid modeling. *ACM Transactions on Graphics* **17**(4): 259–286

[9] Shapiro, V. and Vossler, D.L. (1995) What is a parametric family of solids? In: *Proceedings of Solid Modeling '95 – Third Symposium on Solid Modeling and Applications*, *17–19 May, Salt Lake City, UT, USA,* Hoffmann, C.M. and Rossignac, J.R. (Eds.), ACM Press, New York, pp. 43–54

IEEE COMPUTER SOCIETY