1994

Generic Naming in Generative, Constraint- Based Design

Vallis Capoyleas

Xiangping Chen

Christoph M. Hoffmann
Purdue University, cmh@cs.purdue.edu

Report Number:
94-011

GENERIC NAMING IN GENERATIVE, CONSTRAINT-BASED DESIGN

Vasilis Capoyleas
Xiangping Chen
Christoph M. Hoffmann

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD-TR-94-011
November 1994
(Revised 2/95)

Generic Naming in Generative, Constraint-Based Design*

Vasilis Capoyleas Xiangping Chen Christoph M. Hoffmann
Department of Computer Science, Purdue University
West Lafayette, IN 47907-1398

Report CSD-TR-94-011
November 1994
Revised, February 1995

Abstract

In generative, constraint-based design, users graphically select shape elements of design instances in order to specify shape operations that have generic intent. We discuss techniques for naming algorithmically and generically the identified geometric instance, and report on our experience with implementing these techniques. In a companion paper, we give algorithms for matching the names when editing designs.

1 Introduction

A new generation of CAD systems has become available in which geometric and dimensional constraints can be defined and solved. Such systems allow the user to instantiate generically defined models from user-supplied dimension values and constraints, and permit easy editing of the design in order to derive *design variants*, for example by changing dimension values. But the automatic generation of designs from constraints causes many semantic problems, [7, 8, 13]. Specifically, the user executes some design gestures that graphically interact

*Supported in part by ONR contract N00014-90-J-1599, by NSF Grant CDA 92-23502, and by NSF Grant ECD 88-03017.

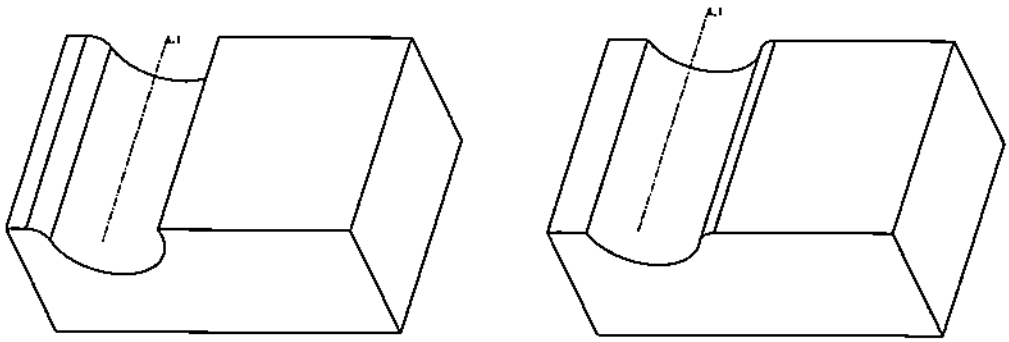

Figure 1: Persistent naming error in design variant.

with the currently instantiated geometric design. The system is then expected to abstract those gestures and produce from them a generic description of the geometric elements the user referred to.

For obvious reasons, design interfaces are centered around visual design gestures. While visual design gestures interact with images that are representations of *instances* of a generic model, their intent is to modify the *generic* model itself. However, geometric elements identified visually need not correspond explicitly to design gestures ever made, and to capture design intent on the basis of the explicit gestures is a profound challenge. Consider for example Figure 1 showing a design and its variant prepared with Pro/ENGINEER Version 12.0. The shape shown there has been designed in three steps:

1. A block has been created by drawing a rectangular profile and extruding it by a specified depth.

2. A round slot has been cut by extruding a circular profile across the block.

3. An edge round of constant radius has been defined by visually identifying one edge and specifying a radius for the round.

After so defining the model on the left, the dimension value locating the center of the slot profile is changed. On regeneration, the edge round "jumps" to a different edge.

The basic problem is to identify the correct edge so as to record and retrieve it to/from design history persistently. The edge that was identified on the left corresponds implicitly to the intersection of the slot surface with the top of the block. Both surfaces, in turn, are the trajectory of explicitly drawn geometric elements, and so can be represented generically. However, the edge on the right, to which the blend jumps after relocating the slot center, is also the intersection of the two surfaces. No generic way is immediately evident that could be executed automatically and distinguish algorithmically between the two edges. Since regeneration of the slot must precede the existence of the two

edges in the new shape instance, any annotation of the previous model instance in which the edge was selected has been lost. The example was created using Pro/ENGINEER Version 12, and demonstrates that the naming schema used by Pro/ENGINEER has a problem. The nature of their naming schema and an analysis of its limitations cannot be given because the implementation of Pro/ENGINEER is guarded as proprietary information.

Prior work [7, 8, 13] has articulated the problem clearly. Others have hinted at the issue without putting it into sharp focus. For instance, [12] observes that to record a picked vertex in the log file, the mouse position is inadequate because, if the log file is edited later, the position may become meaningless. The paper proposes a naming schema derived from an underlying CSG expression, but does not give precise rules for how these expressions are to be constructed.

In [15], Várady proposes a syntax for defining topological expressions that serves to identify geometric elements and features in parts and assemblies. However, the method uses directives such as *left*, *top*, *front*, etc., which must rely on coordinate frames that are understood and remain invariant. This aspect makes the method unsuitable for constraint-based variant generation in which no explicit coordinate system should be assumed.

Solano and Brunet propose in [14] a CSG-based design representation with constraint-based object instantiation. However, as their system description does not include operations that need to reference geometric elements which would arise from feature collisions, their description of the underlying language has no linguistic elements that address the persistent naming problem in the needed generality.

Kripac [11] gives a naming scheme that is characterized by the immediate adjacency of an entity, its orientation in the boundary representation, and a graph recording the history of face merging and splitting. The naming scheme is used for dimensional changes, and the matching algorithms are sophisticated. However, the presentation omits a number of details, most notably under which circumstances exact matches are required and when approximate matches may be acceptable.

In the collection [10] a number of commercial CAD vendors have published papers that acknowledge the importance of the naming problem but give no details on the particular algorithms their respective systems implement. Existing problems such as the one illustrated in Figure 1 urge us to study a naming schema formally in order to build a feature-based design system[9].

This generic naming problem is especially important for our generative, constraint-based design system, Erep, as proposed in [9]. In that system, we explicitly record design gestures or intent into an unevaluated, modeler-independent representation which can be automatically instantiated into an evaluated, modeler-dependent representation with a design compiler [4] following a sound feature attachment semantics [3]. To achieve modeler independence

for the recorded design history, we substitute all geometric references, such as the edge to be rounded in Figure 1, by generic, persistent names. During design compilation, these generic names are first persistently mapped back to geometric references in the boundary representation (Brep) a solid modeler. When the design is changed, a new variant can be constructed based on the generic description and on the history of the previous variant. This involves matching generic names that may not map unambiguously to Brep entities in the new design variant. A companion paper [2] describes in detail the matching algorithms for persistent names when editing design and how to update the unevaluated design representation.

In this paper we define three naming mechanisms to record geometric elements referenced by feature or constraint descriptions. Neither of these mechanisms is complete. We show in Section 5 how to combine them into a comprehensive naming schema.

Obviously, a good generic naming mechanism is important for regenerating constraint-driven, feature-based design. However, the significance of a robust naming schema extends beyond the purely geometric problem of automatically regenerating a design variant correctly. In particular, tolerancing, annotating parts of a design with boundary conditions for engineering analysis, and recording surface finish, are some of the many activities arising in manufacturing applications that also depend on a generic naming schema.

2 Semantics of Model Generation

As pointed out in [7], the semantics of feature-based generative modeling must be specified unambiguously, and implementing the semantics depends on a well-founded naming schema that is the subject of this paper. We review briefly the semantics of feature operations detailed in [3]. We assume that the solution of a 2D sketch, based on specified geometric constraints, is the one the user intended. In view of the points discussed in [1], this assumption is not a triviality, and we make it here only to limit the scope of the paper.

Following [9], we distinguish between generated features, datum features and modifying features. Generated features under consideration are only sweeps of extrusion and revolution, of cross sections that enclose a connected, finite area. The cross section is composed of line segments and arcs. The datum features we consider are datum points, axes and planes; the modifying features are blends and chamfers. We exclude for simplicity the operations of mirroring, patterning, and shelling that were discussed in [9].

Generated features create geometry subject to modalities such as blind extrusions, extrusions from a face to another face, and so on. To implement these modalities, we proceed conceptually as follows (see [3] for details):

1. A *proto feature* is created that is the extrusion or revolution of the spec-

ified cross section to an extent sufficient for generating all necessary surfaces. In the case of a blind extrusion or revolution, the proto feature is simply generated as if the feature were the first feature. In the case of more complicated modalities, the proto feature may extend to intersect the boundaries of the bounding box for the existing geometry.

2. A partial Boolean intersection between the proto feature and the existing geometry creates a set of edges and vertices on the surface of the proto feature. These are used to extract from the proto feature's surface all necessary new boundary elements of the feature after attachment.

3. The relevant surface elements are interpolated in the existing geometry and excess surface geometries are cut away.

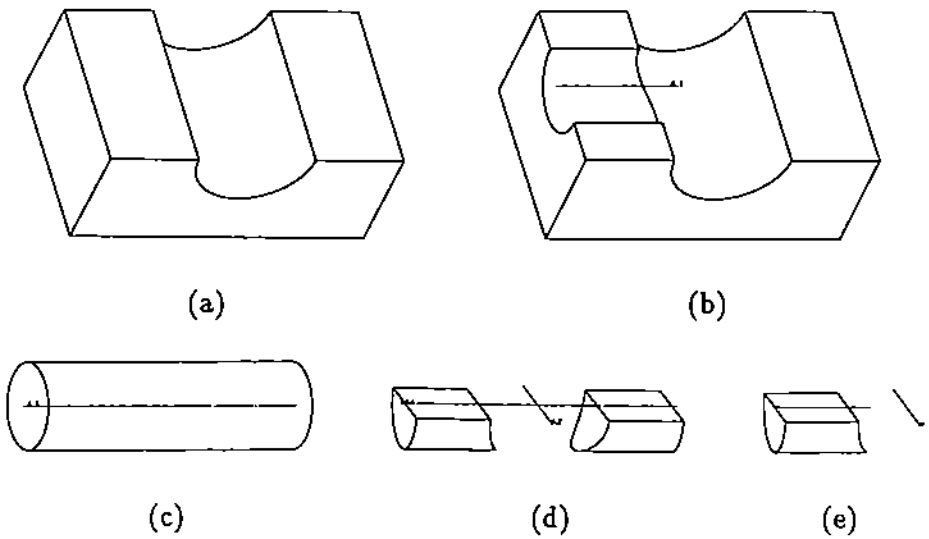

Figure 2: Idea of feature attachment. (a) existing geometry; (b) geometry after the cut is made; (c) proto feature; (d) subdivision of proto feature; (e) needed component of proto feature.

Figure 2 illustrates the process. The existing geometry is shown in Figure 2(a). We want to make a round slot as in Figure 2(b), from the side face to the existing slot. A circle is drawn on the left face. Its extrusion across the extent of the existing geometry is the proto feature, shown in Figure 2(c). Intersection of the proto feature with the existing geometry, shown in Figure 2(d), identifies two potentially relevant components of the proto feature. The cut attributes identify the leftmost component to be the one defining the new feature, shown in Figure 2(e). The desired cut can now be constructed using this component.

Features are constructed sequentially in the order in which the user has defined them. We assume in this paper that the order in which features are defined is not changed, but this restriction can be relaxed with suitable adjustments to the algorithms. The elements referenced for constructing a particular feature are always evaluated with respect to the geometry that exists at the time when the feature is added to the design.

The Brep entities that need to be named persistently fall into two categories. One class of entities corresponds to geometry created explicitly as part of a feature operation. This includes virtually all faces[†] and some edges and vertices. The second class includes entities that comes about through *feature collision*: Such entities are, essentially, edges and vertices in which different feature elements intersect. For example, the intersection edges of a slot and a block are in this category (see Figure 1).

Entities in the first class are uniquely associated with a single feature, whereas entities in the second class are associated with several features and are transient in the sense that, after editing, such entities could disappear or new ones could be created. Some examples of such changes are discussed in [2].

3 Topology-Based Naming

All three feature types require, for their definition, references to existing geometry, and those references must be made generic, so that, upon editing the design variables, the new design variant can be constructed completely automatically. A suitable naming schema for geometric elements directly created by design gestures is not complicated. We describe such a schema first. Then, we concentrate on naming geometric elements that arise implicitly from feature collision, giving a topological naming schema.

3.1 Naming Created Geometric Elements

Generated Feature Elements

Generated features create geometry by sweeping a standardized or sketched cross section. The elements of the cross section can be named as follows: First, $v_1, v_2, \ldots$ will be points of the sketch. In a subsequent extrusion operation, these points will become vertices and edges. Next, $e_1, e_2, \ldots$ will be segments and arcs of the sketch. They become edges and faces in the extrusion. Finally, f will be the area enclosed by the loops of the sketch. It will become two faces of the extrusion. The situation is depicted schematically in Figure 3. We have two instances of each v_i, e_k, and of f, for the front and the back face. Moreover, the

[†]Two planar faces of separate features may overlap and be merged. In that case, there is no unique association.

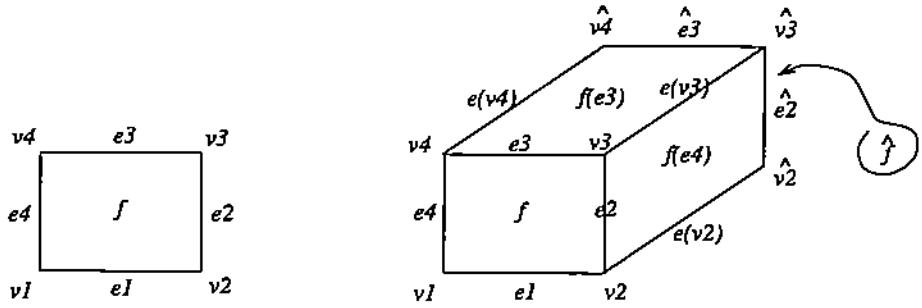

Figure 3: Generic naming in an extrusion: Names assigned by sketcher left, names for extrusion shown right.

side faces are named $f(e_k)$ and the side edges are named $e(v_i)$. The two instances can be ordered by the direction of the sweep and are thereby distinguished from each other.

For revolute sweeps the same naming scheme is adopted, except that a full revolution has neither front nor back face, so that only side faces and edges are named in that case.

Since the names v_i, e_k and f are generated by the sketch, they are made unique in the global context by adding the feature name as qualification. For instance, e_k of feature X is referenced as $X.e_k$.

Merging Named Entities

When generating the first feature, the created shape is the proto feature. Its vertices, edges and faces can be named by this scheme, and the names are unique for valid extrusions and revolutions. Subsequent protrusions, restrictions and cuts first generate a proto feature that is named in the manner defined before. The proto feature is then trimmed against the existing geometry, creating on the surface of the proto feature new edges and vertices. These edges and vertices are given the nonunique names I_e and I_v, respectively, for *intersection edge* and *vertex*. Finally, the boundary elements of the trimmed proto feature are interpolated into the existing geometry as required by the modalities of the feature operation.

When attaching the feature, the vertices, edges and faces of the trimmed proto feature may coincide with existing vertices, edges and faces, and are merged into single logical entities. In this case it is ambiguous how to name the merged geometric elements. We uniformly name the merged entity from the geometry that exists prior to attaching the feature: When two vertices are coincident, the new vertex is named from the existing geometry. When two edges overlap and are merged into one, the new edge is named from the existing geometry. When two faces overlap and are merged, the new face is named from

the existing geometry. In all other cases, the geometric elements of the new feature are logically distinguishable from existing geometry and so keep the name of the proto feature.

Datum and Modifying Features

Datum features are named serially in the order of creation. If the intersection of two datum features is to be referenced, then an explicit feature creation operation will be needed [9] and a name will have been associated with the feature. The same holds for the intersection of a datum with existing geometry.

Chamfers and rounds replace selected edges and vertices with new faces. We label the face replacing edge e or edges l_e with $c(e)$ or $c(l_e)$ in the case of a chamfer and with $r(e)$ or $r(l_e)$ in the case of a round. The bounding edges and vertices are named exactly in the same way as intersection edges and intersection vertices.

3.2 Topological Context Computation

A geometric model instance named in the manner described has a number of elements that have the same name. These include not only elements with the collective names I_v, I_e, $r(l_e)$ and so on, but also edges and faces that have been subdivided by feature collisions. To distinguish between them, we determine their *topological context.*

The adjacencies of the geometric elements form a *context graph.* By convention, a face is adjacent to edges, an edge adjacent to vertices and faces, and a vertex to edges. In many cases the context graph is essentially the adjacency graph of the Brep;[‡] see, e.g., [6]. But, in some cases, several faces of the Brep are grouped into a single logical face and then the context graph is a reduction of the adjacency graph. Note that the context graph is a labeled graph. For example, consider the design of Figure 4. The context graph is shown in Figure 5. The types of the graph nodes — vertex, edge or face — are indicated by the enclosure: a face node has a rectangle, an edge node a circle enclosing it. Vertex nodes are not enclosed.

We define the *immediate context* of a graph node to be its *star*, i.e., the nodes immediately adjacent. For example, the immediate context of f consists of the six edge nodes labeled e_1, e_2, e_3, e_5, e_3, and e_4. Note that e_3 occurs twice. There are two edges of the object of Figure 4 labeled I_e. Their immediate context consists of the faces $f(e_3)$ and $f(e_5)$ and two vertices labeled I_v. It is clear that the edges cannot be distinguished by their immediate context. However, the immediate context of the two adjacent faces differs. One of the $f(e_3)$ faces is adjacent to $e(v_4)$, whereas the other one is adjacent to $e(v_3)$. Consequently,

[‡]Face to vertex adjacency is ignored.

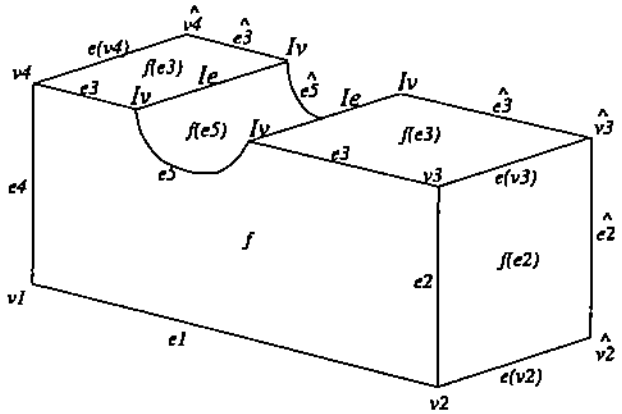

Figure 4: Labeling of object with two features. For simplicity, the names omit the feature qualifier.

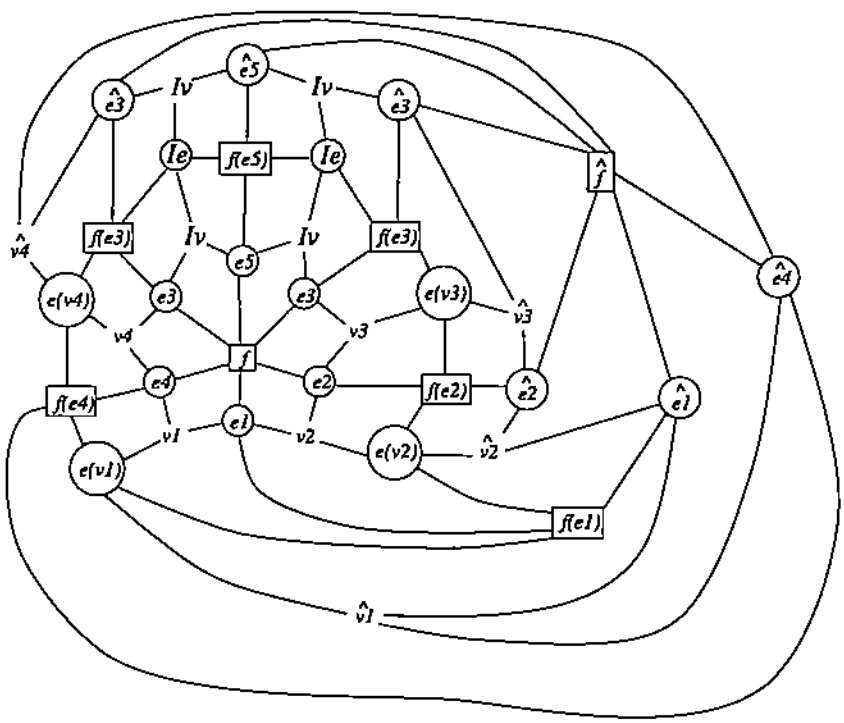

Figure 5: Context graph for object of Figure 4. Faces indicated by boxes, edges by circles.

the two occurrences of $f(e_3)$ can be distinguished, and therefore also the two I_e edges.

We compute the extended context of a geometric element by a breadth-first graph search. The algorithm has the following structure:

1. The extended context of level 1 is the immediate context.
2. The extended context of level $k+1$ is the extended context of level k plus those elements of the immediate context of any node at level k that are not yet in the extended context.

After determining the extended context to some depth, we will be able to differentiate some of the geometric elements of the same type that have the same name. In particular, all elements of the example in Figure 4 can be distinguished by contexts of depth 2. The algorithm is:

Full Topological Context Description
Input: Context graph G and a node r of the graph; depth d.
Output: Context description of the node.
Method:

1. Let R be the set of nodes that have the same label as r.
2. If R contains only r then stop; r is unique without context.
3. Otherwise, for every v in R do the following:
4. By breadth-first search, compute the subgraph reachable from v in up to d steps.
5. Label each subgraph node u by its distance $d_v(u)$ from v.
6. Partition the nodes in the subgraph into equivalence classes, where two nodes are equivalent if they have the same label and the same distance from r.
7. The description of a node v in R is the class structure so computed.

Note that a node can be distinguished from r if it has a different equivalence class structure. The context description can be textually encoded as follows: Each equivalence class is represented by a triple $[d, l, n]$, where d is the distance from r, l is the label, and n is the number of nodes in the class. For instance, the two edges labeled I_e in Figure 4 have the following descriptions:

Left edge:
$[0, I_e, 1]$,
$[1, I_v, 2]$, $[1, f(e_3), 1]$, $[1, f(e_5), 1]$,
$[2, e(v_4), 1]$, $[2, e_3, 1]$, $[2, e_5, 1]$, $[2, I_e, 1]$, $[2, \hat{e}_5, 1]$, $[2, \hat{e}_3, 1]$
Right edge:
$[0, I_e, 1]$,
$[1, I_v, 2]$, $[1, f(e_3), 1]$, $[1, f(e_5), 1]$,

$$[2, e(v_3), 1], [2, e_3, 1], [2, e_5, 1], [2, I_e, 1], [2, \hat{e}_5, 1], [2, \hat{e}_3, 1]$$

The two edges can be distinguished because of the two different classes $[2, e(v_3), 1]$ and $[2, e(v_4), 1]$.

Rather than storing the full context description, we could delete from it those classes that are not distinguishing and that do not contain nodes that are in the context graph on the path from r to a node belonging to a distinguishing class. We call this the *reduced context*. For example, the reduced context of the two edges is

Left edge:	Right edge:
$[0, I_e, 1]$,	$[0, I_e, 1]$,
$[1, f(e_3), 1]$,	$[1, f(e_3), 1]$,
$[2, e(v_4), 1]$	$[2, e(v_3), 1]$

Note that the reduced context sacrifices some resolution when matching in design variants.

3.3 Remarks

Since features are evaluated in a fixed sequence, the context of a referenced element is evaluated for the geometry that exists at the time the new feature is created. Thus, the context graph of feature k is based on the geometry created by features 1 through k only. Later features may well obliterate geometric elements used to construct earlier features, yet they can be referenced at the earlier times.

The algorithm for distinguishing geometric elements by their topological context is essentially a *spectral graph isomorphism* algorithm (see [5]). The context computation determines a *spectrum*, and the reduced context is a *certificate*. Graph nodes with distinct certificates can then be distinguished.

It is well known that spectral algorithms do not solve the graph isomorphism problem and that they fail on highly regular graphs; [5]. It is easy to construct an example in which no topological context can distinguish some geometric features: Consider Figure 6. A toroidal hole has been cut into a block. The two circular edges bounding the cut cannot be distinguished, because the context graph has a structural symmetry that exchanges the two edges. For this reason we need other mechanisms such as that described in Section 4.

4 Orientation Information

The topological context information discussed in the previous section essentially encodes the adjacency structure of feature collision. In addition, we will consider orientation information that is based on the way geometry has been created and

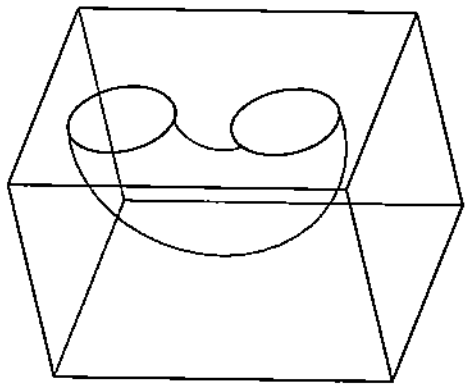

Figure 6: Block with toroidal hole. The edges of the cut cannot be distinguished from the context graph.

the way features lie with respect to each other geometrically. The directional information derived from the feature creation process is called *feature orientation*, and the orientation information locally derived from the geometry of the boundary elements is called *local orientation*.

4.1 Local Orientation

Local orientation information is based on the direction in which an adjacent face uses an edge, and on the face orientation in the solid boundary. By convention, a face is oriented so that the surface normal locally points to the solid exterior, and an edge is used in a direction such that locally, when viewed from the exterior of the solid, the face's interior is to the left of the edge; e.g., [6].

Distinguishing Edges with Local Orientation

An edge use by an adjacent face orders the vertices of the edge, assuming that the edge is not closed. The vertices can be designated explicitly by their names, so that the edge use orientation can be encoded by a triple consisting of the face name and the names of the two vertices in order. In Figure 7 (left), the edge e is used by f_1 from v_1 to v_2, and by f_2 from v_2 to v_1. The edge use can be encoded by the triples

$$L_e = [f_1, v_1, v_2] \qquad \text{or} \qquad L_e = [f_2, v_2, v_1]$$

In most cases, we may use equivalently terminating faces, i.e.,

$$L_e = [f_1, f_3, f_4] \qquad \text{or} \qquad L_e = [f_2, f_4, f_3]$$

Now consider Figure 7 (right). Edge e_1 is used by the adjacent face f_2 in the

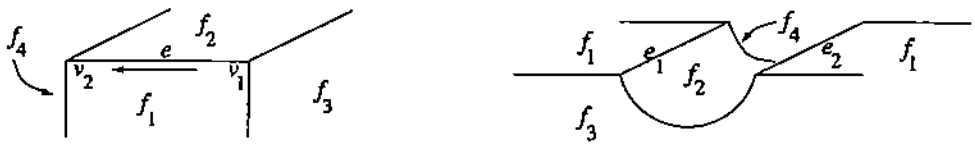

Figure 7: Computation of local orientation of an edge.

direction f_4 to f_3, whereas e_2 is used in the opposite direction:

$$L_{e_1} = [f_2, f_4, f_3] \qquad L_{e_2} = [f_2, f_3, f_4]$$

Therefore, e_1 and e_2 can be distinguished by the local edge orientation with respect to f_2. Of course, the edges can also be distinguished by the local edge orientation with respect to f_1.

Note that the orientation of closed edges cannot be recorded by triples. For those edges an orientation comparison is needed based on comparing the edge use direction with an intrinsic orientation induced by the feature creation, as explained later. See also Figure 12 below. Local orientation cannot distinguish the intersection edges of Figure 6.

Distinguishing Vertices with Local Orientation

Consider the vertex v shown in Figure 8 (left). Generically, a manifold vertex is adjacent to three faces which are the direct context of the vertex. Ordering the faces cyclically, with $\mathbf{n}_1$, $\mathbf{n}_2$, and $\mathbf{n}_3$ the surface normals at the vertex, we may define the *local orientation at* v as the sign of the mixed product of the normals in that order:

$$S_v = \mathrm{sgn}((\mathbf{n}_1 \times \mathbf{n}_2)\mathbf{n}_3)$$

Then S_v is positive if the three vectors form a right-handed coordinate system, and negative if they form a left-handed one. If two surfaces meet with tangent continuity at the vertex, then S_v is zero. The quantity S_v essentially expresses

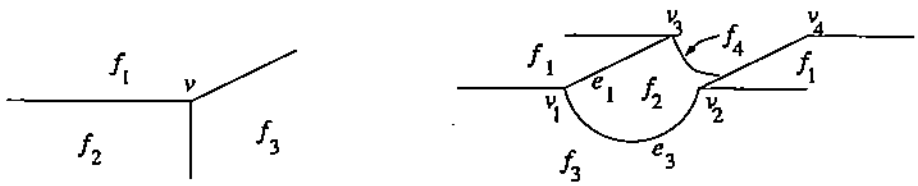

Figure 8: Computation of local orientation of a vertex.

the local geometry at a vertex where three adjacent faces meet transversally. This schema can be extended to vertices with more than three adjacent faces. However, it is not clear whether this geometric information should be expected

to remain invariant under a broad class of feature editing operations, so we do not use it. Instead, we record whether a vertex is a manifold or a nonmanifold vertex, and if it is a manifold vertex, we describe it locally by a list of incident faces, ordered cyclically counter-clockwise, as seen from the vertex exterior. Thus, the four vertices in Figure 8 (right) have the descriptions

$$L_{v_1} = [f_1, f_3, f_2] \qquad L_{v_2} = [f_1, f_2, f_3] \qquad L_{v_3} = [f_1, f_2, f_4] \qquad L_{v_4} = [f_1, f_4, f_2]$$

They can be distinguished if $f_3 \neq f_4$. However, if $f_3 = f_4$, then $L_{v_1} = L_{v_4}$ and $L_{v_2} = L_{v_3}$. An example of this situation is shown in Figure 9. To distinguish these four vertices, other information must be used as explained later.

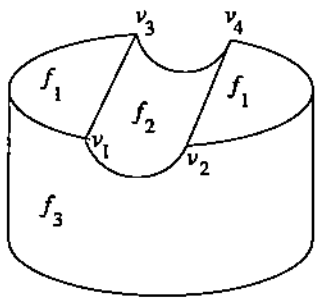

Figure 9: Local orientation is insufficient to identify the four vertices with same direct context.

4.2 Feature Orientation

In the example of Figure 6, the cut was created by revolving a circular profile. The direction of revolution defines an orientation of the surface that can be derived unambiguously from the design gestures and can be thought of as a vector field that is tangent to the surface and perpendicular to the axis of rotation. See also Figure 10. We call this orientation the *feature orientation* since it is defined by the feature creation, and not by the geometric characteristics of the instantiated feature.

Feature orientation information can be defined based on the direction of extrusion or the spin of revolution. It amounts essentially to observing the (local) motion of the sweep of the 2D cross section as the 3D proto feature is created. In the case of extrusions, the extrusion direction is the feature orientation. In the case of revolution, the feature orientation at any point is defined by the instantaneous rotation vector at that point. At singular points, where the rotation axis intersects the revolved surface, the axis orientation can be used instead. Note that the axis orientation and the direction of the revolution are by convention related by a right-hand rule as illustrated in Figure 10.

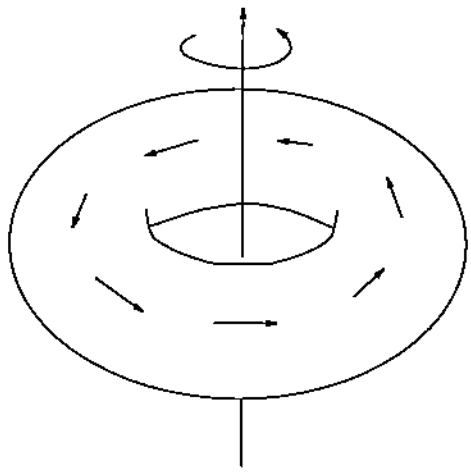

Figure 10: Orientation of torus surface created by revolving a circular profile.

Distinguishing Edges with Feature Orientation

Feature orientation induces a direction of adjacent edges. For edges such as the straight boundaries of the slot in Figure 9, the orientation directs them from v_1 to v_3 and from v_2 to v_4, assuming the slot was an extrusion of a circular profile from the front to the back. Loosely speaking, one could call such an edge *aligned* with the orientation.

More precisely, consider an extrusion or a partial revolution, as seen in Figure 11. Its topology consists of cylinders that could be pinched along edges that are the trajectory of nonmanifold vertices of the swept profile. In the case of such nonmanifold topology, we cut the profile at nonmanifold vertices and consider the resulting cylinders separately. Three types of curves must be considered:

1. Curves that cut the cylinder such that we can unroll the surface into a rectangle.

2. Curves that cut the cylinder into two separate cylinders.

3. Curves that separate the cylinder into a cylinder with a hole and the hole

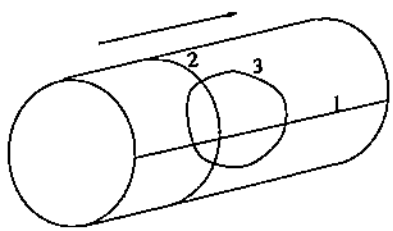

Figure 11: Curve types on a cylindrical topology

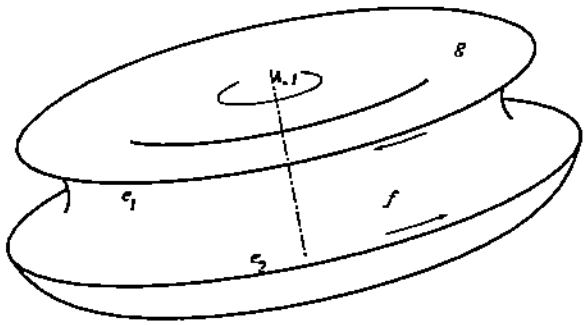

Figure 12: Edges e_1 and e_2 are not distinguished by feature orientation alone

interior which is essentially a disk.

Type 1 is an aligned edge, and can be directed by the orientation of the cylinder surface. See Figure 9 for an example. Type 2 is an edge where the adjacent cylindrical surface, used in the feature defined, either has a source or a sink at the edge. See Figure 6 for an example. Note that edges of Type 2 must occur in pairs, each source edge having a corresponding sink edge. The correspondence is established by the direction of the sweep. Type 3 is an edge where the adjacent surface has both a source and a sink on the edge. If the axis of the toroidal cut were lifted in Figure 6, the two delimiting edges would merge into a single edge of type 3. The same types can be defined analogously for full revolutions with spherical or toroidal topologies.

Using feature orientation information, we can distinguish the two circular edges of the Figure 6. Both are type 2, and at one of them the orientation of the toroidal face has a source, whereas at the other one it has a sink. If more than two such edges are present, they can be sorted by the direction of the feature orientation and distinguished positionally in the order in which they lie with respect to the feature orientation. Note that for toroidal topologies the ordering is cyclical, whereas for partial revolutions or extrusions a total order is imposed.

In Figure 12, edges e_1 and e_2 are both of type 1 and are oriented in the same way by the direction of revolution. They can only be distinguished by observing that face f uses e_2 in the feature orientation direction, whereas e_1 is used by f in the opposite direction.

Two closed edges of type 3 cannot be distinguished using feature orientation except in rather special cases where a cutting plane can be defined that induces a separating curve of type 2. However, type 3 edges arise from feature collisions where the other, intersecting surface usually allows differentiation.

The feature orientation of an edge is recorded by a triple $[F, f, c]$. F is the feature that defines the feature orientation; f is a face of the feature that is adjacent to the edge. If the edge is of type 1, then $c = \pm$ according to whether

the edge is used by f in the direction of the feature orientation or in the opposite orientation. If the edge is of type 2, then $c = \text{in/out}$, where 'in' means that the feature orientation flows from the face exterior across the edge into the face interior, whereas for 'out' the orientation flows from the interior of f to the exterior. If the edge is of type 3, then $c = 0$ and no information is obtained.

The feature orientation information for edges of type 2 can be refined further by ordering all edges of type 2 in the direction of the feature orientation. For extrusions, this is a total ordering, but for revolutions with a toroidal topology the ordering is only cyclical. In such a schema, an expression of the form $[F, f, m, n, q]$ could be used, where F and f are as before. Moreover, the edge is the m^{th} of n edges of type 2, and q designates whether the ordering is cyclical or total.

Distinguishing Vertices with Feature Orientation

Feature orientation does not add a separate naming mechanism for vertices. However, the vertex v may be incident to an edge e that has a feature orientation. If e is of type 1, we observe whether e begins or ends at v. The name of the vertex is then $[e, +]$ or $[e, -]$.

For edges of type 2, this approach does not apply, but we can derive similar information by a slightly more complicated computation. Instead of using the edge tangent, we can use the feature direction vector at the vertex. For edges of type 1 we have conceptually observed whether the inner product of this vector with the edge tangent is positive or negative. Instead of using the edge tangent, we use the normal of an incident face not adjacent to the edge. Consequently, the vertex feature orientation encoding can be uniformly expressed by $[F, f', \pm]$, where F is the feature whose orientation we use, f' is an incident face that is not adjacent to an intersection edge of the feature F, and $\pm$ records the sign of the inner product. Note that f' must be selected such that the inner product does not vanish.

For example, consider vertex v_1 of Figure 9 right. Assuming that the round slot is generated as feature F with an extrusion from the front to the rear, v_1 is labeled $[F, f_3, -]$.

4.3 Combining Local and Feature Orientation

Intersection features are named by a combination of local and global orientations, because both types of orientation provide only limited resolution. In addition, the immediate topological context is added. The combined information is recorded in a formal expression has the following properties:

1. The expression is *canonical*. If one entity name uses one face, another entity uses a different face when in both the same face could have been

used, then an ambiguity could go unnoticed. We avoid this by exploiting the serial creation of features and by using set expressions.

2. The expressions are not circular. If a name for edge e uses a vertex v, then the name of v does not use edge e. We avoid this by using only face names in formal expressions.

3. Where reasonable, we use the generated names explained in Section 3.1.

The naming expressions that combine this information are composed of three subexpressions, the ordered, immediate context, the local orientation information, and the feature information. Naming expressions may be nested.

Naming Intersection Edges

The name is an expression of the form

$$E_e = [C_e, L_e, F_e]$$

where $C_e = [f_1, f_2, ...]$ are the names of all faces adjacent to the edge, cyclically ordered. There is no direction in which the cycle is understood. For example, $[f_1, f_2, f_3, f_4]$ and $[f_4, f_3, f_2, f_1]$ are not distinguished. This is because not every edge has an intrinsic direction that can be defined generically. If edges have an intrinsic orientation derived from the feature orientation, then the cycle direction can be inferred from the feature orientation subexpression.

L_e is the local orientation, which has the form $[V, W, s]$ if the edge is open. For closed edges we have $L_e = []$. Here V and W identify the two vertices bounding the edge. If $s = +1$ then f_1 uses the edge from V to W, and if $s = -1$, then f_1 uses the edge from W to V. Since the F_i are cyclically ordered, it follows that the faces $f_1, f_2, ...$ use the edge in the orientation $[s, -s, s, -s, ...]$. If the vertex identified by V is a created vertex, then V is the name generated for that vertex as described in Section 3.1. Otherwise, V is an expression identifying an intersection vertex as described later.

F_e is the feature orientation of the edge. It has the form $F_e = [F, H]$ where F is a feature and H is a set of feature orientation expressions. F is the latest feature one of whose faces is adjacent to the edge. Let $h_1, h_2, ...$ be all faces belonging to F that are adjacent to the edge. Then $H = \{[h_1, c_1], [h_2, c_2], ...\}$, where $[F, h_i, c_i]$ is a feature orientation triple of the edge.

Naming Intersection Vertices

The name of a vertex is an expression

$$E_v = [C_v, L_v, F_v]$$

Here $C_v = \{f_1, f_2, f_3, ...\}$ is the set of incident faces. It is not ordered if v is a nonmanifold vertex. For manifold vertices, there is a cyclical ordering of the f_i that is counterclockwise as explained before. If $L_v = M$, then the vertex is manifold, if $L_v = N$, then the vertex is nonmanifold.

The expression $F_v = [F, H]$ is the feature orientation information consisting of a feature name F and a set H of orientation informations. For every face f incident to the vertex but not part of the feature F, H contains the expression $[f, s]$, where $[F, f, s]$ is the feature orientation as explained before. Note that $s = 0$ is allowed.

5 Implementation

Both the context graph and the orientation information eliminate ambiguities, but neither technique is complete. Therefore, we have implemented a combination of the two methods. In particular, intersection elements are first encoded using the formal expressions derived in Section 4.3. These expressions include the immediate context. We add to this information the reduced topological context, to some predefined maximum depth. Because both the orientation information and reduced context is canonically organized, the derived naming expressions remain canonical.

We note that ambiguities are not always resolvable by our method. Structural symmetries can be constructed that foil our naming schema. However, such highly symmetric situations seem to occur rather rarely in practice. When they arise, moreover, choosing arbitrarily between several entities with the same name is often acceptable. In particular, when a straight edge is referenced as a boundary of a distance dimension, any edge that has the same underlying line equation meets this reference need. In such cases, certain ambiguities are allowed.

Remaining ambiguities for different classes of entities, if existing, are uniformly handled as follows. If the operation is a modifying feature, such as a chamfer or blend, then the operation is applied to all geometric entities with the same name. If the operation requires exact identification in a dimensioning schema, then the user has to change the dimensioning schema.

5.1 Vertices

Vertices that are not the result of feature collision are named as explained in Section 3.1. Evidently, such names are unique. Intersecting vertices, on the other hand, are named first with the expression E_v explained before. If a vertex is not uniquely named by the expression, then the reduced context is computed.

5.2 Edges

Edges are also classified as intersecting edges and non-intersecting edges. Intersecting edges are first named with the expression E_e explained in Section 4.3. If ambiguities remain for identifying an edge, the reduced context is computed.

A nonintersecting edge is named similarly, except that the immediate context of a non-intersecting edge is replaced by the inherited edge name of the parent edge in the proto feature.

5.3 Faces

All faces are generated as faces of proto features or modifying features, or are a subdivision of such faces. Furthermore, different faces may be merged if they are on the same underlying geometric surface and overlap. A face name is then an expression that contains a part H_0 describing the origin of the face. If the face is from a proto feature, H_0 is one of the following

(`start_f feature.f2D`),(`end_f feature.f2D`),(`side_f feature.f2D`)

If the face is from a chamfer, then $H_0 = c(l_e)$, where l_e is a description of the edges chamfered. If the face is from a round, then $H_0 = r(l_e)$. When faces are merged, then this name is inherited from the face of the earliest feature involved in the merge. In addition, the face name contains an expression naming every adjacent edge of the face.

6 Summary

We have described topological and geometric mechanisms that name generically geometric elements that were not generically named by the user. The naming algorithms are complemented by algorithms for editing generic design, and those algorithms are described in a companion paper [2]. Together, this work constitutes a naming schema that in our experiments has proven to be robust and reliable.

References

[1] W. Bouma, I. Fudos, C. Hoffmann, J. Cai, and R. Paige. A geometric constraint solver. *Computer Aided Design*, page to appear, 1995.

[2] X. Chen and C. Hoffmann. Editing feature based design. Technical Report CSD 94-067, Purdue University, Computer Science, 1994.

[3] X. Chen and C. Hoffmann. Towards feature attachment. *Computer Aided Design*, page to appear, 1994.

[4] X. Chen and C. Hoffmann. Design compilation for feature-based and constraint-based CAD. In *Proc 3rd ACM Symp on Solid Modeling*, 1995.

[5] C. M. Hoffmann. *Group Theoretic Algorithms and Graph Isomorphism.* Springer Verlag, 1982. Lecture Notes in Comp. Sci. 136.

[6] C. M. Hoffmann. *Geometric and Solid Modeling.* Morgan Kaufmann, San Mateo, Cal., 1989.

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

[8] C. M. Hoffmann. Semantic problems in generative, constraint-based design. In *Parametric and Variational Design*, pages 37–46. Teubner Verlag, 1994.

[9] 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 and Product Modeling*, pages 129–164. North Holland, 1993.

[10] J. Hoschek and W. Dankwort. *Parametric and Variational Design.* Teubner Verlag, 1994.

[11] 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.

[12] J.R. Rossignac, P. Borrel, and L.R. Nackman. Interactive design with sequences of parameterized transformations. Technical Report RC 13740, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, New York, 1988.

[13] V. Shapiro, H. Voelcker, and D. Vossler. Boundary to CSG conversion: current status and open issues. IFIP WG5.2 Conference on Geometric Modeling for Product Realization, Rensselaerville, 1992.

[14] L. Solano and P. Brunet. A system for constructive constraint-based modeling. In B. Falcidieno and T. Kunii, editors, *Modeling in Computer Graphics*, pages 61–84. Springer Verlag, 1993.

[15] T. Várady, B. Gaál, and G. Jared. Identifying features in solid modelling. *Computers in Industry*, 14:43–50, 1990.