.

One-Dimensional Selections for Feature-Based Data Exchange

Ari Rappoport[†] and Steven Spitz[‡] and Michal Etzion[§]

Abstract

In the parametric feature based design paradigm, most features possess arguments that are subsets of the boundary of the current model, subsets defined interactively by user selection of boundary entities. Any system for feature-based data exchange (FBDE) must support the exchange of such selections. In this paper we describe in detail an algorithm for supporting one-dimensional selections (sets of edges and curves) for FBDE. The algorithm is applicable to a wide class of FBDE architectures, including the Universal Product Representation (UPR) and the STEP parametrics specification.

Categories and Subject Descriptors (according to ACM CCS): D.2.12 [Interoperability]: data mapping; I.3.5 [Computational Geometry and Object Modeling]: Breps, CSG, solid, and object representations, geometric languages and systems; I.3.6 [Methodology and Techniques]: graphics data structures and data types, languages, standards;

1. Introduction

Data exchange is a fundamentally important operation in solid modeling. From a theoretical perspective, it forces us to clarify the essential aspects of modeling paradigms by considering general system-independent concepts and transformations between their differing concrete implementations. From a commercial perspective, data exchange is still the main approach today for implementing engineering collaboration. There are several other collaboration technologies and methods, but data exchange is still the dominant one.

All modern CAD systems utilize the parametric feature-based modeling paradigm. They all support direct surface design, but the main method through which parts are designed is the feature-based one. When fine control over surfaces is needed, as in industrial design applications, those surfaces are usually combined as features within the parametric feature history. It is thus extremely desirable to address the problem of feature-based data exchange (FBDE).

Today there are two principal documented directions towards FBDE: extensions to STEP, and our UPR architecture. Parametric extensions to STEP (see an overview of the specification in [Pratt04], a description of a small scale prototype system in [Mun03], and the influential EREP project [Hoffmann93]) define a set of procedural commands (widely referred to as *features*, which is the terminology we use in this paper) and mechanisms for their representation and semantics in CAD systems. The proposed STEP parametrics specification does not address the issue of differing feature semantics between CAD systems, leaving it to implementors of STEP file import software.

The Universal Product Representation (UPR) architecture [Rappoport03, Spitz04] takes a radically different approach, using inherent mechanisms for dynamic feature verification and feature rewrite to handle differing semantics and CAD implementation problems. A complete UPR system has been implemented and is being used successfully in real projects.

The two approaches are radically different, but they do share several specific algorithmic and representational problems. One of these problems is how to address features whose arguments include parts of the boundary representation (Brep) of the model (as it exists before the invocation of the feature). More specifically, the problem is how to address user selection of Brep entities. In the context of how CAD systems address this issue, the problem is well-known and is usually referred to as the persistent naming problem [Kripac97, Capoyleas96]. In the context of feature-based data exchange, however, a full persistent naming solution is not essential, as will be shown here.

In this paper we present an algorithm for supporting user selections of one-dimensional Brep entities for feature-based data exchange. By *one-dimensional Brep entities* we mean selection of an edge or curve or a set of edges or curves. The twin problem of selection of two-dimensional Brep entities (faces and sets of faces) will be discussed elsewhere.

As far as we know, the present paper is the first one that identifies this as a non-trivial problem and offers a solution. The STEP specification only specifies the data that resides in the file format and does not address the import algorithm, which is the challenging

[†] The Hebrew University of Jerusalem and PROFICIENCY Ltd.
[‡] PROFICIENCY Inc.
[§] PROFICIENCY Ltd.

part, while neither the papers describing the KAIST project (e.g. [Mun03]) nor the EREP papers [Hoffmann93] describe a solution to the problem.

In Section 2 we explain the concept of 1-D selections in detail. Section 3 reviews the UPR architecture, in the context of which our selection algorithm operates. In Section 4 we provide a formal problem statement. Section 5 presents the main result, detailing the export process and import algorithm. In Section 6 we describe our implementation.

2. Selections in Parametric Feature-Based Design

In this section we explain the concept of user selections in parametric feature-based design and why in the context of data exchange there is no need for a full CAD persistent naming solution.

In parametric feature-based design, the model is represented by a directed a-cyclic graph of operations called features (not to be confused with the term feature when used for manufacturing operations in a multiple view architecture [Shah95]). The graph is usually referred to as the *parametric history* of the model. Most features either create new geometry or modify a part's existing geometry (some features only insert or modify meta-data and other model attributes). The aggregate effect of all features up to and including a given feature generates geometry, structured in a boundary representation (Brep). Recall that a Brep contains two components: a graph (topology) of vertices, edges, faces and shells (*Brep entities*) and their interconnectivities, and a geometric manifestation of each of those entities.

What is important for the purposes of this paper is to understand the role played by the Brep in user interaction and model definition. In interactive feature-based design, which is the dominant paradigm today used by virtually all CAD systems, at any point in time the user sees a 3-D graphical view of the current Brep on the screen. The user either defines new features or edits existing features, and the 3-D display is updated to reflect the changes to the geometric representation generated by the feature graph. However, the Brep is not used only for viewing and for decisions regarding the next design operation. Crucially, the Brep is also used for formal definition of the semantics of some of the future features. This is the main difference between the parametric feature-based design paradigm and classical CSG (in addition to the quantitative difference of the number and types of operations, of course).

Every feature has arguments that define its semantics. For a majority of cases, and certainly so for the more useful and commonly used features, Brep entities constitute arguments for features. The way this is done is by letting the user select a set of Brep entities on the 3-D view and define them as arguments to the present feature. There may be several different such arguments for a single feature.

Here are some examples for features whose arguments include user defined selections:

- Round (or fillet): this feature replaces a sharp edge, or more commonly, a set of sharp edges, by smooth surfaces. When the new surfaces are cylindrical the feature is called a *constant radius* round, and there is a *variable radius* version as well. The user selected Brep entities are in this case the edges to be smoothed.
- Extrude until face: the extrude feature is probably the most common of all, taking a parametric 2-D sketch and extruding it to create a 3-D shape. When an extrude is defined to be *until face* the created 3-D shape is trimmed when it is blocked by an existing model face. That face is the user selected Brep entity in this case.
- Draft: this is a complex feature mostly used for plastic injection molding. A set of partial faces is skewed at a specified angle, creating a global modification of prior geometry. In this case, the user needs to select the set of faces, and optionally sketch curves on those faces to define partial faces.

Selections that serve as feature arguments are defined by the user by selecting Brep entities shown in the 3-D model display. In fact, allowing such selections is a primary constraint on the nature of those 3-D displays, as discussed in [Rappoport96]. Some CAD systems allow the user to select only a subset of the feature argument, completing the rest automatically. The main automatic completion method is to add to the entities explicitly selected by the user all entities that are (recursively) adjacent with smooth geometric connectivity (such as G-1 continuity).

CAD systems must represent selected Brep entities in a way that generalizes over the current geometry, because at any point in time the user may modify the parameters of any feature. When this happens, the system plays the feature history again, a process which usually results in a different geometry. A purely static geometric representation of the selections would hence not be valid. This problem is known as the persistent naming problem. To tackle this problem, CAD systems abstract away some of the properties of the selected entities, usually those that are independent of the numerical values in the model and are functions of more intrinsic properties, such as Brep topology, identities of the features (or carrier surfaces) creating Brep entities, qualitative geometric properties (such as convexity), etc. In other words, they represent selections using generic names that are persistent under parameter changes (hence the term). For a general solution to generic naming of Brep entities see [Rappoport97], and for solutions in the context of CAD systems see e.g. [Kripac97, Capoyleas96].

When performing feature-based data exchange, there are no parametric changes. On the contrary, the main goal is to construct a model in the target system that is as similar as possible to the model in the source system. Hence it is possible, at least in principle, to use representational methods that are different in character and are closer to Brep geometry. The approach taken in this paper is such a method, as described in the next sections.

3. Review of the UPR FBDE Solution

The solution we describe in this paper to the 1-D selections problem is applicable to a wide variety of FBDE architectures. However, the specific algorithmic assumptions we make are valid in the context of our UPR architecture. Therefore, in this section we describe it briefly, emphasizing the aspects that are relevant to this paper.

The UPR differs from all other data exchange approaches in that it recognizes that CAD system capabilities are variable, both in terms of functional semantics and in terms of implementation of theoretically equivalent operations. Due to market forces and the richness of the feature-based design paradigm, it is not realistic to dictate an ultimate set of features that are supported by a CAD system. Each CAD system might provide features that are not directly provided by other systems. In addition, due to the semantical complexity of certain features, a feature's implementation in one CAD system might result in geometry that is somewhat different from the geometry that a different system generates from the 'same' feature. That is, the feature is the same at a certain level of abstraction (usually, overall function as perceived by the user), but different at a detailed geometric level of abstraction. Finally, we should expect the geometry generated by features to be different due to the fact that different systems utilize different tolerances for different operations, a phenomenon that plagues ordinary (geometric) data exchange [Qi04].

The UPR is thus explicitly designed to handle the following two cases: (i) a data item explicitly supported by one system and not by another, and (ii) incompatibilities between systems that can be identified only during run-time due to lack of formal specification of implementational differences.

The UPR representation of a feature makes use of two central concepts to address the above: rewrites and verifications. Each feature has an associated set of rewrites, which are intuitively different ways to import that feature into a system that had not succeeded importing it using other means. So instead of dictating a certain fixed representation, the UPR allows an unlimited number of representations that attempt to simulate the semantics of a feature (usually at decreasing levels of abstraction, starting from fully parametric and ending at fully geometric [Spitz04]). In addition to rewrites, each feature stored a set of verification data, used to dynamically identify whether feature import has succeeded or not.

For the present paper, the concept of verification data is most relevant, because we rely on it in some of our algorithms. In particular, when a partial feature history has been verified as having been imported successfully, this serves as an assertion that various situations are not possible, simplifying the design of our selection algorithms. If such a situation does occur, this signals that there is a bug somewhere in the system, in which case the system attempts a graceful recovery. All of those mechanisms are taken care of at the global architectural level and do not form a part of the selection algorithms, which operate at a more detailed level.

Structurally, the UPR is a star architecture, like STEP. Export and import modules are responsible for communication with the source and target CAD systems respectively. This specific choice, which is a natural one to make considering the economics of developing a solution that is meant to support many CAD systems, is independent of the selection problem discussed in this paper. Our selection algorithms are applicable also to FBDE systems that use a direct source to target translation.

4. Problem Statement

In this section we formally define the problem that this paper deals with, including assumptions on the context in which the solution is used. In general, a FBDE system needs to support all types of user selections that serve as feature arguments. However, due to the fact that the actual algorithms we have developed for one-dimensional and two-dimensional selections are rather different from each other, in this paper we focus on 1-D selections only. Our 2-D selections algorithm will be described in a separate paper. Thus, in this paper, our goal is to

- provide support for one-dimensional user selections that serve as feature arguments in feature-based data exchange systems.

We further state that such support is to be implemented as part of a FBDE system that defines features one after the other in the target CAD system. Although at the moment this is the only approach towards FBDE (as exhibited by the UPR architecture and STEP), it is not absolutely inconceivable that other, conceptually different, approaches are possible. The fact that we assume that features are imported one by one implies the following concrete assumption:

- denote by F the feature for which user selections are defined during the import process. Then when the selections are to be specified, the target CAD system holds a boundary representation of the model as defined by the features up to F. Furthermore, this Brep is available for inspection and usage by the selections support algorithm.

This assumption means that selections can be specified to the target system in terms of identifiers of actual present Brep entities.

A consequence of the previous assumption is that the model defined by the features up to F is also assumed to be correct. This is a fine point that needs to be clarified. As noted in Section 3, a realistic feature-based data exchange architecture cannot commit to transfering each and every feature to the target system such that it is both parametric and generates geometry that is identical to that in the source system. To define correctness we thus have two options: either require that the target model generates identical geometry (up to a geometric tolerance, see below) and allow loss of parametricity in some cases, or require the model to be fully parametric and allow the geometry to be different. In this paper we take the first option, because it conforms to the standard definition of the term 'data exchange' and is the one that is required in practice. Note however that it is possible to take the second option, which is an interesting direction research-wise.

As a result of that choice, we can assume that the model defined by the features up to F contains geometry that is similar to the geometry in the source CAD system, but not necessarily a feature history identical to that in the source CAD system. Now, due to the fact that all practically useful geometric representations today use floating point arithmetic, the best we can actually hope for is that the geometries on both sides are identical up to a geometric tolerance. There are many ways to formalize this concept mathematically, and we will not attempt to do so here because it would certainly be outside the scope of this paper (see [Qi04] for a recent relevant discussion). Our algorithms will utilize the following version of this assumption:

- any point that lies on the boundary of the model in the source system lies at a distance no greater than Delta from the boundary of the model in the target system, and vice versa.

This assumption does not have a negative effect on the robustness of our algorithm. The algorithm is designed to recognize cases in which the assumption does not hold, in which case it signals this to the main processor of the UPR architecture for graceful recovery. As explained in Section 3, in the UPR every feature has an associated verification data, whose purpose is to ensure that the assumption does hold.

So far we have only addressed one component of a boundary representation, namely the geometric pointset defined by it. How-

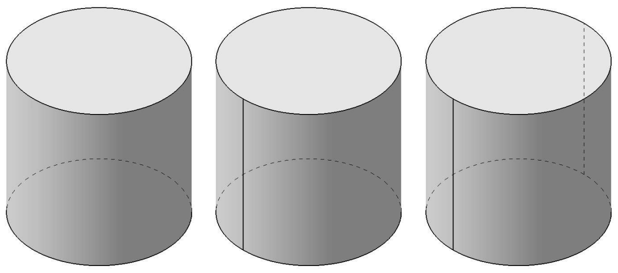

Figure 1: *Brep differences: two (left), three (middle) and six (right) edges to represent a cylinder.*

ever, there is a second component: the boundary graph that represents the vertices, edges and faces and their interconnectivity relationships. Now, different CAD systems may represent this graph in arbitrarily different ways; this is in fact a main difficulty that our algorithm strives to overcome. We do not need to assume any CAD commonality regarding this graph representation (beyond the well-behaved edge overlap assumption given below), because differences not handled by our algorithm are treated by the general rewrites mechanism. Section 5.5 deals with this last issue. A similar comment holds for whether or not our CAD system supports non-manifold geometry or only manifold geometry.

Figures 1 and 2 show how Breps representing the same geometric pointset can differ in their topologies. A cylinder can be represented using two circular edges and three faces (cylindrical face and two disks), or by using an additional edge to split the cylindrical face (usually, this is done in order to establish a well-defined parametrization of the face or to avoid a single face having two bounding unconnected loops), or by using a yet additional edge to split the cylinder (usually, this is done in order to avoid the same edge appearing twice in the same loop.)

Another source of topology differences is shown in Figure 2. A loft feature is defined using cross sections, which are interpolated to a solid. The faces of the solid may include remnants of all of the edges defining the cross sections, or remnants of only some or part (vertices) of them, or be simplified to include non remnants at all.

Because we assume that the geometries of the source and target models are similar, we can in practice safely make another assumption about the correlation between topology and geometry: that when a source edge s intersects a target edge t, it does so in a geometrically well-behaved manner. Formally, we assume that every boundary point in the intersection is either a vertex of s or a vertex of t. This assumption is not strictly needed for our algorithm; it just enables us to implement a specific sub-algorithm (generic overlap test) in a certain efficient way.

In order to simplify the presentation (and because this is the case in all practical systems), we assume that a single edge must be connected. Note that this is only a terminological comment and does not impose any real restriction.

In this paper we deal with the implementation of cross-CAD user defined selections, which is a basic building block for a feature-based data exchange system. We do not deal with higher level operations required in such a system. In particular, we do not address the case where feature semantics on the target system does not allow usage of the type of selections defined in the source system as arguments of that feature. As a simple example, the source CAD system may allow an 'extrude until face' for every type of face, while the target system may only allow planar faces in this role. In the UPR architecture, such semantic differences are handled by the feature rewrite mechanism, which, again, is a higher-level mechanism than the selections mechanism.

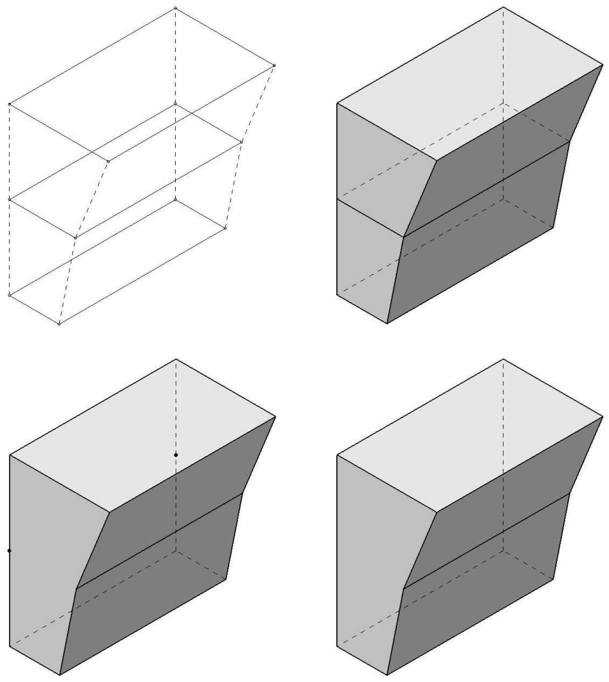

Figure 2: *Brep differences: loft definition using cross sections, and the resulting solids in several CAD systems. Note the vertices in the middles of the vertical edges in the bottom left.*

5. The Algorithm

In this section we present our 1-D selection algorithm. We describe the main idea (5.1), the export process and what data is stored in the UPR (5.2), the main algorithm (5.3) and the main operations it utilizes (5.4), and finally cases that necessitate rewrites (5.5).

5.1. The Main Idea

In Section 2 we had already noted that a generic selection representation in the style of persistent naming solutions is not essential in our case, because the geometries on both sides can be assumed to be identical up to a tolerance. It is thus of conceptual elegance to try to utilize only that static geometry. The general idea is then:

- Export the geometry of the selection into the UPR.
- When selections need to be defined during import, select a subset of the current Brep (in the target system) that covers as much as possible the selections stored in the UPR.
- If there are edges in the UPR selection that cannot be exactly covered by edges in the current Brep, create new edges or split existing edges in the current model so that an exact cover is obtained, or use other feature rewrites to preserve feature semantics as much as possible.

The important thing to note here is that the second step is not trivial. In particular, it is not the case that every selected edge in the source system corresponds to exactly one edge in the target system. The naïve algorithm that loops over the selected edges in the UPR and tries to find for each of them a target system edge identical to it (up to a tolerance) will simply fail. The reason for that is that we cannot make assumptions regarding the topology of the two Breps, only about their geometry, as noted in Section 4. This is a main difficulty that our algorithm attempts to overcome.

5.2. Export to UPR

The goal of the export process is to make sure that all data needed during the import phase is available in the UPR. Since the import algorithm only needs the geometric pointset defining the selection, it is straightforward to represent it in the UPR given that this information is provided by the source CAD system. In principle, any 1-D Brep data structure will be capable of storing the information; due to the philosophy behind the UPR, which is designed to support the union of object types generated by CAD systems, we use the same curve types used in the source CAD system. This means that in principle there should not be any degradation in the quality of the representation, and specifically no loss of tolerance.

To simplify our descriptions, in this paper we will refer to the selection as stored in the UPR as the 'source selection' a term justified by the comments just made.

Because we deal with 1-D selections, there are exactly two types of Brep entities that can participate in each selection: vertices and edges. Our import algorithm can be implemented more efficiently if the adjacency information of those entities is stored explicitly. However, this trivial observation is not a mandatory requirement for the correctness of the algorithm.

One theoretical problem that can be stated is what to do if the source CAD system does not provide access to the Brep entities that comprise the geometry of the selections. It is only a theoretical problem because virtually all modern CAD systems provide detailed access to Breps. In this paper we thus assume that selection geometry is easily accessible through the CAD system API. Standalone identification of selection geometry without using the CAD API is an interesting theoretical computational problem, but one that we will not address in this paper.

In some situations we may export relevant symbolic model data along with the selection geometry. For example, when the model contains several parametric history graphs, the ID of the graph containing each part of the selection can be stored with the selection geometry, in order to make it easier to locate the target image of that body during import or in order to identify the correct body if several bodies overlap geometrically. The IDs of the owning features of each selection part can be used in the same manner (in the terminology of some CAD systems, an owning feature of a Brep entity is the first feature that had caused the entity to be added to the model's Brep). These kinds of techniques are obvious and we will not elaborate on them further in this paper.

5.3. The Import Algorithm

The main part of the algorithm is the process done during import. Recall that the Brep of the model, as generated by the features preceding the feature F whose arguments are our selections, is assumed to be present in the target system. What we seek is a set of entities belonging to that Brep that are an exact cover of the selection geometric pointset. We do not need to know the connectivity structure of the entities to be selected, only the entity set.

The algorithm proceeds one edge at a time (see below). For each edge e in the source selection, we want to find an exact cover made up of target Brep edges. However, in general there is no such exact cover because it may happen that a target edge that is essential for covering e is a proper superset of e. Instead of finding an exact cover, we find any cover, or, if desired, a valid cover for e. A valid cover for a source edge e is a set c of one or more target edges that covers e but does not include points that are outside of the selection pointset. Thus, selecting the edges in a valid cover as arguments of the imported feature F in the target system does not harm the correctness of the selection. For simplicity of exposition, the pseudocode assumes that a valid cover is required; however, as noted in the description of Algo 2 below, it may actually be preferable to find any cover without ensuring its validity.

```
Algo 1: Main loop
  For each edge e in source selection
    Find a valid cover
    If no such set can be found,
      invoke the rewrites procedure (5.5).
  Return the union of all valid covers found (in-
cluding rewrite fixes, if such were done).
```

Note that the same target edge t may participate in more than a single valid cover. For example, suppose that the circular edge of the top of a cylinder was subdivided in the source system into two half-circles e1 and e2. In this case, the target edge t is the valid cover for both e1 and e2. This does not cause any harm to the algorithm, because the final target selection is the union of all valid covers found, which in this case simply amounts to the single edge t. Next we show how we find a valid cover for a source edge e:

```
Algo 2: Finding a cover for a source edge e
  Find a target edge t that overlaps with e
  Cover := t
  While e is not covered and there are remain-
ing target edges to examine
    Add adjacent overlapping edges to t to Cover.
  If e is not covered
    return FAIL
  If validity needs to be verified
    If Cover is a valid cover,
      return Cover + VALID tag.
    Else return FAIL + failure information.
  Else return Cover.
```

We start by finding a single edge t that overlaps with e. We use the term overlap to mean that their pointset intersection is not empty or degenerate (intersection at a vertex.) We extend t with adjacent edges that overlap e, so that t is possibly a chain that overlaps e. The extension process is a (recursive) search on the adjacency graph of the target Brep. The search halts on adjacent edges that do not overlap e.

At the end of this process, we are at one of three possible situations:

1. We have found an exact cover for e. In this case, either both e and t are closed, or the vertices of e coincide with the boundary

vertices of t (the boundary vertices are those vertices that bound a single edge.)

2. We have found a proper (but not necessarily valid) cover for e. In this case, either t is closed and e is open, or t is open and both boundary vertices of t are not in the interior of e (and at least one of them is not on a vertex of e.)
3. We have consumed all adjacencies without being able to cover e. In this case, t is open and at least one boundary vertex of t lies in the interior of e.

In case 1, we have found what we wanted. In case 3, the edge e cannot be covered by edges in the current target Brep, which means that it is not possible to define the desired feature's selections at this stage. In this case a rewrite is required. Treatment of rewrites is explained in Section 5.5 below. In case 2, we have found a cover c for e, but we may want to ensure that it is a valid cover. If it is not, selecting c as the feature's argument may result in selections that are larger than those made in the source system, which may in turn result in different geometries.

The reason why we may prefer not to verify validity of a cover has to do with the expected rate of invalidity occurrence: if it happens only rarely that a cover is not valid, it may be more efficient overall not to verify it at this stage. Recall again that each and every feature holds verification data that can be invoked by the main import loop of the whole architecture, so if the feature fails to import correctly it will be discovered anyway. In many cases it might be the case that it is not important to know exactly why import failed (whether it was a selections problem or another problem), in which case there is no reason to verify a cover's validity at this point in the selections algorithm.

If we do want to verify validity, our task now is to find whether or not a given set c of target edges form a valid cover for a given source edge e. This can be done elegantly as follows. For any target edge t in c, we now find a source cover for it, using a very similar algorithm, in which source and target are transposed. However, there is one fundamental difference between those two stages: at the present stage, we are not looking for a valid cover of t but for any cover of t. It is OK to use any source edge e_i now, even if it includes points that are outside of t, because all we want is to ensure that t's pointset is fully contained in the selection pointset.

In order to be convinced of the correctness of the above algorithm, note that we are at a stage in which we are handling a source edge e. All we need in order to handle it correctly is a valid cover made up of target edges. Those edges are in turn valid if they can be covered by a set of source edges. The crucial point is that those latter source edges will be dealt with in due course, when the main loop (Algo 1) will deal with them. Thus, the process does not have to continue recursively back and forth between the source selections and target Brep. Naturally, we may want to store edge visit tags in order to make the algorithm more efficient, by not checking covers for edges that have already been totally covered.

An alternative formulation of the algorithm is: as before, for each source edge find a set of target edges that cover it, not necessarily an exact covering. In general, the result is a graph of source edges and a graph of target edges. We know that the target graph covers the source graph. To check whether this cover is exact or proper we test whether the boundary vertices of the source graph coincide with the boundary vertices of the target graph (the boundary vertices are the vertices that bound a single edge.) If they do not coincide, then obviously we have a proper cover. If they do coincide, then we must show that the cover is exact. Let's assume that the cover is not exact, so the difference between the target pointset and the source pointset is not empty. Let's take the boundary points of the set difference. At least one of these points is internal to a target edge t (if they were all vertices of target edges, then we would have target edges that do not cover any source edges, which is impossible by construction.) This point must cover a source vertex. By our assumption, the source vertex is not a boundary vertex, so it bounds at least 2 source edges. One of these source edges overlaps t at this vertex. The other source edges cannot overlap t at this vertex, but since they are covered they must overlap other target edges at this point. This implies that there are at least 2 target edges that intersect at this point, so a target vertex must exist at this point, which is a contradiction.

5.4. Overlap Tests

A careful reading of the algorithm as described above shows that the main computational operation that is needed in order to implement it correctly is an overlap test, which will classify the overlap situation between two edges. However, due to efficiency reasons we also use a separate sub-algorithm for finding an initial target edge t that participates in a cover for a given source edge e.

5.4.1. Initial cover edge

For finding an initial cover edge t of a source edge e, we utilize an efficient point based technique. We select a random point p interior to the source edge e, and locate it in the current target model. Due to our assumption of model equivalence between source and target, the projection must lie on the boundary of the target model. Obviously, 'on' the boundary means using an appropriate numerical tolerance. If the point is too far from the boundary something very wrong must have happened, so we signal a serious error and exit the selections module.

Now, there are several possibilities regarding the location of the point p on the target Brep:

1. It is inside an edge t (again, using an appropriate tolerance). This is the common case, where the desired result has been identified immediately.
2. It is on a vertex. This is a rare case, but it can happen. From an implementation point of view, the simplest way to tackle this case is to select a new random point in the edge e until the point p does not lie on a vertex. Alternatively, we take a target edge that is adjacent to the vertex and overlaps the source edge e. If no such edge exists, then as before we signal a serious error. This tradeoff between running time and implementation effort (and thus code complexity) is standard in computational geometry, and systems may use different considerations in balancing the two.
3. It is on a face. In this case the topologies of the source and target model are radically different – there is an edge selected in the source that does not appear in the target topology. We add the edge e to a list of such edges to be handled separately, as explained in Section 5.5.
4. If the tolerance is too large, then the point can lie on more than one edge, more than one vertex, and more than one face. For a theoretically correct treatment of such cases, we would need a *separation assumption* that the minimum distance between vertices is larger than Delta, that the max-min distance between

edges/faces is greater than Delta, etc. Practically, however, it is simpler to reduce the tolerance and try again.

5.4.2. Generic edge overlap

Recall that the edge overlap test needs to classify the situation between the two edges as an exact overlap or a full containment or a partial overlap. As in the previous sub-algorithm, we do this using efficient point based techniques. Recall also our edge overlap assumption, that if a source and a target edges (partially or fully) overlap it is in a geometrically well behaved manner.

We partition the source edge into regions by projecting the vertices of the target edge onto the source edge. One of the vertices must be on the source edge, but the other may not be. The result has one, two or three regions (connectivity components.) Each region is either completely covered by the target edge or disjoint from the target edge. We now classify each region with respect to the target edge by classifying the mid-point of the region. This is basically the standard set Boolean operation CSG algorithm applied to one-dimensional entities.

Recall that the UPR architecture invoked verification algorithms after the import of each and every feature, so a problem will be immediately discovered, and it is easy to identify the source of the problem as an incorrect overlap test (most likely stemming from a tolerance problem) if needed. Hence our overlap test algorithm provides both efficient running time and an eventual guarantee of correctness.

5.5. Rewrites

Recall that the motivation for developing a selection algorithm such as the one detailed in this paper is to allow us to select Brep entities that serve as feature arguments in the context of feature based data exchange. Now, it is certainly possible that those Brep entities are simply not there at the target system when the need for them arises.

It might be argued that this situation means that the previous features have not been imported properly. However, we disagree with such an objection: the Brep topology in a CAD system is part of the internal representation of the system and should not make any difference to the user. What matters to the user are the design intent, as embodied (in today's CAD systems) in the parametric feature history, and the pointset geometry of the resulting model (because this is the thing that eventually obtains a physical manifestation). The internal division into vertices, edges and faces is a necessity resulting from our boundary representation technology. The external availability of Brep entities is useful exactly for the subject of this paper, selections to serve as feature arguments (including annotations), and for model visualization [Rappoport96].

In other words, vertices, edges and faces are important to the user inasmuch as the user is allowed to do something with them. Now, a proper place for ensuring that entities that are supposed to be present are indeed present is exactly when the user actually does something with them, which, in our FBDE context, is when the feature that needs those selections is imported.

Having said all that, the case of useful edges being present in the source system and not being present in the target system or vice versa is a case that we have rarely encountered in practice, apart from datums and in the non-manifold paradigm discussed below.

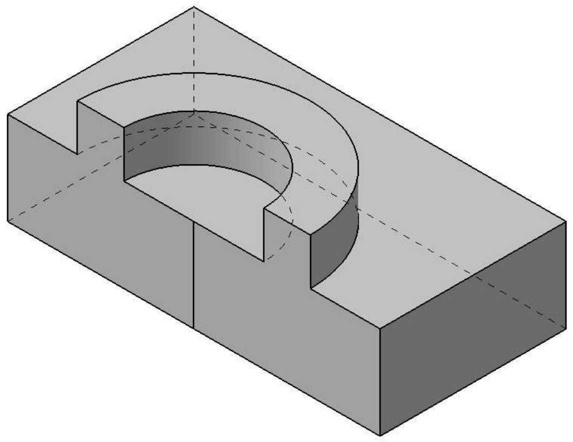

Figure 3: *The revolve on top of the box uses an edge as an axis. The edge is present in the source CAD system (Catia V5) but not in the target system (UG NX), which simplifies Brep when possible.*

This shows us that there probably is a 'user-natural' Brep topology for a given feature history. The Brep topologies in CAD systems are not equivalent (if they were, the algorithm in this paper would be greatly simplified), but the entities by which they differ are in practice usually entities that the user is not allowed to select as useful feature arguments.

A datum example where the Breps do differ is given in Figure 3. In this model, there is a 2-D sketch extruded to a box, and then a revolved sketch is joined on top of the box. The source system (Catia V5) has an edge resulting from the first extrude that is used as the axis of revolution. The edge is there because the 2-D sketch had a vertex at the middle of the bottom edge of the 2-D rectangle extruded. The target system (UG NX) does not allow that edge to be present in the model because it has a policy of simplifying the Brep if possible. Here, our rewrite case handler would try to define in the target system a new feature, a datum axis feature, positioned to coincide with the problematic edge, so that it could serve as the axis of revolution. The new axis feature could be added either fully parametrically (if the target system allows it, say by equations tying its position and orientation with that of the box's faces) or as a fixed axis, thus losing some of the parametricity of the model, but still retaining most of it. This is a nice example of the rewrite concept of the UPR architecture – as we see, a rewrite can insert a wholly new feature if it is needed in order to enable the parametric import of another feature into the target system.

Another case in which CAD systems exhibit meaningful Brep entity differences is the case of non-manifold Breps. This is actually a different modeling paradigm altogether: the modeled object is not a geometric pointset but a geometric pointset having an internal subdivision. Although non-manifold geometry has been long recognized as useful for many applications (most notable for engineering finite-element analyses), only some CAD systems (e.g., I-DEAS) support this paradigm generically today.

Non-manifold support has direct consequences on our ability to handle the selections-on-face case (e.g. item 3 in Section 5.4.1). First let us assume that the target system supports non-manifold representations. We need to add source system selections to the target Brep model as user-selectable entities. Since the target system supports non-manifolds, it is reasonable to expect it to provide a 'split face by adding a new edge' feature to the user. In this case our system would first use this feature in order to create the missing edges on the appropriate faces, and then import the feature requiring those selections using the usual feature import procedure. Note that again, the target model would contain a different number of features from the source model.

It may be the case that a system that supports non-manifold Brep representation does not provide direct support for non-manifold operations to the user, as features that can be added to the feature graph as seen by the user. However, it is still possible in principle that such operations are available to CAD extension programmers through the CAD system's API, and can be added internally to the feature graph. In this case we proceed as before, and the features seen by the user at the source and target system are similar. It is also possible that the target system provides a 'split face' feature even if in general it does not support non-manifold representation.

Even when a 'split face' feature of the kind we had used is not provided, in many systems it can be emulated. Many systems contain a 'patch' feature whose arguments are a solid Brep B and an open surface sheet S having a material side and boundary edges that are assumed to lie on the boundary of the solid B. The result of the feature is to glue S to the boundary of B and discard that part of B's boundary that lies on the non-material side of S. We had used the patch feature in our algorithm for solving the geometry per feature (GPF) problem [Spitz04].

To emulate split face using patch, we prepare a surface sheet that coincides with part of the target face and whose boundary edges include the desired selection edges. The material side is specified to be identical to that of the face. This may result in the desired selection edges being added to the current Brep.

This positive result is however not guaranteed, because it depends on the specific implementation of the patch feature and on the target system's general policy. It is possible that such an input to the patch feature be considered invalid, because it tries to force a non-manifold situation into a system that takes pain to ensure that such situations are not present, or simply because the system detects that no portion of the previous Brep is removed and hence the feature is considered redundant. The problem here is the classic problem [Mäntylä88] of what is considered by CAD systems to be a valid face. Another manifestation of the problem is that many systems actively unify faces by removing edges separating them when those faces are considered to be the 'same' face, e.g., when they are co-planar.

To summarize, our capability of creating selection edges where they do not exist is a function of the feature repertoire of the target system. When the target system does not allow a direct or emulated split face feature, it may still be possible to import the feature F parametrically, by replacing F by a totally different feature combination that achieves the same geometric effect. This should be examined on an individual feature basis. The UPR architecture enables doing that through its support of feature rewrites, using a single feature or a number of features.

6. Implementation

The 1-D selection algorithms described in this paper have been implemented in the UPR architecture at Proficiency Ltd. The current UPR implementation supports the five high-end CAD systems in the market: Catia 4, Unigraphics, I-DEAS, ProEngineer, and Catia 5. Several versions and most of the design features of each system are supported. The data exchange process is controlled by a web server through a web-based user interface. The server locates export and import 'agents' over a network and distributes export and import jobs according to load parameters. The software is being used routinely in real product projects.

In our implementation, the UPR file stores all relevant data, including the selection data, represented geometrically as described in this paper. Intermediate computations (such as point projections for the edge overlap tests) are usually done on the UPR data structures, but obviously any other software library, including that of the target CAD system, could be used as well.

As mentioned in Section 5.5, we rarely encounter the need to invoke the rewrites procedure. This stems from the fact that although the internal Breps used by CAD systems are different, the selection model presented to the user is quite similar.

Figure 4 shows a part made up of a union of four cylinders and a round, resulting in Breps that are substantially different. Figure 5 shows a part originally designed in ProEngineer and exchanged into Catia V5. In both cases, our algorithm finds the correct edges and the UPR architecture performs a perfect feature-based data exchange.

7. Discussion

The issue of supporting one-dimensional selections is a crucial one in feature-based data exchange systems and algorithms. Selections are used as feature arguments in the most frequent and fundamental features, among them rounds and extrusions. Selections are what endows models with true associativity, and FBDE systems must support selections in a way that is as close as can be to the design intent as expressed in the source CAD system.

In this paper we have provided the first solution of which we are aware to this important problem. Our solution is applicable to a wide variety of FBDE architectures, among them the UPR and the STEP architectures, which are the only documented ones at present (note that the UPR has been implemented in practice, while STEP is only a proposed specification.) Our algorithms have been implemented in the UPR architecture, and are thus being used on a daily basis in real projects.

As we had mentioned previously, a full solution to the selections problem should include selections of two-dimensional entities. We have designed and implemented algorithms that solve that problem as well. However, they are different from the algorithms of the 1-D case, so due to space constraints they will be described in a different paper.

Future CAD systems may let users express their intents at a higher level of abstraction than that of feature-based design, for example using purely functional specification. It is reasonable to expect that selections will continue to play a role in such systems as well. In such systems, the data exchange problem could be required to prefer preservation of functional spec rather than of lower level

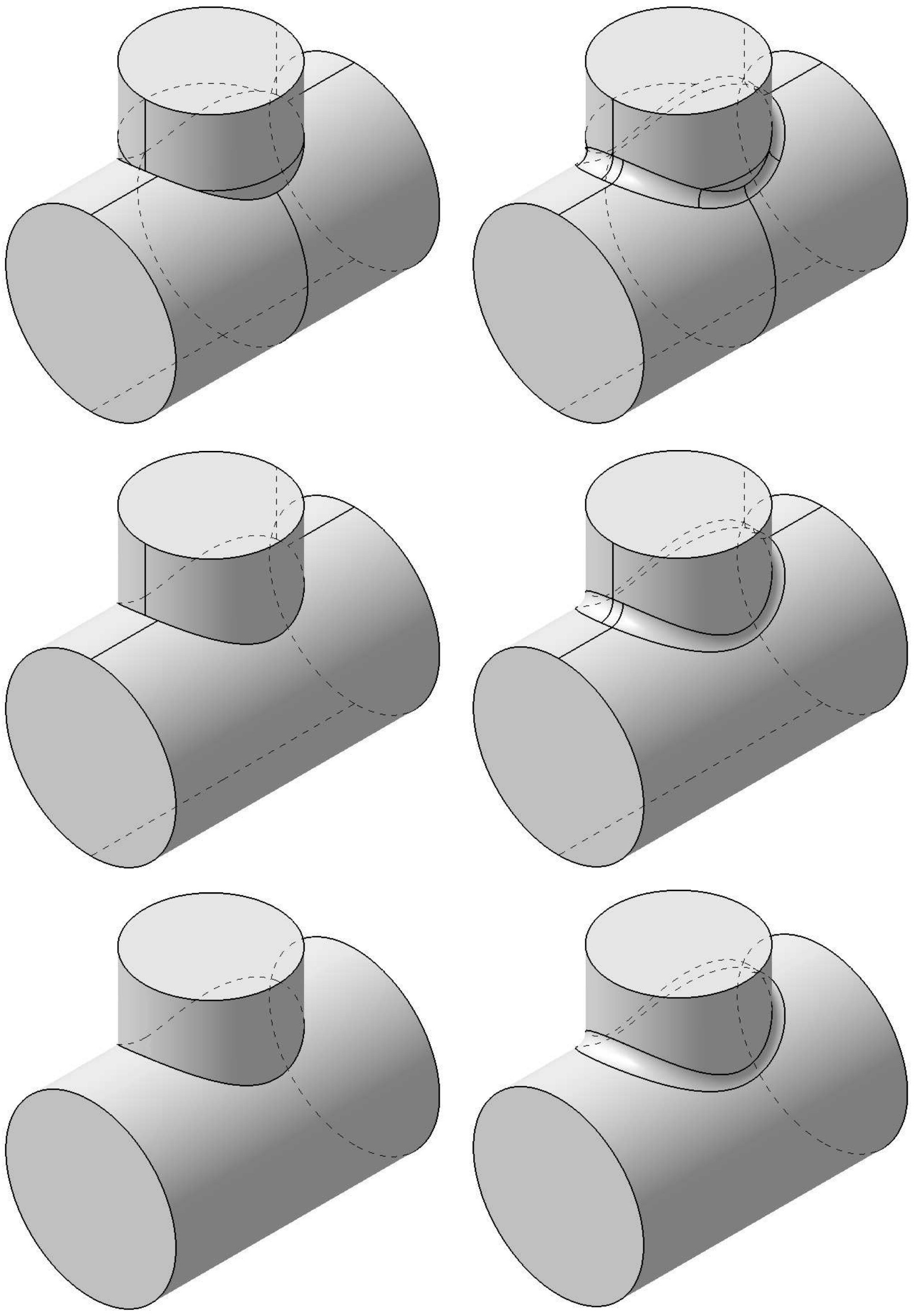

Figure 4: *Three CAD systems, before a round feature (left) and after it (right). Top: Catia V4; middle: Catia V5 (ProEngineer has a similar topology); bottom: UG NX. Note how Brep topology around the round differs.*

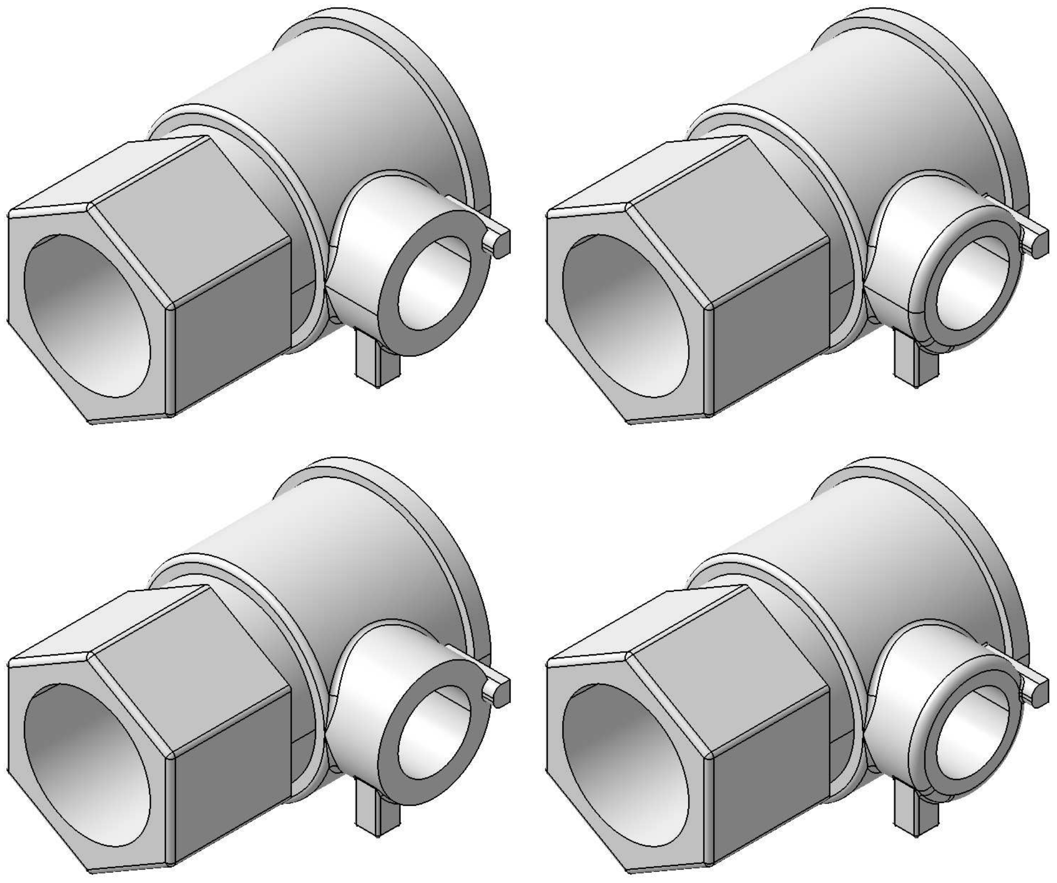

Figure 5: *A real part, in Catia V5 (top) and ProEngineer (bottom), before (left) and after (right) rounding the protrusion on the right side.*

geometry. In such a scenario support for selections would require the development of different algorithms, which would probably be more in the direction of today's persistent naming algorithms.

Acknowledgements. The Proficiency UPR implementation is a collective effort of the Proficiency Data Exchange development team, headed by Alex Tsechansky. The content of this paper is patent protected. We thank Mike Ricci for the valve example.

References

Capoyleas96 Capoyleas, V., Chen, X., Hoffmann, C.M., Generic naming in generative, constraint-based design. Computer-Aided Design, 28(1):17-26, 1996.

Hoffmann93 Hoffmann, C.M., Juan, R., Erep, an editable, high-level representation for geometric design and analysis. In: P. Wilson, M. Wozny, and M. Pratt, (Eds), Geometric and Product Modeling, pp. 129-164, North Holland, 1993.

Kripac97 Kripac, J., A mechanism for persistently naming topological entities in history-based parametric solid models. Computer-Aided Design, 29(2):113–122, 1997. Also: proceedings, Solid Modeling '95, pp. 21–30, ACM Press, 1995.

Mäntylä88 Mäntylä, M., An Introduction to Solid Modeling, Computer Science Press, Maryland, 1988.

Mun03 Mun, D., Han, S., Kim, J., Oh, Y., A set of standard modeling commands for the history-based parametric approach. Computer-Aided Design, 35:1171-1179, 2003.

Pratt04 Pratt, M.J., Extension of ISO 10303, the STEP standard, for the exchange of procedural shape models. Proceedings, Shape Modeling International 2004 (SMI '04).

Qi04 Qi, J., Shapiro, V., Epsilon-solidity in geometric data translation, TR SAL 2002-4, Spatial Automation Laboratory, University of Wisconsin-Madison, June 2004.

Rappoport96 Rappoport, A., Breps as displayable-selectable models in interactive design of families of geometric objects. Geometric Modeling: Theory and Practice, Strasser, Klein, Rau, (Eds), Springer-Verlag, pp. 206-225, 1996.

Rappoport97 Rappoport, A., The Generic Geometric Complex (GGC): a modeling scheme for families of decomposed pointsets. Proceedings, Solid Modeling '97, May 1997, Atlanta, ACM Press.

Rappoport03 Rappoport, A., An architecture for universal CAD data exchange. Proceedings, Solid Modeling '03, June 2003, Seattle, Washington, ACM Press.

Shah95 Shah, J.J., Mantyla, M., Parametric and Feature-Based CAD/CAM, Wiley, 1995.

Spitz04 Spitz, S., Rappoport, A., Integrated feature-based and geometric CAD data exchange. Proceedings, Solid Modeling '04, June 2004, Genova, Italy, ACM Press.