ELSEVIER

Computer-Aided Design 37 (2005) 1266–1284

COMPUTER-AIDED DESIGN

www.elsevier.com/locate/cad

Efficiency of boundary evaluation for a cellular model

R. Bidarra[a,*], J. Madeira[b], W.J. Neels[a], W.F. Bronsvoort[a]

[a]*Computer Graphics and CAD/CAM Group, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Mekelweg 4, NL-2628 CD Delft, The Netherlands*

[b]*Department of Electronics and Telecommunications, University of Aveiro, Campus Universitário de Santiago, P-3810-193 Aveiro, Portugal*

Received 2 February 2004; received in revised form 20 December 2004; accepted 24 December 2004

Abstract

Feature modeling systems usually employ a boundary representation (b-rep) to store the shape information on a product. It has, however, been shown that a b-rep has a number of shortcomings, and that a cellular representation can be a valuable alternative. A cellular model stores additional shape information on features, including the feature faces that are not on the boundary of the product. Such information can be profitably used for several purposes.

A major operation in every feature modeling system is boundary evaluation, which computes the geometric model of a product, i.e. either the b-rep or the cellular model, from the features that have been specified by the user. Since boundary evaluation has to be executed each time a feature is added, removed or modified, its efficiency is of paramount importance.

In this paper, boundary evaluation for a cellular model is described in some detail. Its efficiency is compared to the efficiency of boundary evaluation for a b-rep, on the basis of both complexity analysis and performance measurements for the two types of evaluation. It turns out that boundary evaluation for a cellular model is, in fact, more efficient than for a b-rep, which makes cellular models even more attractive as an alternative to b-reps.

Keywords: Feature modeling; Boundary representation; Cellular model; Boundary evaluation; Efficiency; Complexity analysis; Performance measurements

1. Introduction

Feature modeling is now the predominant approach for product modeling. Shape and related functional information can be stored in a single product model, which can considerably improve the product modeling process [1].

Current feature modeling systems usually maintain a dual representation for a product: on the one hand, a parametric definition of the product, and, on the other hand, a geometric model representing the resulting shape; see Fig. 1. The parametric definition might consist of a large variety of information, normally organized in a graph structure. Typically, the graph represents relations among instances of parameterized features, constraints, Boolean operations, auxiliary geometric entities, etc.

Most commercial feature modeling systems employ a boundary representation (b-rep) as geometric model to store the shape information. B-reps have been successfully used in traditional geometric modeling systems to support specification, visualization and analysis of a model, but they are, in fact, not powerful enough to store all shape information that is relevant in a feature modeling context. Therefore, cellular models have been suggested as an alternative to b-reps. In a cellular model, additional shape information on features can be stored, e.g. not only the faces of the features that are on the boundary of the product, but also those faces that are not on the boundary; see [2] for an overview of several proposals.

The cellular model introduced by the Computer Graphics and CAD/CAM Group of Delft University of Technology [2], for example, has been profitably used for several purposes. First, it allows the semantics of features to be specified, and the validity of a feature model to be maintained, which is essential to make feature modeling

* Corresponding author. Tel.: +31 15 278 4564; fax: +31 15 278 7141.
E-mail addresses: bidarra@ewi.tudelft.nl (R. Bidarra), jmadeira@det.ua.pt (J. Madeira), bronsvoort@ewi.tudelft.nl (W.F. Bronsvoort).
URL: http://graphics.tudelft.nl.

0010-4485//$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2004.12.006

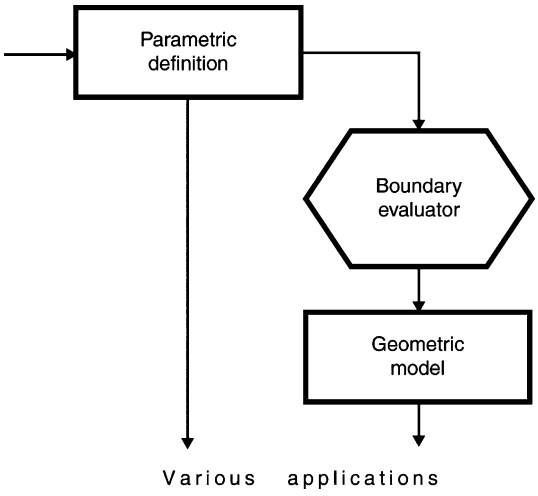


Fig. 1. Generic scheme of current feature modeling systems.

really more advantageous than geometric modeling [3]. Second, more functional information of a feature model can be visualized, e.g. the above-mentioned faces of features that are not on the boundary of the product, which can be very helpful during specification and analysis [4]. Third, it can serve as one of the main data structures for a multiple-view feature modeling system, which can support the integration of several product development phases [5].

Boundary evaluation is the process that computes the geometric model of the product, i.e. either a b-rep or a cellular model, on the basis of the parametric definition [6]; see again Fig. 1. It is a major operation in any feature modeling system, and, since it has to be executed each time a feature is added, removed or modified, its efficiency is of utmost importance.

In this paper, boundary evaluation for a cellular model is described in some detail. On the basis of the higher complexity of a cellular model, compared to a b-rep representing the same product, one can legitimately ask how efficient boundary evaluation for a cellular model is, compared to boundary evaluation for a b-rep.

The main goal of this paper is to answer this question. This is done in two ways: (i) the computational complexity of boundary evaluation algorithms for both representations is analyzed [7], and (ii) statistics emerging from performance measurements of boundary evaluation for both representations are presented and discussed. Performance of boundary evaluation for a b-rep was measured with the commercial feature modeling system Pro/ENGINEER [8], and for a cellular model with SPIFF [3], a prototype feature modeling system developed at Delft University of Technology. Preliminary performance measurements, without any complexity analysis, have been published in [9].

Notice that for the boundary evaluation for the cellular model, in the SPIFF system, the right arguments for the complexity analysis and the explanation of the performance measurements can always be given, but that for the boundary evaluation for the b-rep, in the Pro/ENGINEER system, sometimes only plausible arguments can be given. Insufficient information is available on the latter system to do better here.

In Section 2, model classes are defined for best, average and worst case practical behavior for boundary evaluation, and representative models are presented for each of them. In Section 3, boundary evaluation for a b-rep is discussed, and a complexity analysis of this process is given. In Section 4, the cellular model of SPIFF, and the basic operations on it, are described. In Section 5, boundary evaluation for the cellular model and its complexity analysis are presented. In Section 6, the outcome of the performance measurements for the two types of boundary evaluation is shown. In Section 7, conclusions on the efficiency of boundary evaluation for the cellular model are given.

2. Best, average and worst cases

In this section, three classes of feature models are identified that present, in practice, best, average and worst case behavior for boundary evaluation, and a representative model is proposed for each class.

As will be recalled in the next sections, boundary evaluation for feature models strongly relies on the use of Boolean operations. The computational cost of such an operation, in turn, is directly dependent on the number of topologic entities in each of the operands, as well as on the number of intersections among such entities. Therefore, in the definition of the classes and their representative models, these two factors play a predominant role. The representative models are regular, in the sense that they contain sets of identical features, and are, therefore, suitable for studying the behavior of the boundary evaluation algorithms for both b-rep and cellular model. These models will be used throughout the paper, both in the complexity analyses and in the performance measurements.

So, in this paper, the best, average and worst cases do not refer to mathematically well-defined, unique cases, because this is impossible in the complex domain of feature models considered here. Instead, these cases refer to (i) classes of models which present, respectively, best, average and worst behavior in practice during boundary evaluation, and/or (ii) the representative models of these classes.

2.1. Best case

From a practical point of view, the class of feature models presenting best case behavior for boundary evaluation is made up of all models whose features are disjoint, i.e. for each one of the models, each feature just intersects some base stock.

Fig. 2(a) shows the representative model for this class, which consists of a block with one row of 100 non-intersecting cylindrical through hole features (numbered from 1 to 100). This model can be built from scratch by performing a sequence of 100 add feature operations.

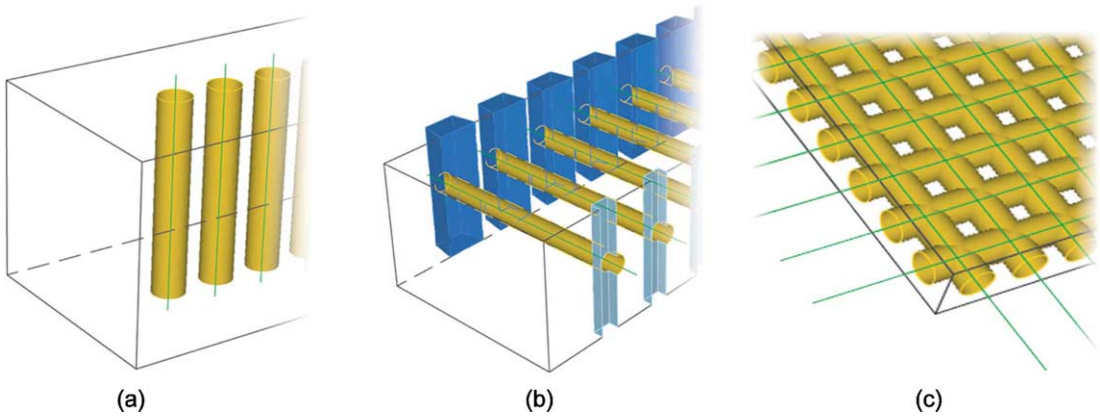


Fig. 2. Representative models for (a) best case, (b) average case and (c) worst case behavior for boundary evaluation.

2.2. Average case

In most real-world models, several features intersect each other. However, given the local character of features, which are typically small compared to the size of a whole model, each feature usually intersects only a limited number of other features.

Therefore, the class of feature models presenting average case behavior for boundary evaluation consists of all models for which each of its m features has a small average number i of intersections with other features, i being independent of m.

The representative model for this class is shown in Fig. 2(b). It consists of a block with one row of 33 'feature groups', each 'group' having three intersecting features: a rib, a slot, and a through hole spanning between both. In terms of boundary entities, each hole feature intersects one face of the rib, two faces of the slot, as well as one face of the block. It is assumed that the features in each 'group' are inserted sequentially: first the rib, then the slot, and finally the through hole. Note that only the computation times for operations on the hole features (numbered 3,6,…,99) will be considered throughout the paper, thus guaranteeing the desired regular behavior of this model regarding boundary evaluation.

2.3. Worst case

Considering the two predominant factors pointed out above, one can easily conclude that there is no absolute *worst case behavior* in terms of boundary evaluation. The number of topologic entities in a model, and of their intersections with a new feature, has virtually no upper bound. So, whichever adverse situation one devises, it is always possible to think of an *even worse* model. An example of a 'very bad' case would be a model with many concave features which intersect each other many times.

From a practical point of view, however, the class of feature models presenting worst case behavior can be thought of being made up of all models for which each of its m features intersects (once) i other features, i being now a fraction of m.

The representative model for this class is shown in Fig. 2(c). It consists of a block with 40 similar through holes, each of them intersecting 20 other through holes, i.e. half of the total number of features. To build this model keeping that fraction (almost) constant, the through holes are added alternately: one 'horizontally' (intersecting all existing 'vertical' holes), the next 'vertically' (intersecting all existing 'horizontal' holes), and so forth.

3. Boundary evaluation for a b-rep

Most commercial modeling systems are deemed history-based: the most important relation in the parametric definition graph is the order of creation of feature instances, which defines the *model history*. Such systems typically use a b-rep as geometric representation.

In history-based systems, adding a new feature to the model, by providing a set of input values for its parameters, appends a new node to the model history, yielding a new parametric definition of the product. Similarly, feature instances can be modified by specifying new values for their parameters, or can be deleted from the model. This is done by modifying, or deleting, the respective feature node in the model history.

3.1. Boundary evaluation

The goal of the boundary evaluator in Fig. 1 is, for history-based systems, to generate a b-rep that matches the parametric definition at each moment [6]. Therefore, whenever a new parametric definition is made, it should be input to the boundary evaluator, and the corresponding new b-rep should be generated.

When a new feature is added to (the top of) the model history, all the evaluator needs to do is to combine the new feature shape with the current b-rep. For this, Boolean union

operations are used to process additive features, and Boolean difference operations to process subtractive features.

However, when a feature is modified in, or removed from, the model history, the current b-rep is, in general, of little use for the boundary evaluator. The simplest way to generate the new b-rep is then to sequentially walk through the nodes in the model history, and re-execute the associated Boolean operations to build a new b-rep from scratch. Fig. 3 presents an example of this re-evaluation process. When the width of the pocket highlighted in Fig. 3(a) is decreased, the whole model history in Fig. 3(b) is sequentially re-executed, yielding the model in Fig. 3(c). So, in this re-evaluation scheme, all features in the model history are processed time and again.

Re-executing the whole model history after modifying, or removing, a feature has the shortcoming that its computational cost is proportional to the number of features in the model history. An improvement to this method, which is currently used in most boundary evaluators, consists of keeping the *intermediate models* between all history steps, in which case only the history steps after the modified, or removed, feature node need to be re-executed. Using this method, the computational cost of boundary evaluation after modifying, or removing, a feature becomes dependent on the feature node position in the model history, but the amount of memory required to store all intermediate models is significant. An alternative improvement consists of storing only the *deltas*, i.e. the model differences, between history steps. This alternative requires less storage, because *deltas* are typically much smaller than intermediate models, but more computation time is needed to rollback from the current b-rep to the state from which the model needs to be re-evaluated. In both alternatives, however, typically several features that are actually left unchanged by the operation have to be re-processed, as is, for example, the case for features 7–11 in Fig. 3.

3.2. Complexity analysis

The basic feature operations on the b-rep model were introduced in Section 3.1. In what follows, these operations are analyzed in terms of the required computation time, and the associated computational complexity, using the classes and representative models defined in Section 2.

3.2.1. Add feature operation

Let us consider a model made up of m features and n boundary faces. Adding a new feature (F) to the model is accomplished by combining the shape of the new feature with the current b-rep model, through the appropriate Boolean union or difference operation.

The required computation time (t) can be decomposed into

$$t = t_{\text{int}} + t_{\text{op}} + t_{\text{upd}},$$

where t_{int} represents the time associated with the identification of the n_{int} boundary faces intersecting F, t_{op} represents the time associated with the processing of these boundary faces to accomplish the required Boolean operation, and t_{upd} represents the time required for the update of the b-rep representation.

Regarding the number of boundary faces involved, one can argue that:

- the time t_{int} is proportional to the total number of faces defining the model boundary (n), since each face has to be tested for intersection with the faces defining the new feature F;
- the time t_{op} is proportional to the number (n_{int}) of faces intersecting the new feature F;
- the time t_{upd} is proportional to the number (n_{op}) of (new and/or modified) faces arising as a result of the performed Boolean operation.

The required computation time can then be written in terms of the number of faces involved in the various

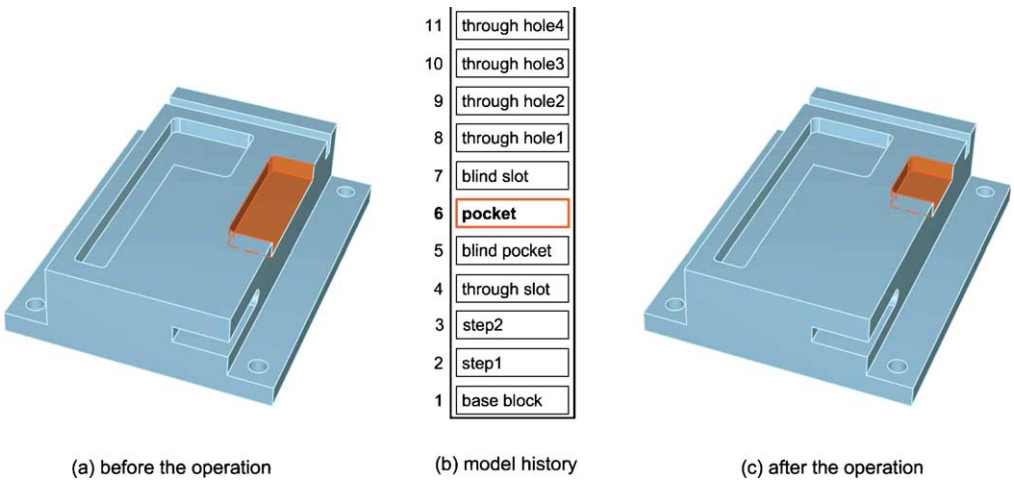


Fig. 3. Boundary re-evaluation after a feature modification.

operations as

$$t = \alpha \times n + \beta \times n_{\text{int}} + \gamma \times n_{\text{op}},$$

where α, β, and γ are (constant) positive factors. Note that these factors express average behavior: in general, not all (intersected or created) faces are equally complex, nor require the same computation time to be processed.

In what follows, given the general equation above, the computation time of the add feature operation will be analyzed for the best, average and worst cases of Section 2.

Best case. For a model in the best case class, the added feature just intersects a few faces of the base stock, and the faces defining the added feature have to be properly combined into the model b-rep.

Consider now the representative model of Fig. 2(a). Clearly, in terms of the number of boundary faces, a new hole feature F intersects just two faces of the block ($n_{\text{int}}=2$) and, after the feature addition, the model boundary is defined by one more face. Regarding the computation time, we then have

$$t = \alpha \times n + 2 \times \beta + 3 \times \gamma.$$

Therefore, the computation time needed for adding a new feature to the model linearly increases with the number of faces already making up the model boundary. And, as a result of that operation, the overall number of faces increases by just one unit. Regarding its complexity order, the computation time has, thus, linear complexity, $O(n)$.

Let us consider now that this model is being built up through a sequence of best case feature additions. Regarding the addition of the kth feature, the increase in the number of boundary faces can be represented as

$$n_k = n_{k-1} + 1,$$

where n_k represents the number of faces after adding the kth feature, with $n_0=6$ representing the (initial) block defined by six faces. For the required computation time t_k we have

$$t_k = \alpha \times n_{k-1} + 2 \times \beta + 3 \times \gamma.$$

The above equations can be further simplified to

$$n_k = k + 6, \quad k = 0, 1, \ldots, 100,$$

and

$$t_k = \alpha \times k + (5 \times \alpha + 2 \times \beta + 3 \times \gamma) = \alpha \times k + \Delta_B,$$

$$k = 1, 2, \ldots, 100.$$

Thus, both the overall number of boundary faces and the computation time associated with adding a feature have, in this case, linear complexity order.

Average case. For a model in the average case class, an added feature F intersects a limited number of other model features and, thus, a limited number of boundary faces ($n_{\text{int}}=r$), and a limited number of boundary faces are generated or modified ($n_{\text{op}}=s$).

Regarding the computation time, we then have

$$t = \alpha \times n + \beta \times r + \gamma \times s.$$

As explained in Section 2, the number of actually intersected faces (r) is smaller than n, and knowing that s depends on both r and the number of faces bounding the inserted feature, it can be concluded that the computation time required by an add feature operation, for the average case, has linear complexity order, similarly to the best case above.

In order to better understand the average behavior, a detailed analysis pertaining to the representative model of Fig. 2(b) will be given in what follows. Note that any added through hole feature intersects a rib and a slot, as well as the block, in such way that:

- the hole feature intersects one of the faces defining the rib, as well as two of the faces defining the slot;
- five boundary faces are generated or modified after the feature addition.

Let us consider now that the model is being built up by a sequence of add feature operations, corresponding to successively adding a rib, a slot and a cylindrical through hole, 33 times. Adding just a rib and a slot feature increases the overall number of boundary faces by eight. Adding next a hole feature further increases the total number of faces by two.

Similarly to what was done for the best case, and regarding the insertion of the kth hole feature, the total number of faces is given by

$$n_k = n_{k-1} + 10,$$

with $n_0=6$ representing the number of faces in the initial model defined just by the block. Note that the 10 additional faces result from the insertion of a 'group' (rib, slot and hole). This recursive equation can be written in closed form as

$$n_k = 10 \times k + 6, \quad k = 0, 1, \ldots, 33.$$

For the computation time associated with the insertion of the kth hole feature, we can now write

$$t_k = \alpha \times (n_{k-1} + 8) + 3 \times \beta + 5 \times \gamma,$$

assuming that a rib and a slot were already added and eight additional faces were created. This can be further simplified to

$$t_k = 10 \times \alpha \times k + (4 \times \alpha + 3 \times \beta + 5 \times \gamma)$$

$$= 10 \times \alpha \times k + \Delta_A,$$

$$k = 1, 2, \ldots, 33.$$

Thus, both the overall number of faces and the computational cost associated with adding a hole feature have, as expected, linear complexity order.

Worst case. When adding a feature to a model in the worst case class, the new feature intersects a substantial fraction of the features making up the model, entailing a large increase in the number of faces defining its boundary.

In order to better understand such worst case behavior, as well as the fundamental difference regarding the previous cases, a detailed analysis pertaining to the representative model of Fig. 2(c) will be given in what follows. Note that this model has an even number (40) of cylindrical through hole features, and that there are as many 'vertical' as 'horizontal' hole features. Any of the 20 horizontal, respectively vertical, hole features intersects once each one of the 20 vertical, respectively horizontal, hole features.

To see how the number of boundary faces grows after adding a new feature, let us analyze how the model can be built through a sequence of feature addition operations, starting with just one block:

- the initial model is the block defined by six faces: $n_0=6$;
- adding the first feature (say, a vertical through hole) results in a boundary defined by seven faces: $n_1=7$;
- adding the second feature (a horizontal through hole) intersecting the first one, both holes splitting each other into two, increases the number of faces by three.

Consider now that several features are successively added to the model, in this alternate way, and that the next feature being added is the kth hole feature, which intersects (k div 2) other hole features, as well as two block faces.

After adding the kth feature, the overall number of faces is then,

$$n_k = n_{k-1} + 2\times(k \text{ div } 2) + 1, \quad k = 1,2,\ldots,40,$$

and the computation time for such an operation is

$$t_k = \alpha\times n_{k-1} + \beta\times(k \text{ div } 2 + 2) + \gamma\times(3\times(k \text{ div } 2) + 3),$$

$$k = 1,2,\ldots,40.$$

Note that, when adding the kth feature, the number of faces added to the model is approximately equal to twice the number of hole features intersected by the added feature.

After some algebraic manipulation, we get a closed formula for the number of faces after adding the kth feature:

$$n_k = \frac{1}{2}(k+1)k + (k+6) - [(k+1) \text{ div } 2],$$

$$k = 1,2,\ldots,40.$$

Grouping the terms we can write

$$n_k \approx \frac{1}{2}k^2 + k + \Delta_W, \quad k = 1,2,\ldots,40.$$

Therefore, the number of faces is proportional to the square of the number of inserted features and has, thus, $O(n^2)$ computational complexity.

The required computation time can be approximated as:

$$t_k \approx \alpha_W\times k^2 + \beta_W\times k + \gamma_W, \quad k = 1,2,\ldots,40.$$

Thus, contrary to the best and average cases, both the overall number of faces and the computational cost associated with adding a new feature for this worst case have quadratic complexity order.

3.2.2. Remove feature operation

Let us consider a model made up of m features and n faces. Removing from the model a given feature F is accomplished by identifying the feature's position in the model history, and sequentially walking through the history nodes and re-executing the associated Boolean operations, as explained in Section 3.1. The computational cost depends, thus, on the position of the removed feature in the model history and on the time required to accomplish each of the necessary Boolean operations.

Suppose the kth feature is to be removed. The required computation time (t_k) can be decomposed into

$$t_k = t_{\text{rb}} + \sum_{j=k+1}^{m} t_{\text{insertion}}(F_j),$$

where t_{rb} represents the time associated with the appropriate rollback of the model history, and the summation represents the time required to re-add the $(m-k)$ features. In the following analysis, we will assume the latter is the dominant term, and t_{rb} will, therefore, be disregarded.

Given the general equation above, the computation time of the remove feature operation will be analyzed for the best, average and worst cases of Section 2.

Best case. To illustrate the best case behavior for feature removal, we use again the representative model of Fig. 2(a): the feature F being removed just intersects the block, and is independent of any other features.

The equation representing the required computation time, for removing the kth feature ($k<m=100$), is then written as

$$t_k = \sum_{j=k}^{99} t_{\text{insertion}}(j),$$

since the features being re-added are all of the same type.

Using the expression previously obtained for the best case for feature addition we get

$$t_k = \sum_{j=k}^{99}\{\alpha\times j + \Delta_B\},$$

which can be written in closed form as

$$t_k = \frac{\alpha}{2}(100-k)(99+k) + (100-k)\times\Delta_B,$$

$$k = 1,2,\ldots,99.$$

This is a quadratic equation in k.

Note that, although the various hole features are disjoint and of the same type, removing any of these features does not have the same computational cost: the position of the removed feature in the model history is crucial.

Average case. To illustrate the average case behavior for feature removal, we use again the representative model of

Fig. 2(b). Note the difference to the previous case: from each 'group' made up of a rib, a slot and a through hole, the first two features remain in the model, and only the hole feature is being removed.

The equation representing the required computation time for removing the kth hole feature (for $k<33$) is written as

$$t_k \approx \sum_{j=k+1}^{33} t_{\text{insertion}}(\text{Rib}_j) + \sum_{j=k+1}^{33} t_{\text{insertion}}(\text{Slot}_j) + \sum_{j=k+1}^{33} t_{\text{insertion}}(\text{Hole}_j),$$

where each summation represents the re-addition of features of the same type. Note that this formula is now approximate: in fact, the actual times for re-adding all these features are slightly lower than those previously obtained for the add feature operation, because the number of faces in the model is now somewhat lower, due to the absence of the kth hole feature.

Let us just analyze the last summation. Using the expression previously obtained for the average case of the add feature operation, we estimate

$$t_{k;\text{Holes}} \approx \sum_{j=k+1}^{33} \{10 \times \alpha \times j + \Delta_A\},$$

which can be written in closed form as

$$t_{k;\text{Holes}} \approx 5 \times \alpha \times (33-k)(34+k) + (33-k) \times \Delta_A,$$

$$k = 1, 2, \ldots, 32.$$

Again, this is a quadratic equation in k. Similar expressions would be obtained for the other two summations, and thus for the total time t_k.

Worst case. For a model in the worst case class, the number of features intersecting the feature F being removed is a fraction of the overall features. To illustrate this situation, the representative model of Fig. 2(c) is used again.

The equation representing the required computation time (for $k<m=40$) is then, similarly as for the best case, written as

$$t_k = \sum_{j=k}^{39} t_{\text{insertion}}(j).$$

Using the expression previously obtained for the worst case for feature addition, we can estimate this time as

$$t_k \approx \sum_{j=k}^{39} \{\alpha_W \times j^2 + \beta_W \times j + \gamma_W\},$$

which is equivalent to

$$t_k \approx \sum_{j=1}^{39} \alpha_W \times j^2 - \sum_{j=1}^{k-1} \alpha_W \times j^2 + \sum_{j=k}^{39} \beta_W \times j + \sum_{j=k}^{39} \gamma_W,$$

$$k = 1, 2, \ldots, 39.$$

The above equation can be written in closed form as

$$t_k \approx 20,540 \times \alpha_W - \frac{1}{6}k(k-1)(2k-1) \times \alpha_W + \frac{1}{2}(40-k) \times (39+k) \times \beta_W + (40-k) \times \gamma_W,$$

which is a polynomial of degree three in k.

In terms of computational complexity, for this worst case model, the remove feature operation is therefore, $O(n^3)$.

3.2.3. Modify feature operation

The computation time required for the evaluation of the modification of a feature is similarly dependent on the sequence number of the feature being modified. After changing the parameters associated with the given feature, it is necessary to sequentially walk through the succeeding nodes in the model history and re-execute the associated Boolean operations.

The required computation time can be estimated in a similar way to what was done above. Note, however, that an additional hole feature is re-added in every case, corresponding to the modified feature. The analysis will, still, result in $O(n^2)$ complexity for the best and average cases, and in $O(n^3)$ complexity for the worst case.

4. The cellular model

In this section, the cellular model used in the SPIFF system is described, with emphasis on its functionality for modifying model topology [2].

4.1. Basic notions

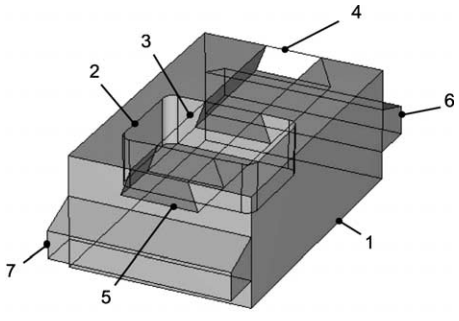
The cellular model is a non-manifold geometric representation of the feature model of a product, and integrates the contributions from all its volumetric features. It represents each part as a connected set of volumetric quasi-disjoint *cells* of arbitrary shape, and represents each feature as a connected subset of these cells. The cells are such that a part is exactly represented, i.e. its shape is not approximated. The cellular subdivision is determined by the property that two cells may never volumetrically overlap. So, whenever two features overlap, their cells are such that one or more cells are shared by the two features, and the remaining cells lie in either of them.

Any two adjacent cells are separated by a number of interior faces in the cellular model. Such faces can be regarded as having two 'sides', designated as partner *cell faces*. A face that lies on the boundary of the cellular model has only one cell face (one 'side'), that of the only cell it bounds. In either case, a cell face always bounds one and only one cell. As a consequence of this cell subdivision, each feature face is represented by a connected set of cell faces. See Fig. 4 for an example of a cellular model containing some overlapping features.

To identify and analyze features in the cellular model, each cell has as attribute an *owner list* indicating which features it belongs to (see again Fig. 4). Similarly, each cell face has an owner list indicating which feature faces it belongs to. Finally, the nature of a cell expresses whether its volume represents 'material' of the product or not, and, similarly, the nature of a cell face expresses whether it lies on the boundary of the product or not.

So, the cellular model contains much more information than only the model boundary of the product. In particular, it contains explicit information on the 'not on boundary' faces of subtractive features and on intersecting features. The cellular model, including its attribute mechanism to maintain the owner lists of cells and cell faces, has been implemented in the SPIFF system using the Cellular Topology Component of the ACIS geometric modeling kernel [10].

Two basic operations have been defined that modify the cellular model: adding a new feature shape to the cellular model, and removing an existing feature shape from the cellular model. The effect of these operations is twofold: (i) they change the topology of the cellular model, and (ii) they update the owner lists of its cellular entities accordingly. Both aspects are described and illustrated for the two operations in the following subsections.



cell 1 - <block>
cell 2 - <block, roundedRectPocket>
cell 3 - <block, throughSlot, roundedRectPocket>
cell 4 - <block, throughSlot>
cell 5 - <block, throughSlot>
cell 6 - <ribBack>
cell 7 - <ribFront>

Fig. 4. Cellular model of a product and owner lists of its cells.

4.2. Adding a feature shape to the cellular model

The goal of this operation is to add the imprint of a new feature shape to an existing cellular model.

In the first stage, the new feature is represented by a one-cell shape, which is dimensioned and positioned relative to the cellular model according to the feature parameters. In the owner list of this cell, only a reference to the new feature is contained, whereas, the owner list of each of its cell faces contains a reference to the respective feature face. This is illustrated in Fig. 5(a) for a rectangular slot feature.

In the second stage, a *non-regular cellular union* operation is performed, between the cell representing the new feature and the cellular model. This Boolean operation computes the cellular decomposition described in the previous subsection, and changes the owner lists of the relevant cells and cell faces, e.g. whenever these are split, so that each entity always 'knows' precisely which feature shapes, or feature faces, it belongs to.

During a non-regular cellular union, first, every cell (C_i) of the cellular model is identified that somehow intersects the new cell (C). Mutual cellular decomposition is then carried out between C and each C_i. This may occur in two ways:

(1) The two cells intersect only over their boundaries, in which case there are no new cells created; instead, their overlapping cell faces are decomposed, yielding partner cell faces that lie inside the boundary intersection (i.e. bounding both cells) and cell faces that lie outside it (i.e. bounding only one of the cells). Split cell faces get as owner list that of the cell face from which they originate.
(2) The two cells volumetrically overlap, in which case the decomposition results in new cells that lie either inside the intersection or outside it. Mostly, a subset of the boundary of the two cells is also decomposed (except when one of the cells lies entirely inside the other), yielding cell faces that lie either on the intersection or outside it. Whenever two cells undergo this mutual decomposition, the owner lists of the new entities are determined as follows:
   (a) a new cell that lies in the intersection of C and C_i gets as owner list the union of the owner lists of C and C_i;
   (b) the other cells resulting from the decomposition get as owner list that of the respective cell from which they originate (either C or C_i);

   and, analogously:
   (a) a new cell face lying on the boundary of both C and C_i gets as owner list the union of the owner lists of the overlapping cell faces from which it originates;
   (b) a new cell face lying on the boundary of either C or C_i gets as owner list that of the respective cell face from which it originates;
   (c) the remaining new cell faces (e.g. partner cell faces of the above) get an empty owner list.

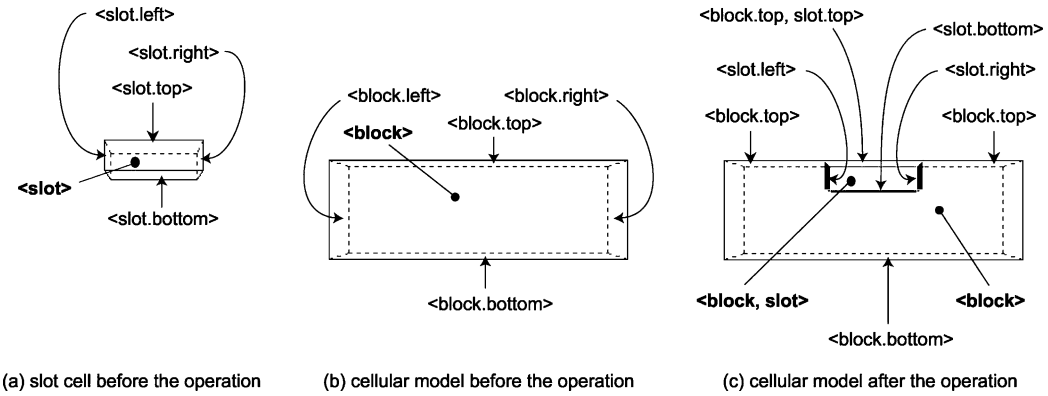


Fig. 5. Adding a feature shape to the cellular model.

Fig. 5(c) illustrates both the cellular decomposition and the owner list propagation after the non-regular cellular union between the slot feature in Fig. 5(a), and a cellular model consisting of a single block (see Fig. 5(b)), which is an example of case 2 mentioned above. The owner lists of both cells before the operation are shown in Fig. 5(a) and (b), though for the sake of legibility, only some cell face owner lists are depicted. After the operation, the block cell has been decomposed into two cells, of which one is shared with the slot. The owner lists of the two new cells, as well as the owner lists of some cell faces, are shown in Fig. 5(c).

4.3. Removing a feature shape from the cellular model

The goal of this operation is to completely remove from the cellular model the imprint of a feature. This operation comes down to a selective removal and merge of topologic entities in the cellular model, and is conceptually much simpler than a conventional Boolean operation. It is carried out in three stages, all of which are confined to the set of cells owned by the feature to be removed.

In the first stage, these cells are walked through in order to remove from their owner lists all references to that feature. Similarly, all references to faces of that feature are removed from the owner lists of the cell faces bounding those cells.

In the second stage, the same set of cells is searched for cells exhibiting an empty owner list. Such cells are removed from the cellular model, which is easily accomplished by removing all one-sided faces bounding them.

In the third stage, each of the remaining cells in the set is analyzed. Whenever it exhibits the same owner list as one of its adjacent cells, they are merged, which is easily accomplished by removing all two-sided faces that separate them. After all such cells have been merged, the cellular model has the minimal number of cells required to represent the remaining features, but it may still exhibit a non-minimal number of faces and edges. A final cleanup is therefore performed, merging any adjacent faces which have the same geometry and whose cell faces have equal owner lists.

Fig. 6 illustrates this operation for a simple part consisting of a block, a step and a rib (see Fig. 6(a)). Fig. 6(b) shows the corresponding cellular model, where its five cells are identified, together with the respective owner lists. When the rib is removed from the model, at the end of the first stage, the cell owner lists are those shown in Fig. 6(c). Cell 5, which was owned exclusively by the rib, has now an empty owner list and is therefore removed in the second stage, after which the cellular model shown in Fig. 6(d) is obtained. In the third stage, adjacent cells 2 and 3, having equal owner lists, are merged, and the same is done for cells 1 and 4, yielding the model in Fig. 6(e). The redundant faces left, highlighted in Fig. 6(e), are subsequently merged with their adjacent faces during the final cleanup, resulting in the final part and cellular model shown in Fig. 6(f) and (g), respectively.

5. Boundary evaluation for a cellular model

The boundary evaluation for the cellular model in SPIFF follows the general scheme given in Fig. 1. The boundary evaluator here produces a cellular model.

The most relevant information at the parametric definition level is structured around the Feature Dependency Graph (FDG), which relates all feature instances, each of them having its own set of parameter values, by means of the *dependency relation* [3]. A dependency is a directed relation between two features, and expresses an attach or positioning/orientation reference that a feature has to another feature in the model.

Adding a feature to the model typically creates one or more dependencies in the FDG between the new feature node and the feature nodes already in the graph. Removing a feature requires eliminating its node from the FDG, though not without first making sure that it has no dependents. Modifying an existing feature, in turn, is done

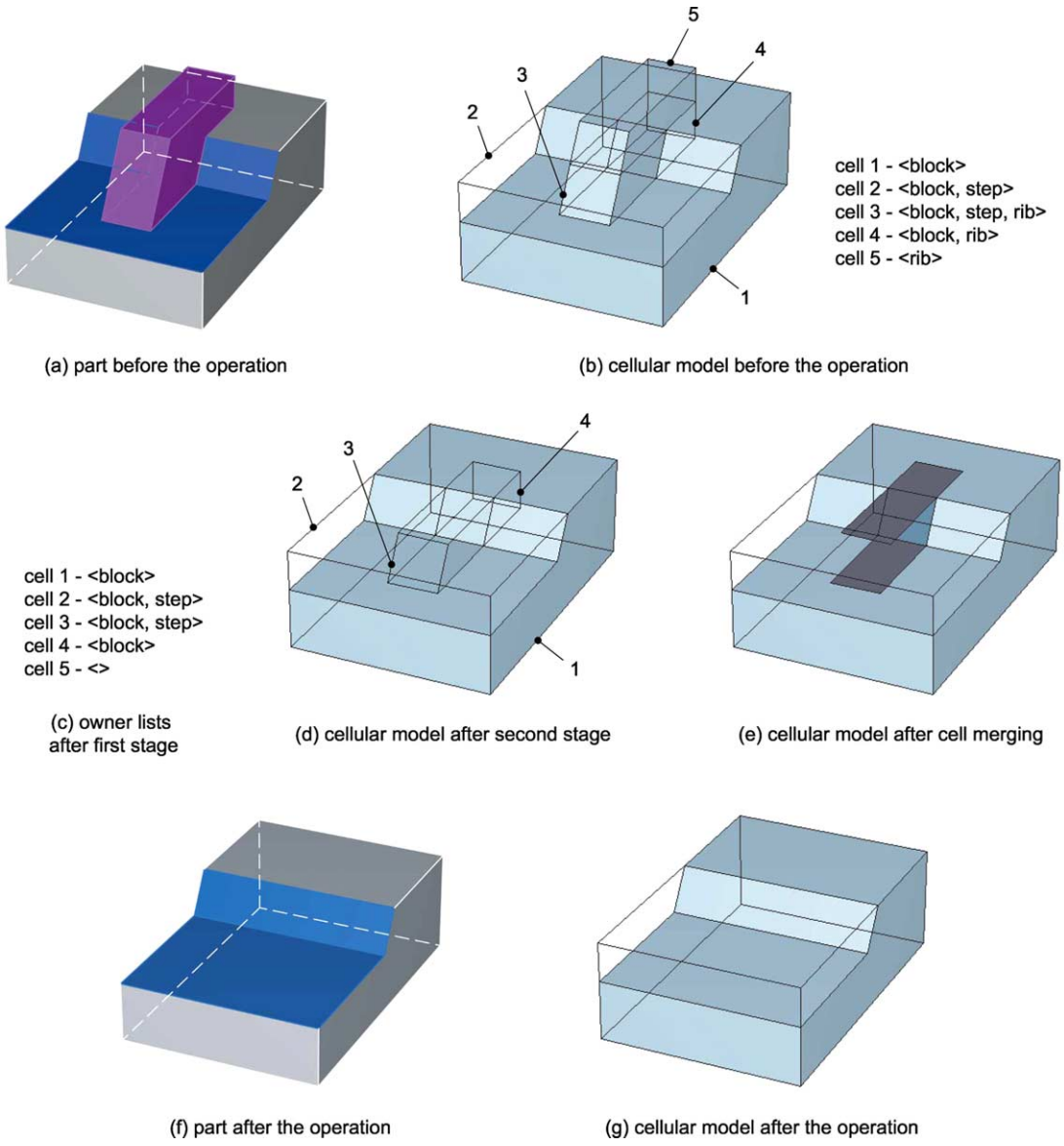


Fig. 6. Removing a feature shape from the cellular model.

by editing its parameters in the respective node in the FDG. Whenever this feature has dependent features, it is likely that the modification will affect some of them as well.

After the FDG has been modified, the boundary evaluator is invoked to update the cellular model accordingly. This is carried out in two phases. In the first phase, the cellular model is incrementally re-evaluated. In the second phase, the new cellular model is interpreted, i.e. the final shape it represents is determined, according to the feature information stored in its cellular entities, and the current dependencies among the features in the FDG.

5.1. Incremental re-evaluation

In contrast with the boundary evaluation described in Section 3, which employs two non-associative Boolean operations (union and difference) to compute the b-rep, only one Boolean operation is used by the SPIFF system to compute the cellular model from scratch: the non-regular cellular union of the shapes of all features (see Section 4). Since it is a union operation, and therefore, commutative and associative, the order in which the shapes are processed is irrelevant for the final cellular model obtained.

The computational cost of building the whole cellular model from scratch is proportional to the number of features in the model, as is the case for computing a b-rep. Fortunately, this is only required when the cellular model needs to be built in one step, e.g. when starting a modeling session with a model file containing a previously created parametric definition.

Once there is a cellular model, re-evaluating it after each modeling operation is done incrementally, i.e. only for the features whose geometry is actually involved in the operation. The computational cost of this incremental evaluation is dependent on the number of such features,

which is usually very low, and is, therefore, independent of the total number of features in the model. The three modeling operations are performed as follows:

(1) Adding a new feature instance to the cellular model: the shape extent of the new feature is combined with the current cellular model. For this, the non-regular cellular union operation is used, as described in Section 4.2.
(2) Removing a feature instance from the cellular model: the current cellular model is modified using the operation to remove a feature shape described in Section 4.3.
(3) Modifying a feature instance in the cellular model: the modified feature, and all its dependent features affected by the operation, need to be taken into account. They are removed from the cellular model and then re-added with their new parameters, using the add and remove feature operations just mentioned.

5.2. History-independent interpretation

Interpretation of the cellular model consists of determining, for each cell, whether the point set represented by the cell belongs to, or represents 'material' of, the product, i.e. determining the nature of that cell. This requires deciding which of the features in its owner list 'prevails', either as additive or as subtractive. To this purpose, precedences among features have to be established.

If, based on some precedence criteria, a global ordering is defined on the set F of all features in the model, say assigning to them unique, increasing precedence numbers, then every cell owner list (referring to a subset of F) can be sorted according to these precedence numbers. The nature of a cell becomes, then, the nature of the last feature in its owner list, i.e. the feature with the highest precedence number. Appropriate precedence criteria produce an interpretation of the cellular model that is unambiguously determined without invoking any model history considerations [3]. This, together with the incremental character of model evaluation, are important advantages of a cellular model.

Because this phase is based on simple owner list queries, and involves no geometric computations at all, its computational cost is negligible compared to that of the incremental re-evaluation, and will, therefore, be left out of the following complexity analysis.

5.3. Complexity analysis

The basic feature operations on the cellular model were detailed in Section 4, and incremental model evaluation after either adding, removing or modifying a feature instance was described in Section 5.1. In what follows, these operations are analyzed in terms of the required computation time and their associated computational complexity.

5.3.1. Add feature operation

Let us consider a model made up of n cells. Adding a new feature (represented by cell C) to the model is accomplished through a non-regular cellular union operation, as explained in Section 4.2.

The required computation time (t) can be decomposed into

$$t = t_{\text{int}} + t_{\text{dec}} + t_{\text{upd}},$$

where t_{int} represents the time associated with the identification of the n_{int} cells intersecting C, t_{dec} the time associated with the mutual cellular decomposition operations, and t_{upd} the computation time associated with the update of model information, given the n_{dec} cells resulting from the decomposition.

Regarding the number of cells involved, one can argue that:

- the time t_{int} is proportional to the total number of cells (n), since each cell has to be tested for intersection with the new cell C;
- the time t_{dec} is proportional to the number (n_{int}) of cells which intersect the new cell C, and which have to be properly decomposed;
- the time t_{upd} is proportional to the number (n_{dec}) of cells arising as a result of the performed cell decompositions.

Note that, after performing the above operations, the total number of cells increases and is given by

$$n_{\text{new}} = n + (n_{\text{dec}} - n_{\text{int}}),$$

where the subtraction corresponds to the substitution of original cells by their new updated counterparts. For simplicity, only the contribution of intersecting cells which overlap volumetrically is here being taken into account; cells which intersect only over their boundaries, as explained in Section 4.2, are handled in a significantly simpler way, and their contribution can here be neglected.

The required computation time can then be written in terms of the number of cells involved in the various operations as

$$t = \alpha \times n + \beta \times n_{\text{int}} + \gamma \times n_{\text{dec}},$$

where α, β, and γ are (constant) positive factors. Note that these factors express average behavior: in general, not all (intersected or generated) cells are equally complex, nor require the same computation time to be processed.

The two equations above will now be used to analyze the computation time for the best, average and worst cases of Section 2.

Best case. Clearly, in terms of the number of cells and the computation time for a model in the best case class, the new cell C intersects just one cell C_1 ($n_{\text{int}}=1$) and, afterwards, the overall number of cells is

$$n_{\text{new}} = n + 1,$$

with $n_{dec}=2$, i.e. two (new) cells result from the intersection, and one of them replaces (or updates) the former cell C_1. Regarding the computation time, we have then

$$t = \alpha \times n + \beta + 2 \times \gamma.$$

Therefore, it linearly increases with the number of cells making up the model.

Let us consider now the representative model of Fig. 2(a), which is being built up through a sequence of best case feature additions. After the insertion of the kth feature, the number of cells (n_k) is given by

$$n_k = n_{k-1} + 1,$$

with $n_0=1$ representing the (initial) model defined by just one cell, and the required computation time for that operation is

$$t_k = \alpha \times n_{k-1} + (\beta + 2 \times \gamma).$$

The above equations can be further simplified to

$$n_k = k + 1, \quad k = 0, 1, \ldots, 100,$$

and

$$t_k = \alpha \times k + (\beta + 2 \times \gamma) = \alpha \times k + \Delta_B,$$

$$k = 1, 2, \ldots, 100.$$

Thus, both the overall number of cells and the computation time associated with adding a feature have, in this case, linear complexity order.

Average case. In terms of the number of cells and the computation time for a model in the average case class, the cell C associated with the added feature intersects r cells C_i ($n_{int}=r$) and, afterwards, the overall number of cells becomes

$$n_{new} = n + (n_{dec} - r).$$

Regarding the computation time, we have then

$$t = \alpha \times n + \beta \times r + \gamma \times n_{dec}.$$

As explained in Section 2, the number of intersections of the added feature with other features is independent of their number (n). Therefore, the number of intersected cells (r) is independent of n, and knowing that n_{dec} depends on r, it can be concluded that the computation time required by an add feature operation for the average case has linear complexity order, similarly to the best case.

In order to better understand the average behavior, a detailed analysis pertaining to the representative model of Fig. 2(b) will be given in what follows. Regarding that model, any added through hole feature intersects a rib and a slot, as well as the block, in such way that:

- the hole feature is decomposed into three cells, each one of them being shared with either the block, the rib or the slot;
- the single cells associated with the rib and with the slot, as well as the block 'main cell', are each decomposed into two cells, one being shared with the hole feature.

Therefore, for this particular model, three cells are intersected by the new feature ($r=3$) and six cells result from the decompositions ($n_{dec}=6$).

The overall number of cells is then, after adding the new hole feature,

$$n_{new} = n + 3$$

and, the computation time for such operation is

$$t = \alpha \times n + 3 \times \beta + 6 \times \gamma.$$

Similarly to the best case, this time linearly increases with the number of cells making up the model.

Let us consider now that the model is being built up by a sequence of add feature operations, corresponding to successively adding a rib, a slot and a cylindrical through hole, 33 times. Note that adding just a rib and a slot feature increases by two the overall number of model cells.

Similarly to what was done for the best case, after the insertion of the kth hole feature, the total number of cells is given by

$$n_k = n_{k-1} + 5,$$

with $n_0=1$. This recursive equation can be written in closed form as

$$n_k = 5 \times k + 1, \quad k = 1, 2, \ldots, 33.$$

For the computation time associated with the insertion of the kth hole feature, we can now write

$$t_k = \alpha \times (n_{k-1} + 2) + 3 \times \beta + 6 \times \gamma,$$

assuming that a rib and a slot were already added and, thus, two additional cells were already created. This can be further simplified to

$$t_k = 5 \times \alpha \times k - 2 \times \alpha + 3 \times \beta + 6 \times \gamma$$

$$= 5 \times \alpha \times k + \Delta_A,$$

$$k = 1, 2, \ldots, 33.$$

Thus, both the overall number of cells and the computation time associated with adding a hole feature have, as expected, linear complexity order.

Worst case. In terms of the number of features and computation time for a model in the worst case class, the insertion of a feature corresponds to a situation, where the new feature intersects a fraction of the features making up the model, entailing a large increase in the number of cells defining the model.

In order to better understand the worst case behavior, as well as the fundamental difference regarding the previous cases, a detailed analysis pertaining to the representative model of Fig. 2(c) will be given in what follows. Remember that this model has an even number (40) of cylindrical through hole features: any of the 20 horizontal, respectively vertical, hole features intersects once each one of the other

20 vertical, respectively horizontal, hole features, as well as 21 times the block.

In order to understand how the number of cells grows after adding a new feature, let us analyze how such a model can be built through a sequence of add feature operations, starting with just one block:

- the initial model has just one cell: $n_0 = 1$;
- adding the first feature (say a vertical through hole) results in a model made up of two cells: $n_1 = 2$;
- adding the second feature (a horizontal through hole) intersecting the first one, both holes now being divided into three cells each (the 'middle' one being shared by both holes), results in a model made up of six cells: $n_2 = 6$.

Consider now that several features are successively added to the model in this alternate way. When the k^{th} hole feature is added to the model, it intersects (k div 2) other features. In terms of the cellular operations, it is easy to see that:

- the new hole feature is decomposed into $(2 \times (k \text{ div } 2) + 1)$ cells, (k div 2) of them being shared with one of the intersected hole features, the other $(1 + k \text{ div } 2)$ being shared with the block;
- each of the (k div 2) intersected hole feature cells is decomposed into three 'new' cells, one of those being shared with the added hole feature;
- the cell associated with the block is decomposed into $(2 + k \text{ div } 2)$ cells, $(1 + k \text{ div } 2)$ of them being shared with the added hole feature.

Therefore, $r = 1 + k$ div 2 cells are intersected by the added feature, and $n_{\text{dec}} = 4 \times (k \text{ div } 2) + 2$ cells result from the decompositions.

The overall number of cells is then, after adding the kth feature,

$$n_k = n_{k-1} + 3 \times (k \text{ div } 2) + 1, \quad k = 2, 3, \ldots, 40,$$

and the computation time for such an operation is

$$t_k = \alpha \times n_{k-1} + \beta \times (k \text{ div } 2 + 1) + \gamma \times (4 \times (k \text{ div } 2) + 2),$$

$$k = 2, 3, \ldots, 40.$$

Note that the number of cells no longer increases by just a constant factor, but by a factor proportional to the number of features in the model: when inserting the kth feature, the number of cells added to the model is approximately three times the number of intersected cells, which in turn is linearly dependent on k.

After some algebraic manipulation, we get a closed formula for the number of cells after inserting the kth feature:

$$n_k = \frac{3}{4}(k+1)k + (k+1) - \frac{3}{2}[(k+1) \text{ div } 2],$$

$$k = 0, 1, \ldots, 40.$$

The number of cells is proportional to the square of the number of inserted features and has, thus, $O(n^2)$ computational complexity.

Therefore, contrary to the best and average cases, both the overall number of cells and the computation time associated with adding a new feature have quadratic complexity order.

5.3.2. Remove feature operation

Let us consider a model made up of m features and n cells Removing from the model a feature F (represented by the set C_F of cells owned by that feature) is accomplished through a selective sequence of operations performing the deletion and merge of topologic entities, as explained in Section 4.3.

Since a feature removal operation is explicitly accomplished in three stages, the required computation time (t) can be decomposed into

$$t = t_{\text{rem}} + t_{\text{empty}} + t_{\text{merge}},$$

where t_{rem} represents the time associated with the removal of all references to F (and its faces) from the owner lists of the cells in C_F (and their cell faces), t_{empty} represents the time associated with the removal from the cellular model of all cells in C_F which have now an empty owner list (i.e. cells which were owned exclusively by feature F), and t_{merge} is the time associated with processing the remaining cells in C_F and, where needed, performing cell and face merging operations in the cellular model.

Regarding the number of cells involved, we can legitimately assume that:

- the time t_{rem} is proportional to the number of cells in C_F, n_{C_F}, or, more precisely, to the total number of elements in their owner lists, n_{owners};
- the time t_{empty} is (again) proportional to the number of cells in C_F;
- the time t_{merge} is proportional to the number of the remaining cells in C_F, n_{remain}.

Similarly to what was done before, and using (different) constant factors to express average behavior, the former equation can be written as

$$t = \alpha \times n_{\text{owners}} + \beta \times n_{C_F} + \gamma \times n_{\text{remain}}.$$

Although the above equation is similar to the corresponding equation for the add feature operation, it has to be remarked that:

- the three terms are (in general) no longer proportional to the overall number of cells in the model, but instead to

the (smaller) number of cells associated with feature F, n_{C_F};
- when no cells with empty owner lists have to be removed, $n_{\text{remain}} = n_{C_F}$;
- instead of an increase in the number of cells, after performing the feature removal, a decrease in the overall number of cells occurs.

Given the expression above, the computation time of the remove feature operation will now be analyzed for the best, average and worst cases of Section 2.

Best case. To illustrate the best case situation for feature removal, we again use the representative model of Fig. 2(a): when removing a hole feature F, in this case defined by just one cell C_h, set C_F has just one element ($n_{C_F} = 1$), which is associated with the feature being removed and the block ($n_{\text{owners}} = 2$).

In the first step, the owner list of C_h has two elements and the reference to F is removed from it. Since, in the second step, the owner list of C_h is not empty, cell C_h remains in C_F. Finally, in the third step, cell C_h is merged with the block 'main cell'.

The equation representing the required computation time can then be written as

$$t = 2 \times \alpha + \beta + \gamma.$$

Note that, since all hole features are disjoint and of the same type, removing any of these features has the same computational cost.

Average case. For a model in the average case class, set C_F has a small number of elements, which is independent of n. To illustrate such a situation, the representative model of Fig. 2(b) is used.

When removing a hole feature F, in this case defined by the three cells C_{hb}, C_{hr} and C_{hs}, shared with the block, the rib and the slot, respectively, set C_F has three elements ($n_{C_F} = 3$). The three owner lists of these cells have a total of seven elements ($n_{\text{owners}} = 7$).

In the first step, the reference to F is removed from the three owner lists, and none of them becomes empty; therefore, no cell is removed from C_F in the second step. Subsequently, in the third step, the three cells C_{hb}, C_{hr} and C_{hs} and their faces are appropriately merged to define the new cells associated with the block, the rib and the slot.

The equation representing the required computation time can then be written as

$$t = 7 \times \alpha + 3 \times \beta + 3 \times \gamma.$$

Similarly to the best case model, the same computation time is required for removing any of the hole features, since they all have the same kind of interaction with the block, a rib and a slot.

Worst case. For a model in the worst case class, the number of features intersecting the feature F being removed is a fraction of the overall number of features, and thus the set C_F has a larger number of elements. To illustrate such a situation, the representative model of Fig. 2(c) is used.

When removing a hole feature F, in this case defined by 41 cells, shared with the block and the intersected holes, set C_F has 41 elements ($n_{C_F} = 41$). The 41 owner lists of these cells have a total of 102 elements ($n_{\text{owners}} = 102$).

In the first step, the reference to F is removed from the owner lists of the 41 cells in C_F, and none of the lists becomes empty; therefore, no cell is removed from C_F in the second step. Subsequently, in the third step, the cells and their faces are appropriately merged to define the new cells associated with the block and each of the formerly intersected holes.

The equation representing the required computation time can then be written as

$$t = 102 \times \alpha + 41 \times \beta + 41 \times \gamma.$$

Note that all steps associated with the removal have, in this case, a significantly higher computational cost than for the best and average case models, due to the number and characteristics of the involved cells.

Similarly to the previous cases, the same computation time is required for removing any of the horizontal or vertical through hole features, since each hole intersects in the same way the remaining features.

5.3.3. Modify feature operation

Modifying a feature F from a given model is accomplished by identifying the features affected by the modification (F and, possibly, some of its dependent features), which are removed from the model and re-added with their new parameters, as mentioned in Section 5.1.

Therefore, a modify feature operation corresponds to the sequential execution of remove and add feature operations, and its total computation time is the sum of the corresponding partial computation times.

Let us consider again the representative models described in Section 2, and assume that one of the cylindrical hole features is modified in the following way: it undergoes a small translation, but without entailing any topologic changes in the feature interactions and in the associated cellular decompositions. Moreover, none of the other features is dependent on the hole feature, remaining therefore unaffected by such a translation: only one hole feature removal and a subsequent hole addition operations are performed for such a modification.

In this way, modifying one such hole feature, in any of the models having m features, requires the time corresponding to the removal step plus the time required for re-adding the modified hole to a model consisting of $(m-1)$ features. Both these times have been derived above for the best, average and worst cases, and turned out to be constant in all cases. So, for the three cases, modifying any feature has again constant computational cost.

6. Performance measurements

To complement the complexity analyses, given in the previous sections, of boundary evaluation for a b-rep and for the cellular model, their performance was measured. In this section the outcome of the measurements is presented and interpreted.

6.1. Setup used for the measurements

The performance measurements for boundary evaluation for a b-rep were done with the commercial system Pro/ENGINEER [8]. Measuring evaluation times in this system was more complex than in SPIFF, because the source code of Pro/ENGINEER could not be adapted for this. However, Pro/ENGINEER uses a so-called trail file to record all user actions during a modeling session. Such a trail file can subsequently serve as a script file for another modeling session, possibly after changing it. By including timestamp commands in a trail file around the boundary evaluation step in the script, and executing this file, the evaluation times could be measured. The measurements were done with Pro/ENGINEER 2001 running under Windows 2000 on an Intel Pentium 2 processor.

The performance measurements for boundary evaluation for the cellular model were done with the prototype feature modeling system SPIFF. Measuring evaluation times could simply be done by including timers in the source code of the system around its boundary evaluation algorithm. SPIFF is running under Linux, and the measurements were done on an Intel Pentium 4 processor.

A major problem was that, because of the measuring unit of the timing commands, the measured CPU times had an inadequate precision compared to their order of magnitude. Therefore, all measurements were done several times, and the results averaged. With some elementary statistical theory, it was determined that 100 measurements were needed to guarantee sufficient accuracy of the average time.

Because the measurements for the two approaches had to be performed on two different computer systems, the CPU times were not directly comparable. To make them more congruent, a simple benchmark was run on both systems to compare their performance in general. This resulted in a normalization factor that was applied to the measured times for Pro/ENGINEER. Notice, however, that although the normalized times for Pro/ENGINEER are mostly in the same order of magnitude as the times measured for SPIFF, care should be taken in comparing their absolute values. They are still dependent on, for example, the available memory, the operating system, and the degree of optimization of the implementation of the modeling software. However, the goal here is not to compare the two approaches in absolute computation times, but, instead, to compare trends in their behavior. The times presented here are perfectly suitable for that purpose.

6.2. Results of the measurements

Computation time was measured for several models; for three of those, already introduced in Section 2, representing best, average and worst case behavior, the results are presented here. For other models, similar results were obtained.

For each model, times required for a series of add, remove and modify feature operations were measured. For the modify operation, a slight (hole) feature displacement was performed, as described in Section 5.3, so that no topology changes occurred in the model. The results are presented in Fig. 7 for the add feature operation with both Pro/ENGINEER (on the left) and SPIFF (on the right), and in Fig. 8 for the remove and modify feature operations, again with both Pro/ENGINEER (on the left) and SPIFF (on the right). The times for the remove and modify feature operations, for one model on one system, were merged in a single graph, in order to facilitate the comparison of their related magnitudes.

The six graphs in each figure show (normalized) computation times for the series of add, remove and modify feature operations. Depending on the operation, for position n on the horizontal axis of each graph, the computation time is depicted to respectively:

(1) add feature n to the model consisting of features 1 to $n-1$;
(2) remove feature n from the complete model;
(3) modify feature n in the complete model.

6.3. Analysis of the experimental results

In this subsection, the measurement results will be analyzed, and trends in the presented graphs will be deduced and, where possible, inter-related. The comparison of these results to the outcome of the complexity analyses given in Sections 3.2 and 5.3 will be left to the final section.

6.3.1. Add feature operation

For position n on the horizontal axis of the add feature operation graphs (Fig. 7), the computation time is displayed for adding feature n to the model consisting of features 1 to $n-1$.

In both systems and for all cases, the cost of adding a feature to the model is in the same order of magnitude, and it increases with the number of features already in the model. For the best and average case models, Fig. 7(a) and (b), this cost increases linearly, whereas, for the worst case model, Fig. 7(c), it increases quadratically. For the first two cases, the cost of adding a feature is always very low, showing that both systems scale well when building models with a considerable number of features.

In the worst case, however, the cost of adding a feature to the model significantly increases as the number of intersections per feature grows from 0 up to 20. For

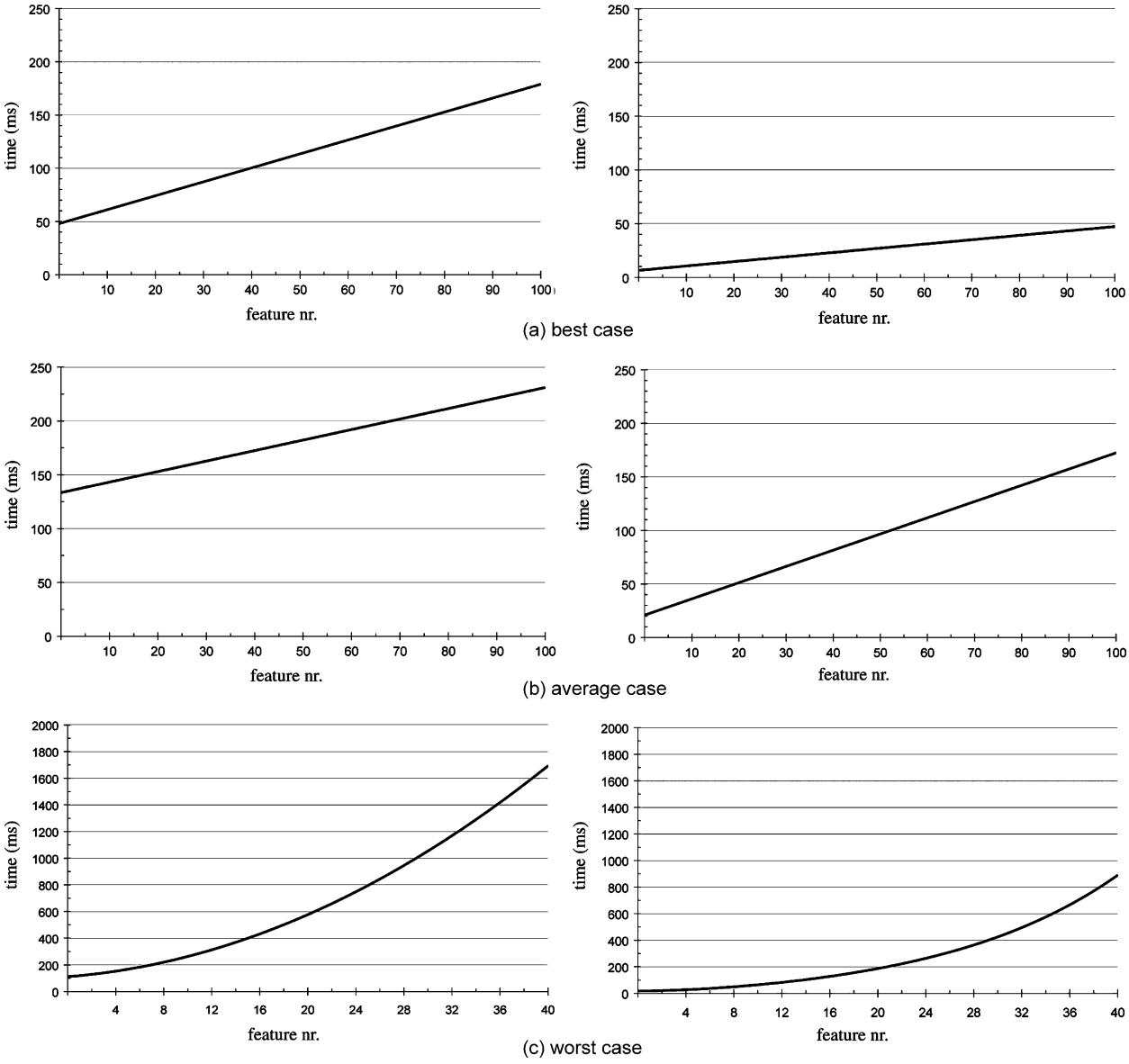


Fig. 7. Measurements of boundary evaluation time for adding a feature in Pro/ENGINEER (left) and SPIFF (right).

Pro/ENGINEER, this cost increases rather rapidly: when the 8th hole feature is added (i.e. when each hole has four intersections), the maximum cost of an add feature operation on the average case model (229 ms) is surpassed. For SPIFF, on the other hand, only when the 18th hole feature is added (i.e. when each hole has already nine intersections) is the maximum cost of an add feature operation on the average case model (172 ms) reached. As the number of hole intersections further increases, the cost of adding each new hole becomes considerably higher, for both systems.

6.3.2. Remove feature operation

For position n on the horizontal axis of the remove feature operation graphs (lighter lines in Fig. 8), the computation time is displayed for removing feature n from the respective complete model.

In Pro/ENGINEER, the cost of removing a feature (on the left of Fig. 8) is generally much higher than the cost of adding a feature (on the left of Fig. 7). The lowest cost for a feature removal operation is for removing the last feature of the model (i.e. the most recent one in the model history); it costs, respectively, around 80 ms (best case), 80 ms (average case) and 770 ms (worst case). Considering these figures, one can wonder which procedure is actually being applied in Pro/ENGINEER for re-evaluating the model. If full intermediate models were being stored between each step of the model history, one would expect that removing this last feature, in whichever model, would mean to simply revert to the intermediate model lastly stored, and thus would have a constant, very low cost. The values above suggest the application of some other history re-evaluation procedure,

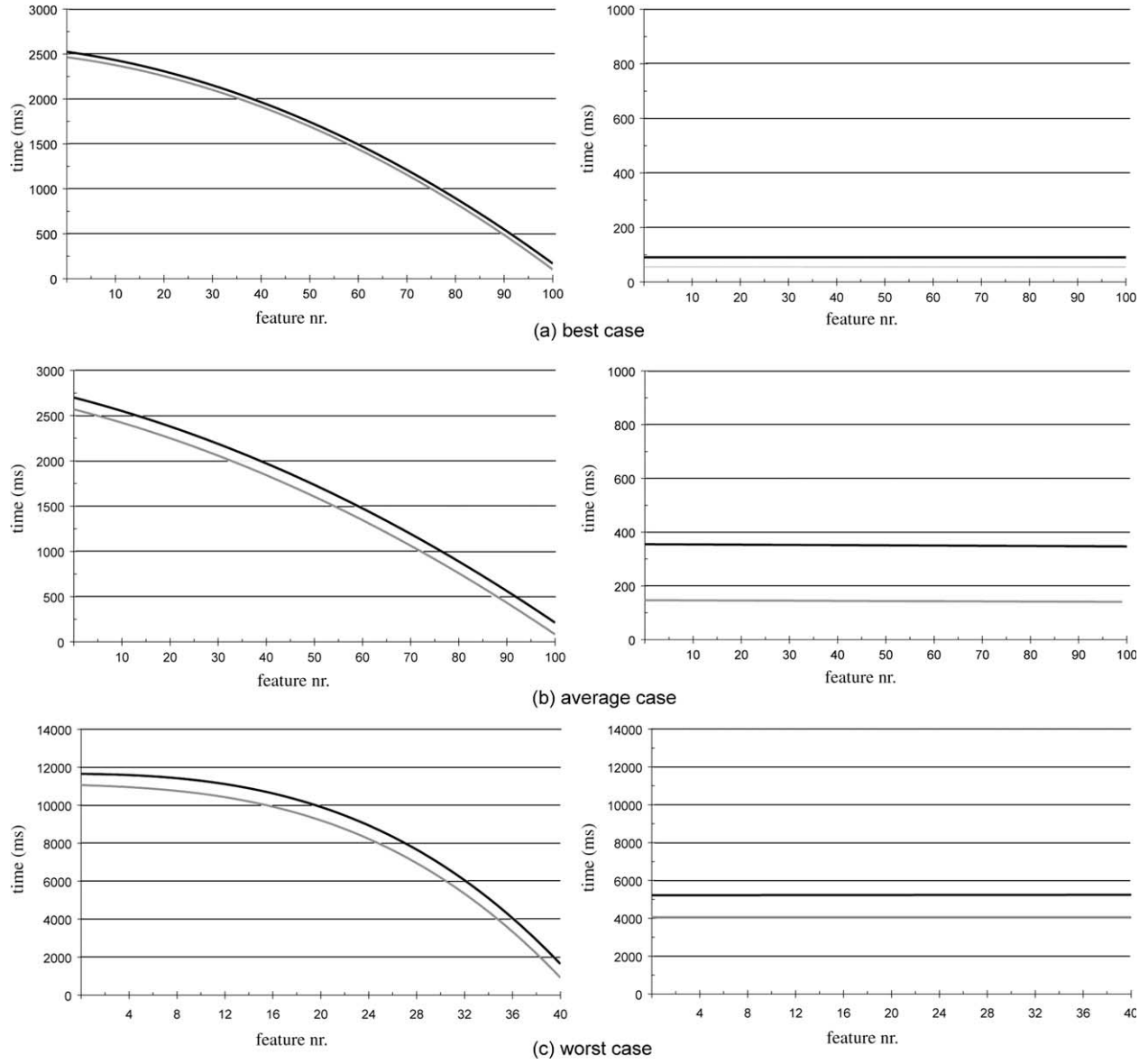


Fig. 8. Measurements of boundary evaluation time for removing (light line) and modifying (dark line) a feature in Pro/ENGINEER (left) and SPIFF (right).

e.g. storing only the *deltas* between history steps. Then, reverting the model, even for a single step, would have a cost which is dependent on the complexities of both the *delta* and the model, which seems to be the case.

In SPIFF, removing a feature, regardless of whether it is the last or not, has a constant cost which is, respectively, around 55 ms (best case), 150 ms (average case) and 4030 ms (worst case).

We can conclude that, for the best and average case, the cost of this operation on the last feature of the model is in the same order of magnitude for both systems; however, for all other features, that cost is significantly higher in Pro/ENGINEER (up to one order of magnitude), whereas, it is constant in SPIFF. For the worst case, the same trends hold, but the constant cost of this operation in SPIFF is one order of magnitude higher than that in the average case, meaning that the remove feature operation is more sensitive to a high number of intersections on the feature in question than the add feature operation.

6.3.3. Modify feature operation

For position n on the horizontal axis of the modify feature operation graphs (darker lines in Fig. 8), the computation time is depicted for modifying feature n in the respective complete model.

As one might reasonably expect, in both systems and for all cases, the cost of modifying a given feature is always higher than the cost of removing that same feature from the model. Particularly, for SPIFF, we can observe that the cost of modifying a given feature (on the right of Fig. 8) is always slightly above the sum of the cost for removing it

(also on the right of Fig. 8) and the cost of adding the last feature in the model (on the right of Fig. 7).

For Pro/ENGINEER, the cost of modifying a feature (on the left of Fig. 8) is generally much higher than the cost of adding a feature (on the left of Fig. 7). However, a surprising result of Pro/ENGINEER in all cases is that the cost of modifying the *last* feature of the model (i.e. the most recent one in the model history) is *always* (slightly) lower than the cost of adding that same feature to the model. As stated in Section 1, we have insufficient information on Pro/ENGINEER to explain such behavior. The lowest cost for a modify operation is for the last feature of the model (i.e. the most recent one in the model history), which takes, respectively, around 120 ms (best case), 240 ms (average case) and 1340 ms (worst case). One should also notice that all graphs on the left of Fig. 8 are concave. In other words, when it comes to re-evaluate a model history consisting of similar steps, each of the more recent steps has a higher cost than any of the older steps. This increase in 'cost per step' towards the end of the model history is stronger in the worst case than in the best case, which can be attributed to the increased complexity of the model as more recent history steps are evaluated.

In SPIFF, modifying a feature, regardless of whether it is the last or not, takes a constant time, which is, respectively, around 90 ms (best case), 360 ms (average case) and 5230 ms (worst case).

The conclusions regarding the trends of the modify feature operation in the two systems are, *mutatis mutandis*, the same as for the remove feature operation.

7. Conclusions

Boundary evaluation for a cellular model has been described, and its efficiency has been compared to the efficiency of boundary evaluation for a b-rep. The computational complexity of boundary evaluation algorithms for both representations has been analyzed, and figures emerging from performance measurements of boundary evaluation for both representations have been presented.

The outcome of the complexity analyses, given in Sections 3.2 and 5.3, has been confirmed by the measurements performed for each feature operation, shown in the graphs of Figs. 7 and 8. In other words, the same trends arise from the analyses and from the measurements. These trends are summarized in Table 1, for both representations and for all operations and cases. Note that, in this table, the trends for the add operation reflect the evolution of the cost while building the whole model feature by feature, whereas, the trends for the remove and modify operations refer to the cost of these operations while applying them on each individual feature of the whole model.

The complexity analyses showed that the computational cost of boundary evaluation for each operation on a feature, is basically dependent on two factors: (i) the overall number of topologic entities in the model, i.e. the model size, and (ii) the number of intersections that the feature in question has. Furthermore, from the three cases analyzed, one can observe that these two factors are not necessarily independent: in the best and average case models, it is the model size that plays the predominant role in the computational cost, as the number of intersections per feature is kept limited; in the worst case model, on the other hand, it is the increase in the number of intersections per feature that drives the growth in model size, and therefore also in the computational cost.

Based on the outcome of the measurements on these different models, the following can be concluded for the cellular model, with respect to all feature operations:

(1) Regarding the factor 'model size' alone, boundary evaluation scales very well for large models.
(2) Regarding the factor 'number of intersections', boundary evaluation scales well up to a moderate number of intersections per feature, its cost remaining much lower than for the b-rep; for a high number of feature intersections, however, its cost can get close to the order of magnitude of that for the b-rep.

From a practical point of view, the average case provides the most relevant information. The number of intersections of each feature in an average case model is limited, which is also typically the case in real-world models, because of the local character of features: usually, features are small compared to the whole product, and intersect only with a small number of other features. Therefore, on the basis of these observations, the following can be concluded:

(1) For the add feature operation, boundary evaluation for the cellular model has the same performance trend as boundary evaluation for a b-rep; for both, the time linearly increases with the number of features already in

Table 1
Trends of boundary evaluation for b-rep and cellular model

Operation	Best case		Average case		Worst case	
	b-rep	Cellular model	b-rep	Cellular model	b-rep	Cellular model
Add feature	Linear	Linear	Linear	Linear	Quadratic	Quadratic
Remove feature	Quadratic	Constant	Quadratic	Constant	Cubic	Constant
Modify feature	Quadratic	Constant	Quadratic	Constant	Cubic	Constant

the model, and the times are in the same order of magnitude.

(2) For the remove and modify feature operations, however, the trends are different. Boundary evaluation for the cellular model has a constant cost that is in the same order of magnitude as the cost of an add feature operation in the same model. Boundary evaluation for a b-rep, on the other hand, has a cost that is dependent on the position in the model history of the feature being removed or modified; this cost follows a quadratic trend and is very high compared to the cost of an add operation.

Usually, most of the time during a modeling session is spent on adjusting features already in the model, rather than on adding new features. The overall conclusion is therefore that boundary evaluation for the cellular model is in practice more efficient than boundary evaluation for a b-rep. Considering this conclusion, and the applications and advantages of cellular models mentioned in Section 1, it can be expected that in the future such models will be increasingly used in feature modeling systems.

Acknowledgements

We thank Klaas Jan de Kraker for many valuable comments on a previous version of this paper.

References

[1] Shah JJ, Mäntylä M. Parametric and feature-based CAD/CAM. New York: Wiley; 1995.
[2] Bidarra R, de Kraker KJ, Bronsvoort WF. Representation and management of feature information in a cellular model. Comput Aided Des 1998;30(4):301–13.
[3] Bidarra R, Bronsvoort WF. Semantic feature modelling. Comput Aided Des 2000;32(3):201–25.
[4] Bronsvoort WF, Bidarra R, Noort A. Feature model visualization. Comput Graphics Forum 2002;21(4):661–73.
[5] Bronsvoort WF, Noort A. Multiple-view feature modelling for integral product development. Comput Aided Des 2004;36(10):929–46.
[6] Requicha AAG, Voelcker HB. Boolean operations in solid modeling: boundary evaluation and merging algorithms. Proc IEEE 1985;73(1):30–44.
[7] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. 2nd ed. Cambridge, MA: The MIT Press; 2001.
[8] Parametric Technology Corporation, Pro/ENGINEER product information, http://www.ptc.com/ (2003).
[9] Bidarra, R, Neels, WJ, Bronsvoort, WF. Boundary evaluation for a cellular model, in: CD-ROM Proceedings of the 2003 ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference, 2–6 September, 2003.
[10] Spatial Corp., 3D ACIS Modeler, Version R11, http://www.spatial.com (2003).

Rafael Bidarra is assistant professor Geometric Modeling at the Faculty of Electrical Engineering, Mathematics and Computer Science of Delft University of Technology, The Netherlands. He graduated in electronics engineering at the University of Coimbra, Portugal, in 1987, and received his PhD degree from Delft University of Technology in 1999. His current research interests include semantic feature modeling, collaborative feature modeling and geometric reasoning. He has published many papers in international journals, books and conference proceedings, and has served as member of several program committees.

Joaquim Madeira is assistant professor at the Department of Electronics and Telecommunications of the University of Aveiro, Portugal. He graduated in electrical engineering in 1986 and received a MSc degree in computer science in 1991, both from the University of Coimbra, Portugal. In 1998 he received his PhD degree in computer science from Darmstadt University of Technology, Germany. His current research interests in the area of geometric modeling include modeling and visualization using polygonal meshes, in particular in medical applications, and algorithm efficiency.

Willem J. Neels is a MSc student at the Faculty of Electrical Engineering, Mathematics and Computer Science of Delft University of Technology, The Netherlands. He is scheduled to graduate in computer science in 2005, with a project on feature recognition for multiple-view modeling, at the Computer Graphics and CAD/CAM Group.

Willem F. Bronsvoort is associate professor CAD/CAM at the Faculty of Electrical Engineering, Mathematics and Computer Science of Delft University of Technology, The Netherlands. He received his MSc degree in computer science from the University of Groningen in 1978, and his PhD degree from Delft University of Technology in 1990. His main research interests are geometric modeling, in particular display and mesh generation algorithms, and feature modeling, in particular semantic feature modeling, multiple-view feature modeling, and freeform feature modeling. He has published numerous papers in international journals, books and conference proceedings, has served as co-chair and member of many program committees, and is book review editor of Computer-Aided Design.