A SEMANTIC BLEND FEATURE DEFINITION

Paulos J. Nyirenda, Rafael Bidarra and Willem F. Bronsvoort

[p.j.nyirenda/a.r.bidarra/w.f.bronsvoort]@tudelft.nl

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, NL-2628 CD Delft, The Netherlands

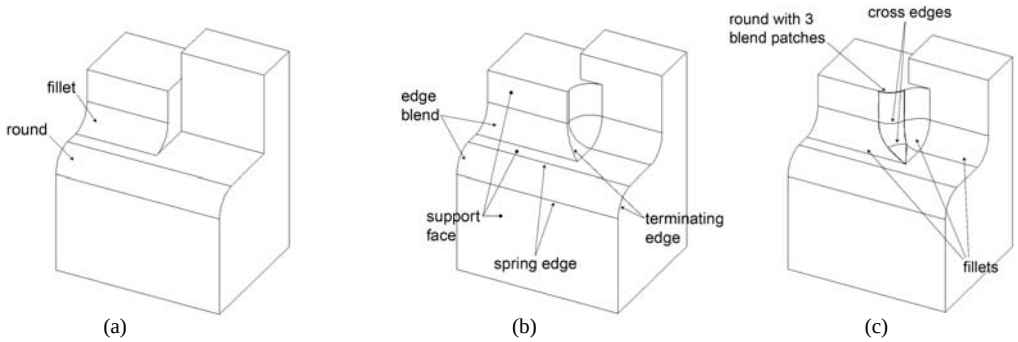
ABSTRACT

In most current feature modelling systems, blends are modelled as a kind of features too. These systems use a boundary representation (Brep) to describe the shape of a product. It is shown that the practice of basing specification of blends on conventional Breps is ineffective.
A solution to this problem is introduced in which blends are described at a higher level. Blends are defined in a feature class, and instances of such a class can be added to a feature model containing relations between all features in the model. Such a semantic description of blends has several merits. In particular, constraints can be defined on them to specify important functional properties, and these properties can be maintained during the whole modelling process through validity maintenance. A new method to determine the shape of a blend, based on its properties, is also demonstrated.

Keywords: blends, feature modelling, feature classes, semantics, validity maintenance.

1. INTRODUCTION

Feature modelling has, in the past decade, become the most popular means of product modelling. Previously, a product model contained only geometric information, but with the increasing use in different application areas, e.g. design, manufacturing and assembly, the need for additional functional information became apparent. A *feature* is the fundamental design component of feature modelling. Example features include ridge, hole, pocket and blend. A *feature model* stores the different features and their associated information. It is imperative for a feature to have a well-defined meaning, or *semantics*, for a particular lifecycle activity [1].
Feature technology still lacks in several ways. A particular shortcoming that this paper addresses relates to semantics for the blend feature, or blend for short. A blend is a sheet or volume used to smooth an intersection between two adjacent face boundaries of one or more features. The problem is that semantics for the blend feature are poorly defined. This limits the capability of capturing design intent in the feature model.
A blend feature is *attached* as a transition between one or more feature faces in the feature model. During modelling, features are modified and each change is associated with an evaluation of the model. A blend feature is always evaluated when a feature it is attached to is modified. This evaluation requires a good 'book-keeping' facility by means of which the semantics of the blend can be checked reliably whenever the feature model is going to change. Lack of such a facility disallows the feature modelling system to maintain the validity of the model. This has led to unreliable updates in blend features. In this paper, a new method to define blend features, in a way that facilitates specification of semantics and validity maintenance, is developed and demonstrated. Tests have been performed on SPIFF, a prototype feature modelling system developed at Delft University of Technology.
The object-oriented concept of a feature class will be used to define blends. A feature class is a parameterized description of the shape and the properties of the feature, which should be satisfied by all instances of the class. All its properties are described as constraints in the corresponding feature class. Parameters (e.g. radius) specify the shape of the blend, and constraints are restraints declared on the shape, e.g. the maximum and minimum radius value permitted. *Methods* in the feature class define the functionality of the blend, e.g. *how* the shape of a blend instance is determined. The most popular method used to create blends is the classic rolling-ball technique, in which a notionally *rolling ball* generates a surface envelope describing the blend [2, 3]. In this paper, a new approach to create a blend is presented. The blend is determined from the information in the corresponding feature class.
The remainder of the paper is outlined as follows. In Section 2, fundamental concepts on blending are reviewed. A blend as a feature is discussed in Section 3, and in Section 4 the new approach to modelling blends is introduced. In Section 5, the



(a) (b) (c)

Fig. 1: Blend geometry description

definition of a blend feature class is presented, and in Section 6, the instantiation of blend features is demonstrated with examples based on the rolling-ball technique. Section 7 demonstrates other types of blends, and Section 8 concludes the paper.

2. FUNDAMENTAL CONCEPTS

Most industrial products have blends. A blend is a softened sharp edge or vertex that instates a smooth transition between faces to improve, for example, manufacturability, strength, and/or pleasing look. Different literature uses various terms to describe blends. This section reviews common terms and concepts that we use in this paper.

Blends may be categorized by construction method, e.g. rolling-ball, spine-based or trimline-based. Fine surveys of these blending techniques, including various mathematical forms, can be found in [2] and [3]. In this section, we discuss the basic concepts of blends using the most widely used method of rolling-ball blends. Note that these concepts are sufficiently general and may be applied to other types of blends.

An operation used to create a blend is known as *blending*. Blending may either remove or add material, depending on the convexity of the model local to the blend. A blend on a convex edge *removes* material from the model to round off the edge, whereas a blend on a concave edge *adds* material to the model. The former is known as a *round* whereas the latter is a *fillet* – see Fig. 1a. Another blend type, a chamfer, is also popular. The key difference is that a chamfer is planar, in contrast with curved faces used in fillets and rounds. In this paper, the term *blend* applies to any of these.

A blend can be either an *edge blend* or *vertex blend*. An edge blend replaces an edge by a face tangent to the two faces adjoining the edge, except a chamfer in which the connection is non-tangential. A vertex blend replaces a vertex by a face connecting the faces adjoining the vertex. This kind of blend is obtained from blended edges that meet at the vertex to smoothly connect all neighbouring blends. Therefore, in this paper, the edge blend will be discussed in detail, and the term blend will be used for this unless explicitly stated otherwise.

The geometry of an edge blend is determined using an imaginary rolling ball that maintains contact with the faces to be blended [2]. The blend face is the envelope of this ball as it rolls along the edge. Blend geometry is associated with four key elements: imaginary *spine curve*, *support faces*, *spring* (or *rail*) and *terminating edges*; see Fig. 1b. A spine curve describes the locus of points traced by the rolling ball's centre. Support faces are the side faces, supporting the rolling ball, of the blend. Spring edges are the two edges of contact traced by the contact points between the rolling ball and the support faces. A terminating edge is the shortest line on the extremes along the blend face that connects the two support faces. This edge is a cross section of the blend face characterizing the shape of the blend. In addition, the *blend radius* refers to the radius of the imaginary rolling ball; this radius can be constant or variable. The former defines so-called *constant radius blends*, whereas the latter describes *variable radius bends*.

During blending, it is not uncommon that a single operation generates many blend faces connected smoothly to each other to form the blend. A *blend sheet* is such a set of blend faces created in a single blending operation [4]; the set is referred to as a *chain*. For example, in Fig. 1c, the round is described by a blend sheet defined by a chain of 3 blend faces. Chains are usually generated when any of the adjoining support faces of the blend changes during blending [2], e.g. when a convex blend is created following a concave blend, as in Fig. 1c. Edges between blend faces in the same chain are known as *cross edges*. Note that a blend sheet can also have a single blend face. In this case, the blend sheet has no cross edge.

Braid [3] summarizes the creation of blends, in current geometric modellers, into four stages. Firstly, blend attributes containing blending details, such as blend radius, are attached to each blend edge(s). Secondly, new blend faces needed for the blend sheet are created. When one examines a blend edge, one sees that the surfaces of the two support faces and the blend radius, together determine the blend sheet geometry. Next, a Boolean operation is used to find the intersection between the blend sheet and the original model being blended. This stage derives the extents of the blend sheet, i.e. the spring and terminating edges bounding the blend face(s). Finally, the blend sheet is created.

Once a blend has been created, it is attached to the model at the specified location. Attaching the blend involves trimming the original faces, and replacing portions of these faces with the blend sheet. In the model, a number of blends may *interact*

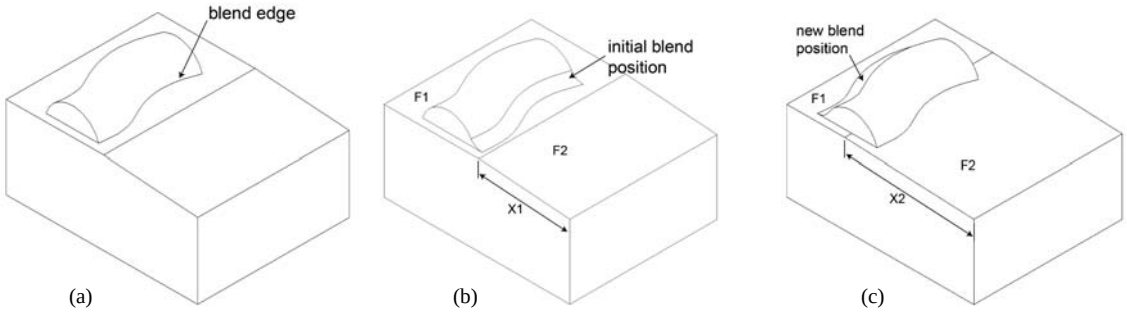


Fig. 2: Problem with specifying blends on the Brep

with each other to form a complex *blend network*. When a blend face is at the end of a blend chain, terminating edges get created. If a blend terminates on a single face, a single terminating edge results; say, between the top face and the round in Fig. 1a. However, if it terminates on multiple faces, multiple terminating edges are created, as in Fig. 1c. Braid [3] illustrates some of the many configurations that can occur with blends in a model.

3. BLENDS AS FEATURES

Most current commercial modelling systems are feature based. The user operates on features, and geometry is created on the basis of these features. He specifies a feature model in an interactive way, typically via a graphical user interface. Feature modelling permits users to associate functional with geometric information. A recent survey on feature modelling can be found in [5].

A feature is defined using a feature class, which is a *structured* description of all the properties of a given feature type, defining a template for all its instances. A feature class definition always includes a generic shape for the feature, and a number of parameters and constraints that characterize this shape. Constraints capture design intent in the shape, e.g. to fix the radius value range from 2.0 to 2.5 mm.

When values of parameters have been specified, an instance of the feature class is created. The feature is then attached to one or more other features already in the model, i.e. one or more of the faces in the feature are coupled to one or more faces in one or more other features. The involved faces are called *attach faces*.

Blends can be modelled as features too, but several problems are faced when doing this. Three major shortcomings are discussed here.

Firstly, in most commercial systems, blends are specified on model boundary elements. In addition, most commercial systems are based on geometry kernels, such as ACIS® and Parasolid®. These are, in fact, Brep modellers. Blends are typically specified on a blend edge selected on the model (see Fig. 2a). Since the kernel modellers being used are Brep modellers, blending changes the model boundary, and the blends, and other features, are no longer represented at the parametric definition level.

Consequently, because the explicit feature model can no longer be referred to directly during regeneration after an edit, current systems sometimes have to guess the matching element whenever ambiguity exists. If the guess is wrong, the model can exhibit unexpected, often undesirable, behaviour. As an example, consider the model shown in Fig. 2a. An edge was chosen for rounding; the result is shown in Fig. 2b. This model is now edited by altering the value of the side length X1. In a particular commercial system, when some value X2 is attained, the blend is surprisingly repositioned as shown in Fig. 2c. It seems reasonable to suspect that the system specifies the blend in terms of boundary elements, e.g. that in Fig. 2b the blend is specified on the face F1 and the face of the freeform ridge feature. By extending the face F2 in Fig. 2c, it becomes ambiguous where to place the blend now. The system's guess corresponds to the same face F1, which is now only on the opposite side of the ridge as F2, thereby repositioning the blend as in Fig. 2c, even when the semantics is clearly different.

In fact, blends are a victim of the so-called *persistent naming problem*, the source of which is generally attributed to deficiencies in the Brep. Hoffmann [6], Kripac [7], Middleditch and Reade [8] and Bidarra et al. [9] argue against basing the representation of features on a conventional Brep. A discussion on the persistent naming problem is not intended in this paper. Interested readers may consult [10] for a survey paper on the problem. Blends seem to suffer even more from the persistent naming problem than other types of features, because of their complex shapes and relations with other features.

The second major shortcoming is that most commercial systems poorly describe properties of the blend feature at the parametric level, i.e. they are predominantly macros at the parametric definition level [5]. A macro is more advanced than explicit geometry, but lacks sufficient information to concisely represent the blend feature. As a consequence, the meaning of the blend feature is not well represented.

The final major shortcoming is that feature modelling systems poorly maintain semantics of blends, if any. Once added to the product model, which is usually a Brep, the blend's meaning is lost. This permits previous design intent to be overruled in certain edit operations. For example, the vertex blend added in Fig. 1b has overruled the properties of the previously created

fillet in Fig. 1a. Such changes could also alter geometric properties, e.g. continuity between adjacent faces, specified during instantiation of the blend feature. These uncertainties are due to the fact that features are just considered as macro operations provided to the user to speed-up the traditional geometric modelling process. As a consequence, the only information stored in the model that can be used for further operations is still and just the geometric information.
The main problem with systems deploying blends as macros is the lack of facilities to specify *validity conditions* on the blend feature. This limitation, and the use of a Brep, has retarded the development of tools to perform specific *validity checks* during the modelling process [1].

4. THE NEW APPROACH

This section introduces the new approach to modelling blends. It is based on the concept of *semantic feature modelling* [1]. In semantic feature modelling, an essential aspect of a feature is that it has a well-defined meaning, or semantics, and this semantics is maintained during the whole modelling process. In this paper, the objective it to improve the definition of current blend features along these lines in three major ways.
Firstly, properties of the blend are specified in a corresponding feature class. A new data structure is provided at a high abstraction level, on which parameters and constraints describe the shape of the blend feature.
Secondly, a new, intuitive way to calculate the blend's shape from the parameters and constraints is developed. The shape is calculated by a method from the blend feature class, and explicitly stored in a blend feature shape, independent of the kernel modeller data structure for the whole model.
Finally, the concept of *validity maintenance* is introduced for blending, so that design intent can be specified and maintained by means of constraints on a blend feature.
A general distinction can be made between a feature class definition and a feature model; see Fig. 3. The feature class declares all the generic information pertaining to the blend feature, whereas the feature model constitutes instances of the blend as well as other features. In addition, the feature model is made up of the *unevaluated feature model* and the *evaluated model*. The unevaluated feature model contains feature shapes (volumes) and a feature dependency graph, whereas the evaluated model is the geometric representation for the overall boundary, consisting of portions of all feature shapes. So, three levels of abstraction can be distinguished: a high-level *blend feature class*, an intermediate level *unevaluated feature model*, and a low-level *evaluated feature model*, labelled, respectively, as (a), (b) and (c) in Fig. 3.

4.1 The blend feature class

One of the main intentions of introducing a blend class definition is to make the blend independent of the kernel data structure. The blend feature class represents parameters, constraints and other generic entities as *class member variables* and describes the generic shape using *methods* of the feature class (see Fig. 3a).
The parameters and constraints are used to specify the generic shape of the blend feature. Parameters are used to define the shape and size, e.g. radius, of the blend feature. Parameters are presented in Section 5.1. A constraint is a restraint imposed either on a parameter (*parameter constraint*) or shape (*shape constraint)*. An example of a parameter constraint is a *value constraint* that is used to limit the range of a parameter value; an example of a shape constraint is a *surface area constraint* used to limit the value of the total surface area of all surfaces of a feature in the evaluated feature model. Constraints will be detailed in Section 5.2.
The generic shape of the blend feature is defined in the blend feature class *method*. The geometry of the shape is described using generic entities called *feature entities*. What is new, contrary to existing definitions, is that feature entities are included in the blend class definition. This makes them explicit in the blend feature shape. A blend feature class can have an arbitrary number of feature entities. The entities are important since they provide an interface to parameterize the generic shape of the blend using the constraints. Two types of feature entities can be identified: *feature geometry* and *feature topology entities* (described in Section 5.3). Furthermore, the feature class defines attach and dependency relations for the blend feature; these are explained in Section 5.4.
As mentioned in Section 2, various types of blends are known from literature, e.g. rolling-ball, spine-based and trimline-based. A blend can be classified according to its characteristics, e.g. properties of its shape. The blend class depicted in Fig. 3a is a *generic abstract* class definition [11], which implies it cannot be directly instantiated by simply assigning values to its parameters. Specific blend classes are derived from this abstract class definition in order to specialize the definition according to the intended blend feature type, e.g. in an edge-blend class based on the rolling-ball technique the method of the super-class (the generic blend class) is redefined to implement the rolling-ball algorithm. This derived class is a generic *specific blend* feature class that can be instantiated, e.g. when parameter values are specified.

4.2 The unevaluated feature model

Each blend feature has its own shape derived from the generic shape in the corresponding class definition, called a *blend feature shape*. This shape is independent of the kernel geometry representation of the whole model, and thus provides an

invariant structure for subsequent modelling operations. The blend feature shape, therefore, defines a concrete interface to the blend; it can be intersected, its volume and surface area (in the evaluated model) can be precisely calculated, and so on, just like for other features. It represents the complete geometry and topology using feature entities. The determination of the blend feature shape is detailed in Section 6.2. The blend feature shape and the other feature shapes already in the feature model make up the unevaluated feature model (see Fig. 3b).

Two essential properties can be identified with feature entities. Firstly, unlike a Brep element in the evaluated model, whose topology can change (see Section 4.3), a feature entity cannot be merged, split or deleted once instantiated, even if its geometry representation is [1]. Consequently, feature entities provide an invariant structure on which to specify the parameters and constraints for the blend feature shape. Secondly, a single feature entity may correspond to one or more geometry representations of the same dimension in the Brep, e.g. a spring feature edge may correspond to a chain of spring edges in the Brep. This structure is achieved by encapsulating the geometry representations, of the corresponding feature entity, in dynamic lists in that feature entity's definition, in order to allow dynamic (de-)allocation of its geometry representations without otherwise changing the structure of the feature entity itself. The use of such lists is crucial here, because blending at the kernel level often causes a single edge to get split up into a set of connected edges (edge chain) or get merged to create a single edge to meet certain levels of continuity, e.g. 2nd order continuity.

In addition, the unevaluated feature model maintains a *feature dependency graph*. This graph is used to describe the relationships among all feature instances in the model. Detailed discussion on the use of the feature dependency graph can be found in [1].

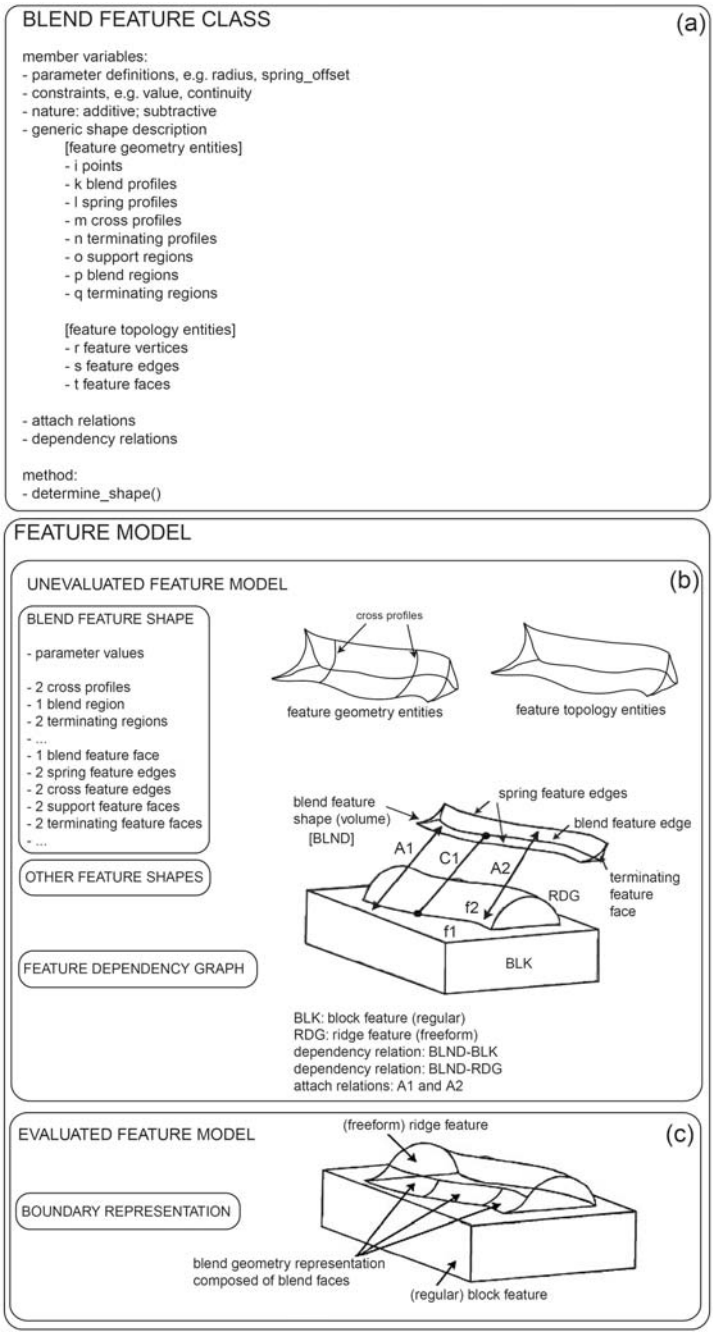


Fig. 3: Levels of abstraction for a blend feature

4.3 The evaluated feature model

The evaluated feature model is depicted in Fig. 3c. Usually, in a geometric kernel, a Brep scheme is used to provide a description of the faces, edges and vertices that make up the complete feature model's boundary, including the entities' adjacency information. When one compares the feature shapes against the feature model (see Fig. 3b respectively, 3c) the feature model is a boundary description of the combination of all feature shapes. More advanced representations than a conventional Brep, particularly a *cellular model* [12], can be used for the feature model to provide additional functionality.

4.4 Instantiating a blend feature class

A blend feature instance can be created from the feature class definition once attachment information and parameter values are available. The required information is specified in the corresponding class definition, e.g. support feature faces and blend

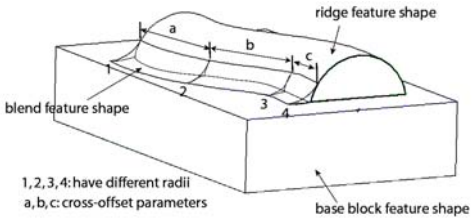


Fig. 4: Radius and cross-offset parameters

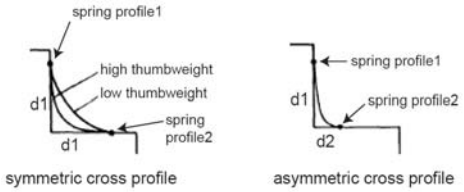


Fig. 5: Cross profile properties

radius. Part of the information is derived from other features already in the feature model, e.g. attach feature faces; other information is directly input by the user, e.g. the value of the blend radius.

During instantiation, firstly the user is prompted for attach feature faces, in the feature model, used to define support feature faces. For each parent feature, a dependency relation is defined between the parent and the blend. Also, the attach feature faces define attach relations for each coupled face-pair.

Secondly, parameter values are requested. Depending on the blend type, different parameters may be required. Part of the supplied information is subsequently used to determine the position of the feature geometry entities, e.g. spring profiles, in the feature model from which the feature topology entities, e.g. spring feature edges, are instantiated.

Thirdly, a record for the blend feature is created. It includes the dependency and attach relations specified in the first step. The same information is also fed into the feature dependency graph; see Fig. 3b.

Finally, the blend feature is inserted into the evaluated model, see Fig. 3c, and the feature model is updated. The link between the feature entities and their corresponding representations is then created (see Section 5.3).

Geometric, functional as well as technological properties of a feature make up its validity conditions, expressing design intent in the feature model. The technology to preserve these properties, in terms of conditions, is known as *validity maintenance*. Also, validity checks can be performed at various stages during creation and modification of the blend feature. Instantiation of a blend feature class is detailed in Section 6.

5. BLEND FEATURE CLASS DEFINITION

In this section, the blend feature class is described in more detail. In the class, parameters, constraints, relations and references to feature entities are represented as *class member variables*. Feature entities are used to describe the generic shape of the blend feature; the parameters, constraints and relations are specified on the feature entities.

5.1 Parameters

The types and number of parameters in a blend feature class depends on the type of blend. For example, one or several *radius* parameters are required for a rolling-ball blend; a single radius for a constant-radius blend and multiple radii for a variable-radius blend. The radius parameter specifies circular cross sections. In a variable-radius blend, every radius is associated with an offset distance along the blend edge, e.g. 'a', 'b' and 'c' in Fig. 4. These offset distances, called *cross-offset parameters*, define the distance between two consecutive cross profiles along a spring profile, such that each cross profile has a corresponding point pair, one point on each spring profile.

Moreover, this approach allows the size of the blend feature to be specified. This is new to blends in existing approaches where the extents are only implicitly derived from the attach faces. For instance, in most commercial systems an edge blend always spans the whole blend edge; single or a sequence of them (a chain). However, in this work, the blend shape can be specified to span only a portion of the blend edge. Powerful dimensioning techniques can be used that do not force the user to dimension the blend feature using explicit coordinate values, e.g. up-to, up-to-next, and so on, by referring to other feature entities, say, feature faces, or even to such references as datum planes.

Parameters can be different for other types of blend than the rolling-ball blend. For example, a blend using trimline-based algorithm does not necessarily need the blend radius to be supplied, but instead the *spring-offset* and *thumbweight parameters* are required. A *spring-offset parameter* defines the extent of trimming back of the support faces along spring profiles, indicating how far each spring profile is locally from the blend profile. Fig. 5 indicates spring-offset parameters, detailed as d1 and d2. Blends can have symmetric or asymmetric cross profiles; in the former the values for spring-offset parameters are equal, whereas in the latter they are different. In addition, the shape of the cross profiles can be controlled by adjusting the so-called *thumbweight* [2]; see Fig. 5. The thumbweight is a measure of the closeness of a cross profile to the support regions: the higher the thumbweight is, the more closely the cross profile follows the shape of the support regions. In addition, the shape can be controlled by independently changing the continuity, between the blend and support feature faces, along the spring profiles.

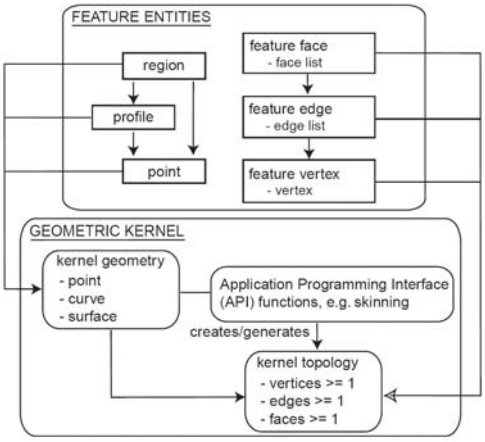


Fig. 6: Feature and kernel entities

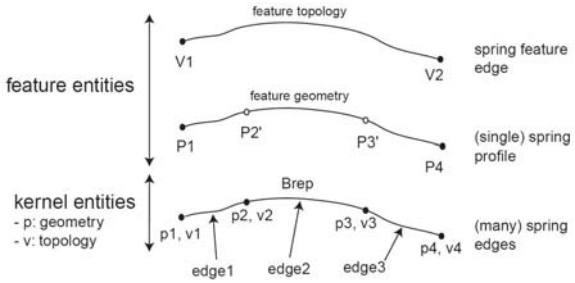


Fig. 7: Mapping between feature and kernel entities

5.2 Constraints

Constraints are part of the validity specification for the blend feature class. As earlier stated, a constraint can be either a parameter or a shape constraint. Parameter constraints include *value* and *algebraic constraints*, and shape constraints include *surface area*, *volume*, *curvature* and *continuity*.

A value constraint limits the range of a parameter's value, e.g. a maximum allowed radius of 2.2 mm. An algebraic constraint algebraically relates two parameter values, e.g. two spring-offset parameter values d1: d2, in Fig. 5 as 2.5:1.

A surface area, volume and curvature constraint can be specified to set the property's allowed range in the blend shape.

Continuity can be specified on the blend feature, depending on the blending technique used. In the rolling-ball algorithm, G^2 continuity can be achieved, although G^1 is usual. In the other types of blends, continuities higher than G^2 are possible.

5.3 The blend feature shape

As mentioned in Section 4.1, the generic shape of the blend is described using feature entities. Feature entities are generic entities used to define geometry and topology of the blend feature shape, namely *feature geometry*, respectively, *feature topology* entities. Feature geometry entity types include *point*, *profile* and *region* and feature topology entity types include *feature vertex*, *feature edge* and *feature face*, see Fig. 6. Feature geometry entities describe the shape of the blend independent of the geometry representation in the kernel. Feature topology entities describe the boundary of the resulting blend feature shape.

An instance of the blend feature has a concrete feature shape, see Fig. 3, that keeps reference to the generic shape in the class definition. In the feature shape, the feature geometry entities, point, profile and region, abstract the geometric point, curve and surface, respectively, in the kernel, as depicted in Fig. 6. For example, the spring profile pertains to the spring curve in the kernel. Analogously, feature topology entities, feature vertex, feature edge and feature face, abstract vertex, edge, face, respectively, in the kernel, e.g. a spring feature edge pertains to a spring edge in the kernel. At the kernel level, every topology element is associated with a geometry element, e.g. the vertex-point {v2, p2} in Fig. 7.

As mentioned in Section 4.2, a feature entity never gets split, merged or deleted, even when the elements in the corresponding representation in the kernel do. Thus, a correct mapping is needed to define the correspondence between the feature topology entities, in the feature shape, and the topological elements in the underlying geometric kernel. To this end, whenever a kernel element gets split, e.g. a spring edge into a spring edge chain (Fig. 7), the corresponding feature geometry entity records the positions, as encapsulations of points in the feature shape, without the feature entity itself splitting, e.g. {p2} encapsulated in {P2'} of the spring profile. To record the one-to-many correspondence between feature and kernel edges and faces, each feature edge and feature face can encapsulate one or more topologic entities in the kernel through dynamic lists for the kernel entities included in its definition, see Fig. 6. Similarly, when kernel topology entities merge, the removed kernel entities are de-referenced by removing them from the lists in the definition. Thus, although the kernel's data structure may change, the feature shape data structure persists. This new facility is explained in Section 6.2 where determination of the blend feature shape is detailed.

5.4 Other information

The feature blend class also defines dependency and attach relations. For example, in Fig. 3b, suppose feature faces {f1, f2} are selected to be the support feature faces. The features owning the attach faces define dependency relations, e.g. BLND-(BLK and RDG). Also, the attach faces specify attach relations for each coupled face-pair. For instance, A1, in Fig. 3b, defines an attach relation between the support feature face, on the block feature (BLK), and the blend feature shape (BLND). Furthermore, the *nature* of the feature, also specified in the corresponding feature class definition, indicates whether

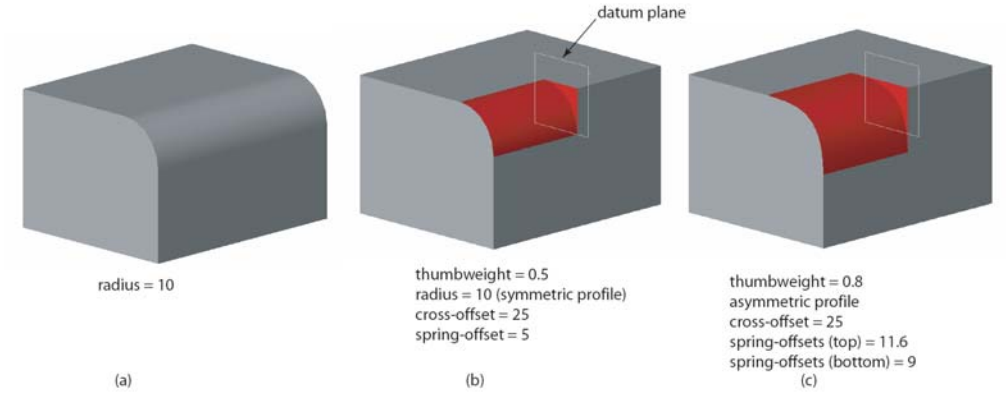


Fig. 8: Specifying a blend feature shape

the instances of the class definition represent material added to or removed from the model: *additive*, respectively *subtractive* natures. For example, a fillet is additive, whereas rounding is subtractive.

6. BLEND FEATURE CLASS INSTANTIATION

This section details instantiation of a blend feature class definition. Instantiation of a blend is somewhat different from other features since most of the information for instantiation is obtained from features already in the model. For clarity, first, the well-known rolling-ball blend will be used to explain the new method. Note, however, that other types of blends, based on the generic class definition, in Section 5, can be handled as well. The following steps describe the instantiation process: specifying instantiation information, computing the shape of the blend, validating and attaching the blend, and determining the evaluated model.

6.1 Specifying instantiation information

First, attach and dependency relations should be defined. The support feature faces the user selects also identify the parent features for the blend, i.e. the features that own the support feature faces. For instance, the support feature faces for the simple blend in Fig. 8a are the top and front feature faces on the block feature; the block defines the parent. One unique aspect to note here is that instead of using Brep faces in the evaluated model to define the support faces, feature faces in the feature shape are used because they refer to the generic shape definition in the feature class, they are explicit and persistent at the parametric definition level (in the unevaluated feature model). Secondly, one or several blend radius parameters are specified: a constant-radius blend (Figs. 8a, b) requires only one value, whereas in a variable-radius blend (Fig. 4) several radii and the associated cross-offset parameters are required. Finally, the blend extent can be defined by using our new approach (explained in Section 5), which allows the user to select one or several other feature faces in the feature model, or even datum planes, e.g. a zx-parallel plane in Fig. 8b. This is the information necessary to define the terminating regions for the blend volume.

6.2 Computing the shape of the blend

This section details the blend feature class *method*, wherein the blend feature shape is actually computed (created and evaluated). In the method, an external geometric kernel, ACIS®, is used to perform geometric computations through a selected set of Application Programming Interface (API) functions.

Currently, the most attractive blending technique, from a modelling point of view, is the rolling-ball method, because it automatically generates some of the required shape information, e.g. spring and cross curves. But a major shortcoming with this technique is that the surface swept by the moving ball, the so-called *canal surface*, is of high algebraic degree, even in relatively simple cases, which often makes resulting blends unstable. Also, due to computational considerations, the representation of such surfaces in exact form is excluded, and generally approximate methods are used [2]. Hence, we take a generic approach that exploits some of the merits of the rolling-ball technique and extends the algorithm to allow other types of blends to be modelled. In this new approach, the shape is computed in four steps: determine profile positions, skin the profiles to determine the blend volume, extract feature topology, and link the feature entities with the kernel entities.

6.2.1 Positioning the profiles

Using the instantiation information, particularly the blend feature edge, support feature faces, and blend radius (or radii and the corresponding cross-offset parameters), the kernel is invoked to execute the rolling-ball blending procedure. As noted earlier, this procedure provides information that is later used to determine the blend volume, e.g. information for positioning

spring profiles, which can be modified for other blend types. Basically, a preliminary rolling-ball geometric blend is computed at Brep level only to provide specific information for positioning entities in the feature model.
The interrogated entities are: blend, spring, terminating and cross (in variable radius blends) curves (and edges). The positions of the corresponding profiles for the feature shape are determined by interpolating the points on these curves, in the kernel, using NURBS. The number of points on each profile does not need to equal the number of control points on the NURBS, and in fact, the profiles do not interpolate control points but evaluated positions on the curves themselves [11]. Additional geometric properties for the NURBS are determined using constraints, e.g. continuity constraint, specified on the corresponding profile. If the extents of the blend were specified by the user using a datum plane, as in Fig. 8b, the geometric domains for the spring, blend and/or cross profiles would be trimmed to the plane. The terminating profiles would then be computed as the intersection curve between the canal surface and the plane. The same steps would be taken if feature faces were selected instead of a plane. Finally, the positions of the feature geometry entities, respectively blend, spring, terminating and cross profiles, are established.

6.2.2 Determining the blend volume and feature topology

With the feature geometry entities positioned, the blend volume is computed. The regions for the surfaces of the blend volume are determined by *skinning* the profiles (Figs. 3b and 4). Skinning is a technique used to create a surface fitting through a series of curves. The spring and terminating profiles are skinned to create the blend, respectively, terminating feature regions. The blend feature region connects to the support surfaces with tangent continuity, see Fig. 8. Closure regions are also created between each of the spring profiles and the blend profile, to complete the blend volume (Fig. 3b). These regions coincide with (part of) the support feature faces in each of the parent features (later used for attaching the blend).
It should be noted that, in our approach, the blend shape is described differently from existing approaches. In particular, the blend feature has its own volume, which is used to imprint the blend shape in the evaluated model (see Fig. 8) as other standard features, e.g. slot and pocket do. On the contrary, current systems only describe blends in the evaluated (geometric) model as surface patches. This difference is crucial because, in the new approach, a blend is explicit in the unevaluated feature model, so that it can be selected, intersected, and so on.

6.2.3 Linking feature and kernel entities

The feature shape is then mapped to the geometric kernel. The link is established through the feature entities, as outlined in Section 5.3. This is accomplished by the dynamic lists, for kernel topology elements, in the corresponding feature entity definitions, and labels are assigned to each of these elements as attributes. One feature edge may correspond to several edges in the Brep. All such edges populate the list in the corresponding feature edge. For example, an edge split in the Brep introduces new bounding vertices that, together with the split edges, create an edge chain, e.g. the spring edge chain in Fig. 7. All point-vertex pairs on the edge chains, e.g. {p2, v2} in the spring edge chain, are also determined by ACIS®. The positions, in the kernel, are recorded by the feature geometry entity (here, the spring profile) as encapsulations in points on the profile, without splitting the feature entity. They can become particularly important if the points are later used to further manipulate the blend or to specify other blend types, as will shortly become clear. Finally, the Brep topology elements are added to (e.g. after splitting) or removed from (e.g. after merging) the list in the feature topology entities for which they are representations.
In this approach, we avoid the persistent naming problem by specifying the blend on the feature entities of other features instead of specifying it on Brep elements in the Brep as most current systems do [9]. The blend feature shape, like the other feature shapes [1], is therefore independent of the Brep data structure.

6.3 Validating and attaching the blend

The blend feature shape is validated when it is added to the feature model or when it is edited. The properties specified on the feature are verified at two levels: during feature class instantiation and shape computation. During instantiation, constraints are enforced, e.g. on the radius and spring-offset. Other constraints are verified during shape computation, e.g. a curvature constraint when generating the blend region. In case of violation, a *validity recovery loop* is entered to allow the user to interactively correct the problem. As an example, the fillet feature in Fig. 9 has a volume constraint of maximum and minimum values, respectively, 46.2 and 55.8 mm^3. When the blend is edited such that the constraint is violated, the system notifies about that particular violation. Importantly, the blend is excluded from the evaluated model, see Fig. 10, until the problem is resolved. The concept of validity recovery for freeform feature modelling is detailed in [13].

6.4 Determining the evaluated model

In the unevaluated feature model, the blend feature has an explicit volume, represented by its feature shape. All feature shapes in the unevaluated feature model, the dependency information and the nature of each shape are input to the

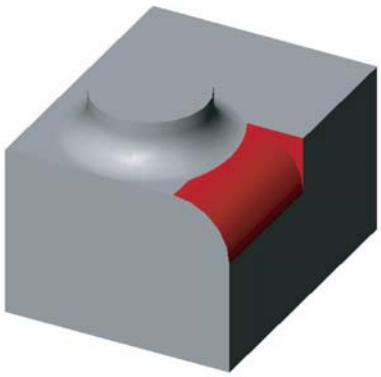


Fig. 9: Blends in a feature model

Fig. 10: Validity maintenance facilities

boundary evaluator. For example, if the blend is subtractive, e.g. the round in Fig. 9, its geometry is computed by a Boolean difference operation with its parent(s).

7. OTHER TYPES OF BLENDS

As mentioned in Section 4.2, other types of blends can be created from definitions derived from the abstract generic blend feature class (Fig. 3a). To implement other blend types, only the *method* in the abstract class, i.e. 'determine_shape()' in Fig. 3a, should be overridden. Specifically, only one change is made: instead of executing the rolling-ball procedure in Section 6.2.1, other procedures, e.g. spine-based and trimline-based procedures, are used. The instantiation information will depend on the blend type (see Section 5).

A completely new blend type has also been developed. It exploits some of the merits in existing techniques. For instance, in the rolling-ball technique only minimal information, viz. support feature faces and radius (or radii and cross-offset parameters), is sufficient to instantiate the blend. This readily provides information to position the blend feature in the feature model.

The method for the new type of blend therefore starts with the rolling-ball procedure for, mainly, positioning the blend in the model. In a rolling-ball blend, cross curves are always circular arcs (symmetric), as in Figs. 8a, b. The blend so determined can only be modified by changing the radius parameter. In our approach, the user is provided with additional parameters to manipulate the blend in the model, e.g. the thumbweight, spring-offset and cross-offset parameters and size (extents). Fig. 8 demonstrates how the three parameters have been used, in our prototype system, to modify the blend feature within the feature model, from Fig. 8a to Figs. 8b and c. Furthermore, in the manipulation from Fig. 8b to Fig. 8c, the originally circular cross profile has intentionally been edited to asymmetric. This is possible because the geometry for a profile is a NURBS surface, which provides additional degrees of modelling freedom. Importantly, the resulting blend still maintains the tangency condition with its support faces in the feature model (Fig. 8c).

In addition, the spring profiles can also be manipulated to create even more complex blends. To demonstrate the idea, we present complex blend shapes in Fig. 11. In these examples, the profiles can be interactively manipulated by means of the points in their definition, either directly or indirectly. Direct manipulation is achieved by using the explicit points in the spring profiles. The user can slide the points within each support feature face, such as 'b' in Fig. 11a has been used to reduce the spring offset 'a' from Fig. 11a to Fig. 11b. This facility also allows making straight spring profiles, e.g. in Fig. 8a, to have arbitrary shapes as in Fig 11. The profiles can as well be manipulated indirectly through parameters. For instance, in Fig. 11b, different *thumb*weight parameters have been used to add smooth depressions in the same way sculptors use the thumb to mould clay. In addition, the figure demonstrates the new facility to specify the extents (size) of the complex blend even after it has already been created in Fig. 11a: here the blend is made to span only a portion of the support feature faces. Current feature modelling systems do not offer the user this option. Finally, observe that the tangency between the blend and the support feature faces is *always* maintained.

8. CONCLUSIONS

In this paper, a new approach to blending has been introduced and demonstrated. It includes three new concepts: the blend feature class concept, a new method to determine the blend shape, and validity maintenance to blending. It demonstrates how the new approach facilitates modelling blends as semantic features.

The feature class provides the basis for high-level definition of the blend. All parameters, constraints and other design information are specified in the corresponding class. The methods of the class are used to determine the shape based on the parameters, constraints and the new shape calculation technique. Validity conditions specified on the blend feature are stored in feature models. This semantic information can be verified within the methods in the blend class definition whenever

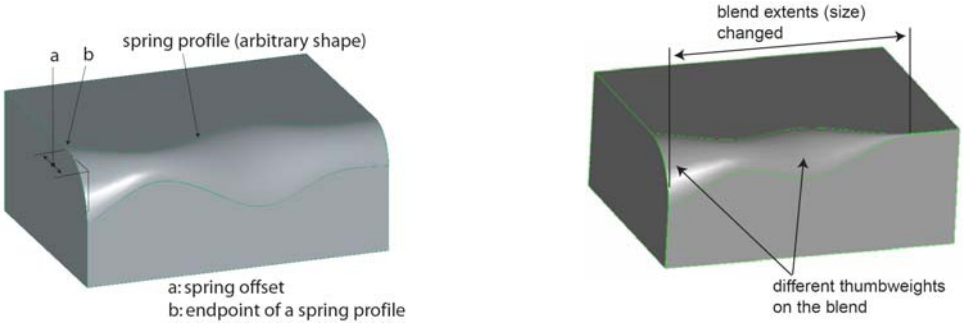


Fig. 11: Other types of blends

an update of its shape is required. All the parametric information about the blend remains explicit to design, thereby making modelling with blends more intuitive and high-level.
Using this generic approach, more complex blending is possible. The basic concept of a class and its methods can be easily extended to other types of blends.

ACKNOWLEDGMENTS
We thank the Netherlands Organization for Scientific Research (STW) for supporting this work under project DIT. 6240.

REFERENCES

[1] Bidarra, R. and Bronsvoort, W. F., Semantic feature modeling. *Computer-Aided Design*, Vol. 32, No. 3, 2000, pp. 201-225.
[2] Vida, J., Martin, R.R. and Varady, T., A survey of blending methods that use parametric surfaces. *Computer-Aided Design*, Vol. 26, No. 5, 1994, pp. 341-365.
[3] Braid, I. C., Non-local blending of boundary models. *Computer-Aided Design*, Vol. 29, No. 2, 1997, pp. 89-100.
[4] ACIS Technical Overview, ACIS Geometric Modeller, Version 12, January 2004, *Spatial Technologies Inc*.
[5] Bronsvoort, W. F., Bidarra, R. and Nyirenda, P. J., Developments in feature modeling. *Computer-Aided Design and Applications*, Vol. 3, No. 5, 2006, pp. 655-664.
[6] Hoffmann, C. M., On the semantics of generative geometry representations, advances in design automation. In: *Proceedings of the 19th ASME Design Automation Conference '93*, Vol. 2, 1993, pp. 411-419.
[7] Kripac, J., A mechanism for persistently naming topological entities in history-based parametric solid models. *Computer-Aided Design*, Vol. 29, No. 3, 1997, pp. 113-122.
[8] Middleditch, A. and Reade, C., A kernel for geometric features. In: *Proceedings of the Fourth ACM Symposium on Solid Modeling and Applications*, Atlanta, Georgia, USA, Hoffmann, C.M. and Bronsvoort, W.F. (Eds.), 14-16 May 1997, pp. 132-140.
[9] Bidarra, R., Nyirenda, P. J. and Bronsvoort, W. F., A feature-based solution to the persistent-naming problem. *Computer-Aided Design and Applications*, Vol. 2, Nos. 1-4 (special issue: *Proceedings of CAD'05*, Bangkok, Thailand, 20-24 June 2005), pp. 517-526.
[10] Marcheix, D. and Pierra, G., A survey of the persistent naming problem. In: *Proceedings of Solid Modeling '02 – Seventh Symposium on Solid Modeling and Applications*, Saarbrücken, Germany, Lee, K. and Patrikalakis, N.M. (Eds.), 17–21 June 2002, pp. 13-22.
[11] Nyirenda, P.J., Mulbagal, M. and Bronsvoort, W.F., Definition of freeform surface feature classes. *Computer-Aided Design and Applications*, Vol. 3, No. 5, 2006, pp. 665-674.
[12] Bidarra, R., de Kraker, K. J. and Bronsvoort, W. F., Representation and management of feature information in a cellular model, *Computer-Aided Design*, Vol. 30, No. 4, 1998, pp. 301-313.
[13] van den Berg, E. and Bronsvoort, W.F., Validity maintenance for freeform feature modelling. Submitted for publication, 2007.