The Generic Geometric Complex (GGC): a Modeling Scheme for Families of Decomposed Pointsets

Ari Rappoport

Institute of Computer Science, The Hebrew University, Jerusalem 91904, Israel.
http://www.cs.huji.ac.il/∼arir. `arir@cs.huji.ac.il`.

Abstract: Modeling of families of geometric objects is a major topic in modern geometric and solid modeling. Object families are central to many important solid modeling applications, including parametric modeling schemes based on features, constraints and design history.

In this paper we introduce the *Generic Geometric Complex (GGC)*, a modeling scheme for families of decomposed pointsets. Each member of the modeled family is modeled using an improved version of the selective geometric complex. Hence, the GGC can be viewed as a generalization of the boundary representation to a modeling scheme for families of objects.

The GGC models a family in the classifying sense, supporting the *object membership classification* query. Association of corresponding boundary entities (e.g. vertices, edges and faces) in different members of the modeled family is supported by the *entity-to-name (E2N)* and *name-to-entity (N2E)* queries. We refer to generic naming mechanisms that possess knowledge only about the boundaries of the modeled objects as *invariant naming* schemes. We discuss several concrete ingredients of generic names, present a general algorithm for invariant naming of entities in selective geometric complexes in any dimension, and completely characterize invariant naming in the 2-D case.

Keywords: Generic Geometric Complex (GGC), shape families, Selective Geometric Complex (SGC), invariant naming, persistent naming, generic naming, classifying models, modeling schemes.

1 Introduction

Geometric modeling deals with the representation and manipulation of geometric objects in a computer. The primary objects of interest have mostly been Euclidean pointsets. The boundary representation (Brep) is the modeling scheme of choice for pointsets in most applications [Mäntylä88]. The Selective Geometric Complex (SGC) [Rossignac88] is a generalization of the Brep, capable of representing decomposed pointsets (pointsets with internal structures and mixed dimensionalities).

In modern geometric and solid modeling there is a shift of interest towards the study of *families* of pointsets. The concept of a family of pointsets is central to all modern modeling paradigms, including parametric modeling, feature-based modeling, modeling with constraints, modeling of assemblies and modeling of engineering tolerances. It is thus essential to provide a formal theoretical foundation to the modeling of families of decomposed pointsets.

1.1 Background and Previous Work

Modeling framework. Requicha [Requicha80] defined a representation to be a structure of symbols from an alphabet and a representation scheme to be a relation between a space of representations and an abstract mathematical space containing the things to be represented. In other words, a representation contains data which can be read, interpreted, and operated upon by algorithms.

In this paper we use the concept of a modeling scheme and the distinction between a modeling scheme and a representation scheme as presented in [Rappoport95]. A *model* is defined to be a 'black box' with a well defined interface stating which *queries* it supports. A *modeling scheme* is a relation between a space of models and an abstract modeling space containing the things to be modeled. A model can thus be viewed as a representation encapsulated by a specification and an implementation of the queries it supports. For example, the most common operations performed on geometric models are display and selection, both of which efficiently supported by the Brep [Rappoport96]. The motivation for these definitions is that the only reason to build a model of something is to do something with it; the queries specify what we desire to do and what are the abstract properties of the modeled thing in which we are interested.

In addition to queries, which are operations relating a model to its external users, there are *synthesis operations*, which enable the creation of models (either from scratch or from other models and representations) and their modifications.

Modeling families of objects. Surprisingly, there is not much previous work on modeling pointset families in general and Brep families in particular. Most studies of families of pointsets have been from the parametric, feature-based and constraint-based points of view. Geometric operation graph (GOG) schemes supporting parametric models were described in [Rossignac86, Rossignac89, Emmerik90, Solano94, Shah96]. These methods concentrate on providing high-level synthesis operations needed to create models but cannot explicitly relate Brep entities (vertices, edges etc) in different members of the modeled family. Since explicit support for those entities is the central ingredient of the Brep, these methods are not very relevant to the study of families of Breps. A scheme for naming single instances in assembly graphs was given in [Rappoport93]. This scheme is also limited to dealing with complete sub-objects. [Shapiro95] raises some questions concerning the relationship between a GOG-like family and the corresponding Brep, but does not provide answers.

Modeling schemes wishing to enable the identification of corresponding Brep entities in different members of an object family can do that by supporting two queries that we call **entity-to-name (**E2N**)** and **name-to-entity (**N2E**)**. E2N gives a unique generic name to an entity in a given object, which is guaranteed to be identical to the name given to this entity in other object members of the same family. N2E identifies the entity having the given name in a given member of the family. We collectively call these queries **generic naming** queries.

There are very few modeling schemes supporting generic naming [Kripac97, Lequette96, Chen95]. These systems are feature and history-based geometric operation graph schemes. The main technique they utilize is tracking of entity splits, unifications and modifications following the execution of synthesis operations. These generic naming schemes rely on explicit knowledge of the operation graph. Following their terminology, we call naming schemes that possess high-level knowledge about the operation graph **persistent naming** schemes.

We refer to generic naming schemes that possess knowledge only about the SGC (boundary and geometric structure) of members in the modeled family as **invariant naming** schemes. The only previous work we are aware of that deals with invariant naming is [Capoyleas96]. However, this work was done as part of a specific GOG modeler [Hoffmann93], and does not discuss the general invariant naming problem. It is limited to the needs and capabilities of that modeler, assuming at certain points knowledge about synthesis operations. The results are mostly applicable only in the context of the world view of this type of modeler.

Constraints specified over Brep entities define a parametric family of Brep objects. Most work on such so-called variational modeling techniques focus on algorithms to solve and analyze systems of constraints (e.g. [Hel-Or94]; a recent survey is given in [Bouma95]). It is assumed that once the constraint solver has computed a consistent set of coordinates for the Brep entities then the final Brep of the object can be easily recovered from them. As far as we know, nobody has studied the relationship between this assumption and the invariant naming problem. Note, however, that it is easy to *represent* such a family without generic names using an example Brep, as proposed in [Pratt96].

1.2 Contribution

In this paper we provide the first study of modeling schemes for families of decomposed pointsets. We present the *Generic Geometric Complex (GGC)*, a modeling scheme for families of pointsets each modeled by a selective geometric complex. We detail the queries supported by a GGC model, the most important of which are entity-to-name and name-to-entity, and discuss the GGC as a classifying model and as a parametric model. We also discuss representing and specifying a GGC using an 'example SGC'.

We define the invariant naming problem and discuss it in an abstract manner. We define the notion of an *ingredient* of generic names of SGC entities and study several concrete ingredients. We present a very general generic naming algorithm based on 'refinement by ingredients', and completely solve the practical 2-D case, proving that a connectivity component of a 2-D GGC can be uniquely named if and only if a single entity in it can be uniquely named.

Section 2 describes a generalization of the SGC. Section 3 gives a formal definition of the GGC. Section 4 defines and presents general solutions to the invariant naming problem. Section 5 completely characterizes the unique naming problem in the practical 2-D case.

2 The Selective Geometric Complex (SGC)

The GGC models a family of Selective Geometric Complexes (SGCs), a concept proposed by Rossignac and O'Connor [Rossignac88]. In this section, we present a version of the SGC that is slightly more general than the original one and that uses different terminology in some cases. The changes are explained and justified where they appear.

An SGC is a model for a decomposed pointset. By the term 'decomposed' we mean that the pointset is divided into disjoint subsets, each of which can be addressed independently. This term is shorter and more general than the description used in the original article [Rossignac88], characterizing the SGC as a model for 'pointsets having internal structures and incomplete boundaries'. The term 'decomposition' naturally implies that the subsets have an arbitrary dimensionality.

An n-dimensional SGC is defined using a set of pointsets that we call *carriers*. In general, a carrier can be any pointset in n-D, although there are two types of carriers that are much more useful than others: algebraic hyper-surfaces (defined as the zero set of a polynomial in n variables) and splines (pointsets defined by parametric polynomials whose parameter spaces are k-dimensional, $k < n$). By convention, space itself is also considered a carrier. In [Rossignac88], only algebraic hyper-surfaces and space itself were allowed. The fact that we allow any pointset is the primary difference between the current presentation and the original one.

The carriers are pointsets that in principle could be represented by any representation scheme. However, the spirit of the SGC is that they should be specified by a single easily computable mathematical formula. Each carrier possesses a mathematical *nature.* For example, a parametric quadric surface, a parametric cubic surface, a parametric cubic curve, an implicit surface of total degree two and an implicit surface of total degree three are of different natures. Depending on the application, there can be a further sub-classification. For example, an ellipsoid and a hyperboloid can be regarded as having different natures, even though both are quadratic implicit surfaces.

Sometimes it is desired to label subsets of a carrier, either because it seems intuitive to do so or because of application requirements. A primary example is the cone: it seems natural to want to label its apex, and perhaps to distinguish between the two subsets lying on different sides of the apex. Such a labeling actually constitutes a decomposition of the carrier, which is exactly what the SGC was designed to model. We assume that all carriers have already been decomposed.

The carriers induce a decomposition of space into entities as follows. The pair-wise intersections of the carriers are divided into maximal connectivity components. Each connectivity component is further decomposed into portions that are relatively open, dimensionally uniform and possess no self-intersections. Each maximal remaining pointset is called an *entity* of the SGC. For example, a 1-D entity is a curve without self-intersections that does not include its end points, which are 0-D entities. A study on the definition of entities in complexes defined by algebraic equations is given in [Shapiro96].

Zero-, one-, two- and three-dimensional entities are called vertices, edges, faces, and cells, respectively. In [Rossignac88], the term ‘cell’ was used for what we here call an entity. We have chosen the word ‘entity’ due to lack of a natural term for a 3-D entity other than ‘cell’.

Note that by definition, all points in an entity belong to exactly the same set of carriers. The converse, however, is not true, since, for example, points belonging to the same set of carriers may belong to different entities if the intersection set between the carriers is composed of several connectivity components.

Note also that the decomposition of space into entities is completely defined once the geometry of the carriers is known. One can say that the ‘real’ degrees of freedom in classical boundary representations and space decompositions are the geometries of the carriers and not the geometries of the entities, even if the entities (vertices, edges, faces) play a more prominent role in applications and in user interfaces.

Not all entities generated by the carriers are contained in the decomposed pointset that the SGC is meant to model. Entities can be labeled as *active* or *passive* to denote whether they are contained in the modeled pointset or not. In general, each entity can be labeled with several symbolic *attributes*. The determination whether an entity is active or not is done according to additional structural or other information. For example, the SGC could have been derived from a series of Boolean operations between simpler SGCs, determining the appropriate label of each generated entity. A particular representation might decide to store only active entities, or entities having some attribute.

The decomposition induced by the carriers into entities might be too fine. The user of the modeled decomposed pointset might wish to treat several entities as a single entity in the modeled pointset. A further optional *simplification* stage records such facts.

3 The Generic Geometric Complex (GGC)

In this section we introduce the Generic Geometric Complex (GGC) as a modeling scheme for a family of SGCs. We describe supported queries related to carriers and entities (Section 3.1), different senses in which the GGC models an SGC family (Section 3.2), consideration for concrete representations for GGCs (Section 3.3), and conclude in Section 3.4 with a formal definition of the GGC.

3.1 Carrier and Entity-Related Queries

The generic geometric complex models a family of SGCs. We must ask ourselves which properties of an SGC family are important so that we know which queries the GGC should support. The most important question is what do members in the family have in common. They must have something in common, otherwise we would not treat them as a family.

The central components of an SGC are its carriers and entities. It is thus natural to define the commonality between members of an SGC family in terms of their carriers and entities.

Carriers

As explained in Section 2, a carrier has both a mathematical nature and a specific geometry. The border between these two components is somewhat fuzzy and not always can be identified by symbolic means. For example, two quadratic polynomial curves are similar symbolically, but we may want an ellipse and a parabola to be considered as having two different natures.

The requirement that carriers have something in common can be interpreted in different ways. We may demand a one-to-one correspondence between the carriers of members in the family, or we may require the existence of a certain set of essential carriers and allow a member to possess additional carriers that do not correspond to carriers in other members. The first alternative is simpler to define, and in any case its study is essential in order to understand the second alternative. Hence in this paper we adopt the first alternative. The important practical implication of this decision is that in this paper we do not handle patterns having differing numbers of occurrences of a carrier.

Corresponding carriers of SGCs in the GGC family must have the same natures. We do not make any demand on their geometries. If it is desired that members should have a particular geometric property in common, then this property should be made part of the definition of a carrier's nature as in the ellipse and parabola example above.

Members in the family are required to possess corresponding carriers. How is this requirement exposed to the user of the family? That is, how is it phrased in terms of supported queries? This is done using two complimentary queries, carrier-to-name and name-to-carrier.

The **carrier-to-name (**C2N**)** query is given an SGC and a pointer to one of its carriers, and computes a unique generic name to the carrier; the name is guaranteed to be identical for the corresponding carrier in every other member. The **name-to-carrier (**N2C**)** query is given an SGC and a generic name, and returns a pointer to the carrier in the given SGC whose name is the given generic name.

Although this is not obvious at first sight, there are two completely different alternative semantics to these queries, depending upon the order in which they are allowed to be executed. If C2N queries can be executed before N2C queries, carrier names are given by the GGC model (the 'black box') without control of the user of the model. Moreover, in this case it is unreasonable to allow the user to execute an N2C query having as input a name 'invented' by the user and not returned by a previous C2N query. Allowing this usage could easily lead to naming inconsistencies and would make a robust and consistent implementation of the C2N query close to impossible.

The other alternative is to let only the users of the GGC specify generic carrier names and remove this responsibility from the GGC model. In this alternative, a generic carrier name is given by the user during the execution of the synthesis operation that introduced the carrier into the GGC. It is the user's responsibility to keep track of these generic names and their intended meanings. The supported queries are still N2C and C2N, but their implementation becomes trivial.

In this paper we take the second alternative and require that generic carrier names are given by the GGC user. This alternative makes the GGC much easier to implement, and at the same time is much more suitable to practical applications. A primary application of the GGC is for supporting geometric operation graph (parametric, feature-based) models. In these models, the graph of synthesis operations is stored as a part of the model, and it is easy to associate a generic carrier name with the operation that created it. If an operation introduces several carriers having similar natures simultaneously, naming them can be done arbitrarily. For example, instantiation of a parametric box creates six carriers (planes), which can be arbitrarily numbered.

From now on we will assume that carriers are symbolically named in an easily identified, consistent manner. We will refer to a carrier's symbolic name as its *identity*.

Entities

Similarly to carriers, we have two alternatives in deciding how entities of different members of the GGC family are related. We could require that there is a one-to-one correspondence between the entities, or we can relax this requirement and only require correspondence between a set of essential entities. Similarly to the carrier case, in this paper we take the former decision, since this case is simpler and must be understood anyway before discussing the second, more general alternative.

Unlike carriers, entities can be distinguished according to their attributes. The most important issue to resolve is whether it is essential that *all* entities have corresponding ones; perhaps this is essential only for the *active* entities.

Our view is that passive entities may play an important role in a modeled family. Indeed, most commercial systems provide 'auxiliary' geometric objects essential during the construction of other objects. The SGC is a 'flattened' version of all carriers participating in the design and specification of an object family, hence there are real advantages in allowing identification of corresponding inactive entities in different family members. We therefore introduce another attribute, the **essential** attribute, to represent the requirement that an entity must have a corresponding one in all family members. In all practical cases, an active entity will always be essential.

Identification of essential corresponding entities in different members of the

GGC family is supported by two complimentary queries, **entity-to-name (**E2N**)** and **name-to-entity (**N2E**)**. E2N is given an SGC and an entity, and returns a name guaranteed to be identical to that returned by E2N given another SGC in the family and the corresponding entity. N2E is given an SGC and a name, and returns the entity having that name. In principle, in the implementation of N2E it does not suffice to address only the given name, and membership of the given SGC to the family should be verified. It is always possible that two entities in two different GGCs will be given symbolically identical names.

SGC entities are derived unambiguously from the SGC carriers, hence here we cannot let the user specify generic names directly. A name given as input to an N2E query must always be a name previously returned by an E2N query. The responsibility of generically naming GGC entities solely belongs to the GGC model. Users can identify corresponding entities of different family members, but they do not possess direct control over the names used for this process.

3.2 The GGC as a Set Model

So far we discussed queries supporting an identification of corresponding components (carriers and entities) in different members in the GGC family. However, we have not discussed how to associate the members themselves with the family. Two options for doing that, the classifying and the parametric approaches, are now detailed.

General Set Models

In order to better understand the two options, recall that there are two major models for modeling general sets, regardless of the nature of their members: classifying models and parametric models. In a *classifying* model, we view the modeled set as a subset of a (usually large) containing space. Each member of the containing space can be represented or modeled by itself. The model supports the *membership classification* query: given a member of the containing space, return ‘Yes’ if it is a member of the modeled set, ‘No’ otherwise. This query implements the characteristic function of the modeled set.

In a *parametric* set model, it is assumed that each member can be given a unique name. Hence, formally, a parametric model is a model for *named* sets, which are different from unnamed sets. The set model supports the *parametric access* (or *named access*, or *indexing*) query: given a name, return the member having that name. The space of names is usually called the *parameter space*, and a name is referred to as a *parameter vector*. It is assumed that the parameter space can be specified rather easily, and that detection of invalid names is easy to do.

The GGC as a Classifying Set Model

When viewing the GGC as a classifying model, the containing space is the space of all families of decomposed pointsets, which is evidently extremely large. A classifying model would then support the ‘pointset membership classification’ (or the ‘SGC membership classification’) query: given an SGC, determine whether it belongs to the modeled family (Figure 1, left). This is similar to how a model supporting the ‘point membership classification (PMC)’ query models a pointset.

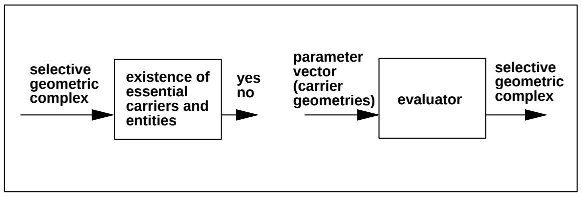
selective geometric complex
existence of essential carriers and entities
yes
no
parameter vector (carrier geometries)
evaluator
selective geometric complex

Figure 1: The GGC as a classifying (left) and parametric (right) set model.

Every model supporting the SGC membership classification query models a family of decomposed pointsets in the classifying sense. However, not every such model is a GGC. There are different classes of such models, distinguished by the type of information that they can make use of. The GGC only knows about the carriers defining its SGCs and treats all of them as having equal status. It is very different from classifying models possessing knowledge about the creation history of the modeled family or about a higher level labeling of entities according to functional features.

Given the N2E query, implementation of the membership classification query is easy. The names of all essential entities are obtained from the GGC model; a N2E query is executed on the classified SGC for every such essential entity. After that, one pass on the given SGC suffices to conclude that it does not possess redundant essential or active entities. The GGC models a family of SGCs in the classifying sense very naturally.

The GGC as a Parametric Set Model

To view the GGC as a parametric model, we can view the vector containing the geometries of the carriers as the name (parameter vector) of a particular SGC member. The GGC black box would in this case be an evaluator of the entities of an SGC given the geometries of its carriers, plus a mechanism to select the active entities.

Useful SGCs defined by a set of carriers denote as active only a subset of the entities in any dimension. It is clear that without an elaborate mechanism to select a subset of the entities as active entities, such a model is useless. This mechanism should examine the SGC resulting from the given carrier geometries, and label entities as active. This is not possible without a scheme for naming entities generically. The mechanism would store a set of generic entity names, search for them in the resulting SGC, and label them as active. The active entities play here the role of essential entities in the classifying point of view.

What happens if an essential generic name from the set cannot be found in the SGC? In some cases perhaps there are additional rules in the selection mechanism to cope with this case. However, in general such a situation in an indication that the given parameter vector does not belong to the parameter space. Hence the real difficulty with this approach is how to determine validity of the parameter vector, which stands in contrast to the basic assumption of parametric schemes, that the validity of a parameter vector should be easy to determine. In a sense, what is needed here is a classifying model for the parameter space, which is almost as difficult to specify and implement as a classifying model for the modeled SGC family.

In addition, although direct parameterized access to the GGC family is possible, it is not very useful. The parameters do not necessarily mean much. What is needed is a higher-level parametric interface. This is exactly the type of interface supplied by constraints on the one hand and geometric operation graph models on the other hand. A deeper study of both would take us too far from the subject of the current paper.

The conclusion of this discussion is that the parametric model is not a particularly natural model for a family of SGCs: it is difficult to verify membership in parameter space, and it is not very useful directly. Note that exhaustive enumeration of all family members is impossible for useful families.

3.3 Representation and Specification using an 'Example SGC'

Any model must contain a concrete representation; a modeling scheme should also possess associated synthesis operations to make it easy to specify and modify a model. The GGC must represent the sets of essential components, which could be directly stored as sets of generic names. However, this representation would make it very awkward to communicate with the model, visualize it and edit it.

In most systems, interaction with a family of objects is done through an 'example object' whose visualization is available to the user at all times. User actions are performed on the example object and are translated internally to generic actions [Hoffmann93, Rappoport96].

We can utilize this fact in the representation of the GGC. Assume that we always have an example SGC contained within the modeled family. The carriers of this example SGC are labeled with their identities, or generic names, as given by the user during synthesis operations. The entities of the example SGC possess the 'essential' attribute as well as the 'active' one. The essential entities can potentially store another attribute, 'name', which stores their generic names. In principle, the generic names can be computed on the fly as they are needed instead of continuously stored as attributes.

The 'example SGC' point of view is so useful that we can add the existence of an example as a query optionally supported by the GGC. It uses *concrete* carriers and entities to represent generic ones.

3.4 Summary

Following are two definitions summarizing the discussion in this section. The first definition is a formal, constructive one, detailing the queries that a GGC model must support.

Definition: a Generic Geometric Complex is a model for a family of decomposed pointsets, each modeled by an SGC. There is a bijection between the carriers and entities of members of the family. The model supports the following queries:

1. *entity-to-name* (E2N): given an SGC whose carriers possess generic names having equal status and an entity in it, return a unique generic name for the entity, guaranteed to be identical to that returned for the corresponding entity in all members of the given SGC's family.

2. *name-to-entity* (N2E): given an SGC in the modeled family and a generic name previously returned by E2N, return the entity having that name.

3. *membership classification:* given an SGC, determine whether it belongs to the modeled family or not.

4. *example* (optional): return a member of the modeled family.

Together, the first two queries define the **invariant naming** problem. Note that the problem potentially possesses a large number of solutions, most of which are useless in practice. Naming schemes are of different quality, and are a function of the intended application. It is difficult to capture this function mathematically.

The second definition is not a formal definition of the GGC as a modeling scheme. Rather, it is an intuitive, descriptional formulation of the modeled family as the family 'spanned' by an example member. Here, designating an entity as 'essential' is interpreted as imposing a constraint on the family, thereby narrowing it.

Definition: a Generic Geometric Complex is the family of SGCs obtained by modifying the geometries of the carriers of a concrete SGC while preserving the existence of generically named entities designated as essential.

4 Invariant Entity Naming

Invariant naming is the fundamental capability of the generic geometric complex. Since carrier names (identities) are given by the user, the real computational challenge facing GGC implementors is entity naming. In this section we analyze the naming problem, discuss several ingredients of a generic name, and present a general naming algorithm.

4.1 Name Ingredients

In general, there is no simple naming scheme guaranteed to produce unique names in all situations. Hence, in order to distinguish between identical names of different entities, several *ingredients* must be used in every name. Every additional ingredient increases the chance that a unique name will be found. However, there are still SGCs for which unique naming is not possible (see Section 5).

There are several classes of ingredients. An ingredient in an entity name is *independent* if it utilizes knowledge only about the entity itself and its carriers. An *entity-dependent* ingredient does not utilize names of other entities, but it can utilize knowledge about their existence. A *name-dependent* ingredient utilizes names of other entities. The dependent ingredients are not allowed to depend upon inessential entities. Ingredients can be considered as attempts to break symmetries in an SGC configuration.

Syntactically, an ingredient is a pair $(\mathtt{S}, D)$ where S is a symbol denoting the ingredient type and D is the data needed by that type. Words written in a fixed font are constant symbols, the others are variables. When sets are enumerated, their members are enclosed in round parentheses. It is easy to define an ordering on ingredients, obtaining a total ordering on the set of generic names.

Dimension and Attributes

An obvious independent ingredient is the dimension n of the entity, denoted by $(\mathtt{Dim}, n)$. If the application uses special attributes, they can also be used, denoted by $(\mathtt{Att}, AttName, V)$, where $AttName$ is the name of the attribute and V is its value.

Carrier Identities

The major independent ingredient is **carrier identities,** denoted by $(\texttt{Id}, (C_{k_1}, \ldots, C_{k_n}))$ where $(C_{k_1}, \ldots, C_{k_n})$ is an ordered list of the carriers generating the entity (it is easy to impose a total ordering on the carriers of any SGC because they have symbolic names already). Obviously, the carrier identities ingredient is not sufficient to uniquely name all entities. The intersection of several carriers may have several connectivity components; a single carrier may be decomposed into several disconnected pieces due to intersections with other carriers or to non-uniform dimensionality which necessitates its decomposition. On the other hand, sometimes this ingredient suffices, e.g. when all carriers are linear.

Special Entity Geometry

In some cases, special geometric properties of an entity can be used as another independent naming ingredient. Such ingredients are denoted by $(\texttt{Geo}, X, D)$ where Geo denotes a geometric ingredient, X is a symbol specifying its actual geometric type, and D the data needed by this specific type. Depending upon the application, such geometric properties could be used as independent name ingredients.

For example, consider two carriers: a 3-D torus C_1 and a 3-D line C_2 (Figure 2). The line is positioned on the torus such that it is tangent to it at one point A and intersects it at a segment BC. The points A, B and C are vertices of the resulting SGC having the same dimensions and carrier identities. However, vertex A has a special geometric property — the fact that it is a point of tangency between the line and the torus. We can thus add the ingredient $(\texttt{Geo}, \texttt{Tangency}, (C_1, C_2))$ to the name of vertex A. Note that in this example the tangency point is unique, thus is sufficient in order to distinguish vertex A from all others. Convexity of an entity is an obvious property that may be used in a geometric name ingredient.

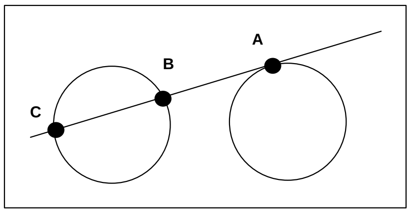

Figure 2: Vertex A is a point of tangency of a line and a torus (cross section shown), while vertices B and C are not.

Carrier Signs (Separation)

An entity-dependent ingredient that is not local is **carrier signs.** Let $SC = \{C_1, \ldots, C_k\}$ be the ordered set of signed carriers, carriers inducing a global sign function on space (e.g. implicit algebraic half-spaces). Let $C_i(p)$ be the sign of point p with respect to the carrier C_i. Then for every entity E, for all $p, q \in E, Signs(p) = Signs(q)$, where $Signs(p) = (C_1(p), \ldots, C_k(p))$. Hence $Signs(E) = (C_1(p), \ldots, C_k(p)),\ \ p \in E$ is well defined. The carrier signs ingredient is denoted by $(\texttt{Sign}, (C_1, \ldots, C_k), (S_1, \ldots, S_k))$ where $S_i \in \{+, 0, -\}$.

Two entities can be distinguished according to carrier signs if there is another *separating* carrier, that is, a carrier with respect to which the signs of points in the two entities are different. Motivated by representation conversion from Brep to CSG, Shapiro studied the usage of carrier signs to distinguish between cells (3-D entities) in algebraic geometric complexes [Shapiro93, Shapiro96].

It may seem that carrier identities is a special case of carrier signs, since the sign is zero for all carriers generating an entity. This statement is mathematically correct but semantically erroneous. When we consider the intended functionality of the generic names, the two ingredients are readily seen as extremely different. Carrier identities are fundamental to an entity; this is the very reason of being of the entity. Carrier signs strongly depend on the geometry of the separating carriers. In Figure 3 we see an SGC in which line A separates vertices B and C. However, only a small portion of line A is active, and this portion does not have any relation with the separated vertices. Using line A as a separating carrier to distinguish between generic names for B and C is equivalent to demanding that these vertices be separated by that line in all members of the family, which is probably not the modeler's intention.

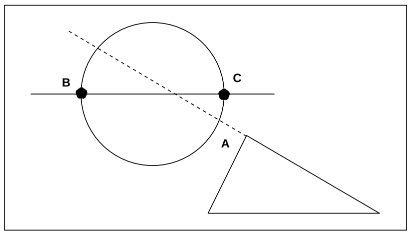

Figure 3: Line A separates vertices B and C, but it is undesirable to use this fact in their generic names.

This example shows that using arbitrary carrier sign to distinguish between arbitrary entities is to be avoided. Nonetheless, it is useful to let the user explicitly indicate specific carriers that can be used to separate specific entities. The system does not have to expose the user to the intricacies of the naming mechanism in order to support this functionality. It is reasonable to expect the user to understand that a family, not a single object, is being designed, and to help the system in making the family model well defined. In the same manner, user guidance may be also required in designating an inactive entity as essential.

To summarize this discussion, we feel that although separation is an important tool in the study of spatial decompositions, it is useful for invariant naming only in certain situations and perhaps only with explicit user guidance.

Adjacency

The most important name-dependent ingredient is **adjacency.** Suppose that an entity E is adjacent to an entity F. Then we can add the ingredient $(\texttt{Adj}, F)$ to E's name. Of course, F must be a generic name. It does not have to be unique, but it must be generic. We sometimes refer to the process of adding such an ingredient as *propagation* of the name of F to the name of E.

Ordered Adjacency

A related name-dependent ingredient is **ordered adjacency.** Suppose that entity E is adjacent to entities $F = \{F_1, \ldots, F_m\}$ (the F_i's are generic names, but not necessarily unique). Suppose further that the set F can be generically ordered *around* E. Then $(\texttt{OrdAdj}, F)$ is an ingredient in a generic name for E.

By 'generically ordered' we mean that the order is required to remain invariant in all members of the modeled SGC family. For example, edges can be sorted counter-clockwise (CCW) around a manifold vertex when viewed from the outside of the solid. Note that the term 'outside' itself must itself be generic, a requirement amounting to generic directions of face normals.

Ordered Use

Ordered use is a name-dependent ingredient very similar to ordered adjacency. Suppose that an entity F_1 is contained in a set F of entities that can be generically ordered around an entity E. Then $(\texttt{OrdUse}, E, F)$ is an ingredient in a generic name for F_1. The difference from ordered adjacency is that there the ingredient was used as part of a generic name for the 'center' entity, the one around which others are ordered, while here the ingredient is of a name for an entity in the ordered set.

A restricted form of both ordered adjacency and ordered usage was called 'local orientation' in [Capoyleas96], where it was always used for manifold vertices and radial edges. Ordered adjacency and use in our sense are more general than local orientation. As a simple example, we can use them for *non*-manifold vertices that can be viewed as manifold vertices when removing all adjacent edges but those in the set F.

Local Ordering

Local ordering is either entity-dependent or name-dependent. Local ordering is most useful when ordering vertices along a curve. The curve can be a carrier or an intersection curve between carriers; in any case it is defined generically. A name-dependent ingredient of a name for a vertex V may be $(\texttt{LocalOrd}, \texttt{Rel}, C, V_1, n)$, meaning that vertex V lies n vertices away relative to vertex V_1 along an open curve C. A generic direction for the curve must be used.

An entity-dependent ingredient for V may be $(\texttt{LocalOrd}, \texttt{Num}, C, n)$, meaning that vertex V is vertex number n along curve C (the starting end is not specified, hence this ingredient could be applied to several vertices). Here the ingredient depends upon the existence of other vertices but not on their names.

Local ordering can also be sometimes used on surfaces. For example, take a parametric surface carrier; SGC edges are curves in the parameter space. If the curves can be classified in a well-defined manner, we could use this fact as an ingredient. Some of the 'feature orientation' ingredients in [Capoyleas96] are of this type. For example, curves in a parameter space having rectangular topology (e.g. surfaces of partial revolution) can be classified either as loops inside the parameter space or as splitting it along one of the rectangle directions. The ingredient $(\texttt{LocalOrd}, \texttt{HorzSplit}, S, n)$ of an edge E denotes that it is the n-th horizontal splitting curve in the parameter space of surface S. This ingredient assumes that the horizontal direction is defined generically (e.g. by the nature of the carrier).

Local Naming

Local ordering can be regarded as a special case of **local naming**, which is a recursive invocation of the whole generic naming problem locally on a carrier. For example, two non-intersecting closed loops in a 2-D parameter space, corresponding to two edges, can perhaps be separated by an auxiliary line carrier in the parameter space.

Connectivity Component Configuration

It may be required to preserve relative configurations of GGC connectivity components across the GGC family. The most obvious example is that a certain component must be contained in another. In general, any knot-type relationship between two or more components can be used in order to break symmetry. Apart from containment, such relationships are difficult to verify. A thorough discussion about this subject is outside the scope of this paper.

Arbitrary Naming

An extremely important ingredient is **arbitrary naming,** which tells the GGC to arbitrarily select and name one of a set of entities having the same name. After this selection and naming, the new arbitrary name $(\texttt{Arb})$ can be used as an ingredient in names of other entities. For example, the GGC of Figure 5 cannot be uniquely named, but once a single entity is arbitrarily named then all other entities can be uniquely named.

To conclude the discussion on ingredients, note that cells (3-D entities) can generally be named only using dependent ingredients; there are no independent ones that can distinguish between cells. An exception is the outermost cell, which is always unique in Euclidean space.

4.2 General Naming Algorithms

There are two general variants of naming algorithms: algorithms computing a complete naming by giving generic names to all entities, and algorithms answering the E2N query by computing a name for a single entity, computing other names only if needed. In the following we give general algorithms to solve each variant if this is possible.

The algorithms we give are general in that they do not require a fixed name structure. Different applications may desire to prioritize usage of name ingredients differently. A desired order of name ingredients is given as a parameter to the general naming algorithm. A particular ingredient A may appear more than once in this ordering, because it is possible that ingredient A can be re-applied after applying another ingredient B, even if A was already applied before B. The primary example for this possibility is given by the adjacency ingredients: in principle, they can be re-applied after each additional ingredient has been applied. The general algorithm can be viewed as a template giving rise to a large number of concrete algorithms.

The ‘Refinement by Ingredients’ Complete Naming Algorithm

Input: (1) an SGC whose entities possess an ‘essential’ attribute, (2) a priority ordering on ingredients.
Output: an invariant naming for the entities if one exists; otherwise, an indication to this fact.

0. Store all entities in a single equivalence class whose name is the empty name.

1. Select the next ingredient.

2. Update all non-unique entity names using this ingredient (simultaneously).

3. Iteratively refine all equivalence classes, until no further refinement is possible.

4. If all equivalence classes contain a single member, return ‘success’.

5. If an additional ingredient can be applied resulting in a change in the equivalence class structure, go to step 1.

6. Return ‘failure’.

At every point during the execution of the algorithm we have a decomposition of the entity set into equivalence classes, such that all entities in an equivalence class have the same generic name. Initially, there is a single equivalence class, because all entities are unnamed.

The main loop of the algorithm (steps 1-5) introduces a name ingredient. It is not necessarily a new ingredient, because, as noted above, ingredient may be used several times. The ingredient is applied to all existing names possible (step 2). In step 3, it is attempted to refine an equivalence class into several classes distinguished by the introduction of the ingredient. Refinement is performed iteratively, because a refinement of one equivalence class may make it possible to refine another[1] . For example, one refinement can make an entity name unique, and now this entity can be used in order to uniquely name an adjacent entity (assuming we can use the adjacency ingredient). The main loop is performed until all entities are uniquely named (step 4) or until no ingredient can be further applied. This happens when no ingredient application can cause any modification to the equivalence class structure.

The algorithm finally halts, because the number of ingredients and entities is finite, and no ingredient can require checking an infinite number of given configurations. Even if an ingredient tries to break symmetry considering all possible entity subsets, the number of those is still finite.

When implementing the algorithm, decisions should be made regarding the data structures used for the representation of equivalence classes and their pointer connectivity with the SGC entities. In addition, name compression could be used to accelerate name comparisons. This is particularly necessary for adjacency ingredients. Name compression can be achieved by deleting common ingredients and leaving only the ones by which equivalence classes differ [Capoyleas96].

[1]In some applications it may be desired not to perform iterative refi nement in order to control the effect of an ingredient.

Naming a Single Entity (E2N)

To name a single entity, we have at least two options. The simpler one is to run the complete naming algorithm above, and stop when the equivalence class containing the desired entity contains a single member. This approach has the advantage that if a unique name exists for the entity then it will be found. However, perhaps some computations were wasted on naming other entities unnecessarily.

A different alternative is to try to name the entity using some independent ingredient, and check if the resulting name is unique. If it is, then we can return it as the answer. If it isn't, we run the algorithm recursively on the adjacent entities, updating all names after each invocation. The main drawback of this algorithm is that the computed name cannot use names that were already computed by previous E2N queries because this would cause names to be sensitive to the order of execution of the query. In addition, the computational savings are probably not that large because checking whether a name is unique or not involves application of the ingredient to all entities. The algorithm of [Capoyleas96] is a variant of this algorithm; it is applicable there because in their system the order of E2N queries is fixed.

Ingredient Priorities

The major decision to be made when designing a naming scheme is the order in which naming ingredients are applied. It is natural to decide that carrier identities will always have the first priority. In some cases, e.g. for linear polyhedra, this ingredient alone is enough to solve the invariant naming problem completely.

Prioritizing other ingredients is application dependent. In general, independent ingredients are more attractive than dependent ones and entity-dependent ingredients are more attractive than name-dependent ones, since it is preferable that names depend upon each other as little as possible.

An Example

In Figure 4 we give an example for the general naming algorithm. There are four carriers: cylinders C_1, C_2, plane P, and point V (an isolated point is a valid carrier). Looking at the object from a top view, we name the faces $FA - FD$, the edges $EA - EF$ and the vertices $VA - VD$. After using the dimension and carrier identities ingredients, we have the following equivalence classes: $V_1 = (VA, VB, VC, VD), E_1 = (EA, EC), E_2 = (EB, ED), E_3 = (EE, EF), F_1 = (FA), F_2 = (FB, FD), F_3 = (FC)$. Applying the ordered adjacency ingredient, we can refine V_1 to obtain $V_1 = (VA, VC), V_2 = (VB, VD)$. If we use E_i

as a substitute for the full generic name of its members, the generic name of V_1 is $((\texttt{Id}, (C_1, C_2, P)), (\texttt{OrdAdj}, (E_1, E_2, E_3)))$ and of V_2 is $((\texttt{Id}, (C_1, C_2, P)), (\texttt{OrdAdj}, (E_2, E_1, E_3)))$.

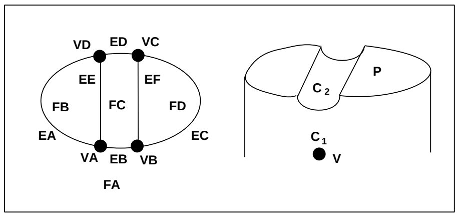

Figure 4: A generic naming example.

It is not difficult to see that no further application of any ingredient (apart, of course, from the arbitrary naming one) can refine the equivalence class structure. Vertices VA and VC are completely symmetric, and so are (VB, VD), (EA, EC), (EB, ED), (EE, EF), and (FB, FD). If, on the other hand, a single entity could be given a unique name then all entities could be uniquely named. For example, we could endow edge EB with the ingredient $(\texttt{Geo}, \texttt{InterTangent}, (C_1, V))$, meaning that this edge intersects the plane incident at point V and tangent to cylinder C_1. Once edge EB is uniquely named, its name can be propagated to uniquely name all other entities.

5 A Characterization of Invariant Naming in 2-D

In 2-D, we can completely characterize the invariant naming problem for the case in which all carriers are either signed (implicit curves) or parametric, and parameter direction is allowed to participate in name ingredients. In virtually all practical applications carriers are implicit or parametric. However, making names dependent upon the direction of curve parametrization somewhat impairs the generality of the characterization.

The theorem below states a necessary and sufficient condition for unique naming in the case described. It shows that there are situations in which unique naming is impossible, and gives a constructive algorithm to name all entities uniquely given a unique name for a single entity. In practice, containment between connectivity components should be taken into account when naming 2-D GGCs.

Theorem: *A connectivity component of a 2-D generic geometric complex having carriers that are either signed or parametric can be uniquely named if and only if it contains a single entity that can be uniquely named.*

Proof: Only if: Figure 5 shows a 2-D generic geometric complex that does not contain any named entity. The vertices can be divided into two equivalence classes, containing vertices $(V1, V3)$ and $(V2, V4)$. It is easy to verify that none of the entities can be named according to any of the ingredients.

If: suppose we have an entity that possesses a name. We will show that the name can always be propagated to adjacent entities.

Case 1: the named entity is an edge E. If it has no adjacent vertices, we are finished. If it has a single adjacent vertex, we can name it as 'the single vertex adjacent to E'. The remaining case is when it has two adjacent vertices. If E's carrier is signed, then the positive face F adjacent to E is uniquely defined. Hence the edges and vertices of F can be ordered counter clockwise (CCW), and this ordering can be used in order to distinguish between the two vertices. If E's carrier is parametric, then the parameter direction is used to distinguish between the vertices.

Case 2: the named entity is a vertex V. If it has no adjacent edges, we are finished. If it has a single adjacent edge whose carrier is C, C being any carrier, then the edge is named as 'the single edge adjacent to V and having carrier C'. The remaining case is when V has two adjacent edges E_1 and E_2 having the same carrier C (there cannot be more than two such edges, because carriers do not possess self intersections and are dimensionally uniform). If C is parametric, we can distinguish between the two edges according to the parameter direction. If C is signed, denote the positive face of $E_1(E_2)$ as $F_1(F_2)$. If F_1 and F_2 is the same face, order its adjacent edges and vertices CCW and use this order to distinguish between the two edges. Otherwise, order each of F_1 and F_2 CCW. On exactly one of these faces the studied edge precedes V. Use this fact to distinguish this edge from the other one. $\diamond$

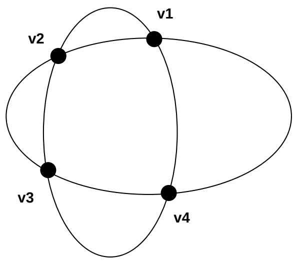

Figure 5: In this generic geometric complex no entity can be uniquely named.

6 Discussion

In this paper we presented the generic geometric complex and discussed in depth the queries it supports and ingredients of generic names. We believe that this is the first general discussion on families of decomposed pointsets. We have not discussed in depth the context in which the GGC is used by applications, and the way specific applications would customize the general naming algorithm and deal with situations in which generic naming is not possible using the ordinary ingredients.

A particularly interesting application is for parametric geometric operation graph (GOG) modeling schemes. The dependency of the resulting parametric model on the creation sequence is often stated as a crucial drawback of the whole parametric paradigm, making it difficult for the designer to modify previous design decisions. The GGC could serve as a 'flat' structure on top of which different feature hierarchies are built in an order-independent way. In addition, in many GOG applications it is prohibitive to store an example object after each synthesis operation, hence concrete representations for generic names should be investigated, including the compression issue.

Another important topic for future work are concrete algorithms enabling treatment of GGCs as parametric set models. The main computational difficulty here is how to ensure that generic names are still valid during the manipulation of the parameter vector.

In some (perhaps even most) applications, constraints between carriers and entities (e.g. two lines must be parallel) are present in the modeled SGC family. In our view, the GGC as described in this paper provides a basic framework for modeling an SGC family; the family can be further refined by adding constraints. That is, a set of constraints defined over the carriers and entities of a GGC select a subset of the modeled family. This subset is naturally modeled in the classifying sense, because it is easy to verify that the constraints are obeyed in a given SGC. Note that the constraints should be defined generically, using either generic names or an example SGC. In this view, the requirement for a generic name for an entity is seen as a constraint upon the entity and perhaps on others.

Acknowledgement. I thank Vadim Shapiro for interesting discussions on the subject of this paper.

References

[Bouma95] Bouma, W., Fudos, I., Hoffmann, C.M., Cai, J., Paige, R., Geometric constraint solver. *Computer-Aided Design*, 27(6):487-501, 1995.

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

[Chen95] Chen, X., Hoffmann, C.M., On editability of feature-based design. *Computer-Aided Design*, 27(12):905-914, 1995.

[Emmerik90] Emmerik, M.J.G.M. van, Interactive design of parameterized 3D models by direct manipulation. Ph.D. Thesis, Delft University Press, 1990.

[Hel-Or94] Hel-Or, Y., Rappoport, A., Werman, M., Relaxed parametric design with probabilistic constraints. *Computer-Aided Design*, 26(6):426-434, 1994. Also in: proceedings, *2nd Symposium on Solid Modeling and Applications (Solid Modeling '93)*, ACM Press, 1993, pp. 261-270.

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

[Kripac97] Kripac, J., A mechanism for persistently naming topological entities in history-based parametric solid models. *Computer-Aided Design*, 29(2):113–122, 1997. Also: proceedings, *3rd Symposium on Solid Modeling and Applications (Solid Modeling '95)*, pp. 21–30, ACM Press, 1995.

[Lequette96] Lequette, R., Considerations on topological naming. Presented at the *IFIP Workshop on Geometric Modeling in CAD*, May 1996, Airlie, VA.

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

[Pratt96] Pratt, M.J., Provision of an explicit constraints schema in the STEP standard. *Theory and Practice of Geometric Modeling (Blaubeuren II)*, Tübingen, Germany, October 1996. Proceedings to be published by Springer-Verlag.

[Rappoport93] Rappoport, A., A scheme for single instance representation in hierarchical assembly graphs. *IFIP Conference on Geometric Modeling in Computer Graphics*, Genova, Italy, June 1993. Published in: Falcidieno, B., Kunii T.L. (Eds), Geometric Modeling in Computer Graphics, pp. 213-224, Springer, 1993 (updated version available).

[Rappoport95] Rappoport, A., Geometric modeling: a new fundamental framework and its practical implications. Proceedings, *3rd Symposium on Solid Modeling and Applications (Solid Modeling '95)*, May 1995, Salt Lake City, pp. 31-42.

[Rappoport96] Rappoport, A., Breps as displayable-selectable models in interactive design of families of geometric objects. *Theory and Practice of Geometric Modeling (Blaubeuren II)*, Tübingen, Germany, October 1996. Proceedings to be published by Springer-Verlag.

[Requicha80] Requicha, A.G., Representations for rigid solids: Theory, methods and systems. *ACM Computing Surveys*, 12:437-464, 1980.

[Rossignac86] Rossignac, J.R., Constraints in constructive solid geometry. *ACM Symposium on Interactive 3D Graphics*, pp. 93-110, ACM Press, 1986.

[Rossignac88] Rossignac, J.R., O'Connor, M.A., SGC: a dimension-independent model for pointsets with internal structures and incomplete boundaries. In: Wozny, M., Turner, J., Preiss, K. (eds), *Geometric Modeling for Product Engineering,* North-Holland, 1988. Proceedings of the 1988 IFIP/NSF Workshop on Geometric Modeling, Rensselaerville, NY, September 1988.

[Rossignac89] Rossignac, J.R., Borrel, P., Nackman, L.R., Interactive design with sequences of parameterized transformations. *Intelligent CAD Systems 2: Implementational Issues,* P. ten Hagen, T. Tomiyama (Eds), Springer-Verlag, 1989.

[Shah96] Shah, J., Mäntylä, M., Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications. Wiley, New-York, 1996.

[Shapiro93] Shapiro, V., Vossler, D., Separation for boundary to CSG conversion. *ACM Transactions On Graphics*, 12(1):35-55, 1993.

[Shapiro95] Shapiro, V., Vossler, D.L., What is a parametric family of solids? Proceedings, *Third Symposium on Solid Modeling and Applications (Solid Modeling '95)*, pp. 43-54, ACM Press, 1995.

[Shapiro96] Shapiro, V., Maintenance of geometric representations through space decomposition. To be published in *Intl. J. Computational Geometry and Applications.*

[Solano94] Solano, L., Brunet, P., Constructive constraint-based model for parametric CAD systems. *Computer-Aided Design*, 26(8):614-621, 1994. Also appears in: Falcidieno, B., Kunii T.L. (Eds), Geometric Modeling in Computer Graphics, pp. 61-84, Springer, 1993.