Breps as Displayable-Selectable Models in Interactive Design of Families of Geometric Objects

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. In classical geometric modeling, the primary objects of interest are geometric pointsets, the primary representation scheme for which is the boundary representation (Brep). Modern geometric modeling focuses on parametric families of pointsets, defined using geometric operation graphs (GOGs), features and constraints. During interactive design of families of objects, users interact with an *example object* from the family. The example object is a pointset, hence is usually modeled using the Brep.
In this paper we study the issue of which modeling scheme is most appropriate for the example object. We identify two major operations which such a modeling scheme must support: *display* and *selection.* Selection can be further decomposed into *picking, invariant naming,* and *persistent naming.* We introduce the term *Displayable-Selectable Models (DS-models)* as a generic term for models providing display and selection functionality. We discuss the suitability of Breps to serve as DS-models and whether other, perhaps simpler, representations could also serve as DS-models.

1 Introduction

Geometric modeling deals with the representation and manipulation of geometric entities in a computer. The major classification of studies in geometric modeling is according to the nature of the geometric entities being studied. In classical geometric and solid modeling, the objects of interest were pointsets. Researchers have focused upon seeking unambiguous representation schemes for various types of pointsets [Requicha80].

In current geometric and solid modeling, the entities of interest are *families* of geometric objects [Hoffmann96, Shapiro95, Rappoport96a]. Such families are usually parametric (that is, members of the family are indexed using a set of external parameters), and may be specified using construction steps, constraints, and high-level application-dependent features [Hoffmann93a, Shah96]. Research on issues related to modeling families of objects gradually receives greater attention than research on the classical pointset topics.

Interactive design of geometric models is greatly aided by visualizing the current state of the designed model. Visualization of a family of geometric objects

is extremely difficult. For that reason, interactive design of object families is usually done by interacting with an *example object* which belongs to the modeled family. Users express their intentions on top of the example object and use it to reconstruct a mental image of the designed family.

The example object is an important participant in interactive design systems, hence the question of which modeling scheme to use for modeling it is important. Since the example object is a pointset, in principle all modeling and representation schemes designed for modeling pointsets can be used to model it.

The boundary representation (Brep) has been a major representation scheme for pointset objects from the early history of solid modeling [Mäntylä88]. A large number of research papers have been written about Breps. Most of these papers have dealt with the design and analysis of specific Brep data structures [Woo85, Mäntylä88, Alla91, Guibas85] and with algorithms operating on Breps, such as Boolean operations [Requicha85, Hoffmann89]. The essence of a Brep is best described as explicitly representing open, dimensionally uniform, connectivity components of intersection entities generated by a set of geometric pointset carriers. This essence was formalized in the notion of a Selective Geometric Complex (SGC) [Rossignac88]. It is generally felt that Breps are well-understood.

Current interactive design systems almost exclusively use the Brep to represent the example object. This choice has important implications on the architecture and performance of the system, because an accurate Brep is not easy to compute robustly and efficiently.

In this paper we discuss the issue of modeling the example object in interactive design systems. Using a geometric modeling framework we have previously developed [Rappoport95], we analyze the requirements from a modeling scheme for the example object, concluding that the two major queries which such a modeling scheme must support are (1) displaying the object, and (2) selecting object boundary entities in a persistent manner. The latter query is further decomposed into three operations: picking, invariant naming, and persistent naming. Picking identifies boundary entities of the example object. Invariant naming translates picking results to an entity name which is identical to the name of that entity in all members of the family and which depends only on the boundary (or SGC) of members in the family. A persistent name is similar to an invariant name, but it can also depend upon additional information, e.g. the design history of the family.

We introduce the term *Displayable-Selectable Model (DS-model)* for a model supporting the above queries. We then examine the suitability of the Brep to serve as a DS-model, asking two complimentary questions: (1) does the Brep support the two queries in an efficient manner, and (2) are there Brep characteristics which are essential for any modeling scheme supporting the two queries (or equivalently, are there modeling scheme which are better suited as DS-models). We show that a complete Brep is an over-kill for display and picking, and that it may be essential for invariant naming. Invariant naming is an important open problem in geometric modeling which is not fully understood at present.

In summary, the main contributions of the paper are:

– Raising the issue of a modeling scheme for the example object in interactive design of families of geometric objects;
– Introducing the term *Displayable-Selectable Model (DS-model)*, encapsulating the requirements from such a modeling scheme;
– Examining the suitability of the Brep to serve as a DS-model.

The structure of the paper is as follows. In Section 2 we briefly review the geometric modeling framework described in [Rappoport95]. In Section 3 we sketch a particular class of representation schemes used to model families of geometric objects, which we refer to as Geometric Operation Graphs (GOGs). In Section 4 we discuss interactive design of GOGs and introduce the term displayable-selectable models.

In Section 5 we examine the essence of the Brep by briefly describing a new generalization of the selective geometric complex. In Sections 6, 7 and 8 the display, picking, and invariant naming operations are studied in detail. For each operation we discuss the amount of support given by the Brep to the operation and the characteristics of Breps which are essential to support the operation.

Although this paper describes several new concepts, its goal is not to provide a detailed report on research results but rather to sketch the larger picture and guide future research. A deeper understanding of the issues brought forth in the paper would perhaps enable us to finally design and utilize hybrid representations which combine the merits of Breps and other representation schemes while masking their drawbacks.

2 Modeling and Representation Schemes

In this section we briefly review the geometric modeling framework presented in [Rappoport95], upon which we base the analysis in this paper.

Requicha's seminal paper [Requicha80] introduced terminology and definitions for the concepts of a representation and a representation scheme, and described several representation schemes for three-dimensional solids. A representation was defined to be a structure of symbols from an alphabet, and a representation scheme to be a relation between an abstract modeling space containing the mathematical entities we want to model and the set of representations. This 'symbol structure' definition is based on the *data* contained in a representation.

Most modern approaches to system analysis and design in general and software engineering in particular emphasize not the data contained in a representation but the capabilities that the data enables and the interface to them. Abstract data types (ADTs), encapsulating data through usage of access functions, have long been advocated as an elegant and practical design and implementation paradigm. Object-oriented analysis and design take this view further by studying the inter-relationships between different object classes [Rumbaugh91].

In [Rappoport95], we presented a geometric modeling framework which enhances the symbol structure definition by combining operations and data. An abstract *modeling space* contains the entities we want to model. An entity specifies a set of operations which it supports. There are two kinds of operations:

queries and *synthesis operations.* The queries are the operations for which the entity is modeled in the first place; they provide the functionality of the model as important to the external world. Synthesis operations are the operations using which entities are created, modified, edited and combined. Synthesis operations are not part of the external functional interface to the entity and are reflected to the outside only as user interface operations or when they are mirrored in the supported queries.

As in Requicha's definitions, the data of each entity has a representation. However, now the main concept is that of a *model.* A model is a representation which supports the required queries and synthesis operations. A *modeling scheme* is a concrete implementation of the models, including their queries and synthesis operations.

In geometric modeling, as in any other system modeling discipline, models are built and stored for the sake of *doing* something with them. It is important to analyze clearly what our models should do (queries) and how we want to specify them (synthesis operations) before choosing a specific modeling scheme.

3 Geometric Operation Graph Representations

By definition, a Brep can only represent knowledge about the pointset of a geometric object. Using a selective geometric complex (SGC), internal structures can be represented as well as the object's boundary. However, clearly there are plenty of applications in which we want to model more than the pointset of the object. We would even say that this is the case in the majority of applications.

In design applications we want to document the history of the design and explanations for design decisions. These are important when additional groups of people are being made involved in some aspect of the model, for example when a manufacturing team finds it necessary or economical to modify the design and it needs to determine whether and which modifications are allowed. Such documentations are important also in concurrent engineering. It is clear that a Brep cannot be used to store such information.

Many applications which do not need history or versioning still need to be capable of viewing the model as more than a pointset. Storage of application-specific features is essential in many cases. Some features can be represented by using attributes associated with entities of a Brep. However, it is not clear whether this is always possible, since there are features which comprise components not present in the Brep at all. For example, a slot can be created by subtracting a rotating block whose axis of rotation is not part of the Brep. It is possible that an SGC can be used to represent such features, but this has not been shown yet.

Modeling spaces containing features, history, versioning, and general modifying operations to geometric objects can be represented by a class of models which we refer to as the *Geometric Operation Graphs (GOG)* [Rappoport96d]. Other tightly related terms are *generative* [Hoffmann96] and *parametric* [Shah96] representations, as discussed below.

A GOG (see Figure 1) is a directed graph whose nodes contain arbitrary operations (or functions), which are usually geometric in nature. An arc from node A to node B denotes the fact that the corresponding output of the operation in A serves as an input to the operation in B. Usually, but not always, one of the outputs of an operation node is a geometric object. The GOG is almost always used in order to represent a *family* of geometric objects rather than a single object, and this family is usually parametric (that is, members of the family are indexed through a parameter vector; see below).

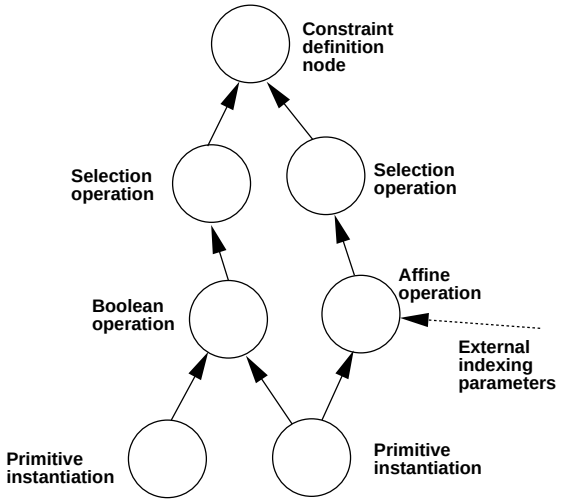

Fig. 1. A General Operation Graph (GOG).

The GOG is a generalization of several known representation schemes in geometric and solid modeling. Some procedural representations can be written as GOGs. However, a GOG can in principle support operations which establish a relation between two objects already present in the system. The semantics of such an operation is that the relation, or constraint, should be maintained from now on. Hence GOGs are more powerful than ordinary procedural representations. On the other hand, although a GOG can be enhanced to support control constructs such as loops, this would take it a bit afar from its intended spirit.

A hierarchical assembly graph is a simple GOG in which operation nodes contain affine operations. [Rappoport93] presented a scheme to directly store information belonging to a single instance of a part in an assembly in a GOG node.

The most obvious example for a GOG in geometric modeling is of course CSG, in which the operations in the nodes of the graph are primitive instantiations, affine operations, and Boolean operations. In CSG every node has a single output, which is a geometric object. The GOG point of view is that it is the operation graph which is the essence and not the specific operations used. CSG

is mostly associated with the Boolean operations (indeed, the affine operations can be propagated into the leaves of the graph and factored out), while the GOG emphasizes the algebraic structure of the resulting model.

Several specific GOG systems have been studied by Rossignac: offset and blending nodes in CSG [Rossignac86a], constraints in CSG [Rossignac86b], and operations as general procedures [Rossignac89]. A GOG with constraints between coordinate systems and an underlying procedural programming language was described in [Emmerik90].

The GOG is an abstract generalization of what some people call 'parametric' representations. However, in practice, the term 'parametric' is defined today by the architecture of a specific commercial product and combines two notions which we feel are separate: (1) specifying the name of a member in a family of represented geometric objects by a parameter vector [Rappoport96a], and (2) the sequential solution of a system of constraints. We feel that the term 'parametric' is perfect for the first notion but is a poor choice for the second notion. Because of the types of modeling spaces that the GOG will be used for, in most cases it will support the first notion. However, the GOG can have operation nodes which declaratively specify constraints without necessarily specifying a solution sequence. For example, such an operation node may be implemented by a numeric constraint solver (a scheme commonly referred to as 'variational', although the word 'variational' is not appropriate since it implies a family of objects which are all variations on the same theme rather than a specific method to specify and compute such a family). Declarative constraints are actually present in some 'parametric' systems, but only at the bottom level of the graph in the form of defining a 2-D sketch to be extruded later. Hoffmann's Erep [Hoffmann93a] is such a GOG. Note that the term 'generative' is sometimes used when referring to such architectures [Hoffmann96]; if desired, the acronym GOG can be interpreted as 'Generative Operational Geometry'.

We view the GOG as a representation for design intent and for the different types of knowledge accumulated and specified during the various stages of the life of a model. At present, the GOG concept is not completely understood, especially the role of declarative constraints nodes. However, there is enough evidence to support the view that the GOG is a major class of modeling schemes in modern geometric modeling.

4 Displayable-Selectable Models

Breps are probably not suitable to serve as the main modeling scheme for entities which are more than simple geometric pointsets. The class of geometric operation graphs is an attractive modeling scheme for such entities. In this section we give a general description of interactive applications involving operation graphs and characterize the functionality required from the 'example object' used by such systems.

One of the primary applications of geometric operation graphs is to represent parametric families of objects. Thus, a major query for GOGs is the indexing

query: given a parameter vector, compute the corresponding object. This object should itself be represented in some representation scheme. There are several usages for such as object, two of which are the most common: visualization and analysis. Since the Brep is considered to be an efficient representation for these operations, it is natural to use Breps as the representation in which the output of the indexing query will be represented. In this context a Brep is an 'evaluated' GOG, in the same way that the process of conversion from CSG (a specific GOG) to a Brep is called 'boundary evaluation'.

In this paper we are more interested in the specification and creation of GOGs than in the queries they support. We assume the following scenario for interactive applications used to define GOGs [Hoffmann93b] (see Figure 2). At any point in time, the user is shown a visualization of the designed GOG through an *example object* which belongs to the family represented by the GOG. The user is capable of adding or deleting a new operation node, and sees the result of applying the operation on the example object. By visualizing the result of the operation, the user can determine whether the operation achieves the desired effect. The user may also want to modify the example object by specifying different parameters. These can be specified numerically, through direct manipulation on the graphical view (shown as a dotted line in Figure 2), or using some other method.

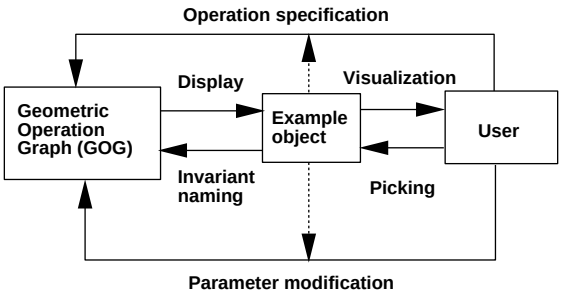

Fig. 2. The relationship between the GOG and the `Display`, `Picking`, and `Invariant Naming` operations.

The first operation which the system must thus support in this kind of interactive scenario is `Display`: given a GOG, a parameter vector, and display parameters, display the resulting object. We deliberately do not say that the system must support boundary evaluation, of course, since our goal in this paper is to specify what needs to be done (here, visualization) rather than how to do it (boundary evaluation). We discuss `Display` in Section 6.

The other operations which the system must support in the scenario described above are related to the way in which the user specifies new operation nodes. An operation node needs to know the type of the operation, which can be easily specified by the user, and the arguments that the operation should receive. There are two main types of arguments: numeric and symbolic data serving as

additional indexing parameters, and geometric data derived from the current GOG. The former type is easy to specify. The latter arguments are specified by selecting the relevant data from the current GOG. Depending of the actual data, this selection can be done on the example object, directly on the operation graph (perhaps using some form of graph visualization), or on other views of the model [Emmerik93].

Operation nodes differ in the number and types of arguments they receive as input. Some operations operate on a whole object as a single unit. Affine and Boolean operations are such operations, which is the deep reason why CSG is so simple to define and implement. However, some operations require as input sub-parts of an object. A sub-part can be specified as the part lying in some spatial location, or more commonly, as a Brep entity. That is, vertices, edges, faces and cells of the example object can be specified as inputs to the new operation node. Systems which support the specification of Brep entities as input to operation nodes are much more powerful and allow a much greater amount of knowledge on the model to be present. It is in this type of selection that we are interested. It is convenient to think of such a selection as a GOG operation node, as shown in Figure 1.

For technical reasons we divide the selection process into three stages, which we regard as operations sufficiently different to deserve different names: picking, invariant naming, and persistent naming. We collectively refer to picking and invariant naming as *selection*, since the word 'selection' best describes how the user thinks about the functionality of these operations.

In the first stage, the user identifies the desired entity of the example object. The output of this operation is the name of this specific entity of the example object; normally, this output would be represented as a pointer to the entity. Respecting common computer graphics terminology, we call this the `Picking` operation. We discuss `Picking` in Section 7.

The output of `Picking` is useful only in conjunction with the example object. However, the user is allowed to arbitrarily modify the indexing parameters to produce a different example object. The arguments given to any operation node in the GOG should thus not be tied to a specific example object. Hence, we must translate the output of `Picking` to a representation which depends only on the properties which are present in all objects in the GOG family. Specifically, it should not depend on the unique geometry of the example object.

The problem of storing arguments to operation nodes such that they will be valid for every choice of parameter vector is usually called *persistent naming* [Kripac95, Hoffmann93b, Lequette96]. The easier problem of giving Brep entities names which are invariant under modifications to the geometry of the Brep carriers is called *invariant naming* [Rappoport96b]. Persistent naming is strongly tied to the semantics of the GOG operation nodes and to the specific application for which the GOG is designed. Due to the inherent technical differences between the two problems and to the difficulty of defining and solving them, they can be discussed separately. In the Erep project [Hoffmann93a], the general persistent naming mechanism is reported in [Chen95], while the issues related to invariant

naming (in the Erep context) are reported in [Capoyleas96].

Both invariant naming and persistent naming are difficult problems, but since the scope of the former is more limited it is easier to study it mathematically. In addition, once invariant naming is available, persistent naming can be implemented as a post-process taking into account the intended semantics of GOG operations. For these reasons we do not deal with persistent naming in this paper, although in principle we should, since Breps may be strongly tied to any persistent naming mechanism. The GOG operation `Invariant Naming` is discussed in Section 8.

The functionality required from the example object is best encapsulated by the term *Displayable-Selectable Models (DS-models)*. A DS-model is a model (in the sense of Section 2) which supports the 'display' and 'selection' queries in their various manifestations. This term emphasizes the functionality required from the model and is neutral regarding specific implementations of that functionality. In the rest of this paper we examine the suitability of Breps as DS-models and whether they possess characteristics which are essential for any DS-model.

5 The Essence of the Brep

The boundary representation is usually described as 'representing the boundary of the object using entities such as faces, edges and vertices'. This vague definition was made precise by Rossignac and O'Connor, who introduced the concept of the *Selective Geometric Complex (SGC)* [Rossignac88]. In this section we give an informal overview of the SGC, including a new relatively minor practical generalization. A major generalization of the SGC, the *Generic Geometric Complex (GGC)* [Rappoport96b, Rappoport96a] is described in Section 8 in the context of the invariant naming problem.

There are three major conceptual stages in the specification of an SGC: carrier definition, intersections, and selection of active entities. Each stage is briefly described below.

A selective geometric complex in R^n is defined using a set of *carriers* embedded in R^n. A carrier is any pointset such that it is convenient for us to think about it as a single unit. The two major types of carriers are implicit and parametric hyper-surfaces. Implicitly defined surfaces, parametric curves, and parametric surfaces are all carriers in R^3. Even a well-defined fractal object may be a carrier. It is mostly in the definition of a carrier that our version of the SGC differs from the original one, in which only implicit half-spaces were allowed.

In general there are no special requirements from a carrier and every pointset is qualified to serve as a carrier. However, there are certain properties and applications for which is would make real sense to limit the generality of the carriers somewhat. To simplify the discussion, in the rest of this section we only deal with 2-D and 3-D SGCs. In practice, there are three major types of carriers: implicit surfaces (defined by a single algebraic equation) spline (piecewise parametric polynomial) curves, and spline surfaces.

The only requirement we impose upon these carrier curves and surfaces is that they do not possess self-intersections. That is, in a parametric curve or surface two different parameter values should not yield the same point, and in an implicit surface there should not be singular points. In case a carrier violates this requirement, it should be decomposed into parts each of which obeys the requirement. The decomposition of the carrier should itself be represented in an SGC.

The next stage in the definition of an SGC is the computation of all mutual intersections of the carriers and splitting of the carriers accordingly into open, dimensionally uniform connectivity components. In general, the intersection between a curve and another curve or surface is a set of curve segments and points; the intersection between two surfaces is a set of surface patches, curve segments, and points. Points which are isolated carriers or intersection results are called vertices; maximal curve segments which are isolated carriers, intersection results or split carriers are called edges; maximal surface patches of isolated or split carriers are called faces; and maximal connectivity components of space are called cells[1].

We now have a set of entities: vertices, edges, faces and cells. The importance of the intersection process is that all points in an entity possess the same characteristic function with respect to the carriers (that is, they belong to exactly the same set of carriers). In a sense, all points in an entity thus possess the same 'name'. We immediately emphasize that entities are not necessarily distinguishable according to these names. Points in different entities could end up with the same characteristic function.

As an example for an SGC, consider Figure 3. There are two carriers (the ellipses), four vertices, eight edges, and six cells. Note that the vertices cannot be distinguished by carrier identities, since they all possess the same classification with respect to the carriers. The edges are different connectivity components of split carriers and hence are indistinguishable by carrier identity as well. In Section 8 we will see that this figure demonstrates a fundamental difficulty with generic geometric complexes.

The final stage in the definition of an SGC is a selection of 'active' entities. The union of the pointsets of the active entities comprises the pointset of the object which the SGC represents. The subset of the active entities whose points lie on the boundary of that object is the Brep of the object. Note that the SGC can thus represent objects with internal structures and with mixed dimensionalities. We call the objects represented by SGCs 'decomposed pointsets'. The selection process can be guided by a graph of Boolean operations defined over the carriers, as is the case when the SGC is used to represent the result of boundary evaluation of a CSG graph, or simply be guided by a Boolean operation performed between two SGCs. There are many additional topics related to SGC's which we do not discuss here. For a deeper study, the reader should consult [Rossignac88].

From an operational interface point of view, a Brep supports the queries

[1] In [Rossignac88], the term 'cell' was used for what we call here an 'entity', because we wanted to endow the word 'cell' with its common interpretation in R^3.

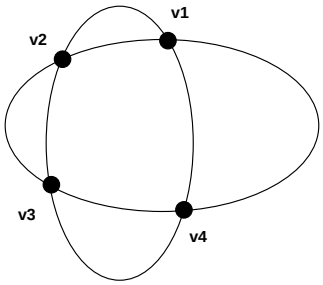

Fig. 3. A selective geometric complex (SGC) presenting fundamental invariant naming difficulties.

listed below. We do not enumerate the synthesis operations through which a Brep can be constructed and edited, since the reason of being of a model is defined only by the queries it supports.

- What types of carriers are present?
- What are the geometries of the carriers?
- What are the entities defined by the spatial inter-relationships of the carriers?
- What are the geometries of the entities?
- What are the adjacencies between the entities?
- What are the local relative orientations between the carriers in an entity?

A concrete model which can answer these queries is called a *complete Brep* in this paper. Naturally, implementations of a complete Brep should address efficiency issues, such as which adjacency relations are explicitly stored, should entities be sorted along each other when possible (for more efficient traversal) etc.

To summarize, the essential ingredient of the boundary representation as it is commonly understood is the explicit subdivision of space induced by a set of pointset carriers. Issues related to time and space efficiency are not considered to be essential ingredients.

6 The `Display` Operation

The most common operation in geometric modeling, an operation which is an inherent necessity in interactive design applications, is display of the designed object. In any interactive design application we need to see the object we design. Visualization is such an important requirement that is it useful even when the designed system or object do not possess an immediate geometric form. In this section we study the relationship between the `Display` operation in GOGs and the Brep.

Recall that `Display` receives three input arguments: a GOG, a parameter vector, and various parameters related to the desired visualization (camera parameters, display style etc). `Display` has three variants, depending upon which

of the three arguments was modified by the user (obviously, if no argument was modified then there is no need to recompute `Display`). The variants have different implications on the performance required from the `Display` operation. In the first variant, the GOG defining the family was modified, by adding or deleting an operation node. In this case the user is willing to wait for the computation of the `Display` operation a few seconds or even more than that if the changes made to the GOG are substantial. In the second variant, the parameter vector indexing the family was modified. For this variant to be useful and to truly support rapid navigation in the modeled family, it is desired that the computation takes less than a second. That is, navigation (or 're-generation') requires `Display` to be at least an order of magnitude faster than GOG modification. Current parametric systems are very far from this goal: re-generation is usually not done interactively. In the third variant only the visualization parameters were modified, in which case it is certainly plausible to require interactive computation.

Due to practical considerations we would strongly prefer to utilize standard graphics display APIs such as OpenGL for the computation of `Display`. In such APIs the basic primitive rendered is the planar polygon. They support modeling and display transformations, rasterization (computation of the pixels covered by a polygon), hidden surface removal (usually using a z-buffer), masking operations on the image and z-buffers, texture mapping, and more. For higher-quality shading, normals to polygons and polygon vertices are needed as well. For optimization purposes, it helps if the polygons given to the graphics system are triangles, or better, triangular meshes in which the triangles are traversed such that triangles which share an edge are given to the system consecutively.

To see how the Brep relates to the `Display` operation, we will ask two questions: first, how much support is given by the Brep to the operation; and second, can the `Display` operation be supported as well by other representations (or equivalently, are there characteristics of the Brep which are not needed by the `Display` operation).

6.1 Brep's support for `Display`

The primary advantage of the Brep for the `Display` operation is that all the polygons which we want to see lie on Brep faces. That is, the boundary representation is attractive not only because it suffices to represent the boundary in order to represent the object uniquely, but also because the boundary is the only thing we are interested in for the purposes of the `Display` operation. Note that when we are interested in seeing internal structures of the object, they are explicitly represented by the Brep's SGC as well.

Given a Brep, the `Display` operation can be implemented by triangulating or polygonalizing the Brep faces and sending them to the graphics system. In addition, it is easy to compute the normals of polygons and polygon vertices. Thus, Brep's support for `Display` is strong but not immediate.

6.2 `Display`'s need of the Brep

As noted earlier, the essence of a complete Brep is that intersections between carriers are computed and represented explicitly. Following are several indications that this may not be necessary for `Display`. Most of these are relevant for the relationship of Breps and visualization in general, not only in the context of general operation graphs.

Edges and vertices. When the display style is with hidden surfaces removed, `Display` does not need to display the intersection edges and vertices. They are implicitly visualized by the fact that faces are full. When the display style is wire-frame, the edges need to be shown, but this effect can be achieved using the z-buffer without actually computing the edges [Emmerik93]. Even if the edges are displayed after computing them explicitly, this computation is only needed to be carried to the resolution of the display device, which is normally orders of magnitude larger than the resolution of floating point computations.

Face interpenetration. Small inter-penetrations of faces into each other, which are avoided using great efforts by algorithms which compute complete Breps, do not harm visualization.

Curved face tessellations. Visualization of a curved face is done by tessellating it into polygons, and this has to be done even if the intersection of the face with other faces was computed exactly. This observation suggests an approach in which the order of operations is reversed or at least adaptively combined. Since `Display` needs an order of magnitude less accuracy than boundary evaluation, this approach is potentially more efficient.

Union. The result of the 'union' operation between two objects can be displayed by simply displaying the objects one after the other. The z-buffer ensures that the resulting image is correct. In this case no computation of geometric intersections is needed.

Fast display of Booleans. Algorithms for rapid visualization of CSG objects on standard platforms are beginning to appear. For example, [Rappoport96c] describes an algorithm which combines graph re-writing, hierarchical convex differences and efficient geometric algorithms (based on convex hulls of 3-D points) to visualize non-trivial CSG models interactively , using the standard graphics pipeline with a z-buffer and a stencil bit plane.

To summarize, complete Breps are not *essential* for performing the `Display` operation correctly; in some (perhaps even most) situations, computation of a complete Brep with all adjacency information is an over-kill. In applications providing interactive, high-level manipulation, model visualization can be achieved by computing crude linear approximations. This is especially true when the display mode is with hidden surfaces removed, but may also be true for wire-frame displays.

7 The `Picking` Operation

Recall that `Picking` refers to identification of Brep entities of the example object through graphical interaction, without persistent storage of the result.

7.1 Brep's support for `Picking`

There are two main methods to implement `Picking` in graphics systems: a graphics-based method and a geometric method. The latter is simply done by intersecting the scene with a ray from the eye to the mouse location, computing the nearest object. The former uses the graphics system in order to detect the identity of the visible polygon closest to the mouse. Which method is faster depends upon the relative performance of the graphics system and the CPU. The graphics-based method is probably easier to implement.

Since the mouse location is discrete, it cannot be expected to fall exactly on an edge or a vertex even if the user intends their selection. Therefore, in both methods, we should first identify faces and then infer selected edges or vertices from them. The alternative, computing the distance from the eye-mouse ray to every edge and vertex, is probably not practical. The Brep supports both methods since all faces are explicitly available and it can be inferred whether or not a ray intersects the face. Actually, Breps support `Picking` even more than they support `Display` since they give the information needed for `Picking` explicitly, while Brep faces need to be further polygonalized or triangulated.

7.2 `Picking`'s need of the Brep

It may seem that by definition a complete Brep of the example object must be computed in order to support `Picking`. However, this is not the case.

Following is a sketch of a possible algorithm which does not need a complete Brep a-priori. Let us assume that the `Display` operation is executed correctly. As a result, a graphics-based method can be used to select the visible face(s) nearest the mouse location. It should not be too difficult to identify whether or not the user has meant to select a face, an edge or a vertex by analyzing the number of faces visible in the vicinity of the mouse. Thus, even if Brep entities are not explicitly represented, they can be computed on-demand by this 'discrete local boundary evaluation' algorithm.

Note also that the performance required from `Picking` is probably slower than that required from `Display`. Since the `Picking` operation identifies an argument to another operation, it is a discrete decision in the design process which does not involve numerical parameters. Usually, the numerical parameters are the ones which we want to be capable of modifying interactively. For example, we want to modify affine operations and visualize interactively the results of Boolean operations defined on the object [Rappoport96c], or we modify constraint parameters and want to see the geometric and physical behavior of the object. As a result, a slower, incremental on-demand computation of `Picking` is acceptable. If the user does not demand highlighting of Brep entities while moving the mouse (which is a nice aid to `Picking`) then the computation of `Picking` is allowed a few seconds.

In summary, it is likely that `Picking` does not need the full power of a complete Brep, although a complete Brep supports it more efficiently.

8 `Invariant Naming` and the Generic Geometric Complex

After obtaining the concrete names of the entities of the example object which were specified by the `Picking` operation, they have to be transformed to a representation in which they are invariant under parameter modifications. The general problem is called 'persistent naming'.

Technically, it is convenient to consider two types of parameter modifications, both of which preserve certain entities of the SGC representing the example object. The first type assumes only knowledge about the SGC of the example object; the second type assumes knowledge about the whole GOG. The `Invariant Naming` operation maps concrete entity names into generic entity names assuming parameter modifications of the first kind. For modifications of the second kind a more elaborate naming scheme is needed. Such a naming scheme typically must consider the intended semantics of GOG operations and is difficult to specify mathematically. Hence in this paper we only discuss invariant naming.

In order to precisely define the invariant naming problem, we briefly describe the *Generic Geometric Complex (GGC)* [Rappoport96b]. The GGC is a model for a family of decomposed pointsets, each modeled by an SGC, supporting the following queries:

1. *Entity-to-name:* 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:* given an SGC in the modeled family and a generic name previously returned by entity-to-name, return the entity having that name.
3. *Membership classification:* given an SGC, determine whether or not it belongs to the modeled family.
4. *Example* (optional): return a member of the modeled family.

The GGC can be viewed as the family 'spanned' by an example member. That 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.

A distinction is made between the *nature* of a carrier and its geometry. The nature of a carrier is the representation scheme used for its pointset plus symbolic and integer parameters used in that representation. For example, an implicit surface and a parametric surface are of different natures, as are two parametric surfaces of different degrees.

The geometry of the carriers is allowed to change without changing properties which were defined to be required. Such properties may include the number of the carriers[2], the nature of each carrier, geometric properties of carriers, geometric relationships between the carriers, the existence of certain entities, geometric

[2] Actually, there are applications for which the number of carriers should be allowed to change as well, e.g. when the number of holes in an object is an external parameter. Discussion of such GGCs will take us too far from our present focus.

properties of entities, symbolic properties of entities, etc. The names of required entities are specified in a way which makes them independent of things which are allowed to change. Carrier geometry modifications are valid as long as they preserve the above properties, including the names of the required entities.

There are several ingredients from which names of required entities in a GGC can be composed. The major ones are:

- Carrier identities: as noted earlier, all points in an entity have the same classification with respect to the carriers. The name of an entity can utilize the names of the carriers to which all of its points belong.
- Invariant geometric properties, such as convexity, tangency etc.
- Classification with respect to carrier signs (separation): some of the carriers induce a singed function on space; all points in an entity have the same sign vector with respect to these carriers. [Shapiro93] can be viewed as studying the possibility of unique naming using this ingredient alone.
- Adjacencies: the name of an entity can recursively utilize the names of adjacent entities. It is preferable to keep the depth of the recursion as small as possible.
- Ordered adjacency and ordered usage: the name of an entity can utilize the relative local orientations of its carriers and carriers of adjacent entities.
- Local ordering: ordering along parametric entities whose parameter space can be ordered.
- Local naming: a recursive invocation of the invariant naming process on a single carrier.
- Arbitrary naming of an entity. This ingredient is needed when there is no other way to distinguish between several entities having the same invariant name. Once an entity has been arbitrarily named, names of other entities can depend upon this name through other adjacency and ordering ingredients.

Figure 4 is a simple example that shows why several ingredients are needed. On the left, the two vertices have the same carrier identities but a different local orientation of the line with respect to the circle (denoted by the dotted curves). On the right, the two marked cells have the same classification with respect to all carriers, but they can be distinguished according to the vertices they are adjacent to. The vertices can in turn be distinguished by carrier identities since they are the result of intersecting different lines.

Figure 3 is a very simple example that shows that there are situations in which all ingredients (but arbitrary naming) are not sufficient. All four vertices in the figure have the same carrier identities. Applying the orientation ingredients, they can further be divided into two sets of a pair of vertices each. However, the adjacency ingredient cannot be applied because no edge or cell can be named without naming the vertices or cells first. This SGC does not possess a unique answer to name-to-entity queries.

In [Rappoport96b] we give an algorithm based on equivalence classes and sequential introduction of name ingredients to induce invariant naming on entities in a GGC. The algorithm is very general, and always finds a unique naming if one exists.

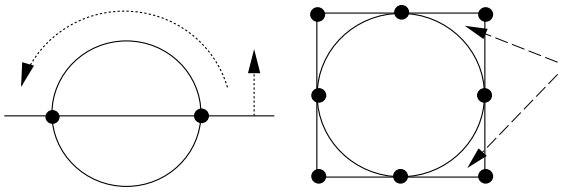

Fig. 4. Examples for the need of several ingredients in invariant entity names in generic geometric complexes.

Certain situations always admit a unique naming. For example, in [Rappoport96b] we proved that a necessary and sufficient condition for naming a connectivity component of a 2-D complex is that a name exists for a single entity in it.

The situation in 3-D is not fully understood at present. The naming scheme in [Capoyleas96] can be viewed as a scheme which utilizes some of the above ingredients in the context of a specific system, and even there it does not solve the problem completely. In some restricted (but useful) cases the problem is easy to solve. For example, if all the carriers are linear half-spaces then each pair can have at most a single intersection entity of each dimension, hence the carrier identity ingredient suffices for naming. Carrier signs are sufficient when all entities are separated by carriers, as studied in [Shapiro93].

8.1 Brep's support for `Invariant Naming`

In light of the discussion above, it is clear that the Brep provides substantial support for `Invariant Naming`. The information present in a Brep about adjacencies, and information easily derived from it about relative local orientations, are of great assistance to any invariant naming scheme. However, we must keep in mind that a Brep alone does not suffice to solve the problem in all cases.

8.2 `Invariant Naming`'s need of the Brep

With `Invariant Naming` we are in a different situation from the former two operations. `Display` and `Selection` are well supported by Breps, and the only issue was whether or not they actually need the full power of a complete Brep. Here, we do not know at this stage even if a complete Brep suffices to solve the invariant naming problem.

Carrier identity, which is probably the important of the three naming ingredients given above, can be computed adaptively in an on-demand basis. The complete adjacency structure of the Brep is not essential in the case of linear half-spaces and separated entities. Perhaps these observations serve as indications that it would be possible to find naming schemes which would not require a complete Brep.

In summary, both invariant naming and persistent naming are fundamental operations in geometric and solid modeling. Both are not well understood at

present. Finding a satisfactory and efficient naming scheme which does not need the full adjacency structure of an SGC is an extremely important open problem in geometric and solid modeling.

9 Discussion

Two important operations which we have not discussed in this paper are collision detection and meshing for finite element simulations. Many applications need to detect collisions between moving objects and act accordingly. Data structures supporting efficient collision detection should probably contain (hierarchical) spatial information not present in Breps. It is not clear whether complete Breps are essential for this application. For example, obviously penetration of one boundary surface into another in the same object does not harm collision detection.

Regarding meshing, we should note that mesh generators are usually interested in a *simplified* Brep and not necessarily in a complete Brep. For example, the primitive geometric operation in the paving family of meshing algorithm is the projection of an arbitrary spatial point on a surface. This primitive can perhaps be better implemented in an incremental, adaptive manner on the GOG model itself or on some partially evaluated version of it.

The reason we have not discussed collision detection and meshing is that at present they are considered as post-processes and are not well integrated with the design process. Clearly, this situation should be changed in the future to support effective optimizations and full product models.

In this paper we have also not discussed object and shape analysis applications. Breps are certainly needed by a large number of such applications.

In summary, we feel that complete Breps, that is, SGC-like Breps explicitly storing all topological entities and their (ordered) adjacency relationships, are probably an over-kill for two important queries required from the example object in interactive design of families of geometric objects. Both `Display` and `Picking` can be supported directly by a GOG model or by a partial Brep derived from it adaptively on-demand. On the other hand, valid and efficient support for `Invariant Naming` (and for persistent naming) is an important open problem in geometric and solid modeling for which a complete Brep may be essential.

Breps are perceived today as essential, or at least as very practical, for solid modeling. This is evidenced by the success enjoyed by the Acis Brep modeler. Modern modeling schemes supporting high-level knowledge, well-defined storage of design intent, features, parameterizations, constraints and history mostly use Breps because they need support for visualization of the model and selection of entities from it. These requirements are best encapsulated by the term Displayable-Selectable Model (DS-model), which emphasizes the functional nature of the desired representation rather than its data and implementation. The discovery of DS-models different from Breps may improve the efficiency and sophistication of solid modeling systems.

Acknowledgement: I thank Vadim Shapiro for his comments.

References

[Ala91] Ala, S.R., Design methodology of boundary data structures. *Intl. J. Comp. Geo. and Apps.* 1(4):207-226, 1991.

[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.

[Emmerik93] Emmerik, M.J.G.M. van, Rappoport, A., Rossignac, J., Simplifying interactive design of solid models: a hypertext approach. *The Visual Computer*, 9:239-254, 1993.

[Guibas85] Guibas, L., Stolfi, J., Primitives for the manipulation of general subdivisions and the computations of Voronoi diagrams. *ACM Transactions On Graphics*, 4(2):74, 1985.

[Hoffmann89] Hoffmann, C., Geometric and Solid Modeling: an Introduction. Morgan Kaufmann, 1989.

[Hoffmann93a] 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.

[Hoffmann93b] Hoffmann, C.M., On the semantics of generative geometry representations. *19th ASME Design Conference,* Albuquerque, New Mexico, September 1993.

[Hoffmann96] Hoffmann, C., Rossignac, J.R., A road map to solid modeling. *IEEE Transactions on Visualization and Computer Graphics,* 2(1):3-10, 1996.

[Kripac95] Jiri Kripac, A mechanism for persistently naming topological entities in history-based parametric solid models. Proceedings, *Third ACM/Siggraph 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.

[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 (an updated version is available from the author).

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

[Rappoport96a] Rappoport, A., Parametric and declarative modeling of families of geometric objects. Presented at the *IFIP Workshop on Geometric Modeling in CAD,* May 1996, Airlie, VA.

[Rappoport96b] Rappoport, A., The Generic Geometric Complex (GGC): a model for families of decomposed pointset. Technical Report, Institute of Computer Science, The Hebrew University, 1996.

[Rappoport96c] Rappoport, A., Spitz, S., Interactive 3-D Boolean operations for conceptual geometric design. Technical Report, Institute of Computer Science, The Hebrew University, 1996.

[Rappoport96d] Rappoport, A., The Geometric Operation Graph (GOG) representation for geometric modeling. In preparation, 1996.

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

[Requicha85] Requicha, A.G., Voelcker, H.B., Boolean operations in solid modeling: boundary evaluation and merging algorithms. *Proc. of the IEEE* 73(1):30-44, 1985.

[Rossignac86a] Rossignac, J.R., Requicha, A.A.G., Offsetting operations in solid modelling. *Computer-Aided Geometric Design*, 3:129-148, 1986.

[Rossignac86b] 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.

[Rumbaugh91] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., Object-Oriented Modeling and Design. Prentice-Hall, 1991.

[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 ACM/Siggraph Symposium on Solid Modeling and Applications (Solid Modeling '95),* pp. 43-54, ACM Press, 1995.

[Woo85] Woo, T.C., A combinatorial analysis of boundary data structure schemata. *IEEE Computer Graphics and Applications* 5:19-27, 1985.