Interactive Boolean Operations for Conceptual Design of 3-D Solids

Ari Rappoport Steven Spitz[†]

Institute of Computer Science, The Hebrew University

Abstract

Interactive modeling of 3-D solids is an important and difficult problem in computer graphics. The Constructive Solid Geometry (CSG) modeling scheme is highly attractive for interactive design, due to its support for hierarchical modeling and Boolean operations. Unfortunately, current algorithms for interactive display of CSG models require expensive special-purpose hardware that is not easily available.

In this paper we present a method for interactive display of CSG models using standard, widely available graphics hardware. The method enables the user to interactively modify the affine transformations associated with CSG sub-objects. The application we focus upon is that of conceptual design, a stage in the design process in which rapid, interactive visualization of the model and high-level design operations are of crucial importance, while the objects are relatively simple.

The method converts the CSG graph to a novel *Convex Differences Aggregate(CDA)* representation. The CDA utilizes graph re-writing techniques, efficient geometric algorithms on convex objects and a built-in hierarchical acceleration scheme. The CDA rendering algorithm is very simple, takes advantage of standard graphics hardware, and makes efficient use of system resources by splitting the work between the graphics system and the CPU.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation — Display algorithms; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling — CSG; J.6 [Computer Applications]: Computer-Aided Engineering — CAD.

Additional Key Words: geometric modeling, solid modeling, conceptual design, Boolean operations, convex differences, convex differences aggregate, convex polyhedra.

1 Introduction

Interactive design of 3-D geometric models is of major importance in computer graphics. The Constructive Solid Geometry (CSG) solid representation scheme [Requicha80] is highly attractive for interactive design, because it lets the user compose objects hierarchically using Boolean operations. Unfortunately, current algorithms for interactive display of CSG models require expensive special-purpose hardware that is not widely available.

In this paper we address the problem of displaying CSG models at interactive rates using standard, off-the-shelf graphics hardware. The application we have in mind is that of conceptual design, a stage in the design process in which rapid, interactive visualization of the model and high-level design operations are of crucial importance, while the objects are relatively simple.

Background. A design process starts with conceptual design and progresses by iterative refinement stages until meeting the design goal [Smithers89]. In geometric design, the initial conceptual design phase is one of the most difficult to computerize. Other stages, such as detailed geometric design and physical analysis, are already supported in current modeling systems. Conceptual design is rarely supported.

The reason why conceptual design is so difficult to support is that it imposes the largest demands for interactivity. Initially, designers need to experiment with a large number of potential designs. This 'navigation in design space' must be done in a very intuitive and fast manner. The vocabulary by which designer intentions are expressed should be very high-level, and is translated into a large number of low-level operations that could be difficult to compute efficiently. Nonetheless, conceptual design is easier than detailed design in that (1) the number of geometric objects involved is usually much smaller, (2) it suffices to support model visualization (other operations are not strictly essential), and (3) interactive visualization is much more important than accuracy of the model.

Most conceptual design is currently done using pen and paper. When designing 3-D models, the great potential advantage of computer graphics is obvious, letting designers inspect their models from different viewpoints and at different scales and make fast modifications. We are interested in direct conceptual design of 3-D models.

Constructive Solid Geometry (CSG) [Requicha80, Hoffmann89] is a well-known representation scheme that represents 3-D solids by a graph whose leaves contain geometric primitives and whose nodes contain Boolean and affine operations. The arcs of the graph denote the fact that an operation uses an object as an argument. The native CSG modeling operations include: (1) instantiation of simple geometric primitives, (2) composition of objects from simpler objects in a hierarchical manner, using the union Boolean operation, (3) performing affine operations (translation, rotation and scaling) on objects, and (4) using one object to modify another by the Boolean difference and intersection operations. These relatively high-level modeling operations make CSG an attractive choice for conceptual design.

A CSG graph models a family of objects spanning a design space. Navigation through this design space is performed by modifying

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

[†]Current address: Programmable Automation Laboratory, Computer Science Department, University of Southern California, Los Angeles, CA 90089-0781. http://www-pal.usc.edu/∼spitz, spitz@usc.edu

the numerical parameters defining the affine transformations. It is crucial that such navigation could be performed in an interactive manner, otherwise the design process is damaged to the point that pen and paper are more effective. Rapid display of CSG objects after modification of their affine parameters is the goal of this paper.

Previous work. Previous work on rendering CSG models can be classified into several categories according to the methods used. The most obvious method is to convert the CSG representation into a boundary representation, which is the native format accepted by virtually all interactive rendering systems. However, all boundary evaluation methods require considerable amounts of computation [Requicha85] and are too slow for interactive modification of the model.

Several well-known graphics algorithms have been customized for CSG display. This includes scanline methods [Atherton83, Bronsvoort87], ray tracing and ray casting [Roth82], image subdivision [Cameron94], octrees [Meagher84] and even point sampling and voxel reconstruction [Breen91]. Again, all of these methods do not provide interactive performance.

Binary Space Partitioning (BSP) trees, known in computer graphics for acceleration of many types of computations, were used for CSG [Thibault87, Naylor92, Naylor96]. Although the results are impressive, these method are basically an acceleration of complete boundary evaluation. They utilize standard graphics hardware only to a limited extent, and limit the range of modifications that can be done on the object.

Many methods were suggested for CSG display using special purpose hardware. These methods have progressed from a special kind of z-buffer [Okino84, VanHook86, Rossignac86] to usage of multiple z and frame buffers and multiple rendering passes [Goldfeather86, Jansen86, Jansen87, Goldfeather89, Rossignac90, Jansen91]. These methods do support real-time rendering of CSG models (naturally, depending upon the model size), but are not practical for every-day use because they require expensive special purpose hardware that is not easily available.

[Wiegand96] presents a method to emulate the algorithm of [Goldfeather89], designed for a parallel pixel-based architecture, on standard graphics hardware. In effect, double z and image buffers are emulated by using multiple passes and memory copying. This method, and a similar method detailed in [McReynolds96], are the most practical methods suggested so far. However, these multi-pass methods are brute-force ones, putting virtually all of the computational load on the graphics system. Demanding applications should try to use all the computational resources provided by the machine, especially considering that today's CPUs are rather powerful and that multi-processor machines are not uncommon. In addition, the brute-force methods do not support many kinds of geometric acceleration schemes.

In summary, none of the existing methods fulfills the goal of interactive display of CSG models while using standard graphics hardware. The methods that partially support that goal (e.g. [Wiegand96]) do not make efficient use of the overall architecture and are too low level and brute-force to provide satisfactory support for conceptual design.

Proposed approach. In this paper we present a method for displaying CSG models, which is particularly attractive for conceptual design. The method enables interactive modification of the numerical parameters of the affine transformations in the CSG graph. The major advantages of the method are:

1. Interactive performance,
2. Utilization of standard, widely available graphics hardware,
3. Splitting the work between the CPU and the graphics system.

In addition, our internal representation speeds up complete approximate boundary evaluation when desired, and possesses a built-in spatial hierarchy.

The method utilizes a combination of graph re-writing, efficient geometric algorithms and standard graphics hardware. The surfaces of solid CSG primitives are approximated by linear facets, and the primitives themselves are expressed as convex polyhedra or Boolean combinations of convex polyhedra. All traditional CSG primitives (box, tetrahedron, pyramid, sphere, cylinder, cone, torus) are easily supported. The torus is modeled as a union of several simple convex pieces. Note that constraints imposed by sub-object convexity are present in other interactive solutions as well [Wiegand96]. Support for free-form objects is more difficult, although there is no inherent reason why they could not be used.

The CSG graph is transformed into a novel *Convex Differences Aggregate (CDA)* representation. The CDA is a union of cells each of which is a containing convex polyhedron minus a set of contained convex polyhedra. Interactions between polyhedral faces that might affect visibility are efficiently detected and handled by a *face binding* mechanism. Display of an aggregate involves standard capabilities of the graphics system (polygon rasterization, z-buffer, a single stencil bit-plane). The process of building a CDA can be divided into two: structure and geometry. Structure is re-computed each time the structure of the corresponding CSG graph is modified. This computation is very fast. Geometry is updated each time visual feedback is needed, in particular when the user modifies the parameters defining the affine operations. This update is done by computing 3-D convex hulls of sets of points.

By itself, the CDA is a *lossy* modeling scheme [Rappoport95] because it approximates geometry using planar faces and because it loses the object hierarchy. However, the method as a whole is lossless, because the original CSG graph is retained.

The paper is structured as follows. The CDA representation is defined in Section 2. Rendering a CDA is detailed in Section 3. Conversion of a CSG object to a CDA is described in Section 4, and general system aspects and results are discussed in Section 5.

2 The Convex Differences Aggregate (CDA)

In this section we introduce the convex differences aggregate (CDA) representation. The CDA utilizes the difference between a containing convex polyhedron and other convex polyhedra. As such, it resembles the 2-D representations in [Sklansky72, Batchelor80, Tor84, Rappoport90, Rappoport92] and the 3-D and n-D representations in [Woo82, Kim90, Rappoport91]. However, the CDA is different from these representations in structure, geometric aspects, and the fact that it is optimized for computer graphics display.

A CDA A represents a pointset $Set(A)$ by a set of *cells* $A = \{C_1, \ldots, C_n\}$. Each cell C represents a pointset $Set(C)$. The pointset represented by the CDA is the union of the pointsets represented by the cells: $Set(A) = Set(C_1) \cup \ldots \cup Set(C_n)$. In principle, the CDA is a pure aggregate of cells and there are no relationships between the cells. Figure 1 shows a CDA consisting of two cells, C_1 and C_2.

A cell C is represented by a single *positive* cell $Pos(C)$, a set of *negative* cells $Neg(C) = \{N_1, \ldots, N_m\}$, and a set of *zero* cells $Zero(C) = \{Z_1, \ldots, Z_k\}$. We denote a cell by $C = \{P; N_1, \ldots, N_m; Z_1, \ldots, Z_k\}$. The pointset represented by a cell is the set difference between that

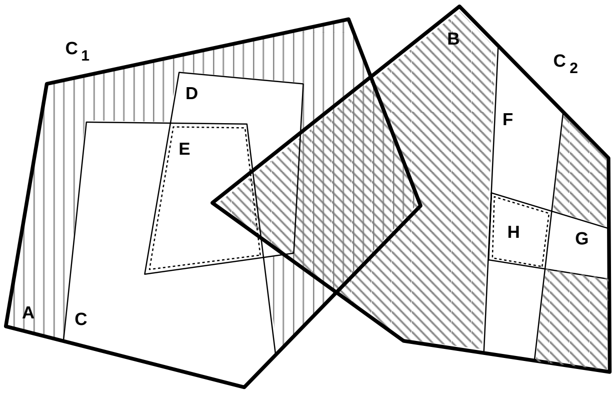

Figure 1 A 2-D CDA consisting of two cells, $C_1 = \{A; C, D; E\}$ and $C_2 = \{B; F, G; H\}$. Each cell has two negative cells intersecting in a single zero cell.

represented by its positive cell and the union of pointsets represented by its negative cells:

$$Set(C) = Set(P) \setminus (Set(N_1) \cup \ldots \cup Set(N_m)).$$

Negative cells represent holes in their positive cells. In Figure 1, C, D, F and G are negative cells. Zero cells do not affect the pointset represented by a cell; they exist solely to assist rendering. Unless stated otherwise, throughout this paper we assume regularized Boolean operations, ensuring the full dimensionality of CSG objects. The result of a regularized Boolean operation between two sets is the topological closure of the result of the Boolean operation when applied to the topological interior of the two sets [Requicha80].

The pointset represented by a positive, negative or zero cell H is given by a convex polyhedron associated with the cell. For simplicity, we will also denote the polyhedron by H. We will also call a face positive (negative, zero) when referring to a face of the polyhedron of a positive (negative, zero) cell. We assume that face normals of positive polyhedra consistently point out of the positive polyhedron, and that normals of negative faces consistently point into the negative polyhedron.

From a set-theoretic modeling point of view, the CDA is a special kind of CSG graph: it contains only the union and difference Boolean operations (no intersections, complements, and affine operations), it possesses a special structure (its depth is exactly three, the first and third levels contain union operations and the second level a difference operation), and all primitives are convex polyhedra.

The CDA imposes additional structural and geometric requirements from its ingredients, which are an integral part of its definition: (1) containment relationships between pointsets of positive and negative cells, (2) existence of zero cells and intersection relationships between their pointsets and that of negative cells, (3) binding between certain faces of negative and positive cells, and (4) binding between certain faces of zero and negative cells. The motivation for these requirements is optimizing the display operation while supporting fast affine modification of geometry.

Containment. Negative cells are required to be contained in their positive cell. Negative cells of dimensionality lower than 3 are redundant and should be discarded (recall that the Boolean operations are regularized).

Zero cells. A zero cell Z_{ij} exists for every intersecting pair of negative cells that belong to the same aggregate cell: $Set(Z_{ij}) = Set(N_i) \cap Set(N_j)$. In Figure 1, each cell has two negative cells intersecting in a single zero cell. A zero cell may exist even when the intersection between the negative cells is of one dimension lower than the cell dimension (but see the definition of zero face bindings below). For example, if two negative 3-D convex polyhedra intersect only in a face, then the resulting zero cell is a degenerate polyhedron with two identical faces having opposite normals.

In Figure 2, negative cells M and N intersect in a zero cell Z which is of full dimensionality in (a) and (b) and of dimension one lower in (c).

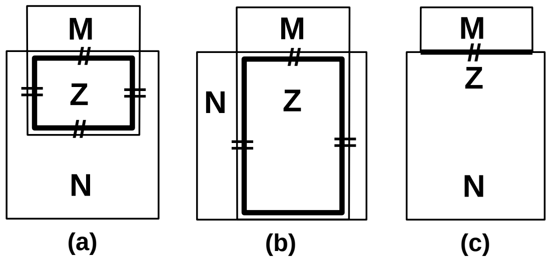

Figure 2 Two negative cells can yield a zero cell of full dimensionality (a, b) or of dimension one lower than themselves (c). Bound zero faces are crossed by two short line segments. Note that the lower face of the zero cell in (b) is not bound.

Positive-negative (P-N) face bindings. If a positive face and a negative face belonging to the same cell are co-planar, there is an explicit binding between these two faces. We say that the positive face is bound *downwards* and that the negative face is bound *upwards.* Note that since both polyhedra are convex, a negative face is bound to no more than a single positive face. However, it is certainly possible that two different negative cells will be co-planar with the same positive face. Thus, a positive face can be bound to several different negative faces.

Negative-zero (N-Z) face bindings. A face of a zero cell can be explicitly bound to a face of one of the two negative cells that yielded the zero cell. This binding exists in one of two situations:

1. The zero face is co-planar with exactly one negative face (e.g. the bound faces in Figure 2(a) and (b)), or
2. The zero face is co-planar with both negative faces, and the normals of the negative faces point in opposite directions (e.g. Figure 2(c)).

We say that the negative face is bound *downwards* and that the zero face is bound *upwards.* The lower zero face in Figure 2(b) is not bound because the normals of the originating negative faces point in the same direction. The bound faces of zero cells are exactly those that do not belong to the union of their two originating negative cells. Since negative polyhedra are convex, their intersection is also convex and a zero face is bound to no more than a single negative face. A negative face can be bound to several different zero faces.

3 Rendering a CDA

In this section we discuss CDA rendering, showing how the structural and geometric properties of a CDA make its rendering very simple. A CDA is rendered using a standard 3-D graphics API such as OpenGL. The graphics system should support polygon-based hidden surface removal (usually, using a z-buffer) and a stencil bitplane.

The rationale behind the convex differences aggregate representation is made clear when considering the rendering operation. We can render aggregate cells independently because their pointsets are unioned; from the point of view of the graphics system, these cells can be treated as separate objects. Positive faces can be rendered correctly because holes in them are modeled explicitly by face bindings. Negative faces can be rendered correctly because interactions between negative cells that might modify visibility relationships are represented by the existence of zero cells, and are handled by bindings between zero and negative faces.

The CDA rendering algorithm is shown in Figure 3. Rendering a CDA starts by clearing the frame buffer and the z-buffer, and then independently rendering all of its cells. Rendering a cell is done by rendering its positive cell and all its negative cells. Rendering a positive or negative cell is done by rendering its polyhedron. Rendering a polyhedron is done by rendering those of its faces that are front-facing (a simple optimization) and not bound upwards. This discards only negative faces bound to positive ones, because zero faces are not rendered directly. Rendering a face starts by clearing the stencil bit-plane. The faces bound to the rendered face (actually, only those that are bound downwards) are rendered onto the stencil; they leave no mark on the frame buffer and z buffers. Then the face itself is rendered on the parts of the frame and z buffers not masked by the stencil.

```
CDARender (CDA $A$):
    clear frame and z buffers.
    for all cells $C_i$ of $A$
        CellRender ($C_i$).
◇

CellRender (Cell $C$):
    PolyhedronRender (Pos ($C$)).
    for all negative cells $N_i$ of $C$
        PolyhedronRender ($N_i$)).
◇

PolyhedronRender (Polyhedron $P$):
    for all faces $F_i$ of $P$
        if $F_i$ is front-facing and not bound upwards
            FaceRender ($F_i$).
◇

FaceRender (Face $F$):
    clear stencil bit-plane $S$.
    for all bound faces $b_j$ of $F$
        render $b_j$ on $S$.
    render $F$ on the parts of the frame and z buffers
    not masked by $S$.
◇
```

Figure 3 Rendering a convex differences aggregate.

A formal proof of correctness of the rendering algorithm is given in [Spitz94]; here we only give an informal argument. First, note that since the pointset represented by the CDA is the union of the pointsets represented by the aggregate cells, the z-buffer automatically guarantees that if cells are rendered correctly then the results are combined correctly.

Next, consider positive faces. Because negative cells are contained in the positive cell, positive faces are never hidden by negative ones. At most, they coincide with them. Since negative cells are of full dimensionality, a negative face that coincides with a positive face represents a through-hole in the positive face. Such negative faces are bound (by definition of the CDA) to the positive face, and masked by the stencil. As a result, the visible pixels of positive faces are exactly those that are rendered.

Finally, consider negative faces. Negative faces that overlap positive faces represent holes in the positive face through which something may be visible. Such faces are bound upwards and are not directly rendered by the algorithm; they were rendered as holes in the stencil when rendering the positive face. Regarding other negative faces, if a negative face does not intersect any other negative face, it might be visible like any ordinary face of a non-convex polyhedron and is rendered similarly to positive faces (note that normals of negative faces by definition point in the correct direction, outside the object represented by the cell). If a face f of a negative cell N is intersected by a negative cell M, there are three cases:

1. The intersection between N and M is not along a co-planar face; the intersection is of full dimensionality. In this case the cell M cuts a through-hole in the face f. By definition, the face f is bound to a zero face having the exact geometry of the through-hole. When the face f is rendered, the stencil masks the through-hole.

2. The intersection between N and M is along a face g of M co-planar with f, and the normals of f and g point in opposite directions. This case is similar to the previous one, since g again comprises a hole in f. This is why we allow degenerate zero cells having lower dimensionality.

3. The intersection between N and M is along a face g of M co-planar with f, and the normals of f and g point in identical directions. In this case f and g can be rendered independently, since they do not hide or create through-holes in each other. This is why in this case there are no zero faces bound to either of them.

The requirement that cell polyhedra must be convex is not strictly necessary for rendering, and is needed only in order to efficiently recompute the geometry of the cells, as explained in Section 4.2. In addition, the fixed three-level depth of the CDA is not essential, but is more efficient for rendering. An arbitrary depth CDA-like representation for n-D polyhedra called the *extended convex differences tree* is described in [Rappoport91].

The deep reason why the rendering algorithm works is, of course, the fact that the CDA is a *solid* representation. Whenever a face has a through-hole, the hole is simply masked and the rest of the face is rendered ordinarily. Because the object is modeled as a solid, if the through-hole is not a real hole through the object, at some point some face will block it. For the same reason, our rendering algorithm will not work when the viewer is located inside the solid.

4 Conversion of CSG to CDA

In this section we show how to convert a CSG graph to a convex differences aggregate representation. Initially, the primitives are tessellated into linear polyhedra according to approximation parameters supplied by the user. The conversion proceeds in two parts: generation of the *structure* of the resulting CDA (Section 4.1), and computation of the *geometry* of the convex polyhedra present in the CDA (Section 4.2). The separation into different structure and geometry computations is only done in order to simplify the presentation. In practice, the two are intermixed to enable pruning optimizations [Spitz94].

4.1 CDA Structure Generation

Generation of the correct structure of a CDA corresponding to a CSG graph is done by a simple graph re-writing procedure. Many of the previous algorithms for rendering CSG objects [Rossignac90, Goldfeather89, Wiegand96] utilize such procedures. Our graph re-writing is different from all of these, although it shares with them the fact that the top level of the resulting expression contains only union operations.

Initially, the CSG graph is converted into an equivalent binary tree by duplicating a node according to the number of its outcoming arcs and by splitting Boolean operations having more than two arguments to a series of binary operations. All affine operations in the resulting CSG tree are now propagated into its leaves and attached to the primitives. All these steps are standard CSG procedures. We now have a tree of Boolean operations in which every primitive is located in its final position.

The CDA is computed recursively (bottom-up computation is also possible). When visiting a node, we call the evaluation algorithm recursively for its children, and merge the returned CDAs to form the CDA returned by the node. Since CSG nodes contain only union, intersection and difference Boolean operations, it is enough to show how the union, intersection and difference of two CDAs can be transformed into a CDA. In the following, denote the two CDAs and their cells by $A = \bigcup_{i=1}^{n} C_i, B = \bigcup_{j=1}^{r} D_j$.

Aggregate union. The union of two aggregates is simply an aggregate containing all of their cells: $A \cup B = \bigcup_{i=1}^{n} C_i \bigcup_{j=1}^{r} D_j$. Obviously, the pointset represented by the new aggregate is indeed the union of the two pointsets represented by A and B. Since all cells are valid before executing the aggregate operation, they are also valid after its execution, and the result is a valid CDA.

Aggregate intersection. Since $\bigcup_{i=1}^{n} C_i \cap \bigcup_{j=1}^{r} D_j = \bigcup_{i=1..n, j=1..r} C_i \cap D_j$, the intersection of two CDAs is a CDA whose cells are the pairwise intersections between the cells of the two aggregates: $A \cap B = \{C_i \cap D_j, i = 1 \ldots n, j = 1 \ldots r\}$. Again, the pointset of the result is the intersection of the original pointsets. However, we also have to show how to implement the cell intersection operation so that the result possesses a valid structure.

Cell intersection. Denote the two cells to be intersected by $C = \{P; N_1, \ldots, N_m; Z_1, \ldots, Z_k\}$, and $D = \{Q; M_1, \ldots, M_s; X_1, \ldots, X_t\}$, and also denote $N = N_1 \cup \ldots \cup N_m$, $M = M_1 \cup \ldots \cup M_s$. Now, $Set(C) = P \setminus N, Set(D) = Q \setminus M$. By De-Morgan's formulae, we have $(P \setminus N) \cap (Q \setminus M) = (P \cap N^c) \cap (Q \cap M^c) = (P \cap Q \cap (N \cup M)^c) = (P \cap Q) \setminus (N \cup M) = (P \cap Q) \setminus (N_1 \cup \ldots \cup N_m \cup M_1 \cup \ldots \cup M_s)$, which is of the structural form desired. The CDA also requires that negative cells are contained within their positive ones, so we intersect each negative cell with $P \cap Q$. P's original cells are already contained in P, so we only have to intersect them with Q; the same holds for Q's original cells. We obtain $C \cap D = \{(P \cap Q); (Q \cap N_1), \ldots, (Q \cap N_m), (P \cap M_1), \ldots, (P \cap M_s)\}$.

Now to the zero cells. Each original zero cell Z_{ij} of C has originated from an intersection between two negative cells N_i, N_j. Hence the new corresponding negative cells $(Q \cap N_i)$ and $(Q \cap N_j)$ may potentially intersect. We can thus intersect Z_{ij} with Q, and add the result (if it is not empty) to the zero cell list. This, however, is not sufficient, because there may be new zero cells arising from intersections of the form $(Q \cap N_i) \cap (P \cap M_j)$. Therefore, we add non-empty intersections of this form to the result as well. To summarize, the intersection between two cells is given by:

$$\{P; N_1 .. N_m; Z_1 .. Z_k\} \cap \{Q; M_1 .. M_s; X_1 .. X_t\} =$$

$$\{(P \cap Q); (Q \cap N_1) .. (Q \cap N_m), (P \cap M_1) .. (P \cap M_s);$$

$$(Q \cap Z_1) .. (Q \cap Z_k), (P \cap X_1) .. (P \cap X_t), (Q \cap N_i) \cap (P \cap M_j) \ldots\}.$$

Note that the only geometric operation in this expression is intersecting two convex polyhedra (all original positive, negative and zero cells are convex polyhedra). This operation will be detailed in Section 4.2.

Note that simple pruning optimizations are possible and should certainly be performed. If the two positive cells P, Q do not intersect, no other intersections calculations need be performed. If a positive cell P does not intersect a negative cell M_j of Q, it does not intersect the zero cells X_l of Q that originated from M_j.

Aggregate difference. $A \setminus B = (\bigcup_{i=1}^{n} C_i) \setminus B = \bigcup_{i=1}^{n} (C_i \setminus B)$. This expression is a union of differences between a cell and an aggregate. For a cell C, we have $C \setminus B = C \setminus \bigcup_{j=1}^{r} D_j = C \cap (\bigcup_{j=1}^{r} D_j)^c = C \cap (\bigcap_{j=1}^{r} D_j^c) = \bigcap_{j=1}^{r} (C \cap D_j^c) = \bigcap_{j=1}^{r} (C \setminus D_j)$. That is, a cell minus an aggregate can be expressed as the intersection of terms of the form $C \setminus D_j$, which is a cell difference operation.

Cell difference. For two cells C, D, $C \setminus D = C \setminus (Q \setminus \bigcup_{i=1}^{s} M_s)$. It is easy to see that, in general, $A \setminus (B \setminus C) = (A \setminus B) \cup (A \cap C)$. Hence, $C \setminus D = (C \setminus Q) \cup (C \cap \bigcup_{i=1}^{s} M_s) = (C \setminus Q) \cup (\bigcup_{i=1}^{s} (C \cap M_j))$. In other words, the difference between cells C and D is an aggregate whose cells are $\{C \setminus Q, C \cap M_1, \ldots, C \cap M_s\}$. The cell $C \setminus Q$ is computed by adding $P \cap Q$ to the negative cell list of C, updating the zero cells as well (Q cannot be added as is because it must be contained within C's positive cell, P). All others are easily handled by transforming the polyhedron M_j to a cell M_j' containing a single positive cell and intersecting the two cells C and M_j' using the cell intersection expression derived above.

In summary, we have seen that aggregate union, intersection and difference can all be reduced to a series of cell intersection operations, which in turn require a single geometric operation, intersection between two convex polyhedra.

In general, the size of the resulting aggregate (in terms of the number of cells that it contains) is usually larger than the combined sizes of the two input aggregates. Theoretically, an aggregate's size can be very large (see [Rossignac94]), but practically, during conceptual design, aggregates are of manageable sizes. The same statement holds for the number of the negative cells, and especially the zero cells, in a cell. Theoretically, all pairs of negative cells may intersect, resulting in a large number of zero cells. However, during the conceptual design stage features tend to be strong and geometric interactions between through-holes (resulting in zero cells) are kept to a minimum.

Example. Figure 4 shows a CSG graph describing a simple car. The car is built as follows. A circle primitive is scaled using an affine operation to provide a wheel frame. A rectangle primitive is instantiated twice, scaled and positioned to form vertical and horizontal wheel holes. The wheel frame minus the holes defines the final wheel. The wheel is instantiated twice to form a left wheel and a right wheel. Note that the geometries of the wheels are different, because a different affine transformation is used. Two rectangles are intersected to form a body frame. A window is subtracted from the body frame to generate the body, which is then positioned in place. The car is the union of the two wheels and the body. The object is of course very simple, but the situation is typical to a conceptual design session: parameterized primitives are instantiated and positioned at the desired locations in the desired scale. They are combined using Boolean operations to form sub-objects. Sub-objects are usually combined using the union operation.

The result after the initial graph re-write is shown in Figure 5(a). The graph was expanded into a tree. All primitives (marked by an internal P) are at their final positions and scale (in the actual data structure affine transformations are attached to the primitives; this is not shown in the figure because we do not consider them to be

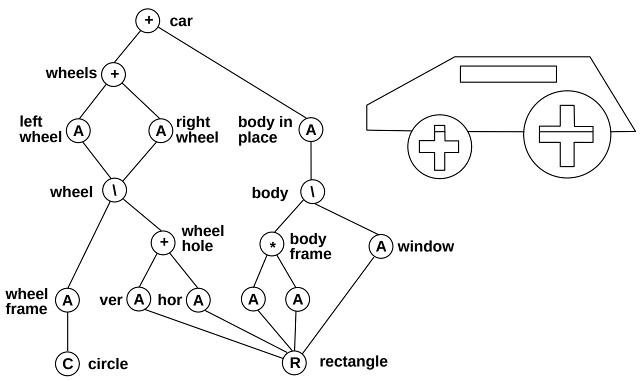

Figure 4 A CSG graph (left) describing a simple car object (right).

part of the tree anymore).

We now describe in a bottom-up fashion all intermediate CDAs computed during the conversion of the tree to a valid CDA. Capital letters denote CDAs, and capital letters with a hat denote CDA cells. The first two CDAs are simply $J = \{\hat{J}_1, \hat{J}_2\}$, $K = \{\hat{K}_1, \hat{K}_2\}$. $\hat{J}_1 = \{B;;\}, \hat{J}_2 = \{C;;\}, \hat{K}_1 = \{E;;\}, \hat{K}_2 = \{F;;\}$. The CDA L results from intersecting two cells: $L = \{\hat{L}_1\}, \hat{L}_1 = \{G \cap H;;\}$. The CDA M results from the difference between two CDAs: $M = A \setminus J = \{\hat{M}_1\}, \hat{M}_1 = \{A; A \cap B, A \cap C; A \cap B \cap C\}$. M possesses a single cell, whose positive cell is A, two negative cells, and a single zero cell. Similarly, the CDA N is given by $N = D \setminus K = \{\hat{N}_1\}, \hat{N}_1 = \{D; D \cap E, D \cap F; D \cap E \cap F\}$. The CDA O also results from the difference between two CDAs: $O = L \setminus I = \{\hat{O}_1\}, \hat{O}_1 = \{G \cap H; G \cap H \cap I;\}$. The CDAs P and Q result from the union of two CDAs: $P = M \cup N = \{\hat{M}_1, \hat{N}_1\}, Q = P \cup O = \{\hat{M}_1, \hat{N}_1, \hat{O}_1\}$. Figure 5(b) shows the final CDA obtained. A single prime on a cell's name (e.g. M') denotes the fact that the cell corresponds to the intermediate CDA M. A double prime (e.g. B'') denotes the fact that the geometry of a negative CDA cell is *not* identical to the corresponding intermediate result having the same name, because it must be intersected by its positive cell.

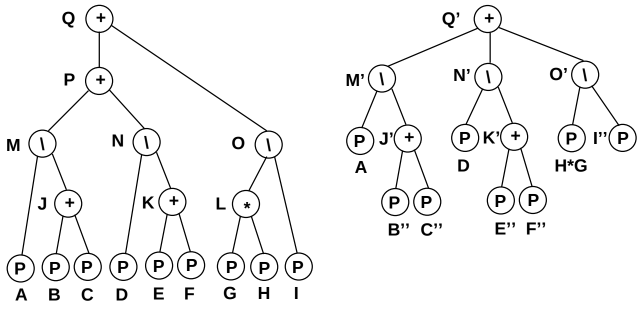

Figure 5 (a) The initial tree corresponding to the car CSG graph, without affine transformations. (b) The final CDA.

4.2 CDA Geometry Computation

The only operation needed in order to compute the geometry of a CDA whose structure was generated from a CSG graph is the intersection between two convex polyhedra. This problem has been extensively dealt with in the computational geometry literature [Muller78, Preparata85, Hertel84, Bieri88, Chazelle87, Chazelle92]. For the sake of completeness, we will briefly outline the algorithm in [Muller78, Preparata85], which is the one we have implemented. Generation of face bindings is a simple book-keeping matter during the execution of the algorithm.

Initially, a single point inside the intersection of the two polyhedra A, B is computed by linear programming. The two polyhedra are now translated so that the common intersection point lies at the origin. Each face is transformed to its dual point: if the plane equation of the face is $ax + by + cz + 1 = 0$, the dual point is (a, b, c). The 3-D convex hull of all dual points (belonging to both A and B) is computed. We have used an excellent available implementation of the QuickHull algorithm [Barber93]. The plane equations of the resulting convex hull are now dualized back to vertices. These are the vertices of the intersection $A \cap B$. The correctness of this algorithm is simple to prove [Preparata85].

The complexity of 3-D convex hull is $O(n \log\ n)$, so this step dominates the theoretical complexity of the algorithm. However, note that in our application the algorithm is repeatedly invoked on polyhedra whose relative geometries are almost unchanged. Therefore, there is reason to hope that incremental computation will reduce the complexity to linear time in practice.

We should note that there are efficient algorithms to *detect* a possible intersection between two convex polyhedra [Dobkin83]. In general, the fact that the CDA utilizes convex objects enables usage of the considerable amount of efficient algorithms for convex objects developed by the computational geometry community.

Note that a full boundary representation of the object represented by a single CDA cell can be computed by performing 2-D difference operations between bound faces. Thus, the CDA saves some computations when a complete boundary evaluation (with approximate primitives) needs to be performed. This is useful, for example, when converting the designed object to the VRML format.

5 Implementation and Results

In this section we give a brief description of our implementation. It is not our intention to describe a complete system; a complete discussion of system aspects would have to include important issues such as general user interface, direct 3-D manipulation of geometric primitives, visualization of the CSG graph structure, treatment at the object level, etc. All these are beyond the scope of the present paper. Our purpose in this section is only to let the reader understand the system context of our algorithms and data structures. A CSG system emphasizing user interface issues and direct 3-D manipulation is described in [Emmerik93].

The system was implemented on SGI workstations running Irix 5.3. Figure 6 shows the general system architecture. The user can perform four types of operations: (1) modification of the structure of the CSG graph, by adding or deleting nodes and arcs, (2) selection of sub-objects to manipulate, (3) modification of the affine operation associated with the selected sub-object, and (4) manipulation of the graphics view. All operations are done through a user interface module. Obviously, this general architecture suits many of the previous CSG display algorithms as well.

Operation 1 necessitates a re-computation of a CDA (both structure and geometry) from the modified CSG. This is the only operation requiring computation of CDA structure. The result of Operation 2, the current selected object, is stored in a separate data structure. In practice, this is a collection of pointers to CDA and CSG nodes. Operation 3 is the main one, triggering a re-computation of the geometry of the selected CDA nodes. Affine parameters are modified using a 'direct manipulation device' (DMD) [Emmerik93] for editing 3-D local coordinate systems (Figure 9, (a) and (e)). Operation 4 is done directly at the level of the graphics API and does not affect the CSG or CDA data structures.

Figures 7–9 show some objects designed using the system. We

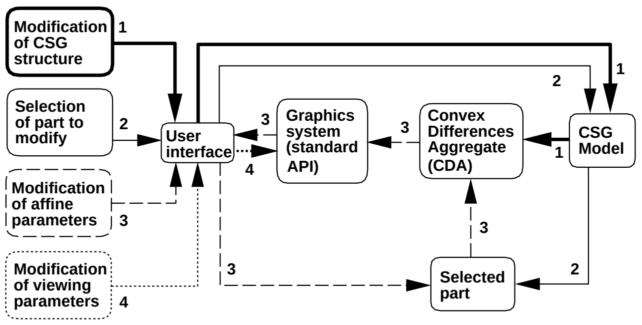
Modification of CSG structure
1
Selection of part to modify
2
Modification of affine parameters
3
Modification of viewing parameters
4
User interface
3
4
Graphics system (standard API)
3
Convex Differences Aggregate (CDA)
1
CSG Model
2
1
3
Selected part
2
3

Figure 6 General system architecture.

found it useful to consistently use two different colors during editing to emphasize that the arguments of the difference operation do not have a symmetric role. In Figure 7 there are four cells, each containing a single difference operation. This is a common case, which is dealt with very efficiently by the CDA. In Figure 8, we see how the system can be used in order to investigate interesting geometric relationships between objects. In (c), there is a single CDA cell. Its positive cell is formed by the intersection of the two 'outer' cones, and it has two negative cells originating from the two subtracted cones. In (d), there are two cells, having one and two negative cells respectively. In Figure 9(a–g) we tried to convey the feeling of interactively moving a hole ((h) shows an intersection). The complexity of the CDA structure depends on whether the cube pattern is modeled using a union of nine small cubes or a single flat box minus four slabs. These examples show the advantage and pleasure derived from interactive 3-D Boolean operations using basic primitives.

Note that the relatively coarse tessellation of the cylinder in Figures 7 and 9 is not harmful for conceptual design, where dominant geometric features, their composition and interrelationships are the important issues.

The computational bottleneck in the system is the geometry update following modification of affine transformations of selected nodes. The current system implements only the raw conversion algorithm, without adding additional efficiency schemes. For example, all geometric computations on the geometric elements are repeated on every user event, even if the elements have not been modified, and no spatial acceleration scheme is used to prune possible intersections between negative cells (a quadratic number of such intersections is possible). To give some idea regarding the raw performance of the method, consider that the examples in Figures 7–9 run in about 5–30 frames per second on a 100 MHZ R4000 SGI Indigo with a GR2-XZ graphics board.

6 Discussion

Computer graphics and geometric modeling have not yet been able to provide adequate support for conceptual design. The CSG modeling operations, in particular the Boolean operations, are highly attractive for design. In this paper we presented a method that constitutes a step towards interactive conceptual design of 3-D solids, by allowing interactive modifications of affine transformations in CSG graphs.

Interactive editing and visualization of general CSG models is a truly difficult problem; in this paper we focused on the sub-problem of conceptual design of relatively simple 3-D solids, showing that interactive Boolean operations can be achieved for that purpose. Obviously, beyond a certain level of object complexity the method will no longer provide interactive performance. Because the present implementation incorporates virtually no efficiency schemes beyond that of the raw algorithm, it is difficult to assess at present the maximal object complexity supported. Two measures of object complexity are the number of products in the disjunctive form of the CSG tree and the number of negations in a product. The present system is useful mostly for cases when the maximal number of negations is very small (say, less than 10). Performance grows roughly linearly in the number of products.

In the future, we plan to extend our system in the following directions. First, we will try to improve efficiency by using incremental geometric algorithms, spatial acceleration schemes and integration with multi-pass methods. We believe that the combination of these techniques will provide better scaling to larger models and will take advantage of temporal coherence. Note that a spatial acceleration scheme can be naturally provided by the CDA itself, since each cell in the CDA possesses a two-level hierarchy: convex set minus contained convex sets. If two positive cells do not intersect, their negative children do not intersect as well. Thus, the CDA is suitable for efficient collision detection schemes such as the one described in [Ponamgi95].

Second, we will enhance the system with higher-level functionality, most notably by adding constraints to the set of modeling operations available to the designer. Constraints are also important for object positioning, since the visual feedback provided by tessellated models may not suffice for tangency, incidence and right angle relationships. Constraints may be defined between node coordinate systems or between entities of the object's boundary (such as vertices and edges). Supporting the latter while enabling the user to modify the degree of tessellation of non-planar surfaces may require efficient handling of the persistent naming problem [Rappoport96, Rappoport97], a challenging problem in geometric and solid modeling.

Finally, one of the main ideas in the CDA approach is that faces in a product are not represented explicitly, as done by most CAD systems, but are represented as the difference between an enclosing convex polygon and a set of possibly overlapping convex polygons. We intend to study other possible applications of this idea in geometric modeling.

Acknowledgements. I thank Eyal Ofek for his help with the images and the video, Michal Etzion and Ofir Amir for improving the implementation, Seth Teller for his polyhedra code, and Dani Lischinski, Ofri Sadowsky, Nadav Aharoni and the Siggraph reviewers for their comments on the paper.

References

[Atherton83] Atherton, P.R., A scan-line hidden surface removal procedure for constructive solid geometry. *Computer Graphics*, 17(3):73–82, 1983 (Siggraph '83).

[Barber93] Barber, C.B., Dobkin, D.P., Huhdanpaa, H.T., The Quickhull algorithm for convex hull, GCG53, The Geometry Center, Minneapolis, 1993 (ftp.geom.umn.edu/pub/software/qhull.tar.Z).

[Batchelor80] Batchelor, B.G., Hierarchical shape description based upon convex hulls of concavities. *J. of Cybernetics*, 10:205–210, 1980.

[Bieri88] Bieri, H., Nef, W., Elementary set operations with *d*-dimensional polyhedra. Computational Geometry and its Applications, LNCS 333, Springer-Verlag, 1988, pp. 97–112.

[Breen91] Breen, D.E., Constructive cubes: CSG evaluation for display using discrete 3-D scalar data sets. *Eurographics '91*, 127–142, 1991.

[Bronsvoort87] Bronsvoort, W.F., An algorithm for visible-line and visible-surface display of CSG models. *The Visual Computer*, 3:176–185, 1987.

[Cameron94] Cameron, S.A., Direct drawing from CSG models with hidden-line removal. *CSG 94, Set-Theoretic Solid Modeling: Techniques and Applications,* Information Geometers, 1994, pp. 179–192.

[Chazelle87] Chazelle, B., Dobkin, D., Intersection of convex objects in two and three dimensions, *Journal of the ACM*, 34(1):1–27, 1987.

[Chazelle92] Chazelle, B., An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21(4):671–696, 1992.

[Dobkin83] Dobkin, D.P., Kirkpatrick, D.G., Fast detection of polyhedral intersection. *Theoret. Comput. Sci.,* 27:241–253, 1983.

[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.

[Goldfeather86] Goldfeather, J., Hultquist, J.P.M., Fuchs, H., Fast constructive solid geometry display in the pixel-power graphics system. *Computer Graphics*, 20(4):107–116, 1986 (Siggraph '86).

[Goldfeather89] Goldfeather, J., Molnar, S., Turk, G., Fuchs, H., Near real-time CSG rendering using tree normalization and geometric pruning. *IEEE CG&A*, 9:20–28, May 1989.

[Hertel84] Hertel, S., Mäntylä, M., Mehlhorn, K., Nievergelt, J., Space sweep solves intersection of convex polyhedra. *Acta Inform.*, 21:501–519, 1984.

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

[Kim90] Kim, Y.S., Convex decomposition and solid geometric modeling. Ph.D. Thesis, Mech. Eng., Stanford University, 1990.

[Jansen86] Jansen, F.W., A pixel-parallel hidden surface algorithm for constructive solid geometry. *Eurographics '86*, Elsevier Science, NY, 29–40, 1986.

[Jansen87] Jansen, F.W., CSG hidden surface algorithms for VLSI hardware systems. In: *Advances in Computer Graphics Hardware I*, Springer-Verlag, 1987, pp. 75–82.

[Jansen91] Jansen, F.W., Depth-order point classification techniques for CSG display algorithms, *ACM Trans. on Graphics*, 10(1):40–70, 1991.

[Meagher84] Meagher, D.J., Interactive solids processing for medical analysis and planning. *Computer Graphics '84, NCGA Conference Proceedings*, vol. 2, NCGA, 1984, pp. 96–106.

[McReynolds96] McReynolds, T., Blythe, D., Programming with OpenGL: Advanced Rendering, course #23, Siggraph '96, pp. 36–40.

[Muller78] Muller, D.E., Preparata, F.P., Finding the intersection of two convex polyhedra. *Theoret. Comput. Sci.*, 7:217–236, 1978.

[Naylor92] Naylor, B., Interactive solid modeling using partitioning trees. *Graphics Interface '92*, pp. 11–18.

[Naylor96] Naylor, B., Destructive solid geometry for interactive entertainment and training, *CSG 96*, Cambridge, UK, 1996.

[Okino84] Okino, N., Kakazu, Y., Moritomo, M., Extended depth-buffer algorithms for hidden-surface visualization. *IEEE CG&A*, 4:79–88, 1984.

[Ponamgi95] Ponamgi, M., Manocha, D., Lin, M.C., Incremental algorithms for collision detection between solid models. Proceedings, *3rd ACM/Siggraph Symposium on Solid Modeling and Applications (Solid Modeling '95)*, ACM Press, 1995, pp. 293–304.

[Preparata85] Preparata, F., Shamos, M. I., Computational Geometry: An Introduction, Springer-Verlag, New-York, 1985.

[Rappoport90] Rappoport, A., Using convex differences in hierarchical representations of polygonal maps. *Graphics Interface '90*, pp. 183–189.

[Rappoport91] Rappoport, A., The extended convex differences tree (ECDT) representation for n-dimensional polyhedra. *Intl. J. Comput. Geom. and App.,* 1(3):227–241, 1991. Also: proceedings, *ACM/Siggraph Symposium on Solid Modeling Foundations and CAD/CAM Applications (Solid Modeling '91)*, ACM Press, 1991, pp. 139–148.

[Rappoport92] Rappoport, A., An efficient adaptive algorithm for constructing the convex differences tree of a simple polygon. *Computer Graphics Forum*, 11(4):235–240, 1992.

[Rappoport95] Rappoport, A., Geometric modeling: a new fundamental framework and its practical implications. Proceedings, *3rd ACM/Siggraph Symposium on Solid Modeling and Applications (Solid Modeling '95)*, ACM Press, 1995, 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.

[Rappoport97] Rappoport, A., The Generic Geometric Complex: a modeling scheme for families of decomposed pointsets. Proceedings, *4th ACM/Siggraph Symposium on Solid Modeling and Applications (Solid Modeling '97)*, ACM Press, 1997, pp. 31–41.

[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.

[Rossignac86] Rossignac, J.R., Requicha, A.A.G., Depth-buffering display techniques for constructive solid geometry. *IEEE CG&A*, 6:29–39, Sep. 1986.

[Rossignac90] Rossignac, J.R., Wu, J., Correct shading of regularized CSG solids using a depth-interval buffer. In: Grimsdale, R.L., Kaufman, A., (eds), *Advances in Computer Graphics Hardware V,* Springer-Verlag, Berlin, 1990, pp. 117–138.

[Rossignac94] Rossignac, J.R., Processing disjunctive forms directly from CSG graphs. *CSG 94, Set-Theoretic Solid Modeling: Techniques and Applications,* Information Geometers, 1994, pp. 55–70.

[Roth82] Roth, S.D., Ray casting for modeling solids. *CVGIP*, 18(2):109–144, 1982.

[Sklansky72] Sklansky, J., Measuring concavity on a rectangular mosaic. *IEEE Tran. Com.*, 21:1355–1364, 1972.

[Smithers89] Smithers, T., AI-based design versus geometry-based design, or why design cannot be supported by geometry alone. *Computer-Aided Design*, 21(3):141–150, 1989.

[Spitz94] Spitz, S., Interactive Boolean operations and collision detection on polyhedral solids. Amirim project report, Institute of Computer Science, The Hebrew University, July 1994.

[Thibault87] Thibault, W.C., Naylor, B.F., Set operations on polyhedra using binary space partitioning trees. *Computer Graphics*, 21(4):153–162, 1987 (Siggraph '87).

[Tor84] Tor, S.B., Middleditch A.E., Convex decomposition of simple polygons. *ACM Trans. on Graphics*, 3(4):244–265, 1984.

[VanHook86] Van Hook, T., Real-time shaded NC milling display. *Computer Graphics*, 20(4):15–20, 1986 (Siggraph '86).

[Wiegand96] Wiegand, T.F., Interactive rendering of CSG models. *Computer Graphics Forum*, 15(4):249–261, 1996.

[Woo82] Woo, T., Feature extraction by volume decomposition. Technical Report 82-4, Dept. of Indus. and Oper. Eng., The University of Michigan, 1982.

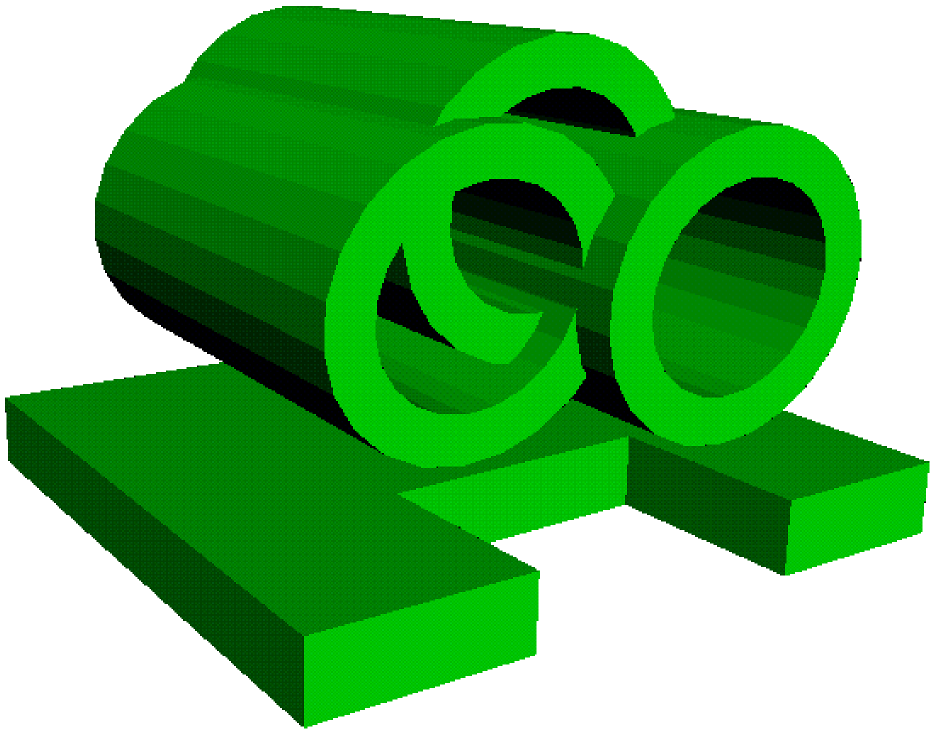

Figure 7 An abstract statue. The object is modeled as a union of the base and the three tubes. Each tube is the difference between two cylinders, and the base is the difference between two blocks. All the `holes' can be moved, scaled and rotated in real time.

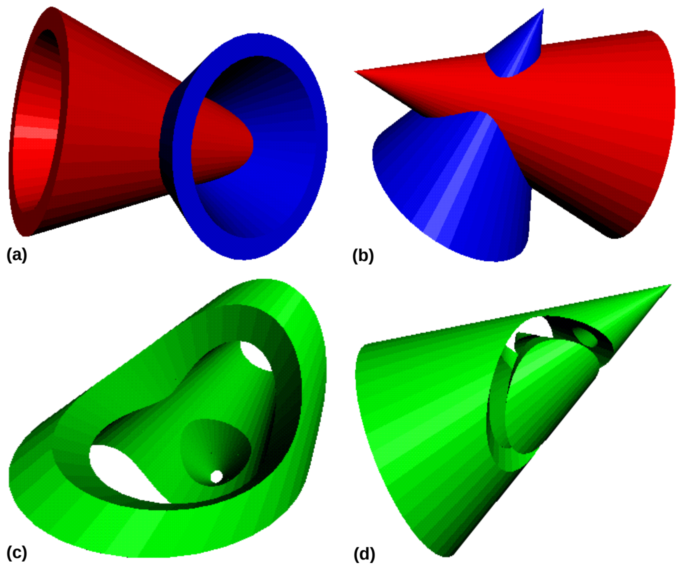

Figure 8 Visualization of the geometric interrelationships between two ice cream cones, each defined as the difference between two cones. (a) union, bottom view; (b) union, side view; (c) intersection; (d) difference. Affine operations can be executed on all constituents of this model in real time.

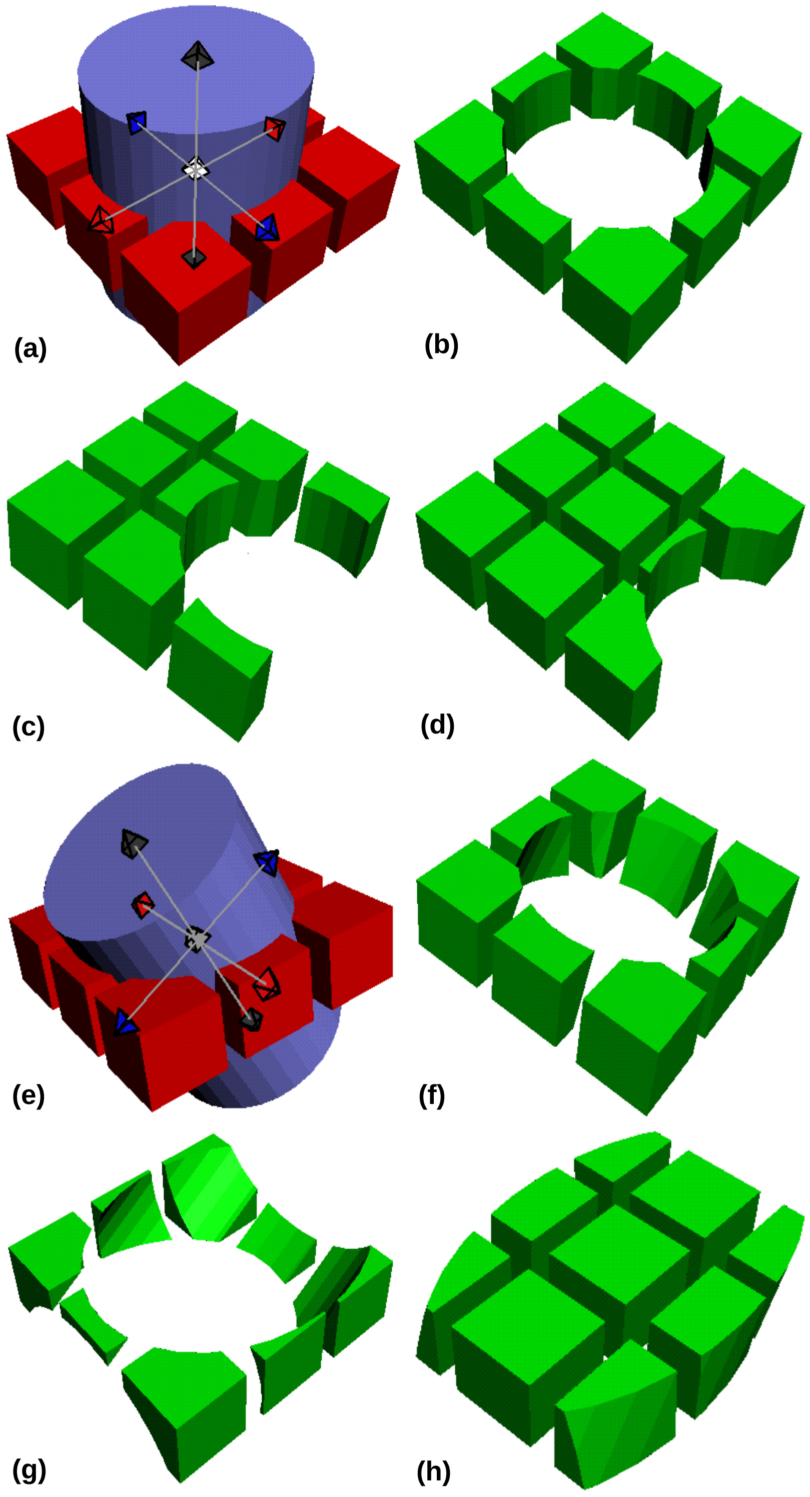

Figure 9 Interaction between a cylinder and a pattern of small cubes. (a) manipulation view; (b-d) moving a cylindrical hole; (e) manipulation view when scaling and rotating the cylinder; (f-g) rotating the hole; (h) intersection with a scaled-up cylinder.