# Approximate Boolean Operations on Free-form Solids

Henning Biermann Daniel Kristjansson Denis Zorin
NYU Media Research Laboratory*

## Abstract

In this paper we describe a method for computing approximate results of boolean operations (union, intersection, difference) applied to free-form solids bounded by multiresolution subdivision surfaces.

We present algorithms for generating a control mesh for a multiresolution surface approximating the result, optimizing the parameterization of the new surface with respect to the original surfaces, and fitting the new surface to the geometry of the original surfaces. Our algorithms aim to minimize the size and optimize the quality of the new control mesh. The original control meshes are modified only in a neighborhood of the intersection.

While the main goal is to obtain approximate results, high-accuracy approximations are also possible at additional computational expense, if the topology of the intersection curve is resolved correctly.

**CR Categories and Subject Descriptors:** G.1.2 [**Approximation**]: Approximation of surfaces and contours. I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling: Boundary representations, Constructive solid geometry (CSG), Curve, surface, solid and object representations.

**Additional Keywords:** Subdivision surfaces, Multiresolution surfaces, Geometric modeling.

## 1 Introduction

Boolean operations are a natural way of constructing complex solid objects out of simpler primitives. This approach is very common in computer-aided geometric design, as many artificial objects can be constructed out of simple parts, such as cylinders, rectangular blocks, and spheres.

Few computational representations of solids are closed with respect to boolean operations. This means that the result of a boolean operation cannot be represented precisely in most cases. One way to avoid this problem is to use a tree of boolean operations as the object representation and implement various algorithms directly on such representation. This approach is referred to as constructive solid geometry (CSG) [14]. However, for many applications CSG is not the most efficient or appropriate. Most commonly, boundary representations (B-Reps) of solids are used and boolean operations have to be implemented in B-Rep framework. Such an implementation is quite difficult for higher-order B-Reps as it requires intersecting parametric surfaces, separating them into pieces and constructing new surfaces out of these pieces.

Existing systems typically treat a B-Rep as a collection of trimmed spline patches, sharing the boundaries. The boundaries of individual patches are often matched only approximately, as it is difficult to ensure that two trimming curves in different parametric domains are identical in space. Each intersection operation leads to increasingly complex and difficult to handle trimming curves. It is difficult to apply smooth deformations to the resulting models, since special care must be taken to avoid cracks, etc. An elementary operation required for this surface representation is to intersect two trimmed NURBS patches, which is a difficult problem by itself. As a result, boolean operations are often slow and not fully robust, although excellent results are achieved by some solid modeling cores.

Figure 1: "Venus with drawers" (after S. Dalí) is created using union, intersection and difference operations.

For many computer graphics and animation applications such high-precision and complex techniques are not essential. The most difficult cases such as the case of two identical, but slightly rotated intersecting objects are often of little relevance. At the same time, keeping the calculations efficient and robust is important, as well as ensuring the complexity of the model is manageable.

In this paper, we present a new approach to computing the result of boolean operations on B-Rep solids. We represent the boundary surfaces as piecewise-smooth subdivision surfaces, described in greater detail in Section 2. For brevity, we are going to call such solids *free-form* solids. The advantage of this representation is its simplicity: the surface is defined by a control mesh with tagged creases and corners, as well as sets of details added at finer levels. Continuity and smoothness of the surface are guaranteed automatically. Representations of this type are increasingly popular, as they considerably simplify modeling complex free-form objects.

*{biermann,danielk,dzorin}@mrl.nyu.edu



ACM SIGGRAPH 2001, 12-17 August 2001, Los Angeles, CA, USA

While the problem we are solving is similar to the traditional CAGD problem, our work is primarily motivated by requirements of applications in computer animation and conceptual design. We aim at fast and robust *approximate* calculations; it should be possible to mix boolean operations with free-form deformations and other types of surface manipulation.

These goals radically change the set of problems that we need to solve: surface-surface intersection, usually regarded as the central part of an implementation of boolean operations, becomes secondary. In particular, we relax the requirement that the topology of the intersection curve is computed precisely. Our primary emphasis is on algorithms for construction of the approximating multiresolution surface for the result. Rather than adjusting our representation to the needs of boolean operations, e.g. by introducing trimming curves, we develop algorithms that allow us to keep the representation of the results simple.

Our main contributions include:

• an algorithm for constructing a coarse control mesh for the result of a boolean operation;

• algorithms for defining and optimizing a parameterization of one multiresolution subdivision surface (result) over another (one of the original surfaces).

• a hierarchical fitting procedure for a surface parameterized over another surface.

### 1.1 Previous Work

Our work is most closely related to, and was done in parallel with, the work of Litke et al. [23] on trimming subdivision surfaces. A few similar issues have to be addressed in both cases. In particular, the connectivity of the control mesh for the trimmed subdivision surface has to be changed, and the new surface needs to be parameterized over the original. However, for trimming there is no need to merge the control meshes of two separate surfaces, and in our case there is no need to use a special combined subdivision scheme [20] for representing the intersection curve. [23] does not optimize parameterizations and surface fitting issues are resolved differently.

Linsen [22] has developed a technique for blending of subdivision surfaces. While [22] presents a construction of a combined control mesh for a blend of two subdivision surfaces, the issues of matching the geometry of the intersection curve and approximation of the result of boolean operations are not considered.

Multiresolution subdivision surfaces were introduced in [24, 26, 35]; we use piecewise smooth subdivision rules of [4] to be able to represent sharp features on multiresolution surfaces. Fitting of subdivision surfaces is discussed in [13, 15].

Our work can be contrasted with the work of Rappoport et al. [27] and earlier work of Goldfeather et al. [11] and Rossignac [8] on efficient rendering CSG objects. While boolean operations on CSG objects are straightforward, substantial effort is required to render them and interactive rendering is possible only for simple objects. On the other hand, it is much more difficult to implement boolean operations on multiresolution surfaces, but interactive rendering is straightforward for surfaces of substantial complexity.

Extensive literature exists on solid modeling with B-Reps (surveys can be found in [1, 28]). The emphasis there is on accuracy and correct and consistent handling of degenerate cases, issues that we avoid by replacing the requirement of topological correctness of the result with the weaker requirement of topological consistency.

To compute approximate intersection curves we use perturbation techniques of [30].

The part of our work on parameterization optimization builds on the techniques used in mesh optimization community [9, 10]. Different methods to solve similar problems were proposed in [25] and [19].

### 1.2 Overview of the algorithm

We apply a boolean operation (intersection, difference or union) to two free-form solids bounded by parametric surfaces (Figure 2). The details of our surface representation are discussed in Section 2. We assume that each bounding surface is an orientable closed surface embedded in $\mathbf{R}^3$. A surface $M$ of this type separates the space into a bounded and an unbounded connected volume. The free-form solid defined by $M$ is the bounded volume.

The main steps of our procedure are illustrated in Figure 3.

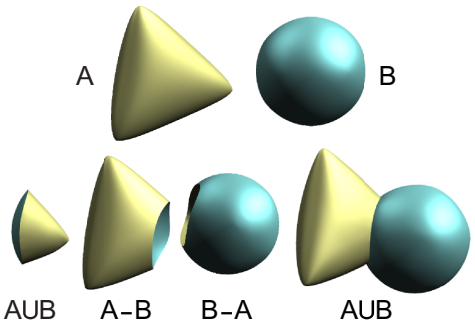



Figure 2: Elementary boolean operations on simple subdivision surfaces.

*Step 1.* Compute an approximate intersection curve, finding its images in each of the two parametric domains of the original surfaces.

*Step 2.* Construct the connectivity of the control mesh for the result, and an initial parameterization of parts of the resulting surface over the domains of the original surfaces.

*Step 3.* Optimize the parameterization of the result over the original domains.

*Step 4.* Determine geometric positions for the control points of the result using hierarchical fitting.

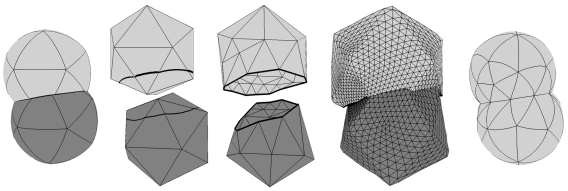


Figure 3: Left to right: Two separate surfaces intersected. Images of the intersection curve in the parametric domains. Parametric domains after cutting. Parametric domain for the result after optimization. Final result after fitting.

**Notation.** All quantities with index 1 refer to the first solid, and all quantities with index 2 refer to the second solid. The parametric domains of the surfaces of the solids are denoted $M_1$ and $M_2$, the maps defining the surfaces are $f_i : M_i \to \mathbf{R}^3$ (Figure 4). Greek letters are reserved for arbitrary points in the parametric domains; e.g. $\alpha = (u, v, w, i)$ where $i$ is the triangle index and $(u, v, w)$ are the barycentric coordinates, $u + v + w = 1$.

| | |
|---|---|
| $M_i$ | domains for original surfaces |
| $f_i : M_i \to \mathbf{R}^3$ | evaluation maps for the original surfaces |
| $c(t) : I \to \mathbf{R}^3$ | spatial intersection curve |
| $c_i(t) : I \to M_i$ | images of the intersection curve in the domains of original surfaces |
| $M_i'$ | parts of the domains $M_i$ to be combined into a new domain |
| $p_i' : M_i' \to M_i$ | parameterization of $M_i'$ over the domain of the original surface $M_i$ |
| $c_i'(t) : I \to M_i'$ | images of the intersection curve in domains $M_i'$ |
| $\widetilde{M}$ | domain for the result formed by merging $M_i'$ |
| $\widetilde{M}_i$ | subdomains of $\widetilde{M}$ parametrized over $M_i$ |
| $\widetilde{p}_i : \widetilde{M}_i \to M_i$ | parameterizations of $\widetilde{M}_i$ |
| $\widetilde{c}(t) : I \to \widetilde{M}$ | image of the intersection curve in $\widetilde{M}$ |
| $p_v^j$ | control point at a vertex $v$ of a parameter domain at refinement level $j$ |

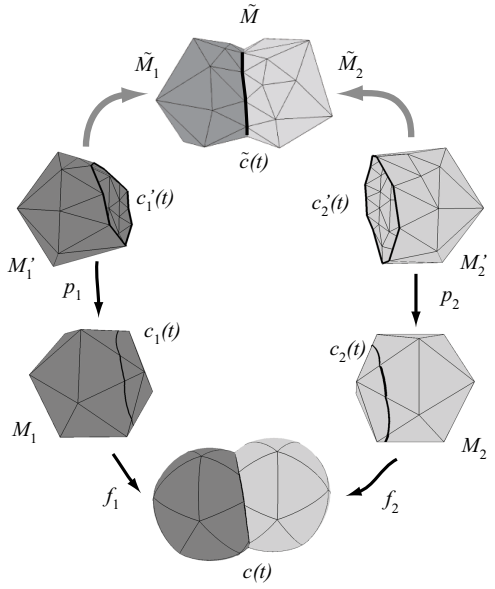



Figure 4: Maps used in the algorithm.

## 2 Multiresolution Subdivision Surfaces

Before describing the algorithm in greater detail, we briefly review subdivision surfaces with a special focus on parameterization. Subdivision surfaces are defined by an initial control mesh. We use a variant of Loop's subdivision scheme for triangular meshes with rules for corners and creases [4].

Multiresolution surfaces extend subdivision surfaces by introducing *details* at each level. Each time a finer mesh is computed, it is obtained by adding detail offsets to the subdivided coarser mesh. The process of reconstructing a surface from the coarse mesh and details is called *synthesis*. Formally, let $S$ be the subdivision operator (a matrix mapping control points on a coarser level to a finer level) let $p^l$ be the vector of control points on level $l$. Given the detail coefficients $d^{l+1}$ for the next subdivision level, the rule for computing the control points on the finer level is $p^{l+1} = Sp^l + d^{l+1}$.

The inverse process of converting the data specified on a fine resolution level to the sequence of detail sets and the coarsest level mesh is called *analysis*. For analysis, we need a way of obtaining the coarse mesh from the fine mesh. This can be done in a number

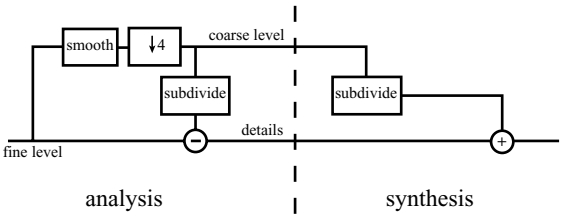



Figure 5: Synthesis and analysis diagrams for multiresolution surfaces.

of ways: simple Laplacian smoothing or Taubin's smoothing[34], quasi-interpolation [23] or least-squares fitting. The synthesis and analysis diagrams are shown in Figure 5. Figure 6 shows smooth surfaces corresponding to different levels of resolution.

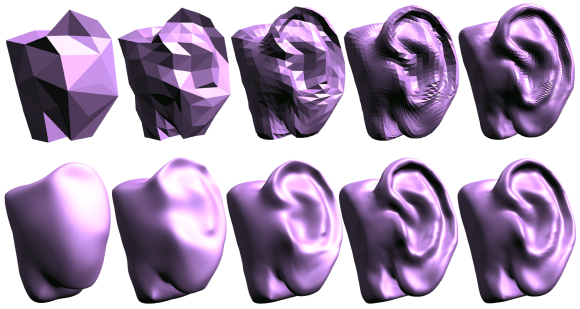


Figure 6: Multiresolution surface. Upper row: coarse-to-fine hierarchy of control meshes. Lower row: corresponding surfaces.

For the purposes of this work, a specific choice of analysis method is irrelevant; it is important to note that given a multiresolution mesh represented as the coarsest mesh and details on finer levels one can reconstruct the surface uniquely, without knowing what analysis method was used. In the areas on the surface resulting from a boolean operation where the details need to be recomputed we use fitting and quasi-interpolation to obtain the details, as described in Section 5.

**Parameterization over the initial control mesh.** Suppose the initial control mesh is a simple polyhedron, i.e., it does not have self-intersections. (We do not need this assumption, but it simplifies the presentation.) Suppose each time we apply the subdivision rules to compute the finer control mesh, we also apply midpoint subdivision to a copy of the initial control polyhedron.

Note that each control point that we insert in the mesh using subdivision corresponds to a point in the midpoint-subdivided polyhedron. Another important fact is that midpoint subdivision does not alter the control polyhedron regarded as a set of points; and no new vertices inserted by midpoint subdivision can possibly coincide.

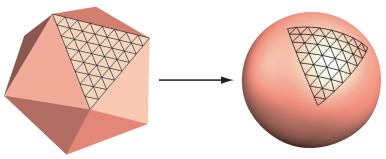


Figure 7: Dyadic parameterization: the surface is parameterized over the coarsest level control mesh.

As we repeatedly subdivide, we get a mapping from a denser and denser subset of the control polygon to the control points of a finer and finer control mesh. In the limit, we get a map from the control polygon to the surface (Figure 7). This parameterization

permits direct evaluation at arbitrary parameter position following the approach of [32], which trivially generalizes to multiresolution setting.

We use the term *parametric domain of a surface* for the top-level control mesh when we discuss parameterizations. The triangles of the top-level control mesh are referred to as *parametric triangles*. *Parametric position* of a point is defined by an index of the triangle in which it is located, together with barycentric coordinates in the triangle.

We reserve the term *vertices* for the vertices of triangles in the parametric domain; the vertices of the three-dimensional control meshes are referred to as *control points*. Multiple control points (one for each level $l \geq i$) correspond to the vertices on levels finer than $i$.

**Linear vertex charts.** Quite often, we need to move points continuously in the parametric domain of a subdivision surface. In this case, it is convenient to use local charts, i.e. maps of parts of the parametric domain (the control mesh) to the plane. We use one of the simplest types of charts, piecewise linear charts. A piecewise linear chart maps one ring of triangles $N_1(v)$ around a vertex $v$ of valence $k$ of the control mesh to a regular $k$-gon $\Pi_k$ in the plane. Let $g : N_1(v) \rightarrow \Pi_k$ be the map from the neighborhood of a vertex to the chart. We can move a point $p$ in one of the triangles in $N_1(v)$ anywhere within the neighborhood: we map it to the plane using the map $g$, apply a transformation $T$ in the plane, and map it back to get the new position $p'$: $p' = g^{-1} \circ T \circ g(p)$. The point $p'$ can end up in any triangle of $N_1(v)$. We use this simple procedure to move points in the parametric domains in two cases: intersection curve snapping (Section 4) and parameterization optimization (Section 5).

# 3 Intersection Curve

For simplicity, we assume in the exposition that the objects intersect along a single curve; in the case of multiple intersection curves, all considerations apply to each curve individually.

The goal of the first step of our algorithm is to find a piecewise-linear approximation to the intersection curve $c : I \rightarrow \mathbf{R}^3$, where $I$ is an interval, along with its images in the domains $M_1$ and $M_2$, $c_i : I \rightarrow M_i$, $i = 1, 2$.

The problem of intersecting two surfaces has received a lot of attention (e.g. [3], [16], [29], [18]). The main difficulty is that the topology of the intersection is unknown in general and may be unstable with respect to small perturbations of the surfaces.

However, we observe that the cases where the problem is ill-conditioned are typically of least interest to us. It is relatively unlikely that it is necessary to find intersections of two slightly touching objects precisely. Thus, we can weaken the requirements of our algorithm and only require that it produces intersections with *valid* topology rather than correct topology. In other words, we require that there are small perturbations of the original surfaces such that the intersection curve has that topology. This allows us to replace the problem of intersecting smooth surfaces with the problem of intersecting approximating meshes. Intersecting polyhedra is a substantially simpler problem, although some effort has to be invested to obtain a fast and robust algorithm.

It should be noted that one can theoretically obtain true intersection topology in all cases excluding degenerate (e.g. single point of contact) by using adaptive refinement following an approach similar to [17]. However, as the topology of the intersection curve can be highly complex even for simple surfaces, many levels of refinement can be required in certain areas, which is contrary to our goals of efficiency and robustness.

The naive mesh intersection algorithm (intersect each pair of triangles from two meshes; construct intersection curves as connected sequence of triangle pair intersections) is inefficient and is not robust. We address these problems with bounding box hierarchies and control point perturbation.

**Bounding box hierarchies.** To accelerate the algorithm we use axis-aligned bounding box hierarchies for each mesh. This appears to be the most efficient approach for our data. Using tighter bounding volumes, such as nonaligned bounding boxes or higher order volumes which are useful for collision detection (see [21]) does not necessarily lead to major improvements in performance for computing intersections. Most collision algorithms are optimized for quick exclusion testing, but in our case we are expecting collisions. Using axis-aligned bounding boxes allows each collision test to be executed quickly, each one localizing the collision further.

**Perturbation method for computing intersections.** The polyhedral intersection algorithm relies crucially on the test whether an edge is intersected by a triangle. Usually, this test is implemented with the above-predicate, which determines whether a point $p_0$ is above or below the plane of a triangle. Consider a triangle with points $p_1, p_2, p_3 \in \mathbf{R}^3$. A point is above the triangle if the determinant $\det[p_0, p_1, p_2, p_3]$ is positive, where the points $p_i$ are represented in homogeneous form. The evaluation of this determinant is error-prone due to rounding errors in floating point arithmetic.

One can find a tight bound for the error [31], and if it leads to an undetermined result, resort to exact or arbitrary precision arithmetic [5, 31].

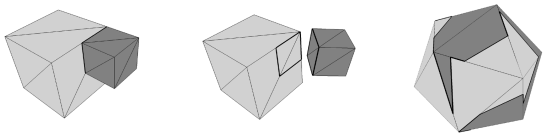
We use a simpler approach based on the perturbation scheme of [30]. In the case of a determinant sign uncertainty, we abort the intersection computation, perturb the input points by a small amount (depending on the triangle size) and perform the test again for all triangles affected by the perturbation. This is consistent with our goal of finding a consistent approximate intersection reliably and efficiently. Nevertheless, we note the concern about perturbation schemes in geometric modeling: parallel edges and other degeneracies are often intentional design decisions [7, 6, 33, 30].

Specifically we replace each point $p_i$ with a linear function $p_i(\varepsilon) = p_i + \varepsilon r_i$, where $r_i$ is a random direction. To determine the sign for the original data, we use

$$\lim_{\varepsilon \rightarrow 0^+} \operatorname{sign} \det[p_i(\varepsilon), p_j(\varepsilon), p_k(\varepsilon), p_l(\varepsilon)].$$

We can easily determine the sign of the expression: the determinant is a cubic polynomial in $\varepsilon$, and the sign is determined by coefficient of the linear term, if its sign can be computed reliably. If it can not, we choose a different perturbation, and recompute affected points. A more satisfying way of dealing with the problem is to use accurate calculations on the determinant.

Figure 8 shows some examples of intersections where degenerate cases are resolved with perturbation.



(a) (b) (c)

Figure 8: Degeneracies resolved by perturbation. (a) Two cubes with incident edges and vertices. (b) Intersection curves. (c) Intersection of two identical icosahedra. The polyhedra are rendered with perturbation to illustrate the topologically valid intersection.

# 4 Cutting and Merging Parametric Domains

Once the intersection topology is determined, we proceed to cut the parametric domains of the original solids and combine them into a

single parametric domain for the resulting object. At this stage we do *not* determine the positions of the control points for the new object; nor do we establish a final correspondence between the points of the new parametric domain and the original domain. We do assign initial values for this parameterization to provide initialization for optimization later. We used three main considerations when choosing the algorithm for cutting the domains:

- the topology of the cut should be the same as the topology of the intersection curve;
- as few as possible vertices should be added to the mesh representing the domain;
- the valence of the inserted vertices should be kept small.

The last two requirements often conflict with each other. For example, it might be necessary to split a single edge of the original domain into many pieces to capture the topology of the intersection curve. If we do not insert any additional vertices and simply connect all the points on the edge to the opposite vertex, this will considerably increase its valence. We prevent such situations by inserting vertices using only quadrisection of a triangle. This operation has to be followed by bisections of adjacent triangles to keep the mesh of the domain conforming; as a result, the valence of vertices may still increase, but much more slowly.

It should be noted that our algorithm intentionally ignores the geometry of the intersection curve and of the original surfaces to the extent it is possible without violating the first requirement.

There are two steps in constructing the domain for the result of a boolean operation: cutting each of the original domains $M_1$ and $M_2$ using the intersection curve, and merging relevant parts into a single domain. At the first stage, several pieces are produced for each of the initial meshes; depending on the operation, one or the other piece has to be discarded. We assume that the normals of the surfaces bounding the solids are oriented outwards, which allows us to identify the part of the mesh to be discarded locally.

More formally, the output of this algorithm is a new parametric domain $\widetilde{M}$, separated into two parts $\widetilde{M_1}$ and $\widetilde{M_2}$, such that $\widetilde{M_1} \cap \widetilde{M_2}$ is a single chain of crease edges forming a piecewise-linear curve $\widetilde{c}(t)$ along with maps $p_i$, $i = 1, 2$ from vertices of $\widetilde{M_i}$ to $M_i$. Note that on the curve $\widetilde{c}(t)$ both maps $p_i$ are defined. The additional property that we require is that $p_i(\widetilde{c})$ is in the image of the intersection curve $c_i$ in the parametric domain $M_i$, $i = 1, 2$, and for every point $\alpha$ on the curve $\widetilde{c}$ $p_i(\alpha) = c_i(t)$, $i = 1, 2$, for some $t$. The last condition ensures that the common curve $\widetilde{c}$ of subdomains $M_1$ and $M_2$ is mapped one-to-one to the spatial intersection curve $c(t)$.

## 4.1 Cutting

The result of cutting is a set of two domains $M_1'$ and $M_2'$ that are combined into the resulting domain $\widetilde{M}$ in the next stage. Simultaneously, a map $p_i'$ is constructed for each of the domains that maps it to the original domain $M_i$, and the image of the boundary $p_i'(\partial M_i')$ is contained in the image of the intersection curve $c_i(t)$.

Each of the domains is constructed gradually by refining a copy of one of the original domains $M_i$. Initially, $M_i'$ is just a copy of $M_i$, and $p_i'$ maps vertices of $M_i'$ to the corresponding vertices of $M_i$. For the intersection curve we also maintain two temporary images $c_i'(t)$ which define the position of the intersection curve in the new domains. Again, these images are initialized to copies of $c_i(t)$.

The cutting algorithm has two alternating steps: refinement and snapping (Figure 9). The goal of snapping is to identify points of the intersection curve $c_i'(t)$ with nearby vertices of the parametric domain. Snapping is optional, but important for obtaining domains

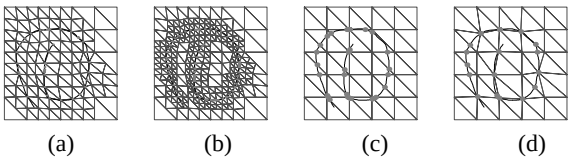



Figure 9: Refinement and snapping. (a) and (b) Refinement steps. (c) Curve in one of the domains $\widetilde{M_i}$, i=1,2. (d) Snapping the curve to vertices.

of low complexity. The goal of refinement is to reduce the size of triangles to simplify the shape of the intersection of the curve $c_i'(t)$.

**Refinement.** A triangle containing a part of the intersection curve (*a curve triangle*) is refined if the curve intersects the triangle boundary more than twice, does not intersect it at all, or intersects it twice but on the same side. We call such a triangle *bad*. When a triangle is refined, the maps $p_i'$ are assigned values in the domain $M_i$ for new vertices using midpoint subdivision of barycentric coordinates in the corresponding triangle of $M_i$. On each refinement step the positions of the images of the intersection curve $c_i'$ in the new domain $M_i'$ are recomputed by converting the barycentric coordinates in the parent triangle to the coordinates in the new triangles.

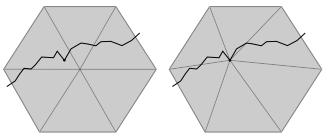


Figure 10: Left: Before snapping, all parameter values for curve vertices are mapped to chart; Right: After snapping, all chart coordinates are mapped back to triangles and barycentric coordinates for a new parameterization of the center vertex.

**Snapping.** We snap the curve $c_i'$ to a vertex of $M_i'$ if there is a point $\alpha$ on the curve for which one of the barycentric coordinates is larger than a *snapping constant* in the range 0 to 1. For most of the organic surfaces we used a snapping constant of 0.8, snapping to the curve if one of its sample's barycentric coordinates was 0.8 or greater. For the more mechanical shapes we effectively turned off snapping by setting this constant to 1.0, creating a closer match to the curve at the cost of a finer triangulation.

Snapping moves the curve to the vertex in two steps (Figure 10). First, we move the image of the vertex in the vertex chart $g_v(v)$ to the image of the curve $g_v(c_i'(t))$; a piecewise linear map $r$ maps the $k$-gon $\Pi_k$ to itself, with the point $g_v(v)$ mapped to a point $g(\alpha)$ on the curve. Then we apply the inverse transformation $r^{-1}$ to each point of the curve to move it inside $\Pi_k$. Finally, we apply the inverse of the parametric map $g_v$ to obtain new positions of the curve points in the parametric domain $M_i$. As a result, the point $\alpha$ of the curve $c_i'$ is mapped to $v$, and the curve is continuously shifted in the domain. Snapping may produce new bad triangles, forcing us to refine again. The complete cutting algorithm is given by the following pseudo-code.

**Algorithm** Snap and Refine
**do**
  **foreach** triangle vertex $v$ on the curve
    find closest curve point $\alpha$ to $v$
    snap $v$ to $\alpha$ if possible
  **if** there are bad triangles
    refine bad triangles
**while** there are bad triangles

Upon termination, one can easily find a sequence of edges which is topologically equivalent to the intersection curve: one simply has to split the triangles intersected by the curve in two (Figure 11). The values in the intersection curve images $c_i'(t)$ are used to set the initial parametric positions $p_i'(t)$ for the vertices on the edges along which we cut the domain $M_i'$.

Once the domain is cut, the part which is not required to construct the result of the boolean operation is removed. Note that in the resulting domain $M_i'$ the parametric positions in $M_i$ of all interior vertices are determined by midpoint subdivision of barycentric coordinates. Only the positions on the boundary are shifted towards the image of the intersection curve; this may create folds in the initial parameterization, which are removed at later stages.

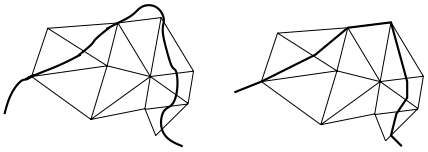


Figure 11: Left: curve in the parametric domain. Right: topologically equivalent strip of edges.

The procedure we have described is used for all vertices except for crease vertices. These vertices are constrained to snap only to the points on $c_i'(t)$ which are also on the same crease.

### 4.2 Merging

Next, the domains $M_1'$ and $M_2'$ are joined along their boundaries (Figure 12). The output of this stage is the domain $\widetilde{M}$ for the result, with the intersection curve corresponding to a sequence of crease edges. We describe the algorithm for a single connected intersection curve to simplify the presentation, but it can be applied to multiple intersection curves without any changes.

Initially, correspondence between points of the boundaries of the domains $M_1$ and $M_2$ is specified indirectly: points $\alpha_1$ and $\alpha_2$ on the boundaries of $M_1$ and $M_2$ are identical, if $p_i'(\alpha_i) = c_i(t)$ for some $t$ and $i = 1, 2$, i.e. they correspond to the same position on the intersection curve. However, it is not true in general that if $\alpha_1$ is a vertex, corresponding $\alpha_2$ is a vertex.

In order to match the vertices on corresponding boundaries, we will use similar snapping and refinement steps as before, modifying the domains $M_1'$ and $M_2'$, and the parameterizations $p_1'$ and $p_2'$. If one domain is lacking a boundary vertex, we can create a new one using refinement. Boundary vertices which almost coincide can be snapped together.

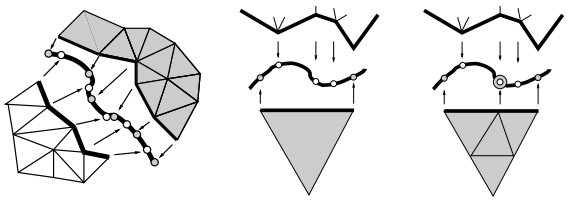


Figure 12: Domain merging. Left: Boundary vertices do not match up. Middle and Right: Triangle split matching a vertex on the other domain.

The algorithm has two phases and is entirely symmetric for both domains $M_1'$ and $M_2'$. The first phase is an iteration where close pairs of unmatched boundary vertices are snapped together. Suppose the domain $M_1'$ has a vertex $v_1$ with $p_1'(v_1) = c_1(t_1)$, and the other domain $M_2'$ a vertex $v_2$ with $p_2'(v_2) = c_2(t_2)$, and $|c(t_1) - c(t_2)| < \varepsilon$ for a choice of snapping constant $\varepsilon$. Then we adjust $p'(v_1)$ and $p'(v_2)$ so that $p_i'(v_i) = c_i((t_1 + t_2)/2)$. $\varepsilon$ is chosen to be a fraction of the minimal spatial distance between sequential vertices on the curve taken from the same side.

The second phase creates new vertices by refinement. Consider a boundary edge $e$ of $M_1'$ that corresponds to an intersection curve segment from $t_0$ to $t_1$. Assume that this segment contains a boundary vertex of the other domain at curve parameter $t$ for $t_0 < t < t_1$. In this case we split edge $e$ to get a new vertex $v$ and assign a parametric value $p'(v_1)$ to be $c_1(t)$. We repeat the steps until all vertices are matched.

Finally, we are in a situation when the boundaries of $M_1'$ and $M_2'$ can be trivially identified, to produce the domain $\widetilde{M}$ for the resulting surface. The subdomains $\widetilde{M_1}$ and $\widetilde{M_2}$ are simply images of $M_1'$ and $M_2'$ in the joined domain, and parameterizations $\widetilde{p}_i$ over the original domains are given by reassigning values of $p_i'$ on $M_i'$ to corresponding vertices in $\widetilde{M}$.

During the merging process, we also take care to mark the intersection as a crease, and mark as corner vertices any vertex formed by the intersection of a crease with the intersection curve. We further mark concave and convex sectors for subdivision rules of [4] based on the angle of a sector on the new surface in the limit, evaluated on the original surfaces.

## 5 Parameter Optimization

The snapping and merging steps guarantee that every vertex of the newly constructed domain $\widetilde{M}$ can be located in one of the original domains $M_1$ and $M_2$. This allows us to evaluate the corresponding surface positions for any point $\alpha$ in $\widetilde{M}$ as $f_i(p_i(\alpha))$ for $i = 1$ or 2. However, the maps $p_i$ are not one-to-one and can introduce substantial distortion into the surface shape.

As the next step of our algorithm, we optimize the parameterizations $\widetilde{p}_i$ of $\widetilde{M}_i$. No new maps or domains are created. This step has two goals:

- ensure that the parameterization is one-to-one;
- and that images of the triangles $\widetilde{M}$ have aspect ratios not too far from one.

We used two methods to optimize the parameterization, widely used Laplacian smoothing (e.g. [9]) and area-to-perimeter ratio maximization [2], combined with the optimization technique of [10].

The advantage of Laplacian smoothing is that it is relatively easy to evaluate and accelerate. However, it is known to produce results with flipped or extremely thin triangles, especially near boundaries with concave corners. Such boundaries are common in the meshes produced by taking differences of free-form solids. The second, slower, method is used to improve the parameterization and eliminate flipped triangles.

**Optimization functionals**. We define the distortion measures that we minimize for a vertex of a planar mesh, and then explain how we compute these quantities for a vertex of a parametric domain mapped into another parametric domain.

*Laplacian smoothing* minimizes the difference between the position of a vertex $q(v)$ and the barycenter of surrounding vertices:

$$E_{\text{Laplace}}(v) = \sum_{w \in N_1(v)} \|q(v) - q(w)\|^2$$

To obtain the functional to minimize we simply sum $E_{\text{Laplace}}(v)$ over all vertices.

The second distortion measure that we use is equivalent to *area-to-perimeter ratio*, which favors equilateral triangles [2]. Instead of computing the perimeter, we use the sum of squares of edges, to make it a smooth function of point positions.

$$E_{\text{ap}}(v) = -\min_{[u,v,w]} \frac{\text{Area}([q(u), q(v), q(w)])}{\|q(u)-q(v)\|^2 + \|q(u)-q(w)\|^2 + \|q(v)-q(w)\|^2}$$

where $[a, b, c]$ denotes a triangle with vertices $a, b, c$ and $[u, v, w]$ ranges over the triangles of $N_1(v)$. It is easy to show that the minimal value of the functional is attained for an equilateral triangle. In this case, to obtain the functional we take the maximal value over all vertices, which amounts to taking the maximal value over all triangles. The two distortion measures are compared in Figure 13.

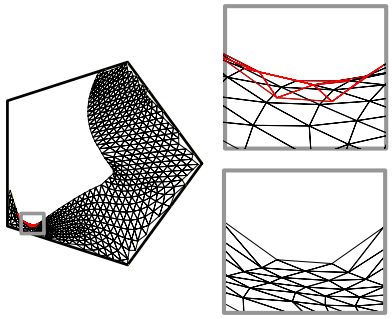


Figure 13: Left: Parameter optimization in a single chart. The boundary of the parameterization has concavities. Upper right: Laplacian smoothing produces a fold on the boundary (red). Lower right: Area-to-perimeter ratio optimization punishes for flipped triangles and produces a one-to-one parameterization.

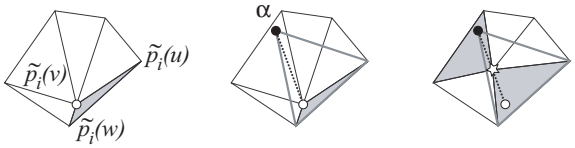



Figure 14: Area-to-perimeter ratio optimization. Left: initial ring; the triangle with worst aspect ratio is highlighted. Middle: position $\widetilde{p}_i(v)$ of vertex $v$ is moved along the dotted line segment towards optimal position $\alpha$. Right: the optimal position is marked with star. For this position distortion of two highlighted triangles is the same.

**Computing the distortion measures in a parametric domain.** The simplest approach to compute one of the distortion measures for a vertex $v$ of the domain $\widetilde{M}$ is to map the positions $\widetilde{p}_i(w)$ of the vertices $w$ of $\widetilde{M}$ adjacent to $v$ to the plane, and evaluate the functional here. As long as the distortion introduced by the map is small, we can safely use the distortion measure computed in this way for optimization. Linear charts described in Section 2 can be used to map points to the plane if we assume that all points $\widetilde{p}_i(w)$ involved in computing a single term in the functional are contained in a single triangle ring of one of the original domains $M_i$. In other words, the following condition holds:

**Single-ring condition.** *The parametric image $\widetilde{p}_i(N_1(v))$ of a ring of triangles centered at a vertex $v$ of $\widetilde{M}_i \subset \widetilde{M}$ is contained in a ring of triangles $N_1(w)$ for some vertex $w$.*

However, after the initial step (cutting and merging of the original domains) one cannot guarantee that images of all triangles of $\widetilde{M}$ are contained in a single ring of triangles in one of the domains $M_i$. Snapping may spread a few triangles between rings (Figure 15). We use adaptive subdivision of positions $\widetilde{p}_i$ of vertices of $\widetilde{M}$ in the parametric domain to refine the mesh $\widetilde{M}$ until the condition is satisfied.

**Subdivision in parametric domain.** As the vertices of a triangle of $\widetilde{M}_i$ may map to different triangles of $M_i$ it is not immediately clear how to subdivide parametric values at these vertices. We use the following approach:

Suppose $v_1$ and $v_2$ are two vertices in $\widetilde{M}_i$ such that $\widetilde{p}_i(v_1)$ and $\widetilde{p}_i(v_2)$ are in different nonadjacent triangles of $M_i$. Dijkstra's shortest path algorithm is used to find the chain of triangles between $v_1$ and $v_2$. The vertex $w$ in $\widetilde{M}_i$ inserted on the edge connecting $v_1$ $v_2$ is assigned position $\widetilde{p}_i(w)$ which is the center of the middle triangle on the path, if there is one. We apply subdivision until all vertices in $\widetilde{M}$ satisfy the single-ring condition.

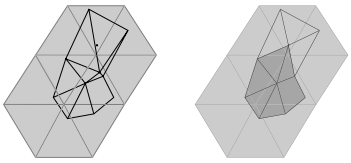


Figure 15: Left: Each one-ring is contained within a single chart from original mesh. Right: The dark one-ring is no longer contained in any chart.

**Optimization procedures.** A single step of Laplacian smoothing consists of moving the position $\widetilde{p}_i(v)$ of a vertex $v$ of $\widetilde{M}_i$ to the average of the positions of surrounding vertices. This can be done using the linear charts as described above; we move the position of each vertex as far as possible towards the barycenter without violating the single-ring condition for any of the rings depending on it.

For the area-to-perimeter ratio, the distortion is minimized by the following procedure (Figure 14). We pick the triangle $[v, w, u]$ in the ring $N_1(v)$ with maximal distortion and move the position $\widetilde{p}_i(v)$ of the vertex $v$ along the segment connecting the old position with a point $\alpha$, such that the triangle $[\alpha, \widetilde{p}_i(w), \widetilde{p}_i(u)]$ is equilateral. We do a search on the segment for the position that would minimize the area-to-perimeter distortion for the triangle, while keeping distortion of all other triangles lower, and respecting the single-ring condition for surrounding triangles (Figure 13).

## 6 Fitting

The previous stages of the algorithm yield the domain $\widetilde{M}$ for the resulting surface and parameterizations $\widetilde{p}_i$ of all vertices in subdomains $\widetilde{M}_i$ $i = 1, 2$ over the domains of the original surfaces. Furthermore, these parameterizations have the single-ring property. This allows us to compute the positions in $M_i$ not only for top-level vertices of $\widetilde{M}$, but also for any vertex added to $\widetilde{M}$ by subdivision. For this we only need to be able to assign parametric coordinates to the new vertices, which we do using linear charts as in Section 5.

However, no geometry is computed for the resulting surface. We fix the parameterizations and regard the new surface and parts of the old surfaces as functions on the newly constructed domain (Figure 16). Our goal is to compute the positions of control points for an approximation to the result, avoiding introducing details on fine levels. This is achieved by fitting surfaces defined by the control points to the original surfaces. The fitting procedure can be performed adaptively, increasing resolution where necessary. We consider the simplest version: we fit a subdivision surface with the

control mesh obtained by subdividing $\widetilde{M}$ $m$ times, to the original surface. We perform the fit in a hierarchical top-down manner, to obtain a multiresolution representation in the process.

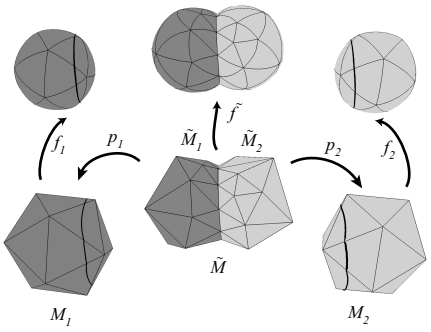



Figure 16: Surface fitting: We minimize the difference between the new and original surfaces.

Observe that we can evaluate the old surface $f_i$ at any vertex at any subdivision level of the result domain $\widetilde{M}$ using the composition $f_i \circ \widetilde{p}_i$. We use a generalization of Stam's technique [32] to piecewise smooth subdivision surfaces to evaluate $f_i$ at arbitrary points of the domains $M_i$, $i = 1, 2$.

Let $V_m$ be the set of vertices of the $\widetilde{M}$ after $m$ subdivision steps. Then the difference between two surfaces can be measured by the following functional:

$$\sum_{i=1,2} \int_{M_i} \| f_i(\widetilde{p}_i(\alpha)) - \sum_{v \in V_m} p_v^m B_v^m(\alpha) \|^2 d\alpha$$

where $p_v$ are the control points for the resulting surface on subdivision level $m$ and $B_v^m(\alpha)$ are the basis functions at vertices of level $m$. This functional is quadratic in $p_v^m$. The integrals can be computed explicitly, but we found that the results are not substantially different from replacing the continuous integrals with differences of control points $n$ levels below the level being fitted (we use $n = 3$). As a result, (6) is replaced with a different functional:

$$\sum_{i=1,2} \sum_{v \in V^{m+n}} \| f_i(\widetilde{p}_i(v)) - \sum_{v \in V_{m+n}} [S^n p^m]_v \|^2$$

where $[S^n p^m]_v$ is the control point at vertex $v$ obtained as a result of subdividing control mesh $p^m$ $n$ times. In vector notation, the expression above can be written as $\sum_{i=1,2} \sum_{v \in V^{m+n}} \| q^{n+m} - S^n p^m \|^2$, with $q_v = f_i(\widetilde{p}_i(v))$.

It is possible to show that the relative difference between the continuous and discrete functionals is bounded and derive accurate bounds using estimates on the magnitude of subdivision basis functions. The discrete form is a standard least-squares fit problem, which can be solved by a number of efficient methods (e.g. conjugate gradient). However, we have found that visually better results are obtained by imposing additional constraints on movement of the control points: on level $m$ we allow the points to move only in normal direction to the surface constructed by the fit on level $m - 1$. In this way, we obtain a mesh similar to the normal mesh of [12]. While the accuracy of the fit in the mean square sense decreases, the visual surface quality improves, as this approach prevents forming folds and ripples. At this time, we have no formal justification for imposing such constraints. It should be noted that the area that needs to be fitted grows when fitting finer levels of the multi-resolution mesh. This area grows by the size of the subdivision mask on the previous level, but this is not a large concern since it only adds a layer of vertices around the perimeter while the number of vertices internal to the optimized area grows exponentially at each subdivision level.

If high-accuracy approximation of the result is desired, once a good approximation is achieved by the fit, we switch to *quasi-interpolation* to compute further details on the surface ([23]). This is done solely for efficiency.

A careful examination of (6) reveals that it is possible to optimize not only the positions of the control points $p^m$, but also the parameterizations $\widetilde{p}$ to obtain a better approximation; our optimization of the parametric maps described in Section 5 tends to work in this direction. However, we make no attempt to optimize (6) directly; this is a possible direction for future research.

## 7 Results

We have tested our algorithm on various closed multiresolution surfaces. Figure 2 shows all the operations possible with two objects $A$ and $B$. Figure 22 shows the coarsest level triangulation. Note the low valence of the new vertices near the curve. Figure 1 also used all three boolean operations in its construction. The remaining figures show useful operations possible with boolean operations on free-form solids.

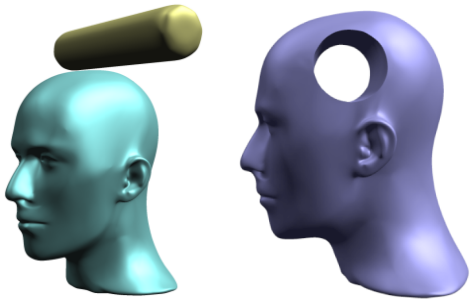


Figure 17: Subtracting a cylinder from the mannequin head.

## 8 Conclusion and Future Work

While our work addresses a classical problem in geometric modeling, our emphasis is quite different from most of the work we are familiar with. The algorithms that we have developed primarily address the problem of constructing a valid and usable model for the result of the boolean operation, rather than computing precisely all geometric objects characterizing the result (i.e. the intersection curve and parametric images of the intersection curve in the domains of the objects). Thus our algorithms can be viewed as complimentary to work on surface-surface intersections. Any accurate algorithm can be used to compute the intersection curve instead of our approximate algorithm. As future work, we plan to explore integration of precise surface-surface intersection algorithms into our framework. The algorithms described in Section 5 typically improve parameterizations. However, even defining rigorously measures of quality of the parameterization of one surface over another requires additional research. In Euclidean domains efficient techniques such as multigrid dramatically accelerate convergence of linear methods such as Laplacian smoothing. It is unclear how to apply similar techniques to functions with values in parametric domains.

Boolean operations on meshes with significantly different complexity can result in high valence vertices on the resulting mesh.

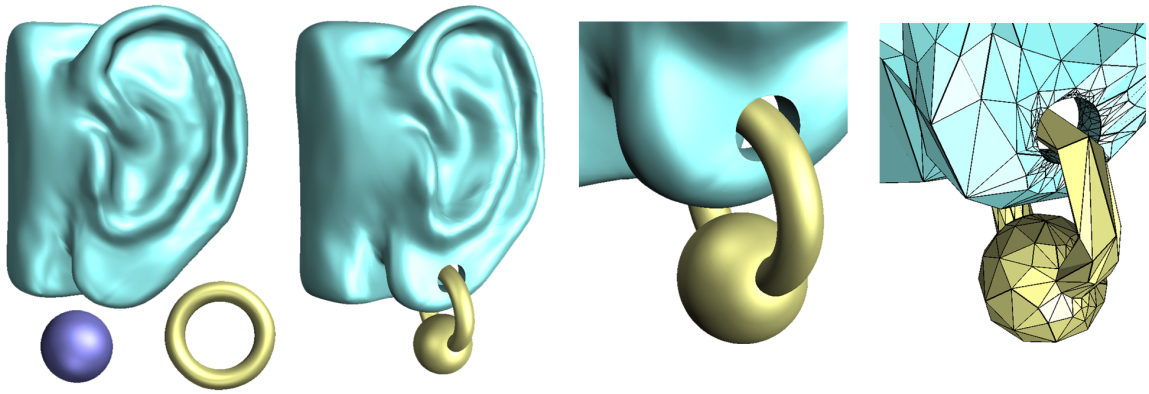


Figure 19: Modeling with multiresolution surfaces. The earring is assembled from a sphere and a torus. The ear is pierced with an enlarged version of the torus. The ear and the pierced ear are represented as multiresolution surfaces.

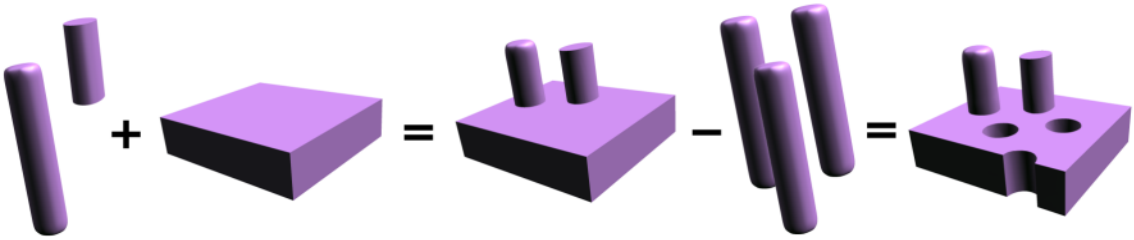


Figure 20: Unions and differences of piecewise-smooth surfaces. The resulting surfaces have creases and corners.

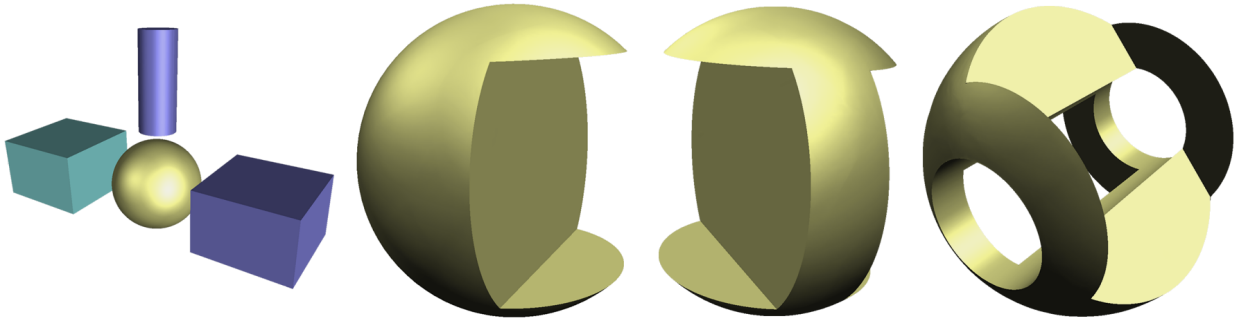


Figure 21: Sequence of difference operations: Subtracting two boxes and a cylinder from a sphere. The result has convex and concave corners. The subdivision scheme that we use [4] represents these features explicitely.

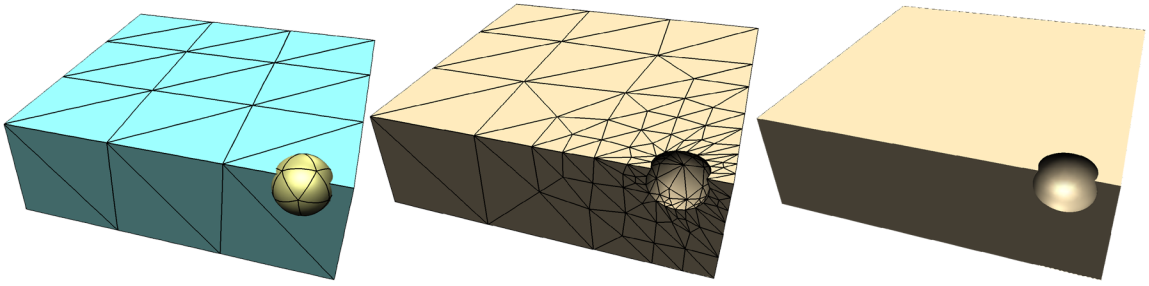


Figure 22: Difference of objects of different scale. Left: input surfaces. The patches of the box are much larger than the ones of the sphere. Middle: control mesh for the difference. The patch size changes gradually. Right: resulting surface.

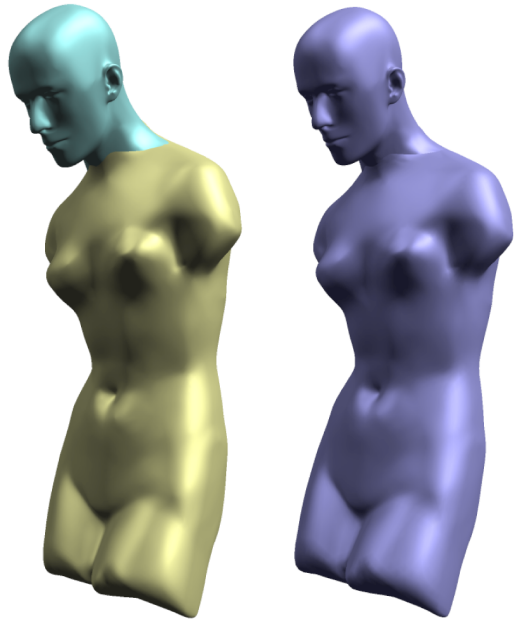


Figure 18: Union of the head and the body. Left: Original solids. Right: New solid obtained by union.

Any algorithm that significantly increases the number of regular vertices would result in a better surface.

It appears that we are able to approximate the results of boolean operations arbitrarily well, assuming that the topology of the intersection curve was resolved correctly. However, there is no guarantee that this is the case, and our algorithms require further analysis.

Our current implementation is not optimized for speed; the time required for operations is typically short: from real time to about five seconds for objects with larger control meshes such as the head/cylinder difference. We believe that for simple objects, boolean operations can be performed instantaneously.

## References


[1] A. Agrawal and A. Requicha. A paradigm for the robust design of algorithms for geometric modeling". *Computer Graphics Forum*, 13(3):33–44, 1994.

[2] R. E. Bank. *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations – Users' Guide 7.0*, volume 15 of *Frontiers in Applied Mathematics*. SIAM Books, Philadelphia, 1994.

[3] R. E. Barnhill, G. Farin, M. Jordan, and B. R. Piper. Surface/surface intersection. *Computer Aided Geometric Design*, 4(1-2):3–16, July 1987.

[4] Henning Biermann, Adi Levin, and Denis Zorin. Piecewise smooth subdivision surfaces with normal control. *Proceedings of SIGGRAPH 2000*, July 2000.

[5] C. Burnikel, S. Funke, and M. Seel. Exact arithmetic using cascaded computation. *ACM Symposium on Computational Geometry*, (14):175–193, 1998.

[6] Christoph Burnikel, Kurt Mehlhorn, and Stefan Schirra. On degeneracy in geometric computations. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (Arlington, VA, 1994)*, pages 16–23, New York, 1994. ACM.

[7] Katrin Dobrindt, Kurt Mehlhorn, and Mariette Yvinec. A complete and efficient algorithm for the intersection of a general and a convex polyhedron. In *Algorithms and data structures (Montreal, PQ, 1993)*, pages 314–324. Springer, Berlin, 1993.

[8] D. Epstein, N. Gharachorloo, F. Jansen, J. Rossignac, , and C. Zoulos. Multiple depth-buffer rendering of csg. Technical report, IBM Research Report, 1989.

[9] D. A. Field. Laplacian smoothing and delaunay triangulations. *Comm. Applications: Numerical Methods*, (4):709–712, 1988.

[10] Lori Freitag, Mark Jones, and Paul Plassmann. A parallel algorithm for mesh smoothing. *SIAM J. Sci. Comput.*, 20(6):2023–2040 (electronic), 1999.

[11] Jack Goldfeather, Jeff P. M. Hultquist, and Henry Fuchs. Fast constructive-solid geometry display in the pixel-powers graphics system. *Computer Graphics (Proceedings of SIGGRAPH 86)*, 20(4):107–116, August 1986. Held in Dallas, Texas.

[12] Igor Guskov, Kiril Vidimče, Wim Sweldens, and Peter Schröder. Normal meshes. *Proceedings of SIGGRAPH 2000*, pages 95–102, July 2000.

[13] Mark Halstead, Michael Kass, and Tony DeRose. Efficient, fair interpolation using Catmull-Clark surfaces. In *Computer Graphics Proceedings*, Annual Conference Series, pages 35–44. ACM Siggraph, 1993.

[14] Christoph M. Hoffmann. *Geometric and solid modeling: a introduction*. San Mateo, California: Morgan Kaufmann, 1989.

[15] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Huber Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconsruction. In *Computer Graphics Proceedings*, Annual Conference Series, pages 295–302. ACM Siggraph, 1994.

[16] Josef Hoschek and Dieter Lasser. *Fundamentals of computer aided geometric design*. A K Peters Ltd., Wellesley, MA, 1993. Translated from the 1992 German edition by Larry L. Schumaker.

[17] Leif P. Kobbelt, Katja Daubert, and Hans-Peter Seidel. Ray tracing of subdivision surfaces. *Eurographics Rendering Workshop 1998*, pages 69–80, June 1998. Held in Vienna, Austria.

[18] Shankar Krishnan and Dinesh Manocha. An efficient surface intersection algorithm based on lower-dimensional formulation. *ACM Transactions on Graphics*, 16(1):74–106, January 1997. ISSN 0730-0301.

[19] Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. *Proceedings of SIGGRAPH 98*, pages 95–104, July 1998.

[20] A. Levin. Combined subdivision schemes for the design of surfaces satisfying boundary conditions. *Computer Aided Geometric Design*, 16(5):345–354, 1999.

[21] Ming Lin and Stefan Gottschalk. Collision detection between geometric models: A survey. *Proceedings of IMA Conference on Mathematics of Surfaces*, 1998.

[22] Lars Linsen. Schneiden und vereinen von kontrollnetzen (intersection and merging of control meshes.). Master's thesis, Universität Karlsruhe, 1997. in German.

[23] Nathan Litke, Adi Levin, and Peter Schröder. Trimming for subdivision surfaces. Technical report, Caltech, 2000.

[24] Michael Lounsbery, Tony DeRose, and Joe Warren. Multiresolution analysis for surfaces of arbitrary topological type. *Transactions on Graphics*, 16(1):34–73, January 1997.

[25] Lee Markosian, Jonathan M. Cohen, Thomas Crulli, and John F. Hughes. Skin: A constructive approach to modeling free-form shapes. *Proceedings of SIGGRAPH 99*, pages 393–400, August 1999. Held in Los Angeles, California.

[26] K. Pulli and M. Lounsbery. Hierarchical editing and rendering of subdivision surfaces. Technical Report UW-CSE-97-04-07, Dept. of CS&E, University of Washington, Seattle, WA, 1997.

[27] Ari Rappoport and Steven Spitz. Interactive boolean operations for conceptual design of 3-d solids. *Proceedings of SIGGRAPH 97*, pages 269–278, August 1997. Held in Los Angeles, California.

[28] J. Rossignac and A. Requicha. Solid modeling. In J. Webster, editor, *Encyclopedia of Electrical and Electronics Engineering*. John Wiley and Sons, 1999.

[29] T. Sederberg and T. Nishita. Geometric hermite approximation of surface patch intersection curves. *Computer Aided Geometric Design*, 8(2):97–114, 1991.

[30] R. Seidel. The nature and meaning of perturbations in geometric computing. *Discrete Comput. Geom.*, 19(1):1–17, 1998.

[31] Jonathan Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. Technical report, Carnegie Mellon University, 1996.

[32] Jos Stam. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. *Proceedings of SIGGRAPH 98*, pages 395–404, July 1998. Held in Orlando, Florida.

[33] A. James Stewart. Local robustness and its application to polyhedral intersection. *Internat. J. Comput. Geom. Appl.*, 4(1):87–118, 1994.

[34] Gabriel Taubin. A signal processing approach to fair surface design. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 351–358. ACM SIGGRAPH, Addison Wesley, August 1995.

[35] Denis Zorin, Peter Schröder, and Wim Sweldens. Interactive multiresolution mesh editing. *Proceedings of SIGGRAPH 97*, pages 259–268, August 1997.