EUROGRAPHICS 2010 / T. Akenine-Möller and M. Zwicker
(Guest Editors)

*Volume 29 (2010), Number 2*

# Exact and Robust (Self-)Intersections for Polygonal Meshes

Marcel Campen and Leif Kobbelt

Computer Graphics Group, RWTH Aachen University, Germany

**Abstract**

*We present a new technique to implement operators that modify the topology of polygonal meshes at intersections and self-intersections. Depending on the modification strategy, this effectively results in operators for Boolean combinations or for the construction of outer hulls that are suited for mesh repair tasks and accurate mesh-based front tracking of deformable materials that split and merge. By combining an adaptive octree with nested binary space partitions (BSP), we can guarantee exactness (= correctness) and robustness (= completeness) of the algorithm while still achieving higher performance and less memory consumption than previous approaches. The efficiency and scalability in terms of runtime and memory is obtained by an operation localization scheme. We restrict the essential computations to those cells in the adaptive octree where intersections actually occur. Within those critical cells, we convert the input geometry into a plane-based BSP-representation which allows us to perform all computations exactly even with fixed precision arithmetics. We carefully analyze the precision requirements of the involved geometric data and predicates in order to guarantee correctness and show how minimal input mesh quantization can be used to safely rely on computations with standard floating point numbers. We properly evaluate our method with respect to precision, robustness, and efficiency.*

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.5]: Computational Geometry and Object Modeling—

## 1. Introduction

Changing the topology of polygonal surfaces along curves of intersection has proven to be a complicated task. Operations of this kind are necessary in various applications, ranging from the computation of Boolean combinations to the repair of self-intersecting meshes and the tracking of surfaces of deforming material (cf. Figure 1). The correct and consistent determination of the intersection loci is numerically hard to handle, and the discrete nature of a polygonal mesh, consisting of entities of differing dimensionality, introduces further challenges that have to be met in order to achieve robustness.

Boolean set operations on polygonal meshes are a common practice in modeling tools, CAD/CAM applications, simulation systems and many other areas of computer graphics. They implement the intuitive concept of removing or adding solid parts in order to construct complex objects from simpler ones. However, existing methods have various drawbacks, ranging from robustness issues to poor performance, due to the described challenges in intersection handling.

Related applications that similarly require the topology of meshes to change, like mesh repair or front tracking, face the same problems. The input is a mesh that contains self-intersections and/or redundant internal structures, possibly introduced by merging or splitting of soft material during the process of deformation simulation. These artifacts have to be resolved by changing the topology of the represented surface at the intersections to obtain a plausible result.

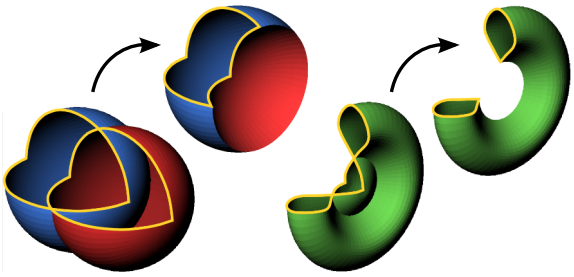


**Figure 1:** *Illustration of (self-)intersections in polygonal meshes (cut open for visualization, cuts yellow). These are resolved by modifying the surface topology, e.g. resulting in Booleans or outer hulls depending on the specific strategy.*


Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

We present a general scheme that is able to perform such topology changing operations on polygonal meshes exactly and robustly, i.e. it is algorithmically correct and complete, producing accurate output for any valid input. Compared to state-of-the-art methods that exhibit these features, we achieve higher performance and a smaller memory footprint. The main ingredient, which is the key to achieving robustness and exactness while maintaining high performance, is a paradigm of plane-based geometry representation and processing. By applying it rigorously, it allows us to completely avoid arbitrary precision arithmetics that are commonly employed for the sake of robustness in this context.

The explicit handling of all possible intersection constellations and degenerate contact situations, that is necessary in most methods that deal with intersecting meshes, is error-prone and leads to high algorithmic complexity. Polyhedra representation schemes that rely on binary space partitioning (BSP) have shown to be able to greatly reduce efforts that have to be spent on this issue. They fit nicely into the plane-based setting, and we employ them to reduce complexity and achieve robustness in an elegant manner.

In order to significantly improve efficiency, we do not process the input objects globally. We build upon the fact that actual changes to the input only have to be made at the (self-)intersections and apply an operation localization scheme. The intersection region is covered by a set of small volume cells, and processing is performed locally within each cell. The global solution is then composed from the local solutions and unaltered portions of the input.

The rest of this paper is structured as follows: In Chapter 2 we introduce previous and related work that our approach builds upon or can be compared to. In Chapter 3 we give a description of the paradigm of plane-based geometry representation that is the key ingredient of our localized intersection handling scheme presented in Chapter 4. The application of these concepts to the problem of Boolean operations is then presented in Chapter 5 and extended to the construction of outer hulls for mesh repair and mesh-based front tracking in Chapter 6. Finally, experimental results and a discussion are provided in Chapters 7 and 8.

## 2. Related work

Boolean operations for polyhedral solids have a long history of notorious robustness issues since they were introduced in the 1980s [RV85, ABJN85, LTH86]. By implementing such methods with arbitrary precision arithmetics and explicitly handling numerous difficult case distinctions and all kinds of degeneracies, one could possibly achieve accuracy and robustness, but at the price of unacceptable performance.

Sugihara and Iri [SI89] introduced the concept of plane-based representations for polyhedra. By using plane equations as primary geometric information they were able to perform rudimentary modeling operations robustly. This idea was later picked up by Fortune [For97] and coupled with symbolic perturbation techniques in order to achieve more general modeling operations – at the expense of increased algorithmic complexity.

Naylor, Amanatides, and Thibault [TN87, Thi87, NAT90] discovered that binary space partitioning (BSP) structures can be used to represent polyhedral objects, and Boolean operations can be performed by merging such structures. The implementation of this merging procedure turns out to be much simpler than other approaches due to its recursive nature. Additionally, numerous case distinctions are avoided, and degeneracies do not require special attention.

Quite recently Bernstein and Fussell [BF09] married these two concepts of plane-based geometry representation and BSP merging with the goal of constructing exact and robust Boolean operators with low algorithmic complexity. In our work, we build upon this promising idea. They managed to achieve this goal within a system that processes polyhedral objects that are already represented by plane equations – conversion from other representations like standard polygonal meshes requires repair heuristics and may affect correctness. Additionally, efficiency was mainly achieved for the inner loop of the processing, which lends the system to sculpting applications, but costly pre- and post-processing steps that perform multi-stage conversions between different types of representation lead to a reduced overall efficiency. Finally, their method outputs a polygon soup without connectivity information, which implies additional efforts if the face neighborhood needs to be established.

Hence, the operations provided by the CGAL package [GHH*03] probably still embody the state-of-the-art in robust, exact Booleans. They are able to perform calculations with arbitrary precision arithmetics and operate on a Nef polyhedra structure [Nef78]. While this is able to represent dangling and isolated mesh elements as well as open boundaries, this generality is rarely needed in practice. The requirement of arbitrary precision arithmetics for robust operation significantly affects efficiency. Additionally, the underlying data structure is fairly memory inefficient and thus inhibits the processing of large meshes. In contrast, our scheme, applied for Booleans, is guaranteed to produce correct results by using only fixed precision arithmetics and consumes far less memory.

Operation localization schemes have been employed mainly for two reasons: in order to increase efficiency and in order to restrict quality-impairing operations to regions where they are inevitable. In the area of intersection handling, Bischoff and Kobbelt [BK05] applied localization to the problem of model repair, Wojtan et al. [WTGT09] as well as Du et al. [DFG*06] in order to handle topological changes in deforming meshes, and recently Pavić et al. [PCK09] employed such a scheme to the computation of Boolean operations on solids. In all these cases, only the regions that are actually affected by some change are processed. They are re-

placed by substitute parts obtained by volumetric iso-surface extraction [LC87, JLSW02] that can be performed robustly. However, the application of such schemes usually requires the modified mesh parts to be stitched to clipped unmodified parts. The requisite operations tend to introduce severe robustness issues. In contrast, during our plane-based processing, this clipping turns out to be an easy task.

The general limitation of the aforementioned and other methods that use volumetric representations to deal with intersections and topology changes is that of discreteness: small or nearby surface features are not resolved correctly due to a limited resolution. Varadhan et al. [VKSM04, VKZM06] presented criteria for the adaptive subdivision of volumetric grids that at least allow to give guarantees regarding topological correctness during conversion to and from such representations. These could possibly be applied in these methods in order to enhance the quality of the output, but only to a certain level and at the expense of a higher complexity that strongly depends on the feature structure.

## 3. Plane-based geometry representation

The primary geometry information commonly used in the representation of objects by polygon meshes are the vertex coordinates. The geometry of the edges and faces is implicitly defined by these values. In contrast, we choose to explicitly represent the geometry of the faces by plane equations and let vertices be defined implicitly by plane intersections.

### 3.1. Basic concept

The concept of plane-based representation of polygonal meshes was first described by Sugihara and Iri [SI89]. This kind of representation provides one important advantage when it comes to tasks that involve changing the topology of solids represented by meshes like the evaluation of Boolean expressions: no new primary geometry information has to be constructed to obtain the resulting polyhedron – it is composed of a subset of the planes of the input polyhedra. Hence, opposed to the case of using vertex coordinates, that inevitably necessitates the *construction* of new geometry information, only geometric *decision predicates* are required to compose the output polyhedron from the face planes of the input. Since the input is usually given in a numerical representation with finite precision, we can determine an upper bound on the precision that is needed to make correct decisions. The a priori knowledge of this upper bound allows us to use *fixed precision* predicates that are specifically tailored to the precision required in the worst case, resulting in a vastly better performance compared to techniques for *arbitrary numerical precision*, that are usually required when constructions are part of the processing.

Our processing is rigorously based on this paradigm of plane-based geometry representation that allows us to perform fully robust, exact computations using only fast fixed precision predicates. These predicates [BF09] take planes as arguments and check coincidence and co-orientation of planes, orientation of a plane with respect to a point defined by the intersection of three planes, and whether three planes intersect in a unique point. The latter one can also be used in negated form to check if three planes, that are known to intersect in a common point, even intersect in a common line. We implement them using filtering techniques proposed by Shewchuk [She97].

### 3.2. Exact conversion

We consider the conversion from vertex-based mesh representation to our internal plane-based representation an integral step of our processing. Conforming to our goal of achieving exact operations, we strive to perform this conversion robustly and without error.

Let the vertex coordinates of the input mesh be represented with a precision of $L$ bits including sign, which means that vertices are located at one of $2^L$ positions uniformly spaced within the bounding box along each axis. Further let $\delta \leq 2^{K-1-L}$ be the relative length (in max-norm) of the longest edge in the mesh (relative to the bounding box), which is equivalent to saying we need $K$ bits to represent edge vector coordinates.

The coefficients of the plane equation $n_x x + n_y y + n_z z + d = 0$ for a planar face $F$ with vertex positions $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \ldots$ can be computed by matrix determinants. The most complex one is $d = -|\mathbf{p}_0\ \mathbf{p}_1\ \mathbf{p}_2|$, composed of six triple products that need to be added or subtracted ($n_x$, $n_y$, and $n_z$ are sums of six double products only). It can also be computed from differences due to $|\mathbf{p}_0\ \mathbf{p}_1\ \mathbf{p}_2| = |\mathbf{p}_0\ \mathbf{p}_1 - \mathbf{p}_0\ \mathbf{p}_2 - \mathbf{p}_0|$. Since each triple product multiplies one vertex coordinate ($L-1$ bits plus sign) with two edge vector coordinates ($K-1$ bits plus sign), we end up with a (conservatively estimated) maximum precision of $M = (L-1) + 2(K-1) + 3 + 1$ bits (including one sign bit) that is needed to guarantee exact calculation. By substitution we obtain $M \geq 3L + 2\ \log_2 \delta + 3$, relating vertex coordinate precision $L$ and plane coefficient precision $M$ by the maximum edge length $\delta$.

Using this formula, one can determine the precision the data types and predicates for the planes need to be tailored to in order to handle the desired class of input. In our prototypical implementation we choose to embed plane coefficients into standard floating point numbers ($M = 53$), since the floating point units of modern CPUs are highly optimized and much work has been done on adaptive exact predicates for floating point numbers [She97, Pri91]. Then, assuming for instance that the input is represented with $L = 20$ bits precision, we end up with an allowed maximum relative edge length of $\delta \leq 2^{-5}$. Moreover we can explicitly exploit the fact that x86-CPUs internally use extended-precision floating point numbers ($M = 64$) anyway (e.g. exposed as `long double` by most C/C++ compilers), and

for instance allow an input precision of $L = 22$ bits and even obtain $\delta \leq 2^{-2.5} \approx 0.18$ which should be satisfied in almost all practically relevant cases. For $\delta \leq 2^{-5.5} \approx 0.022$ we can even handle full IEEE 754 single precision input ($L = 24$ bits) while guaranteeing exact conversion.

Note that these precision and/or edge length bounds that are imposed by a specific implementation do not really restrict the class of input meshes that can be handled correctly: one can always apply quantization to the input coordinates and/or subdivision to long edges in order to make the input conform to the bounds. Using quantization may of course introduce geometrical error by shifting vertices slightly, but only in the magnitude order of about $2^{-20}$ or even much smaller when assuming or enforcing the absence of very long edges. Most importantly, the inner topology of the polygon mesh is not affected, which is in contrast to Bernstein and Fussell [BF09] where vertices are often split into a collection of valence 3 vertices due to round-off error, additionally introducing nearly degenerate polygons inbetween the original input polygons.

### 3.3. Polygon construction

With this exact plane construction at hand, we are now able to convert a mesh of planar polygons into its plane-based representation without error. If the input contains non-planar polygons (usually due to slight numerical deviations), we can triangulate them to yield a geometry that can be represented by planes. We use a plane-based polygon data structure [BF09] that represents a polygon by its supporting plane (constructed from three of the vertices of the polygon) and a list of bounding planes (that are non-coplanar with the supporting plane and implicitly define the edges) ordered circularly. Each of these bounding planes is constructed from two consecutive vertices of the polygon and an arbitrary third point that does not lie in the supporting plane and is representable with input precision.

## 4. Localized intersection handling

We now present our general scheme for exact and robust handling of (self-)intersections in plane-based polygon meshes. In the subsequent chapters, it is applied to specific problems. The basic concept is illustrated in Figure 2.

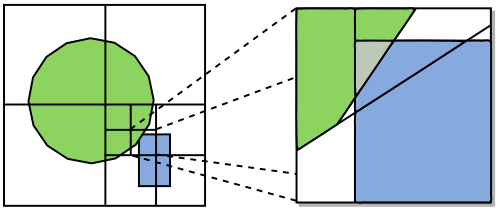


**Figure 2:** *2D-Illustration of the basic concept: an octree is refined to the intersections. BSPs are nested into the affected cells to locally perform the appropriate topological changes.*

### 4.1. Localization

We construct a set of disjunct convex spatial cells, such that their union contains the curves of intersection of the input mesh(es) in its interior. Additionally, in order to avoid special case handling, we require their bounding planes to contain none of the vertices of the input. We call them *critical cells*. Only within these critical cells processing is applied locally. We strive to construct the critical cells in a way that they contain (i.e. are intersected by) an approximately constant number of input polygons. While the BSP techniques we are going to apply within the cells have time complexities of at least $O(n \log n)$ ($n$ being the number of input polygons) in the global case, we are thereby able to lower that to about $O(\sqrt{n})$ plus cell construction and final joining of the local results in many cases. This is due to the fact that the effort per cell is in the order of $O(1)$ and the number of critical cells is usually $O(\sqrt{n})$ if the mesh resolution is somewhat uniform; if it varies significantly, i.e. the input contains polygons of widely differing sizes or bad aspect ratios, this number of critical cells might be higher or lower. One reason is the fact that the local mesh resolution in the intersection regions might differ from the average. Another reason for this sensitivity to non-uniformity is the fact that polygons might span multiple cells. This might induce noticeable overhead if very large or long polygons intersect smaller cells.

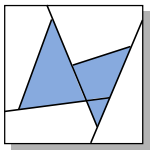
While sophisticated methods could be used to cover the intersection regions with as few disjunct convex cells containing a constant number of polygons as possible, we choose to construct such cells by fairly simple adaptive refinement of an octree. This scheme does not yield the minimum number of critical cells possible, but we felt that further efforts, e.g. for the construction of more flexible kd-trees, would be too expensive to be overcompensated by the consequent improvement in cell count.

We start with an octree root cell encompassing the whole collection of input objects, and then recursively subdivide a cell whenever its closed volume contains a (self-)intersection and still more than $m$ input polygons. This value $m$ is the only parameter of our method, and a good general choice proved to be $m = 17$, since it consistently resulted in minimal runtimes in most of our experiments. Using this subdivision rule the octree adapts to the local mesh resolution. Depending on the particular application, different methods can be used to determine if a cell contains a (self-)intersection (cf. Chapters 5 and 6). Note that this test may also be made conservative for efficiency reasons. We keep track of polygons intersecting cells by storing links at cells and distributing them to the children when subdividing. Polygon-cell intersection tests can be performed inaccurately for efficiency by testing against sufficiently enlarged cells since false-positives are not problematic. The leaf cells that contain (self-)intersections are the critical cells we apply our local processing to – they are disjunct, convex, and contain all curves of intersection by construction. By positioning the

octree in a way that the (axis-aligned) cell boundaries are located between possible vertex coordinates, we also easily prevent special cases arising from input vertices, edges, or faces lying exactly on the cell boundaries.

### 4.2. BSP techniques

As observed by Thibault and Naylor [TN87] any polyhedron can be represented by a BSP-tree with labeled leafs. Each leaf corresponds to a spatial cell and is labeled 'inside' or 'outside', as illustrated here in 2D. The boundary between inside and outside cells defines the polyhedron. Such a BSP-tree can be constructed by recursively *inserting* the polygons of the polyhedron one by one into an initially empty BSP-tree and creating new splitting planes on demand [TN87]. Afterwards, each cell of the BSP-tree lies either completely inside or completely outside of the polyhedron and is labeled accordingly. For self-intersecting meshes inside and outside is not inherently defined, such that only (initially) unlabeled BSPs can be constructed and labels have to be subsequently created depending on the application scenario. Note that these inside/outside labels implicitly specify the surface topology in the BSP – in its core the task of topology modification reduces to label switching in this setting.



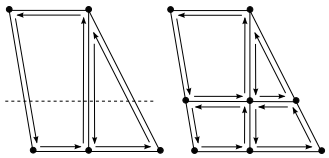
Since we are going to apply operations locally, we need to nest BSPs into the critical cells, i.e. we need to restrict them to the volume of a convex region. We clip the polygons that are to be inserted against the boundary planes of that critical cell by the splitting routine presented by Bernstein and Fussell [BF09]. Note that it is this clipping that requires the critical cells to be convex. Since we are performing all requisite polygon clipping and splitting operations in the plane-based setting, no error is introduced: only exact predicates are used, no geometric constructions are involved.

In order to obtain a polygonal boundary representation from a BSP representation in the end (after application-dependent label switching, cf. Chapters 5 and 6), we apply the boundary extraction method presented by Thibault [Thi87]: a polygonal representation of a splitting plane node is sent down the subtrees of that node, partitioning it into fragments that separate exactly two cells. If a fragment separates an inside from an outside cell, it is part of the boundary. In the end this yields a complete polygonal boundary representation without connectivity information. Our method for establishing connectivity is presented in the next section.

For the task of determining BSP-cell labels when non-oriented or self-intersecting meshes are converted into BSPs, specific tools are required. Note that the volume represented by the BSP is not partitioned into inside and outside, but simply into different *components* (sets of BSP-cells) by the polygonal input surface in this case (cf. Figure 3, left). For some applications (cf. Section 6.2) it is necessary to identify them and set labels accordingly, i.e. mark some components (more precisely: their contained cells) as inside, the others as outside. We determine these components by a flood-filling on the BSP-cells that respects the input surface as limit. In order to guide this flood-filling, we determine the BSP-cell adjacency graph, that contains an edge between two cells iff they share a common cell face. This is done by a slightly modified variant of the boundary extraction method of Thibault: the polygonal fragments that separate two arbitrary cells correspond (are dual) to the edges we need for our adjacency graph (cf. Figure 3, middle). We also send the input polygons down the BSP-tree to determine those edges that connect cells that are separated by the input surface. These edges are deleted in the graph, and we can start our flood-filling on the resulting graph to conquer the components of the partitioned volume.

### 4.3. Clipping and connecting

The regions of the input meshes that are affected by intersections have to be replaced by the modified boundaries obtained from the processing by BSP techniques. To this end, we clip the input meshes against the critical cells. This effectively restricts the input meshes to the non-critical region. We again perform this clipping in the plane-based setting to achieve exactness and robustness. The input meshes, represented by a mesh data structure based on halfedges, are partially transformed into plane-based representation first of all, maintaining connectivity: all polygons intersecting critical cells are converted and each polygon bounding plane is associated with the respective halfedge. Each polygon of these plane-based mesh parts is then clipped against each critical cell it intersects by successively splitting it by the six boundary planes of a cell and discarding the inner part. The clipping is performed using an extended version of the polygon splitting routine used in Section 4.2 that maintains connectivity appropriately and creates new halfedge and vertex entities where required.



In the end, all that is left to do is to establish full connectivity in order to create a mesh out of the polygonal boundary

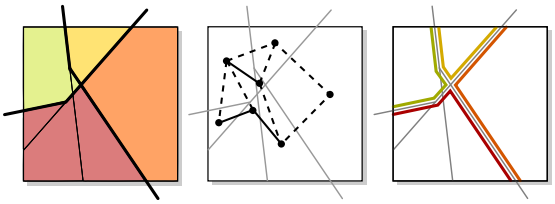
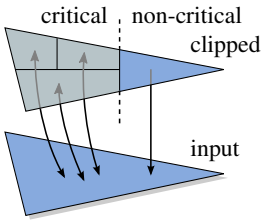


**Figure 3:** *Left: critical cell with local BSP, partitioned into four components by the input surfaces (thick lines). Middle: the corresponding BSP-cell adjacency graph, edges dual to input polygons dashed. Right: extracted boundaries of the four components conquered by flooding in the graph.*

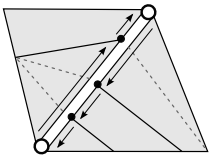
parts. The part in the non-critical region still is connected, but the polygons within a critical cell need to be connected (local connectivity) and the connectivity between these critical parts and between critical and non-critical parts (global connectivity) has to be established. We do this by identifying vertices, defined implicitly by plane triples, of different extracted polygons that are coincident. This can be decided exactly using the orientation predicate.

**Local connectivity:** Since, for a good choice of parameter $m$ (cf. Chapter 4.1), the number of polygons in a critical cell is low, we can test each vertex against each other – the application of more sophisticated techniques is needless at this scale. In this way we obtain shared-vertex information within the critical cell.

**Global connectivity:** In order to avoid testing each vertex against each other – which would be prohibitively costly at this global scale – we make use of links between input polygons and corresponding extracted polygons we maintain during BSP-tree construction and extraction, as illustrated here. This allows us to restrict vertex coincidence tests to fragments of the same original polygon. Note that in general also fragments of adjacent original polygons could need to be connected, but the special positioning of the octree effectively prevents this (cf. Section 4.1).




As the extracted polygon set does not necessarily form a polygonal complex, T-junctions may be present. These can be resolved without affecting geometry by simply introducing an additional degenerate $180^\circ$ corner in the opposite polygon, or by splitting this polygon in two with a new edge. We detect loops of topologically open halfedges that are completely linear by using our exact collinearity predicate. Those loops of length larger than two halfedges contain T-junctions, as illustrated in the opposite figure. We determine the relative order of both sides' T-vertices along the line (exactly using the orientation predicate again) and per halfedge collect an ordered list of the opposite T-vertices that are to be inserted on that edge. Afterwards, we convert the plane-based mesh parts into standard representation by computing coordinates for each involved vertex and topologically incorporating the collected T-vertices into the polygon edges. Computation of the coordinates of a vertex involves intersecting three planes. This can be done to any desired precision by techniques presented by Priest [Pri91]. Of course, if not computed to full precision, but for instance rounded to input precision, a small geometrical error is introduced, that – in rare cases – might also lead to microscopic self-intersections



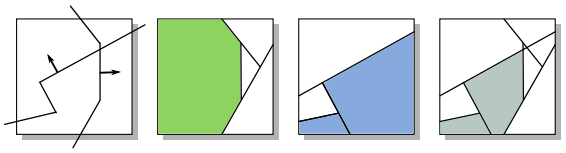


**Figure 4:** *2D illustration of two input surfaces intersecting within a critical cell, their BSP-representations (inside cells colored) and the merged BSP (using "intersection" logic)*

of the output. This is an instance of the omnipresent *geometric rounding problem*, see e.g. Li et al. [LPY05] for further details on this topic.

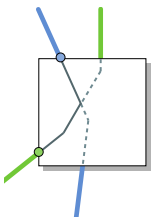
Note that the application of BSP-based techniques introduces some (purely topological) polygon fragmentation. While this is heavily reduced and locally restricted to the intersection regions due to our localization scheme, it might still be desirable to eliminate unnecessary fragmentation. A mesh simplification procedure [KCS98] tailored to condensing only exactly coplanar polygons can be applied for this task without impairing accuracy of the result.

## 5. Boolean operations

Using the topology modification scheme presented in the last chapter, we are now able to perform truly exact, robust Boolean operations on polygonal meshes that represent solids. Our goal is to provide a method that is efficient on the whole, i.e. from standard polygon mesh input to polygon mesh output, eliminating the drawbacks of the method of Bernstein and Fussell as outlined in Chapter 2.

The (conservative) intersection test required for the refinement criterion of the octree is very simple in this case, since intersections occur between different input objects: whenever an octree cell is intersected by different meshes, it is split. Within a critical cell, we construct a labeled BSP for each input object. Inside/outside labels are obtained from the orientation of the polygons. These BSPs are then recursively merged using the technique proposed by Naylor et al. [NAT90] and the desired logical expressions are applied to the labels at the leafs. This results in a BSP-representation of the Boolean combination restricted to the critical cell (cf. Figure 4), and we can extract the polygonal boundary.

After establishing connectivity, some of the non-critical parts remain unconnected, as illustrated here. These parts with open boundaries do not belong to the surface of the actual Boolean combination – they lie in its interior or exterior and are deleted in order to obtain the final result. Special attention has to be paid to non-intersecting components. Note that only objects that actually intersect are dealt with by now. In constellations where input objects do not intersect, or are composed of several disjunct components, which do not all intersect, we have to take additional measures to evaluate their nesting and decide which



ones to keep and which ones to discard. We apply a ray-shooting approach [Hav99] making use of our octree as spatial search structure [FP02] to speed up this process.

## 6. Outer hull construction & front tracking

Various methods for tracking the surface of fluids and other deformable materials explicitly by a polygon mesh have been presented in the past [GGL*95, TBE*01, Jia07]. Opposed to approaches that implicitly capture such surfaces, e.g. level set methods [Set99], they do not suffer from aliasing caused by an underlying discrete grid and hence are able to handle structures of much higher detail. However, they introduce the need for an explicit handling of changes in surface topology. Some approaches basing on imperfect heuristics with fall-back strategies for the case of failure have been presented [BLS03, LT05, BB09]. Other methods make use of implicit, volumetric representations and (partially) resample the surface [WTGT09, DFG*06, Mül09]. Unfortunately, this approach in general removes any surface detail in the proximity of the topological change or – depending on the geometry – even farther away, and small material parts might get lost. Making use of our topology modifying outer hull construction that is presented in the next section, we are able to handle the topological events that occur during such simulations robustly and exactly without altering surface geometry.

Besides this dynamic setting, there are closely related tasks concerning mesh repair in static configurations. Our technique can, for instance, be used to construct a geometrically meaningful manifold mesh from a mesh that contains self-intersections and superfluous internal structures. In this context, the application of some concept of an outer hull (in a volumetric setting) has been proposed by several authors [BPK05, NT99, ABA02].

### 6.1. Basic concept

The fundamental difference between these repair and front tracking tasks and the Boolean operations is the fact that intersections might not only happen between different objects, but also within one object. Hence we have to exchange the method for determining the inside and outside of our final object – it is not defined by some Boolean expression.

We define the inside of the material to be the volume that is not reachable from infinity without crossing the surface, the outside to be the volume that is reachable. We call the boundary between the outside and inside volumes defined in this way the *outer hull* of the input mesh. In particular, we define this reachability sensitive to surface orientation, i.e. crossings of surface parts that are back-facing with respect to the direction of crossing are ignored. This is important in order to allow meshes to split into disconnected parts and to permit the emergence of new “tunnels” through the material. The defined boundary is then referred to as *orientation-sensitive outer hull*. Figure 5 illustrates these concepts. Note that they are inherently oblivious to internal voids, which is

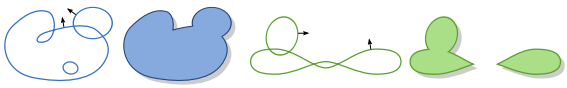


**Figure 5:** *Blue: Illustration of the* outer hull *of objects with (self-)intersections. Surface normals are depicted by arrows. Intersections are resolved by topological changes, effectively merging the parts. Green: The* orientation-sensitive outer hull. *The object splits where it is "thinner than zero".*

familiar and at times intentional in the mentioned fields of application [WTGT09, BPK05].

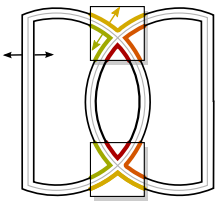
We proceed by first constructing the outer hull and then modifying it into the orientation-sensitive outer hull.

### 6.2. Processing

In this application scenario, self-intersection in meshes have to be handled. This necessitates another intersection test for the octree construction. We employ a variation of *Optimized Spatial Hashing* [THM*03] that uses exact triangle-triangle intersection tests. It proved to outperform all other advanced self-intersection detection schemes we tried, e.g. those building upon spatial search structures [LAM01, MKE03] or those making use of shape regularity [VMT94, Par04].

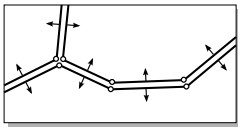
Within a critical cell we construct a BSP from all input polygons. It is partitioned into several components by the intersecting surface (cf. Figure 3), but at this local scale we are not able to determine which components are outside according to our notion. Hence, we cannot determine the desired labeling of the leafs of the BSP. We bypass this by creating local solutions for all alternatives and making a decision after composing them to global solutions. We apply the flood-filling method presented in Chapter 4.2 in order to identify the different components and extract their polygonal boundaries. We orient the polygons such that their normals point to the interior of the respective component.

Before global connectivity is established, we have to additionally create an orientation-reversed copy of the clipped input in the non-critical region. This accounts for the fact that in the local solutions for the critical region each input surface part is also represented twice – by the boundaries of the two adjoining components, as depicted in 2D in the opposite figure. By then establishing connectivity, the parts join up to form the boundaries of the components the global volume is partitioned into by the input mesh. One of these polygonal boundaries is the outer hull. It is one of the boundaries with maximal extent in any direction and can be disambiguated by the orientation of its polygons (outward normals).



In order to turn the outer hull into the orientation-sensitive

outer hull, all we have to do is *superimpose* those component boundaries that have a coincidence with the outer hull where the input is back-facing. Here superimposition simply means adding with mutual cancellation, i.e. coincident but counter-oriented parts are deleted (cf. Figure 6). In this way we effectively account for components that would have been reached in a global flood-filling process that ignores back-facing limits. The fact that allows this operation to be performed very easily is that of *compatible* component boundaries: by construction, each polygon in one of the component boundaries has a perfectly congruent and coincident, but reversely oriented counterpart in one of the other components, as depicted in 2D here. Hence we can select a polygon from the outer hull where the input mesh is back-facing, add the component its counterpart belongs to to the outer hull, and remove all introduced counterpart-pairs. Connectivity information can simply be adapted during this process. By doing this until no further component can be added, we finally yield the orientation-sensitive outer hull. An example of such a hull is depicted in Figure 1.



As already mentioned in the previous chapter, special care has to be taken when the input consists of several independent components. The following measures account for such situations: First, components that lie completely in the interior of another one (determined by ray-shooting as described in Chapter 5) are discarded in the beginning. This removes invisible interior surface parts and for instance handles merge events between components that moved in a way that one of them completely entered another one during one step of simulation. Then we partition the set of remaining components into equivalence classes w.r.t. the transitive hull of relation $R = \{(a,b) \mid \text{components } a \text{ and } b \text{ intersect}\}$ and apply our described processing to each equivalence class separately. The reason for this strategy is the fact that each equivalence class has its own outer hull component, and its detection out of the set of extracted components is easier when we know that there is only one.

## 7. Experiments and results

We evaluate our method with respect to precision, robustness and efficiency by performing suitable experiments on a system with Intel Core i7 2.67GHz CPU and 6GB RAM.

In order to provide evidence for the exactness of our algorithm and the involved conversions, we converted 25 different models with complexities ranging from 36 to 1M faces into plane-based representation and then into BSP-representation, followed by an extraction and conversion back to a polygon mesh. Afterwards, we compared input to output by calculating the Hausdorff-distance from the set of input vertices to the set of output vertices. Note that in this test setting theoretically no rounding should be involved in the final computation of the vertices that correspond to input vertices. In all cases the distance has indeed been zero.

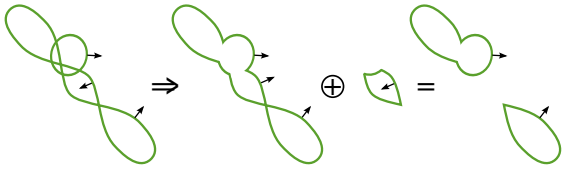


**Figure 6:** *Illustration of an input object (left) with self-intersections and its outer hull (middle left). By* superimposing *those extracted components (middle right), that coincide with the hull where the input is back-facing w.r.t. the hull, we obtain the orientation-sensitive outer hull (right).*

We check the robustness of our approach by processing object constellations that are hard to handle in that they often lead to failures in tools for Boolean operations: for several objects we shifted a copy as slightly as possible, i.e. increased one coordinate component of each vertex to the next representable value. We then computed the difference between these two objects and always obtained a closed manifold output, free from topological error, whose distance to the reference computed by CGAL from the same input was sub-precision, i.e. zero when rounded to input precision.

Our prototypical implementation still leaves room for optimization. We used only single-stage filtered adaptive predicates. Since a major part of the execution time of our algorithm is spent with evaluating geometric predicates, applying multi-stage filtering [She97] would probably increase performance. However, comparing the performance of our method for Boolean operations with CGAL, the current best practice in exact, robust Booleans, our prototype already shows considerably higher efficiency in terms of time and space. We compute Boolean operations on a series of large meshes representing the same object with increasing polygon count. We take the IPHIGENIE model depicted in Figure 7 and perform a union operation with a copy rotated by varying angles around the center point. We use the same input meshes for both algorithms and compute our final output coordinates to full precision. Results are presented in Figure 7. We clearly see how our method benefits from the fact that the region affected by intersections decreases as the angle increases, due to our localization scheme. We also see that the advantage over CGAL increases as the input complexity grows: the runtime of CGAL's implementation seems to be nearly linear, while our method shows almost $O(\sqrt{n})$ behavior as explained in Section 4.1. Of course, the octree construction has higher complexity but this has no significant influence at the lower input resolutions. Regarding the space efficiency, we were unable to run tests with CGAL on input with more than 200K faces, since the internal representation already consumes about 5.3GB of main memory, while our method required less than 300MB. All in all, our method performed 2.5 to 13 times faster than CGAL, and judging from extrapolation it would have been about 25 times if the more complex inputs could have been processed.

| | # faces | 25K | 50K | 100K | 200K | 400K | 800K | 1600K | 3200K |
|---|---|---|---|---|---|---|---|---|---|
| 30° | # cells | 1.7K | 2.4K | 3.3K | 4.7K | 7.2K | 8.6K | 12.9K | 16.5K |
| | LBSP | 6.9s | 10.4s | 14.0s | 19.1s | 27.4s | 40.0s | 59.2s | 95.8s |
| | CGAL | 17.6s | 33.5s | 81.3s | 113.9s | – | – | – | – |
| 60° | # cells | 0.9K | 1.3K | 1.8K | 2.5K | 3.4K | 4.8K | 7.2K | 9.1K |
| | LBSP | 4.2s | 5.8s | 8.4s | 11.8s | 16.7s | 25.5s | 38.1s | 65.3s |
| | CGAL | 16.9s | 31.4s | 60.5s | 117.0s | – | – | – | – |
| 90° | # cells | 0.7K | 1.0K | 1.3K | 1.8K | 2.5K | 3.4K | 5.2K | 6.5K |
| | LBSP | 3.3s | 4.6s | 6.4s | 8.9s | 13.6s | 21.0s | 32.8s | 56.6s |
| | CGAL | 16.1s | 31.1s | 60.6s | 114.0s | – | – | – | – |

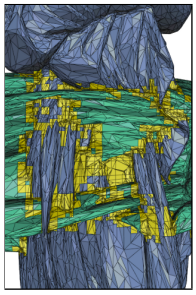


**Figure 7:** *Results of the* IPHIGENIE *experiment for different rotation angles, executed by our localized BSP method (LBSP) and CGAL. We operated on input complexities ranging from 25K to 3200K faces, where CGAL was unable to process input* $> 200K$ *due to memory requirements beyond 10GB. The number of critical cells shows* $O(\sqrt{n})$ *behavior as expected. Times are given for the whole processing pipeline from standard polygon mesh input to polygon mesh output, including all required conversions. On the right, a close-up of the* $50K–90^\circ$ *instance after processing is depicted. The critical region is colored yellow.*

In order to show the applicability of the outer hull concept to mesh repair tasks, we computed the outer hull of objects containing self-intersections, introduced during modeling by bending parts of the model. Figure 8 shows two original models and their repaired versions. These have been verified to be completely free of self-intersections by pairwise triangle intersection tests using exact arithmetic. The Hausdorff-distance to the original meshes always was sub-precision, i.e. zero when measurements are taken with the used input precision. Processing took 1.2s for the COW model (6K faces) and 1.9s for the WOMAN model (12K faces).

Regarding the application of our orientation-sensitive outer hull approach to the problem of topology changes during explicit front tracking, we performed some preliminary tests that indicate suitability for this task even with finely tesselated meshes. Since meshes that are free from self-intersections do not need any processing by our method – except for the determination of this fact – runtime is low for such time-steps in the animation: about 100ms for a test mesh with 15K faces, about 300ms for 50K faces, or about 1600ms for 400K faces. Only in the event of merges or splits in the surface, topology has to be adapted. The whole mesh post-processing for a step of animation then for instance took 5.2s for a 400K mesh that contained nine small self-intersections that had to be resolved, An in-depth examination of this field of application will be part of our future research.

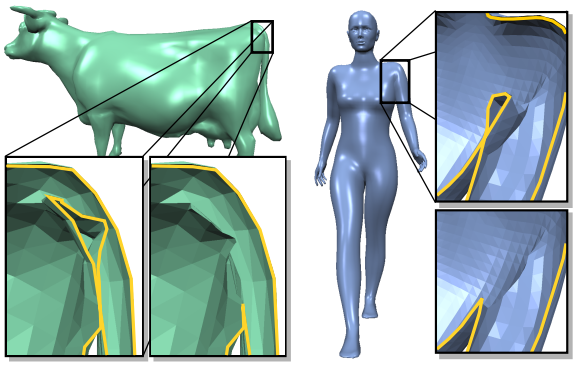


**Figure 8:** *Models* COW *and* WOMAN *containing self-intersections. This is visible in the magnified cross-sections (cut curves highlighted yellow). By applying our outer hull method, interior parts are removed and intersections resolved, as can be seen in the right and lower magnifications.*

Degenerate contact situations that result in non-manifold edges or vertices are handled by our scheme. In case a manifold output mesh is desired, these can be split up into simple edges and vertices – either topologically merging the solids in contact or separating them.

## 8. Conclusion

We presented a novel scheme for the execution of topological changes in polygonal meshes. Using plane-based representations, BSP techniques and a localization scheme we are able to perform operations like the evaluation of Boolean expressions over polyhedra or the construction of outer hulls of self-intersecting meshes exactly, robustly, and efficient in time and space. We showed how our technique can be applied to explicit front-tracking of deformable material, evaluated the efficiency of our approach and demonstrated its correctness and completeness on challenging examples.

In the area of solid modeling, we are currently exploring the possibility of computing Minkowski sums of polygonal objects robustly and exactly using the paradigm of plane-based geometry processing in conjunction with BSP techniques. While arithmetic operations cannot be restricted to predicates in this context, the required constructions do not cause increases in the precision requirements, which promises to allow for an efficient processing.

Another valuable direction for future research is finding a remedy for the inability of handling inner voids that currently restricts the utility of the outer hull method in particular areas of application.

**Acknowledgements** The WOMAN model is courtesy of MIRALab, University of Geneva, and the COW model has been obtained from the AIM@SHAPE repository.

## References

[ABA02] ANDÚJAR C., BRUNET P., AYALA D.: Topology-reducing surface simplification using a discrete solid representation. *ACM Trans. Graph. 21*, 2 (2002), 88–105.

[ABJN85] AYALA D., BRUNET P., JUAN R., NAVAZO I.: Object representation by means of nonminimal division quadtrees and octrees. *ACM Trans. Graph. 4*, 1 (1985), 41–59.

[BB09] BROCHU T., BRIDSON R.: Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing 31*, 4 (2009), 2472–2493.

[BF09] BERNSTEIN G., FUSSELL D.: Fast, exact, linear booleans. *Comput. Graph. Forum 28*, 5 (2009), 1269–1278.

[BK05] BISCHOFF S., KOBBELT L.: Structure preserving cad model repair. *Comput. Graph. Forum 24*, 3 (2005), 527–536.

[BLS03] BREDNO J., LEHMANN T. M., SPITZER K.: A general discrete contour model in two, three, and four dimensions for topology-adaptive multichannel segmentation. *IEEE Trans. Pattern Anal. Mach. Intell. 25*, 5 (2003), 550–563.

[BPK05] BISCHOFF S., PAVIC D., KOBBELT L.: Automatic restoration of polygon models. *ACM Trans. Graph. 24*, 4 (2005), 1332–1352.

[DFG*06] DU J., FIX B., GLIMM J., JIA X., LI X., LI Y., WU L.: A simple package for front tracking. *J. Comput. Phys. 213*, 2 (2006), 613–628.

[For97] FORTUNE S.: Polyhedral modelling with multiprecision integer arithmetic. *CAD 29*, 2 (1997), 123–133.

[FP02] FRISKEN S. F., PERRY R. N.: Simple and efficient traversal methods for quadtrees and octrees. *Journal of Graphics Tools 7*, 7 (2002), 2002.

[GGL*95] GLIMM J., GROVE J. W., LI X. L., SHYUE K.-M., ZENG Y., ZHANG Q.: Three dimensional front tracking. *SIAM J. Sci. Comp 19* (1995), 703–727.

[GHH*03] GRANADOS M., HACHENBERGER P., HERT S., KETTNER L., MEHLHORN K., SEEL M.: Boolean operations on 3d selective nef complexes. In *ESA* (2003), pp. 654–666.

[Hav99] HAVRAN V.: A summary of octree ray traversal algorithms. *Ray Tracing News 12*, 2 (Dec. 1999).

[Jia07] JIAO X.: Face offsetting: A unified approach for explicit moving interfaces. *J. Comput. Phys. 220*, 2 (2007), 612–625.

[JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J. D.: Dual contouring of hermite data. In *SIGGRAPH* (2002), pp. 339–346.

[KCS98] KOBBELT L., CAMPAGNA S., SEIDEL H.-P.: A general framework for mesh decimation. In *Graphics Interface* (1998), pp. 43–50.

[LAM01] LARSSON T., AKENINE-MÖLLER T.: Collision detection for continuously deforming bodies, 2001.

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH* (1987), pp. 163–169.

[LPY05] LI C., PION S., YAP C.-K.: Recent progress in exact geometric computation. *J. Log. Algebr. Program. 64*, 1 (2005), 85–111.

[LT05] LACHAUD J.-O., TATON B.: Deformable model with a complexity independent from image resolution. *Computer Vision and Image Understanding 99*, 3 (2005), 453–475.

[LTH86] LAIDLAW D. H., TRUMBORE W. B., HUGHES J. F.: Constructive solid geometry for polyhedral objects. *SIGGRAPH Comput. Graph. 20*, 4 (1986), 161–170.

[MKE03] MEZGER J., KIMMERLE S., ETZMUSS O.: Hierarchical techniques in collision detection for cloth animation. In *WSCG* (2003).

[Mül09] MÜLLER M.: Fast and robust tracking of fluid surfaces. In *SCA '09: Proc. Symp. Comput. Animation* (New York, NY, USA, 2009), pp. 237–245.

[NAT90] NAYLOR B., AMANATIDES J., THIBAULT W.: Merging bsp trees yields polyhedral set operations. In *SIGGRAPH Comput. Graph.* (New York, NY, USA, 1990), ACM, pp. 115–124.

[Nef78] NEF W.: *Beiträge zur Theorie der Polyeder*. Herbert Lang Verlag, Bern, 1978.

[NT99] NOORUDDIN F. S., TURK G.: *Simplification and repair of polygonal models using volumetric techniques*. Technical Report GITGVU -99-37, Georgia Institute of Technology, 1999.

[Par04] PARK S. C.: Triangular mesh intersection. *Vis. Comput. 20*, 7 (2004), 448–456.

[PCK09] PAVIĆ D., CAMPEN M., KOBBELT L.: Hybrid booleans. *Computer Graphics Forum, to appear* (2009).

[Pri91] PRIEST D. M.: Algorithms for arbitrary precision floating point arithmetic. In *Symp. Comput. Arithm.* (1991), pp. 132–145.

[RV85] REQUICHA A. A. G., VOELCKER H. B.: Boolean operations in solid modeling: Boundary evaluation and merging algorithms. In *IEEE Proc. 73, 1* (1985), pp. 30–44.

[Set99] SETHIAN J. A.: *Level Set Methods and Fast Marching Methods*. Cambridge University Press, June 1999.

[She97] SHEWCHUK J. R.: Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete & Computational Geometry 18*, 3 (Oct. 1997), 305–363.

[SI89] SUGIHARA K., IRI M.: A solid modelling system free from topological inconsistency. *J. Inf. Process.. 12*, 4 (1989), 380–393.

[TBE*01] TRYGGVASON G., BUNNER B., ESMAEELI A., JURIC D., AL-RAWAHI N., TAUBER W., HAN J., NAS S., JAN Y.-J.: A front-tracking method for the computations of multiphase flow. *J. Comput. Phys. 169*, 2 (2001), 708–759.

[Thi87] THIBAULT W. C.: *Application of binary space partitioning trees to geometric modeling and ray-tracing*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, 1987.

[THM*03] TESCHNER M., HEIDELBERGER B., MÜLLER M., POMERANTES D., GROSS M. H.: Optimized spatial hashing for collision detection of deformable objects. In *VMV* (2003), pp. 47–54.

[TN87] THIBAULT W. C., NAYLOR B. F.: Set operations on polyhedra using binary space partitioning trees. *SIGGRAPH Comput. Graph. 21*, 4 (1987), 153–162.

[VKSM04] VARADHAN G., KRISHNAN S., SRIRAM T. V. N., MANOCHA D.: Topology preserving surface extraction using adaptive subdivision. In *SGP* (2004), pp. 241–250.

[VKZM06] VARADHAN G., KRISHNAN S., ZHANG L., MANOCHA D.: Reliable implicit surface polygonization using visibility mapping. In *Symp. Geom. Proc.* (2006), pp. 211–221.

[VMT94] VOLINO P., MAGNENAT-THALMANN N.: Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Comput. Graph. Forum 13*, 3 (1994), 155–166.

[WTGT09] WOJTAN C., THÜREY N., GROSS M., TURK G.: Deforming meshes that split and merge. *ACM Trans. Graph. 28*, 3 (2009), 1–10.