On the Convexification of Unstructured Grids From A Scientific Visualization Perspective

João L.D. Comba[1], Joseph S.B. Mitchell[2], and Cláudio T. Silva[3]

[1] Federal University of Rio Grande do Sul (UFRGS) comba@inf.ufrgs.br
[2] Stony Brook University jsbm@ams.sunysb.edu
[3] University of Utah csilva@cs.utah.edu

Summary. Unstructured grids are extensively used in modern computational solvers and, thus, play an important role in scientific visualization. They come in many different types. One of the most general types are non-convex meshes, which may contain voids and cavities. The lack of convexity presents a problem for several algorithms, often causing performance issues.

One way around the complexity of non-convex methods is to convert them into convex ones for visualization purposes. This idea was originally proposed by Peter Williams in his seminal paper on visibility ordering. He proposed to fill the volume between the convex hull of the original mesh, and its boundary with "imaginary" cells. In his paper, he sketches two algorithms for potentially performing this operation, but stops short of implementing them.

This paper discusses the convexification problem and surveys the relevant literature. We hope it is useful for researchers interested in the visualization of unstructured grids.

1 Introduction

The most common input data type in Volume Visualization is a *regular (Cartesian) grid* of *voxels*. Given a general scalar field in $\Re^3$, one can use a regular grid of voxels to represent the field by regularly sampling the function at grid points $(\lambda i, \lambda j, \lambda k)$, for integers i, j, k, and some scale factor $\lambda \in \Re$, thereby creating a regular grid of voxels. However, a serious drawback of this approach arises when the scalar field is *disparate*, having nonuniform resolution with some large regions of space having very little field variation, and other very small regions of space having very high field variation. In such cases, which often arise in computational fluid dynamics and partial differential equation solvers, the use of a regular grid is infeasible since the voxel size must be small enough to model the smallest "features" in the field. Instead, *irregular grids* (or *meshes*), having cells that are not necessarily uniform cubes, have been proposed as an effective means of representing disparate field data.

Irregular-grid data comes in several different formats [37]. One very common format has been *curvilinear grids*, which are *structured* grids in computational space that have been "warped" in physical space, while preserving the same topological structure (connectivity) of a regular grid. However, with the introduction of new methods for generating higher quality adaptive meshes, it is becoming increasingly common to consider more general *unstructured* (non-curvilinear) irregular grids, in which there is no implicit connectivity information. Furthermore, in some applications *disconnected* grids arise.

Preliminaries

We begin with some basic definitions. A *polyhedron* is a closed subset of $\Re^3$ whose boundary consists of a finite collection of convex polygons (*2-faces*, or *facets*) whose union is a connected 2-manifold. The *edges* (*1-faces*) and *vertices* (*0-faces*) of a polyhedron are simply the edges and vertices of the polygonal facets. A bounded convex polyhedron is called a *polytope*. A polytope having exactly four vertices (and four triangular facets) is called a *simplex* (*tetrahedron*). A finite set S of polyhedra forms a *mesh* (or an *unstructured grid*) if the intersection of any two polyhedra from S is either empty, a single common vertex, a single common edge, or a single common facet of the two polyhedra; such a set S is said to form a *cell complex*. The polyhedra of a mesh are referred to as the *cells* (or *3-faces*). We say that cell C is *adjacent* to cell C' if C and C' share a common facet. The adjacency relation is a binary relation on elements of S that defines an *adjacency graph*.

A facet that is incident on only one cell is called a *boundary facet*. A *boundary cell* is any cell having a boundary facet. The union of all boundary facets is the *boundary* of the mesh. If the boundary of a mesh S is also the boundary of the convex hull of S, then S is called a *convex* mesh; otherwise, it is called a *non-convex* mesh. If the cells are all simplicies, then we say that the mesh is *simplicial*.

The input to our problem will be a given mesh S. We let c denote the number of connected components of S. If $c = 1$, the mesh is *connected*; otherwise, the mesh is *disconnected*. We let n denote the total number of edges of all polyhedral cells in the mesh. Then, there are $O(n)$ vertices, edges, facets, and cells.

We use a coordinate system in which the viewing direction is in the $-z$ direction, and the image plane is the (x, y) plane. We let ρ_u denote the ray from the viewpoint v through the point u.

We say that cells C and C' are *immediate neighbors* with respect to viewpoint v if there exists a ray ρ from v that intersects C and C', and no other cell $C'' \in S$ has a nonempty intersection $C'' \cap \rho$ that appears in between the segments $C \cap \rho$ and $C' \cap \rho$ along ρ. Note that if C and C' are adjacent, then they are necessarily immediate neighbors with respect to very viewpoint v not in the plane of the shared facet. Further, in a convex mesh, the *only* pairs of cells that are immediate neighbors are those that are adjacent.

A *visibility ordering* (or *depth ordering*), $<_v$, of a mesh S from a given viewpoint, $v \in \Re^3$ is a total (linear) order on S such that if cell $C \in S$ visually obstructs cell $C' \in S$, partially or completely, then C' precedes C in the ordering: $C' <_v C$. A visibility ordering is a linear extension of the binary *behind* relation, "$<$", in which cell C is *behind* cell C' (written $C < C'$) if and only if C and C' are immediate neighbors and C' at least partially obstructs C; *i.e.*, if and only if there exists a ray ρ from the viewpoint v such that $\rho \cap C \neq \emptyset$, $\rho \cap C' \neq \emptyset$, $\rho \cap C'$ appears in between v and $\rho \cap C$ along ρ, and no other cell C'' intersects ρ at a point between $\rho \cap C$ and $\rho \cap C'$. A visibility ordering can be obtained in linear time (by topological sorting) from the behind relation, $(S, <)$, provided that the directed graph on the set of nodes S defined by $(S, <)$ is acyclic. If the behind relation induces a directed cycle, then no visibility ordering exists. Certain types of meshes, (e.g., Delaunay triangulations [16]) are known to have a visibility ordering from any viewpoint, *i.e.*, they do not have cycles, and thus can be called *acyclic meshes*.

Spatial Decompositions

There is a rich literature in the computational geometry community on spatial decompositions. See Nielson, Hagen and Müller [25] for an overview of their importance in the context of visualization applications.

Spatial decomposition is an essential tool in finite element analysis and geometric modeling. Applications require high-quality mesh generation, in which the goal is to triangulate domains with elements that are "nice" in some well-defined sense (e.g., triangulations having no large angle [3]). See the recent surveys of Bern and Eppstein [4], Bern and Plassmann [5], and Bern [2], and the book of Edelsbrunner [16], for a comprehensive overview of the literature.

A problem extensively studied in the early years of computational geometry was the polygon triangulation problem, in which the goal was to decompose a simple polygon, or a polygon with holes, into triangles. A milestone result in two-dimensional triangulations was the discovery by Chazelle [6] of a linear-time algorithm for triangulating a simple polygon. Optimization problems related to decompositions of polygons into convex pieces have been studied in many variations; see Chazelle and Dobkin [7] and the survey of Keil [21].

In three or more dimensions, decomposition of polyhedral domains into "triangles" (tetrahedra) is substantially more complex. Ruppert and Seidel [27] have shown that it is NP-complete to decide if a (non-convex) polyhedron can be tetrahedralized without the addition of Steiner points. Chazelle and Palios [10] show that a (non-convex) polyhedron having n vertices and r reflex edges can always be triangulated (with the addition of Steiner points) in time $O(nr+r^2 \log r)$ using $O(n+r^2)$ tetrahedra (which is worst-case optimal, since some polyhedra require $\Omega(n+r^2)$ tetrahedra in any triangulation).

A *regular triangulation* in dimension d is the vertical projection of the "lower" side of a convex polytope in one higher dimension. The most widely studied regular triangulation is the Delaunay triangulation of a point set, which is the projection of the downward-facing facets of the convex hull of the lifted images of the input points onto the paraboloid in one higher dimension. An alternative characterization of a Delaunay triangulation is that the (hyper)sphere determined by the vertices of each triangle (simplex) of a Delaunay triangulation is "site-free," not containing input points in its interior. See Edelsbrunner [15], as well as the book of Okabe, Boots, and Sugihara [26] and the recent survey articles of Fortune [17]

Chazelle et al. [8] have examined how selectively adding points to an input set in three dimensions results in the worst-case size of the Delaunay triangulation being provably subquadratic in the input size, even though the worst-case size of a Delaunay triangulation of n points in space is $\Theta(n^2)$.

The meshes we study here are decompositions of polyhedral domains and piecewise-linear complexes, in which the decomposition is required to respect the facets of the input. A *constrained Delaunay triangulation* is a variation of a Delaunay triangulation that is constrained to respect the input shape, while being, in some sense, "as Delaunay as possible." Such decompositions have desirable properties, favoring more regular tetrahedra over "skinny" tetrahedra. This makes them particularly appealing for interpolation, visualization, and finite element methods. Two-dimensional constrained Delaunay triangulations have been studied by, e.g., Chew [11], De Floriani and Puppo [14], and Seidel [29]. More recently, three-dimensional constrained Delaunay triangulations have been studied for their use in mesh generation; see the surveys mentioned above ([2,4,5,16]), as well as Weatherill and Hassan [39]. Shewchuk [30–34] has developed efficient methods for three-dimensional constrained Delaunay triangulations, including, most recently [34], provable techniques of inserting constraints and performing "flips" (local modifications to the mesh) to construct constrained Delaunay and regular triangulations incrementally.

Exploiting Mesh Properties

Meshes that conform to properties such as "convexity" and "acyclicity" are quite special, since they simplify the algorithms that work with them. Here are three instances of visualization algorithms that exploit different properties of meshes:

- A classic technique for hardware-based rendering of unstructured meshes couples the Shirley-Tuchman technique for rendering a single tetrahedron [35] with Williams' MPVO cell-sorting algorithm [41]. For the case of acyclic convex meshes, this is a powerful combination that leads to a linear-time algorithm that is provably correct, *i.e.*, one is guaranteed to get the right picture.[4] When the mesh is not convex or contains cycles, MPVO requires modifications that complicate the algorithm and its implementation and lead to slower rendering times [13,22,36].
- A recent hardware-based ray casting technique for unstructured grids has been proposed by Weiler et al [40]. This is essentially a hardware-based implementation of the algorithm of Garrity [19]. Strictly speaking, this technique only works for convex meshes. Due to the constraints of the hardware, instead of modifying the rendering algorithm, the authors employ a process of "convexification", originally proposed by Williams [41], to handle general cells.
- The complexities of the simplification of unstructured grids has led some researchers to employ a convexification approach. As shown in Kraus and Ertl [23], this greatly simplifies the simplification algorithm, since it becomes much simpler to handle the simplification of the boundary of the mesh. Otherwise, expen-

[4]The rendering technique of Shirley and Tuchman [35] requires certain modifications as proposed in Stein et al [38].

sive global operations are necessary to guarantee that the simplified mesh does not suffer from self intersections.

The “convexification” concept as proposed by Williams [41] is the process of turning a non-convex mesh into a convex one. The basic idea is that this process can be performed by adding a set of non-overlapping cells that fill up any holes or non-convex regions up to the bounding box of the original mesh. Also, Williams proposes that all the additional cells be marked “imaginary”. This is exactly the concept that is used in the works of Weiler et al [40] and Kraus and Ertl [23]. In [23, 40], the non-convex meshes were *manually* modified to be convex by the careful addition of cells. This approach is not scalable to larger and more complex data.

In this paper, we discuss the general problem of convexification. We start by reviewing Williams’ work, and discuss a number of issues. Then, we talk about two techniques for achieving convexification: techniques based on constrained and conforming Delaunay tetrahedralization, and techniques based on the use of a binary space partition (BSP). Finally, we conclude the paper with some observations and open questions. One of the goals of this paper is to help researchers be able to choose among tools and options for convexification solutions.

2 Williams’ Convexification Framework

In his seminal paper [41] on techniques for computing visibility orderings for meshes, Williams discusses the problem of handling non-convex meshes (Section 9). (Also related is Section 8, which contains a discussion of cycles and the use of Delaunay triangulations.) After explaining some challenges of using his visibility sorting algorithm on non-convex meshes, Williams says:

> “Therefore, an important area of research is to find ways to convert non-convex meshes into convex meshes, so that the regular MPVO algorithm can be used.”

Williams proposes two solution approaches to the problem; each relies on “treating the voids and cavities as ‘imaginary’ cells in the mesh.” Basically, he proposes that such non-convex regions could be either **triangulated** or **decomposed** into convex pieces, and their parts marked as imaginary cells for the purpose of rendering. Implementing this “simple idea” is actually not easy. In fact, after discussing this general approach, Williams talks about some of the challenges, and finishes the section with the following remark:

> “The implementation of the preprocessing methods, described in this section, for converting a non-convex mesh into a convex mesh could take a very significant amount of time; they are by no means trivial. The implementation of a 3D conformed Delaunay triangulation is still a research question at this time.”

In fact, Williams does not provide an implementation of any of the two proposed convexification algorithms. Instead, he developed a variant of MPVO that works on non-convex meshes at the expense of not being guaranteed to generate correct visibility orders.

The first convexification technique that Williams proposes is based on triangulating the data using a conforming Delaunay triangulation. The idea here is to keep adding more points to the dataset until the original triangulation becomes a Delaunay triangulation. This is discussed in more details in the next section.

The second technique Williams sketches is based on the idea of applying a decomposition algorithm to each of the non-convex polyhedra that constitute the set $CH(S) \setminus S$, which is the set difference between the convex hull of the mesh and the mesh itself. In general, $CH(S) \setminus S$ is a union of highly non-convex polyhedra of complex topology. Each connected component of $CH(S) \setminus S$ is a non-convex polyhedron that can be decomposed into convex polyhedra (e.g., tetrahedra) using, for example, the algorithm of Chazelle and Palios [10], which adds certain new vertices (Steiner points), whose number depends on the number of “reflex” edges of the polyhedron. In general, non-convex polyhedra require the addition of Steiner points in order to decompose them; in fact, it is NP-complete to decide if a polyhedron can be tetrahedralized without the addition of Steiner points [27].

2.1 Issues

Achieving Peter Williams’s vision of a simple convexification algorithm is much harder than it appears at first. The problem is peculiar since we start with an existing 3D mesh (likely to be a tetrahedralization) that contains not only vertices, edges, and triangles, but also volumetric cells, which need to be respected. Furthermore, the mesh is not guaranteed to respect global geometric criteria (e.g., of being Delaunay). Most techniques need to modify the original mesh in some way. The goal is to “disturb” it as little as possible, preserving most of its original properties.

In particular, several issues need to be considered:

Preserving acyclicity. Even if the original mesh has no cycles, the convexification process can potentially cause the resulting convex mesh to contain cycles. Certain techniques, such as constructing a conforming Delaunay tetrahedralization, are guaranteed to generate a cycle-free mesh. Ideally, the convexification procedure will not create new cycles in the mesh.

Output size. For the convexification technique to be useful the number of cells added by the algorithm needs to be kept as small as possible. Ideally, there is a provable bound on the number of cells as well as experimental evidence that for typical input meshes, the size of the output mesh is not much larger than the input mesh (*i.e.*, the set of additional cells is small).

Computational and memory complexity. Other important factors are the processing time and the amount of memory used in the algorithm. In order to be practical on the meshes that arise in computational experiments (having on the order of several thousand to a few million cells), convexification algorithms must run in near-linear time, in practice.

Boundary and interior preservation. Ideally, the convexification procedure adds cells only “outside” of the original mesh. Furthermore, the newly created cells should exactly match the original boundary of the mesh. In general, this is not feasible without subdividing or modifying the original cells in some way (e.g., to break cycles, or to add extra geometry in order to respect the Delaunay empty-circumsphere condition). Some techniques will only need to modify the cells that are at or near the original boundary while others might need to perform more global modifications that go all the way “inside” the original mesh. One needs to be careful when making such modifications because of issues related to interpolating the original data values in the mesh. Otherwise, the visualization algorithm may generate incorrect pictures leading to wrong comprehension.

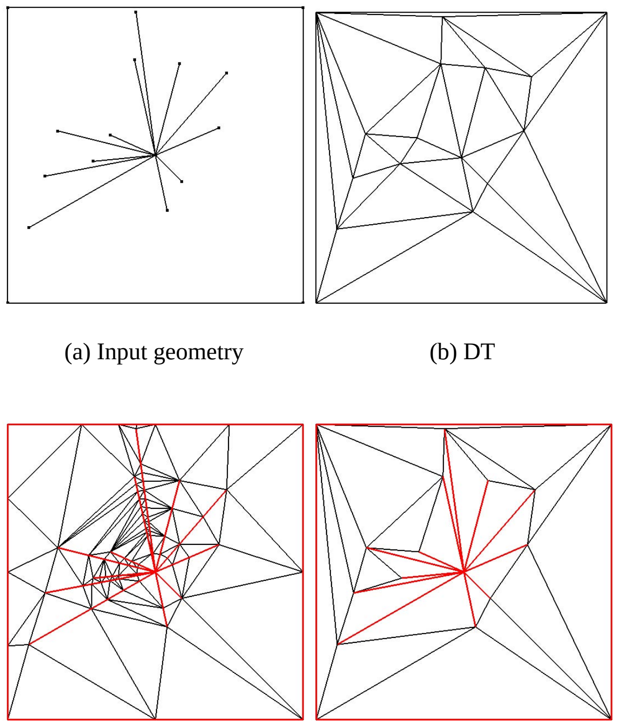
Robustness and degeneracy handling. It is very important for the convexification algorithms to handle real data. Large scientific datasets often use floating-point precision for specifying vertices, and are likely to have a number of degeneracies. For instance, these datasets are likely to have many vertices (sample points) that are coplanar, or that lie on a common cylinder or sphere, etc., since the underlying physical model may have such features.

3 Delaunay-Based Techniques

Delaunay triangulations and Delaunay tetrahedralizations (DT) are very well known and studied geometric entities (see, e.g., [16, Chapter 5]). A basic property that characterizes this geometric structure is the fact that a tetrahedron belongs to the DT of a point set if the circumsphere passing through the four vertices is empty, meaning no other point lies inside the circumsphere. Under some non-degeneracy conditions (no 5 points co-spherical), this property completely characterizes DTs and the DT is unique.

Part of the appeal of Delaunay tetrahedralizations (see Figure 1(b)) is the relative ease of computing the tetrahedralizations. As a well-studied structure, often used in mesh generation, standard codes are readily available that compute the DT. The practical need of forcing certain faces to be part of the tetrahedralizations led to the development of two main approaches: *conforming* Delaunay tetrahedralizations and *constrained* Delaunay tetrahedralizations. Here, we only give a high-level discussion on the intuition behind these ideas; for details see, e.g., [32].

Given a set of faces $\{f_i\}$ (Figure 1(a)) that need to be included in a DT, the idea behind *conforming* Delaunay tetrahedralizations (Figure 1(c)) is to add points to the original input set in order that the DT of the new point set (consisting of the original points *plus* the newly added points) is such that each face f_i can be expressed as the union of a collection of faces of the DT. The newly added points are often called *Steiner* points. A challenge in computing a conforming DT is minimizing the number of Steiner points and avoiding the generation of very small tetrahedra. While techniques for computing the traditional DT of point sites are well known, and reliable code exists, conforming DT algorithms are still in active development



(a) Input geometry (b) DT

(c) Conforming DT (d) Constrained DT

Fig. 1. Different triangulation techniques. (a) The input geometry; (b) the Delaunay triangulation; (c) a conforming Delaunay triangulation with input geometry marked in red – note how the input faces have been broken into multiple pieces; and, (d) the constrained Delaunay triangulation. Images after Shewchuk [31].

[12, 24]. The particular technique for adding Steiner points affects the termination of the algorithm, and also the number and quality of the added geometry.

For convexification purposes, the conforming DT seems to be a good solution upon first examination, and was one of the original techniques Williams proposed for the problem. One of the main benefits is that since a conforming DT is actually a DT of a larger point set, it must be acyclic. On closer inspection, we can see that conforming DTs have a number of potential weaknesses. First, if the original mesh was not a DT, we may need to completely re-triangulate it. This means that internal structures of the mesh, which may have been carefully designed by the modeler, are potentially lost. In addition, the available experimental evidence [12] suggests that a considerable number of Steiner points may be necessary. Part of the problem is that when a face f_i is pierced by the DT, adding a *local* point p to resolve this issue can potentially result in modifications to the mesh deep within the triangulation, not just in the neighborhood of the point p. Another potential issue with using a conforming DT is the lack of available robust codes for the computation. This is an issue that we expect soon to be resolved, with advances under way in the computational geometry community.

The constrained DT (Figure 1(d)) is a different way to resolve the problem of respecting a given set of faces. While in a conforming DT we only had to make sure that each given face f_i can be represented as the union of a set of faces in the conforming DT, for a constrained DT we insist that each face f_i appears exactly as a face in the tetrahedralization. In order to do this, we must *relax the empty-circumsphere criterion* that characterizes a DT; thus, a constrained DT *is not* (in general) a Delaunay tetrahedralization. The definition of the constrained DT requires a modification to the empty-circumsphere criteria in which we use the input faces $\{f_i\}$ as blockers of visibility and empty-circumsphere tests are computing taking that into account. That is, when performing the tests, we need to *discard* certain geometry when the sphere intersects one (or more) of the input faces. We refer the reader to [32] and [16, Chapter 2] for a detailed discussion. In regions of the mesh "away from" the input faces, a constrained DT looks very much like a standard DT. In fact, they share many of the same properties [30].

Because we are not allowed to add Steiner points when building a constrained DT, they have certain (theoretical) limitations. A particularly intriguing possibility is that it may not be possible to create one because some polyhedra cannot be tetrahedralized without adding Steiner points. (In fact, it is NP-complete to decide if a polyhedron can be tetrahedralized without adding Steiner points [27].) Further, constrained DTs suffer from some of the same issues as conforming DTs in that they may require re-triangulation of large portions of the original mesh. While it may be possible to maintain the Delaunay property on the "internal" portions of the mesh, away from the boundary faces, it is unclear what effect the non-Delaunay portions of the mesh near the boundary have on global properties, such as acyclicity, of the mesh. At this point, some practical issues related to constrained DTs are an area of active investigation [30, 33]; to our knowledge, there is no reliable code available for computing them.

Whether using a conforming or a constrained Delaunay tetrahedralization, the robust computation of the structure for very large point sets is not trivial. Even the best codes take a long time and use substantial amounts of memory. Some of the interesting non-convex meshes we would like to handle have on the order of ten million tetrahedra or more. In the case that the whole dataset needs to be re-triangulated, it is unclear if these techniques would be practical.

4 Direct Convexification Approaches Using BSP-trees

The Binary Space Partitioning tree (BSP-tree) is a geometric structure that has many interesting properties that can be explored in the convexification problem. For instance, the BSP-tree induces a hierarchical partition of the underlying space into

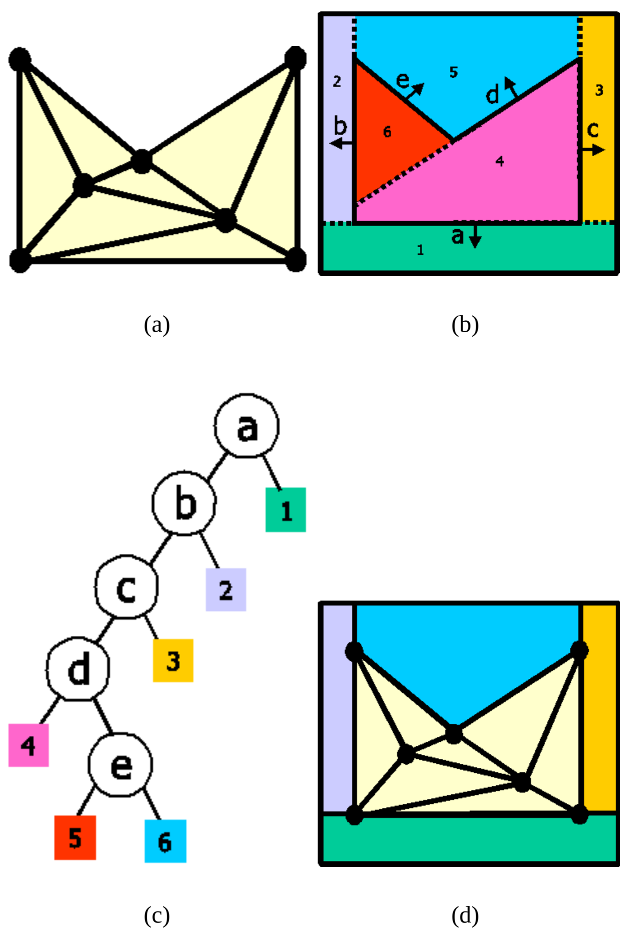


Fig. 2. Using BSP-trees to fill space. (a) The input non-convex mesh; (b) the BSP decomposition using the boundary facets of the input mesh; (c) the corresponding BSP tree; and, (d) the input mesh augmented with BSP cells.

convex cells that allows visibility ordering to be extracted by a priority-search driven by the viewpoint position (in a near-to-far or far-to-near fashion) [18]. In Figure 2 we show how the BSP is used to capture the structure of the empty space.

4.1 Implicit BSP-Tree regions

The visibility-ordering produced by the BSP-tree was explored in [13] to produce missing visibility relations in projective volume rendering. The approach relies on using the BSP-trees to represent the empty space surrounding a non-convex mesh.

Since the empty space $CH(S) \setminus S$ and mesh S have a common intersection at the boundary facets of the mesh S, a BSP-tree was constructed using cuts along the supporting planes of the boundary facets. The construction algorithm starts with the collection of boundary facets of the mesh, and uses an appropriate heuristic to choose a cut at each step to partition the space. The partition process associates each facet with the corresponding half-space (two half-spaces if a facet is split), storing the geometric representations of the boundary facets along the partitioning plane at the nodes of the BSP. The process is recursively repeated at each subtree until a stopping criterion is satisfied.

The resulting BSP-tree partitions the space into convex cells that are either internal or external to the mesh. If a consistent orientation for the boundary facet normals is used, these sets can be distinguished by just checking to which side a given leaf node is with respect to its parent (see Figure 2).

In this approach no effort was made to enumerate explicitly the convex regions corresponding to the empty space in the BSP-tree. However, their implicit representation was used to help provide the missing visibility ordering information in the empty space surrounding the mesh.

Central to this approach is the extraction of visibility relations between interior regions (mesh cells) and exterior regions (the convex cells of the empty space induced by the BSP-tree). The boundary facets of the mesh S are the common boundary between these two types of regions. The approach used in [13] explores one way to obtain the visibility relations, using the visibility ordering produced by the BSP-tree to drive this process. This is done by using a visibility ordering traversal in the BSP-tree with respect to a given viewpoint (in a far-to-near fashion). When an internal node is visited we reach a boundary facet of the mesh. Only facets facing away from the viewing direction impose visibility ordering restrictions, and, for these, two situations can arise, as follows.

The first case happens when the facet stored at the node was not partitioned by the BSP-tree, and therefore is entirely contained in the hyperplane (visible). Visiting an entirely visible boundary facet allows the visibility ordering restriction imposed by this facet into the incident mesh cell to be lifted, which may lead to the inclusion of the cell in the visibility ordering if all restrictions to this cell were lifted.

The second case happens when the boundary facet is partially stored at the BSP node, which indicates that is was partitioned by another cut in the BSP. In this case it is not possible to lift the visibility ordering restriction, since other fragments were not yet reached by the BSP traversal (and therefore not entirely visible). At the moment that the last facet fragment is visited, a cell may be able to be included in the visibility ordering. The solution proposed in [13] uses a counter to accumulate the number of facet fragments created, decrementing this counter for each fragment visited, and lifting the conditions imposed by the fragment when the counter gets to zero.

However, the partition of boundary facets by cuts in the BSP-tree has additional side effects that need to be taken into consideration. In such cases, the BSP traversal is not enough to produce a valid visibility ordering for mesh cells. This happens because the BSP establishes a partial ordering between the convex cells it defines, and a mesh cell that is partitioned by a BSP cut lies in different convex cells of the

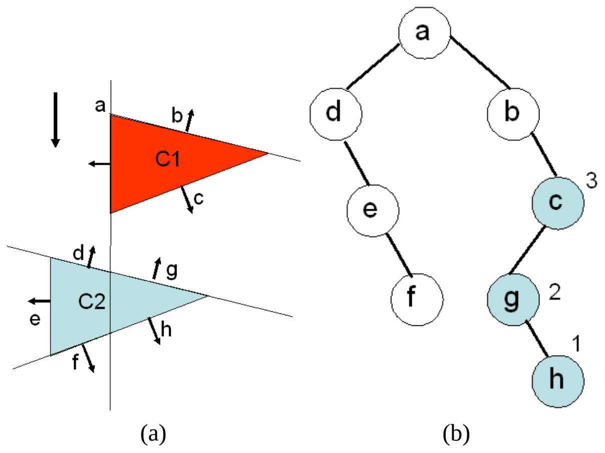


Fig. 3. Partially projected cells. Two cells, (a), and the corresponding BSP-tree, (b). The moment that the traversal reaches node c, cell $C1$ cannot be projected, but has to wait until a partially visited cell $C2$ has been projected.

BSP. In Figure 3 we have an example in which a cell $C1$ cannot enter the visibility ordering because a partially visited cell has facet fragments that were not yet visited. If the ordering to be produced is between the cell $C1$ and the two sub-cells of $C2$, then the BSP ordering suffices.

Cells that have partially visited facets need special treatment; the collection of all such cells at any given time is maintained in a partially projected cells list (PPC). It can be shown that a valid visibility ordering can be produced by the partial orderings provided by mesh adjacencies ($<_{ADJ}$), the ordering produced by the BSP-tree traversal ($<_{BSP}$), and an additional intersection involving cells in the PPC list ($<_{PPC}$). The PPC test increases the complexity of the algorithm; however, it is guaranteed not to generate cycles.

4.2 Explicit BSP-trees regions

The implicit use of the convex regions induced by the BSP-tree in the previous approach required a BSP-traversal to drive the visibility ordering procedure. Another approach is to compute explicitly such convex regions (*filler cells*) and combine them with the mesh to form a convex mesh.

The construction of the BSP-tree uses, as before, partitioning cuts defined by the planes through the boundary facets, except that a different heuristic is used to select the cuts. The algorithm that computes the filler cells needs to perform the following tasks:

- **Computing the geometry of the filler cells:**
  Extracting convex regions associated with nodes of the BSP is straightforward; it can be done in a top-down manner, starting at the root of the tree with a bounding

box that is guaranteed to contain the entire model. In order to obtain the convex regions of the left and right children, the convex region associated with the node is partitioned by the hyperplane. The resulting two convex regions are associated with the children nodes, and the process continues recursively. Figure 4 illustrates this process.

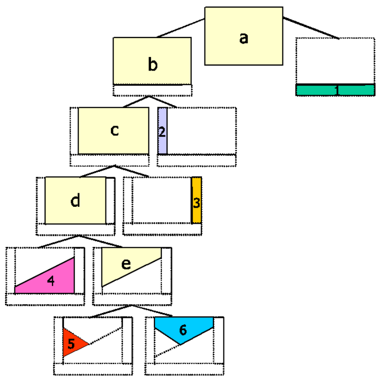


Fig. 4. Geometric computation of filler cells. Illustration of the recursive procedure that applies a partitioning operation to the cell of a node.

- **Computing topological adjacencies between mesh and filler cells:**
  The extraction of topological information in the BSP is not as straightforward. One difficulty that arises is the fact that a cell may be adjacent, by a single facet, to more than one cell. (The cells do not form a cell complex.) The fact that the BSP has arbitrary direction cuts makes the task even harder, requiring an approach that handles numerical degeneracies. The topological adjacencies that need to be computed include filler-to-filler adjacencies, mesh-to-filler and filler-to-mesh adjacencies (see Figure 5).

This convexification approach needs to satisfy the requirements posed before; we briefly discuss them in the context of this approach:
Preserving acyclicity: Although the internal adjacencies of the mesh may not lead to cycles in the visibility ordering, the addition of filler cells may lead to an augmented model (mesh plus filler cells) that contains cycles. Since the mesh is assumed acyclic, cycles do not involve only mesh cells, and from the visibility ordering property of BSP-trees, cycles do not involve only filler cells. Cycles will not involve runs of several filler to mesh cells (filler-mesh), or vice-versa (mesh-filler), since each one of the runs is acyclic. However, cycles can happen in filler-mesh-filler or mesh-filler-mesh cells.

It is still an open problem how to design techniques to avoid or to minimize the appearance of cycles. (See [1, 9] for theoretical results on cutting lines to avoid cycles.) Also, it would be interesting to establish bounds on the number of cells in a cycle.

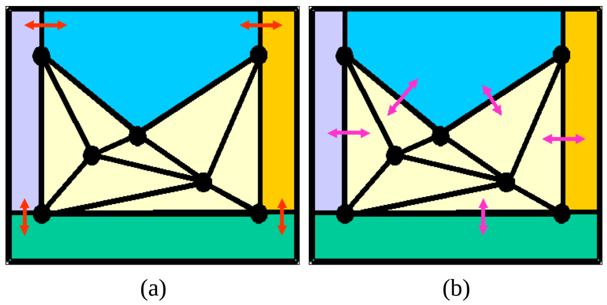


Fig. 5. Topological adjacencies. Filler-to-filler adjacency relations (a) and mesh-to-filler (and vice-versa) relations (b) that need to be computed.

Output size: The number of cells generated is directly related to the size of the BSP-tree. Although the BSP can have worst-case $\Theta(n^2)$ in $\Re^3$, in practice the use of heuristics reduces the typical size of a BSP to linear. Preliminary tests show that one can expect an increase of 5-10% in the number of cells produced.
Computational and memory complexity: The computational cost of the algorithm is proportional to the time required to build a BSP for the boundary faces. The extraction of geometric and topological information of the BSP is proportional to the time to perform a complete traversal of the BSP.
Boundary and interior preservation: The BSP approach naturally preserves the boundary and interior of the mesh, since it only constructs cells that are outside S. This requires that the mesh has the interior well defined, *i.e.*, each connected component of the boundary is a 2-manifold. A consistent orientation of boundary facet normals allows an easy classification of which cells of the BSP are interior or exterior to the mesh.
Robustness and degeneracy handling: The fundamental operations used in the construction of BSP-trees are point-hyperplane classification and the partition of a facet by a hyperplane. The fact that geometric computations rely on only these two operations allows better control of issues of numerical precision and floating point errors. Of course, unless one uses exact geometric computation [28, 42], numerical errors are inevitable; however, several geometric and topological predicates can be checked to verify if a given solution is numerically consistent. The literature on solid modeling has important suggestions on how to do this [20], as in the problem of converting CSG solids to a boundary representation. The possibility of having nearly coplanar boundary facets needs to be treated carefully, since it may require the partition of a facet by a nearly coplanar hyperplane.

The filler cells obtained after a convexification algorithm need to be added to the non-convex mesh, with updates to the topological relationships. In particular, three new types of topological relationships need to be added: filler to filler adjacencies, filler to mesh adjacencies and mesh to filler adjacencies. This problem is complicated by the fact that adjacencies do not occur at a single facet (*i.e.*, a cell can be adjacent to more than one cell, as the cells do not necessarily form a cell complex). Again,

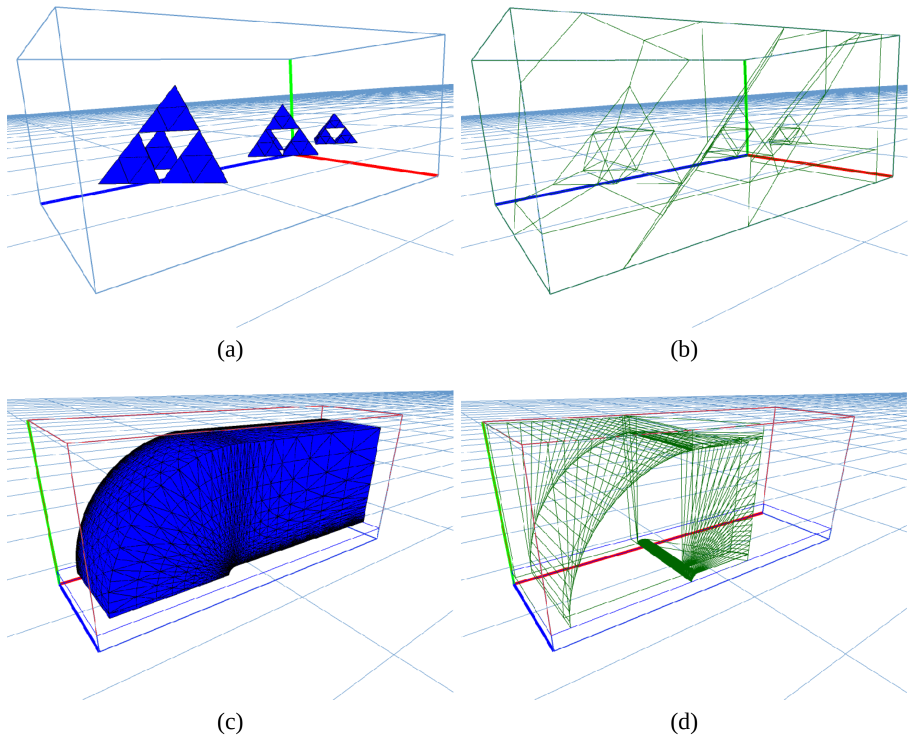


Fig. 6. Explicit BSP regions. Two sample meshes ((a) and (c)) and the correspond BSP-regions that fill space ((b) and (d)).

geometric and topological predicates that guarantee the validity of topological relationships need to be enforced (*e.g.*, if a cell c_i is adjacent to c_j by way of facet f_m, then there must exist a facet f_n such that c_j is adjacent to c_i by way of facet f_n).

5 Final Remarks

This work presents a brief summary of the current status of strategies to compute a convexification of space with respect to a non-convex mesh. We present a formal definition of the problem and summarize the requirements that one solution needs to fulfill. We discuss two possible solutions. The first is based in Delaunay triangulations; we point out some of the difficulties faced by this approach. We discuss the use of BSP-trees as a potentially better and more practical solution to the problem. However, many problems are still open. For example, what is a practical method for convexification that avoids the generation of cycles in the visibility relationship?

Acknowledgements

We thank Dirce Uesu for help in the preparation of images for this paper. Figure 1 was generated using Jonathan Shewchuk's `Triangle` software. The work of João L. D. Comba is supported by a CNPq grant 540414/01-8 and FAPERGS grant 01/0547.3. Joseph S.B. Mitchell is supported by NASA Ames Research (NAG2-1325), the National Science Foundation (CCR-0098172), Metron Aviation, Honda Fundamental Research Lab, and grant No. 2000160 from the U.S.-Israel Binational Science Foundation. Cláudio T. Silva is partially supported by the DOE under the VIEWS program and the MICS office, and the National Science Foundation under grants CCF-0401498, EIA-0323604, and OISE-0405402.

References

1. M. de Berg, M. Overmars, and O. Schwarzkopf. Computing and verifying depth orders. *SIAM J. Comput.*, 23:437–446, 1994.
2. M. Bern. Triangulations and mesh generation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry (2nd Edition)*, chapter 25, pages 563–582. Chapman & Hall/CRC, Boca Raton, FL, 2004.
3. M. Bern, D. Dobkin, and D. Eppstein. Triangulating polygons without large angles. *Internat. J. Comput. Geom. Appl.*, 5:171–192, 1995.
4. M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 1 of *Lecture Notes Series on Computing*, pages 23–90. World Scientific, Singapore, 1992.
5. M. Bern and P. Plassmann. Mesh generation. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 291–332. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
6. B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.
7. B. Chazelle and D. P. Dobkin. Optimal convex decompositions. In G. T. Toussaint, editor, *Computational Geometry*, pages 63–133. North-Holland, Amsterdam, Netherlands, 1985.
8. B. Chazelle, H. Edelsbrunner, L. Guibas, J. Hershberger, R. Seidel, and M. Sharir. Selecting heavily covered points. *SIAM J. Comput.*, 23:1138–1151, 1994.
9. B. Chazelle, H. Edelsbrunner, L. J. Guibas, R. Pollack, R. Seidel, M. Sharir, and J. Snoeyink. Counting and cutting cycles of lines and rods in space. *Comput. Geom. Theory Appl.*, 1:305–323, 1992.
10. B. Chazelle and L. Palios. Triangulating a non-convex polytope. *Discrete Comput. Geom.*, 5:505–526, 1990.
11. L. P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4:97–108, 1989.
12. D. Cohen-Steiner, E. Colin de Verdière, and M. Yvinec. Conforming Delaunay triangulations in 3d. In *Proc. 18th Annu. ACM Sympos. Comput. Geom.*, 2002.
13. J. L. Comba, J. T. Klosowski, N. Max, J. S. Mitchell, C. T. Silva, and P. Williams. Fast polyhedral cell sorting for interactive rendering of unstructured grids. In *Computer Graphcs Forum*, volume 18, pages 367–376, 1999.
14. L. De Floriani and E. Puppo. A survey of constrained Delaunay triangulation algorithms for surface representaion. In G. G. Pieroni, editor, *Issues on Machine Vision*, pages 95–104. Springer-Verlag, New York, NY, 1989.

15. H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.
16. H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge, 2001.
17. S. Fortune. Voronoi diagrams and Delaunay triangulations. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry (2nd Edition)*, chapter 23, pages 513–528. Chapman & Hall/CRC, Boca Raton, FL, 2004.
18. H. Fuchs, Z. M. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. *Comput. Graph.*, 14(3):124–133, 1980. Proc. SIGGRAPH '80.
19. M. P. Garrity. Raytracing irregular volume data. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):35–40, Nov. 1990.
20. C. Hoffmann. *Geometric and Solid Modeling*. Morgan-Kaufmann, San Mateo, CA, 1989.
21. J. M. Keil. Polygon decomposition. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 491–518. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
22. M. Kraus and T. Ertl. Cell-projection of cyclic meshes. In *IEEE Visualization 2001*, pages 215–222, Oct. 2001.
23. M. Kraus and T. Ertl. Simplification of Nonconvex Tetrahedral Meshes. In Farin, G. and Hagen, H. and Hamann, B., editor, *Hierarchical and Geometrical Methods in Scientific Visualization*, pages 185–196. Springer-Verlag, 2002.
24. M. Murphy, D. M. Mount, and C. W. Gable. A point-placement strategy for conforming Delaunay tetrahedralization. In *Proc. 11th ACM-SIAM Sympos. Discrete Algorithms*, pages 67–74, 2000.
25. G. M. Nielson, H. Hagen, and H. Müller. *Scientific Visualization: Overviews, Methodologies, and Techniques*. IEEE Computer Society Press, Washington, DC, 1997.
26. A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Chichester, UK, 1992.
27. J. Ruppert and R. Seidel. On the difficulty of triangulating three-dimensional nonconvex polyhedra. *Discrete Comput. Geom.*, 7:227–253, 1992.
28. S. Schirra. Robustness and precision issues in geometric computation. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 14, pages 597–632. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
29. R. Seidel. Constrained Delaunay triangulations and Voronoi diagrams with obstacles. Technical Report 260, IIG-TU Graz, Austria, 1988.
30. J. R. Shewchuk. A condition guaranteeing the existence of higher-dimensional constrained Delaunay triangulations. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 76–85, 1998.
31. J. R. Shewchuk. Constrained Delaunay tetrahedralizations, bistellar flips, and provably good boundary recovery. Talk slides; available from author's web page.
32. J. R. Shewchuk. Lecture notes on Delaunay mesh generation. Technical report, Department of Electrical Engineering and Computer Science, University of California at Berkeley, 1999.
33. J. R. Shewchuk. Sweep algorithms for constructing higher-dimensional constrained Delaunay triangulations. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 350–359, 2000.

34. J. R. Shewchuk. Updating and constructing constrained Delaunay and constrained regular triangulations by flips. In *Proc. 19th Annu. ACM Sympos. Comput. Geom.*, pages 181–190, 2003.
35. P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. In *San Diego Workshop on Volume Visualization*, volume 24 of *Comput. Gr*, pages 63–70, Dec. 1990.
36. C. T. Silva, J. S. Mitchell, and P. L. Williams. An exact interactive time visibility ordering algorithm for polyhedral cell complexes. In *1998 Volume Visualization Symposium*, pages 87–94, Oct. 1998.
37. D. Speray and S. Kennon. Volume probes: Interactive data exploration on arbitrary grids. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):5–12, November 1990.
38. C. Stein, B. Becker, and N. Max. Sorting and hardware assisted rendering for volume visualization. *1994 Symposium on Volume Visualization*, pages 83–90, October 1994. ISBN 0-89791-741-3.
39. N. P. Weatherill and O. Hassan. Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. *International Journal for Numerical Methods in Engineering*, 37:2005–2039, 1994.
40. M. Weiler, M. Kraus, M. Merz, and T. Ertl. Hardware-Based Ray Casting for Tetrahedral Meshes. In *Proceedings of IEEE Visualization 2003*, pages 333–340, 2003.
41. P. L. Williams. Visibility ordering meshed polyhedra. *ACM Transaction on Graphics*, 11(2):103–125, Apr. 1992.
42. C. Yap. Towards exact geometric computation. *Comput. Geom. Theory Appl.*, 7(1):3–23, 1997.