Eurographics Symposium on Geometry Processing (2005)
M. Desbrun, H. Pottmann (Editors)

Atomic Volumes for Mesh Completion

Joshua Podolak and Szymon Rusinkiewicz

Princeton University, Princeton NJ

Abstract

The increased use of scanned geometry for applications in computer graphics and 3D hardcopy output has highlighted the need for general, robust algorithms for reconstruction of watertight 3D models given partial polygonal meshes as input. We present an algorithm for 3D hole filling based on a decomposition of space into atomic volumes*, which are each determined to be either completely inside or completely outside the model. By defining the output model as the union of interior atomic volumes we guarantee that the resulting mesh is watertight. Individual volumes are labeled as "inside" or "outside" by computing a minimum-cost cut of a graph representation of the atomic volume structure, patching all the holes simultaneously in a globally sensitive manner. User control is provided to select between multiple topologically distinct, yet still valid, ways of filling holes. Finally, we use an octree decomposition of space to provide output-sensitive computation time. We demonstrate the ability of our algorithm to fill complex, non-planar holes in large meshes obtained from 3D scanning devices.*

1. Introduction

Polygonal representations are widely used in computer systems and applications for modeling 3D geometry. One method of creating a polygon-mesh representation of a 3D object involves scanning the object from a number of different viewpoints and constructing the mesh from a combination of surfaces obtained by those scans. During the reconstruction process, however, there will usually exist a number of areas for which no data was obtained. This occurs because most of the 3D scanners obtain depths based on some form of parallax. In order to obtain this parallax, two different, unobstructed lines of sight are necessary between the scanner and the object. If this requirement is not met for some point on the object's surface, there will be no depth value at that point. Other reasons for missing or unreliable data include self occlusions, low reflectance coefficients of the surface, or high grazing angles for one or both of the lines of sight. While incomplete data may be sufficient for certain applications, many others require a watertight surface. Therefore, it is useful to have a method for filling in the areas of missing data.

Areas in a surface of scanned input that contain no data result in holes in the mesh bounded by rings of *half-edges*, mesh edges that are adjacent to only one triangle. In simple cases, it is sufficient to create a patch by triangulating the half-edges around the holes. However, for general hole topologies, simple triangulation is insufficient, and a more flexible method must be considered. Another possible problem with triangulation is that the added patch might intersect portions of the mesh away from the hole, creating an inter-penetrating object. In order to ensure that the resulting mesh is not inter-penetrating, a solution that takes into account the entire mesh is necessary. For example, in Figure 1, a naive triangulation of the half-edge rings (shown in red) will intersect the cylinder running through the sphere. Instead, in order to fill the hole in the sphere correctly, our algorithm connects the half-edge rings on either side of the sphere. Note that connecting the rings on the sphere to the rings on the cylinder is not a valid solution as it will be inconsistent with existing normals. Therefore, in a valid solution, the cylinder must be filled.

Finally, because there potentially exist multiple desirable solutions, the user must have the ability to influence the output surface. In Figure 2 for example, the outer half-torus shown on the left is filled by our algorithm in two topologically dissimilar ways. The ability to choose the desired topology of the filled mesh and to incorporate additional constraints is important for versatile hole-filling.

Overall, robust hole-filling should satisfy the following criteria:

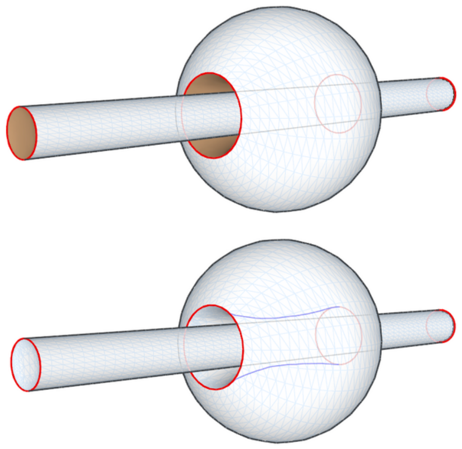


Figure 1: *These images are a stylistic rendering of a result produced by our algorithm. In the top image, the hole in the sphere (the backfaces of the model are drawn in brown and the boundary of the hole is red) cannot be triangulated naively in a manner consistent with existing faces and normals without intersecting the surface of the cylinder running down its center. Instead, our algorithm, as shown in the bottom image, correctly creates a patch connecting the two rings.*

1. Produce a non-self-intersecting watertight mesh;
2. Process arbitrary holes in complex meshes;
3. Avoid changing, approximating or re-sampling the original data away from the holes;
4. Incorporate user-provided constraints to allow the selection of multiple topologically differing solutions;
5. Process large scanned meshes with a running time proportional to the size of the holes, rather than that of the input mesh;

We propose an approach that meets these criteria through a two-step process. In the first step, a bounding cube of the input mesh is partitioned into atomic volumes. A volume is *atomic* if it cannot be intersected by the polygons of the mesh. By this formulation, each atomic volume will be either entirely inside the final output mesh (a *inside* volume) or outside it (an *outside* volume). We define the output model to be the union of the interior atomic volumes, thus implicitly identifying the output mesh as the boundary between the inside volumes and the outside volumes. This implicit definition guarantees that the resulting object is watertight. Indeed, once certain basic constraints are met, *any* classification of atomic volumes into inside and outside regions will yield a watertight surface. (Note that the solution might not be manifold unless steps are taken to ensure that volumes with trivial boundaries are not possible.) The use of atomic volumes also guarantees that the surface approximated by the input mesh is not changed.

We use an octree decomposition to simplify the partition of space into atomic volumes. A full partition is only necessary near the holes. This provides output sensitivity.

We create a graph representation in order to decide which of the atomic volumes are inside and which are outside the model. In this representation, each atomic volume is represented by a node, and weighted edges connect nodes that correspond to neighboring volumes. The weight of an edge is the cost of adding a face separating those two volumes. This graph is separated using a min-cut algorithm into two sub-graphs, one containing all the nodes representing interior volumes and the other containing all the exterior volumes. The use of a min-cut algorithm ensures a global solution, while the edge-weights in the graph provide flexibility in deciding the characteristics of the filling patch.

At the end of the first step, each atomic volume has been labeled as inside or outside and the boundary between the inside and the outside of the model is watertight. However, the resulting surface can be faceted. In the second step of the algorithm, we preform a smoothing pass to make the underlying atomic volume structure less visible. By enforcing the rule that the labeling of the atomic volumes cannot be changed during the smoothing step, we ensure that the correctness of the final model has not been compromised. That is, unlike most traditional mesh filtering algorithms, we guarantee that the smoothed mesh remains non-self-intersecting.

2. Previous and Related Work

There exist several general methods for creating watertight meshes from various types of initial data. Methods such as Power Crust [ACK01], or Spectral Watertight Surface Reconstruction [Kol03] create meshes from point clouds and can ensure watertight surfaces during reconstruction. These methods may be thought of as performing hole filling, since they reconstruct surfaces across areas of irregular, possibly sparse samplings. Hole filling on a larger scale (from areas with missing data) is interpreted as reconstruction of areas with even lower sampling rates. While the surface generated in these cases will be watertight, the topology of the mesh may not be the topology of the original model, especially if the holes are very large.

Other dedicated hole-filling methods assume that a partial surface exists and the missing areas must be reconstructed from an existing polygonal model. These methods can be categorized into two types:

Geometric methods attempt to triangulate the closed loops of half-edges on the boundary of a hole, but do not necessarily enforce a non-inter-pentrating triangulation. In [Lie03], for example, the triangulation is refined so that the vertex density of the patch corresponds to the vertex density of the surrounding surface. Subsequent passes over the filled areas can warp vertex positions to smooth the patch

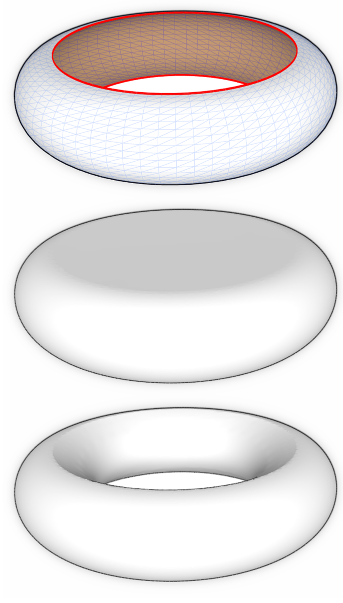


Figure 2: *Complex holes can be filled in different manners. In the example above, (Top) the hole in the torus is bounded by two rings of edges (shown in red). (Middle) The hole is filled using two using two patches with a disc topology. (Bottom) The hole is filled using a patch with a ring topology. The bottom two images are solutions produced by our hole-filling algorithm, which is capable of handling any topology and efficiently producing any desirable watertight solution.*

or to introduce geometrical texture based on surrounding regions.

Volumetric methods obtain the surface implicitly as the boundary between volumetric regions that are labeled as inside and those labeled as outside the model, thus ensuring that the resulting surface is manifold.

In [Ju04] meshes are repaired by contouring the half-edge loops surrounding the holes, filling them based on local constraints. Using a volumetric data structure as the underlying foundation, the resulting surface is guaranteed to be watertight, and the use of an octree data structure coupled with local hole-filling ensures efficiency. The weakness of this scheme is that the original surface is only approximated and even the topology of the input surface may be changed. In addition, defining a hole merely as a ring of half-edges restricts the use of this method, since it would not be able to fill holes such as the one in Figure 1.

Volumetric Diffusion [DMGL02] uses the zero set of a signed distance function derived from the original mesh to define the surface of a model. Initially, this function is only defined where data exists. The holes are then filled by extending this function using a diffusion process until the zero set is watertight. This method will work for models with an arbitrary topology and allows use of data from other sources, such as space carving, but does not retain the original surface (since the model is approximated using a signed distance function). More importantly, the heat-diffusion equations limit the signed distance function to extend in straight lines, and therefore might not yield solutions in cases where the surfaces that must be connected are in non-converging directions. [Mas04] presents a variation that can better handle curvature, extending the surface using a quadric approximation of the signed distance function. More generally, the disadvantage of this approach is that a solution is obtained by growing local patches from the boundaries of the holes until they connect. In contrast to this method, the atomic volumes algorithm solves a *global* optimization to directly determine the shape of the added geometry, rather than relying on *local* behavior governed by a differential equation. In addition, we avoid resampling the original surface away from the hole.

[MF97] introduced a mesh repair process to fix errors and inconsistencies in models. In this approach, a manifold mesh is created from a polygon soup by splitting a bounding volume of the model in a BSP tree. Nodes in the tree are considered to be on opposite sides (i.e., inside or outside) of the mesh based on the percentage of area between the nodes filled in by the input mesh. The repaired mesh is created from a globally optimal solution calculated on the tree. Our method incorporates the same insights of performing volumetric decomposition and globally solving for inside/outside, but specializes them to the hole-filling problem while providing output-sensitive computation time, explicit user control over topology, and guaranteed non-intersecting smoothing.

Graph Cuts have long been widely used in both the graphics and vision communities. Recent work with graph cuts includes segmentation [SM00], image and video synthesis [KSE*03], and surface reconstruction from images [PSQ05]. In our method, the graph formulation comes directly creates from the atomic volume structure used to define the object. We use the algorithm of [KS96] to segment our graph.

3. Atomic Volume Creation

A natural way to divide all of space into atomic volumes such that each will be either totally inside an object or totally outside is to tetrahedralize a bounding volume of the object. This tetrahedralization must be *constrained*, since existing faces of the mesh must not intersect any tetrahedra. Constrained 3D tetrahedralization is not a simple problem, and there are a number of algorithms that attempt to triangulate 3D objects while minimizing various parameters [MV92, She98]. A weakness of using such a method is that a large number of tetrahedra will be generated in areas that are not near a hole, and while it is clear that the patched surface will never go near these volumes, they are included as separate atomic volumes. Consequently, the graph-cut al-

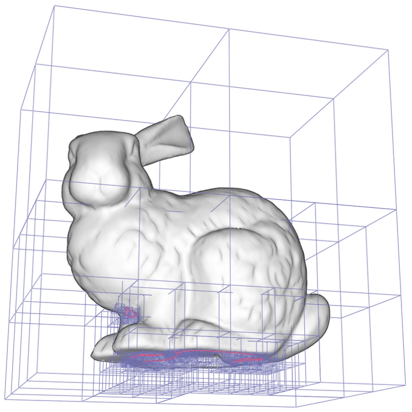
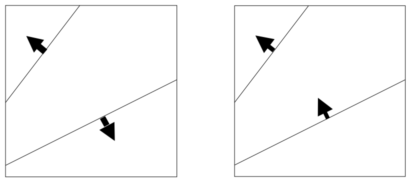


Figure 3: *The cube containing the entire head of the bunny may be represented by two atomic volumes. It makes no difference which regions of the cube are inside and which regions are outside or how many connected components there are in that cube, because the boundary between these halves is completely defined by the input mesh. The octree needs to be refined only in areas where the boundary is not fully defined. This occurs only near the holes.*

gorithm must consider them separately, considerably slowing down the process.

3.1. Octree

We use an octree scheme to limit tetrahedralization to areas near the holes. This retains the ability to efficiently fill holes with highly irregular boundaries while at the same time allowing the patch to span large holes. To create the octree, a bounding volume of the mesh is adaptively split into cubes until each cube contains a trivial (for hole-filling purposes) portion of the mesh. Cubes that do not contain the boundary of a hole generally need not be split any further, since either these cubes do not contain any part of the initial mesh (in which case in the final solution the atomic volumes within such a cube can be either all labeled inside or all labeled outside), or they have a number of faces in them that do not bound the hole. In the latter case, the cube is partially inside and partially outside the model. However, we make the critical observation that the boundary between the inside and outside atomic volumes in the cube is completely defined by the input mesh. Any partitioning of such a cube into atomic volumes is ineffectual, because any consistent labeling of atomic volumes in the cube will yield a boundary that includes only the existing mesh triangles. In Figure 3, for example, the cube containing the entire head of the bunny does not need to be partitioned because the patch filling the holes in the base of the bunny will never go near that cube. It is critical to note that the number of connected components in such a cube makes no difference. Because the surface dividing the inside parts from the outside parts is *completely*



(a) Consistent (b) Must be subdivided

Figure 4: *(b) must be subdivided because the middle area is on both sides of the input surface (the normals are pointing outward).*

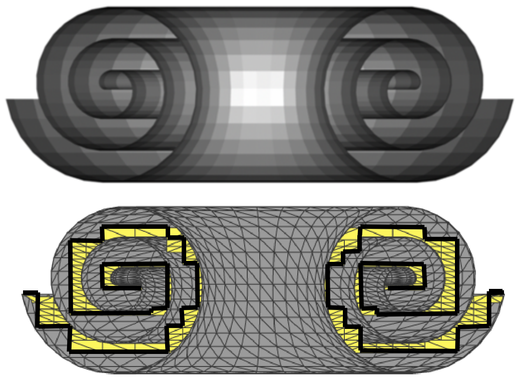


Figure 5: *A cross section of a spiral torus filled using our method. Note that considering existing faces and normals, the only correct way to fill this hole is with a patch that spirals out along the surface.*

defined, the algorithm need not consider that cube at all. It is sufficient to define the cube as "partially inside and partially outside the model" and to record that the boundary of the model inside the cube is already known.

There are rare cases when the patch filling the holes must pass through cubes not near the holes. For example, consider the cross-section of the spiral torus shown in Figure 5. The hole in that model can only be filled with a single patch that spirals out along the input surface. In this example, octree cubes containing more than one connected component of the torus need to be partitioned because the patch will necessarily pass through those cubes. More generally as shown in Figure 4, any cube containing a volume that bounds *both* the inside and the outside of the input surface must be partitioned. These cases can be easily checked for at subdivision time.

Note that an underlying assumption of this method is that the triangles of the input mesh have valid (i.e., globally-consistent) normals. Applying this algorithm to a non-orientable model (such as a Möbius strip) will cause conflicting constraints on some of the atomic volumes, and will cause the process to fail. This is part of the requirement that the input mesh be non-self-intersecting and orientable. This is always the case for the output of range scan reconstruction approaches such as VRIP [CL96], which was used to reconstruct the meshes for all the scanned examples in the paper. Note that our algorithm currently operates on triangle meshes, and does not take special advantage of the fact that the mesh may have been obtained from a volume. If the volumetric data were used directly, then using an atomic volume approach would be even simpler and would still produce a global solution rather than relying on the local behavior of the existing surface.

Our method of octree creation is based on a simplified version of the octree used by [MV92, BDE92], since the only faces that trigger subdivision are the ones surrounding holes. The only criterion for stopping octree subdivision is the triviality of the atomic volumes. Using such a simple criterion may cause the octree to be of significant depth in order to connect the patch with an initial mesh that has closely spaced vertices, but this will only occur near the holes. As shown by [MV92] cube size will be no smaller than one quarter the distance between two vertices. Because the octree is divided only near the holes, the number of octree cubes depends only on the relative size of the hole with respect to the model, and on the size of the triangles surrounding the hole. We found that if, for example, the input mesh is subdivided – increasing the number of triangles by a factor of four – the number of octree cubes increases only by a factor of two.

If using an "unbounded" octree depth is problematic, another option is to limit the octree to a fixed depth. If a cube reaches the max depth, that cube is tetrahedralized (or split into atomic volumes using a BSP tree). In our effort to maintain the input mesh, we have avoided changing or removing any of the input triangles, but since the only requirement is to not change the surface away from the holes, offending faces around the holes could also be removed to ease tetrahedralization.

As shown in Figure 6, the completed octree consists of three type of cubes:

Blank cubes do not contain any faces of the original mesh. The cube can therefore be considered to be either entirely inside or entirely outside the model.

Inside/Outside (IO) cubes contain faces of the mesh that are not adjacent to a hole. For these cubes, even though there is no explicit partition into atomic volumes, the boundary between the inside and outside is precisely known and can be ignored.

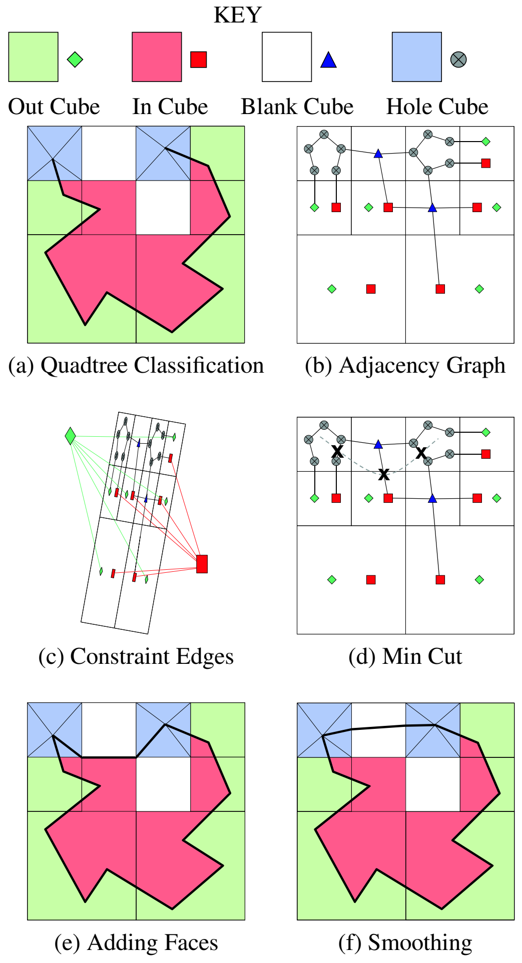


Figure 6: *These images show the steps of the algorithm pipeline on a 2D example. (a) A quadtree is created out of a bounding square of the input curve. Squares are split until triangulation of the end points is trivial. The quadtree square types are labeled: green and red squares are **IO** squares, **blank** squares are white, and **hole** squares are colored in blue and triangulated. (b) The adjacency graph is created. **IO** squares get two graph nodes, **blank** squares get one, and **hole** squares get one node per triangle. Nodes corresponding to neighboring volumes are connected by an edge with a finite weight. (c) Constraint Edges are added. All **IO** nodes are constrained by edges connecting them to the proper node. (d) The min-cut pass splits the graph into two sub-graphs. (e) The corresponding edges are added to the mesh creating a watertight surface. (f) Smoothing is preformed on the patch.*

Hole cubes contain either a single vertex that lies on the boundary of a hole (boundary vertex), or a single edge that lies on the boundary of a hole (boundary edge). These cubes are trivially tetrahedralized into atomic volumes, with a single center point (the boundary vertex in the cube, or a point on the boundary edge in the cube) as the head of all the tetrahedra, while the bases of the tetrahedra are obtained by triangulating each of the faces of the cube. The tetrahedra created do not intersect the faces in the hole cubes, and therefore may each be labeled as inside or outside separately.

4. Label Assignment

Once space has been partitioned, each atomic volume is labeled as either either inside or outside the model. As shown in Figure 6d we use a graph cut [KS96] algorithm to label the volumes in a global manner.

4.1. Graph Nodes

We create a graph by assigning each octree cube a number of nodes corresponding to the number of significant atomic volumes it contains. Blank cubes correspond to a single node in the graph. IO cubes are always described by two nodes, one node representing the atomic volumes inside the model, and the other node representing the atomic volumes outside the model. Hole cubes are allotted one node for each tetrahedron contained within. In addition, the graph contains two other nodes: the source node and the sink node. After running the min-cut on the graph, the atomic volumes whose nodes are on the source side of the graph will be labeled as inside, while the volumes whose nodes are on the sink side will be labeled as outside.

4.2. Graph Edges

Edges in the graph are designated as Constraint Edges or Boundary Edges, and are given appropriate weights.

Constraint Edges are edges with an edge-weight of infinity, and connect the source and sink to nodes corresponding to atomic volumes that have been pre-defined as inside or outside. For example, a node corresponding to the inside volumes of an IO cube will have a constraint edge connecting that node to the source node. Because the weight of the edge is infinite, this edge will never be cut by the min-cut algorithm, guaranteeing that the interior volumes of that cube are labeled as inside the model at the end of the process.

In a similar manner, any user-defined constraints (as described below) are represented with edges of infinite weight connected to the proper node.

Boundary Edges are edges with a finite edge-weight, placed between any two nodes with adjacent atomic volumes. If the boundary between two volumes is intersected by a face of the input surface then no edge is needed. In such a case, both volumes must be IO cubes, and therefore the boundary between those two cubes is pre-defined. In all other cases, an edge is added.

In most cases, adding the edge means connecting the two nodes corresponding to neighboring atomic volumes in the graph. When connecting to an IO node, however, an extra test is necessary in order to decide if to connect to the inside portion of the cube, or the outside portion. We take advantage of the fact that the boundary between an atomic volume and an IO cube is always a polygon that does not intersect any input triangles. Furthermore, the boundary between the inside volumes and the outside volumes in the cube is completely defined. Therefore, we may determine whether *any* point is inside or outside by checking the normal at the closest point on the surface. This lets use determine if that atomic volume should be connected to to the inside node of the IO cube or the outside node.

A node corresponding to an atomic volume on the boundary of the octree is connected by an edge to the sink (exterior) node with a finite edge weight. If this edge is cut, a face on the bounding cube is added.

The exact weight of the boundary edges determines the nature of the patch generated. For example, if all boundary edges are given an edge weight of one, the hole will be filled with a minimum number of faces. If the edge weight used is the area of the boundary between the two atomic volumes, the surface area of the patch will be minimized. It is important to note that no matter what edge-weights are given, as long as they are finite, the algorithm will yield a correct, watertight surface. In the examples in this paper, we use weights based on surface area, yielding minimum-area patches.

4.3. User Constraints

In addition to the constraints imposed on the atomic volumes by the initial polygon mesh, further constraints may be added based on other sources of information such as space carving [DMGL02], shadow carving [SRBP02], or direct user input. As shown in Figure 7, if a portion of space needs to be outside or inside the model in the final solution, the node corresponding to the volume that contains that area gets an additional constraint edge, ensuring that the node falls in the correct sub-graph. Cubes with multiple contradicting constraints are subdivided. Once the graph is partitioned, any polygon that corresponds to a cut edge is added to the mesh, creating a watertight surface.

5. Smoothing

The next step in our algorithm involves smoothing the resulting patch. Because the first phase of the algorithm deals only with atomic volumes, the surface created by a union of such volumes will necessarily be faceted, and some smoothing is necessary in order to create a visually pleasing patch. The surface is smoothed by moving the vertices of the newly created patch faces. In our implementation, each point on the patch is pulled as if it had a spring connected to each of its neighbors, providing an approximation to Laplacian

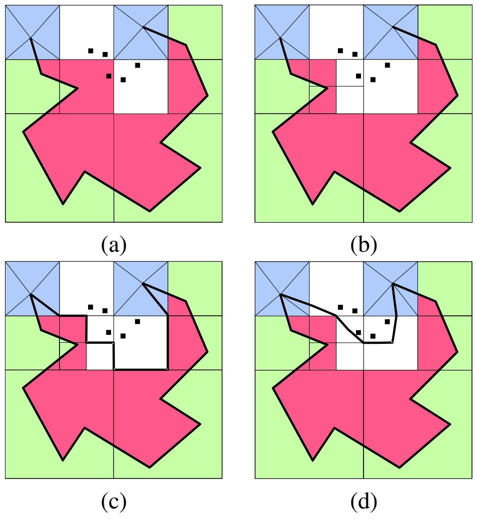


Figure 7: *(a) Points constrained to be outside the mesh are added. (b) The octree only needs to subdivide where there are contradicting constraints. (c) The min-cut is run normally on the new graph and (d) the resulting patch is smoothed.*

mesh smoothing. Points are moved simultaneously to their new location. Points from the input mesh are not allowed to move.

In order to smooth the vertices correctly, the topology of the graph must be preserved. This still provides leeway for smoothing as the atomic volumes can be warped as long as the graph adjacency information is unchanged. In our implementation, limited warping of the atomic volumes is allowed. For every atomic volume, the "center" of the volume is defined as either the centroid or, if user constraints are used, the point (or multiple points) in the cube specified by the user. Vertices may be moved as long as each remains within its own area, defined by the centers of the neighboring atomic volume. In practice, for each point, we check if moving it will cause the centers of the neighboring atomic volumes to fall outside of their volume. If moving a point fails this test, the point is not moved during that iteration. An additional constraint is that IO cubes cannot be violated, as we have no knowledge of how the surface behaves inside them. We found that this method yielded smooth patches for the models tested.

In order to allow more freedom with smoothing, an optional extension of this method would be to allow smoothing without any constraints (with a possible result of a self-intersecting surface). The new surface will then intersect a new set of atomic volumes. Edges with (a small) finite weight can be added from the source and sink nodes to these volumes to "suggest" that the surface pass through them. Running a new min-cut on the new graph will incorporate these suggestions while maintaining all the constraints imposed by the user and by the input mesh. A subsequent constrained smoothing pass will ensure unfaceted results.

Model	Number of faces	Number of nodes	Time
Skull	1,250	10,481	8 sec
Bunny	70,000	17,300	78 sec
Angel	340,000	530,000	17.5 min
Toes	350,000	570,000	19 min

6. Results

We tested our algorithm on a number of complex meshes, including well-known examples of scans with holes. The results are topologically accurate and preserve the original data away from the holes. The models were run on a Pentium 4 with 2GHz and 3GB of RAM. Timings and octree sizes are shown in the table above.

Figure 8 shows our results on the Stanford bunny model, which has holes in the base and feet. Our technique fills the holes, while preserving the existing geometry. The skull model (Figure 10) contains a large proportion of missing geometry. Our unsmoothed result shows that the initial atomic volume decomposition produced relatively large flat areas, showing that space was not subdivided except near the boundaries of the existing mesh. Smoothing produces a final result.

Figures 9 and 11 show examples of large, complex scanned meshes, each containing around 350,000 polygons. Notice that the model of the toes of Michelangelo's David has been filled in two different ways: in 11b, no user-based constraints are added, and the two toes are connected by a handle. In 11c, the user adds constraint points to select the correct way to fill the largest hole. Although the selection was done manually in this case (by constraining a few cubes to be outside the surface), space-carving data could be used to obtain the same result automatically.

7. Conclusions and Future Work

This paper investigates the problem of hole-filling, and proposes a solution based on adaptive spatial decomposition and graph cuts. Utilizing atomic volumes as the foundation of the algorithm allows user-based constraints such as space carving to be added to the input, and the min-cut algorithm provides an intuitive global method to split space into inside and outside portions.

While other types of atomic volumes may be employed (splitting the entire volume into tetrahedra, or using a BSP tree), the adaptive octree data structure allows the algorithm to focus on the areas surrounding the holes, and utilizing cubes as the basic volume allows easy smoothing.

Smoothing the patch after it has been constructed must be done carefully. Perturbation of the surface must not cause self-intersection or the breaking of one of the user-based constraints. The octree data structure allows local inspection of the surface to ensure correctness.

The existing algorithm could be extended in several ways. First, our current implementation of smoothing allows only the boundaries between atomic volumes, not the centers of the volumes, to move. A more flexible smoothing scheme would allow the centers of the atomic volumes to move as long as the topology of the graph describing the volumes does not change. In addition, we have only explored the simplest method of smoothing, based on minimizing surface area. However, the framework is general enough to be adapted to any smoothing method, such as minimizing thin-plate energy [Mas04], or texture synthesis from existing local surface data [SACO04].

Currently all edges between the graph nodes and the source and sink have an edge weight of infinity. A additional way to influence the resulting patch is to add "hints," as opposed to constraints, about what parts of space are inside the model and what parts are outside. By adding finite weighted edges between such nodes and the corresponding source/sink node, we allow atomic volumes to be on the wrong side of the model "for a price." An example of such a hint would be to add a "soft" symmetry constraint, for a translational or rotational symmetry that is either specified manually or detected automatically [KFR04]. Atomic volumes would therefore be suggested to be inside or outside based on the symmetric counterpart in the input mesh.

Finally, a more general line of research is to take advantage of our dual surface- and volume-based representation to enforce topological constraints and non-intersection during a variety of other geometric signal processing algorithms. Localized filtering, deformation, and collision are all applications that would benefit from an adaptive, local data structure that guarantees results that are manifold and nonintersecting.

References

[ACK01] AMENTA N., CHOI S., KOLLURI R. K.: The power crust, unions of balls, and the medial axis transform. *Computational Geometry 19*, 2-3 (2001), 127–153. 2

[BDE92] BERN M., DOBKIN D., EPPSTEIN D.: Triangulating polygons without large angles. In *Proceedings of the eighth annual symposium on Computational geometry* (1992), ACM Press, pp. 222–231. 5

[CL96] CURLESS B., LEVOY M.: A volumetric method for building complex models from range images. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), ACM Press, pp. 303–312. 5

[DMGL02] DAVIS J., MARSCHNER S., GARR M., LEVOY M.: Filling holes in complex surfaces using volumetric diffusion, 2002. 3, 6

[Ju04] JU T.: Robust repair of polygonal models. *ACM Trans. Graph. 23*, 3 (2004), 888–895. 3

[KFR04] KAZHDAN M., FUNKHOUSER T., RUSINKIEWICZ S.: Symmetry descriptors and 3D shape matching. In *Proc. Symposium on Geometry Processing* (2004). 8

[Kol03] KOLLURI R. K.: Spectral watertight surface reconstruction, 2003. 2

[KS96] KARGER D. R., STEIN C.: A new approach to the minimum cut problem. *J. ACM 43*, 4 (1996), 601–640. 3, 6

[KSE*03] KWATRA V., SCHÖDL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003 22*, 3 (July 2003), 277–286. 3

[Lie03] LIEPA P.: Filling holes in meshes. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2003), Eurographics Association, pp. 200–205. 2

[Mas04] MASUDA T.: Filling the signed distance field by fitting local quadrics. In *3DPVT* (2004), pp. 1003–1010. 3, 8

[MF97] MURALI T. M., FUNKHOUSER T. A.: Consistent solid and boundary representations from arbitrary polygonal data. In *Proceedings of the 1997 symposium on Interactive 3D graphics* (1997), ACM Press, pp. 155–ff. 3

[MV92] MITCHELL S. A., VAVASIS S. A.: Quality mesh generation in three dimensions. In *Proceedings of the eighth annual symposium on Computational geometry* (1992), ACM Press, pp. 212–221. 3, 5

[PSQ05] PARIS S., SILLION F., QUAN L.: A surface reconstruction method using global graph cut optimization. *International Journal of Computer Vision* (2005). to appear. 3

[SACO04] SHARF A., ALEXA M., COHEN-OR D.: Context-based surface completion. *ACM Trans. Graph. 23*, 3 (2004), 878–887. 8

[She98] SHEWCHUK J. R.: Tetrahedral mesh generation by delaunay refinement. In *Symposium on Computational Geometry* (1998), pp. 86–95. 3

[SM00] SHI J., MALIK J.: Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 8 (2000), 888–905. 3

[SRBP02] SAVARESE S., RUSHMEIER H. E., BERNARDINI F., PERONA P.: Implementation of a shadow carving system for shape capture. In *3DPVT* (2002), pp. 12–23. 6

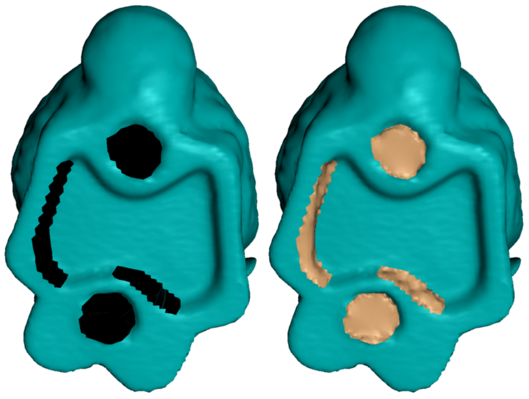


Figure 8: *The well-known bunny model is filled with an octree depth of 12. The hole filling process took a little longer than a minute*

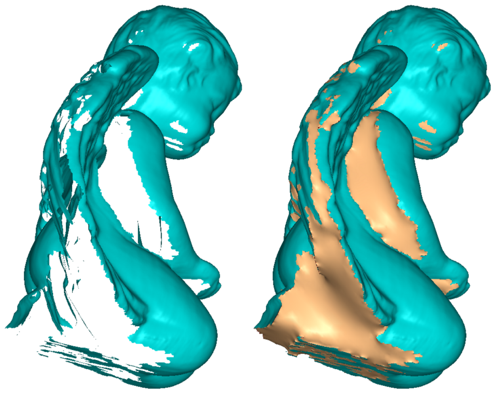


Figure 9: *A scanned angel model contained numerous holes that take up a large portion of the surface. The octree reached a maximum depth of 23 to split some of the vertices surrounding the holes, though only 530,000 total nodes were required. The hole-filling process took about 15min.*

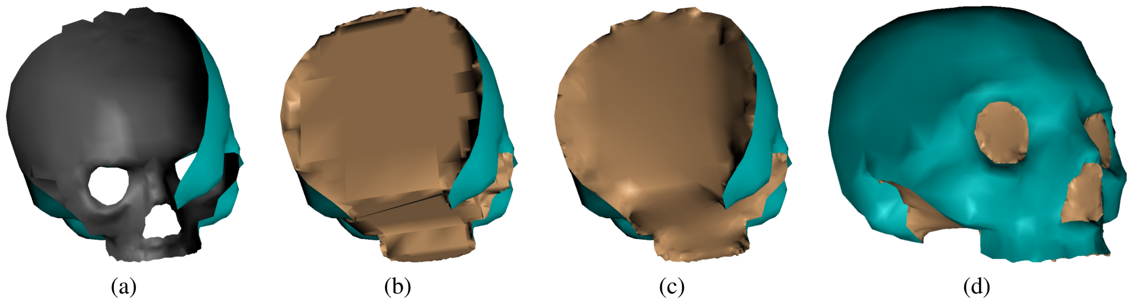


Figure 10: *(a) Our algorithm filled this skull model, even though a large percentage of the surface area was missing. (b) The min-cut produces an initial result. (c,d) This is smoothed to produce a more pleasing solution.*

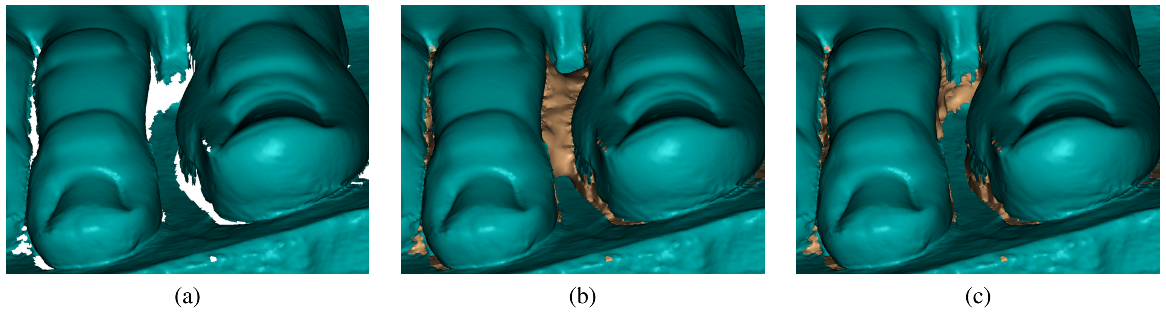


Figure 11: *(a) The toes of Michelangelo's David. (b) The repaired model without user-based constraints. Note the "tunnel" connecting the two toes. (c) Adding user-based constraints. By asserting that the area between the two toes is empty (manually in this case, though space-carving data could also be used), a new patch is created, correctly filling the hole between the toes.*