Eurographics Symposium on Geometry Processing (2007)
Alexander Belyaev, Michael Garland (Editors)

Unconstrained Isosurface Extraction on Arbitrary Octrees

Michael Kazhdan[1], Allison Klein[2], Ketan Dalal[2], and Hugues Hoppe[3]

[1]Johns Hopkins University, Baltimore MD, USA
[2]Microsoft, Redmond WA, USA
[3]Microsoft Research, Redmond WA, USA

Abstract

This paper presents a novel algorithm for generating a watertight level-set from an octree. We show that the level-set can be efficiently extracted regardless of the topology of the octree or the values assigned to the vertices. The key idea behind our approach is the definition of a set of binary edge-trees *derived from the octree's topology. We show that the edge-trees can be used define the positions of the isovalue-crossings in a consistent fashion and to resolve inconsistencies that may arise when a single edge has multiple isovalue-crossings. Using the edge-trees, we show that a provably watertight mesh can be extracted from the octree without necessitating the refinement of nodes or modification of their values.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Boundary Representations

1. Introduction

Extracting the level-set of an implicit 3D function is a fundamental technique for many computer graphics applications. Traditionally, an implicit function is represented by a regular sampling of the function's values on a voxel grid. While simple, this approach has the drawback that memory usage is based upon the voxel grid's resolution rather than actual model complexity. Therefore, the need for higher resolution models, coupled with memory limitations of commodity computers, has driven a move toward the use of octrees for adaptively representing the implicit functions.

Extracting the level-set from the octree representation is, however, more complicated than extracting it from the regular voxel grid. In the regular grid case, each voxel can be processed independently to generate triangulations of the portion of the level-set that the voxel intersects [WMW86, LC87, NH91, KBSS01]. Since the same decisions are made on both sides of a voxel face, the extracted triangulation is guaranteed to be watertight. However, as seen in Figure 1, the same approach applied to the octree's leaves can yield an inconsistent definition of edges across a face, causing cracks.

Although there has been a large amount of work in this area, previous approaches have primarily focused on designing solutions that adapt the octree in order to ensure that inconsistencies cannot arise. In practice, this is often done by refining leaf nodes in the tree and replacing scalar values associated to the vertices with interpolated values from adjacent vertices. In addition, many of these methods assume that the isovalue at which the surface is extracted is fixed, making it difficult to apply these methods to real-time data exploration where a 3D volume is analyzed by considering the family of surfaces extracted at different isovalues.

In response to these issue, this paper presents a novel algorithm for generating a guaranteed watertight level-set for an arbitrary octree (where both the octree structure and the associated vertex values are unconstrained), without assuming a fixed isovalue. To do this, we introduce of a set of binary *edge-trees*, derived from the octree's topology, that can be used to extract a polygonal mesh which is provably watertight. As part of our method, we use an algorithm from [BS95] for computing a minimum area triangulation of a three-dimensional polygon to transform the extracted polygon mesh into a triangulated level-set.

We begin our discussion in Section 2 by reviewing previous work in isosurface extraction from octrees. We introduce edge-trees and show they can be used to generate a watertight mesh in Section 3. We present results in Section 4 and conclude by summarizing our approach in Section 5.

2. Related Work

Initial work in the extraction of isosurfaces from octrees focused on computational efficiency. Using a complete octree representation, [WG92] encode the minimum and maximum values of the samples falling within a node. This encoding provides a method for efficiently identifying regions of space through which the isosurface cannot pass, thereby avoiding unnecessary traversal of finer nodes. However, this approaches require that the octree be complete and, as a result, the memory overhead remains cubic in resolution.

The memory overhead of maintaining a complete octree has motivated a focus on the extraction of isosurfaces from adaptive octrees. These methods have been primarily applied to the simplification of 3D meshes [Blo88, MS93, SZK95, SFYC96, JLSW02, HWC*05], but have also been applied to the real-time exploration of volumetric data [OR97, WKE99] and the extraction of level-sets from the solution of linear systems realized on an octree [OBA*03, LGF04, KBH06, BGOS06]. These methods can be broadly classified into two categories.

Primal Methods The approach taken by the primal methods is to extend the Marching Cubes algorithm by resolving the inconsistencies that arise when adjacent leaf nodes in an octree are of different depths.

When the octree values are obtained from a complete voxel grid (or by sampling an implicit function), the inconsistencies can be resolved by both using the grid (or function) to consistently define the positions of isovalue-crossings and by using refinement to prevent edges from having multiple isovalue-crossings [Blo88, MS93]. A generalization of this method that generates topology-preserving extractions is presented in [VKSM04]. The limitation of these methods is that they require knowledge of the original function which may not be available. Additionally, the refinement is dependent on the choice of isovalue, making it difficult to apply these methods to applications such as real-time volume data exploration.

Although initially proposed for the context of complete octrees, a more general solution is described in the work of Westermann *et al.* [WKE99]. Assuming a restricted octree in which the depth-disparity between adjacent leaf nodes is never greater than one, the authors show that a watertight surface can be extracted by modifying sample values in regions where adjacent leaf nodes are of different depths. This solution has the advantage that it does not assume the existence of a prior implicit function and the pre-processing of the octree is independent of the isovalue. However, the restriction on the depth disparity between adjacent leaf nodes can necessitate refinement of the original octree, often resulting in a marked increase in both the memory and time of octree processing.

Dual Methods An alternate approach for extracting isosurfaces from octrees was proposed in [JLSW02], in which the feature sensitive extraction of [KBSS01] is generalized to the dual setting. Rather than defining the positions of isovalue-crossings along edges of octree nodes, the authors use Hermite data associated to zero-crossing edges to define isovertex positions in the interior of the nodes, and use the edge-adjacencies between leaf nodes to connect these interior vertices. Although problems with non-manifold and self-intersecting surfaces are addressed in subsequent work [SJW07, JU06], it is still hard to extend the method to the general case since the tagging of zero-crossing edges with Hermite data assumes a fixed isovalue.

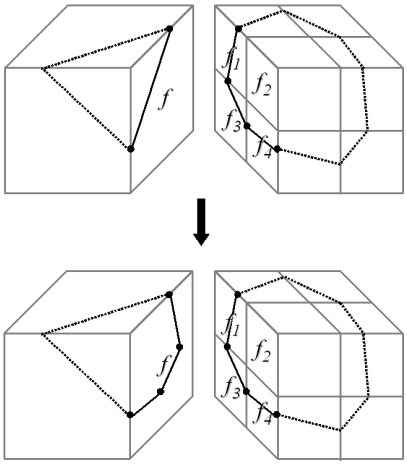

Figure 1: *A coarser leaf node with face f (left) and the finer leaf nodes adjacent across f (right): If leaf nodes are polygonized independently, edges defined by f will not align with edges defined by f_1, f_3, and f_4 (top). This cracking can be resolved by replacing the edge defined by f with the edges defined by the finer faces (bottom).*

A hybrid (dual/primal) method is proposed in [SW04]. Using the fact that the polyhedra of the dual to the partition defined by the leaves of the octree can be thought of as cubes (possibly with collapsed edges), the authors propose a setting in which the Marching Cubes algorithm can be applied to extract a watertight mesh from an octree by assigning scalar values to the interior of the leaf nodes. However, because the surface is extracted from the dual partition, vertices of the octree that are endpoints of zero-crossing edges are not guaranteed to be separated by the extracted surface, which can result in the introduction of topological artifacts.

In contrast to previous work, we show that a watertight surface can be extracted from the octree without restricting its topology, constraining the values at the vertices, or fixing the isovalue. Using a primal approach, we extract the surface directly from the samples, defining a level-set without having to infer the values of the function at new locations.

3. Isosurface Extraction

In this section, we describe a general method for extracting a watertight surface from an octree without imposing any restrictions on the topology of the tree or the values associated to its vertices. We begin by reviewing early work of Bloomenthal [Blo88] that uses an octree to obtain a polygonization of an implicit surface. Although the method assumes a fixed isovalue and constrains the octree, we show that the approach can be generalized to the context of arbitrary values and unconstrained octrees through the use of the *edge-trees* encoded by the octree. We conclude this section by describing an algorithm for computing the minimal area triangulation of the polygons, providing a way to transform the polygon mesh into a triangle mesh.

3.1. Motivation

At a high level, the method proposed in [Blo88] generates the polygon mesh corresponding to the zero-surface by associating a set of isopolygons to each leaf node independently. For each leaf node, the method iterates over the faces of the node, applying a marching-squares-like algorithm. Iso-edges separating vertices with positive value from those with negative value are defined for each face and, linking together iso-edges sharing a common isovertex, the isopolygons associated to the node are obtained.

To ensure watertightness, the algorithm is altered in the case that two face-adjacent leaf nodes are of different resolutions. As shown in Figure 1, instead of using the iso-edges computed from the coarser face, the iso-edges from the finer, face-adjacent neighbors are copied (with flipped orientation) to define the set of iso-edges associated to the coarser node.

However, after copying iso-edges from the finer nodes to the coarser one, the set of iso-edges associated to the coarser node may no longer form a closed loop. To address this concern, the implicit function is used in two ways: First, in the case that the implicit function exhibits multiple zero-crossings along an edge of a leaf node, the leaf node is refined and values are associated to the new vertices by sampling the implicit function. Second, the position of an isovertex along a zero-crossing edge is defined as the (unique) root of the implicit function along the edge.

Generalizing this solution to unconstrained surface extraction poses several challenges: (1) Since no implicit function is assumed, the positions of zero-crossings may be defined inconsistently. (2) The method can require modification of the octree's topology. And, (3) the refinement of leaf nodes is dependent on the choice of isovalue.

To resolve these challenges, we introduce *edge-trees* and show that they can be used to define a watertight polygon mesh without refining the octree or modifying its values.

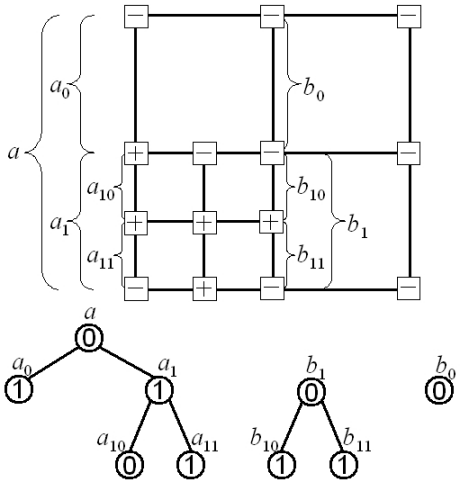

Figure 2: *A quadtree decomposition of a square (top) with three of its twelve edge-trees (bottom).*

3.2. Edge-Trees

Given an octree with scalar values associated to the vertices, we define a set of edge-trees whose nodes are in one-to-one correspondence with the edges of the nodes in the octree. The topology of the trees is inherited from the topology of the octree and, for a given isovalue, a binary flag is associated to each edge-tree node indicating whether the endpoints of the associated edge in the octree are on the same or opposite sides of the isovalue.

As an example, Figure 2 shows a quadtree decomposition of a square with three of its twelve edge-trees. The root of the quadtree defines the left edge a. The two left children of the root define the edges a_1 and a_2, children of a in the edge-tree, and the two left children of the bottom left child define the edges a_{10} and a_{11}. Note that though b_0 and b_1 lie along a single line, they are the roots of distinct edges-trees because there is no quadtree node with edge e such that $b_0, b_1 \subset e$.

To define the value associated to an edge-tree node, the values at the endpoints of its associated octree edge are compared against the isovalue. So, for example, even though there are multiple isovalue-crossings along the edge b_1, it is assigned a value of 0 since both endpoints are on the same side of the isovalue.

We observe that since the nodes of the octree define a recursive partition, the edges of the octree nodes have the property that for any two edges e_1 and e_2:

- The interiors of e_1 and e_2 do not overlap, or
- $e_2 \supseteq e_1$ and e_2 is an ancestor of e_1 in an edge-tree, or
- $e_1 \supseteq e_2$ and e_2 is a descendant of e_1 in an edge-tree.

Defining Consistent Isovertices To define the position of isovertices in a consistent manner, we observe that edge-trees satisfy the property that the binary value of an interior

node is equal to the sum (mod 2) of its children's values. Thus, if an edge e has value 1 (corresponding to an isovalue-crossing) it must have a unique child that also has value 1.

Recursively traversing down the tree, we trace the unique path $e \supseteq \cdots \supseteq \tilde{e}$ with the property that every node along the path from e to the leaf node $\tilde{e}$ has value 1. Thus, we can define the isovertex associated to e as the isovertex p defined by $\tilde{e}$. We denote this assignment as $p \in e$. (Note that using this notation, $p \in e$ and $e' \supseteq e$ does not imply $p \in e'$.)

If there is only one isovalue-crossing along e, this definition of isovertex position is consistent with the *minimal edge* definition in [JLSW02].

Figure 3 shows how this definition of isovertex position alleviates part of the difficulty in octree polygonization. Rather than using the edge e to define the isovertex position at p (top), we traverse down the edge-tree to node e_1 (middle), and use the root associated with e_1 to define the isovertex position. As a result, the faces on either side of e contribute iso-edges with p_1 as an endpoint and part of the isopolyline is closed off.

Closing the Isopolylines As shown in Figure 3 (middle), we may still need to seal the isopolylines associated to a leaf node N to obtain a set of closed isopolygons. To do this, we show that whenever the isovertex $p \in e$ has valence one, there exists a unique *twin* isovertex $p' \in e'$, with e' in the same edge-tree, that also has valence one. Thus, adding the iso-edge (p, p') to the set of iso-edges associated to N will reduce the number of valence-one isovertices along N.

To obtain the twin isovertex of p, we traverse the parents of e until we find the first edge $\tilde{e}$ which has value 0. Since $\tilde{e}$ is the first such edge, we know that both children of $\tilde{e}$ have value 1. Traversing down the second child of $\tilde{e}$ – the one that is not an ancestor of e – through nodes with value 1, we obtain the leaf node e' defining the twin isovertex p'. (We prove the existence of such an edge $\tilde{e}$ below.)

Figure 3 (bottom) shows how this definition of the twin isovertex alleviates the difficulty in octree polygonization. Identifying the isovertex p_{00} associated to e_{00} as an endpoint of an isopolyline, we trace through the ancestors of e_{00} to get to the edge e_0, the first edge which has value 0. Following the other child of e_0 we arrive at the edge e_{01} defining the twin isovertex p_{00}. Adding the edge from p_{00} to p_{01}, the isopolylines become isopolygons, as desired.

Since the number of isovertices along an edge with valence one must always be even, in the case of a restricted octree [WKE99], there can be at most two valence-one isovertices and the twin of p can simply be defined as the only other valence-one isovertex p' along the same edge.

Proof of Correctness For the extraction algorithm to result in a watertight polygonal mesh, the polygons in the mesh must be shown to satisfy two properties: (1) The extracted isopolylines must be closed loops, and (2) each edge in the polygon mesh must be shared by exactly two isopolygons.

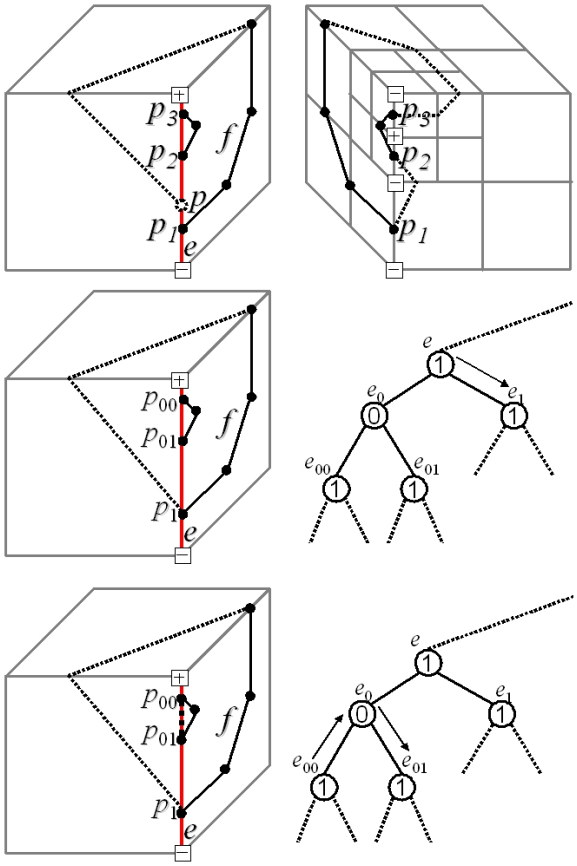

Figure 3: *In the case that the edge e exhibits more than one sign change, the polygonization method of [Blo88] fails (top). Using the edge-trees to define the position of the isovertices in a consistent fashion (middle), and pair up open isovertices (bottom) results in a valid polygonization.*

Closedness

We prove that the isopolylines are closed by considering the set of iso-edges associated to a leaf node of the octree, and showing that every endpoint has valence two.

Consider the case when N is a leaf node in the octree containing an isovertex p, defined by the leaf $e \ni p$ in the edge-tree. In this case, there are two leaf nodes N_1 and N_2 in the octree which can define iso-edges on N containing p.

These are the nodes that are either face-adjacent to N (if N is coarser than N_i) or equal to N, and lie on opposite sides of the edge e. Formally, these are defined to be the leaf nodes containing faces $f_i \subset \partial N_i$ such that: (1) $f_1 \neq f_2$, (2) $f_i \subset \partial N$, (3) there exists an edge $e_i \subset \partial f_i$ with $e_i \supseteq e$, and (4) the neighbor of N_i across f_i is a leaf node no finer than N_i.

The isovertex p is an endpoint of an isopolyline on N if and only if only one of the e_i satisfies $p \in e_i$. Without loss

of generality, we assume $p \in e_1$ and separately consider the cases $e_1 \supseteq e_2$ and $e_2 \supset e_1$.

Case 1 ($e_1 \supseteq e_2$): Since $p \in e_1$, all the edges along the chain $e_1 \supseteq \cdots \supseteq e$ must have value one. Since $e_1 \supseteq e_2 \supseteq e$, e_2 must be an element of the chain and hence must also have p as an isovertex, contradicting our initial assumption.

Case 2 ($e_2 \supset e_1$): Since $p \notin e_2$, there must exist an edge $\tilde{e}$ with value 0 along the chain $e_2 \supset \cdots \supset e_1$. Thus, the twin p' of p is well-defined and satisfies $p' \in e'$ for some $e_2 \supset e'$.

Denoting by N_1' and N_2' the leaf nodes in the octree which can define iso-edges on N containing p', it is not hard to show that (up to re-indexing) $e_2 = e_2'$ and $\tilde{e} \supset e_1'$. Since $e_1' \supseteq e'$, the edge e_1' must lie on the chain $\tilde{e} \supset \cdots \supset e'$, whose elements all have value 1 except $\tilde{e}$, and therefore $p' \in e_1'$. Additionally, since $e_2 \supseteq \tilde{e}$ and since $\tilde{e}$ has value 0, we know that $p' \notin e_2$, hence p' must also be have valence one. Thus, adding the iso-edge (p, p') will have closed the isopolyline at p, contradicting the assumption that p has valence one.

Edge Manifoldness

We demonstrate that anytime we close an isopolygon by adding an edge between an isovertex and its twin, that new iso-edge will be shared by exactly one other isopolygon. (Since the initial set of iso-edges is obtained by copying from finer leaf nodes to coarser leaf nodes across a common face, we know that these edges must be shared by exactly two isopolygons.)

To show the above, we prove that if the isovertex p is an endpoint of an isopolyline associated to a leaf node N, then p is also an endpoint of an isopolyline associated to exactly one other leaf node N'. Thus, the edge (p, p') will appear in exactly two isopolygons.

We consider the two cases in which an isovertex $p \in e$ can exist: Either the (edge-tree leaf) e is adjacent to three octree nodes, or it is adjacent to four. Figure 4 shows these two configurations, with the bottom row providing a visualization of the nodes looking down the e-axis.

In both cases, a simple counting argument shows that if p is an endpoint of an isopolyline along N_i then there exists one and only one other node N_j (with $i \neq j$) such that p is also an endpoint of an isopolyline along N_j.

To count the number of nodes adding the iso-edge between p and its twin p', we begin by associating a binary value n_i to each node N_i, indicating whether there exists an edge $e_i \subset \partial N_i$ such that $p \in e_i$. Next, we associate a value f_{ij} to each pair of face adjacent nodes N_i and N_j, indicating the existence of an iso-edge contained in the shared face between N_i and N_j which has p as an endpoint – setting f_{ij} equal to 1 if either n_i or n_j is equal to one. Finally, we associate a value c_i to each node N_i, indicating whether p is an endpoint of an isopolyline associated to the node N_i – setting c_i equal to 1 only if one of f_{ij} and $f_{ij'}$, for nodes N_j and $N_{j'}$, face adjacent to N_i, is equal to one.

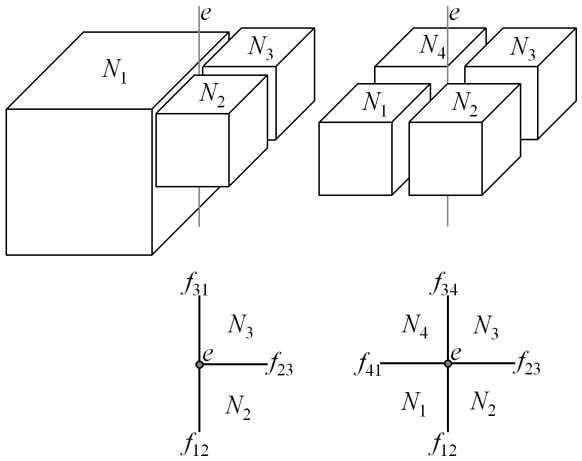

Figure 4: *The two configurations in which an isovertex $p \in e$ can exist: Either e is adjacent to three leaf nodes in the octree (left), or it is adjacent to four (right).*

The values of c_i for the two cases are shown in Table 1, which is abridged by using the symmetries of the rows under reflection and rotation. Additionally, we omit the case $n_1 = 1$ for the three-node case since e cannot lie on an edge of N_1.

Three-Node Case

n_1	n_2	n_3	f_{12}	f_{23}	f_{13}	c_1	c_2	c_3
0	0	0	0	0	0	0	0	0
0	1	0	1	1	0	1	0	1
0	1	1	1	1	1	0	0	0

Four-Node Case

n_1	n_2	n_3	n_4	f_{12}	f_{23}	f_{34}	f_{14}	c_1	c_2	c_3	c_4
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1	0	1	1	0
1	1	0	0	1	1	0	1	0	0	1	1
1	0	1	0	1	1	1	1	0	0	0	0
1	1	1	0	1	1	1	1	0	0	0	0
1	1	0	1	1	1	1	1	0	0	0	0
1	1	1	1	1	1	1	1	0	0	0	0

Table 1: *The values of c_i, indicating if the isovertex p is an endpoint of an isopolyline associated the leaf node N_i.*

As the tables indicate, given the configuration of nodes shown in Figure 4, either the edge from p to its twin p' is never added ($\sum c_i = 0$), or it is added exactly twice ($\sum c_i = 2$), and the iso-edges in the mesh satisfy the condition that each one is shared by exactly two isopolygons.

3.3. Polygon Triangulation

Given the polygon mesh extracted from the octree, we obtain a triangle mesh by independently triangulating each of the individual isopolygons.

In general, the problem of triangulating a 3D polygon is both open and hard [BDE98]. The problem is open because even determining whether the polygon can be triangulated – i.e. whether the polygon boundary is knotted – is an open problem. The problem is hard because it can be shown that even if the polygon boundary is not knotted, there exist vertex configurations that require the introduction of exponentially many new vertices in order to obtain a triangulated surface that is not self-intersecting.

What makes *our* triangulation problem tractable is the observation that the isopolygons obtained in the extraction step are well-behaved. Specifically, each of the isopolygons forms a simple closed curve on the surface of a convex body (the nodes of the octree).

Recent work in differential geometry [MY80, MY82] shows that for a simple, closed curve on the boundary of a convex body, the minimal area surface bounded by the curve and homeomorphic to a disk is not self-intersecting. Guided by this result, we triangulate an isopolygon by finding the minimal area triangulation homeomorphic to a disk. It is important to note, however, that our triangulations are not minimal area surfaces, only minimal area triangulations. Thus, though we have found that in practice the extracted triangulations do not self-intersect, this is not guaranteed.

To compute the minimal area triangulation, we use a method from [BS95] which, for completeness, we describe below.

Algorithm

To compute the minimal area triangulation of a polygon $P = \{p_1, \ldots, p_n\}$, two observations can be made. First, because the triangulation is homeomorphic to a disk, it follows that if the edge between p_i and p_j is in the minimal area triangulation, then the induced triangulations of the two sub-polygons $P_{i,j} \equiv \{p_i, \ldots, p_j\}$ and $P_{j,i} \equiv \{p_j, \ldots, p_i\}$ must also have minimal area. Second, since for any p_i the edge between p_i and p_{i+1} must be a boundary edge of the triangulation, there must exist a vertex p_k such that triangle $T_{i,i+1,k} \equiv \{p_i, p_{i+1}, p_k\}$ is in the triangulation.

Thus, the minimal area triangulation can be determined by computing the minimal area triangulations of nested sub-polygons of P. Defining $A(i,j,k)$ to be the area of the triangle $T_{i,j,k}$ and $M(i,j)$ to be the area of the minimal area triangulation of $P_{i,j}$, this leads to a simple recursive definition for the value of $M(i,j)$:

$$M(i,j) = \begin{cases} 0 & \text{if} \quad i = j \\ 0 & \text{if} \quad i = j+1 \bmod n \\ \min\limits_{k \in (j,i)} M(i,k) + M(k,j) + A(i,j,k) \end{cases}$$

Caching the 2D array of values (and corresponding choice of k) as values of $M(i,j)$ are computed makes it possible to compute M in $O\left(n^3\right)$ time and $O\left(n^2\right)$ space. The minimal area triangulation can then be determined by starting at the entry $M(1,2)$ and iteratively following the k-values to obtain the triangles.

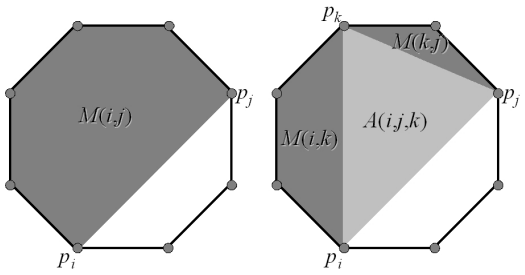

Figure 5: *To compute the minimal area triangulation, we compute the minimal area $M(i,j)$ of the sub-polygons $P_{i,j}$ (left). We do this recursively by iterating over the vertices p_k between p_j and p_i, and finding the vertex minimizing the sum of areas $A(i,j,k) + M(i,k) + M(k,j)$ (right).*

4. Results

Using the edge-trees, we have shown that it is possible to extract an isosurface from an octree without requiring either refinement of the tree or modification of the values associated to the vertices. In this section, we evaluate our approach in the context of mesh simplification by considering the implications of octree refinement on the simplification and by comparing our results to those obtained using dual marching cubes.

For a given 3D surface, we generate an adaptive octree by starting with the root node and recursively refining nodes that intersect the surface and whose intersection with the surface is not "too flat". We assign a value to the vertices of the octree by sampling the Euclidean distance transform. For simplicity, we define the measure of flatness to be the ratio of the length of the mean curvature vector of the intersected surface to its area.

4.1. The Cost of Octree Refinement

To evaluate the effects of octree refinement, we modify the construction of the tree to ensure that the octree is restricted so that the depth-disparity between adjacent nodes is never greater than one ([WKE99]). We do this by further refining the leaves of the octree in the case that they are too coarse.

Figure 6 shows the results of our experiment for the simple case of a cube. We compare the surface returned when the octree is adapted to the flatness of the model and is refined to satisfy the depth-disparity constraint (left), and the surface returned when the octree is adapted to the flatness of the model without refinement (right). Results for analogous experiments on more complex models are shown in Figure 8.

Although the difference between the surfaces extracted using the constrained and the unconstrained octrees is barely

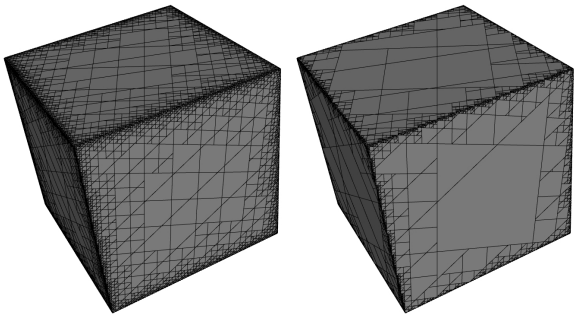

Figure 6: *Isosurfaces extractions obtained using a constrained adaptive octree (left), and an unconstrained adaptive octree (right), showing the underlying polygon mesh. Though both approaches extract surfaces preserving sharp edges, a constrained octree results in many more triangles in flat regions.*

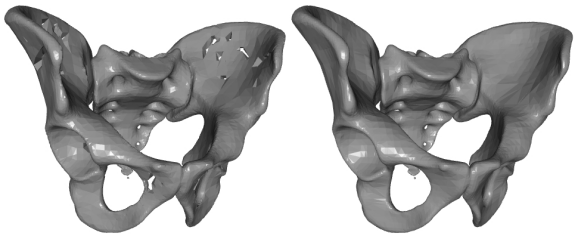

Figure 7: *Reconstructions of a pelvis obtained using the dual marching cubes algorithm (left) and using our method (right). While both methods generate watertight surfaces from an unconstrained octree, only primal methods guarantee that the surface passes through all isovalue-crossing edges in the octree. One possible consequence, topological artifacts, can be observed here.*

perceptible, the computational cost and memory overhead is more apparent. These statistics are summarized in Table 2 which compares the octree complexity (in nodes), surface complexity (in vertices), and extraction time (in seconds). As the table indicates, constraining the octree limits the simplification process and results in surfaces satisfying the same flatness criteria but with 20% more vertices. More significantly, we find that using the constrained octree more than doubles the average number of nodes needed to represent the sample data, forcing an unnecessary increase in both memory and computation time.

4.2. Comparison to Dual Marching Cubes

One of the motivations for using a primal approach for surface extraction is that, because an isovertex is introduced along every isovalue-crossing edge, the isosurface is guaranteed to separate samples that lie on opposite sides of the isovalue. In contrast, methods such as dual marching cubes [SW04] which extract a surface from an induced partition do not guarantee that endpoints of an octree-edge on opposite sides of the isovalue will be separated by the extracted surface.

As an example Figure 7 compares the surfaces extracted from an adaptive octree sampling the Euclidean distance transform of a pelvis. In the left image, we see the surface obtained using dual marching cubes, where the vertex associated to the interior of a node is defined as the node's center and the value is obtained by explicitly sampling the distance transform. In the right image, we see the surface extracted using our primal method.

In regions such as the ilium, the flatness of the surface implies that the adaptive octree is much coarser than the thickness of the model and the dual partition does not generate samples interior to the surface. As a result, despite the existence of vertices in the primal representation that sample the interior of the pelvis, dual marching cubes generates a reconstruction with topological holes.

Although the performance of dual marching cubes could be improved by using an implicit function to refine the octree (e.g. [VKZM06]), in these experiments we focus on evaluating the extraction performance in the case of unconstrained octrees, and do not adapt the topology to dual marching cubes.

5. Conclusion

In this work, we have provided a novel algorithm for extracting an isosurface from an octree. Introducing the notion of *edge-trees*, we address the traditional problems in octree-based surface extraction, providing a method for directly extracting a watertight mesh without constraining the topology of the octree or modifying vertex values.

References

[BDE98] BAREQUET G., DICKERSON M., EPPSTEIN D.: On triangulating three-dimensional polygons. *Computational Geometry 10*, 3 (June 1998), 155–170.

[BGOS06] BARGTEIL A., GOKTEKIN T., OBRIEN J., STRAIN J.: A semi-Lagrangian contouring method for fluid simulation. *ACM Transactions on Graphics 25* (2006), 19–38.

[Blo88] BLOOMENTHAL J.: Polygonization of implicit surfaces. *Computer Aided Geometric Design 5* (1988), 341–355.

[BS95] BAREQUET G., SHARIR M.: Filling gaps in the boundary of a polyhedron. *Computer Aided Geometric Design 12*, 2 (1995), 207–229.

[HWC*05] HO C., WU F., CHEN B., CHUANG Y., OUHYOUNG M.: Cubical marching squares: Adaptive feature preserving surface extraction from volume data. In *Proceedings of EUROGRAPHICS 2005* (2005), pp. 537–545.

	Marching Cubes			Constrained Octree			Unconstrained Octree		
Model	Res.	Verts.	Time	Nodes	Verts.	Time	Nodes	Verts.	Time
Cube	128^3	37,944	0.23	21,689	7,354	0.08	9,025	5,512	0.05
	256^3	153,448	1.13	49,345	16,350	0.24	19,097	11,545	0.09
	512^3	617,210	7.13	105,225	34,582	0.48	38,897	23,818	0.27
Armadillo	128^3	17,424	0.22	17,945	6,326	0.08	9,961	5,956	0.08
	256^3	73,634	0.94	49,809	17,436	0.28	24,185	15,530	0.17
	512^3	300,216	6.19	95,081	33,003	0.55	44,033	28,450	0.39
Dragon	128^3	11,198	0.19	15,697	5,643	0.08	9,345	5,433	0.05
	256^3	49,850	0.86	45,841	16,618	0.25	25,401	15,375	0.19
	512^3	205,456	5.92	82,521	28,797	0.47	42,273	26,025	0.33
Bunny	128^3	19,842	0.14	15,889	5,665	0.08	8,121	5,189	0.05
	256^3	80,676	0.84	34,081	11,858	0.17	15,545	9,940	0.13
	512^3	325,574	6.14	61,281	20,581	0.36	24,969	16,244	0.22

Table 2: *A comparison of octree isosurface extraction at different resolutions based on: octree complexity (in nodes), surface complexity (in vertices), and extraction time (in seconds). By all three criteria, our approach improves upon previous work.*

[JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J.: Dual contouring of Hermite data. *ACM Transactions on Graphics (SIGGRAPH '02) 21* (2002), 339–346.

[JU06] JU T., UDESHI T.: Intersection-free contouring of an octree grid. In *Proceedings of Pacific Graphics* (2006).

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. In *Symposium on Geometry Processing* (2006), pp. 73–82.

[KBSS01] KOBBELT L., BOTSCH M., SCHWANECKE U., SEIDEL H.: Feature-sensitive surface extraction from volume data. In *Computer Graphics (Proceedings of SIGGRAPH 01)* (2001), pp. 57–66.

[LC87] LORENSEN W., CLINE H.: Marching cubes: A high resolution 3d surface reconstruction algorithm. In *Computer Graphics (Proceedings of SIGGRAPH 87)* (1987), pp. 163–169.

[LGF04] LOSASSO F., GIBOU F., FEDKIW R.: Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics (SIGGRAPH '04) 23* (2004), 457–462.

[MS93] MÜLLER H., STARK M.: Adaptive generation of surfaces in volume data. *The Visual Computer 9* (1993), 182–199.

[MY80] MEEKS W., YAU S.: Topology of three dimensional manifolds and the embedding problems in minimal surface theory. *The Annals of Mathematics 112*, 3 (November 1980), 441–484.

[MY82] MEEKS W., YAU S.: The existence of embedded minimal surfaces and the problem of uniqueness. *Mathematische Zeitschrift 179*, 2 (June 1982), 151–168.

[NH91] NIELSON G., HAMANN B.: The asymptotic decider: Resolving the ambiguity in marching cubes. In *IEEE Visualization* (1991), pp. 83–91.

[OBA*03] OHTAKE Y., BELYAEV A., ALEXA M., TURK G., SEIDEL H.: Multi-level partition of unity implicits. *ACM Transactions on Graphics* (2003), 463–470.

[OR97] OHLBERGER M., RUMPF M.: Hierarchical and adaptive visualization on nested grids. *Computing 59* (1997), 365–385.

[SFYC96] SHEKHAR R., FAYYAD E., YAGEL R., CORNHILL J.: Octree-based decimation of marching cubes surfaces. In *IEEE Visualization* (1996), pp. 335–342.

[SJW07] SCHAEFER S., JU T., WARREN J.: Manifold dual contouring. In *IEEE Transactions on Visualization and Computer Graphics* (2007), pp. 610–619.

[SW04] SCHAEFER S., WARREN J.: Dual marching cubes: Primal contouring of dual grids. In *Proceedings of Pacific Graphics* (2004), pp. 70–76.

[SZK95] SHU R., ZHOU C., KANKANHALLI M.: Adaptive marching cubes. *The Visual Computer 11* (1995), 202–217.

[VKSM04] VARADHAN G., KRISHNAN S., SRIRAM T., MANOCHA D.: Topology preserving surface extraction using adaptive subdivision. In *Symposium on Geometry Processing* (2004), pp. 235–244.

[VKZM06] VARADHAN G., KRISHNAN S., ZHANG L., MANOCHA D.: Reliable implicit surface polygonization using visibility mapping. In *Symposium on Geometry Processing* (2006), pp. 211–222.

[WG92] WILHELMS J., GELDER A. V.: Octrees for faster isosurface generation. *ACM Transactions on Graphics 11* (1992), 201–227.

[WKE99] WESTERMANN R., KOBBELT L., ERTL T.: Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer 15* (1999), 100–111.

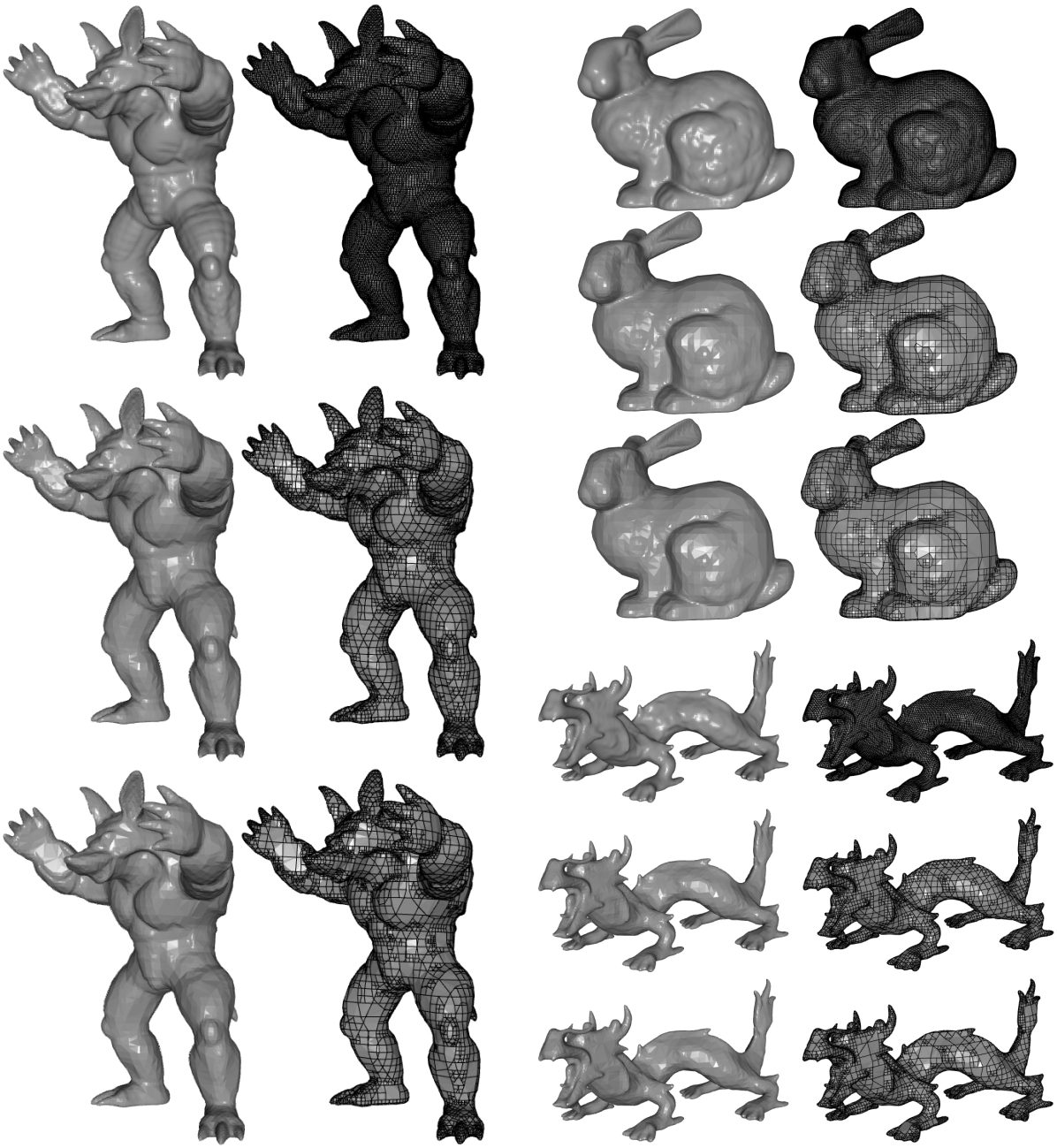

Figure 8: *Additional comparisons of isosurface extraction methods. For each model, the top row shows results obtained using marching cubes on a regular grid, the middle row shows results from the constrained octree extraction, and the bottom row shows results from our unconstrained method. In each case, our method preserves surface detail while using fewer vertices.*

[WMW86] WYVILL G., MCPHEETERS C., WYVILL B.: Data structures for soft objects. *The Visual Computer 2* (1986), 227–234.