Discrete k-d Tree Hierarchy for Isosurface Extraction

ANONYMOUS AUTHOR(S)

We present a discrete k-d tree hierarchy for isosurface extraction. By defining and evaluating a per-dimensional version of the quadratic error function, we construct our k-d tree directly from the volume data to be visualized, thus achieving better spatial partitioning than is possible with octrees. An efficient approach for directly extracting a polygon from the k-d tree is also developed. Since the nodes of the k-d tree are axis-aligned, we can naturally extend a topology-preserving simplification algorithm and a manifold-guaranteeing polygonization algorithm, with some modifications, to the new k-d tree hierarchy.

CCS Concepts: • **Computing methodologies → Volumetric models**;

Additional Key Words and Phrases: isosurface, k-d tree, spatial partition, contour simplification

ACM Reference format:
Anonymous Author(s). 2021. Discrete k-d Tree Hierarchy for Isosurface Extraction. 1, 1, Article 1 (April 2021), 8 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

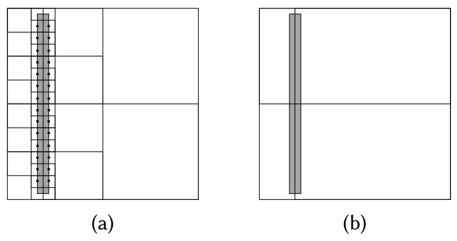


Fig. 1. Adaptive contouring of a 2D thin object based on an quadtree (a) and a ideal k-d tree (b). Redundant vertices are drawn as black round points.

1 INTRODUCTION

The modeling of the isosurface of a volume dataset has broad applications, such as visualizing scanned medical data and physical simulations or providing a foundation for subsequent work such as finite element analysis and numerical simulations. Usually, a volume dataset can be regarded as a 3D scalar field representing an implicit function $f : R \leftarrow R^3$. The actual meaning of the value of this function can be the temperature, speed or density at the corresponding position. Extracting a level set of f is the process of generating a piecewise linear approximation of the equation of the form $f = c$, where c denotes a constant value.

Typically, the scalar field is uniformly sampled using a structured grid. However, to account for the rapid growth of the volume resolution, an efficient adaptive representation needs to be constructed. Previously, octrees have been widely used for adaptive isosurface extraction. Despite its convenience for polygon generation, however, an octree hierarchy is rarely a good spatial partition since each node in an octree is always divided into eight equal subnodes splitting out from the middle of a cube. This middle-divided hierarchy frequently introduces many unnecessary intermediate nodes and redundant triangles, as shown in figure 1.

Although some extensions has improved the adaptability of the octree, the amount of elimination is still limited. For instance, contouring a list of adjacent thin objects requires a deep subdivision of octree nodes for methods obtaining multiple contour components in a cell, as the number of contour components is strictly limited(shown in figure 2). Also, when representing a thin-and-flat object near to an another rough surface using algorithms above, either the rough surface pollute the thin object, or the thin boundary limit the simplification of its flat surface, as show in figure 3.

[Gress and Klein 2003] attempted to build a well-balanced k-d tree hierarchy of volume data from polygon inputs. Their k-d tree hierarchy can generate a more compact reconstruction result of the original input than octrees. However, since the volume data of interest may come from a variety of sources, including computed tomograpy (CT) scanners, magnetic resonance imaging (MRI) scanners, and constructive solid geometry (CSG) trees, their algorithm is not suitable for general volume datasets.

Contributions

In this paper, we propose a k-d tree hierarchy for the direct extraction of isosurfaces from volume datasets. Our main contributions are the following:

- A theoretical error-driven subdivision plane determination algorithm for building a k-d tree that is representative of a volume dataset. Our construction algorithm shows a good sense of feature distribution per dimension, thus yielding a better adaptability.
- A topology-preserving k-d tree node clustering algorithm.
- An algorithm for manifold-guaranteeing polygonization from the proposed k-d tree hierarchy.

2 RELATED WORKS

Given a uniform grid with a signed scalar value at each grid point, there are two main schemes for extracting an isosurface. The Marching Cubes (MC) scheme [Lorensen and Cline 1987] considers sign changes at each grid edge and places a vertex exactly at the intersection point of the zero-level isosurface and the grid edge. To generate polygon meshes, a look-up operation is performed to select a polygonization strategy for each single cell from among 16 cases (256 cases after rotation). Some variants of the MC scheme include spatial partitioning with tetrahedra [Doi and Koide 1991] and with

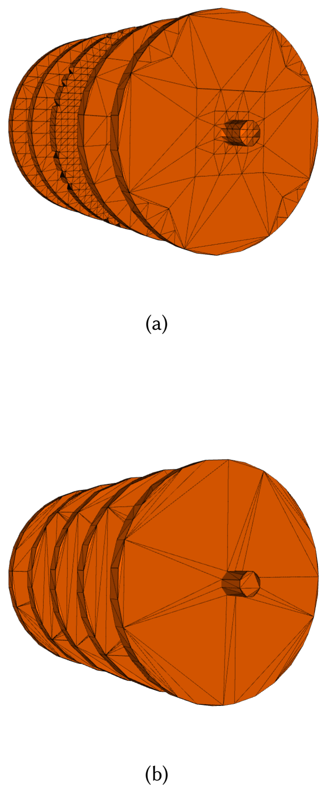


Fig. 2. A list of a disk connected with a pole defined by CSG. The polygonal surfaces are generated by Manifold Dual Contouring using an octree (a, 7868 triangles) and by our algorithm using a discrete k-d tree (b, 990 triangles)

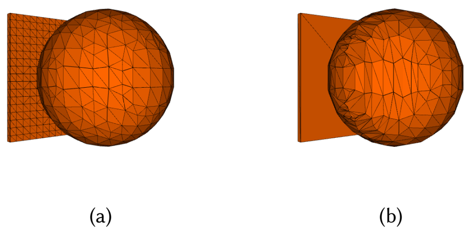


Fig. 3. Contour results obtained using an octree (a) with 1,952 triangles and an discrete k-d (b) tree with 856 triangles under an error threshold of 1e-1.

pentahedra [Narayan et al. 2018]; both reduce the number of cases considered and avoid ambiguous contouring.

In contrast, the Dual Contouring (DC) scheme [Ju et al. 2002] creates one vertex for each heterogeneous cell instead of each signed edge. The main advantage of DC is its ability to reproduce sharp features. By utilizing surface normal information, it is possible to place a vertex exactly at the feature point of the isosurface inside each cell. After that, it is simple to connect adjacent node vertices to form the edges of the extracted isosurface. By iterating on every signed grid edge, the DC scheme connects four adjacent nodes to efficiently generate a quad.

With regard to adaptive contouring, the algorithms for this purpose, including both dual and primal methods, apply bottom-up clustering by collapsing the octree nodes throughout the hierarchy. As a result, all cubes at the same resolution level possess edges of equal length in all three dimensions. For primal methods [Shekhar et al. 1996; Shu et al. 1995] based on the famous MC algorithm [Lorensen and Cline 1987], each signed edge of each leaf node of the octree contains a vertex. Eight child cells can be replaced with a single parent cell under certain conditions. After polygonization in multi-resolution cells, an additional patching process to address discontinuities between cubes of different resolutions is needed [Shekhar et al. 1996; Shu et al. 1995]. For dual methods [Ju et al. 2002], each inhomogeneous octree node contains a representative vertex, and clustering results in approximate vertices representing coarser nodes. The polygonization process is no different from that in the uniform situation except that 2 of the 4 adjacent nodes during quad generation may be the same node. In this situation, a triangle is generated rather than a quad.

To overcome some of the drawbacks of these 2 main schemes for isosurface contouring, several variants have been introduced. Before reviewing these works, we need to first address 2 main problems that arise during node clustering:

(1) **Better adaptability of surface features.** Volume datasets are often unevenly distributed in R^3, causing imbalance of the octree. Furthermore, the spatial structure of an octree limits the adaptability of the contours. For example, for a thin object, the simplification process will terminate early and then generate redundant vertices.
(2) **Preservation of the 2-manifold and topology during clustering.** Applications such as finite element analysis and 2D texture mapping require the model to be a 2-manifold and to have no self-intersections. However, the dual scheme can easily generate a model with no manifold edges and vertices.

A good approach to handling both problems 1 and 2 is to store more than one vertex per uniform grid cell, as done by [Schaefer et al. 2007], [Zhang et al. 2004] and [Nielson 2004]; in this case, some limitations of cube-sampled cells will be eliminated (e.g., the inability to contour thin and sharp objects). Moreover, since the surface generated by the MC scheme from a uniform grid is a 2-manifold, its dual surface will also be a manifold. However, redundant vertices can still be introduced by the middle-divided octree hierarchy. Schaefer proposed the idea of performing spatial partitioning with a dual grid, resulting in a noncubical spatial partition [Lewiner et al. 2010; Schaefer and Warren 2004]; the resulting surface is guaranteed to be a 2-manifold and offers better adaptability than an octree, but it may be too expensive to compute, especially in the case of multiple different level thresholds (the dual grid needs to be rebuilt for each threshold).

Gress proposed a spatial partitioning approach based on a k-d tree as a volume method for polygon remeshing [Gress and Klein

2003]. By evaluating an error metric for polygon inputs as proposed by [Lindstrom 2000], they split each node of the k-d tree with a plane passing through or nearby the corresponding feature point. However, their method cannot accommodate other types of input.

3 CONTOURING ON A UNIFORM GRID

We start from Nieson's approach [Nielson 2004] with Lobello's fix [Lobello et al. 2018] to contouring different components of a single cell. The surface generated here is dual to the surface generated by the original MC scheme. Unlike the MC scheme, Nieson's approach considers only one vertex per component (called a dual vertex). By iterating on each active edge of the uniform grid, the associated components of 4 adjacent cells are connected to form a quad. As proven in [Nielson 2004], there is a one-to-one correspondence between the active edges of the uniform grid and the dual vertices. The algorithm thus produces a surface dual to the uniform grid.

By incorporating Hermite data, we obtain the Hermite Extension of Dual Marching Cubes, which simultaneously handles multiple vertices per cell and preserves the sharp features of the isosurface. The position of the representative vertex of each surface component inside a cell is determined by minimizing a quadratic error function (QEF) [Ju et al. 2002] corresponding to each intersection point between that surface component and the edges of the lattice cell.

Note that since every component intersects with at least 3 edges of the cell, the representative vertex is most likely located inside the cell. If it is not, then following the suggestion of [Ju et al. 2002], we project the vertex to the cell surface.

Our next task is to find all intersection points of a single component and combine them to build a QEF. For convenience in subsequent operations, we store a list of components directly within each cell. We treat all adjacent positive corners as belonging to a single component. By simply iterating on all homogeneous edges of an inhomogeneous cell, all separate cell components should be well clustered. After that, we check the remaining inhomogeneous edges and build QEFs in the proper order. With the resulting independent QEFs for each component, we finally compute the best position for each vertex as the exact point that minimizes the QEF. To generate the final mesh, we need to store indices pointing to those components for each signed edge. Thus, when iterating on every active edge of the grid, we can easily find each related component of each set of 4 adjacent cells.

4 DISCRETE K-D TREE

Traditional contouring methods for scalar fields often take advantage of an octree hierarchy. Such an octree hierarchy is actually a mipmap of a uniform grid. Each coarser cell is a generated cluster consisting of 8 children. The edge length of an octree cell will be 2^n $(n \geq 0)$ times that of the original uniform grid edge. If a single uniform grid cell is regarded as the minimum unit, a full octree of depth n has a resolution of $2^n \times 2^n \times 2^n$. If the limitation of equal lengths in each dimension is removed, then cuboid cells with dimensions of $n \times m \times k$ can also be well clustered. Consider such a cell consisting of 2 cuboid subcells with dimensions of $(n-j) \times m \times k$ and $j \times m \times k$; we can recursively define a k-d tree hierarchy whose axis-aligned splitting plane positions are discrete, which we call a discrete k-d tree. One of the greatest advantages of a discrete k-d tree is there is no need to resample from the original data. One can simply make use of the already sampled values from the uniform grid, or the finest level of an octree, as some classic algorithms have previously done.

Because the nodes in a k-d tree are usually cuboid cells with different sizes in different dimensions, all of the sampled data inside each rectilinear cell need to be summed up. Also, to choose the best subdivision plane and axis for a coarse k-d node, the division plane that best approximately divides the cell into 2 equal-error nodes needs to be determined. Therefore

4.1 3-Dimensional Quadratic Error Function

Given a set of n sample points $S(p) = p_0, p_1, \ldots, p_i, \ldots, p_n$, following [Ju et al. 2002], we find the best approximate vertex position by minimizing the QEF

$$E_1(x) = \sum \|\nabla f(p_i) \cdot (x - p_i)\|^2$$

or, written in matrix form,

$$E_1[x] = x^T A^T A x - 2x^T A^T b + b^T b$$

where p_i denotes the position vector of a sample point, A is a matrix whose rows are $\nabla f(p_i)$, and b is a vector whose entries are $n_i \cdot p_i$.

Note that $E_1(x)$ is actually a sum of dot products representing the general error, with no dimensional distinction. If we consider a per-dimensional version of the QEF, separating the x, y, and z dimensions yields a new expression for the error function: $E_2(x) = \sum (n_i \otimes (x - p_i))^2$. Here, $E_2(x)$ is a 3D vector, where each component of $E_2(x)$ represents the general error in the corresponding dimension. This can be useful because when choosing the best axis for the subdivision plane of a k-d tree node, the dimension with the maximum variance should be selected. We can rewrite $E_2(x)$ in matrix form as follows:

$$E_2[X] = diag((A \otimes x - C)^T (A \otimes x - C))$$
$$= diag(x^T \otimes A^T A \otimes x) - 2diag(x^T \otimes A^T C) - diag(C^T C)$$

where C is a matrix whose rows are $\nabla f(p_i) \otimes p_i$. To compute $E_2(x)$, three 1×3 vectors, $diag(A^T A)$, $diag(A^T C)$, and $diag(C^T C)$, need to be stored. Since $A^T A$ is already stored for E_1, there are only 6 additional floats for each QEF. Compared with the the original QEF storage requirement [Ju et al. 2002] (10 floats), the new metric imposes an acceptable linear memory consumption increase of 60%.

Taking advantages of the per dimensional version of QEF, Our contouring result become sensitive to feature distribution upon 3 dimensions. Figure 4 shows a competitive result of representing objects with unblanced feature distributions with octree and kdtree

Note the following important property of E_1: when each representative vertex is placed at the minimum position of the corresponding QEF, it is guaranteed that a combined QEF consisting of n QEFs will obtain an error larger than that of any of the n individual QEFs; this can be proven below:

Let x_{min} denotes the ideal minimize point of E_1, we have:

$$E_1(x) \geq E_1(x_{min})$$

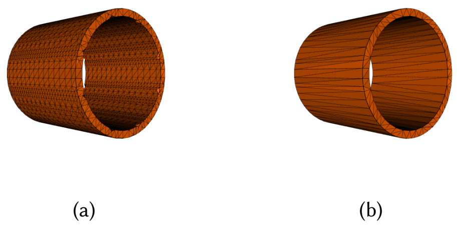


Fig. 4. Contour results for a scalar field with a different distribution in each dimension. The k-d tree result (b) with 384 triangles shows good sensitivity to the per-dimensional error, while the octree result (a) with 6912 triangles is limited both by the thin feature and knowing no knowledge about per-dimensional error metric E_2

let p_{n+1} denotes an extra sampler point, E_1' denotes the combined QEF of set $S \cup p_{n+1}$, and let x'_{min} denotes the minimize point of E_1', we have

$$E_1'(x'_{min}) = \|\nabla f(p_{n+1}) \cdot (x'_{min} - p_{n+1})\|^2 + \sum^{n} \|\nabla f(p_i) \cdot (x'_{min} - p_i)\|^2$$
$$= \|\nabla f(p_{n+1}) \cdot (x'_{min} - p_{n+1})\|^2 + E_1(x'_{min}) \geq E_1(x_{min})$$

knowing the property above, the combined QEF of 2 cuboid regions should be greater than or equal to the 2 original QEFs. The property provides us with the possibility of performing a binary search along any axis of an n-dimensional cuboid region, since can accelerate our subdivision-plane-finding process described below, as we will discuss in the next section.

4.2 Interval Summation

Consider the problem of obtaining summed QEFs in a given $n \times m \times k$ region, where m, n, and k denote the side lengths of a cuboid in the x, y, and z directions, respectively. The most naive idea would be to directly iterate on all homogeneous, identical nodes inside this region. This process has a time complexity of O(n). Using a 3D summed area table, this iteration could be done in O(1) time, but it would require a total memory size equivalent to an $n \times m \times k$ uniform grid, which would violate the principle of sparse representation of the scalar field. In fact, the widely used octree hierarchy provides us with a sparse structure with O(log(n))-level acceleration. We first compute the mipmap of the uniform grid, yielding an octree whose leaf and internal nodes contain the summation results for the areas covered by those nodes. The summation process is performed recursively from the root node to a node at a level such that the given area is fully covered. If the intersection of the given area and an octree node is equal to the node area, we add the QEF of this node to the result and terminate the recursive call. If the intersection is not fully equal to the node area but has a volume greater than 0, we recursively call the summation process for the node's eight children. If the intersection has a volume of 0, we do nothing and terminate the process. For simplicity, a 2d comparison of calculating a interval summation of a rectangular region upon a uniform grid and a quadtree is shown in figure 5

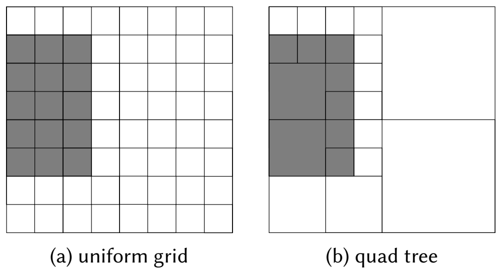


Fig. 5. Comparison of interval summation based on a 2D uniform grid (a) and a quad tree (2D mipmap) (b).

4.3 Construction of a Discrete k-d Tree

In this section, we can finally introduce our top-down manner of building the entire discrete k-d tree. Starting from a scalar field S with dimensions of $n \times m \times k$ represented in the form of a uniform grid or a sparse octree, our algorithm first builds a mipmap of S, which is actually an octree, as described above. Then, we start our recursive construction of the discrete k-d tree. Given the $n \times m \times k$ region associated with the scalar field, our task is to find the proper axis (either x, y, or z) and discrete position of a plane with which to subdivide it. To determine axis, the dimension with the maximum variance is preferred. If the length of the region in maximum-variance axis is 1, which can't be futher divided, then we continue to test axes with the second and the last maximum variance, util the region reaches a resolution of 1×1×1. Regarding the choice of position, suppose that we have chosen axis 0; there are $n-1$ discrete possible positions inside the cuboid. We therefore perform a binary search along the chosen axis. In every step of the binary search, the E_1 values of the 2 parts of the divided region are compared, and the final position is chosen as the one that divides the region into 2 parts with approximately equal errors. The recursive construction process continues until we reach a region size of 1×1×1. A recursive branch is pruned when the summed QEF of the corresponding region contains 0 intersection points. This results in a sparser k-d tree, which saves considerable memory.

5 TOPOLOGY-PRESERVING NODE CLUSTERING

As studied in [Schaefer et al. 2007; Zhang et al. 2004], we consider topology-preserving conditions during node clustering. We use simple, yet efficient bottom-up principles to judge whether a node can be clustered. A node is clusterable when one of the follow 3 conditions is satisfied:

- N is a leaf node.
- N has one clusterable child, and the 8 corners of N are inhomogeneous (not all corners share the same sign).
- N has 2 clusterable children, and all 4 parallel edges of N intersecting the subdivision plane exhibit no more than one sign change.

A signed clusterable node that obeys these conditions is equivalent to an identical node since no more than one sign change occurs on any of its 12 edges. We can naturally identify the components inside such a node cell using the method mentioned in section 3.

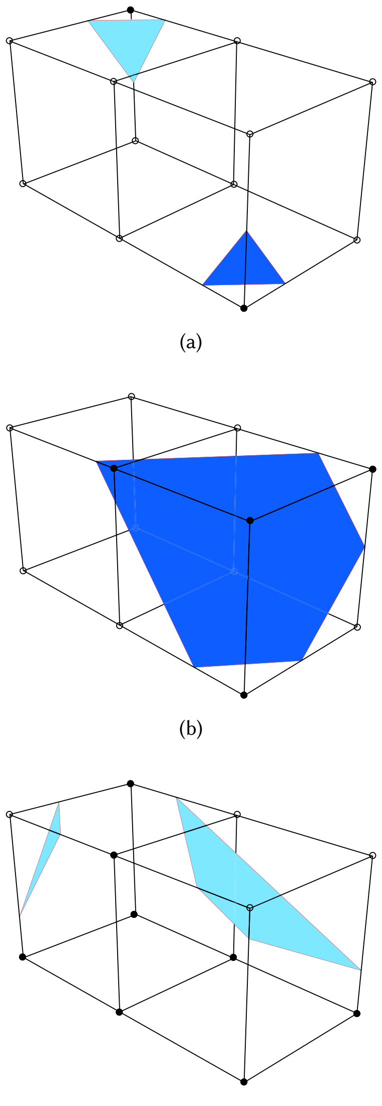


Fig. 6. Some configuration of a k-d tree node with it 2 children. Surface component are drawn in dark blue (for back face) and light blue (for front face). Also, the sign of lattice point of the 2 adjacent children nodes are drawn with black or empty point. K-d tree nodes represent by (a) and (b) are clusterable as non of their 4 parallel edges intersecting the subdivision plane exhibit more than one sign change, while (c) denotes a unclusterable configuration.

Three examples during k-d node clustering are shown in figure ??. The only difference lies in how we sum associated QEFs. The components of a coarse clusterable node are obtained by combining adjacent components of its child nodes while simultaneously inheriting disadjacent components.

During clustering, all sign changes are preserved; thus, we can claim that the surface generated from a simplified version of the k-d tree is topologically equivalent to that generated from the finest level of the k-d tree.

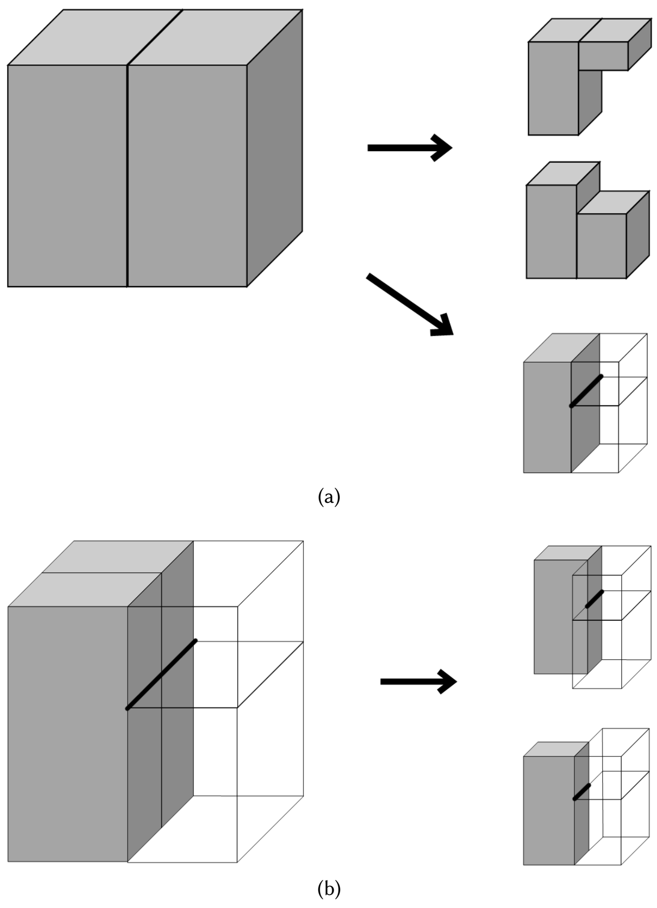


Fig. 7. Recursive function faceProc for iterating on minimal faces (a) and function edgeProc for iterating on minimal edges (b).

The so-called *clustering* process does not really require us to delete clusterable nodes that are below the error threshold. With error thresholds of different magnitudes, we can obtain different levels of detail (LODs) of the reconstructed surface. Furthermore, view-independent isosurface extraction is also possible.

6 ADAPTIVE POLYGON GENERATION

For octree-based methods, [Ju et al. 2002] proposed an efficient scheme for creating a triangle fan or quad connecting the vertices associated with leaf octree nodes containing a given edge. The main advantage of their method is that it is unnecessary to look up neighboring nodes. In contrast, the only method of directly extracting isosurfaces from k-d trees [Gress and Klein 2003] generates polygons by reconstructing the connectivity of the half-edges; hence, it is inefficient and requires additional storage space.

Motivated by [Ju et al. 2002], we extract our isosurface by recursively finding the minimal edges of the k-d tree. The definition of a minimal edge that is adopted here is slightly different from the typical definition: Given a discrete k-d tree and a threshold e, we call a *k-d tree node* whose error exceeds the threshold but has no children whose error exceeds the threshold a *leaf node of threshold e*. Similarly, we call an *edge* that is surrounded only by leaf nodes of threshold e a *minimal edge of threshold e*. It is obvious that each minimal signed edge of threshold e is surrounded by 3 or 4 adjacent leaf nodes of threshold e in the k-d tree; thus, we can use a similar approach to identify all minimal signed edges.

To avoid inefficient neighbor finding among adjacent nodes of the k-d tree, we rely on 3 iterating functions: `cellProc(n)`, `faceProc(n1, n2)` and `edgeProc(n1, n2, n3, n4)`. Starting with a given k-d tree root node, `cellProc` recursively calls itself on each child of the current node, and `faceProc` is called on the pair of its 2 children.

The behavior of `faceProc` is more complicated. The `faceProc` function cares only about the common face of n1 and n2. Let us assume that n1 is not a leaf of threshold e and that the subdivision plane of n1 has the same axis as the common face; then, the algorithm recursively calls itself on n1' and n2, where n1' denotes a child of n1 that shares the common face plane of n1 and n2. If the subdivision plane of n1 has an axis different from that of the common face, then `cellProc` recursively calls itself on n1.child1, n2 and n1.child2 and simultaneously calls `edgeProc` on n1.child1, n1.child2, n2 and n2; note that the last 2 parameters of `edgeProc` here are the same, indicating that there are only 3 adjacent nodes surrounding the edge being processed. Whether there are actually 4 surrounding adjacent nodes will be subsequently discovered by the `edgeProc` function.

The `edgeProc` function first identifies whether the node indicated by the 2 identical parameters has a subdivision plane that exactly contains the edge being processed and, if it does, replaces the 2 identical parameters with the 2 children of that node. Afterward, the `edgeProc` function recursively calls itself on 2 subedges, which can be identified from a subdivision plane of the 4 or 3 surrounding nodes that is orthonormal to the edge being processed. If there are no orthonormal subdivision planes beyond the surrounding nodes, then the algorithm replaces each of them with its child that shares the edge being processed and recursively calls `edgeProc`.

Figure 7a and 7b depicts the recursive calling of `faceProc` and `edgeProc`.

Finally, our algorithm terminates on `edgeProc` when all surrounding nodes are leaves of threshold e; since there are either 3 or 4 surrounding nodes, we can directly generate a triangle or quad to compose the final mesh.

7 RESULTS AND COMPARISON

In order to compare our algorithm with octree-based methods, we have chosen an manifold guarantee extension proposed by Ju [Schaefer et al. 2007].

To applying our algorithm on multiple sources of datasets, including CSG, CT scanners, polygons, we first convert those sources to a uniform sampled grid. Then an octree or a k-d tree hierarchy on the uniform grid was constructed. And because of that the construction is expensive. However to show best adaptability of various sources, we find the uniform grid most suitable for test. By running test cases on a 2.7 GHz Intel Core i5 PC with 8GB memory we have confirmed that our method leads to almost 50% fewer triangles than octree based methods when contouring common scalar fields, as shown in figure 8, 9, 10 and table 1.

8 CONCLUSION AND FUTURE WORK

In this paper we have present a hierarchy structure named discrete k-d tree for extracting isosurface from volume data. We make use of gradient information of f and compute a per-dimensional version of QEF to guide spatial partition. Furthermore, we find a simple topology-preserving node cluster criterion that guarantees only a 2-manifold can be generated. Finally, a efficient polygon generation algorithm is developed.

We believe that isosurfacing algorithms designed for octrees can be naturally extended to the discrete k-d tree hierarchy. With a better spatial partition, those algorithms can produce more compact results than octrees. However, the construction process is still somewhat slow. Actually it can be further optimized by applying a heuristic sampling method without a full summation of all uniform sampled data for a given region, but will result a less accurate approximate to the surface feature.

REFERENCES

Akio Doi and Akio Koide. 1991. An Efficient Method of Triangulating Equi-Valued Surfaces by Using Tetrahedral Cells. *IEICE Transactions on Information and Systems* 1 (1991), 214–224.

Alexander Gress and Reinhard Klein. 2003. Efficient representation and extraction of 2-manifold isosurfaces using kd-trees. In *11th Pacific Conference onComputer Graphics and Applications, 2003. Proceedings.*, Vol. 66. 370–397.

Tao Ju, Frank Losasso, Scott Schaefer, and Joe D. Warren. 2002. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, Vol. 21. 339–346.

Thomas Lewiner, VinÃŋcius Mello, Adelailson Peixoto, SinÃľsio Pesco, and HÃľlio Lopes. 2010. Fast Generation of Pointerless Octree Duals. *Computer Graphics Forum* 29, 5 (2010), 1661–1669.

Peter Lindstrom. 2000. Out-of-core simplification of large polygonal models. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques.* 259–262.

Ricardo Uribe Lobello, Florent Dupont, and Florence Denis. 2018. MULTI-RESOLUTION DUAL CONTOURING FROM VOLUMETRIC DATA. In *International Conference on Computer Graphics Theory and Applications.* 163–168.

William Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *Computers & Graphics* (1987).

Akshay Narayan, Jaya Sreevalsan-Nair, Kelly Gaither, and Bernd Hamann. 2018. ISO-SURFACE EXTRACTION FROM HYBRID UNSTRUCTURED GRIDS CONTAINING PENTAHEDRAL ELEMENTS. In *International Conference on Information Visualization Theory and Applications.* 660–669.

Gregory M. Nielson. 2004. Dual Marching Cubes. In *Proceedings of the conference on Visualization '04.* 489–496.

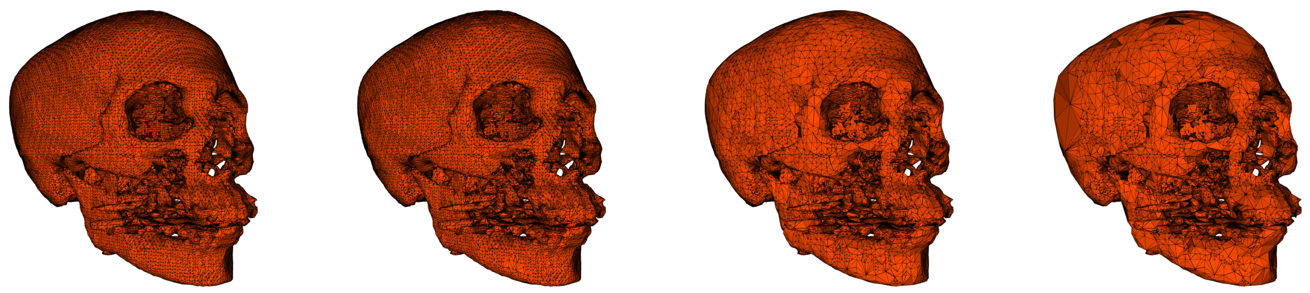
Scott Schaefer, Tao Ju, and Joe D. Warren. 2007. Manifold Dual Contouring. *IEEE Transactions on Visualization and Computer Graphics* 13, 3 (2007), 610–619.

Scott Schaefer and Joe D. Warren. 2004. Dual marching cubes: primal contouring of dual grids. In *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.* 70–76.

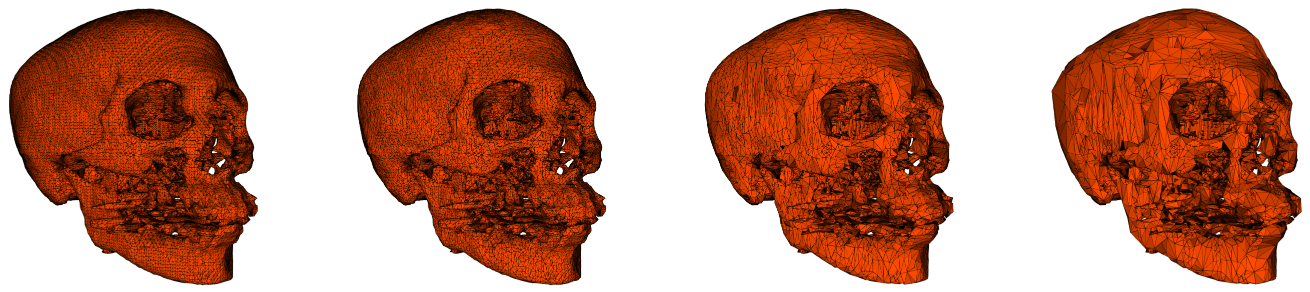
Raj Shekhar, Elias Fayyad, Roni Yagel, and J. Fredrick Cornhill. 1996. Octree-based decimation of marching cubes surfaces. In *Proceedings of the 7th conference on Visualization '96*, Vol. 25. 335–342.

Renben Shu, Chen Zhou, and Mohan S. Kankanhalli. 1995. Adaptive marching cubes. *The Visual Computer* 11, 4 (1995), 202–217.

Nan Zhang, Wei Hong, and Arie E. Kaufman. 2004. Dual Contouring with Topology-Preserving Simplification Using Enhanced Cell Representation. In *Proceedings of the conference on Visualization '04.* 505–512.

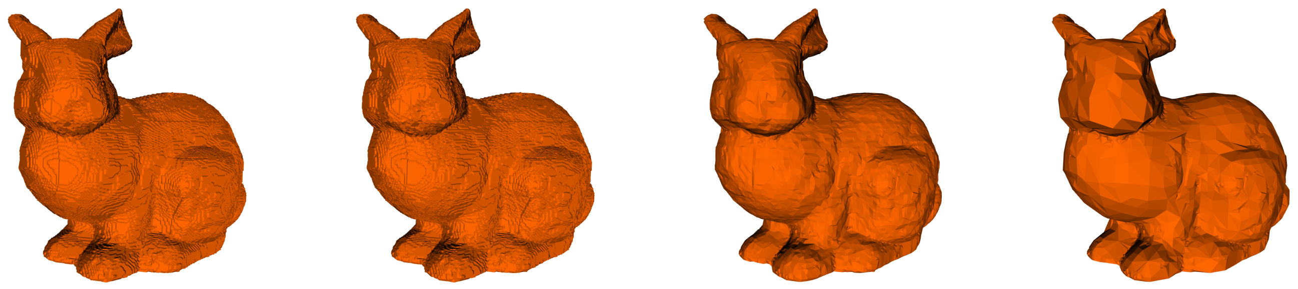


(a) 197,140 triangles (b) 184,126 triangles (c) 99,500 triangles (d) 83,582 triangles

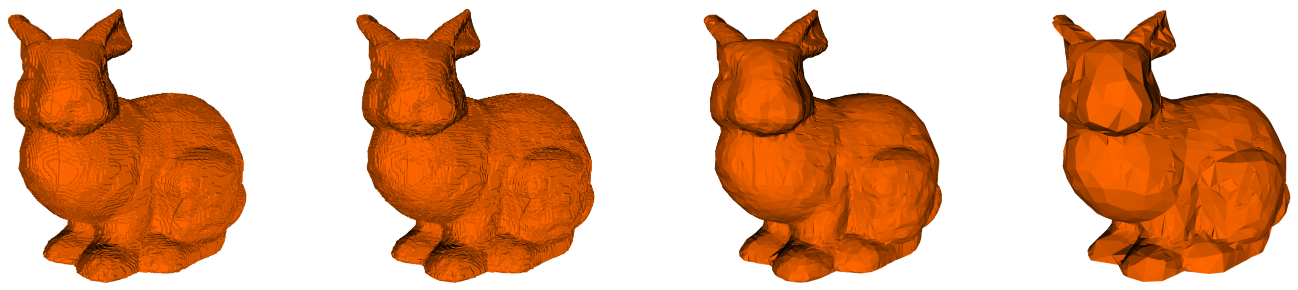


(e) 197,140 triangles (f) 150,640 triangles (g) 69,046 triangles (h) 58,126 triangles

Fig. 8. Adaptive contour result of a row CT scanned data using octree (top) and k-d tree (bottom) with topology preservation, the four columns have -1, 0.001, 0.1 and +inf error thresholds respectively. Note that thin features are very common in this dataset, thus limits the maximum amount simplification in both k-d tree and octree

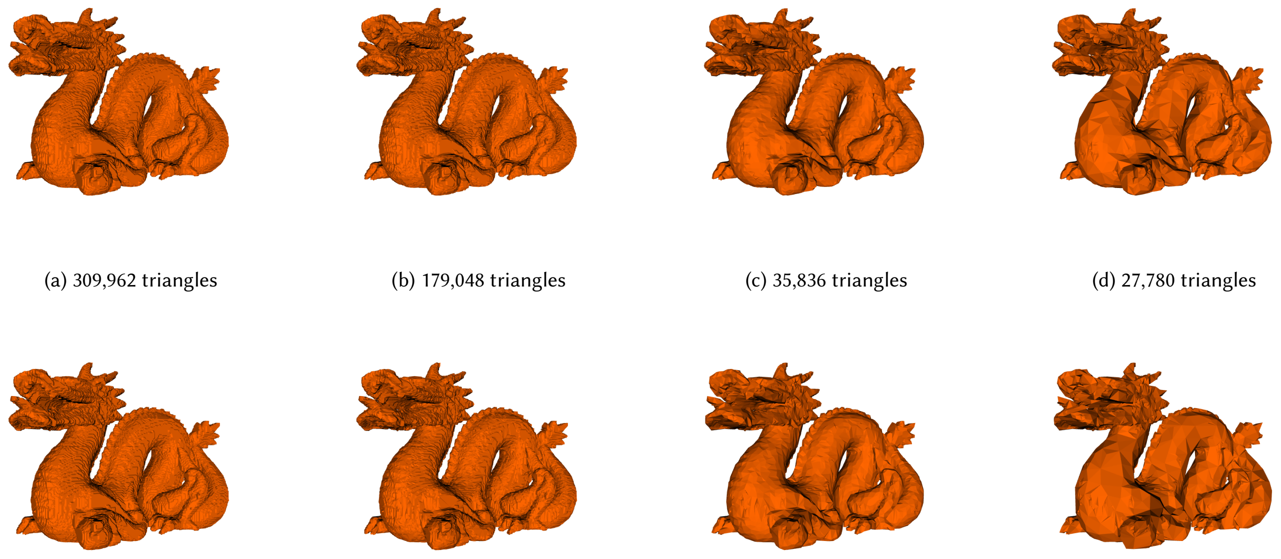


(a) 371,415 triangles (b) 221,090 triangles (c) 26,492 triangles (d) 9,738 triangles



(e) 371,415 triangles (f) 103,798 triangles (g) 16,230 triangles (h) 7,238 triangles

Fig. 9. Adaptive contour result of a scan-line converted bunny polygons using octree (top) and k-d tree (bottom) with topology preservation, the four columns have -1, 0.01, 1 and 100 error thresholds respectively



(a) 309,962 triangles (b) 179,048 triangles (c) 35,836 triangles (d) 27,780 triangles

(e) 309,962 triangles (f) 89,331 triangles (g) 22,608 triangles (h) 17,104 triangles

Fig. 10. Adaptive contour result of a scan-line converted dragon polygons using octree (top) and k-d tree (bottom) with topology preservation, the four columns have -1, 0.01, 1 and 100 error thresholds respectively

Scalar field sources	resolution	threshold	octree triangles	k-d tree triangles	octree construction time	k-d tree construction time (using mipmap)
CSG(fig 2)	128	0.01	7,868	990	7.304s	5.205s
CSG(fig 3)	128	0.1	1,952	856	3.183s	3.296s
CT scanner(fig 8)	128	0.01	147,080	102,880	9.854s	31.745s
Polygons(bunny, fig 9)	256	0.01	221,090	103,798	105.025s	87.926s
polygons(dragon, fig 10)	256	0.01	179,048	89,331	71.686s	63.016s

Table 1. Comparison statistics of contouring volume data from different kind of sources