March 2003

Feature preserving manifold mesh from an octree

Koji Ashida
University of Pennsylvania

Norman I. Badler
University of Pennsylvania, badler@seas.upenn.edu

Poster session at the 8th ACM Symposium on Solid Modeling and Applications (SM'03), held 16-20 June 2003. Published in the Proceedings, pages 292-297.
Publisher URL: http://doi.acm.org/10.1145/781606.781654

Feature preserving manifold mesh from an octree

Abstract
We describe an algorithm to generate a manifold mesh from an octree while preserving surface features. The algorithm requires samples of a surface (coordinates) on the octree edges, along with the surface normals at those coordinates. The distinct features of the algorithm are:
- the output mesh is manifold,
- the resolution of the output mesh can be adjusted over the space with octree subdivision, and
- surface features are generally preserved.
A mesh generation algorithm with this combination of advantages has not been presented before.

Keywords
computer graphics, computational geometry and object modeling, algorithms, manifold, feature, octree

Comments
Poster session at the 8th ACM Symposium on Solid Modeling and Applications (SM'03), held 16-20 June 2003. Published in the Proceedings, pages 292-297.
Publisher URL: http://doi.acm.org/10.1145/781606.781654

Feature Preserving Manifold Mesh from an Octree

Koji Ashida Norman I. Badler

March 13, 2003

Abstract

We describe an algorithm to generate a manifold mesh from an octree while preserving surface features. The algorithm requires samples of a surface (coordinates) on the octree edges, along with the surface normals at those coordinates. The distinct features of the algorithm are:

- the output mesh is manifold,
- the resolution of the output mesh can be adjusted over the space with octree subdivision, and
- surface features are generally preserved.

A mesh generation algorithm with this combination of advantages has not been presented before.

1 Introduction

Recent advances in feature sensitive mesh generation allow reproduction of sharp features without significant increase in polygon complexity (e.g. [KBSS01] and [JLSW02]). Meshes built as isosurfaces of a volumetric scalar field tend to lose high frequency information. Using a finer resolution grid is not a very effective solution. Feature sensitive methods tend to reproduce sharp features by preserving exact zero crossing coordinates and surface normals at these points.

On the other hand, octrees have been used to efficiently generate meshes and to produce adaptive meshes[1] (e.g. [WG92], [MS93], [SFYC96], [SMW97], and [OR97]).

Dual Contouring (**DC**) [JLSW02] generates meshes from octrees while preserving surface features, but the output of the algorithm is non-manifold. A manifold mesh is a mesh in which each vertex has a neighborhood homeomorphic to a disk (e.g. [Wei99]). There are cases when manifold meshes are more desirable than non-manifold. This is because many mesh algorithms work only on manifold meshes, and many others require more complex procedures to handle non-manifold cases. Extended Marching Cubes (**EMC**) [KBSS01] generates a manifold mesh while preserving surface features. The output of **EMC** does not have adaptive resolution.

Our contribution is combining the advantages of the previous work in one algorithm: namely, producing a manifold mesh from an octree while preserving surface features.

[1] Adaptive meshes are meshes with more polygons where necessary, and less polygons elsewhere to achieve lower total polygon count and higher accuracy at the same time.

2 Method

2.1 Overview

We first describe the input data structure and then describe how our method differs from Marching Cubes (**MC**) [LC87] and **DC**.

An octree (e.g. [Sam90]) is a set of cubic cells. Each cell can be either a terminal cell or a non-terminal cell. A non-terminal cell has eight child cells which equally divide the cell into cubes. The set of terminal cells of an octree fills the cubic domain space. By proper scaling, we assume the desired surface is enclosed within this domain cubic space.

Collecting vertices, edges, and faces of the octree cells, we obtain a domain mesh. Note that there is redundancy in those domain mesh components. For example, a vertex of a cell coincides with vertices of

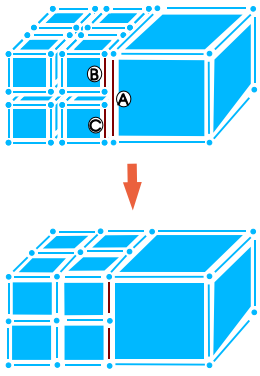

Figure 1: Consider mesh components of octree cells (top). Removing redundant components while choosing the smallest ones yield the mesh on the bottom. Since edges A / B, and A / C overlap, only B and C remain in the domain mesh.

Domain mesh	**MC**	**DC**	Our method
v	(ϕ)	(sign of ϕ)	(sign of ϕ)
e	$v \times 1$	$f \times 1$	$f \times 1$
f	$e \times n$	$e \times n$	$e \times n$
c	$f \times n$	$v \times 1$	$v \times n$

Table 1: Comparison of mesh generation methods. v, e, f, and c denote a vertex, edge, face, and cell, respectively. $a \times 1$ means only one mesh component a is porduced, and $a \times n$ means $n \geq 1$ mesh components a are produced for one domain mesh component. ϕ denotes a acalar value associated to a domain mesh vertex, but it is not a part of the output surface mesh.

many other cells. We consider the domain mesh to have only unique components. Furthermore, when different levels of edges and faces overlap, only the smallest ones are included (Figure 1). This also means an octree cell may be adjacent to more than eight vertices, twelve edges, and six faces.

In an octree domain mesh, there are edges, faces, and cells of different sizes. Those can be distinguished by the subdivision depth level of the cell they are associated to. For example, since the domain cubic cell is considered to be at level 0, its edges and faces are also level 0. Child cells of level 0 cell are level 1 cells. Their edges and faces are level 1 edges and faces.

A vertex of an octree cell is associated to a scalar value. This value may be a signed distance to the desired surface from that vertex. For our work, the magnitude of the scalar value is not important. Only the sign of the value is significant.

Many mesh generation methods, including **MC**, **DC**, and our method produce mesh components (i.e. vertex, edge, or face) matching components of the domain mesh (octree or uniform grid in **MC**) components. Table 1 summarizes what components **MC**, **DC**, and our method produce for each component of a domain mesh. For example, **MC** produces one mesh vertex for each grid edge which exhibits a sign change. This is shown in the second row ("e" row) "**MC**" column of the table.

DC type methods have characteristics of preserving features, therefore, we follow **DC** and generate vertices, edges, and faces of the mesh, matching cells, faces, and edges of an octree. In order to generate a manifold mesh, however, we generate one or more vertices for each cell which exhibits a sign change ("c" row "Our Method" column of the table). Here we give an intuition of why this is necessary. **MC** generates a manifold mesh. If **DC** would generate a dual mesh[2] of **MC** output, it would be manifold. This is how **EMC** generates a manifold mesh. As seen clearly in Table 1, we need to generate one or more vertices for a cell in order to make a dual of **MC** mesh.

[2]A dual mesh is a mesh obtained by changing vertices to faces and faces to vertices. A dual of a manifold mesh is also a manifold mesh.

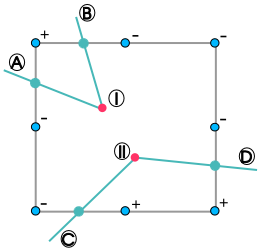

Figure 2: If the surface samples are given consistently with the signs, there are always even number of samples around an octree face. The figure shows a possible assignment of the signs for a given set of samples. If we decide to match mesh face A to B, and match C to D, then we have mesh edges I and II. Since there were four mesh faces, we get $4/2 = 2$ mesh edges.

This description, however, omits some details of how exactly we generate meshes. Our output mesh also is not an exact dual of the **MC** mesh. In the next subsection, we detail how our algorithm generates a mesh from an octree.

2.2 Generating a mesh from an octree

In order to produce a mesh, we first identify mesh faces on octree edges. On octree faces, we find intersections of those mesh faces to determine mesh edges. Collecting those mesh edges around an octree cell, we produce mesh vertices. This procedure is performed on each level of an octree in a bottom up manner.

There will always be exactly one face for each octree edge which exhibits a sign change. This is because an edge is always adjacent to two vertices, each of which has a sign. If the desired surface requires more than one face on an edge, this edge has to be subdivided, i.e. one of the adjacent cells needs to be subdivided.

An octree face might be adjacent to more than four edges. However, if the surface samples are consistent with the sign changes, there should always be an even number of samples on the edges surrounding a face. When a pair of them intersect, there is a mesh edge. For an octree face with f mesh faces, we need $f/2$ mesh edges (Figure 2).

An octree cell might be surrounded by more than six faces. If we know how mesh faces are connected on each of those faces, we know how cycles of mesh faces are connected. There may be more than one cycle of mesh faces around an octree cell. For each cycle of mesh faces, we generate a mesh vertex (Figure 3).

If we find all the mesh face cycles on terminal octree

cells and keep track of correspondences of those cycles and mesh vertices, we can connect mesh edges to mesh vertices. Since all the mesh vertices are connected to one cycle of mesh facesand all mesh edges are shared by two mesh faces, the resulting surface is manifold.

Now, our only concern is in what order we identify those mesh components. For example, to identify mesh vertices in a cell, all the mesh edges on the octree faces around this cell should have been identified. By traversing the tree in a bottom up manner, we can guarantee this precondition. To show why, we first set a hypothesis.

Induction hypothesis: *When we are working on an octree level l, we have identified all the mesh edges associated to octree faces of level $> l$.*

We first assign unique ID's to octree edges, faces and cells. The octree edge ID's are used to identify the mesh faces, too. Assume the octree has a maximum level of m. We now consider an arbitrary level l_a $(0 \leq l_a \leq m)$ of the octree. All the mesh edges of level $> l_a$ are already identified (induction hypothesis). We first go through all the level m faces. By using the procedure described in the next subsection, we pair up mesh faces surrounding those octree faces. A pair of mesh faces corresponds to a mesh edge. This finds all the mesh edges of level $\geq l_a$. Next, we look at the octree cells of level l_a. An octree cell of level l_a is surrounded only by faces of level $\geq l_a$, which are all already identified. We collect those mesh edges surrounding the cell. Each mesh edge is associated to two mesh faces. Ordering mesh faces with mesh edges as connections (using the mesh face ID's) produces cycles of mesh faces. Each cycle corresponds to a mesh vertex. Vertex coordinates are computed by solving Quadric Error Metrics (**QEM**) ([GH97] and [Lin00]). It finds the 3D coordinate of the vertex which minimizes the sum of squared distances to the mesh faces planes. The same procedure is used in [JLSW02].

QEM is solved using Singular Value Decomposition (**SVD**) (e.g. [PFTV88]). We set small singular values to zero. Singular values are considered too small when it is smaller than γ times the largest singular value. $\gamma = 0.01$ worked fine for all our experiments.

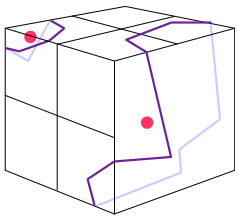

Figure 3: A sequence of connected mesh faces create a cycle. For each cycle, we make one mesh vertex (the red dots). This is the vertex the mesh faces and the mesh edges participating in the cycle are adjacent to.

Algorithm 1 Mesh construction procedure. F_f is the number of mesh faces around F. C_F is the number of octree faces around C. $A \rightarrow \mathcal{H}(k)$ denotes the operation of storing A into the hashtable with key k. The lines starting with $\Rightarrow$ are continuations of the previous line.

```
for level l = m down to 0 do
  for each octree face F of level l do
    identify mesh faces f_i (0 ≤ i < F_f) around F
    pair up mesh faces as
        ⇒ P = {{f_i, f_j}, {f_k, f_l}, ...}
    P → H(F)
  for each octree cell C of level m do
    identify ctree faces F_i (0 ≤ i < C_F) around C
    make cycles of octree faces as
        ⇒ V = {{f_i, f_j, ...}, {f_k, f_m, ...}, ...}
    for each cycle c ∈ V do
      generate a mesh vertex v (coordinate)
      for each member mesh face f ∈ c do
        keep v as one of the vertices of f
        keep f if it is new
  merge octree faces of level l and
      ⇒ re-hash as level l − 1 faces
output all the generated mesh vertices
output all the generated mesh triangles
```

After this, the induction hypothesis holds for level $l_a - 1$. Since the induction hypothesis is trivially true for level m, the bottom up traversal was shown to find mesh components properly.

We keep intermediate information in a hashtable For each octree face with a sign change, we get a list of mesh edges (i.e. mesh face pairs). This list is stored in the hashtable with the octree face ID as a key. We look up a list of mesh edges using an octree face ID and find all the mesh edges surrounding an octree cell.

After identifying all the mesh vertices of level l, we merge all those mesh edge lists of level l and re-hash them as a mesh edge list of level $l - 1$. For example, if octree face F^{l-1} is a parent octree face of F_0^l, F_1^l, F_2^l, and F_3^l, we merge the lists associated to F_0^l, F_1^l, F_2^l, and F_3^l into one and re-hash this list with F^{l-1} as a key. This simplifies finding mesh edges around an octree cell; we only need to look up the hashtable six times for each octree cell.

The procedure is summarized in Algorithm 1.

2.3 Paring mesh faces

If there are four or more mesh faces on an octree face, we have to determine the paring of those mesh faces. We developed heuristics to accomplish this, which try

to avoid self intersections of the resulting mesh.

We first consider a graph problem. As shown in Figure 4, we have even number of nodes on the perimeter of a square. We have to connect those nodes without crossing the lines. Since there are an even number of nodes, there is always a valid paring. For any pairing of the nodes, there has to be at least one pair of nodes next to each other on the perimeter. We call this configuration an *adjacent pair* as shown in the figure. Removing this pair still yields a valid pairing.

These observations give us a procedure to produce a valid pairing. Given a set of nodes, we pick a adjacent pair, make them a pair and remove them from the set. Repeating this will yield a pairing of all the nodes. A pairing made using this procedure is always valid and any valid pairing can be produced using this method.

To select an adjacent pair, we use the geometrical information. The input point and its associated normal corresponding to a mesh face define a plane. The intersection of this plane with the octree face is a line. When the lines of adjacent nodes intersect inside the square, it can be a mesh face pair which defines a mesh edge (Figure 5).

In order to avoid self intersections, we first find all the intersections of the adjacent nodes. For each node, we find with which adjacent node its line intersects first. If this relationship is mutual, they are picked as a pair (Figure 6).

When there is no mutual preference, it is ambiguous. When this happens, we use offsetting heuristics to determine a better separation of the surfaces; connect the faces so that the resulting surfaces are maximally separated from each other.

Figure 7 shows an example of such a case. In this case, the region in the middle can be either inside or outside of the surface. In either case, there is no self intersection. We compare the distance between the surfaces and choose the pairing which maximizes this distance.

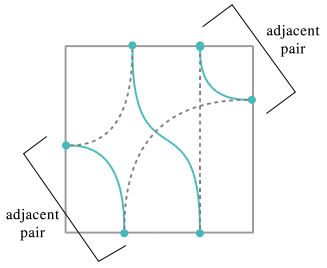

Figure 4: Since there are even number of nodes, there is always a valid pairing. The cyan lines show an example of a valid pairing, and the dotted lines show an example of an invalid pairing. The invalid pairing does not only causes self intersection, but also makes inside / outside of the mesh inconsistent with the input data. In a valid paring, there is always an adjacent pair.

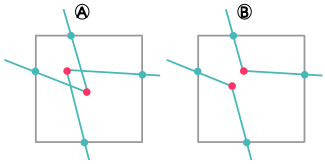

Figure 5: Both A and B are valid pairing, but A causes self intersection while B does not. The red dots are the mesh edges.

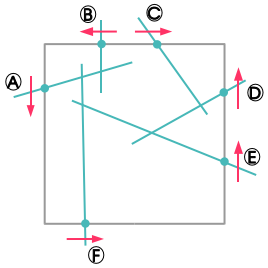

Figure 6: For each node, we use the sample coordinate and normal to find which adjacent node it intersects with first. The arrows indicate the adjacent neighbor each node intersects with first. In this case, only node C and D have the mutual preference and chosen as a pair.

2.4 Building an octree from a mesh

We believe the input data structure (octree plus surface samples / normals) can be constructed from a number of different sources, e.g. volume data, mathematical function, and another mesh. Here we show the procedure with mesh input. **EMC** has shown a similar procedure.

We split the polygons (faces) of the input mesh into octree cells. If a polygon face spans multiple cells, it is split into multiple polygons each of which fits in one octree cell. An octree cell is then associated to a set of polygon faces. For a non-terminal cell, the set of polygon faces associated to is the union of the polygon faces associated to its child cells. The set of polygon faces associated to all the terminal cells is equivalent to the original input mesh.

This polygon classification is performed at the same time with octree generation. It is a recursive depth first construction of the tree. Starting with the level 0 cell, at each cell we check if the cell needs to be split, and if so, split the cell and recurse into the child cells.

We decide whether or not to split a cell using some criterion; if a cell does not hold any polygon, this cell does not need to be split, if we reach a predefined max-

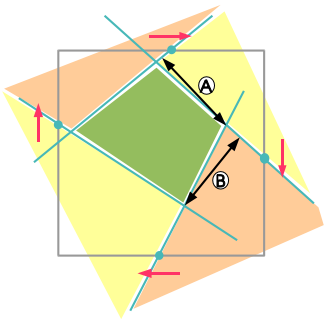

Figure 7: The areas painted beige and yellow are determined to be inside or outside of the surface, e.g. beige areas are inside and yellow areas are outside. However, the green area can be either inside or outside and still does not introduce a self intersection. In this case, we choose the pairing which maximizes the distance between the surfaces. In other words, we compare the distance A and B. If A is larger, the green area connects the yellow areas, and if B is larger, the green area connects the beige areas. The cases with more nodes are resolved analogously.

imum subdivision level, we do not subdivide further, and in all other cases, we use the associated polygons to decide whether or not to split. The cell is split if the expected error is larger than some threshold α. An expected error for a cell is $E = \sum_i {a_i}^2 ({n_i}^T x - {n_i}^T p_i)^2$ where a_i, n_i, and p_i are the area, normal, and a point on the ith polygon associated to the cell. x is the coordinate of the vertex which minimizes E. The equation can be represented as **QEM** and solved with **SVD**.

Note, however, that the resulting output mesh is not guaranteed to have the same topology as the original mesh. When there are more than one cycle of mesh faces around a cell, we always generate the same number of mesh vertices (Figure 3), where the desired surface might have those faces connected as a tunnel. We believe this happens very rarely since if there is a tunnel, the cell has large expected error and should be split further.

3 Experiments

We used three input meshes, normalized to fit in a cube of unit length in each dimension, and used different maximum subdivision levels to generate a series of output meshes. The threshold for subdivision α is $1.0 * 10^{-10}$ for all the experiments. The output meshes are compared to input meshes using the method and program of [PCS98]. The Hausdorff distances measured by this program quantify how different two sets of surfaces are, which shows the faithfulness of the reconstruction.

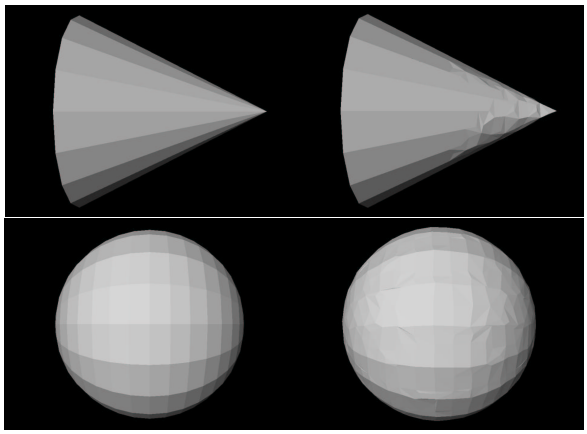

Figure 8: The cone (top) and sphere model (bottom). Left images are the originals, and the right images are reconstructions with maximum octree level 4.

The first input mesh is a cone shown in Figure 8 top left. The right image shows the output of the algorithm. The figures show that the algorithm was able to reproduce sharp features (the crease around the bottom circle and the apex). Table 2 shows number of vertices of the output meshes as well as Hausdorff distance of those meshes to the original. Note the output mesh is still converging to the original after exceeding the original's mesh complexity. We believe it is difficult for most surface reconstruction methods to obtain as good mesh as a hand-crafted original. The bottom two images of Figure 8 show the original and the result of sphere mesh reconstruction.

Finally, we use the Stanford bunny as input. This mesh not only is the largest in our test meshes, but also is non-manifold. Figure 9 shows the rendering of the input and our output.

4 Summary

We suggested an algorithm to generate a manifold mesh from an octree. This algorithm has the advantage of preserving features. By adjusting the octree subdivision, the output mesh can have more detail in some regions and less detail elsewhere to save the total polygon complexity. This saves computation for later processes, including rendering and network transfer.

Components of the resulting mesh are associated to components of domain (octree) mesh. By traversing the octree bottom up and going through the octree components, we identified mesh components and their connectivities. Ambiguities are resolved with heuristics, which try to avoid self intersection of the resulting surface. The input octrees (with surface samples and normals) are generated from an existing mesh and the results were compared to the orignal.

Figure 9: The original Stanford bunny model (left) and the output our algorithm at maximum octree level 7 (right).

	Cone ($n = 22$)		Sphere ($n = 382$)		Bunny ($n = 35947$)	
L	v	d	v	d	v	d
1	8	0.068172	8	0.099943	8	0.099999
2	32	0.032648	56	0.017715	30	0.099997
3	176	0.014581	232	0.010288	164	0.099993
4	752	0.006219	958	0.002350	742	0.033924

Table legend
- n: number of vertices in the original mesh
- L: maximum octree level
- v: number of output vertices
- d: Hausdorff distance

Table 2: Mesh reconstruction results.

References

[GH97] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proc. of SIGGRAPH '97*, pages 209–216, 1997.

[JLSW02] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. In *Proc. of SIGGRAPH '02*, pages 339–346, 2002.

[KBSS01] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H. P. Seidel. Feature sensitive surface extraction from volume data. In *Proc. of SIGGRAPH '01*, pages 57–66, 2001.

[LC87] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proc. of SIGGRAPH '87*, pages 163–169, 1987.

[Lin00] P. Lindstrom. Out-of-core simplification of large polygonal models. In *Proc. of SIGGRAPH '00*, pages 259–262, 2000.

[MS93] H. Muller and M. Stark. Adaptive generation of surfaces in volume data. *The Visual Computer*, 9:182–199, 1993.

[OR97] M. Ohlberger and M. Rumpf. Hierarchical and adaptive visualization on nested grids. *Computing*, 59(4):365–385, 1997.

[PCS98] C. Rocchini P. Cignoni and R. Scopigno. Metro: measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, June 1998.

[PFTV88] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes*. Cambridge University Press, 1988.

[Sam90] H. Samet. *Applications of Spatial Data Struc-tures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, 1990.

[SFYC96] R. Shekhar, E. Fayyad, R. Yagel, and J. Fredrick Cornhill. Octree-based decimation of marching cubes surface. In *Proc. of Visualization '96*, pages 335–342, 1996.

[SMW97] M. Stark, H. Muller, and U. Welsch. Variations of the splitting box scheme for adaptive generation of contour surfaces in volume data. In *Scientific Visualization: overviews, methodologies and techniques*, pages 337–356, 1997.

[Wei99] E. Weisstein. World of mathmatics. http://mathworld.wolfram.com, 1999. page /DualPolyhedron.html.

[WG92] J. Wilhelms and A. V. Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992.