Dual Marching Cubes: Primal Contouring of Dual Grids

Scott Schaefer and Joe Warren
Rice University
6100 Main St.
Houston, TX 77005
sschaefe@rice.edu and jwarren@rice.edu

Abstract

We present a method for contouring an implicit function using a grid topologically dual to structured grids such as octrees. By aligning the vertices of the dual grid with the features of the implicit function, we are able to reproduce thin features of the extracted surface without excessive subdivision required by methods such as Marching Cubes or Dual Contouring. Dual Marching Cubes produces a crack-free, adaptive polygonalization of the surface that reproduces sharp features. Our approach maintains the advantage of using structured grids for operations such as CSG while being able to conform to the relevant features of the implicit function yielding much sparser polygonalizations than has been possible using structured grids.

1. Introduction

Implicit modeling has become a popular method for extracting surfaces from volumetric information arising from sources such as MRI data and CAT scans. In this technique, a volumetric function $f(x, y, z)$ is given, whose value typically represents density information or distance to a surface. To extract a surface from this function, we use a level set of $f(x, y, z) = c$. Notice that we can always reduce this problem to examining the zero-contour of the function by simply subtracting c.

This form of modeling has several advantages over traditional modeling techniques. Since $f(x, y, z)$ encodes the surface, modifications to the surface (such as CSG operations) can be performed simply by modifying the underlying functions $f(x, y, z)$. The topology of the surface is also implicitly defined by $f(x, y, z)$ so topological modifications are easy because the topology doesn't have to be changed explicitly. Finally, this technique is volumetric and additional information can be gleaned from the underlying function such as distance to the surface simply by evaluating $f(x, y, z)$.

Since representing functions that contain arbitrary geometry can be difficult, many techniques sample the function $f(x, y, z)$ on a structured grid. Such grids are typically axis-aligned and have vertices from a uniform sampling of space (uniform grids) or some dyadic sampling (octrees). The main advantage of using these grids is that operations (such as CSG) are easy to perform because the grids between two different implicit surfaces align and the operation can be reduced to an operation on the vertices of the grid.

Methods that use structured grids for implicit modeling are quite abundant. Perhaps the most popular is Marching Cubes [9]. Marching Cubes takes as input a uniform grid whose vertices are samples of the function $f(x, y, z)$ and extracts a surface as the zero-contour. For each cube in the grid, Marching Cubes examines the values at the eight corners of the cube and determines the intersection of the surface with the edges of the cube. Then Marching Cubes provides a lookup table indexed by the sign configuration at the eight corners that yields the topology of the surface inside of that cube. After processing each cube in the grid, the surface is complete.

There have been many attempts to extend Marching Cubes from uniform to adaptive grids such as octrees. However, these methods all require some sort of patching between cubes of different resolution [12, 11, 5]. Recently several dual approaches to contouring have been introduced that effectively eliminate this patching problem [2, 10, 4, 13]. In these methods, each cell in the grid is given a representative vertex. For methods such as Dual Contouring, this vertex is placed at sharp features of the surface, which allows the method to reproduce sharp features such as edges and corners. Polygons are then generated by examining *minimal edges* (an edge that contains no smaller edge) in the octree. Then for each minimal edge the surface passes through, a polygon is generated that connects the vertices of the cells containing that minimal edge. Since the surfaces produced by these methods are topologically dual to the surfaces produced by Marching Cubes, we call these

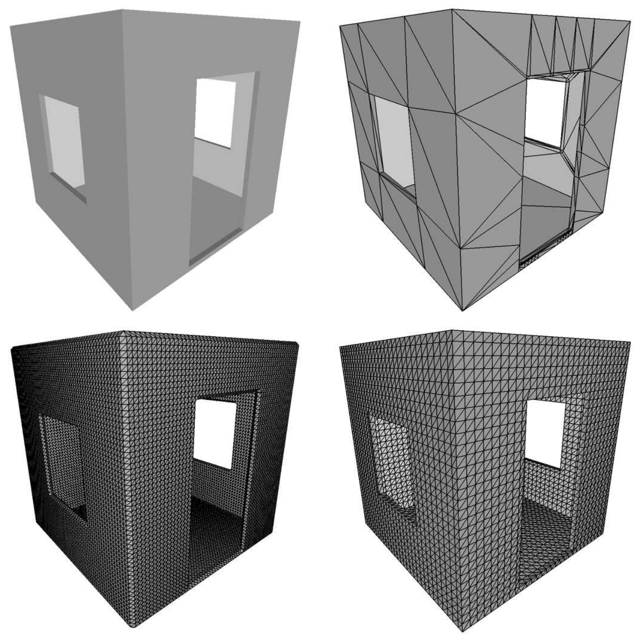

Figure 1. A thin-walled room defined via CSG (upper left). Polygonal approximations were generated by Marching Cubes (lower left, 67K polys), Dual Contouring (lower right, 17K polys) and Dual Marching Cubes (upper right, 440 polys). Using Dual Marching Cubes, the size of the contour mesh is insensitive to the thickness of the walls.

methods *dual* contouring methods and cube-based methods *primal* contouring methods.

However, all of these methods are limited in their expressivity because they are based on structured grids. For instance, to represent very thin features, these methods require that a vertex of the grid lies inside the surface and another, adjacent vertex outside the surface. Since the grids are structured in nature, extracting very small, thin features may require a very fine grid. Figure 1 (top, left) shows a room generated by CSG operations with very thin walls. Three different contouring methods have been used to produce the surface from the CSG tree. Both Marching Cubes and Dual Contouring rely on structured grids and require fine grids (and, consequently, large amounts of polygons) to reproduce the correct topology of the room. While Dual Contouring is a multi-resolution contouring algorithm that can adaptively simplify the surface, the amount of simplification is still limited by the thickness of the walls.

Balmelli et al [1] attempt to solve this problem by constructing a uniform grid whose vertices are allowed to conform to the features of the surface using an importance map. While this approach yields better tessellations, it is still limited by uniform sampling and cannot reconstruct sharp features accurately. Furthermore, performing operations such as CSG between two objects becomes difficult because the vertices of the two grids no longer align and resampling must be performed.

Varadhan et al [13] provide an alternate solution based off of Dual Contouring [4] and directed distance fields [6]. In that paper the authors provide a technique that can reconstruct a two-sheeted surface in a cell of the octree. This allows their method to reproduce very thin walls. However, their algorithm is still based off of a primal grid and requires the isosurface to intersect the edges of the primal grid in order to be reconstructed faithfully.

Contributions

We propose a fundamentally different approach to contouring, which we entitle Dual Marching Cubes. The grid that we perform contouring on will be topologically dual to the structured grids used by other techniques. Therefore, we denote this type of grid as a *dual grid* and structured grids as *primal grids*. Our underlying data structure is an octree that adaptively samples $f(x, y, z)$. To generate a surface, we extract a dual grid to this primal octree. The vertices of this dual grid are positioned at the features of the implicit function $f(x, y, z)$ inside each cell of the octree. Using this dual grid that conforms to the features of $f(x, y, z)$, we generate the surface using a generalized version of Marching Cubes. Contouring on this dual grid yields several advantages

- Since we use a structured grid to sample $f(x, y, z)$, we maintain the advantages associated with structured grids for operations such as CSG.
- The vertices of our dual grid align with the features of $f(x, y, z)$, which allows us to reproduce sharp features similar to the reproduction achieved by [6, 4].
- The surfaces produced by this method are adaptive polygonalizations that are crack-free and topologically manifold.
- Due to the use of a dual grid for contouring, we can reproduce small, thin features in the surface such as walls or tubes without excessive subdivision of the octree.

First, we describe the creation of the dual grid that we perform contouring on, which proceeds in two steps: feature isolation and topology creation. Feature isolation takes an octree as input and generates a single vertex for each cell at a feature of $f(x, y, z)$ inside that cell. After building the vertices of the dual grid, we construct the topology of the grid using a simple extension of the traversal algorithm of Ju et al. [4]. We then describe how Marching Cubes can be applied to this dual grid to generate a surface.

2. Feature isolation

Given a function $f(x, y, z)$ and an octree, our goal in feature isolation is to construct a vertex for each cell in the octree that will become a vertex of the dual grid. In contrast to methods such as Dual Contouring, this vertex is not aligned with the features of the surface but with features of the implicit function. Therefore, each vertex will not only contain a position (x, y, z) but a scalar value w indicating the estimated value of $f(x, y, z)$ at that vertex. Our approach to detecting features from implicit functions is similar to the technique of Garland [3] for detecting features of surfaces. To determine the vertex that approximates the feature of $f(x, y, z)$ inside of a cell c, we use quadratic error functions (QEFs) (as developed in [3]). To generate these QEFs, we compute tangent planes to the graph of $f(x, y, z)$ on a grid of points sampled over c. At a sample point (x_i, y_i, z_i), the tangent plane to $w = f(x, y, z)$ has the equation $w = T_i(x, y, z)$ where

$$T_i(x, y, z) = \nabla f(x_i, y_i, z_i) \cdot ((x, y, z) - (x_i, y_i, z_i))$$

and $\nabla f(x_i, y_i, z_i)$ is the gradient of $f(x_i, y_i, z_i)$. To build the QEF, we square this equation and sum over all sample points yielding

$$E(w, x, y, z) = \sum_i \frac{(w - T_i(x, y, z))^2}{1 + |\nabla f(x_i, y_i, z_i)|^2}. \tag{1}$$

The denominator of this expression normalizes the contribution of each tangent plane to have equal weight.

Next, we minimize this quadratic function over the cell c to find the vertex of the dual grid (w, x, y, z). If the minimizer is underdetermined, we follow the method of Lindstrom [7] and use the pseudo-inverse to position the minimizer as close as possible to the center of c.

Notice that our description of this feature isolation phase does not specify what format $f(x, y, z)$ must take on. Specifically, our description allows for a wide variety of possible inputs. For instance, uniform scalar grids typically associated with Marching Cubes can be used where each cube in the grid is regarded as a trilinear function so that $f(x, y, z)$ and $\nabla f(x, y, z)$ are well defined. Alternatively, we can restrict our sampling above to the grid points of this uniform grid and use divided differences to estimate $\nabla f(x, y, z)$ at these vertices. The latter strategy has the advantage of defining a single normal $\nabla f(x, y, z)$ at the grid vertices instead of several discontinuous normals as the former method does. Directed distance fields provide an alternative to uniform, scalar fields and can provide more accurate gradients to help reproduce sharp features.

CSG trees are another possible input to our algorithm. CSG performs set operations on solids where the leaves of the CSG trees are solids and interior nodes are operations such as union and intersection. To use such CSG trees in our algorithm, we must convert the trees to scalar valued functions $f(x, y, z)$ that contain a well defined gradient $\nabla f(x, y, z)$. At the leaves of the CSG tree, we replace the solids with their signed Euclidean distance functions for primitives such as planes, cylinders, etc... At interior nodes, we replace union and intersection operations with Min and Max operations respectively. Evaluation of the CSG tree at a point (x, y, z) simply involves evaluation of the distance functions of each leaf at (x, y, z) and performing the corresponding operations at the interior nodes of the CSG tree. To compute the gradient, $\nabla f(x, y, z)$ is evaluated at the corresponding point at each of the leaves of the CSG tree and passed upwards during the Min/Max operations along with the value of the function at that point. Figures 1 and 6 were each generated by evaluating a CSG tree.

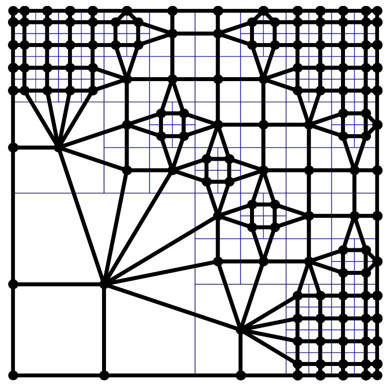

Figure 2. Connectivity of the dual grid (thick black) for a primal quadtree (thin blue).

2.1. Octree construction

Our current description of feature isolation has assumed that an octree is present to partition space into cells. However, sometimes it is desirable to approximate $f(x, y, z)$ to a given tolerance ϵ. To perform this approximation, we present a top-down octree construction algorithm.

This octree construction algorithm proceeds by starting with a single cell c as the octree. We use the sampling algorithm above to sample $f(x, y, z)$ finely on a uniform grid over c (a random sampling strategy could also be used). Next, we construct a QEF for c using those sample points and minimize the QEF. If the error $E(w, x, y, z)$ from equation 1 is greater than ϵ, then we subdivide the cell into eight sub-cells and proceed recursively. The procedure stops when all of the leaves of the octree have error less than ϵ and constructs a minimal octree to approximate $f(x, y, z)$.

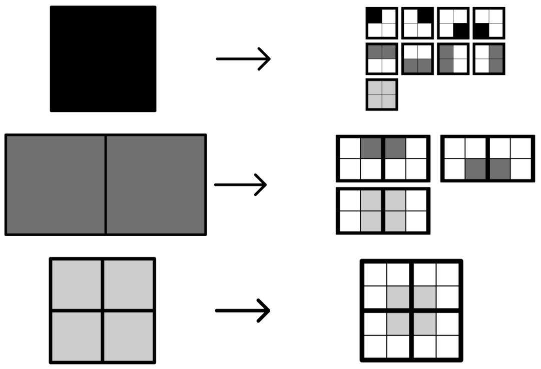

Figure 3. Recursive functions `faceProc` (black), `edgeProc` (dark gray) and `vertProc` (light gray) used in enumerating cells of the dual grid.

3. Topology creation

Given an octree, feature isolation generates vertices of the dual grid that the surface will be extracted from. Topology creation actually generates the topology of the dual grid. This dual grid is topologically dual to the octree. Therefore, for each vertex in the octree, a cell in the dual grid will be created whose vertices are the feature vertices inside of each cube in the octree containing that vertex. Figure 2 shows the topology of a dual grid created from an example quadtree where each vertex of the dual grid is placed at the center of its cube. Although our algorithm operates on octrees, the discussion in this section will focus on quadtrees for simplicity; however, the algorithms described here naturally extend to octrees.

Given a quadtree q, our task is to enumerate each cell of the dual grid in an efficient manner without any explicit neighbor finding in the quadtree. Our solution is a recursive traversal of q that enumerates tuples of all leaf squares that share a common vertex and is a simple extension of the quadtree traversal used in [4].

The traversal involves three recursive functions `faceProc[`q_1`]`, `edgeProc[`q_1`,`q_2`]` and `vertProc[`q_1`,`q_2`,`q_3`,`q_4`]`. Given an interior node q_1 in the quadtree, `faceProc[`q_1`]` recursively calls itself on the four children of q_1 as well as calling `edgeProc` on all four pairs of edge-adjacent children of q_1 and one call to `vertProc` on its four children. Given a pair of edge-adjacent interior nodes q_1 and q_2, `edgeProc[`q_1`,`q_2`]` recursively calls itself on the two pairs of edge-adjacent children spanning the common edge between q_1 and q_2 as well as making a single call to `vertProc` on the four children of q_1 and q_2 that touch the midpoint of this common edge. Finally, given four interior nodes q_1, q_2, q_3 and q_4 that share a common vertex, `vertProc[`q_1`,`q_2`,`q_3`,`q_4`]` recursively calls itself on the four children that meet at the common vertex. During these calls, if one of the q_i is a leaf, then its children cannot be used for subsequent recursive calls. In this case, a copy of q_i is passed to the recursive call for each child of q_i required by the recursion. Figure 3 depicts the mutually recursive structure of these three functions.

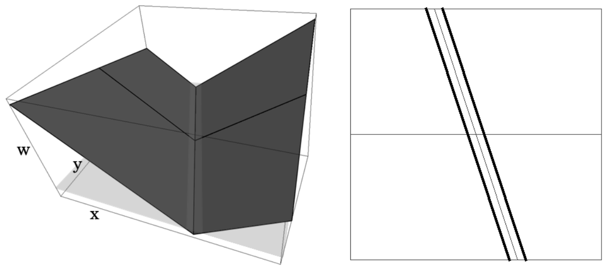

Figure 4. The dual grid of the *Max* of two linear functions with the plane $w = 0$ drawn transparently (left). The zero-contour of this function (bold) as well as the projections of the dual grid onto the contour plane (right).

The recursive calls to these functions terminate when all q_i are leaves of the quadtree. At this point in `vertProc`, we have four leaves sharing the same corner. We then construct a cell of the dual grid by connecting the vertices generated by feature isolation for each cell to topologically form a square. In adaptive cases, one of the q_i may actually be repeated, which causes the square to geometrically form a triangle. Note the running time of this method is linear in the size of the quadtree since there is one call to `faceProc` for each square in the quadtree, one call to `edgeProc` for each edge in the quadtree and one call to `vertProc` for each vertex in the quadtree.

As described above, calling `faceProc[`q_1`]` generates only cells that are interior to the root of the quadtree. To extend the dual grid to the boundary of the quadtree, we treat the quadtree as being the center square in a 3×3 degenerate rectangular grid. Four squares in this grid degenerate to edges of the quadtree while the remaining four squares degenerate to the vertices of the quadtree. Calling `edgeProc` with the root of the quadtree and each of its degenerate edge neighbors generates those tiles touching the corresponding edge of the quadtree. Likewise, calling `vertProc` at each vertex of the root with its three degenerate vertex neighbors generates the single dual cell touching the vertex. Figure 2 shows a dual grid generated using this method that contains vertices on the edges and corners of the grid.

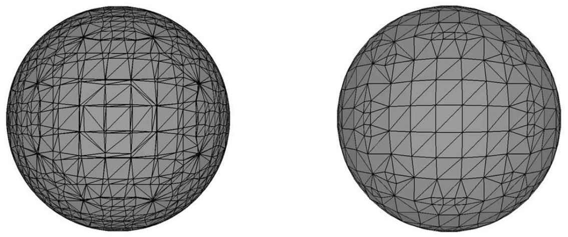

Figure 5. Contoured sphere without sliver elimination (left) and with sliver elimination (right).

4. Contouring dual grids

After generating the dual grid, we extract the surface using a simple extension of Marching Cubes to these dual grids. Marching Cubes was originally designed to operate only on axis-aligned cubes. However, the cells of our dual grid have arbitrary vertices in space. Our solution is to notice that each cell in the dual grid generated by the recursive algorithm in section 3 is topologically equivalent to a cube (although some vertices may be repeated due to the multi-resolution structure of the octree). Using the scalar values w at the vertices of the dual grid, we compute edge intersection points for each edge that contains a sign change. Then we use the look-up table provided by Marching Cubes to generate the topology of the surface interior to that cell.

Since the vertices of the dual grid lie on the features of $f(x, y, z)$, the contours of this dual grid also exhibit sharp features similar to those produced by Extended Marching Cubes and Dual Contouring. The principle advantage of Dual Marching Cubes over these methods is that the underlying octree used to produce an equivalent contour is much sparser.

To understand this phenomena, we consider a 2D example. Figure 4 (left) contains the distance function for two closely space, parallel lines shown on the right of the figure. These two linear functions are formed by taking the *Max* of the distance function for each line. The left of the figure shows the dual grid of $f(x, y)$ which consists of four quads where the height of the vertex is formed by the scalar value w. Contouring each of these quads yields four line segments independent of the spacing of the two parallel lines. In methods such as EMC and Dual Contouring, the density of the primal grid used to resolve the two separate contour lines of this function depends on the separation distance of the two contours. With Dual Marching Cubes, the grid used in contouring the function adapts to features of the distance function as opposed to features of the contour. This distinction often avoids the need for high levels of refinement to separate closely spaced features of the contour.

As a more complex 3D example, consider the thin-walled room of figure 1. This room is designed as a sequence of CSG operations based on planar primitives. The lower left mesh is the Marching Cubes contour of this distance function on a fine grid to reproduce the thin walls (note the rounding of sharp edges). The lower right mesh, produced by Dual Contouring, exactly reproduces the shape of the room but cannot simplify the numerous flat regions in the mesh due to the thinness of the room's walls. Our method computes an octree to approximate the CSG tree using the method of section 2.1. The resulting surface on the upper right was generated by Dual Marching Cubes by contouring the dual grid of that octree. This surface exactly reproduces the shape of the room but uses far fewer polygons. Note that as the thickness of the walls of this room decreases, the number of polygons used to contour this room using Marching Cubes and Dual Contouring increases dramatically while the number of polygons produced by Dual Marching Cubes remains roughly constant.

One drawback with using Marching Cubes to contour these dual grids is that it tends to produce numerous sliver polygons when the vertices of the dual grid lie close to the plane $w = 0$. The solution to this problem is to position the vertices of the dual grid to lie exactly on $w = 0$ when possible. The effect on the resulting contour is to eliminate most sliver polygons. To achieve this positioning, we first compute the minimizer $\hat{p}$ of the restricted QEF $E(0, x, y, z)$. If the residual at $\hat{p}$ is less than ϵ, $\hat{p}$ is used as the vertex for that cube. Otherwise, we perform feature isolation as before. Figure 5 shows the contours generated by Dual Marching Cubes illustrating sliver elimination.

Figure 6 depicts an example of a rocket constructed as a sequence of CSG operations and the approximating models using Dual Marching Cubes, Marching Cubes and Dual Contouring. The rocket is an interesting example because the thin fins on the bottom of the shape exhibit five-fold symmetry. While primal grids perform well when features are axis aligned, these fins do not conform to parameter lines on the primal grid. Consequently, methods such as Marching Cubes and Dual Contouring have a difficult time reproducing these thin features. On the other hand, Dual Marching Cubes constructs a dual grid that conforms to the fins and can reconstruct them accurately despite the fact that the fins do not align with the octree.

Figure 7 illustrates adaptive surface extraction using a horse and a dragon. On the horse, the polygons are more coarse on the body than along the neck and the legs of the horse. Despite the fact that the legs are thin compared to the rest of the horse, they are reproduced accurately. The dragon also shows an adaptive polygonalization where its face and feet contain finer polygons than the rest of its body. Though we have simply run Marching Cubes on these models without any multi-resolution extensions, the structure

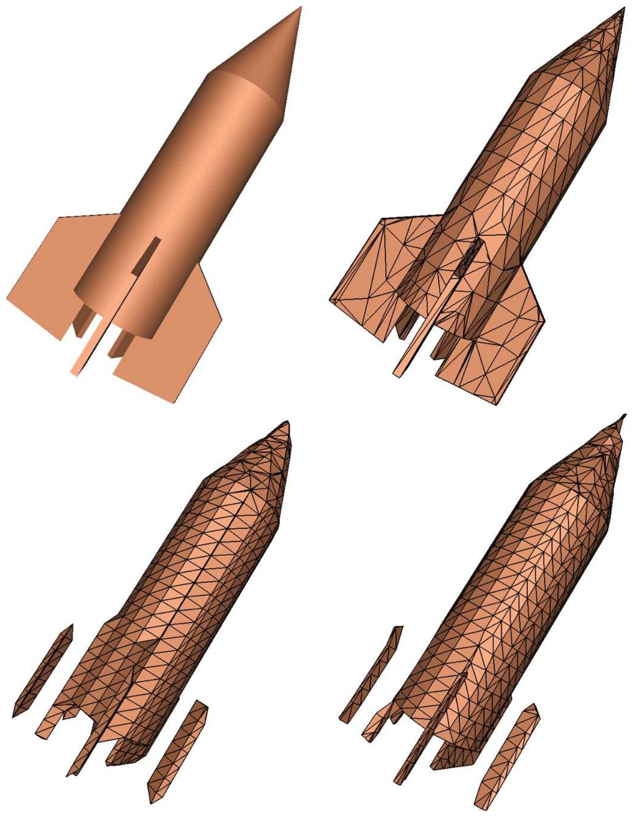

Figure 6. CSG model of a rocket (upper left) and models approximating the shape using the same number of polygons. Dual Marching Cubes (upper right), Marching Cubes (lower left) and Dual Contouring (lower right).

of the dual grid naturally generates these multi-resolution contours. Furthermore, since we use Marching Cubes to contour these models, the resulting surface is topologically manifold.

5. Implementation

When implementing this method, we abstract out the different possible inputs to our algorithm as a function that we can evaluate to determine the scalar value and gradient at our sample points. Since we are estimating sharp features using QEFs, the same problems arise for volumes as found in surface methods. For instance, when minimizing equation 1, the feature vertex may lie outside of the cube that generated it. In this case, we minimize the QEF over the boundary of the cube as in Lindstrom [8].

Also, notice that there is nothing that requires we place vertices at the features of the distance function. While this

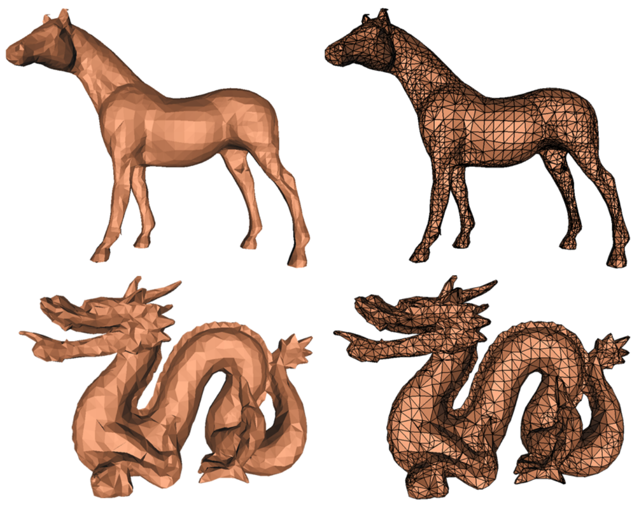

Figure 7. Adaptive surfaces extracted using Dual Marching Cubes.

placement reproduces sharp features, one could modify the feature isolation phase to simply sample $f(x, y, z)$ at the center of each cube. The resulting algorithm is a contouring algorithm similar to [10] except no special cases are generated in adaptive configurations and a manifold surface is always produced.

As far as performance is concerned, contouring dual grids takes no longer than contouring primal grids. Since dual grids conform better to the features of $f(x, y, z)$, sparser grids can be used to extract surfaces, which can actually decrease the time taken to contour. However, dual grid generation is still somewhat slow. Feature isolation took 0.1 seconds for figure 1, 2.5 seconds for figure 6 and several minutes for figure 7 using the octree construction algorithm from section 2.1 on a 3GHz Pentium.

One way of solving this problem is to realize that most of the dual grid does not directly contribute to the extracted surface. Only cells of the dual grid that contain a sign change will contain a piece of the surface. Therefore, it may be possible to develop an algorithm that restricts the feature isolation and topology creation phases to only those cells that contain a piece of the extracted surface. We anticipate the speed of this optimized method to be comparable with other popular methods.

6. Future Work

We believe that the concept of dual grid generation has applications outside of contouring. In fact, this grid generation actually extracts a piecewise linear approximation to a given function $f(x, y, z)$ over a cubical domain. With this in mind, dual grid generation should be an excellent method for adaptively tiling several intersecting parametric surfaces due to its ability to adapt edges of the dual grid to the intersection curves. In robotics, distance functions also play a crucial role in applications such as path planning. Therefore, another possible application of this dual grid would be to compute an accurate, adaptive approximation to the distance function and then perform path planning on the dual grid.

7. Acknowledgements

We'd like to thank Tao Ju for his help in creating several of the figures in the paper as well as the Stanford Graphics Laboratory and Cyberware for providing the models of the dragon and horse.

References

[1] L. Balmelli, C. J. Morris, G. Taubin, and F. Bernardini. Volume warping for adaptive isosurface extraction. In *Proceedings of the conference on Visualization '02*, pages 467–474. IEEE Computer Society, 2002.

[2] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 249–254. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, July 2000.

[3] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, pages 209–216, Los Angeles, California, August 1997. ACM SIGGRAPH / Addison Wesley.

[4] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. *ACM Transactions on Graphics*, 21(3):339–346, July 2002. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[5] T. Ju, S. Schaefer, and J. Warren. Convex contouring of volumetric data. *The Visual Computer*, 19(7–8):513–525, 2003.

[6] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature-sensitive surface extraction from volume data. In *Proceedings of SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 57–66. ACM Press / ACM SIGGRAPH, August 2001.

[7] P. Lindstrom. Out-of-core simplification of large polygonal models. In *Proceedings of SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 259–262. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, July 2000.

[8] P. Lindstrom and C. T. Silva. A memory insensitive technique for large model simplification. In *Proceedings of the conference on Visualization '01*, pages 121–126. IEEE Computer Society, 2001.

[9] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, volume 21, pages 163–169, Anaheim, California, July 1987.

[10] R. N. Perry and S. F. Frisken. Kizamu: A system for sculpting digital characters. In *Proceedings of SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 47–56. ACM Press / ACM SIGGRAPH, August 2001.

[11] R. Shekhar, E. Fayyad, R. Yagel, and J. F. Cornhill. Octree-based decimation of marching cubes surfaces. In *IEEE Visualization '96*, pages 335–344. IEEE, October 1996.

[12] R. Shu, C. Zhou, and M. S. Kankanhalli. Adaptive marching cubes. *The Visual Computer*, 11(4):202–217, 1995.

[13] G. Varadhan, S. Krishnan, Y. Kim, and D. Manocha. Feature-sensitive subdivision and iso-surface reconstruction. In *IEEE Visualization 2003*, pages 99–106. IEEE, 2003.