Dual Contouring: “The Secret Sauce”

Scott Schaefer, Joe Warren
Rice University*

Abstract

This paper provides several contributions related to dual contouring implicit data including solving rank deficient QEF’s, reducing memory requirements, and performing fast polygon updates during CSG. First, we describe our method for solving QEF’s and provide a method using mass points that improves vertex placement in the rank deficient case. This improvement leads to a technique for handling vertex placement in the presence of non-manifold sign configurations. Next, we provide a method for reducing the space requirements for storing the implicit data needed to generate the contour. Finally, we describe our method for storing the geometry data in a format suitable for fast display on modern graphics hardware as well as techniques updating the data-structure in constant time during CSG operations.

CR Categories: I.3.5 [Computation Geometry and Object Modeling]: CSG—Curve, surface, solid and object representations

Keywords: implicit functions, contouring, crack prevention, quadratic error functions, polyhedral simplification

1 Introduction

One technique for representing surfaces is implicit modeling. In this paradigm the surface is modeled as the zero contour of a 3D function. This function is typically represented using a 3D grid of data where the function is sampled at the vertices of the grid. In practice these grids of data arise in a variety of applications such as medical imaging and visualizing geological data.

There are several methods capable of extracting surfaces from these grids of data; however, one of the most well-known is the Marching Cubes algorithm of Lorenson and Cline [Lorenson and Cline 1987]. The algorithm proceeds as follows: for each edge in the grid that contains a sign change (i.e.; the vertices on its endpoints are of different sign), place a vertex on that edge where the contour intersects it. Marching Cubes then provides a table that stores the triangulation of the vertices on the edges of the grid and is indexed by the eight signs at the corners of a cube. For each cube in the grid, look up its sign configuration in the table and generate polygons according to the triangulation stored there. Notice that this method produces vertices that lie on the edges of the grid and polygons are produced internal to the cubes in the grid. We call these methods *primal* methods.

*e-mail: {sschaefe,jwarren}@rice.edu

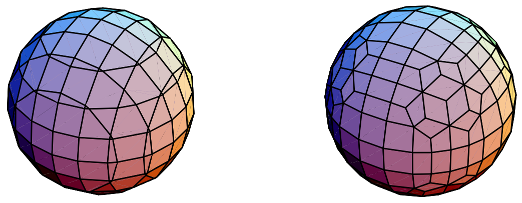

Figure 1: A sphere contoured using Marching Cubes (left) and Dual Contouring (right).

Dual methods [Gibson 1998; Perry and Frisken 2001] take a different approach to contouring producing vertices on the surface interior to each of the cubes in the grid and polygons dual to edges of the grid. SurfaceNets [Gibson 1998] is an example of a dual method. For each cube that contains a sign change, a vertex is generated at the centroid of the intersection points on the edges that would have been generated by Marching Cubes. Then, for each edge in the grid that exhibits a sign change, a polygon is produced connecting the vertices of the four cubes containing that edge. We call this a dual approach to contouring because for each vertex generated by this method, Marching Cubes generates a polygon; and for each vertex produced by Marching Cubes, this method generates a polygon. Therefore, the polygonal contours created are dual to each other.

Dual techniques introduce a number of advantages over traditional primal methods. Because vertices are placed interior to cubes and not on the edges of the grid, there is more freedom in the positioning of those vertices. Dual techniques also produce polygons of more uniform size than primal methods, which can sometimes generate very small polygons depending on how the contour intersects the grid. Also, primal methods generate a noticeable gridding effect in the surfaces because the edges of the polygons are constrained to lie along the grid planes as seen in figure 1.

In Siggraph 2002, the authors of this paper introduced a new technique that took a dual approach to contouring as well [Ju et al. 2002]. Entitled “Dual Contouring,” this method provided several advantages over traditional contouring methods. First, the grid of data was represented as a signed octree similar to Adaptive Distance Fields [Frisken et al. 2000; Perry and Frisken 2001], which resulted in substantially reduced memory requirements. Second, the method augmented the data representation with hermite data (exact intersection points and normals) as was done in [Kobbelt et al. 2001] to reproduce sharp features such as edges and corners accurately (see figure 2). From this hermite data, quadratic error functions (QEF’s) were generated for each cube in the octree, which enabled the method to collapse the polygonal models upward in a Lindstrom-like [Lindstrom 2000] manner to reduce the number of polygons present. The authors then provided a recursive method for generating a closed contour for an unrestricted, signed octree.

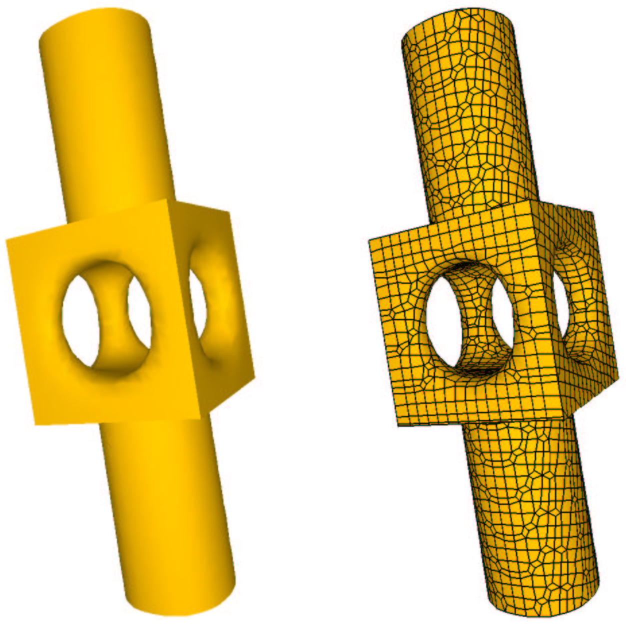

Figure 2: Reporduction of sharp features. A mechanical part (left) and its dual contour (right).

This paper expands on the techniques presented in the original Dual Contouring paper [Ju et al. 2002]. Due to lack of space, not all of the details could be written in the original paper on Dual Contouring. Specifically, we concentrate on details important from an implementation perspective. First, we discuss our implementation of QEF's and describe how the rank deficient case is detected and solved. This method then leads to a solution for positioning vertices in the presence of non-manifold geometry. Next, we discuss space considerations including a description of the data-structures used to reduce the memory requirements of our method. Finally, we end with a technique for polygon insertion and deletion during CSG operations that is performed in constant time and generates a data-structure suitable for fast display on modern graphics hardware.

2 Quadratic Error Functions

Dual Contouring utilizes the hermite data on the edges with a sign change to position vertices inside of cubes. Each piece of hermite data is the equation for the tangent plane of the surface where the surface crosses that edge and is represented as an exact intersection point, p_i, and a normal, n_i. The method places a vertex inside of each cube containing a sign change at the solution to equation 1 that minimizes $E[x]$. This equation can also be represented in matrix form as the least squares solution to $Ax = B$ where A is a $n \times 3$ matrix whose rows are n_i and B is a $n \times 1$ matrix whose entries are $n_i \cdot p_i$. Notice that this error function is quadratic in terms of the variable x and, hence, the name QEF.

$$E[x] = \sum_i (n_i \cdot (x - p_i))^2 \qquad (1)$$

In practice, the error function, $E[x]$, is not represented as a list of plane equations because the space required to store this function is linear in the number of planes. Garland [Garland and Heckbert 1997] popularized a technique for mesh simplification that stored this error function using a normal equation form, which resulted in constant space usage. In this representation, the only matrices stored are A^TA, A^TB, and B^TB. Using the matrix form for equation 1, $E[x]$ can be written as $(Ax-B)^T(Ax-B)$, which is then expanded into the form

$$E[x] = x^TA^TAx - 2x^TA^TB + B^TB \qquad (2)$$

where A^TA is a symmetric 3×3 matrix, A^TB is a 3×1 matrix, and B^TB is a single scalar. Therefore, only 10 quantities need to be stored in order to represent the error function, $E[x]$. This representation also has the advantage that merging two QEF's together is just a matter of adding the respective entries of the three separate matrices and uses only a constant amount of space.

One disadvantage that the representation in equation 2 has is that it is numerically unstable. Forming the matrix A^TA squares the condition number (a quantity measuring the numerical stability) of the matrix. Therefore, this form of the error function exhibits poor accuracy when evaluating $E[x]$.

Dual Contouring introduced a new, numerically stable method for the QEF based on the QR decomposition of a matrix. This method computes an orthogonal matrix Q as a sequence of Givens rotations such that the product of Q and $(A\,B)$ is of the form

$$\begin{pmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots \end{pmatrix} = \begin{pmatrix} \hat{A} & \hat{B} \\ 0 & r \\ 0 & 0 \\ \dots & \dots \end{pmatrix} \qquad (3)$$

where $\hat{A}$ is an upper triangular 3×3 matrix. Solving the new set of equations $\hat{A}x = \hat{B}$ yields the same solution as equation 2. This technique also has the advantage that evaluating $E[x] = (\hat{A}x - \hat{B})^T(\hat{A}x - \hat{B}) + r^2$ is much more numerically stable. However, this stability is not free. While merging two QEF's using the normal equations takes only 10 operations, merging two QEF's using the QR representation takes approximately 150 operations.

2.1 Solving QEF's

There have been many different methods proposed for solving QEF's [Lindstrom 2000; Lindstrom and Silva 2001; Kobbelt et al. 2001; Garland and Heckbert 1997]. The majority of these methods rely on using the *pseudoinverse* of the matrix A^TA, which finds the solution x with minimal L_2 norm. The pseudoinverse, M^+, of a matrix M has the property that the L_2 norm of the matrix $MM^+ - I$ is minimized. This pseudoinverse is defined for any matrix and, for a matrix of full rank, the pseudoinverse of M is M^{-1}.

Typically, the pseudoinverse is calculated using the *SVD* decomposition of the matrix. The singular value decomposition of a matrix, M, results in three matrices of the form $M = U^TDV$ where U and V are orthogonal matrices and D is a diagonal matrix. The elements in the diagonal of D, s_i, are called the *singular values* of the matrix M. The SVD is particularly simple to calculate when the matrix is of the form A^TA because $U = V$ and the rows of U are the eigenvectors of A^TA and the singular values in D are the eigenvalues of A^TA.

Since U and V are orthogonal matrices, the inverse of M is then $M^{-1} = V^TD^{-1}U$. However, when M is rank deficient, some of the singular values will be 0 and D^{-1} cannot be explicitly formed. Because D is a diagonal matrix, D^{-1} is also diagonal with each entry in its diagonal as $\frac{1}{s_i}$. However if a singular value, s_i, in D is 0 or close to 0, $\frac{1}{s_i}$ results in a large, if not infinite, value in D^{-1}. In this situation the pseudoinverse performs a method for rank detection. For each entry in D, if the ratio of that singular value, s_i, to the maximum singular value, s_{max}, is less than some constant, then the value in D^{-1} is truncated to 0; otherwise, it is $\frac{1}{s_i}$.

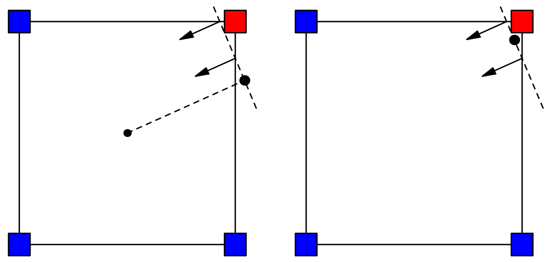

Figure 3: Minimizing towards center of square (left). Mass point approach (right).

In Dual Contouring, we solve for the minimizer of the QEF in several steps. We form the QEF by first initializing a 4x4 matrix to 0. Then, for each edge in a cube that contains a sign change, we append the plane equation described by the hermite data on that edge to the bottom of the matrix and perform Given's rotations on this 5x4 matrix to bring it into upper triangular form. Notice that the orthogonal matrix Q is never explicitly formed during this procedure.

To solve the QEF for the minimizer, we extract the three submatrices $\hat{A}$, $\hat{B}$, and r. Next we form $A^TA = \hat{A}^T\hat{A}$ and compute the SVD of that matrix. As noted above, finding the SVD only requires that we compute the eigenvalues and eigenvectors of the matrix A^TA. When computing the pseudoinverse, A^TA^+, we use a different method of truncation on the singular values. Instead of truncating based on the relative magnitude of the value to the largest singular value, we truncate based on the absolute magnitude of that singular value.

Truncation based on the absolute magnitude of the singular values is not typically done. However, we use normalized vectors in the hermite as opposed to area weighted normals. Several papers [Hoppe 1999] have shown area weighted normals to give superior results in terms of polygons simplification. In Dual Contouring the normals are associated with edges in the grid and only give an indication of the polygon that intersects that edge and provides no information about the surface contained within the adjacent grid squares. Therefore, we felt that the benefits achieved by area weighted normals in polygon simplification would not translate into implicit modeling. In fact, we found that using area weighted normals occasionally produced artifacts where features were lost due to incorrect truncation of singular values. By using normalized vectors, the extracted singular values will be proportional to the number of vectors that align in the direction of the corresponding eigenvectors. Therefore, truncating singular values at an absolute magnitude of .1 gives a very robust method of detecting noise present in the data.

One problem that using the pseudoinverse introduces is that it finds the point on the solution space with minimal L_2 norm, which means the closest point to the origin. When the system of equations is underdetermined (as is the case in a flat area or along an edge), then the answer produced can be located far from the cube that generated it. Kobbelt [Kobbelt et al. 2001] and Lindstrom [Lindstrom 2000] combat this problem by translating the center of the cube that generated the QEF to the origin, solving the equations, and then inverting the translation. This process effectively finds the solution with minimal distance to the center of the cube.

Ideally, whatever point on the solution space is chosen, it should have the property that if the solution space intersects the cube that generated the QEF, then the point should lie inside of that cube. Figure 3 (left) illustrates a 2D case where this property does not hold. Here the line that intersects the square represents the solution

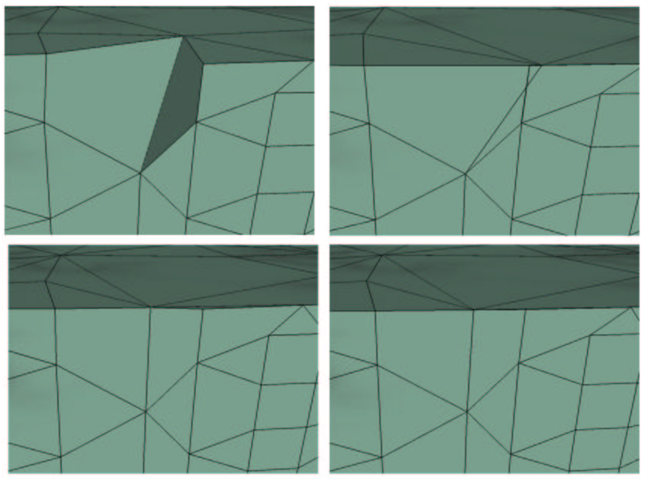

Figure 4: Mass points without feature dimension (upper left). Minimized vertices using QEF's without mass point feature dimension (upper right). Mass points using feature dimension (lower left). Minimized vertices using QEF's with mass point feature dimension (lower right).

space. Notice that by minimizing the distance to the center of the cube (shown as a projection), a point outside of the cube is generated even though part of the solution space is interior to the square. This method is only guaranteed to yield the correct result if the solution space intersects a sphere centered at the middle of the cube with diameter equal to the length of the sides of the cube.

In Dual Contouring we take a different approach to solving this problem. Instead of finding a point in the solution space that minimizes the distance to the center of the cube, we minimize towards a that we call the *mass point*. The mass point for a cube is the average of the exact intersection points on the edges of the cube. Because the mass point is a convex combination of points on the edges of the cube, it has the property that it always lies inside of the cube that generated it. We solve for the minimizer, x, of the equation $Ax = B$ with minimal distance to a point, p, in space by letting $x = c + p$ and computing

$$c = (A^TA)^+(A^TB - A^TAp). \qquad (4)$$

When the system has full rank $(A^TA)^+ = (A^TA)^{-1}$ and the solution is the least squares solution shown previously. Figure 3 shows a comparison between these two methods. In this figure the mass point solution (right) generates a minimizer equal to the mass point, which is interior to the cube. This newer method relies on the assumption that the solution space will not be far from the intersection points on the edges and then minimizes towards a point inside of the cube close to the solution.

The addition of a mass point to the QEF adds an additional 4 floats to the representation: $\{\sum x_i, \sum y_i, \sum z_i, n\}$. When merging QEF's upward in the octree during simplification the respective components of the mass point are summed together. The problem with this method is that when large numbers of cubes are collapsed together, it is possible to produce a mass point far from the actual solution space. These aggregate mass points then suffer from the same problems as minimizing towards the center of the cube.

To combat this effect, we introduce an alternate method for combining mass points using feature detection. In addition to the data already stored for the mass point, we also store the dimension of the mass point. This dimension is equal to the dimension of the feature present in the QEF for that mass point (1 = plane, 2 = edge, 3 = corner). This dimension is computed during the pseudoinverse

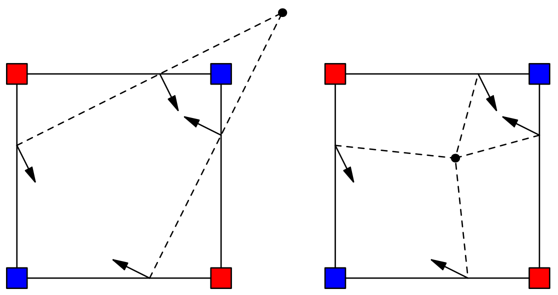

Figure 5: Non-manifold geometry generates a undesired point outside of the square (left). Using the mass point prevents spikes in the surface (right).

and is equal to three minus the number of singular values truncated. The algorithm for merging two mass points is then

1. If the mass points are of the same dimension, merge the two mass points like before by summing their components.

2. If the mass points are of differing dimensions, set the combined mass point equal to the mass point of the highest dimension. The dimension of the new mass point is also equal to that of the higher dimensional mass point.

This algorithm has the property that the resulting mass point is only the average of the points that generated the feature of highest dimension. Since the minimizer of the merged QEF is likely to be near the feature of the highest dimension, this method produces a point inside of the cube close to the feature. In our testing this technique solved many problems that we experienced with point placement such as polygons folding back onto themselves.

Figure 4 shows a piece of a temple model generated using Dual Contouring that illustrates the difference between the two techniques. The left side of the figure depicts the vertices of the temple positioned at the mass points without using feature dimension (top) and with feature dimension (bottom). Notice that the mass points using the dimensional quantity are positioned much closer to the solution space than the mass points without the feature dimension. The right side of the figure shows the minimized vertices using the QEF's and the mass points from the left side of the figure. The minimized vertex on top does not use feature dimensions and causes a fold in the mesh. The minimized vertex on the bottom uses feature dimensions and is positioned on the solution space inside of the cube that generated the vertex so that no such folding occurs.

Another benefit that this method has is that it provides a way of handling point placement for non-manifold geometry. In the case where multiple, separate pieces of a surface intersect the same cube, QEF's can do a poor job of positioning the minimizer (see figure 5, left). Since Dual Contouring uses a signed grid to determine the topology of the resulting surface, non-manifold configurations are simple to detect using the signs at the eight corners of a cube. In these cases, positioning the point at the minimizer of the QEF is usually not desired and can result in "spikes" in the surface. In these cases, we use the position of the mass point to position the vertex inside of the cube. Using the mass point's position has the advantage that it is a point inside of the cube so that no folding occurs and it is near the portions of the surface that intersect that cube.

model	size	interior	mixed	empty/full
Chinese cube	128^3	12.5%	40.2%	47.3%
temple	256^3	12.5%	38.5%	49.0%
david	512^3	12.5%	38.0%	49.5%

Figure 6: Percentage of different types of nodes for three different models.

3 Space Considerations

When using a volumetric representation for models, space can quickly become an issue. We use several techniques to reduce space usage in Dual Contouring. Instead of using a uniform grid to describe the implicit surface, we use an octree with empty and solid space collapsed maximally. This data structure results in a large space savings because the space required to store the grid has been reduced from being proportional to the volume of the object to the surface area instead.

Even with the octree structure, we managed to reduce the space requirements more. We identified three different types of nodes present in the octree and specialized the data stored in each type of node to reduce wasted space. The three different types of nodes inside of the octree are interior nodes, homogeneous leaves, and heterogeneous leaves. While these nodes have some data elements in common, there are many fields that are specific to that type of node.

All three types of nodes store the depth of the node in the octree (for the adaptive contouring algorithm). Interior nodes store 8 pointers to the children (32 bytes), a QEF (68 bytes), and the signs at the corners of the cube (8 bytes). The QEF of an internal node stores the aggregate QEF for all of the node's children. Homogeneous nodes are leaves in the octree that are either completely empty or completely full and have been collapsed maximally. These nodes only store a single number (1 byte) representing the sign at all eight corners of the node. Heterogeneous nodes are leaves of the octree that contain a sign change and, therefore, a portion of the surface being generated. These leaves store the signs at the corners of the cube (8 bytes), the QEF (68 bytes), and twelve pointers to the hermite data on the edges (48 bytes). The edge pointers are all set to NULL unless there is a sign change on that edge, in which case the hermite data (16 bytes) populates that pointer. We store the hermite data as a normal $< n_x, n_y, n_z >$ and an alpha component, which indicates where the exact intersection point is located on that edge.

Figure 6 displays a table listing the percentages of each type of node for three different models. Notice that the amount of homogeneous nodes (empty/full) dominates the two other types. By specializing the data representation for the different types of nodes in the octree, we reduce the space consumption of the data structure by approximately fifty percent. The space requirements can be reduced even further by removing the QEF's from the data structure resulting in a reduction of another twenty percent. These QEF's can be computed dynamically from the hermite data on the edges, but this approach requires a tradeoff between size and processing time.

4 Polygon Generation

Because Dual Contouring uses an implicit representation for surfaces, approximate CSG operations can be applied to the surfaces very quickly. Due to our use of an octree representation for the data, these operations only take time proportional to the combined surface area affected by the CSG operation as opposed to the affected volume. However, polygons need to be added and removed

Index	Data	Polygons	Node
0	{x,y,z}	{1,5,2,0}	ptr
…	…	…	…

Figure 7: Structure of the vertex array. *Data* is information sent to graphics card. *Polygons* is a list of polygons containing this vertex. *Node* is a pointer to the node that generated this vertex.

Index	Vertices
0	{0,1,2,3}
…	…

Figure 8: Structure of the topology array. *Vertices* is a list of four indices into the vertex array for the vertices that make up this quadrilateral.

from the previous surface in order to display the new surface. Furthermore, the polygons and vertices produced need to be stored in a format that allows for fast display on modern graphics hardware. We provide an algorithm used in Dual Contouring that achieves constant time insertion and deletion of polygons and vertices from a tightly pack topology/geometry data structure optimized for display on modern graphics cards.

In order to achieve higher frame rates, graphics hardware prefers to receive geometry data in a topology/geometry representation, which is a tightly packed list of polygons and vertices respectively. This structure allows the hardware to transform and light each vertex exactly once. In our representation, we need additional pieces of data stored with vertices that is not sent to the graphics card. To maintain a tightly packed structure, we use auxiliary arrays to store the additional data indexed in the same manner as the list of vertices. However, for simplicity, we will described the data structure as being a single entity in an array, but it should be understood that the extra information is stored separately.

4.1 Data Structures

In Dual Contouring, each heterogeneous node (and possibly interior nodes if the model is collapsed) contains an integer that indexes into a vertex array for the vertex contained within that node. Each entry in the vertex array contains the data sent to the graphics card (position, normal, texture coordinates, etc...), a list of indices into the topology array for each quad containing this vertex, and a pointer to the node that generated this vertex (figure 7). The topology array simply stores a list of four indices into the vertex array in each entry (figure 8).

Constructing these data structures is straight forward from the octree. A single pass over the octree generates the vertices in the vertex array with the node pointers populated, but no polygons in its list. Next, the multi-resolution contouring algorithm described in Dual Contouring is performed. For each polygon generated, it is appended onto the end of the topology array and its index is added to the list of polygons for the four vertices in that polygon.

During a CSG operation, vertices are added and removed from the data structure dynamically corresponding to addition or deletion of heterogeneous nodes in the octree. Removing a vertex also has the effect of removing the polygons connected to that vertex. After the CSG operation is finished, we perform a truncated version of the full polygon generation pass to add the new polygons into the model. The alterations of the vertex and topology arrays caused by a CSG operation are performed using four methods: *remove polygon*, *remove vertex*, *add polygon*, and *add vertex*.

Remove polygon takes an index of the polygon to remove from the topology list as input. The procedure simply walks over the four vertex indices in the list for that polygon and requests that the vertex removes that polygon's index from its list of polygons. Next, the polygon is removed from the polygon list by copying the polygon at the last position in the topology list over the entry in the list being deleted. Finally, the method processes each vertex index in the new polygon's list and requests that those vertices renumber the polygon in its list from the previous index of that polygon to its new index. This function ensures that the data structure is consistent and that the polygons are tightly packed in the topology list.

Remove vertex is the function called during a CSG operation when a heterogeneous node is modified or deleted. This function processes the list of polygons stored in the vertex to be removed from the highest index to the lowest index. For each polygon index in the vertex's list, *remove vertex* calls *remove polygon* on that index. Processing the polygons from highest index to lowest index eliminates problems arising from renumbering the polygons in the deleted vertex's polygon list. Next, the method copies the vertex from the last position in the vertex array to the index of the deleted vertex. The vertex index of the node stored in the moved vertex is then updated to point to the vertex's new position. Next, for each polygon in the moved vertex's list, the procedure request that that polygon renumbers the index of the vertex in the old position to the new index.

When a CSG operation adds a new heterogeneous node, a new vertex is added into the vertex list by appending the vertex to the end of the vertex list with an empty polygon list and a pointer to the node that generated it. Similarly, adding a polygon appends the polygon to the end of the topology list and requests that each referenced vertex in the vertex array add that new polygon to its polygon list.

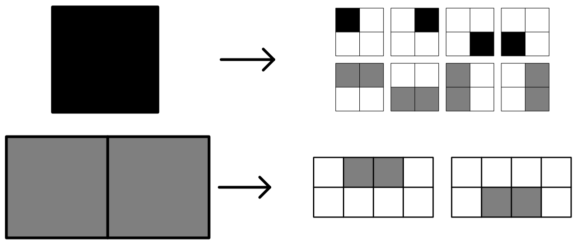
Figure 9: Recursive calls used to generate a closed, complete contour for a multi-resolution octree.

4.2 Recursive Polygon Generation

During a CSG operation, any nodes modified by the operation (including new nodes) are marked as well as all of the parents of those nodes up to the root. When a CSG operation finishes, old vertices have been deleted from the vertex array, new vertices have been added, and polygons associated with the old vertices have been deleted from the topology list. However, the CSG operation has not created the new polygons in the affected regions. In order to generate those polygons, we perform a truncated version of the recursive method described in Dual Contouring.

Dual Contouring introduced a new, recursive method for contouring multi-resolution, unrestricted octrees. The recursive calls for contouring a quadtree in 2D are shown in figure 9. Dark regions in the figure correspond to a call to *processFace* and gray regions to *processEdge*. Starting at the root of the quadtree with a call to *processFace*, the recursive calls continue by generating four new calls to *processFace* on its four children and four calls to *processEdge*. Each *processEdge* call produces two new calls to *processEdge* on two pairs of the children as shown in the figure. *ProcessFace* terminates when the node is a leaf. *ProcessEdge* terminates when both of its arguments are leaves and then examines the shared minimal edge (an edge in the grid that contains no smaller edge). If that minimal edge contains a sign change, then *processEdge* generates a line connecting the two minimizers of the corresponding grid squares. This process continues and generates a closed, complete contour.

Our truncated contouring pass uses the same recursive calls as the full polygon generation pass. The only change made is to the termination criteria. *ProcessFace* now terminates whenever its argument is a leaf or when the node being processed was not marked as changed by the CSG operation. Likewise, *processEdge* continues until both of its arguments are leaves or until the function finds that neither node was modified by the CSG operation. By truncating the polygon generation pass, this new procedure only takes time proportional to the affected surface area of the CSG'd region.

5 Conclusions

We have provided insight into many of the details of the original Dual Contouring paper that were ommitted or shortened. We described our implementation of the quadratic error metric (QEF) in more detail including our mass point solution for the rank deficient case. Since it is simple to detect non-manifold vertices using implicit geometry, we also provided a method for positioning vertices in the presence of non-manifold using these mass points. Next, our data-structures used to store the implicit representation were revealed and we illustrated how we achieved the space savings quoted in our paper. Finally, we provided a technique for inserting and deleting polygons in constant time and a fast, truncated polygon generation pass that reconstructs the surface after a CSG operation.

References

FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, 249–254.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH 97*, ACM SIGGRAPH / Addison Wesley, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, 209–216.

GIBSON, S. F. F. 1998. Using distance maps for accurate surface reconstruction in sampled volumes. In *1998 Volume Visualization Symposium*, IEEE, 23–30.

HOPPE, H. 1999. New quadratic metric for simplifying meshes with appearance attributes. In *IEEE Visualization 1999*, 59–66.

JU, T., LOSASSO, F., SCHAEFER, S., AND WARREN, J. 2002. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 339–346.

KOBBELT, L. P., BOTSCH, M., SCHWANECKE, U., AND SEIDEL, H.-P. 2001. Feature-sensitive surface extraction from volume data. In *Proceedings of SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, 57–66.

LINDSTROM, P., AND SILVA, C. 2001. A memory insensitive technique for large model simplification. In *IEEE Visualization 2001*, 121–126.

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

LORENSON, W., AND CLINE, H. 1987. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics 21*, 4, 163–169.

PERRY, R. N., AND FRISKEN, S. F. 2001. Kizamu: A system for sculpting digital characters. In *Proceedings of SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, 47–56.