

Experimental Results on MergeSharp

Arindam Bhattacharya and Rephael Wenger
The Ohio State University, Columbus, Ohio, USA

OSU Technical Report
OSU-CISRC-3/13-TR05

May 2, 2013

Abstract

A number of papers present algorithms to construct isosurfaces with sharp edges and corners from hermite data, i.e. the exact surface normals at the exact intersection of the surface and grid edges. We discuss some fundamental problems with the previous algorithms and describe a new approach, based on merging grid cubes near sharp edges which produces significantly better results. Our algorithm requires only gradients at the grid vertices, not at each surface-edge intersection point. We also give a method for measuring the correctness of the resulting sharp edges and corners in the isosurface.

1 Introduction

A scalar field is a mapping $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ which assigns a scalar value to each point in \mathbb{R}^3 . An implicit surface is a set $f^{-1}(\sigma)$ for some real $\sigma \in \mathbb{R}$. An isosurface with isovalue σ is a polygonal approximation to the implicit surface $f^{-1}(\sigma)$.

Various algorithms have been proposed for constructing isosurface approximations to a surface $f^{-1}(\sigma)$ with sharp edges and corners from distance fields [GK04, HWCO05, KBSS01, ZHK04] or from an explicit definition of f [AB03, JLSW02, HWCO05, VKKM03, SW04]. All these algorithms rely on the ability to compute the exact intersection point of the isosurface and each grid edge and an exact surface normal at that intersection point. Ju et. al. [JLSW02] coined the term “hermite” data to de-

scribe such inputs.

The algorithms listed above have a number of drawbacks. Firstly, two of the algorithms have fundamental problems when surfaces or portions of surfaces are not oriented along grid axes. The algorithms in [KBSS01] and [JLSW02] generate an isosurface vertex for each grid cube containing a positive vertex and a negative vertex. When the isosurface vertices are restricted to the generating cube, notches can appear along the sharp edges and sharp corners can be cut off. When these vertices are allowed to extend into neighboring grid cubes, the isosurfaces triangle or quadrilaterals can become degenerate. The isosurface vertices can also become incorrectly ordered along sharp edges. The incorrect ordering creates folds and overlapping isosurface triangles in the isosurface mesh. The software Polymender by Ju [Ju04] creates isosurfaces with degenerate and overlapping mesh triangles. These problems are further described in Section 5.

Algorithms [VKKM03, ZHK04] don’t have the problems discussed in Section 5 because they compute multiple intersections of a grid edge and the isosurface to represent thin or sharp features. Algorithms [AB03, GK04, HWCO05] increase the grid resolution until such problems disappear. Either approach exacerbates a second problem.

The algorithms listed above rely upon precise calculation of both the intersection point and the normal. They have little tolerance for approximation errors in those values. Those algorithms which compute multiple intersections or increase the grid resolution create small thin polygons near sharp features. Such polygons are very sensitive

to approximation error.

The last problem is that all these algorithms are restricted to hermite data or variations such as a signed distance field which permit the computation of surface normals along grid edges.

In [CDR07], Cheng, Dey and Ramos describe a method for meshing piecewise smooth complexes using protecting balls around the sharp or boundary edges of those complexes. We apply this idea to isosurface reconstruction by merging grid cubes around sharp edges and corners, creating regions which act like protecting balls. A single isosurface vertex is placed in each such region.

By placing a single isosurface vertex near the center of each region, we guarantee that the isosurface vertices on sharp edges and corners are well separated from any other isosurface vertices. Separating these isosurface vertices avoids the creation of degenerate quadrilaterals or the incorrect ordering of isosurface vertices along sharp edges. It also avoids the creation of notches along sharp edges.

The separation of isosurface vertices on sharp edges makes the sharp edge much less sensitive to changes in vertex location. Our algorithm is tolerant of small approximation errors in the intersection location or normals.

Because our algorithm is tolerant of approximation errors in isosurface vertex location, we no longer require hermite data as input. Instead we can use gradient grid data, regular grids with scalar values and gradients at each of the grid vertices. We compute isosurface vertex locations directly from this data, using gradients from neighboring cubes to increase reliability. As shown in Section 11, our algorithm is tolerant of noise in this gradient data.

While the central difference formula approximates gradients from a scalar grid [CK07], those approximations are very poor near sharp edges and corners where there are discontinuities in the gradient field. Nevertheless, the tolerance of our algorithm to noisy gradients suggests that more sophisticated techniques may produce gradients which are reliable enough for our algorithm.

Previous papers on reconstructing isosurfaces with sharp edges and corners lacked any quantitative measure of the quality of the reconstruction. The lack of such measure makes it difficult to evaluate the claims of these papers or compare the results of their algorithms in any systematic way. In Section 10, we propose a simple method for measuring the quality of the reconstructed sharp edges

and corners. Our evaluation method helps us quickly find errors in the sharp edge and corner reconstructions and allows us to evaluate our algorithm on numerous test data sets without requiring visual inspection of the results of each test. It also permits us to test and compare parameter changes to our algorithm and to compare our algorithm with Polymender.

Our paper contains three major contributions:

1. We present a new algorithm based on cube merging for constructing isosurfaces with sharp edges and corners. Our algorithm solves some fundamental problems with previous techniques and is significantly more robust.
2. We show how gradient grid data, not just hermite data, can be used to calculate the locations of isosurface vertices on sharp edges and corners.
3. We present a simple method for evaluating the quality of our reconstruction of sharp edges and corners, and evaluate our algorithm using that method.

2 Definitions

A scalar grid vertex is *negative* if its scalar values is less than the isovalue. A grid vertex is *positive* if its scalar values is greater than or equal to the isovalue.

A grid edge is *bipolar* if one endpoint is negative and one endpoint is positive.

A grid facet is *ambiguous* if two diagonally opposite vertices are positive and the other two vertices are negative.

A grid cube \mathbf{c} is a *vertex neighbor* of grid cube \mathbf{c}' if \mathbf{c} shares a vertex with \mathbf{c}' .

3 Related Work

The Marching Cubes Algorithm [LC87] by Lorensen and Cline places all the isosurface vertices on grid edges. If the isosurface is approximating a level set $f^{-1}(\sigma)$ which has sharp edge or corner inside a grid cube, then isosurface vertices would have to be placed inside the grid cube to model the sharp edge or corner. Since the Marching Cubes Algorithm does not place isosurface vertices inside

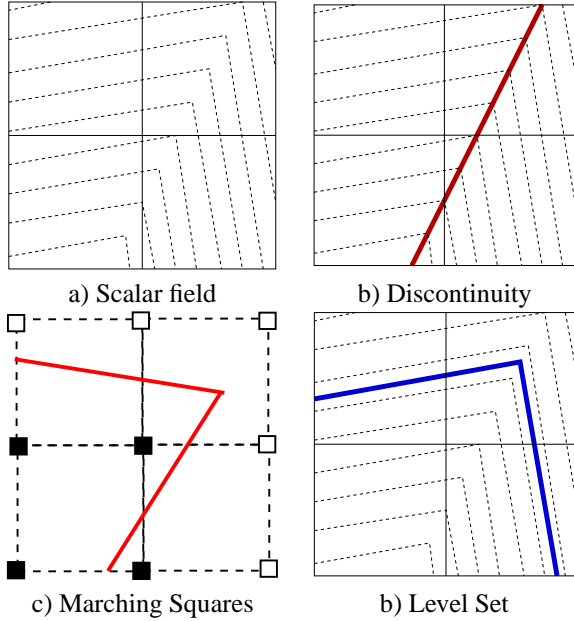


Figure 1: a) Non-smooth scalar field. b) Discontinuity (red) in the gradients. c) Isocontour (green) produced by the Marching Squares Algorithm. d) Level set (blue) with sharp corner.

grid cubes, it will not reconstruct the sharp feature but instead create a beveled edge or corner. (See Figure 1.c.) Moreover, the beveling will depend upon the intersection of the level set with the grid edges, not upon any intrinsic properties of the level set.

Instead of adding isosurface vertices on grid edges, dual contouring algorithms place isosurface vertices inside grid cubes intersected by the isosurface. Quadrilaterals connect isosurface vertices in the four adjacent grid cubes which share a common grid edge. Gibson [Gib98a, Gib98b], gave the first dual contouring algorithm, placing at most one isosurface vertex inside each grid cube. Later, Nielson [Nie04] described a dual contouring algorithm, Dual Marching Cubes, which sometimes adds multiple isosurface vertices inside a grid cube.

Because the dual contouring algorithms add isosurface vertices inside cubes, they have the capability of representing sharp features in the isosurface. However, the algorithms by Gibson and Nielson position the isosurface vertices by using linear interpolation to predict the inter-

section of the isosurface and grid cube edges and averaging the predicted locations. The resulting isosurface has smoothed edges similar to the isosurface created by Marching Cubes.

The Extended Marching Cubes Algorithm by Kobbelt, Botsch, Schwannecke and Seidel [KBSS01] constructs a parametric representation of an implicit surface with sharp features from a grid of directed distances to that surface. Each grid vertex stores the directed distances in the x , y and z directions to the surface. Using linear interpolation, the algorithm by Kobbelt et al. computes a set of isosurface vertices on grid edges. It also computes surface normals at each of these isosurface vertices based on the directed distances at nearby grid vertices. Grid cubes with widely varying surface normals are identified as containing sharp features. If a grid cube does not have sharp features, an isosurface patch is retrieved from a lookup table as in Marching Cubes. If a grid cube has sharp features, then an additional isosurface vertex is added to the interior of the grid cube and connected to the isosurface vertices on the cube edges. The new isosurface vertex is positioned to minimize its least squares distance to tangent planes of the neighboring isosurface vertices. The final step applies edge flipping to connect vertices on sharp features in adjacent cubes.

Ju, Losasso, Schaefer and Warren [JLSW02, SW02] gave an alternative approach using dual contouring to construct parametric representations of implicit surfaces with sharp features. Input to their algorithm is Hermite data instead of directed distances but the difference is minimal. Hermite data contains the exact intersection points of a surface with a regular grid and the exact normals. These values are easily computed from implicit surface representations.

The algorithm by Ju et al. retrieves the intersection points of grid edges and the implicit surface from the Hermite data along with the normals at each intersection point. The normals define tangent planes at each intersection point. As in [KBSS01], the algorithm positions the isosurface vertex within a grid cube to minimize the least squares distance to tangent planes.

Varadhan, Krishnan, Kim and Manocha [VKKM03] extended the dual contouring algorithm of Ju et al. by modeling multiple intersections of an isosurface and adding more than one isosurface vertex per grid cube. Input to their algorithm is directed distances to implicit sur-

faces.

Zhang, Hong and Kaufman [ZHK04] presented a multi-resolution dual contouring algorithm which adds more than one isosurface vertex per grid cube. When input is a scalar grid, the algorithm computes the vertex locations using averaging as in [Gib98a, Gib98b, Nie04] and does not reconstruct the sharp features. When input is a directed distance field, the algorithm follows the approach in [JLSW02] and reconstructs the sharp features.

Algorithms by Ho et. al. [HWC05] and Ashida and Badler [AB03] approximate the intersection of a surface and each grid cube boundary by a polygonal curve. They connect the curve to a single isosurface vertex in the interior of the cube. Sharp features are represented by appropriate positioning of the isosurface vertex and the curve vertices.

Schaefer and Warren [SW04] gave an innovative, novel approach to constructing isosurfaces with sharp features. From the original scalar grid, they constructed a dual grid whose vertices and edges were on sharp isosurface features. They applied Marching Cubes to the dual grid to extract the isosurface.

Ju and Udesi in [JU06] show that the dual contouring algorithm may create meshes which self-intersect, i.e., mesh triangles may intersect on their interior. They present a modification of the dual contouring algorithm which fixes this problem.

Except for the original dual contouring algorithm by Gibson and Nielson's Dual Marching Cubes, all the dual contouring algorithms listed above support multiresolution isosurface extraction.

If the isovalue does not equal the scalar value of any grid vertex, the Marching Cubes algorithm creates an isosurface which is a manifold. Unfortunately, Gibson's dual contouring algorithm [Gib98a, Gib98b] often generates non-manifold isosurfaces, irrespective of the isovalue. Nielson's algorithm [Nie04] represents each Marching Cube's isosurface patch by a single dual isosurface vertex, avoiding many of the non-manifold cases. However, if a grid cube with configuration C16 or C19 (Figure 3 in [Nie04]) shares an ambiguous facet with another cube with configuration C16 or C19, the isosurface edge between the two cubes will lie in four quadrilaterals. Thus, Nielson's algorithm does not guarantee a manifold isosurface.

Schaefer, Tao and Warren [SJW07] generalize Niel-

son's algorithm to multiresolution isosurfaces. They claim that their algorithm produces non-manifold isosurfaces. However, since they rely upon Nielson's algorithm as a basis, their claim seems to be incorrect.

Greß and Klein [GK04] and Zhang and Qian [ZQ12] give dual contouring algorithms which produce multiple isosurface vertices within a cube to represent the different isosurface patches. Because these algorithms split isosurface edges and their incident vertices whenever the edges pass through ambiguous facets, these algorithms produce isosurfaces which are manifolds.

Hermite data provides both the intersection of the isosurface and each grid edge and the isosurface normal at each intersection. This defines a tangent plane at each intersection. The dual contouring algorithms compute the point which minimizes the least squares distance to a set of tangent planes and use that point as an isosurface vertex location.

Garland and Heckbert [GH97] represented the least squares distance to a set of tangent planes by a 4x4 matrix which they called the quadric error measure (QEM). The matrix size is independent of the number of tangent planes. Lindstrom [Lin00] used the quadric error measure to compute the point which minimizes the least squares distance to the represented set of tangent planes. When the tangent planes define an edge or a smooth portion of the isosurface, Lindstrom's algorithm selects the point closest to the cube center. The feature sensitive dual contouring algorithms all use some variation of the quadric error measure to compute isosurface vertex locations.

There are numerous papers on generating and meshing surfaces other than isosurfaces when the surfaces have sharp features. We mention only the ones most relevant to this paper.

Cheng, Dey and Ramos [CDR07] described an algorithm to mesh piecewise smooth complexes using weighted Delaunay triangulations. They choose points along the boundaries of each smooth piece and construct protecting balls around each such point by assigning a weight to each point. They then choose sample points from the smooth portion of the surface outside of the protecting balls and returned the weighted Delaunay triangulation of the points.

Salman et al. [SYM10] and Dey et al. [DGQ*12] use the protecting balls from [CDR07] to reconstruct surfaces with sharp features from point cloud data. Both papers

identify sharp features and protect them with balls. They then reconstruct the surface using the protecting balls.

4 Computing Vertex Locations

Lindstrom [Lin00] described how to use the quadric error measure from [GH97] to find the point closest to a set of planes. The algorithms for computing isosurfaces with sharp features all rely upon the quadric error measure to position isosurfaces on sharp edges and corners.

As noted in the previous section, hermite data determines tangent planes to the isosurface, one at each intersection of the grid edge and the isosurface. For a grid cube \mathbf{c} , the k tangent planes on its edges give a set of k equations

$$Mx = b$$

where M is a $k \times 3$ matrix and x and b are column vectors of length k . In general, this system is over-determined so we wish to find the least squares solution. The least squares solution is the solution to

$$M^T Mx = M^T b.$$

The 3×3 matrix $A = M^T M$ and the column vector $b' = M^T b$ gives the quadric error measure.

The singular valued decomposition (SVD) of A is $A = U\Sigma V$ where

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{pmatrix}.$$

σ_1, σ_2 , and σ_3 are the singular values of A . If all three singular values of A are large, then \mathbf{c} contains a sharp corner. If two singular values are large, then \mathbf{c} contains a sharp edge. Otherwise, cube \mathbf{c} does not contain a sharp feature.

Let

$$\sigma'_i = \begin{cases} \sigma_i & \text{if } \sigma_i/\sigma^* > \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

where σ^* is the largest singular value and ε is a threshold parameter. Let $A' = U\Sigma'V^T$ where Σ' is the diagonal matrix with diagonal entries $(\sigma'_1, \sigma'_2, \sigma'_3)$.

When A has three large singular values, $A' = A$ and there is a single point x such that $A'x = b'$. When A has two large singular values, $\{x : A'x = b'\}$ is a line. When A has one large singular value, $\{x : A'x = b'\}$ is a plane.

Let

$$\sigma_i^+ = \begin{cases} 1/\sigma'_i & \text{if } \sigma'_i \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Let Σ^+ be the diagonal matrix with diagonal entries $(\sigma_1^+, \sigma_2^+, \sigma_3^+)$. As in [Lin00], compute:

$$x = q_{\mathbf{c}} + V\Sigma^+U^T(b' - Aq_{\mathbf{c}}). \quad (1)$$

When A has three large singular values, x is the point solving $Ax = b$. When A has two large singular values, x is the point closest to $q_{\mathbf{c}}$ on the line $A'x = b$. When A has only one large singular value, x is the point closest to $q_{\mathbf{c}}$ on the plane $A'x = b$. Lindstrom uses the center of grid cube \mathbf{c} as the point $q_{\mathbf{c}}$.

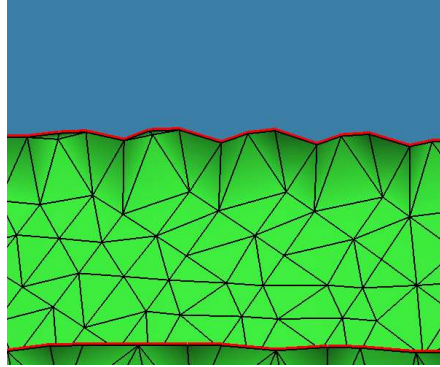
As described above, matrices A and b' are determined by tangent planes. However, they can also be determined using gradients. Let g_i and s_i be the gradient and scalar value, respectively, at point p_i . Let σ be the isovalue. The set $\{x : g_i \cdot (x - p_i) + s_i = \sigma\}$ is a plane in 3D. Equivalently, this plane is $\{x : g_i \cdot x = \sigma - (g_i \cdot p_i + s_i)\}$. Let g_i be the rows of M be $g_i/|g_i|$ and the elements of b be $(\sigma - (g_i \cdot p_i + s_i))/|g_i|$. (We divide by $|g_i|$ so that all normal directions have equal weight.) Compute $A = M^T M$ and $b' = M^T b$ and solve as above. This formulation allows us to compute sharp isosurface vertex locations directly from gradients, without first transforming the gradients to hermite data.

Instead of setting $q_{\mathbf{c}}$ in Equation 1 to be the center of cube \mathbf{c} , Schaefer and Warren [SW02] propose setting $q_{\mathbf{c}}$ to the centroid of the intersections of the edges of \mathbf{c} and the isosurface. For reasons discussed in the next section, this choice of $q_{\mathbf{c}}$ improves the reconstruction results.

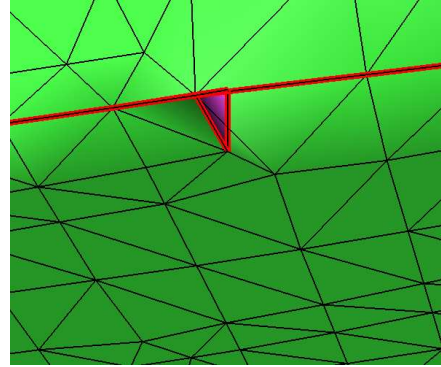
Hermite data gives the locations of the intersections of the edges \mathbf{c} and the isosurface. If the input is a gradient grid, then these intersections must be computed from the input. A simple, but inaccurate approach, is to use linear interpolation to compute these intersection points. A more accurate approach is to use the gradients at the edge endpoints to determine the intersection point.

5 Problems with Vertex Locations

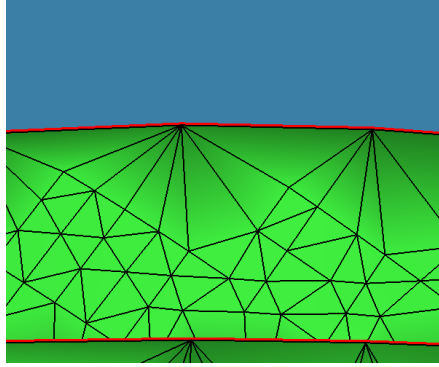
The Extended Marching Cubes algorithm by Kobbelt et. al. [KBSS01] and the dual contouring algorithm by Ju et. al. [JLSW02, SW02] compute an isosurface vertex for



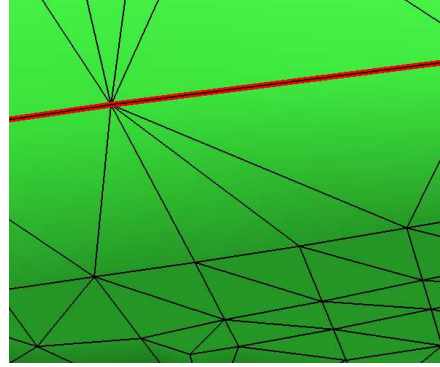
(a) Dual contouring with clamping. Notches along the sharp edge (red) of the isosurface.



(b) Dual contouring with no clamping. Purple triangle is a self intersection in the isosurface.



(c) MergeSharp of the region in 2a shows no notches.



(d) MergeSharp of the region in 2b shows no intersections.

Figure 2: Isosurface errors produced by dual contouring and corresponding MergeSharp results. Mesh edges with large dihedral angle are colored red.

each grid cube using the quadric error measure (QEM). When the surface is relatively smooth, the isosurface vertex lies within the grid cube. However, if the surface has a sharp feature, the vertex location computed using QEM may lie outside the grid cube. Should the isosurface vertex be placed at the location outside the grid cube?

The problem of isosurface vertex locations lying outside the grid cube was noted by Schaefer and Warren in [SW02]. One reason an algorithm may compute a location outside a grid cube is that it chooses the wrong location on a sharp edge. Schaefer and Warren used the centroid of the intersections of the cube edge and the isosurface for the point p in Equation 1 to make it more likely

that a point on the intersection of the sharp edge and the grid cube is chosen. However, what happens if the sharp edge or sharp corner does not intersect the grid cube? In that case, Schaefer and Warren’s algorithm will still return a location outside the grid cube.

Consider a grid cube \mathbf{c} which generates a vertex location p on a sharp edge or corner which does not intersect \mathbf{c} . Let $\mathbf{c}' \neq \mathbf{c}$ be the grid cube containing p . If cube \mathbf{c}' has a bipolar edge, then it also generates an isosurface vertex v' . Placing v in \mathbf{c}' may create degenerate mesh triangles or it may create overlapping triangles as the mesh folds back on itself. (See Figure 11.b.) On the other hand, if \mathbf{c}' does not have any bipolar edge, then it does not gener-

ate an isosurface vertex. Clamping v to lie inside \mathbf{c} will create ridges in the isosurface or cut off the corner. (See Figure 11.a.)

One plausible approach might be to clamp p to cube \mathbf{c} only if \mathbf{c}' contains a bipolar edge. As shown in Section 11, this approach also produces numerous errors.

For cubes whose facets are parallel to the grid facets, grid cubes which intersect the surface corners or edges will almost always have bipolar edges. Thus clamping v will not create ridges along the sharp edges and will not cut off corners. The same holds for cylinders or annuli whose edges whose central axes are parallel to grid edges. On the other hand, rotating these surfaces by any significant angle with respect to the grid creates many sharp corners or edges which intersect cubes with no bipolar edges.

6 Merging Grid Cubes

To address the problems discussed in the previous section, we use a technique similar to the protecting balls from [CDR07]. We identify the grid cubes whose isosurface vertices lie on sharp edges or corners, and select a subset such that no two selected cubes are vertex neighbors. (A grid cube \mathbf{c} is a *vertex neighbor* of grid cube \mathbf{c}' if \mathbf{c} shares a vertex with \mathbf{c}' .) We merge each selected cube \mathbf{c} with its vertex neighbors. We generate a single vertex for the merged region.

Let C_{corner} be the grid cubes whose isosurface vertices lie on sharp corners. Let C_{edge} be the grid cubes whose isosurface vertices lie on sharp edges. We start by selecting cubes from C_{corner} .

To select the cubes from C_{corner} , we compute the vertex location $p_{\mathbf{c}}$ using Equation 1. We use the centroid of the intersections of the edges of \mathbf{c} and the isosurface as point $q_{\mathbf{c}}$. We process the cubes of C_{corner} in increasing order of the L_{∞} distance between $p_{\mathbf{c}}$ and $q_{\mathbf{c}}$. This causes cubes which are “closer” to the corners to be processed first.

As we select cubes, we merge their vertex neighbors with them. A cube which has been merged with a selected cube is labelled “covered”. All cubes are initially uncovered. We select only uncovered cubes. When we select cube \mathbf{c} , we merge the uncovered vertex neighbors of \mathbf{c} with \mathbf{c} . Those vertex neighbors become covered. Thus no two selected cubes can be vertex neighbors. If the L_{∞}

```

MERGE(Grid,  $C_{\text{selected}}$ )
1 foreach cube  $\mathbf{c}$  do
2   | Compute the centroid  $q_{\mathbf{c}}$  of the intersections of
   | the edges of  $\mathbf{c}$  and the isosurface;
3   | Compute vertex location  $p_{\mathbf{c}}$  using Equation 1;
4 end
5  $C_{\text{corner}} \leftarrow$  cubes  $\mathbf{c}$  where  $p_{\mathbf{c}}$  is on a sharp corner;
6  $C_{\text{edge}} \leftarrow$  cubes  $\mathbf{c}$  where  $p_{\mathbf{c}}$  is on a sharp edge;
7 Sort  $C_{\text{corner}}$  and  $C_{\text{edge}}$  in increasing order by
  |  $|p_{\mathbf{c}} - q_{\mathbf{c}}|_{\infty}$  (the Linf distance);
8 Mark all cubes as Uncovered;
9 MERGECUBES( $C_{\text{corner}}, C_{\text{selected}}$ );
10 MERGECUBES( $C_{\text{edge}}, C_{\text{selected}}$ );
11 foreach cube  $\mathbf{c} \in C_{\text{corner}} \cup C_{\text{edge}}$  do
12   | if ( $\mathbf{c}$  is not selected) and ( $\mathbf{c}$  is not Covered) then
   |   |  $p_{\mathbf{c}} \leftarrow q_{\mathbf{c}}$ ;
13 end

```

Algorithm 1: Algorithm MERGE.

```

MERGECUBES( $C, C_{\text{selected}}$ )
1 foreach cube  $\mathbf{c} \in C$  do
2   | if (cube  $\mathbf{c}$  is Uncovered) and ( $|p_{\mathbf{c}} - q_{\mathbf{c}}|_{\infty} \leq \gamma$ )
   | then
3     | if (NoLARGEANGLETRI( $\mathbf{c}, C_{\text{selected}}$ )) then
4       | Add  $\mathbf{c}$  to  $C_{\text{selected}}$ ;
5       | foreach vertex neighbor  $\mathbf{c}'$  of  $\mathbf{c}$  do
6         | if cube  $\mathbf{c}'$  is Uncovered then
7           |   |  $\mathbf{c}'.\text{MergeWith} \leftarrow \mathbf{c}$ ;
8           |   | Mark  $\mathbf{c}$  as Covered;
9           | end
10        | end
11      | end
12    | end
13 end

```

Algorithm 2: Algorithm MERGECUBES.

distance between $p_{\mathbf{c}}$ and $q_{\mathbf{c}}$ is greater than some threshold, we do not select \mathbf{c} .

The procedure, as just described, may still create many degenerate or near-degenerate triangles. Cubes near a sharp edge could be close enough together to form a triangle yet far enough apart so that no cube region covers

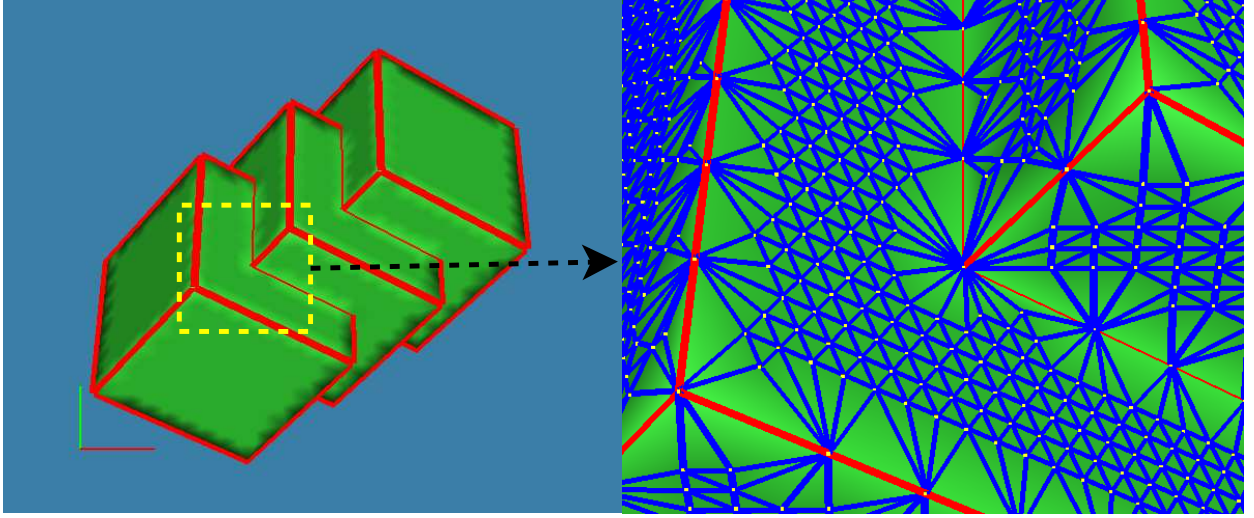


Figure 3: Cube stack closeup

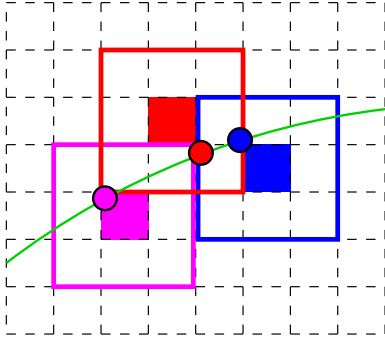


Figure 4: 2D illustration of cube stack (blue,magenta, red) which are close enough together to form a triangle yet far enough apart so that no $3 \times 3 \times 3$ cube region covers the other two cubes. Sharp edge is represented by the green curve. Each cube generates an vertex location (with the same color as the cube) on the sharp edge. The triangle formed by the three vertices is almost degenerate.

the other two cubes. (See Figure 4.) When the vertices generated by the cubes are mapped to the sharp edge, the angle at the middle vertex becomes 180° or near 180° . To avoid such problem triangles, we do not select a cube which forms a large angle triangle with already selected

```

NOLARGEANGLETRI(c,  $C_{\text{selected}}$ )
1 for each pair  $\mathbf{c}', \mathbf{c}'' \in C_{\text{selected}}$  near c do
2    $R' \leftarrow \text{Reg3x3x3}(\mathbf{c}) \cap \text{Reg3x3x3}(\mathbf{c}')$ ;
3    $R'' \leftarrow \text{Reg3x3x3}(\mathbf{c}) \cap \text{Reg3x3x3}(\mathbf{c}'')$ ;
4   if (some facet of  $R'$  has a bipolar edge and some
5     facet of  $R''$  has a bipolar edge) then
6      $\alpha \leftarrow \text{max angle of triangle } \Delta(p_{\mathbf{c}}, p_{\mathbf{c}'}, p_{\mathbf{c}''})$ ;
7     if ( $\alpha \geq 140^\circ$ ) then return (false);
8   end
9 end
10 return (true);

```

Algorithm 3: Algorithm **NO**LARGEANGLETRI.

cubes. (See Algorithm **NO**LARGEANGLETRI.)

Checking each pair of cubes in C_{selected} can be a time consuming operation. We use a simple nearest neighbor data structure based on a low resolution subgrid to quickly identify points in C_{selected} which are near **c**.

7 Isosurface Construction

The algorithm **MERGE** constructs regions with irregular shapes around selected vertices. Instead of extracting the

dual isosurface from those regions, we extract the dual isosurface from the full grid and then merge isosurface vertices which are from cubes in the same region. This is simpler than trying to extract the dual isosurface from the regions.

Our algorithm for isosurface construction has four steps. First, compute an isosurface vertex location for each grid cube with a bipolar edge. Next, select a set of isosurface vertices on sharp corners and edges and their surrounding regions. Apply Nielson’s Dual Marching Cubes algorithm [Nie04]. to construct the dual contouring isosurface from the full grid. Finally, for each merged region, merge isosurface vertices generated by the cubes in that region. Merging isosurface vertices transforms isosurface quadrilaterals to triangles and creates some degenerate quadrilaterals which are removed.

Nielson’s Dual Marching Cubes algorithm avoids most (but not all) of the non-manifold configurations created by the original dual contouring algorithm. However, the merging of isosurface vertices in a region can create such non-manifold configurations.

8 Computing Hermite Data

Our data set consists of the scalars and the gradients at the grid vertices. To compare our algorithm to algorithms using hermite data, we will compute the surface normals at the intersection of the grid edges and the isosurface. One way to do this is to use linear interpolation on the endpoints of the bipolar grid edges. As illustrated in Figure 5, this approach can give a very wrong result.

The gradient at the point of intersection of the cube edge and the isosurface, is not a combination of the gradients of the endpoints. Rather it is a copy of the gradient at one of the edge endpoints.

A much more accurate calculation, is achieved by utilizing the gradients at the grid edge endpoints. As shown in Figure 5, a grid vertex v_i at point p_i with scalar value s_i and gradient g_i determines a scalar field $f_i(x) = (x - p_i) \cdot g_i + s_i$. The level set $f_i^{-1}(\sigma)$ is a plane in \mathbb{R}^3 which approximates $f^{-1}(\sigma)$ near p_i .

The combination of the scalar fields, along the cube-edge is a piece-wise linear curve as shown in Figure 5. The intersection of the scalar representing the isovalue σ and the curve produces the exact point of intersection and

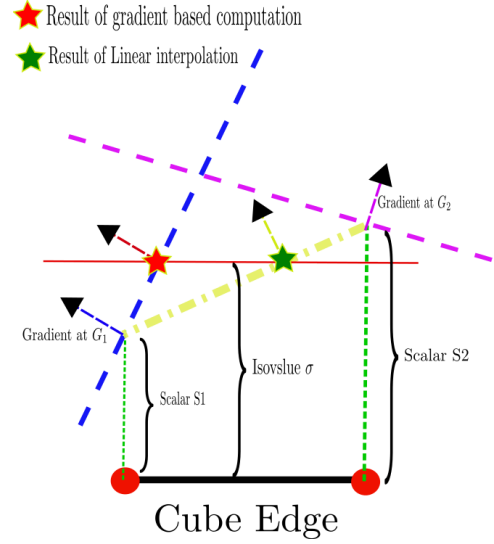


Figure 5: Linear interpolation versus gradient based calculation.

the gradient at this point is the gradient of the determining edge endpoint.

9 Gradient Selection

We compare the effect of using different grid vertex gradients in the computation of the sharp-vertex in dual methods as mentioned in Section 4. This is in contrast to the dual contouring Ju et. al. [JLSW02] which uses the exact intersection and the surface normals. For a cube that is intersected by the isosurface, the set of gradients to be used to determine the isosurface vertex can be determined by using one of the following options.

- **Gradient Cube (gradC):** The gradients of the vertices of the intersected cube.
- **Gradient Cube Determined (gradCD):** A subset of the gradients of endpoints of bipolar cube edges. For each bipolar edge, we choose the endpoint and gradient which determines the intersection of the edge and the isosurface (see Section 8 and Figure 5).
- **Gradient Neighbor (gradN):** The gradients of the vertices of the intersected cube and vertices adjacent

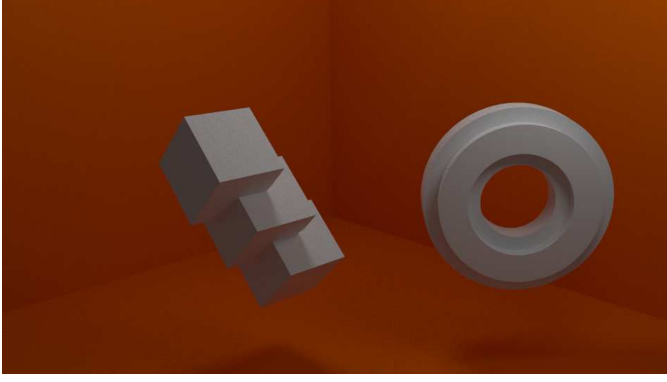


Figure 6: Base elements of our dataset, the cube stack and the flange.

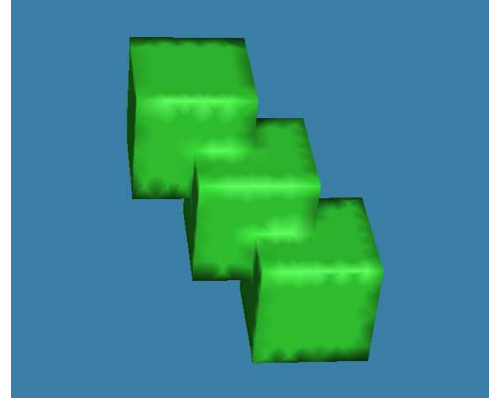
to the cube vertices.

- **Gradient Neighbor Selective (gradNS):** A subset of the gradients of the vertices in the intersected cube and vertices adjacent to the cube vertices. Unlike gradN, a gradient is selected only if the iso-plane generated by the scalar value, gradient at the vertex and the supplied isovalue intersects the edge.

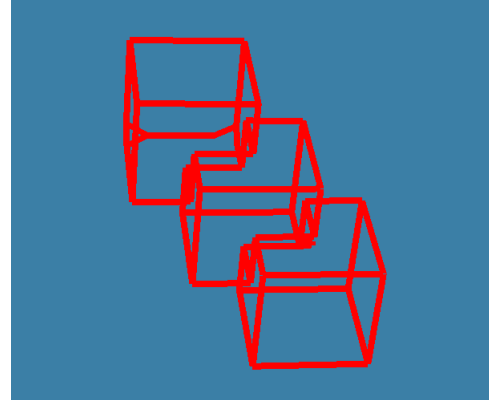
10 Measuring Correctness of Sharp Features

A procedure called FindSharp is used to compute and visualize sharp edges. FindSharp computes the dihedral angle between adjacent surface polygons. It reports any isosurface edges whose dihedral angle is greater than a threshold (140°) or which are incident on three or more isosurface polygons. We visualize such edges by drawing them in red. We can visualize them either with the mesh (Figure 7c) or without the underlying isosurface (Figure 7b). Figure 7a shows the triangle mesh itself.

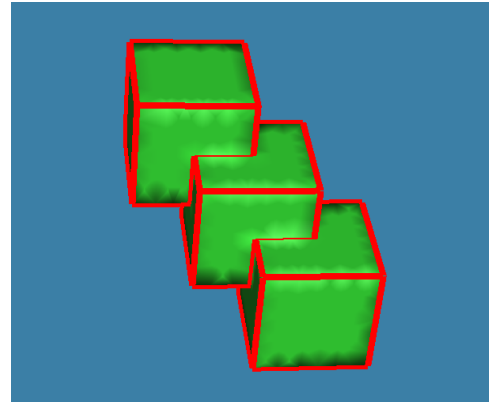
The set of isosurface edges reported by FindSharp is a graph embedded in \mathbb{R}^3 . Procedure CountDegree measures the correctness of the sharp feature reconstruction by counting the number of vertices of each degree in this graph. The absolute difference between this and the expected number gives the number of errors for that degree. The total error is the sum of those errors.



a) Triangle mesh of the cube stack data set.



b) FindSharp result showing the feature lines.



c) The mesh along with the features.

Figure 7: Comparing different views of the cube stack dataset.

For example, the flange isosurface shown in Figure 8b has no corners, so the graph of its sharp edges should have no vertices of degree other than two. Any vertices of any other degree are errors. The isosurface of cube-stack has 32 corners and saddles, as shown in Figure 7b, 8a. The graph of its sharp edges should have exactly 32 degree 3 vertices. Any vertices of degree 1 or degree more than three are errors.

11 Experimental Results

11.1 Benchmark Datasets and Evaluation interpretation

We developed two sets of data to test our algorithm. The cube stack datasets sample a scalar field $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ where $f(p)$ is the minimum of the L_∞ distance to three points, q_1, q_2 and q_3 . Isosurfaces in the datasets are three (overlapping) cubes, whose centers are q_1, q_2 and q_3 (Figure 8a). To tilt the cubes, we used orthogonal frames other than the standard one given by the x, y, z axes, and computed the L_∞ metric in those other frames. The cube stack datasets test the performance of our algorithm on sharp corners or saddles

The flange datasets sample a scalar field $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ where $f(p)$ is the combination of the distance $d_C(p)$ to a cylinder and the distance $d_P(p)$ to a plane orthogonal to the cylinder axis. The function f is defined as:

$$f(p) = \max(\min(d_C(p), d_P(p)), \max(d_C(p), d_P(p))/2).$$

Isosurfaces in these datasets are flanges with sharp concave and convex edges. (Figure 8b). The flange datasets test the performance of our algorithm on sharp edges.

We embedded each scalar field in a 100x100x100 grid at various angles to the grid axes. The grids contain both scalar values AND the gradients of the underlying scalar field at each grid vertex. There are 7 flange (Table 1) and 18 cube-stack data sets, which we test with five different isovalues. Thus, there are 35 test cases for flange and 90 test cases for cube-stack. Figure 6 shows renderings of our data set. Table 1 provides the details for the flange data sets. In Figure 8b we show three of the flange datasets as mentioned in Table 1, the different orientations help us to properly test how our and other algorithms per-

Data Set Index	Center			Direction		
1	40.1	30	57.8	1	0	0
2	40.1	30	57.8	1	1	0
3	40.1	30	57.8	1	1	1
4	40.1	30	57.8	2	2	5
5	40.1	30	57.8	2	5	7
6	40.1	30	57.8	1	1	4
7	40.1	30	57.8	1	1	7

Table 1: Flange Data set information. Note the data sets are at an offset from the grid vertices. The direction determines the vector pointing to the major axis.

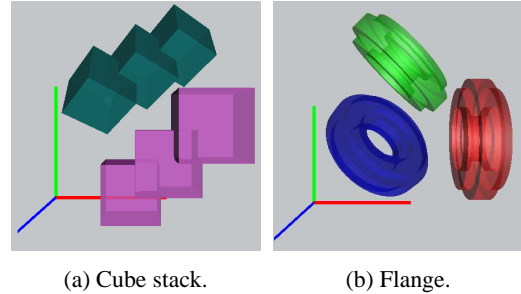


Figure 8: Elements from our evaluation datasets. The cube stacks and flanges are rotated at various angles to test the robustness of the algorithms.

form. Figure 3 shows the result of MergeSharp on one of the cube stack datasets.

We applied FindSharp and CountDegree to the isosurfaces produced from each data set and measured the total degree errors as described in the previous section. Table 2 gives an example of some data sets, the isovalue used, the direction in which the data set is tilted, the errors for different degree and the total errors in a tabular form. Isosurface which have poor representation of sharp features produce numerous graph vertices with degree one, three or higher.

We also display the number of errors in histograms. For instance, Figure 12 shows the comparison of MergeSharp with Dual Contouring with different options. The X axis represents the test case identifier and the Y axis represents the number of errors. We show results both for flange and cube-stack data sets.

We compared our algorithm to the openly available pro-

Dataset	Direction	Isovalue	Degree			Degree 3 Errors	Total Errors
			1	3	≥ 4		
Flange	(1,1,4)	3.23	11	152	98	152	261
Flange	(1,1,1)	3.5	0	2	0	2	2
Cube stack	(1,1,0)	4.8	0	32	0	0	0
Cube stack	(1,1,1)	3.23	3	32	3	0	6
Cube stack	(1,1,1)	4.6	0	27	6	5	5

Table 2: Degrees in graph of sharp edges for various datasets and isovalues. All datasets are regular scalar grids of dimensions $100 \times 100 \times 100$.

gram Polymender [Ju04]. Polymender can be used to construct hermite data for a scalar field representing distance to a polygonal mesh. More specifically, Polymender outputs the coordinates in dcf format of the intersection of grid edges and the isosurface. It also outputs the isosurface normals at those intersection points. We compared Polymender with MergeSharp on four triangle mesh data sets downloaded from grabcad.com/library/software/stl-for-3d.

Figure 9, 10, 15, 16, 17 compares isosurfaces produced by Polymender and MergeSharp. Table 3 compares Polymender and MergeSharp in terms of the errors produced. For all test cases, we set the Polymender octree depth to 8 and the Polymender scale parameter to 0.8.

11.2 Evaluation

The flange and cube-stack data sets contain scalar values and gradients at grid vertices. To compare with dual contouring algorithms [JLSW02], we generate a hermite equivalent of the grid data by computing the edge intersection and the normals at the intersection using the procedure described in Section 8.

We first compared MergeSharp with our own implementation of dual contouring with no cube merging. We implemented three variations of the dual contouring algorithm. The first variation, called clampGrid, clamped isosurface vertices to their generating grid cube. The second variation, called noClamp, allowed isosurface vertices to be located outside their generating grid cube. The third variation, called clampConflict, clamped isosurface vertices to their generating grid cube c only if they were situated in a neighboring grid cube c' which had a bipolar edge. The histograms are shown in Figure 12.

As illustrated in Figure 12, clampGrid generates more errors than noClamp and MergeSharp. clampConflict performs better in some tests than clampGrid. An example of a typical clamping error is shown in Figure 2a. The noClamp algorithm, has fewer errors than clampGrid, but it also produces problems such as flipped-triangles as shown in Figure 2b. MergeSharp has the fewest number of errors and performs better than the other options. The corresponding regions shown in Figure 2c and Figure 2d illustrate how MergeSharp performs in the cases where other options fail to produce the correct output (Figure 2a, Figure 2b).

From Figure 2a we observe, clamping generates ‘notches’ in the output. This happens when the vertex generated by methods discussed in Section 4 are clamped to default locations (such as centroid of edge-intersections) or clamped to the grid cube. While noClamp solves the problem of notches, without a proper ordering of the vertices being placed outside of the corresponding cube, in many cases we see flipping of triangles (Figure 2b).

Next we compare computing the isosurface vertex from grid vertex gradients to computing the isosurface vertex from the surface normals and edge intersections. For this we use the gradients mentioned in Section 9 namely gradCD, gradC, gradN, gradNS with hermite equivalent (computed as mentioned in Section 8).

The resulting histograms are shown in Figure 13. We note some observations from the graph. GradN on the flange dataset produces a considerable number of errors. Looking at the cube gradients and adjacent neighbors without some selection criteria, means smooth cubes near sharp features (edges/corners) tend to place the isoverices near these sharp features, which leads to deterioration

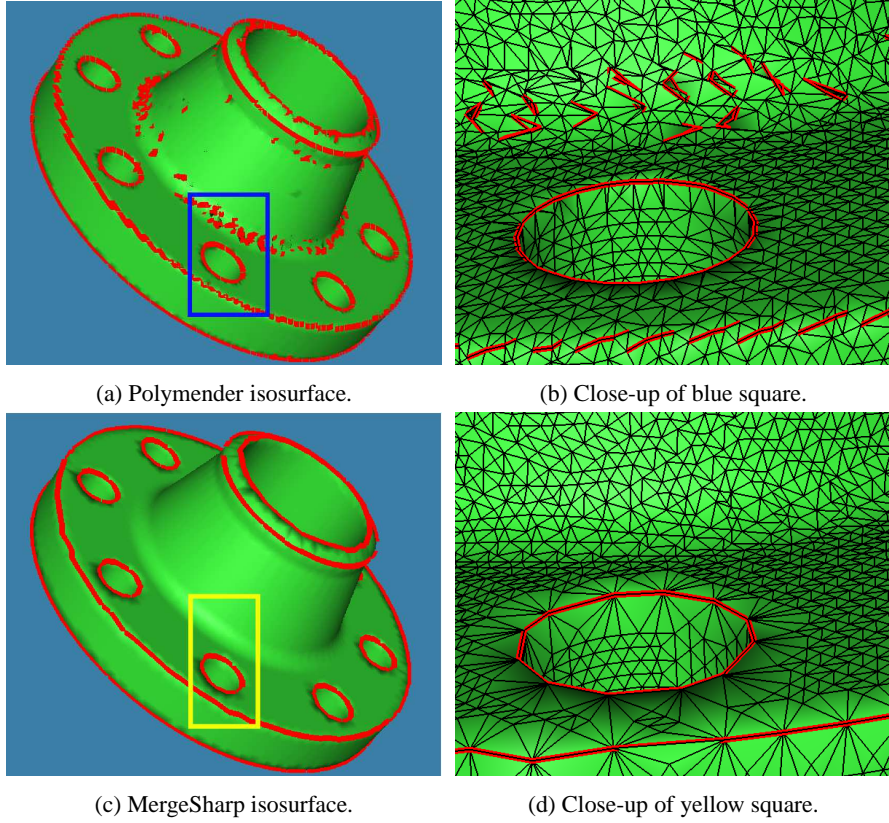


Figure 9: Isosurfaces from the weld dataset with ‘sharp ’ mesh edges (large dihedral angles) marked in red. 9a), 9b) Polymender generates disconnected sharp mesh edges representing a sharp edge of the object. It also generates mesh edges with large dihedral angle in smooth regions of the surface. 9c), 9d) MergeSharp correctly generates a connected curve of sharp mesh edges to represent a sharp edge of the object. It does not generate mesh edges with large dihedral angles in the smooth region of the surface.

of the mesh and a higher number of errors.

In Figure 14 we focus on hermite, gradCD, gradC and gradNS. Blank columns (for example flange test case 5) are cases with zeros errors. In Table 6 we see test case 5 is oriented along the X-axis (Table 1). We conclude that axis aligned data sets are less effected by the conflict and flipping errors. As we tackle data sets which are not axis aligned, such as flange test-id 25 [Table 6], whose axis is tilted towards the coordinate (2,5,7) [Table 1] the errors increase.

We further observe in Figure 14a, GradCD produces equivalent results to hermite in most cases. The gradC im-

plementation generates more error in certain cases. Both gradC like gradN might select points on nearby sharp features (edges/corners) even if the cube is in a smooth region. In contrast, GradNS produces similar results to hermite with few errors.

In Figure 14b, we note that in certain cases for cube stacks, gradN performs better than gradC. We infer that in these cases, looking at just the cube gradients is not enough. Gradients from neighboring locations play a crucial role in determining the vertex location and so gradN performs better.

Comparison of gradNS and hermite leads us to con-

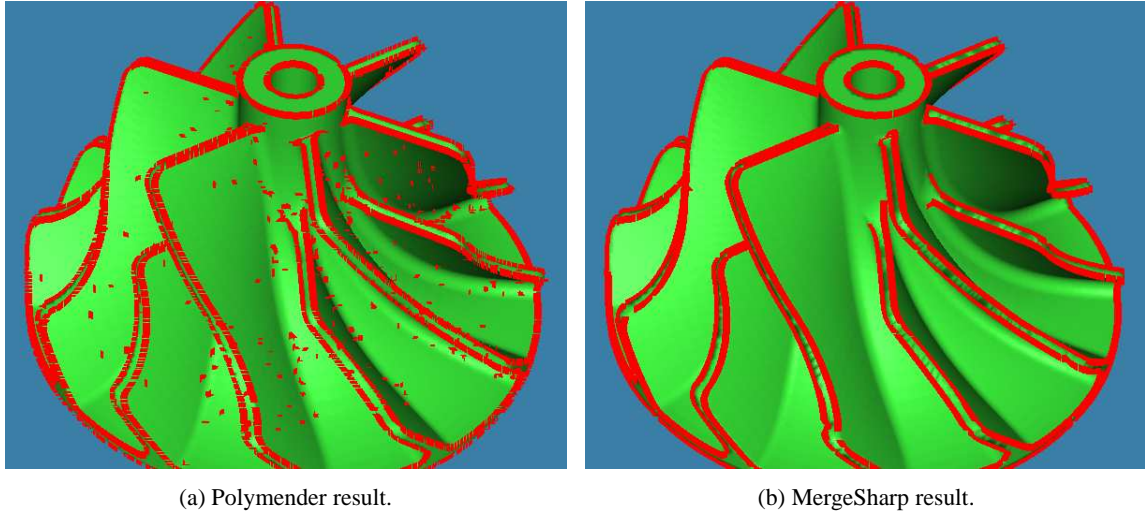


Figure 10: Overview of Polymender and MergeSharp results. FindSharp output is shown in red.

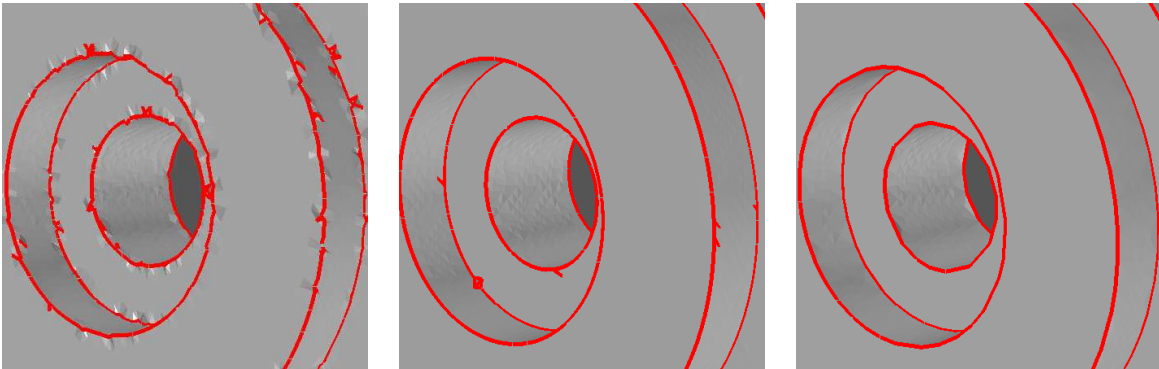
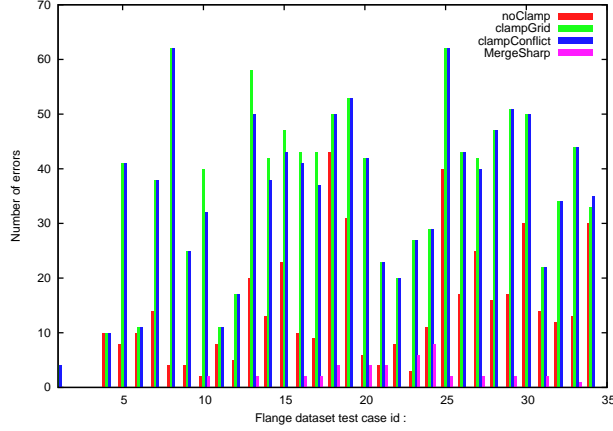
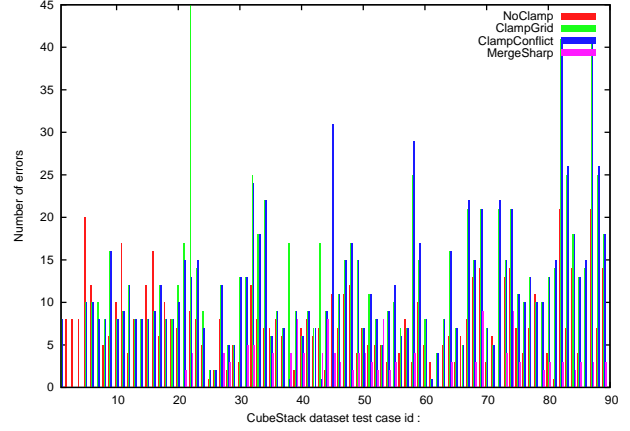


Figure 11: Flange isosurfaces. Sharp isosurface edges are marked in red. a) Dual contouring with clamping to grid cubes producing ridges in the sharp edges. b) Dual contouring with no clamping to grid cubes. Degenerate triangles and folds appear as red edges sticking out from the red circles around the flange. c) Dual contouring based on cube merging.

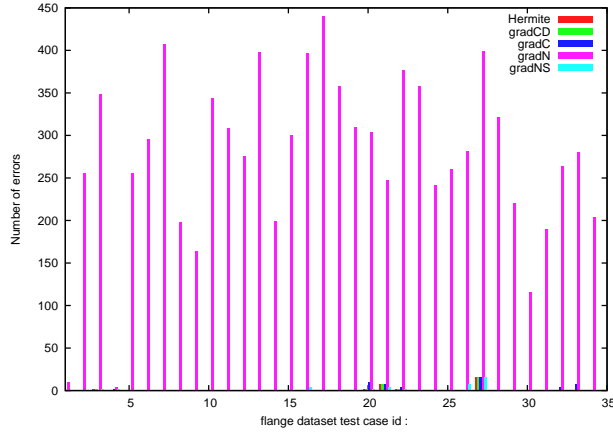


(a) Flange dataset.

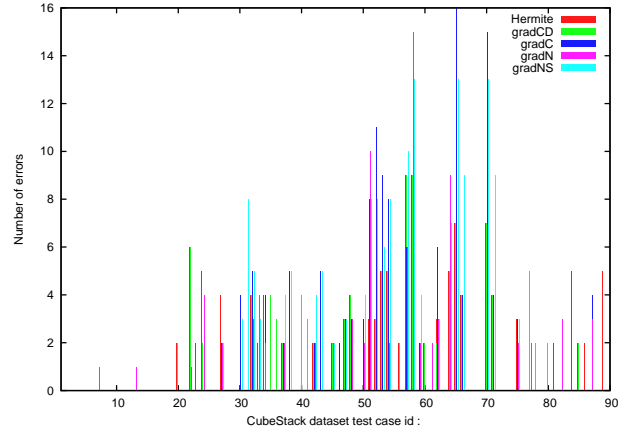


(b) Cube stack dataset.

Figure 12: Comparison of MergeSharp and dual contouring. In ClampGrid, isosurface vertex positions are clamped to their generating cube. In NoClamp, isosurface vertex positions are not clamped to their generating cube. In ClampConflict, isosurface vertex positions are clamped to their generating cube if they initially lie in some cube c' which has a bipolar edge.



(a) Flange dataset.



(b) Cube stack dataset.

Figure 13: Comparing MergeSharp on hermite data computed from gradients (gradEC) and using gradients directly (gradCD, gradC, gradN and gradNS).

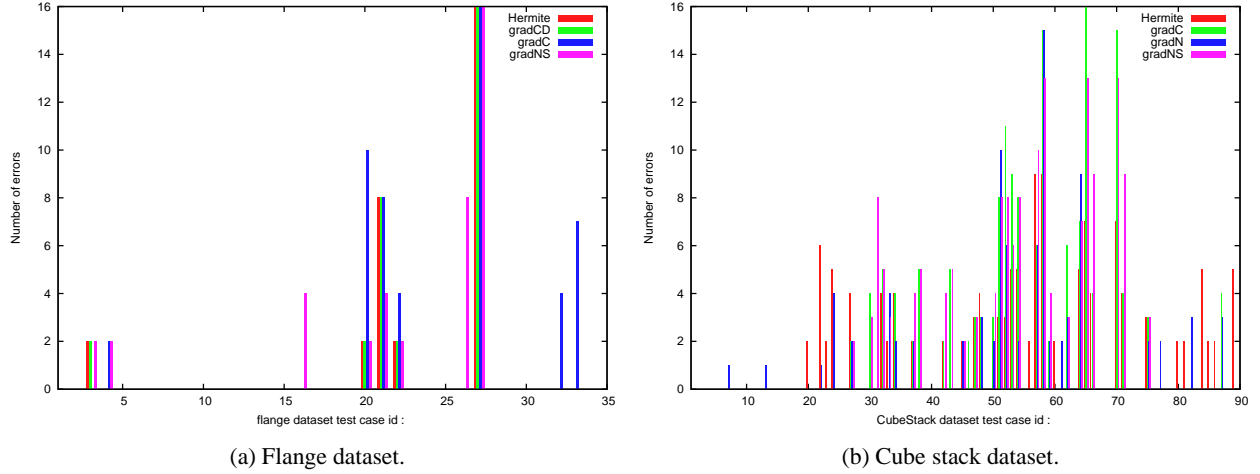


Figure 14: A more detailed comparison of MergeSharp on hermite data computed from gradients (gradEC) and using gradients directly. (Empty columns refer to cases where no errors were produced.)

clude that gradients at grid vertices can be used instead of surface normals to compute isosurface vertices.

11.3 Polymender and MergeSharp Comparison

We also show comparisons of results from MergeSharp and Polymender. Figure 10, 9, 17 15 shows overviews, closeup views of errors produced by polymender and the corresponding regions for MergeSharp.

In Table 3 we provide detailed results of error generated on the data sets. We can observe that MergeSharp performs better than Polymender in both CountDegree numbers and visual comparisons.

As in [SW02] we truncate the singular values based on the relative magnitude to the largest singular value, as discussed in Section 4. For all previous tests, we have used 0.1 as the threshold. In Figure 18 we show the results of running MergeSharp with other threshold values. We observe that setting a very small threshold such as 0.05 leads to more errors. Increasing the threshold to 0.2 does not create a noticeable change in the number of errors in flange but does create some problem cases with more errors in cube stacks. When the cutoff is small, the increased number of large singular values leads to false identification of features. Edges are classified as corners

or smooth regions are classified as sharp edges. A high threshold on the other hand misses some corners.

Lindstrom [Lin00], finds the vertex p_c closest to the cell center that is also closest to the surface tangent planes. In their dual contouring paper Ju et. al. [JLSW02] use the ‘mass point’ instead of the cube center. We show results for computing the centroid, by computing the edge intersection in two different ways. First, we use a gradient based computation to find the edge intersections. Next, we use simple interpolation along the edge to find the intersection. In Figure 19 we compare these two methods along with using the cube-center as the ‘mass point’. We observe that using the cube-center produces the most number of errors. These occur due to the creation of notches as shown in Figure 2a. Distance to centroid from the gradient computation gives competitive results in all test cases and is used as default for our tests.

Finally, we claim that our algorithm was robust to noise. We test this by perturbing the gradients of our data sets uniformly by 10 and 20 degrees.

Figure 20 contains results comparing MergeSharp on perfect data and on data with noisy gradient directions. We observe from the histogram that when gradients are uniformly perturbed upto 10 degrees, MergeSharp is very competitive to no noise results.

In Figure 21 we compare the results from running

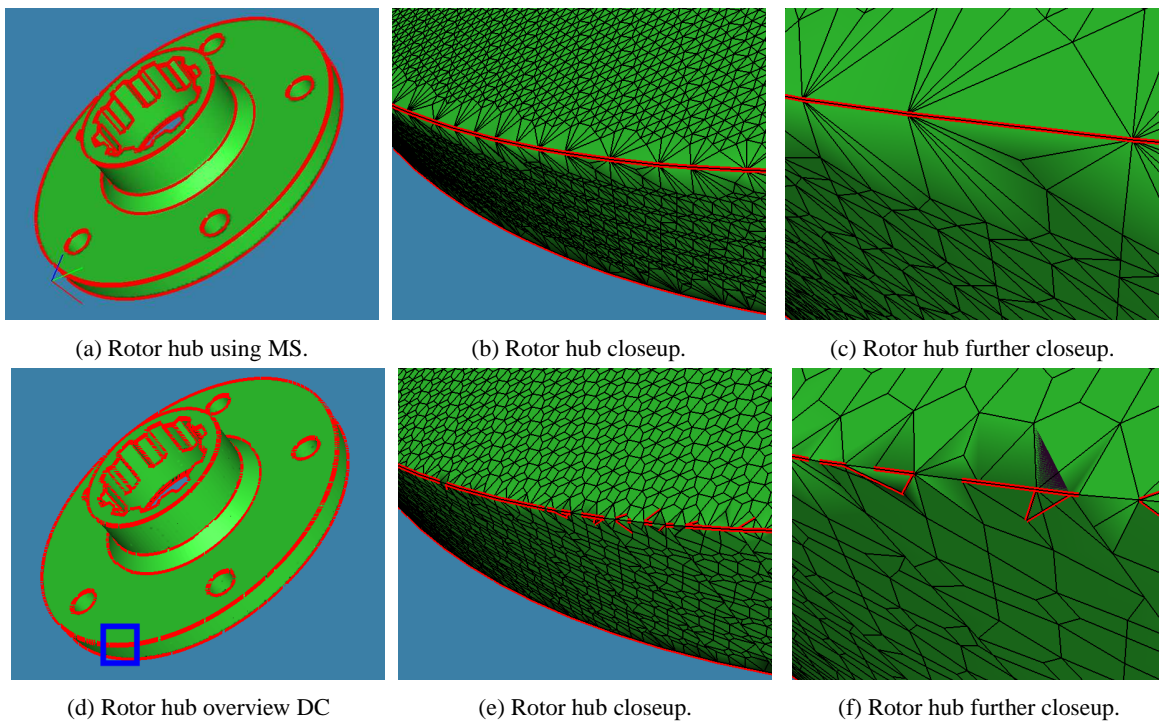
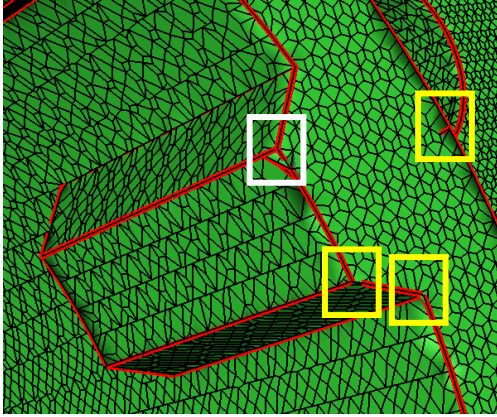
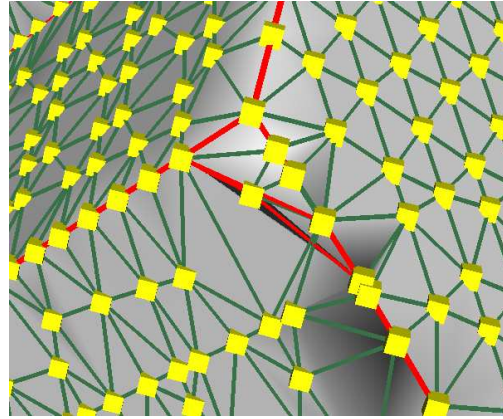


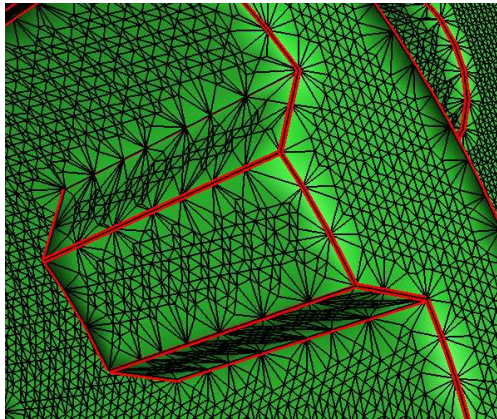
Figure 15: Comparison of Polymender and MergeSharp on the rotor hub dataset.



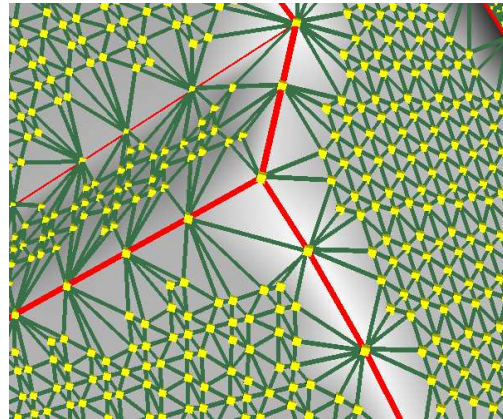
(a) Polymender result on the rotor dataset. Yellow and white rectangles indicate some problematic regions.



(b) Closeup of the white rectangle above shows the problems with the Polymender near the corner.



(c) MergeSharp result on the rotor dataset.



(d) MergeSharp reconstruct the corner accurately.

Figure 16: Close up of Polymender and MergeSharp results.

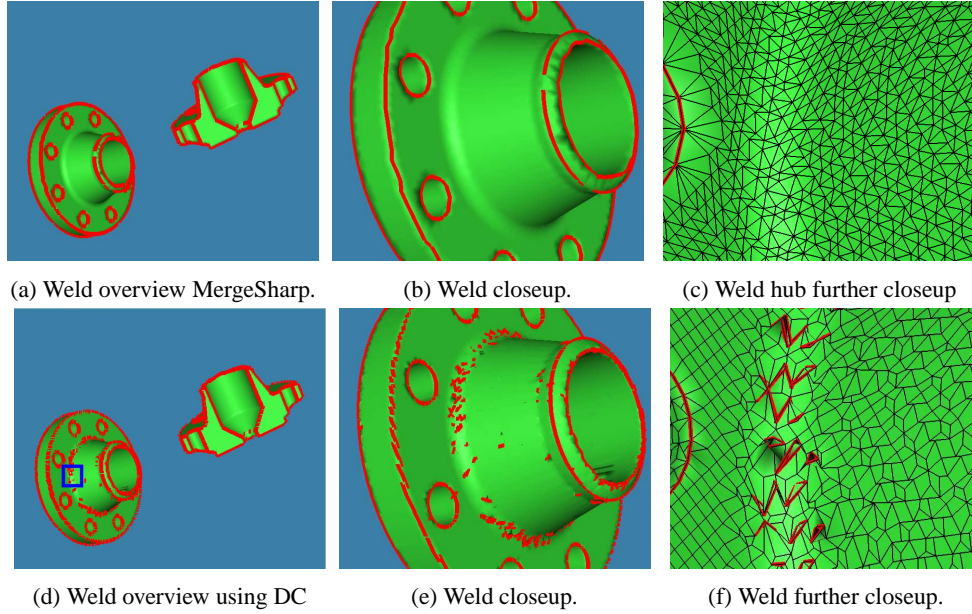


Figure 17: Comparison of Polymender and MergeSharp on the Weld dataset.

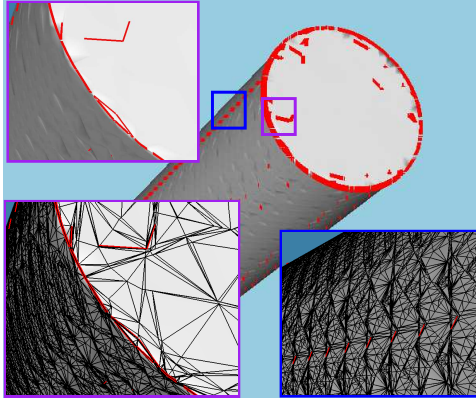


Figure 22: Results of [MS10] on a non-axis aligned cylinder.

MergeSharp and dual contouring with no clamping on noisy data. As seen in Figure 21a the dual contouring output has numerous small errors and the findSharp numbers generated are far more than can be represented in Figure 20. MergeSharp Figure 21b produces far less errors.

We also ran the available code from [MS10] on a cylinder aligned at direction $(1,1,1)$ we show the result in Figure 22. We note that the algorithm generates a lot of very small triangles. In general the results were not competitive to polymender or mergeSharp.

11.4 Timings

We ran our experiments on a standard desktop with four giga-byte ram and with two cores Intel CPU. We show timing results both for hermite data and gradient data. The hermite information for the mesh was generated by the Polymender tool.

Active cubes refer to the number of grid cubes intersected by the isosurfaces. The time taken is divided into following parts. Time to ‘position’ the dual vertices, this time is common to all dual based algorithms. Time to ‘merge’, this is the time taken by MergeSharp. The total time also includes time to extract the triangles.

An isosurface vertex is generated for each active cube using singular value decomposition. This is the most time consuming step (Column ‘position’ of Table 4) and is common to all dual algorithms. Accordingly the time to

Dataset	Polymender				MergeSharp			
	Degree 1	Degree 3	Degree ≥ 4	Total $\neq 2$	Degree 1	Degree 3	Degree ≥ 4	Total $\neq 2$
Weld	437	429	176	1042	48	34	3	85
Brake	674	490	282	1446	7	35	3	45
Rotor arm	58	72	19	149	2	6	2	10
Rotor	647	485	188	1320	2	6	2	10

Table 3: Comparison of Polymender and MergeSharp on triangle mesh data sets.

Dataset	Dimensions	Num Active Cubes	Time (seconds)			
			position	merge	extract	total
Weld	256^3	60389	6.14	0.13	1.36	13
Flange(g)	250^3	154380	9.98	0.22	1.25	20
Flange(h)	250^3	154380	11.1	0.22	1.25	20
Rotorhub	256^3	174323	11.6	0.3	1.4	20

Table 4: Run times for MergeSharp . Active cubes are the cubes intersected by the isosurface, position refers to the time to find the isovertex. Merge is the time taken by our algorithm. Flange (g) refers to the case when we have gradient information and Flange (h) refers to the case when we have hermite data.

position increases with increase in the number of active cubes. Time to run MergeSharp (col ‘merge’ Table 4) also increases with increase in number of active cubes. The increase in time to merge is small compared to the increase in the time to position and is insignificant compared to the total time.

We also show that the time taken to position is less when we have hermite data(Flange(h)) than when we have gradient data (Flange(g)). When we use gradient data, computing the correct gradients for the cube involves some time and thus the difference in timings.

To compute the isosurface vertex location we use the Eigen Library, which we found to be quite slow in practice. We think a dedicated singular value decomposition solver for a 3x3 matrix would decrease the time to position isovertices.

A more detailed timings comparison is shown in the Figure 23. Fig 23a shows how the increase in grid-size effects merge times. The X axis shows grid size for data sets increasing from 100 to 350 cubes per-axis. The Y axis shows the times spent. Apart from the factors mentioned above, we also show the time spent for extracting the dual isovertex from the gradient data and from hermite data separately. Figure 23b shows how the number of active

cubes and the number of merged cubes are affected by the increase in grid-size. We see as with Table 4, the number of active cubes increase with increase in datasize (see Fig 23b) accordingly the time taken to compute the isovertices increases (position Isovert hermite and Pos Isovert grad, see 23a). While time to merge also increases, it remains a small fraction of the total time taken.

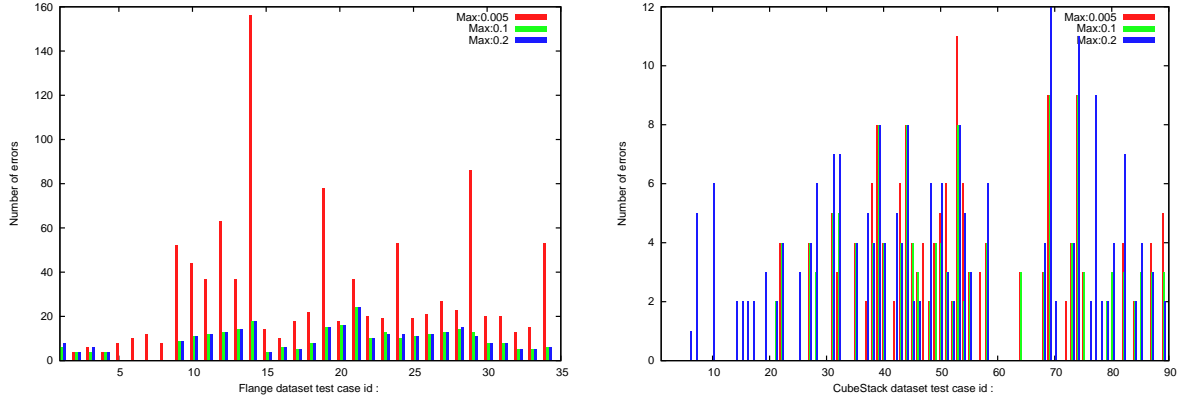


Figure 18: Comparing different thresholds for the singular value. Singular values below the threshold are set to 0 in computing vertex locations.

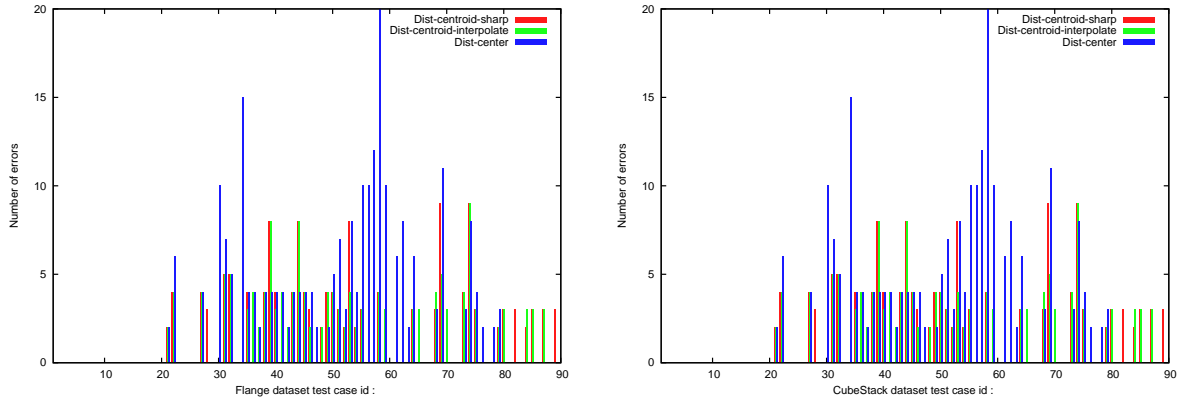


Figure 19: Comparing calculating isosurface vertex position (Equation 1) using distance to cube center and centroid of intersections of isosurface and grid edges. Approximate intersections using interpolation(Dist-centroid-interpolate) or using gradients at edge endpoints (Dist-centroid-sharp).

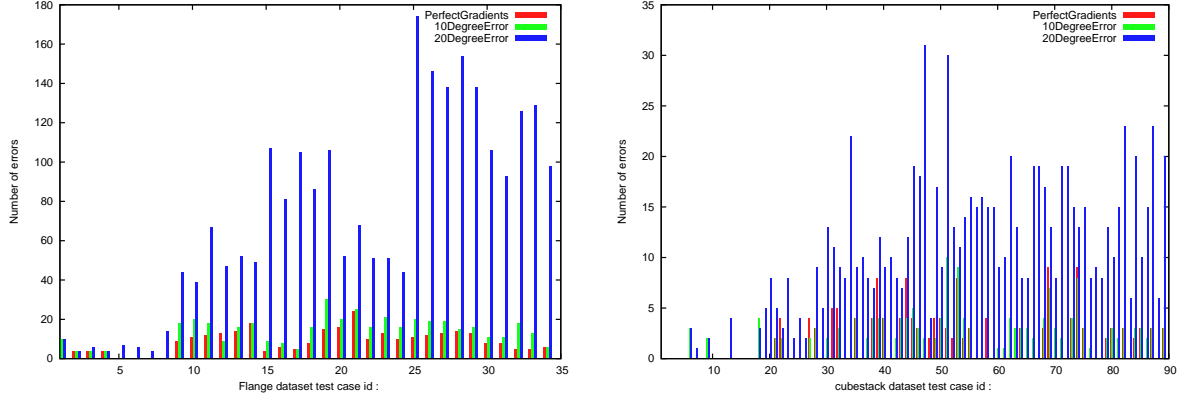


Figure 20: Effect of adding uniform noise to the gradients. Gradients were perturbed uniformly within the given angular bound.

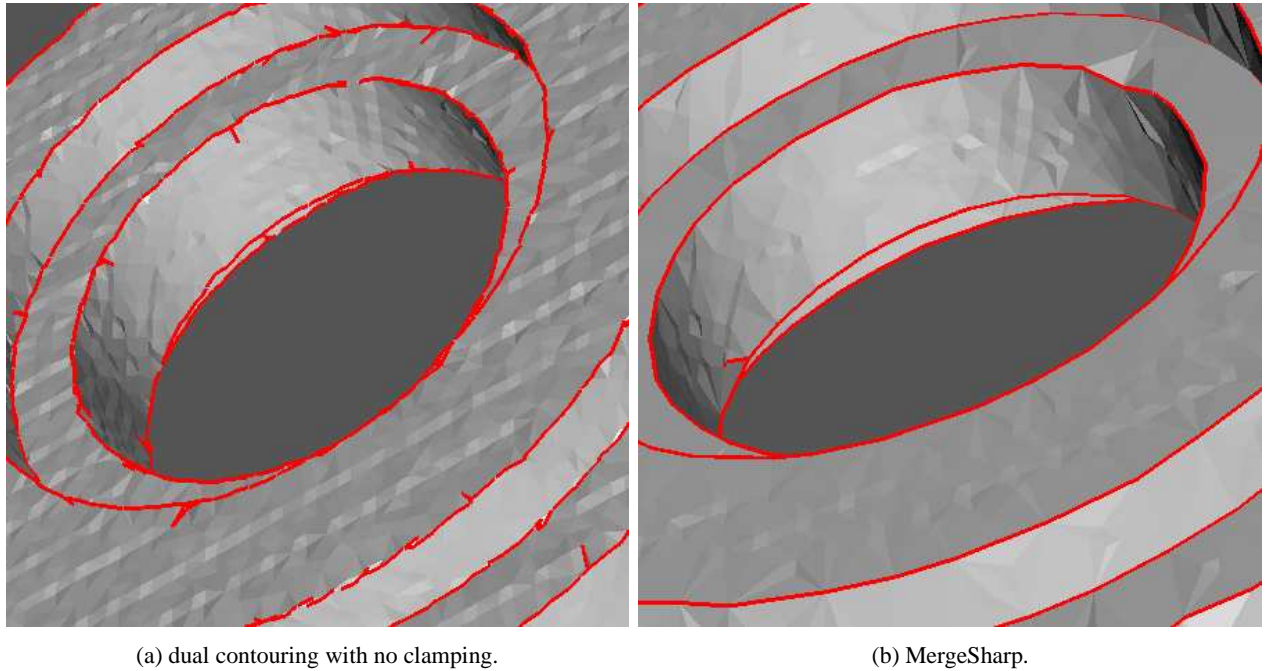
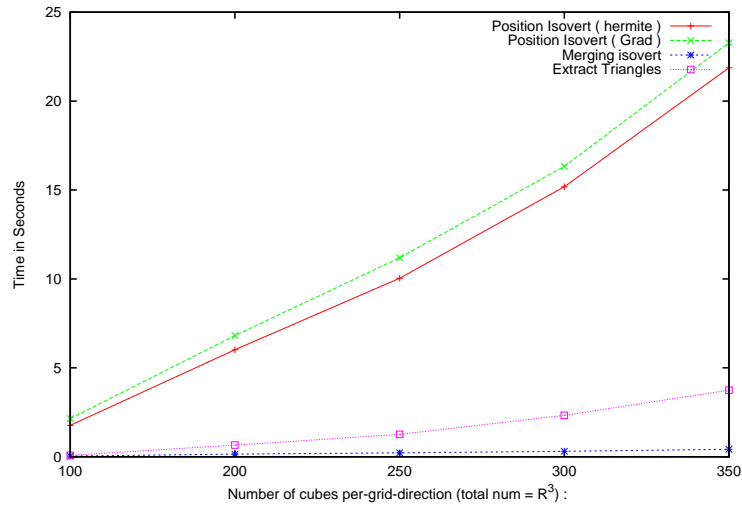
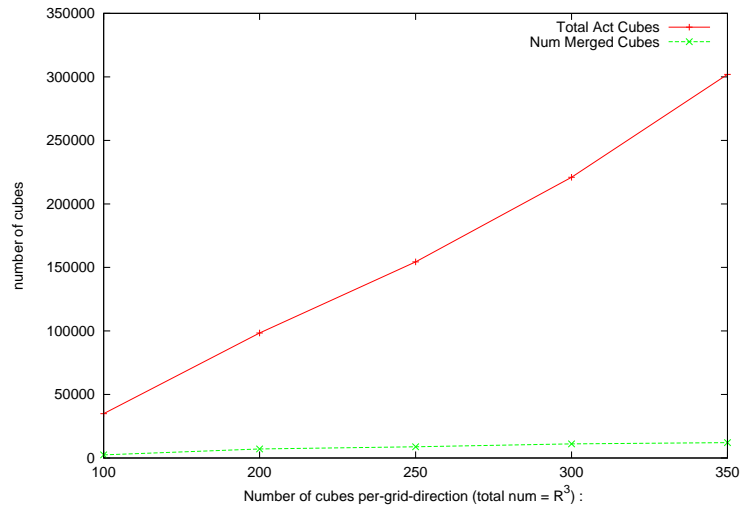


Figure 21: Noisy data where gradients were perturbed uniformly by an angle of 20 degrees. a) Isosurface from dual contouring with no clamping. b) MergeSharp isosurface.



(a) Timing comparison.



(b) Number of cube comparisons.

Figure 23: Time taken by MergeSharp algorithms. Fig a compares the time taken to position isovertices for gradient and hermite data, the time to merge and extract triangles. Fig b shows how the number of merges and number of active cubes changes with increase in dataset size.

TestId	Dataset Type	Data set index	isovalue
1	flange	1	3
2	flange	1	3.8
3	flange	1	3.23
4	flange	1	3.5
5	flange	1	3.9
6	flange	2	3
7	flange	2	3.8
8	flange	2	3.23
9	flange	2	3.5
10	flange	2	3.9
11	flange	3	3
12	flange	3	3.8
13	flange	3	3.23
14	flange	3	3.5
15	flange	3	3.9
16	flange	4	3
17	flange	4	3.8
18	flange	4	3.23
19	flange	4	3.5
20	flange	4	3.9
21	flange	5	3
22	flange	5	3.8
23	flange	5	3.23
24	flange	5	3.5
25	flange	5	3.9
26	flange	6	3
27	flange	6	3.8
28	flange	6	3.23
29	flange	6	3.5
30	flange	6	3.9
31	flange	7	3
32	flange	7	3.8
33	flange	7	3.23
34	flange	7	3.5
35	flange	7	3.9

Table 6: The ‘TestId’ in the comparison graphs, refer to a particular combination of the dataset oriented at a particular direction and an associated isovalue. For example test case 2, is a flange index 5 with isovalue of 3.9. Index 5 in Table 1 shows that it is tilted towards the direction (2,5,7).

12 Discussion and Future Work

The dual contouring algorithm in this paper produces measurably better results than the simpler dual contouring algorithm in [JLSW02] and its implementation in Poly-mender [Ju04]. It also shows measurable robustness under noise and it can run directly from gradient data, not just hermite data.

We do not have implementations of the other algorithms for constructing isosurfaces with sharp features and so were unable to do measurable comparisons. We discuss those algorithms below.

The Extended Marching Cubes [KBSS01] behaves quite like the dual contouring algorithm of [JLSW02]. Thus, we believe that our algorithm would also do measurably better against Extended Marching Cubes on the same test cases given in 11.

Our algorithm is a variation of dual contouring but the central ideas could equally well have been applied to the Extended Marching Cubes framework. Doing so would give Marching Cubes isosurface patches in the smooth regions and isosurface patches generated by our algorithm near the sharp features.

The algorithms in [AB03, GK04, HWC05, VKKM03, ZHK04] implicitly address the problems described in Section 5 by adding more vertices and polygons around sharp features. Our approach is the exact opposite, using fewer, not more, vertices to represent the isosurface near sharp features. We don't know how well those algorithms would perform on the findSharp and countDegree tests of Section 10, but they may do quite well. However, because of their subvoxel constructions, we believe that they are very sensitive to errors in their input data. We are interested in computing sharp features from gradient data and ultimately scalar data where the gradients are computed from a scalar grid. The precise position and gradient information needed for those algorithms is probably not available in scalar input data.

Schaefer and Warren's algorithm in [SW04] is fundamentally different than all the others. They construct a dual grid whose vertices and edges are on sharp isosurface features and apply Marching Cubes to that grid. As noted in their paper, Marching Cubes produces many sliver triangles. While they suggest a method for reducing the number of such slivers, we suspect that the remaining slivers will cause the isosurface to perform poorly on the tests

of Section 10. More importantly, we think the two step process of constructing a dual grid representing the sharp features and extracting the isosurface from that dual grid makes their algorithm very sensitive to errors in the input data.

Many of the dual contouring algorithms support multi-resolution contouring. Our algorithm can easily be modified to support multi-resolution contouring in the smooth regions of the grid. Supporting multi-resolution contouring around the sharp features would be more difficult, but should be possible.

Equation 1 computes the location on a sharp feature closest to some point q_c . Point q_c can be the center of cube c or the centroid of the intersection points of the isosurface and the cube edges. When the input is not Hermite data, the intersection points can be computed using linear interpolation or using gradients at the edge endpoints. We compared all three possibilities for q_c . In Figure 19 we show that using the centroid of intersection points instead of the center measurably improved the algorithm of [JLSW02, Ju04], confirming the findings from [SW02]. For our experimental results in 11, we used the centroid of intersection points computed using gradients from the edge endpoints.

The use of fewer, not more, isosurface vertices around sharp features may seem counter-intuitive but we believe it is in fact correct. Continuity in smooth regions implicitly gives information about surface location. This information is missing near sharp features requiring the use of more sample points to determine feature location. Thus isosurface resolution should be lower, not greater, around sharp features. However, if our algorithm processes smooth regions with high levels of detail as regions containing a sharp feature, then some of that detail will be lost. This is an unfortunate, unintended consequence of our processing of sharp regions. We plan to run experiments to determine the extent of this problem. We also think that some extra processing to determine smooth regions with high levels of detail will help ameliorate this problem.

Kobbelt et. al. in [KBSS01] point out that singular values are sensitive to the number of normals pointing in a given direction, not just the differences between directions. For this reason, they measure the differences between normal directions to detect sharp features and then use the quadric error measure to locate isosurface vertices

on such features. We think that using differences between gradient directions to detect sharp features could improve our algorithm.

The major weakness of our algorithm is our inability to guarantee that the output isosurface is a manifold. We can check that merging vertices in a region does not create non-manifold regions and block merges which do so. The challenge is to make such a check simple and quick, yet does not block merges except where absolutely necessary. For almost all the experimental results in Section 11, the outputs were manifolds suggesting that few blocks would be required. We are currently working on guaranteeing that the output isosurface is a manifold.

Our ultimate goal is to reconstruct sharp isosurfaces from just scalar data. Gradient information must be computed from the scalar data, and thus inherently have some errors. We believe that our algorithm which is robust under gradient errors is an important step toward our goal. The other necessary step toward meeting this goal is a simple, robust algorithm for computing reliable gradients near sharp features.

References

- [AB03] ASHIDA K., BADLER N. I.: Feature preserving manifold mesh from an octree. In *Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications* (2003), ACM Press, pp. 292–297.
- [CDR07] CHENG S.-W., DEY T. K., RAMOS E. A.: Delaunay refinement for piecewise smooth complexes. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2007), SODA '07, Society for Industrial and Applied Mathematics, pp. 1096–1105.
- [CK07] CHENEY E. W., KINCAID D. R.: *Numerical Mathematics and Computing*. Brooks/Cole Publishing Co., Pacific Grove, CA, USA, 2007.
- [DGQ*12] DEY T. K., GE X., QUE Q., SAFA I., WANG L., WANG Y.: Feature-preserving reconstruction of singular surfaces. *Comp. Graph. Forum* 31, 5 (Aug. 2012), 1787–1796.
- [GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1997* (1997), pp. 209–216.
- [Gib98a] GIBSON S. F. F.: Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. In *Proceedings of the First International Conference on Medical Image Computing and Computer-Assisted Intervention, MICCAI 1998* (1998), Springer-Verlag, pp. 888–898.
- [Gib98b] GIBSON S. F. F.: Using distance maps for accurate surface representation in sampled volumes. In *Proceedings of the 1998 IEEE Symposium on Volume Visualization* (1998), pp. 23–30.
- [GK04] GRESS A., KLEIN R.: Efficient representation and extraction of 2-manifold isosurfaces using kd-trees. *Graphical Models* 66, 6 (2004), 370–397.
- [HWC05] HO C., WU F., CHEN B., OUHYOUNG M.: Cubical marching squares: Adaptive feature preserving surface extraction from volume data. *Computer Graphics Forum* 24 (2005), 2005.
- [JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J.: Dual contouring of hermite data. *ACM Transactions on Graphics* 21, 3 (2002), 339–346.
- [Ju04] JU T.: Robust repair of polygonal models. *ACM Transactions on Graphics* 23, 3 (Aug. 2004), 888–895.
- [JU06] JU T., UDESHI T.: Intersection-free contouring on an octree grid. In *Proceedings of the 14th Pacific Conference on Computer Graphics and Applications* (2006).

- [KBSS01] KOBELT L. P., BOTSCH M., SCHWANECKE U., SEIDEL H.-P.: Feature sensitive surface extraction from volume data. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2001* (2001), ACM Press, pp. 57–66.
- [LC87] LORENSEN W., CLINE H.: Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics* 21, 4 (1987), 163–170.
- [Lin00] LINDSTROM P.: Out-of-core simplification of large polygonal models. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2000* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 259–262.
- [MS10] MANSON J., SCHAEFER S.: Isosurfaces over simplicial partitions of multiresolution grids. *Computer Graphics Forum* 29, 2 (2010), 377–385.
- [Nie04] NIELSON G. M.: Dual Marching Cubes. In *Proceedings of IEEE Visualization 2004* (2004), IEEE Computer Society, pp. 489–496.
- [SJW07] SCHAEFER S., JU T., WARREN J.: Manifold dual contouring. *IEEE Transactions on Visualization and Computer Graphics* 13, 3 (May 2007), 610–619.
- [SW02] SCHAEFER S., WARREN J.: *Dual Contouring: The Secret Sauce*. Tech. Rep. TR 02-408, Dept. of Computer Science, Rice University, 2002.
- [SW04] SCHAEFER S., WARREN J.: Dual marching cubes: Primal contouring of dual grids. In *Proceedings of the Computer Graphics and Applications, 12th Pacific Conference* (2004), IEEE Computer Society, pp. 70–76.
- [SYM10] SALMAN N., YVINEC M., MERIGOT Q.: Feature preserving mesh generation from 3d point clouds. *Computer Graphics Forum* 29, 5 (2010), 1623–1632.
- [VKKM03] VARADHAN G., KRISHNAN S., KIM Y. J., MANOCHA D.: Feature-sensitive subdivision and isosurface reconstruction. In *Proceedings of IEEE Visualization 2003* (2003), IEEE Computer Society, pp. 99–106.
- [ZHK04] ZHANG N., HONG W., KAUFMAN A.: Dual contouring with topology-preserving simplification using enhanced cell representation. In *Proceedings of IEEE Visualization 2004* (2004), IEEE Computer Society, pp. 505–512.
- [ZQ12] ZHANG Y., QIAN J.: Dual contouring for domains with topology ambiguity. In *Proceedings of the 20th International Meshing Roundtable*, Quadros W. R., (Ed.). Springer, 2012, pp. 41–60.