SurfaceNets for Multi-Label Segmentations with Preservation of Sharp Boundaries

Sarah F. Frisken
Brigham and Women's Hospital

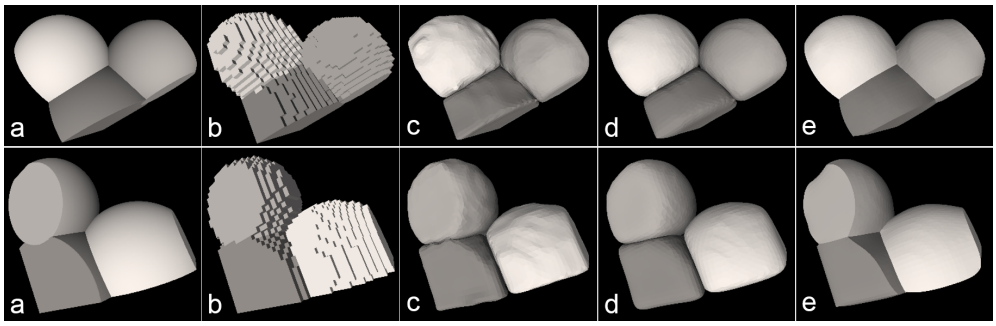

Figure 1. Surfacing a 3D label map with multiple materials. Top and bottom rows show different views of the same model. a) initial model of three intersecting spheres inside a box. b) a cuberille model generated after sampling the model in a volume of resolution 32 x 32 x 32. Surface meshes generated from the sampled volume using c) Marching Cubes (MC) [Lorensen and Cline 1987], d) Flying Edges (FE) [Schroeder et al. 2015], and e) SurfaceNets for multi-label segmentations. Surfaces are rendered with flat shading for comparison. SurfaceNets surfaces are smoother, and edges are sharper than in MC and FE surfaces and topology is preserved where the three spheres meet (where a hole is created by MC and FE).

Abstract

We extend 3D SurfaceNets to generate surfaces of segmented 3D medical images composed of multiple materials represented as indexed labels. Our extension generates smooth, high-quality triangle meshes suitable for rendering and tetrahedralization, preserves topology and sharp boundaries between materials, guarantees a user-specified accuracy, and is fast enough that users can interactively explore the trade-off between accuracy and surface smoothness. We provide open-source code in the form of an extendable C++ library with a simple API, and a Qt and OpenGL-based application that allows users to import or randomly generate multi-label volumes to experiment with surface fairing parameters. In this paper, we describe the basic SurfaceNets algorithm, our extension to handle multiple materials, our method for preserving sharp boundaries between materials, and implementation details used to achieve efficient processing.

 ISSN 2331-7418

1. Introduction

Surfacing methods are used to generate triangle- or quadrilateral-meshes from sampled data and implicit functions for rendering and finite element modeling. Sampled data can be binary (representing a single object), indexed (with each index or 'label' representing a different material), or grey-scale (where values can be an image intensity values, e.g., from a Computed Tomography scan, or a sampled implicit function such as a distance field). Indexed volumes representing multiple materials are a standard output of medical image segmentation; surfacing such data is thus an important component in medical applications such as surgical planning, surgical simulation, and image-guided therapy. Marching Cubes, introduced by Lorensen and Cline [1987], is by far the most commonly used method for surfacing sampled volumes and implementations of Marching Cubes are readily available [Schroeder et al. 2018]. However, Marching Cubes suffers from some quality issues and speed is limited by smoothing which is performed as a post-processing step. SurfaceNets was developed for both binary data and sampled implicit functions to address these issues and was shown to produce better surfaces and higher quality triangle meshes [de Bruin et al. 2000]. Many extensions and improvements have been made to both Marching Cubes and SurfaceNets as described below. SurfaceNets was originally patented by Mitsubishi Electric Research Labs [Gibson 2000], but this patent has recently expired. The goal of this paper is to 1) extend the original SurfaceNets algorithm to generate high-quality, non-overlapping surfaces from multi-label segmented volumes, and 2) provide an open-source implementation of this method in the form of a well-documented C++ library to encourage others to test, extend and improve this implementation.

2. Related Work

We review volume surfacing methods in the three major categories of Marching Cubes, Marching Tetrahedra, and Dual Contouring.

2.1. Marching Cubes

Marching Cubes was originally introduced to construct anatomical models from 3D medical images such as Computed Tomography (CT) and Magnetic Resonance Imaging (MRI), which are composed of grey-scale image values stored in a regular 3D grid. The grid partitions space into hexahedral cells (Figure 2). The algorithm *marches* through the sampled volume to identify cells that intersect a specified iso-surface in the image data (i.e., to identify cells that have some corner values inside the iso-surface and some outside the iso-surface). It then constructs surface patches in the identified cells and stitches these patches together to form the surface mesh. Considering symmetries, 15 unique inside/outside patterns for the 8 cell corners can be identified, with each pattern defining its own surface patch. When considering surface

quality, the important detail is that **surface patch vertices are constrained to lie on cell edges**; the location of a vertex on a particular edge is determined by the grey-scale values of the two cell corners at each end of the edge. Surface fairing (i.e., to smooth the surface and improve triangle shape) is done as a post process independent of the original data. This negatively impacts speed, surface quality and surface accuracy.

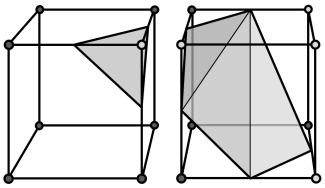

Figure 2. Marching Cubes surface patches for two different cells. The surface intersects a cell if at least one of the 8 corner values is different from the others (Here, light/dark corners are inside/outside respectively). For each cell intersecting the surface, a surface patch is constructed which lies entirely inside the cell. *Surface vertices are constrained to lie on cell edges*.

Since its introduction, there have been multiple extensions to Marching Cubes (e.g., as reviewed by Newman and Yi [2006] and De Araéjo et al. [2015]), and a similar, though less well known, approach was published by Wyvill et al. [1986]. Extensions include methods for data stored in octrees [Schaefer and Warren 2004; Paiva et al. 2006], methods for multi-material data represented as indexed values [Wu and Sullivan Jr 2003; Bischoff and Kobbelt 2006; Reitinger et al. 2005], as probabilities [Hege et al. 1997] and as volume fractions [Anderson et al. 2010], methods preserving fine detail and sharp features [Kobbelt et al. 2001; Schaefer and Warren 2004; Chica et al. 2008], methods that guarantee manifold surfaces [Nielson 2004] or better quality triangles [Dietrich et al. 2009], and methods that optimize processing speed [Schroeder et al. 2015; Vega et al. 2019].

2.2. Marching Tetrahedra

Marching Tetrahedra was first published by Bloomenthal [1994] to tesselate implicit surfaces without requiring a grid sampling of the implicit function. Like Marching Cubes, the algorithm locates cells (in this case tetrahedral cells) containing the surface, determines vertices on edges of the tetrahedra that intersect the surface, constructs a surface patch within each tetrahedra to connect the vertices, and then stitches the surface patches together to form the surface mesh. Marching Tetrahedra has been extended to handle multi-material data represented as indexed values [Nielson and Franke 1997; d'Otreppe et al. 2012] and volume fractions [Bonnell et al. 2000].

2.3. Dual Contouring

Dual Contouring methods are so-called because they are applied to the *dual* of the Marching Cubes hexahedral grid. As in Marching Cubes, cells and cell edges intersected by the iso-surface are identified. However, rather than placing vertices on cell edges and constructing surfaces patches within cells, vertices are placed inside cells intersected by the iso-surface. Quadrilateral surface patches are generated about each edge that intersects the iso-surface by connecting vertices of the 4 hexahedral cells that touch the edge (Figure 3). The advantages over methods that place surface vertices on cell edges are 1) **vertices are not constrained to lie on cell edges**, i.e., they can move about inside the cell; 2) surface fairing is performed in the context of the image data and 3) we can specify a maximum distance between the faired surface and the iso-surface by constraining how far vertices can move from the cell center.

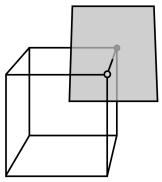

Figure 3. Dual contouring methods place surface vertices inside cells. Surface quadrilaterals are constructed by connecting vertices in neighboring cells that share an edge intersected by the iso-surface (e.g., here a surface quadrilateral is constructed for the top-right edge by connecting the vertex in this cell to the vertices of its right, top, and top-right neighbor cells). *Surface vertices can be moved around inside the cell to improve surface quality*.

SurfaceNets, which we introduced to generate smooth surfaces from binary segmentations of medical image data [Gibson 1998], was possibly the first Dual Contouring method. We subsequently extended SurfaceNets to handle grey-scale data [de Bruin et al. 2000; Frisken et al. 2000], showed that it provides smother surfaces with better quality triangles than Marching Cubes [de Bruin et al. 2000] and extended it for data stored in octrees [Perry and Frisken 2001]. SurfaceNets has also been extended to handle multiple materials [Bertram et al. 2005; Baxter et al. 2011]. Lysenko [2012] published source code for a version of SurfaceNets suitable for GPU implementation. However, his method uses a simplified smoothing operator, does not handle multiple materials, and does not ensure sharp boundaries between materials.

Ju et al. [2002] (who coined the term 'Dual Contouring') presented a Dual Contouring method that preserves sharp edges and corners. This method requires Hermite data, (i.e., data in the form of an implicit function where both the function and its derivatives are available at specified locations). In Ju's method, the Hermite data, i.e., the edge-surface intersection position and the surface normal at that position, are determined for each edge intersecting the surface. Cell vertices are located using the Hermite data by minimizing an energy function that preserves sharp features. Vari-

ations of Ju's method can generate multiple vertices per cell to handle thin features and fine detail [Varadhan et al. 2003; Zhang et al. 2004], handle multiple materials [Schaefer and Warren 2004; Zhang et al. 2008; Zhang et al. 2010], and generate surface meshes from data in octrees [Ju et al. 2002; Varadhan et al. 2003; Zhang et al. 2004; Schaefer et al. 2007]. Recent work has extended Dual Contouring methods to generate volumetric tetrahedral meshes (e.g., for finite element methods) [Zhang et al. 2010; Liang and Zhang 2014]. A GPU implementation was presented by Schmitz et al. [2009; 2010].

While SurfaceNets is technically a Dual Contouring method, the above methods require Hermite data so they cannot be applied to indexed segmentation data. In addition, positioning vertices using energy minimization limits how vertices can be moved within cells, thus constraining surface fairing. The approach we present below preserves sharp boundaries and edges between multiple materials without these constraints.

3. SurfaceNets for Binary Data

We first provide a description of the basic SurfaceNets algorithm and then show how it is enhanced to handle multiple materials and to preserve boundaries between materials.

3.1. Surface Generation

A regularly sampled binary volume with dimensions $N_x \times N_y \times N_z$ consists of $N_x\, N_y\, N_z$ binary samples (where 1 typically represents points inside the shape and 0 typically represents points outside the shape), and $(N_x - 1)(N_y - 1)(N_z - 1)$ cells, each cell having 8 corner samples as illustrated in Figures. 2 and 3. A cell is inside or outside the shape if its corner values are all 1 or all 0, respectively. If at least one of the cell's corner values is different from the others, the surface crosses the cell, so we place a vertex inside the cell. If the surface crosses the cell, there will be at least one edge with different corner values which we call a surface edge. For each surface edge, we construct a quadrilateral of the surface mesh from the cell vertex and the vertices of neighboring cells that share the edge.

By processing cells in left-to-right, bottom-to-top, back-to-front order we can generate the mesh in a single pass. For each cell intersecting the surface, we create a vertex for the cell and consider generating quadrilaterals for the cell's left-bottom, left-back, bottom-back edges, knowing that the neighbors required for generating these quadrilaterals have already been processed and that the cell's remaining 9 edges will be processed as the traversal continues. For the sake of simplicity, ensuring closed surfaces, and removing expensive bounds checking, we make sure the surface does not cross outside faces of the volume. This can easily be done by padding each out-

side face with zeros. This condition can be dropped, (e.g., to stitch together multiple volumes), at the cost of added complexity at volume boundaries.

We use two additional data structures. First, we define a 12-bit bitflag to encode edge crossings for each cell, with each bit representing one of the cell's 12 edges and each bit set to one if the corresponding edge is a surface edge and zero otherwise. Second, we use a vertex data structure that stores the 3D (i, j, k) index of the cell containing the vertex (for fast access during surface fairing and for determining vertex positions), the vertex's 3D floating point offset from the cell's center (which changes during surface fairing), and the cell's 12-bit bitflag. Pseudocode for the basic SurfaceNets algorithm for binary data is given in Listing 1.

3.2. Surface Fairing

The goal of surface fairing is to adjust mesh vertex positions to make the surface smoother and to improve the shape of surface elements by making the spacing between vertices more consistent. This makes it less likely to have small triangles and triangles with small angles, both of which can be problematic for rendering and finite element methods. In the basic SurfaceNets algorithm, this is done by iteratively moving each vertex in the SurfaceNet towards the average position of the vertices of cells that are face-neighbors of the vertex's cell, while constraining each vertex to lie within a specified range of the cell center. Pseudocode for the surface fairing algorithm is given in Listing 2.

StepSize is a value between 0 and 1. Values closer to 1 provide faster smoothing while values closer to zero have better convergence properties. *AllowedRange* could be a specified distance from the cell center (e.g., 1 mm) or a requirement that the vertex remain inside the cell. AllowedRange provides a means for limiting shrinkage of the surface and ensuring a maximim deviation of the smoothed surface from the data. *Optimal smoothing* is a subjective tradeoff between smoothness and fidelity to the initial surface. One of the advantages of SurfaceNets is that it is fast enough for iterative adjustment of smoothing parameters, i.e., the number of iterations of the surface fairing algorithm, StepSize and AllowedRange. This is demonstrated in the application described in Section 5.

4. SurfaceNets for Multi-Label Segmentations

4.1. Surface Generation

Similar to SurfaceNets for binary data, we place a vertex in a cell if at least one of the cell's 8 corner material indices is different from the other corner indices. A

```
vertex list ← empty
mesh quads ← empty

// Traverse volume left-to-right, bottom-to-top, back-to-front
for (k = 0 to N_z − 2) {
  for (j = 0 to N_y − 2) {
    for (i = 0 to N_x − 2) {

      // Set bitflag from the cell's corner values
      bitflag ← 0
      if (any of cell's 12 edges have a zero crossing) {
        (corresponding bits in bitflag) ← 1
      }
      if (bitflag is 0) { // surface does not intersect this cell
        continue to next cell
      }

      // Create a vertex for this cell
      vertex ← cell index (i, j, k), offset (0, 0, 0), bitflag
      vertex list ← add vertex

      // Construct mesh quadrilaterals for back, bottom, left edges
      if (bitflag indicates a left-bottom edge crossing) {
        quad ← vertex of cell and vertices of its left, bottom,
                and left-bottom neighbors
        quad list ← add quad
      }
      if (bitflag indicates a left-back edge crossing) {
        quad ← vertex of this cell and vertices of its left, back,
                and left-back neighbors
        quad list ← add quad
      }
      if (bitflag indicates a bottom-back edge crossing) {
        quad ← vertex of cell and vertices of its bottom, back
                and bottom-back neighbors
        quad list ← add quad
      }
    }
  }
}
```

Listing 1. Basic SurfaceNets Algorithm

```
iterate until an optimal smoothing is reached {
  for each vertex in vertex list {
    cell ← indexed from vertex (i, j, k)
    vertexPosition ← derived from vertex (i, j, k) and vertex offset
    localAverage ← avg. pos. of cell's face neighbor
        vertices (if present)
    vertexPosition ← vertexPosition + stepSize * (localAverage)
    vertex offset ← update using vertex (i, j, k) and vertexPosition
    vertex offset ← constrain vertex offset to lie inside
        AllowedRange
  }
}
```

Listing 2. SurfaceNets surface fairing algorithm

surface edge is defined to be a cell edge that connects a pair of cell corners with different material indices. We construct a quadrilateral for each surface edge the cell's vertex and the vertex of the three neighboring cells that share the surface edge. Each quadrilateral separates exactly two materials (the materials of the surface edge's two corners); the top and bottom faces of the quadrilateral are labeled with the appropriate material (e.g., for setting color or texture during rendering). While quadrilaterals are associated with two materials, cell vertices can be associated with up to 8 materials (one for each unique corner material). This has advantages and disadvantages that are beyond the scope of this paper, but we note that others have extended SurfaceNets and Dual Contouring methods to place more than one vertex in a cell that contains more than two materials.

We traverse the cells in left-to-right, bottom-to-top, back-to-front order and generate cell vertices the same way as for binary data but with a modification to help generate sharp edges and boundaries. Specifically, we add a vertex type and 6 face types to the vertex bitflag. The face type of each of the cell's 6 faces is determined by the number of materials crossing the face, which is determined by the face's 4 corner material indices. The face types specify either no surface crossing (SolidFace), a single surface crossing (SurfaceFace), or two or more surfaces crossing (JunctionFace). These face types are illustrated in Figure 4. The vertex type is determined from the cell's 6 face crossings. If all faces have no surface crossings or a single surface crossing the face, the vertex is a surface vertex (SurfaceVertex). If 1 or more of the faces has more than one surface crossing the face, the vertex is an edge vertex (EdgeVertex). Surface fairing handles these two vertex types differently.

4.2. Surface Fairing with Sharp Boundaries

The multi-material SurfaceNet is treated as a single structure composed of a union of two-sided surface patches each of which separate two materials. A surface enclosing a

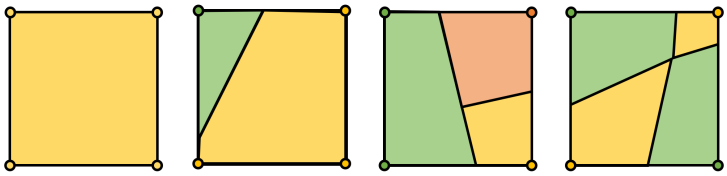

Figure 4. Face types from left-to-right: SolidFace, SurfaceFace, JunctionFace (2 JunctionFace examples shown). JunctionFace includes faces intersected by more than two materials and faces intersected by two materials where vertices of the same material are located on diagonally opposite corners (far right).

single material may be composed of multiple surface patches. Treating the SurfaceNet as a single structure rather than as a set of closed surfaces surface, makes surface fairing faster and preserves topology (i.e., by preventing the surfaces of two materials that touch each other from pulling apart or intersecting each other during smoothing).

To preserve sharp boundaries between objects, we modify the surface fairing method outlined in Section 3.2 to consider the vertex type. Recall that SurfaceVertices lie in cells whose 6 faces separate at most two materials. For these vertices, we use the surface fairing method for binary data of Section 3.2. In contrast, EdgeVertices lie in cells with at least one JunctionFace (Figure 4). A JunctionFace is a face intersected by more than two materials or a face intersected by two materials where vertices of the same material are located on diagonally opposite corners. Note that one of those materials could be the background or 'empty' material. This indicates that the cell contains an edge where three (or more) materials meet. To preserve such edges, during surface fairing we encourage the cell vertex *to move along the edge* and discourage it from moving perpendicular to the edge. To do so, we compute LocalAverage (Section 3.2) *using only vertices from JunctionFace neighbors* and ignore vertex positions from SurfaceFace neighbors. Figure 5 shows how this method improves edge quality.

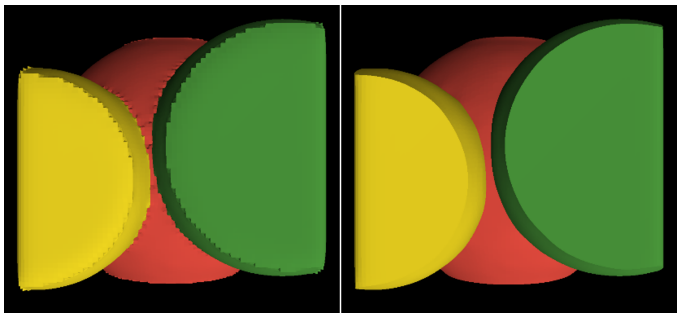

Figure 5. Edges between materials (left) without and (right) with sharp boundary preservation.

4.3. Exporting Surfaces for Rendering

A SurfaceNet consists of a mesh of quadrilaterals, with each quadrilateral separating two materials. We provide two example methods for exporting SurfaceNets to conventional formats for rendering and further processing (e.g., tetrahedral meshing for finite element modeling). These two methods are provided for visualization and validation, and as templates for custom export methods. The first example method exports the SurfaceNet as C-style arrays of vertices and triangle indices for rendering via OpenGL. We generate two triangles for each quadrilateral. Each triangle consists of three vertices so that mesh vertices are duplicated where triangles meet). Each vertex consists of 8 floats: 3 positions (x, y, z), 3 surface normal components (n_x, n_y, n_z), and two texture coordinates (s, t), where (n_x, n_y, n_z) is the face normal of the triangle (to provide flat shading which is best for validating surface quality). The texture coordinates encode material indices of the top and bottom faces of the triangle. To ensure that texture coordinates lie in $[0, 1]$, we compute s and t by dividing the material index by the number of materials, N_I. For rendering using OpenGL, we generate a 1D texture map with N_I components. The texture map stores an RGBA color for each material; the s and t values associated with vertices of a particular triangle thus provide the color of the top and bottom face of the triangle. Changing the color for a material or making its surface or the boundary between two materials transparent simply requires modifying this texture map.

The second method exports a SurfaceNet as a set of surfaces, with each surface enclosing a single material. Vertices are consistently ordered in counterclockwise order to distinguish between inside and outside faces for rendering. Each surface is stored in a C++ std::vector of floating point vertex positions and a C++ std::vector of integer triangle indices, where each index references a vertex in the vertex vector. In the sample application provided with this library (Section 6), we export this geometry as OBJ files (a standard representation used for rendering and geometry processing).

4.4. Reducing Surface Self-Intersections

Each quadrilateral can be partitioned into two triangles as illustrated in Figure 6. Because SurfaceNets quadrilaterals are generally non-planar, the choice of the partitioning can affect the local surface shape. When a quadrilateral is highly non-planar (e.g., near an edge), the choice of partitioning for neighboring quadrilaterals can cause triangles to intersect each other. To avoid local self-intersections, we can consider both patterns and chose the partitioning that minimizes local surface area. This is slightly slower, but may be critical for some applications (e.g., tetrahedralization and 3D printing). We apply this technique for the export method used to generate OBJ files.

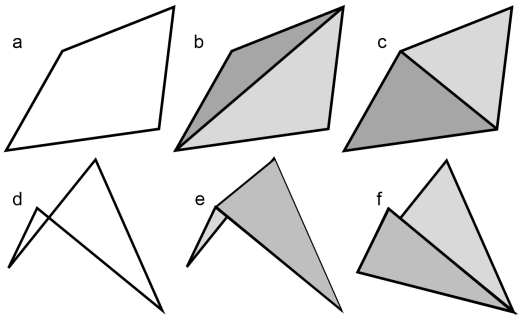

Figure 6. Quadrilaterals (a, d) can be partitioned into two different triangle patterns (b, c and e, f). When the quadrilateral is non-planar (d), the choice of partitioning can impact local shape.

5. A Simple Application

To test and demonstrate the proposed extensions of SurfaceNets, we also provide a sample C++ application using Qt for input, output and controlling parameters, and OpenGL for rendering and view control. A screenshot of this application is shown in Figure 7. All examples in this manuscript were generated using this application. Geometry can either be generated or input from a file. To input from a file, the user specifies the name of a file in binary format composed of 8-bit or 16-bit indices stored in left-to-right, bottom-to-top, back-to-front order, the image dimensions, and voxel size. To generate geometry, the user specifies the image dimensions, voxel sizes, and a desired number of spheres. The spheres are generated with random sizes and radii within the specified volume. Figure 7 shows the SurfaceNet for 100 spheres randomly placed in a $256 \times 256 \times 256$ volume of $1 \text{ mm} \times 1 \text{ mm} \times 1 \text{ mm}$ voxels. Qt sliders allow the user to experiment with different surface fairing parameters and to edit the color and visibility of each material. The system creates OpenGL data internally for rendering. OBJ files can be exported using the Qt interface.

6. Results

Figures 1, 7, and 8 show the high-quality surfaces that can be generated from multi-label data using SurfaceNets. All models in this paper are rendered using flat shading to better evaluate surface quality.

Figure 9 shows the effect on surface fairing of the maximum distance a vertex can move from cell center and the number of smoothing iterations. Because there are trade-offs between how accurately the surface fits the original data, speed, and smoothness, it is useful to be able to fine tune these parameters. The speed of the proposed method makes interactive tuning possible.

Figure 10 shows that this method handles volumes with non-cubic voxels well.

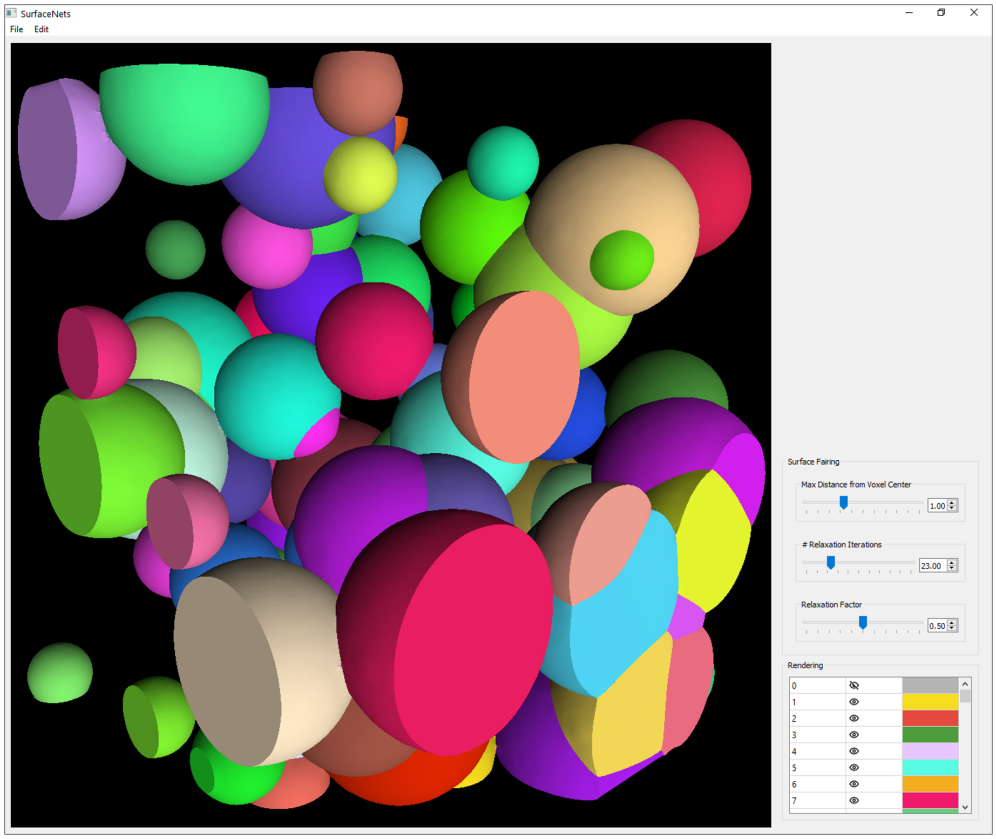

Figure 7. A screen shot of a simple application for testing and demonstrating the proposed method. The model here was generated from 100 spheres with random center positions and radii inside a $256 \times 256 \times 256$ volume. Sliders on the right control panel are used to adjust surface fairing parameters. The SurfaceNet is rendered using flat shading for better evaluation of surface smoothness.

Medical data often consists of non-cubic voxels, e.g., 3D volumes composed of a stack of 2D images, where the spacing between image pixels is smaller than the spacing between image planes. In Figure 10, voxel (x, y, z) sizes are $1 \text{ mm} \times 1 \text{ mm} \times 4 \text{ mm}$. SurfaceNets does not exhibit the terracing artifacts present for this kind of data in other surfacing methods where smoothing is performed as a post-processing step.

Figures 11 and 12 compare surfaces generated with SurfaceNets and Flying Edges from a multi-label segmentation of a brain (using the Flying Edges implementation in the Segmentations module of 3D Slicer 4.11 [Kikinis and Pieper 2011]). SurfaceNets constructs a single mesh, ensures a maximum relaxation error, and maintains sharp edges and corners between materials; while Flying Edges generates and relaxes each surface independently, does not guarantee a maximum error, and does not attempt to maintain sharp edges and corners between materials. These differences are high-

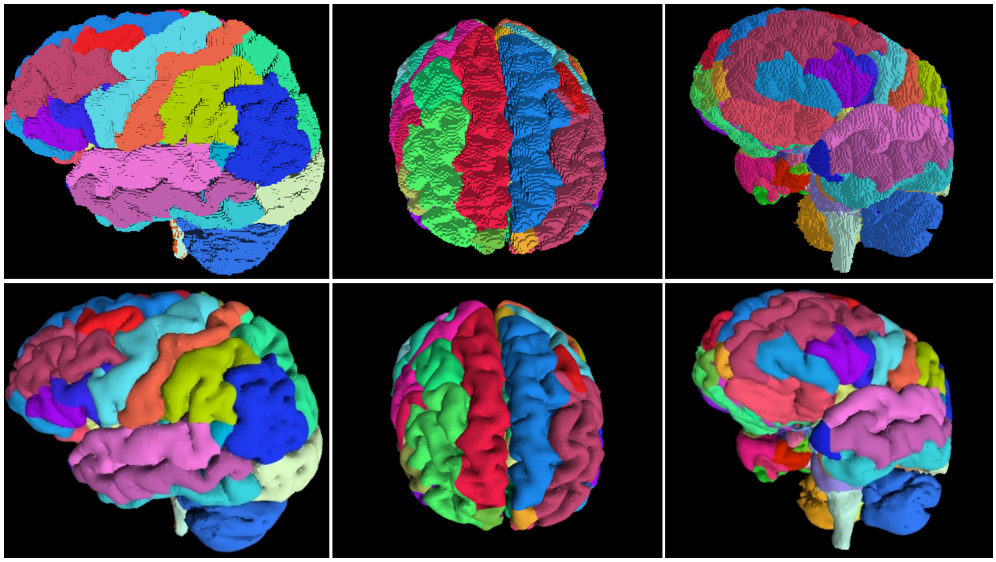

Figure 8. A SurfaceNet generated from a manually and semi-automatically segmented brain partitioned into more than 100 different anatomical structures. Top row shows 3 views of the binary segmented volume. Bottom row shows corresponding views of the SurfaceNet surface. Volume size is $256 \times 256 \times 159$ voxels and voxel sizes are $0.95\ \mathrm{mm} \times 0.95\ \mathrm{mm} \times 1.5\ \mathrm{mm}$.

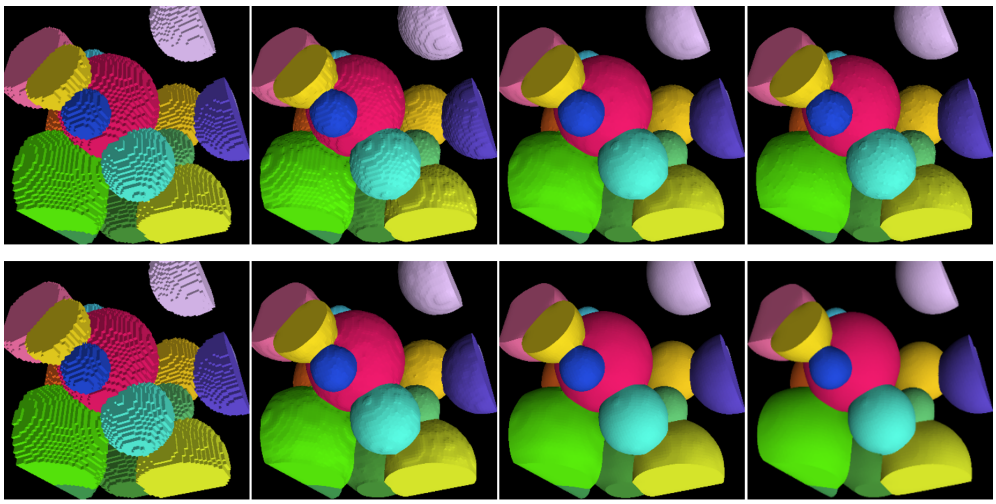

Figure 9. The effect of surface fairing parameters on smoothness and accuracy. Vertices are constrained to lie within $0.5\ \mathrm{mm}$ and $1\ \mathrm{mm}$ of the center of the $1\ \mathrm{mm} \times 1\ \mathrm{mm} \times 1\ \mathrm{mm}$ voxels for the top and bottom rows, respectively. From left to right: no smoothing, too few smoothing iterations, optimal iterations, too many iterations for the given accuracy.

lighted in Figure 11 and 12. In Figure 11, areas where brain sulci are labeled according to function (as opposed to geometry) are correctly partitioned by color only by SurfaceNets but are incorrectly separated by Flying Edges (white circles), resulting

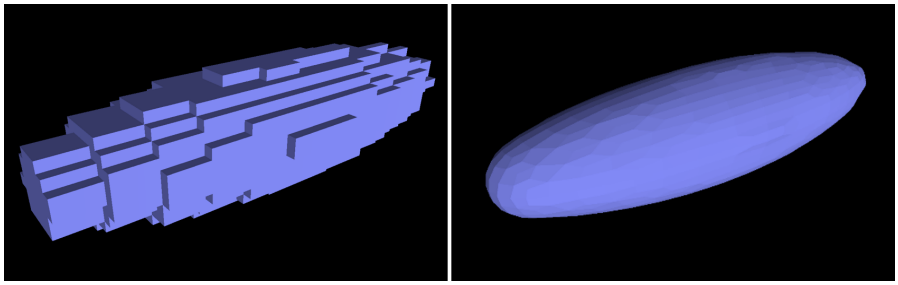

Figure 10. SurfaceNets handles smoothing of non-cubic voxels well. Left: a cuberille surface of an ellipsoid with voxel size $1\ \mathrm{mm} \times 1\ \mathrm{mm} \times 4\ \mathrm{mm}$. Right the SurfaceNet generated for this model.

in inaccurate brain anatomy. In Figure 12, while SurfaceNets preserves anatomical structure, Flying Edges incorrectly separates structures of the optic chiasm (white circles) and degrades the small mammillary bodies (yellow circles).

Figure 13 shows total CPU time required for generating, fairing, and exporting an OpenGL model for rendering with SurfaceNets for a variety of models of randomly generated intersecting spheres. Surface generation time is roughly proportional to volume size, while surface fairing and surface export for rendering (and hence the total time) are roughly proportional to the number of quadrilaterals in the surface mesh. Total times range from sub-second for hundreds of quads to 3.5 seconds for 800K quads. These tests were performed on a Dell XPS 8940 desktop with a i7-11700 Intel processor @ 2.50 GHz with 16 logical processors. These times are fast enough to interactively optimize surface fairing parameters and compare very favorably with Marching Cubes and Flying Edges surfacing. While a detailed timing comparison against these methods is outside the goals of this paper, we note that our SurfaceNets implementation running on a single processor required approximately 4 seconds to generate, fair and export a surface from a multi-material volume ($256 \times 256 \times 256$ with 100 randomly generated spheres). As a rough comparison, the highly optimized parallel implementation of Flying Edges in 3D Slicer's [Kikinis and Pieper 2011] Segmentations module also required approximately 4 seconds for the same model, but the method was running on all 16 processors (with an even load balancing between processors observed in the Windows Task Manager). The Marching Cubes implementation in 3D Slicer's Greyscale Model Maker module required multiple minutes to generate, smooth and render the same model.

7. Discussion

In this paper, we have reviewed the basic SurfaceNets algorithm and presented an extension that allows it to handle 3D segmented medical images, where segmented regions are represented as material indices. We have presented a method that pre-

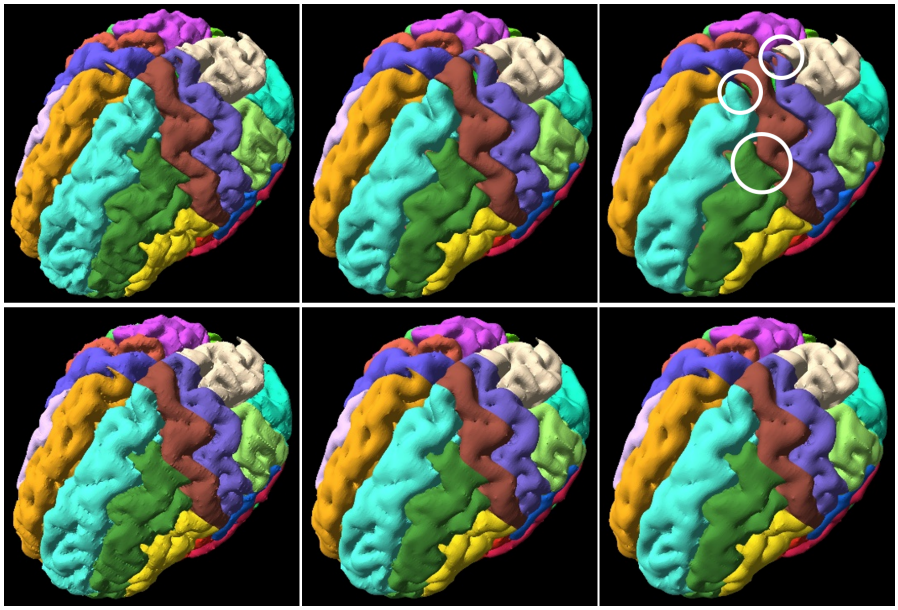

Figure 11. Superior view of the brain model of Figure 8 with low to high levels of smoothing (left to right). Top row: Flying Edges. Bottom row: SurfaceNets. Flying Edges surfaces were generated using the Segmentations module in 3D Slicer 4.11, with smoothing set to 25%, 50% and 100% (left to right). SurfaceNets surfaces were generated with parameters: {maximum distance from cell center in voxels, # relaxations, and relaxation factor} of $\{0.6, 10, 0.4\}$, $\{1.0, 18, 0.4\}$ and $\{1.4, 26, 0.4\}$ (left to right). Parameters were manually chosen to achieve comparable smoothing for the two methods in regions of low curvature. White circles show where Flying Edges generates inaccurate anatomy (see text).

serves sharp edges where multiple materials meet, described methods for exporting a multi-material SurfaceNet to two different standard representations (OpenGL and OBJ files), and describe a simple application to demonstrate and test this method with data input from a file or randomly generated by the application.

Given the large body of prior work on surfacing methods and the ubiquity of Marching Cubes and its extensions, it is somewhat daunting to write a paper on surface meshing. However, we believe that SurfaceNets offers a viable alternative that 1) reduces terracing artifacts in indexed data to provide smoother, better-quality surfaces, which we believe is sorely needed in the medical field, 2) preserves topology, 3) handles non-cubic voxels well, and 4) is fast enough for interactive setting of surface fairing parameters. We hope that by providing an open-source implementation of SurfaceNets we will encourage others to build upon and improve this work, perhaps by incorporating some of the many improvements and extensions described in Section 2 that have been proposed for Marching Cubes, SurfaceNets and Dual Contoring (e.g., handling implicit data, octree data structures, parallel processing, GPU

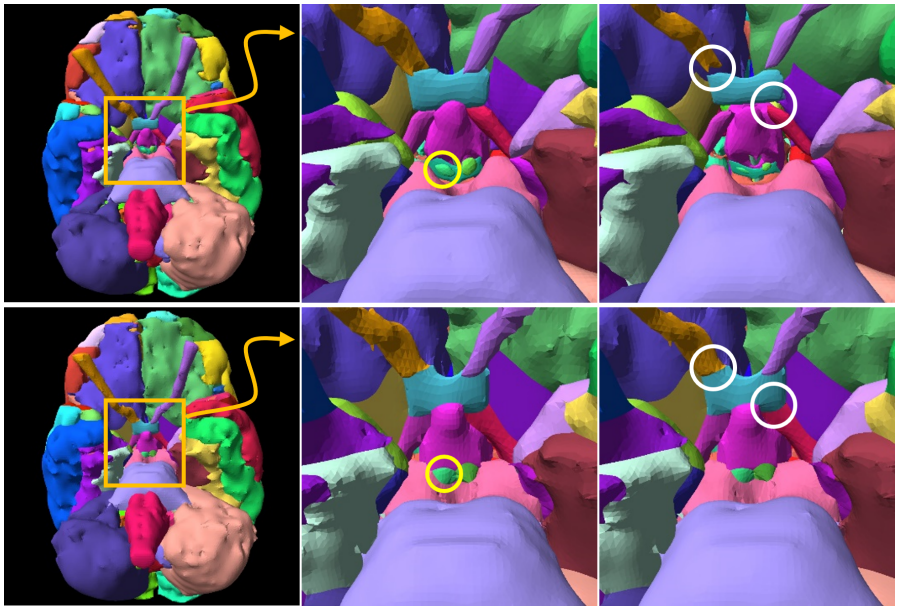

Figure 12. Inferior view of the surfaces of Figure 11. Top row: Flying Edges; bottom row: SurfaceNets. Left images show surfaces with medium levels of smoothing. Middle and right rows show close ups of the indicated region for surfaces with medium and high levels of smoothing respectively. During smoothing, Flying Edges does not preserve connections between structures (white circles) and can degrade small structures (yellow circles) while SurfaceNets preserves the anatomy.

implementations, etc.).

Acknowledgements

The author would like to acknowledge Michael Halle for helpful discussions that helped inspire this work. This work was supported in part by the U.S. National Institute of Health under grants R01 EB027134-01 and P41 EB028741-01

References

ANDERSON, J. C., GARTH, C., DUCHAINEAU, M. A., AND JOY, K. I. 2010. Smooth, volume-accurate material interface reconstruction. *IEEE Transactions on Visualization and Computer Graphics 16*, 5, 802–814. URL: https://ieeexplore.ieee.org/abstract/document/5383354/. 36

BAXTER, J. S., PETERS, T. M., AND CHEN, E. C. 2011. A unified framework for voxel classification and triangulation. In *Medical Imaging 2011: Visualization, Image-Guided Procedures, and Modeling*, vol. 7964, SPIE, 933–940. URL: https://doi.org/10.1117/12.877715. 37

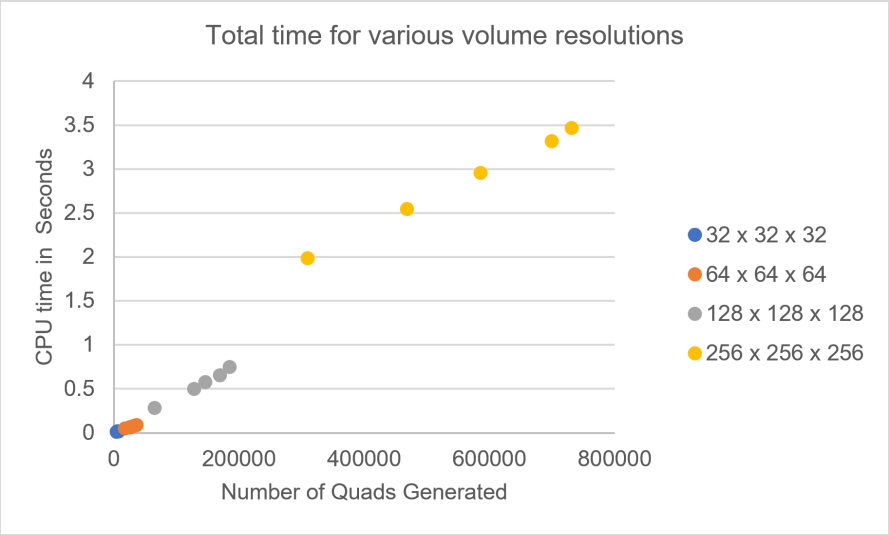

Figure 13. The total time required to generate the SurfaceNet, fair the surface, and export geometry for rendering varies approximately linearly with the number of quadrilaterals in the surface mesh, which depends on the dimensions of the 3D volume containing the multi-label data and the complexity of the surface in the data. In this test, the complexity of the surface was controlled by the number of randomly generated spheres composed to generate the labeled data.

BERTRAM, M., REIS, G., VAN LENGEN, R. H., KÖHN, S., AND HAGEN, H. 2005. Non-manifold mesh extraction from time-varying segmented volumes used for modeling a human heart. In *Proceedings of EuroVis 2005: The Eurographics/IEEE VGTC Conference on Visualization*, Eurographics, 199–206. URL: http://dx.doi.org/10.2312/VisSym/EuroVis05/199-206. 37

BISCHOFF, S., AND KOBBELT, L. 2006. Extracting consistent and manifold interfaces from multi-valued volume data sets. In *Bildverarbeitung für die Medizin 2006*. Springer, 281–285. URL: https://link.springer.com/chapter/10.1007/3-540-32137-3_57. 36

BLOOMENTHAL, J. 1994. An implicit surface polygonizer. *Graphics Gems 4*, 324–350. URL: https://people.eecs.berkeley.edu/~jrs/meshpapers/Bloomenthal.pdf. 36

BONNELL, K. S., SCHIKORE, D. R., JOY, K. I., DUCHAINEAU, M., AND HAMANN, B. 2000. Constructing material interfaces from data sets with volume-fraction information. In *Proceedings of Visualization 2000*, IEEE. URL: https://ieeexplore.ieee.org/abstract/document/885717/. 36

CHICA, A., WILLIAMS, J., ANDÚJAR, C., BRUNET, P., NAVAZO, I., ROSSIGNAC, J., AND VINACUA, À. 2008. Pressing: Smooth isosurfaces with flats from binary grids. *Computer Graphics Forum 27*, 1, 36–46. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2007.01039.x. 36

DE ARAÚJO, B. R., LOPES, D. S., JEPP, P., JORGE, J. A., AND WYVILL, B. 2015. A survey on implicit surface polygonization. *ACM Computing Surveys (CSUR) 47*, 4, 1–39. URL: https://dl.acm.org/doi/abs/10.1145/2732197. 36

DE BRUIN, P. W., VOS, F., POST, F. H., FRISKEN-GIBSON, S., AND VOSSEPOEL, A. M. 2000. Improving triangle mesh quality with SurfaceNets. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, 804–813. URL: https://link.springer.com/chapter/10.1007/978-3-540-40899-4_83. 35, 37

DIETRICH, C. A., SCHEIDEGGER, C. E., COMBA, J. L., NEDEL, L. P., AND SILVA, C. T. 2009. Marching cubes without skinny triangles. *Computing in Science & Engineering 11*, 2, 82–87. URL: https://ieeexplore.ieee.org/abstract/document/4784402/. 36

D'OTREPPE, V., BOMAN, R., AND PONTHOT, J.-P. 2012. Generating smooth surface meshes from multi-region medical images. *International Journal for Numerical Methods in Biomedical Engineering 28*, 6-7, 642–660. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.1471. 36

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, the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, 249–254. URL: https://dl.acm.org/doi/abs/10.1145/344779.344899. 37

GIBSON, S. F. 1998. Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, 888–898. 37

GIBSON, S. F., 2000. Surface net smoothing for surface representation from binary sampled data, July. US Patent 6,084,593. 35

HEGE, H.-C., SEEBASS, M., STALLING, D., AND ZÖCKLER, M., 1997. A generalized marching cubes algorithm based on non-binary classifications. ZIB-Report SC-97-05. URL: https://nbn-resolving.org/urn:nbn:de:0297-zib-2741. 36

JU, T., LOSASSO, F., SCHAEFER, S., AND WARREN, J. 2002. Dual contouring of hermite data. In *Proceedings of SIGGRAPH 2002, the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, 339–346. URL: https://dl.acm.org/doi/abs/10.1145/566570.566586. 37, 38

KIKINIS, R., AND PIEPER, S. 2011. 3D Slicer as a tool for interactive brain tumor segmentation. In *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, IEEE, 6982–6984. URL: https://ieeexplore.ieee.org/abstract/document/6091765/. 45, 47

KOBBELT, L. P., BOTSCH, M., SCHWANECKE, U., AND SEIDEL, H.-P. 2001. Feature sensitive surface extraction from volume data. In *Proceedings of SIGGRAPH 2001, the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, 57–66. URL: https://dl.acm.org/doi/abs/10.1145/383259.383265. 36

LIANG, X., AND ZHANG, Y. 2014. An octree-based dual contouring method for triangular and tetrahedral mesh generation with guaranteed angle range. *Engineering with Computers 30*, 2, 211–222. URL: https://link.springer.com/article/10.1007/s00366-013-0328-8. 38

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of SIGGRAPH '87, the 14th Annual Conference on Computer Graphics and Interactive Techniques*, Association for Computing Machinery, 163–169. URL: https://doi.org/10.1145/37401.37422. 34, 35

LYSENKO, M., 2012. Smooth voxel terrain (part 2). URL: https://0fps.net/2012/07/12/smooth-voxel-terrain-part-2/. 37

NEWMAN, T. S., AND YI, H. 2006. A survey of the marching cubes algorithm. *Computers & Graphics 30*, 5, 854–879. URL: https://www.sciencedirect.com/science/article/pii/S0097849306001336. 36

NIELSON, G. M., AND FRANKE, R. 1997. Computing the separating surface for segmented data. In *Proceedings of Visualization'97*, IEEE, 229–233. URL: https://ieeexplore.ieee.org/abstract/document/663887/. 36

NIELSON, G. M. 2004. Dual marching cubes. In *Proceedings of Visualization 2004*, IEEE, 489–496. URL: https://ieeexplore.ieee.org/abstract/document/1372234/. 36

PAIVA, A., LOPES, H., LEWINER, T., AND DE FIGUEIREDO, L. H. 2006. Robust adaptive meshes for implicit surfaces. In *Proceedings of the 19th Brazilian symposium on computer graphics and image processing*, IEEE, 205–212. URL: https://ieeexplore.ieee.org/abstract/document/4027069/. 36

PERRY, R. N., AND FRISKEN, S. F. 2001. Kizamu: A system for sculpting digital characters. In *Proceedings of SIGGRAPH 2001, the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, 47–56. URL: https://dl.acm.org/doi/abs/10.1145/383259.383264. 37

REITINGER, B., BORNIK, A., AND BEICHEL, R. 2005. Constructing smooth non-manifold meshes of multi-labeled volumetric datasets. In *Proceedings of WSCG 2005*, Eurograpics. URL: https://otik.uk.zcu.cz/handle/11025/10972. 36

SCHAEFER, S., AND WARREN, J. 2004. Dual marching cubes: Primal contouring of dual grids. In *Proceedings of the 12th Pacific Conference on Computer Graphics and Applications*, IEEE, 70–76. URL: https://ieeexplore.ieee.org/abstract/document/1348336/. 36, 38

SCHAEFER, S., JU, T., AND WARREN, J. 2007. Manifold dual contouring. *IEEE Transactions on Visualization and Computer Graphics 13*, 3, 610–619. URL: https://ieeexplore.ieee.org/abstract/document/4297690/. 38

SCHMITZ, L. A., DIETRICH, C. A., AND COMBA, J. L. 2009. Efficient and high quality contouring of isosurfaces on uniform grids. In *2009 XXII Brazilian Symposium on Computer Graphics and Image Processing*, IEEE, 64–71. URL: https://ieeexplore.ieee.org/abstract/document/5395251/. 38

SCHMITZ, L., SCHEIDEGGER, L. F., OSMARI, D. K., DIETRICH, C. A., AND COMBA, J. L. D. 2010. Efficient and quality contouring algorithms on the gpu. *Computer Graphics Forum 29*, 8, 2569–2578. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2010.01825.x. 38

SCHROEDER, W., MAYNARD, R., AND GEVECI, B. 2015. Flying edges: A high-performance scalable isocontouring algorithm. In *Proceedings of the 2015 IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV)*, LDAV '15, IEEE Computer Society, 33–40. URL: https://doi.org/10.1109/LDAV.2015.7348069. 34, 36

SCHROEDER, W., MARTIN, K. M., AND LORENSEN, W. E. 2018. *The visualization toolkit: an object-oriented approach to 3D graphics*. Prentice-Hall, Inc. URL: https://gitlab.kitware.com/vtk/textbook. 35

VARADHAN, G., KRISHNAN, S., KIM, Y. J., AND MANOCHA, D. 2003. Feature-sensitive subdivision and isosurface reconstruction. In *Proceedings of Visualization 2003*, IEEE, 99–106. URL: https://ieeexplore.ieee.org/abstract/document/1250360/. 38

VEGA, D., ABACHE, J., AND COLL, D. 2019. A fast and memory-saving marching cubes 33 implementation with the correct interior test. *Journal of Computer Graphics Techniques 8*, 3. URL: https://www.jcgt.org/published/0008/03/01/. 36

WU, Z., AND SULLIVAN JR, J. M. 2003. Multiple material marching cubes algorithm. *International Journal for Numerical Methods in Engineering 58*, 2, 189–207. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.775. 36

WYVILL, G., MCPHEETERS, C., AND WYVILL, B. 1986. Data structure for soft objects. *The Visual Computer 2*, 227–234. URL: https://link.springer.com/article/10.1007/BF01900346. 36

ZHANG, N., HONG, W., AND KAUFMAN, A. 2004. Dual contouring with topology-preserving simplification using enhanced cell representation. In *Proceedings of Visualization 2004*, IEEE, 505–512. URL: https://ieeexplore.ieee.org/document/1372236. 38

ZHANG, Y., HUGHES, T. J., AND BAJAJ, C. L. 2008. Automatic 3D mesh generation for a domain with multiple materials. In *Proceedings of the 16th international meshing roundtable*, Springer, 367–386. URL: https://link.springer.com/chapter/10.1007/978-3-540-75103-8_21. 38

ZHANG, Y., HUGHES, T. J., AND BAJAJ, C. L. 2010. An automatic 3D mesh generation method for domains with multiple materials. *Computer methods in applied mechanics and engineering 199*, 5-8, 405–415. URL: https://www.sciencedirect.com/science/article/pii/S004578250900214X. 38

Index of Supplemental Materials

jcgt.org/published/0011/01/03/SurfaceNets.zip
Contains a zip file with source code, a README document, and a description of the open-source MIT license in place for the source code. The source code includes a C++ library

for Multi-Material SurfaceNets and the application of Section 5. These are provided with a Visual Studio solution for convenience.

Author Contact Information

Sarah F. Frisken
Brigham and Women's Hospital
75 Francis St., Boston, MA, 02115
sfrisken@bwh.harvard.edu

Sarah F. Frisken, SurfaceNets for Multi-Label Segmentations with Preservation of Sharp Boundaries, *Journal of Computer Graphics Techniques (JCGT)*, vol. 11, no. 1, 34–54, 2022
http://jcgt.org/published/0011/01/03/

Received:	2021-07-10		
Recommended:	2021-12-27	Corresponding Editor:	Alexander Wilkie
Published:	2022-02-28	Editor-in-Chief:	Marc Olano