A Smoothing Algorithm for the Dual Marching Tetrahedra Method

by

Sean Johnson

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2011 by the
Graduate Supervisory Committee:

Gerald Farin, Chair
Andrea Richa
Vineeth Nallure Balasubramanian

ARIZONA STATE UNIVERSITY

December 2011

ABSTRACT

The Dual Marching Tetrahedra algorithm is a generalization of the Dual Marching Cubes algorithm, used to build a boundary surface around points which have been assigned a particular scalar density value, such as the data produced by and Magnetic Resonance Imaging or Computed Tomography scanner. This boundary acts as a skin between points which are determined to be “inside” and “outside” of an object. However, the DMT is vague in regards to exactly where each vertex of the boundary should be placed, which will not necessarily produce smooth results. Mesh smoothing algorithms which ignore the DMT data structures may distort the output mesh so that it could incorrectly include or exclude density points. Thus, an algorithm is presented here which is designed to smooth the output mesh, while obeying the underlying data structures of the DMT algorithm.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Machines such as MRI scanners are capable of determining density at discrete locations in an object, allowing us to peer into the inner workings of closed systems such as the human body. However, in order to fully utilize the data produced, there have been several algorithms proposed. For data sets aligned to a 3 dimensional lattice, such as that produced by an MRI, it is common to use the “Marching Cube” algorithm, which forms “voxels”, or units of volume, as cubes along the grid. The points in the lattice are segmented based on their density, to either be inside an object, or outside an object. Then, the cubes are placed using these “inside” points as their vertices. The surface cubes are then used to form a “skin” around the data points determined to be on the “inside” of such an object. (Lorensen and Cline, 164)

However, the Marching Cubes and Dual Marching Cubes algorithms may be generalized to fit more diverse data sets. The “Dual Marching Tetrahedra” algorithm instead uses tetrahedra for its voxels, which do not require the same constraints on the data set as the Cubes algorithms, which contain all right angles as each data point must be aligned to a grid (Nielson, “Dual Marching Tetrahedra”, 184). The algorithm also allows for more complex segmentation of the data points by the introduction of an ambiguous case, which we may use when it is not clear that a point should be marked as “inside” or “outside”. (Nielson, “Dual Marching Tetrahedra”, 188)

Unfortunately, since we are building a surface around discrete data, the results are not necessarily smooth. Furthermore, these algorithms build non-continuous surfaces, and so there is some ambiguity regarding vertex placement, as some locations may be smoother than others. However, smoothing algorithms which seek to adjust the surface once it is formed, may distort the surface considerably. In some such algorithms, extra points will be added in order to interpolate between the points already in place. Also, the goal of the surface is to entirely enclose the points “inside” the object, and to exclude the points “outside” the object. Smoothing algorithms which work after the surface is defined will have no knowledge of the original data set. Thus, they may shift the points to violate the correct point inclusion of the surface, where “inside” points are/or outside, and “outside” points are inside.

Hence, we will now look into a way of producing smoother results, during the creation of the original mesh. Thus, we will be able to maintain the correct point inclusion.

Keep in mind throughout this discussion, that these results can be implemented in both 2D, and 3D versions. In many cases, the 2D version is considerably simpler; however, we are more interested in the 3D case, due to its inherent complexity, and its relationship to the Marching Cubes algorithm.

CHAPTER 2

PREVIOUS WORK ON SMOOTHING

Given a set of data points in space and their associated densities, there are a variety of ways to produce a boundary surface. Here we will concentrate on the Marching Cubes, Dual Marching Cubes, and Surface Nets algorithms, which all use somewhat similar approaches in creating a surface. In each algorithm, there is the idea of a "voxel", which is some unit of volume. For the algorithms presented in this chapter, all use a cube shaped voxel.

In the Marching Cubes algorithm, we are given a rectilinear grid of data (Lorensen and Cline, 164).

We can define the points of this grid as

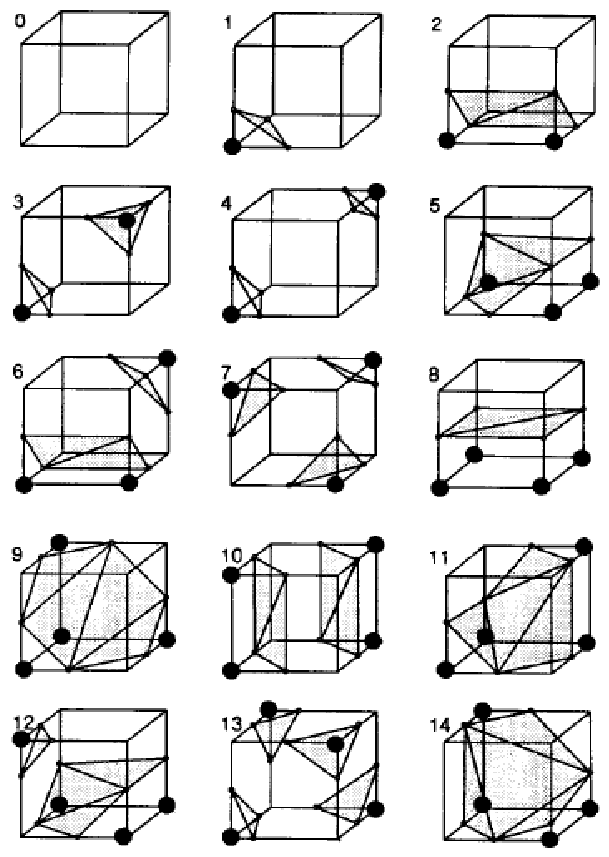
$F_{i,j,j} = F(i\Delta x, j\Delta y, k\Delta z)$, where $\Delta x, \Delta y, \Delta z$ are positive lengths of the sides of the cube.

So our rectilinear grid would be defined by

$$L = \{(i\Delta x, j\Delta y, k\Delta z),\ i = 1, \dots, N_x; j = 1, \dots, N_y, k = 1, \dots, N_z\}.$$

Furthermore, we can define our cubes with the diagonal from $(i\Delta x, j\Delta y, k\Delta z)$ to $((i+1)\Delta x, (j+1)\Delta y, (k+1)\Delta z)$ (Nielson, "Dual Marching Cubes", 490).

Then, given some threshold value, we specify each vertex of the cube as either "inside" or "outside" the surface. Then, for each cube, we create our surface itself by using a lookup table to create surface elements, and connecting them to the surface elements of adjacent cubes. (Lorensen and Cline, 165).



Each cube shaped voxel may have points which are "inside" or "outside" of an object. Depending on which set the vertices of the cube belong to, we may create a polygonal surface separating those vertices. While there are considerably more cases than those shown above, we may generalize all of the cases into the 14 above.

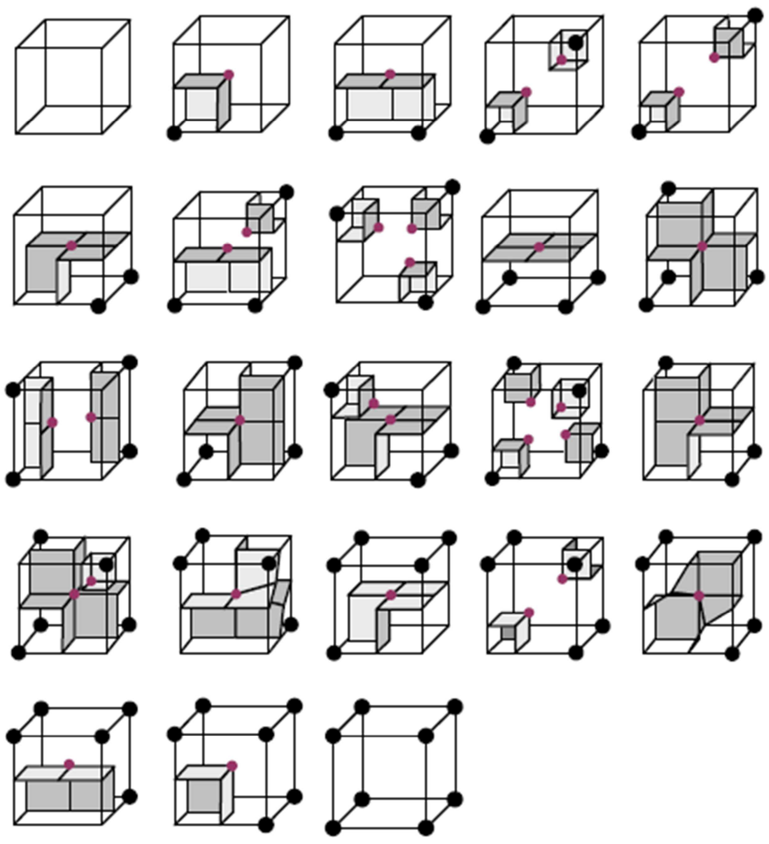
Marching Cubes surface lookup table (Lorensen and Cline, 165). (Figure 1)

The lookup table provides some configuration of polygons which separate the "inside" and "outside" vertices for a cube. So, for each cube, a configuration is selected from the table. Once a configuration is selected, its geometry is compared to the geometry of the adjacent cubes, which allows the geometry to be connected. Once complete, this will then provide a polygonal surface separating the two sets of vertices.

This process was modified by the Dual Marching Cubes algorithm to use surface patches. The surface patches themselves are left somewhat undefined, except that each surface patch is not necessarily polygonal. That allows for greater flexibility in its construction. According to the original paper, surface patches are created using the following definition:

> **Definition**: Let S be in [the collection of surfaces for the Marching Cubes], then S^* is a surface comprised of a collection of quad patches with the following properties
>
> 1) For each patch F_j of S there is a vertex Q_j of the dual surface, S^*, lying in the interior of the voxel containing F_j.
>
> 2) For every vertex V_i, of the marching cubes surface, S, there is one quad patch P_i of S^*. The vertices of the quad patch are the vertices that associate with each of the four patches of S that have V_i in common.
>
> 3) For every edge of S there is an associated edge of S^*. The edge of S lies in the voxel face intersected by the associated edge of the dual surface S^* (Nielson, “Dual Marching Cubes”, 491).

This leads to a lookup table similar to the one used by Marching Cubes.

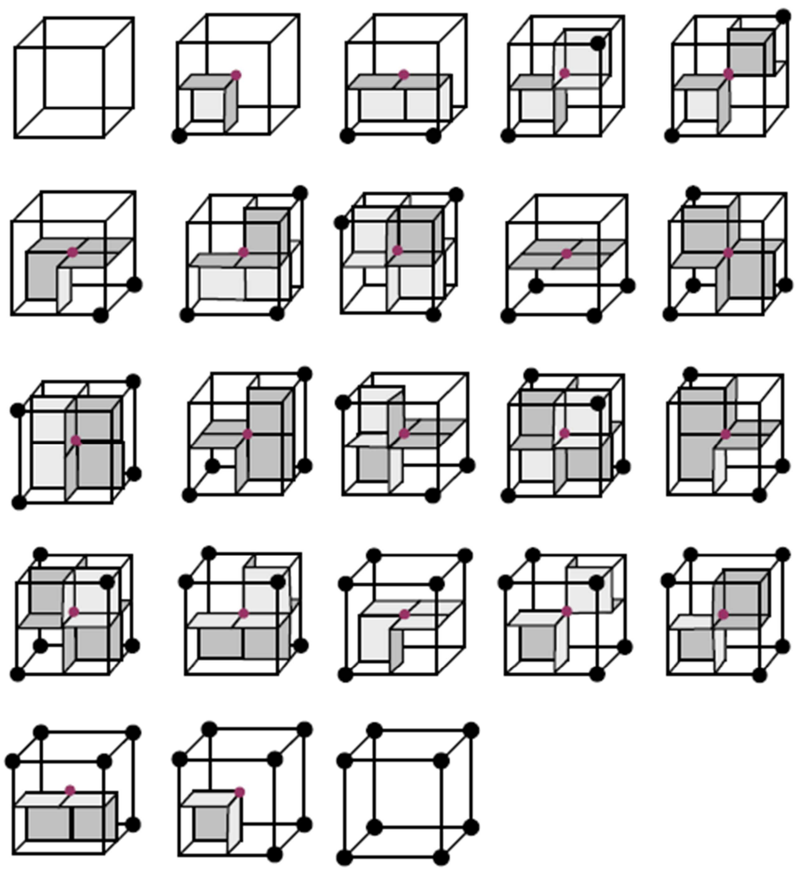


Here we show the patch based lookup table used in the Dual Marching Cubes algorithm.

Dual Marching Cubes lookup table (Nielson, "Dual Marching Cubes", 491). (Figure 2)

We can see that in the Marching Cubes algorithm, each cube contains 0 or more polygonal surfaces. In the Dual Marching Cubes algorithm, we instead place a vertex for each of these surfaces, and attach them with patches to the adjacent neighbors.

However, in the Dual Marching Cubes paper, it is shown that the DMC can be modified to use a lookup table which includes only one vertex per voxel. This is also known as the Cuberille Method (Chen et al).



Here we show a modified form of the lookup table for the Dual Marching Cubes algorithm. In thise case, instead of a potential maximum of four vertices per voxel, we unify the vertices to create a single vertex per voxel. This is also used in the Cuberille method.

DMC Lookup Table using one vertex per voxel (Nielson, "Dual Marching Cubes", 496). (Figure 3)

This alternate lookup table uses edges which are a subset of the Surface Nets method (Nielson, "Dual Marching Cubes", 496), which also use a single vertex per cube approach.

The Surface Nets method is used to produce a smooth boundary. It is a modified form of Laplacian smoothing, which is a well established method of smoothing geometric data (Taubin, 1). In the case of Laplacian smoothing, each vertex in a mesh is moved to the barycenter of its neighboring vertices. However, there is significant shape distortion, and after several iterations, the mesh structure will eventually shrink down to a single point (Taubin, 1). Taubin does introduce several methods to counteract the shrinkage, however none take the underlying data points into consideration.

Surface Nets on the other hand are designed to address the underlying data points, while applying the principles of Laplacian smoothing. Similar to the previous algorithms, cubes are used once again (Gibson, 892). Again each data point is classified as “inside” or “outside” of an object, in a process which is defined as “Segmentation”. The cubes are then determined to be either “entirely inside”, “entirely outside” or “on the surface” of the object, based on being made up of purely “inside” points, purely “outside” points, or a mixture of the two, respectively. Then, for every surface cube, a vertex is placed in the center, and connected to the vertex of the adjacent surface cubes. The surface itself is defined identically to the lookup table shown above. The Surface Net may now be “relaxed”, by moving the vertex of each cube to a position equi-distant from its neighbors (Gibson, 893).

This method differs distinctly from the Laplacian method by one constraint. Each vertex must be constrained to the inside of its original surface cube. The surface will thus be faithful to the underlying segmentation, and will favor the segmentation, rather than overall smoothness. Furthermore, it will create sharp areas where the underlying data specifies it (Gibson, 894).

The method defined in this work draws considerable inspiration from the Surface Nets method.

CHAPTER 3

DUAL MARCHING TETRAHEDRA ALGORITHM

Let's take a look at the Dual Marching Tetrahedron method of mesh generation. We will show both the 2D and 3D versions, as both may be extended with a smoothing algorithm. Again, this is based on the Cuberille Method, which uses nearly identical logic.

2.1 2D Algorithm

We will take in some set of 2 dimensional points, each assigned some density. We are attempting to build a boundary between “inside”, and “outside” points. The boundary will be a line, which we will then smooth (Nielson, “Dual Marching Tetrahedra”, 188).

Let's call the points $p_k = (x_k,\ y_k,\ d_k)$ where $x_k,\ y_k \in \mathbb{R}$ and $d_k \in \mathbb{R}^+$. We can group the points into the set $P = \{p_k\ :\ k = 1,\ \dots,\ n\}$. In order to build our mesh, we have to run several algorithms:

a.) Segmentation Algorithm

In the original Marching Cubes algorithm, the points were split into two sets, based on the density of each element: those inside and those outside the object in question. While the DMT algorithm can work in a similar fashion, there is an opportunity to account for an ambiguous case, where the density is unclear as to whether or not the point should be placed in or out of the object.

Here we split P into three separate sets, based the density of each element.

Specifically, the sets are P_i, P_{out}, and $P_{unknown}$, which are for points inside the object, points outside the object, and points which we are unable to determine their inclusion in the object, respectively. The proposed segmentation algorithm uses two selected threshold values: t_i and t_{out}. Without loss of generality, points density greater than or equal to$t_i$are placed in$P_i$and points with density less than or equal to t_{out} are placed in P_{out}. All points with density between t_i and t_{out} are placed in $P_{unknown}$. Although it is beyond the scope of this paper, it is possible to define another threshold, of points which are too dense to be inside, and put them also into the "outside" set.

We will not consider the points in the "unknown" case, and so we can now define $P_{valid} = P_i \cup P_{out}$, which thus only stores the "inside" and "outside" points.

This algorithm can be easily implemented by iterating through P, and classifying each p_k as "inside", "outside", or "unknown", and then either deleting the "unknown" points, or entering the "inside" and "outside" cases into some new P_{valid} set. However, specific implementations may vary significantly as far as data structures are concerned, and so for the rest of this discussion, we will continue to use the notation and conventions already in place.

b.) Triangulation

Now that we have divided our points into appropriate sets, we will create a 2D mesh of triangles which connect all of the points in P_{valid}. These units of area help to divide up the surface for our boundary. We can represent a each triangle as a 3-tuple of points from P_{valid}. For a triangle defined by points $p_i,\ p_j,\ p_k$, we write $T_{i,j,k}$. Furthermore, we define the union of the volumes as a surface, *I*, which is subject to a few constraints:

i.) No $T_{i,j,k} \in I$ is degenerate, meaning $p_i,\ p_j,\ p_k$ are non-collinear.

ii.) The interiors of any two triangles in *I* do not intersect.

iii.) Two triangles in *I* only intersect at a common edge.

This kind of triangulation may be achieved through a Delaunay Triangulation (Nielson, “Dual Marching Tetrahedra”, 186). In the 2D case, the Delaunay Triangulation is defined to be a set of triangles connecting a set of points, such that the circumcircle of any triangle contains no points (Okabe et al., 94). There are a variety of algorithms available for constructing such a structure (Nielson, “Tools for Triangulation”).

c.) Building the Boundary

Now that we have defined our triangles, we will actually build the boundary.

i.) For each $T_{i,j,k} \in I$, we define it as active if and only if at least one of its vertices is in$P_i$and at least one is in P_{out}. So let $I_{active} \subseteq I$ such that, if $T_{i,j,k} \in I$ is active, then $T_{i,j,k} \in I_{active}$.

ii.) For each triangle, $T_{i,j,k} \in I_{active}$ we define a center point $v_{i,j,k} \in V$. There is no specific requirement for the location of $v_{i,j,k}$, other than that it is inside of $T_{i,j,k}$. For our purposes, at this point we may simply set $v_{i,j,k}$ as the centroid of $T_{i,j,k}$.

Note: This placement is guaranteed to put the vertex inside of the triangle.

iii.) We now will analyze which triangles are adjacent to one another via common edges. Let's define an edge as two points of a triangle, $p_i,\ p_j$, which we write as $D_{i,j}$. Let *J* be the set of all triangle edges, such that $D_{i,j} \in J$. Furthermore, we define an edge as active if and only if at least one of its vertices is in P_i and at least one is in P_{out}. So let $J_{active} \subseteq J$ such that, if $D_{i,j} \in J$ is active, then $D_{i,j} \in J_{active}$. We define two triangles as adjacent if they share an edge.

iv.) Furthermore, we will now define an edge $e_{i,j,k} \in E$, which connects the center points of the two triangles which are adjacent across edge $\mathrm{D}_{i,j} \in J_{active}$.

We now have a boundary defined by edges which separate the “inside” and “outside” points.

2.2 3D Algorithm

As stated before, we will take in some set of 3 dimensional points, each assigned some density.

We can simply extend the definition of our points 3D. Let's call these points $p_k = (x_k,\ y_k,\ z_k,\ d_k)$ where $x_k,\ y_k,\ z_k \in \mathbb{R}$ and $d_k \in \mathbb{R}^+$. We can group the points into the set $P = \{p_k : k = 1,\ \ldots,\ n\}$. Again, in order to build our mesh, we have to run several algorithms:

a.) Segmentation Algorithm

This portion of the algorithm is identical to the 2D version.

b.) Tetrahedrization

Here, since we can no longer constrain our points to a planar surface, we build our "voxels", or units of volume. For our purposes, we will use tetrahedra to define these volumes, which we can represent as a 4-tuple of points from P_{valid}. For a tetrahedron defined by points $p_i,\ p_j,\ p_k,\ p_l$, we write $T_{i,j,k,l}$. Furthermore, we define the union of the volumes as a tetrahedronal *I* which is subject to a few constraints:

i.) No $T_{i,j,k,l} \in I$ is degenerate, meaning $p_i,\ p_j,\ p_k,\ p_l$ are non-coplanar.

ii.) The interiors of any two tetrahedra in *I* do not intersect.

iii.) Two tetrahedra in *I* only intersect at a common triangular face.

Again, a Delaunay Triangulation is used, but is extended to the 3D case using circumspheres (Nielson, “Tools for Triangulations”). Similarly, there are a variety of methods to find a Delaunay Tetrahedrization (Nielson, “Tools for Triangulations”), (Blandford, Blelloch, Kadow), (Field), (Hoshiko, Kawahara).

c.) Building Mesh

Now that we have defined our volumes, we will actually build the “skin” of the mesh.

i.) For each $T_{i,j,k,l} \in I$, we define it as active if and only if at least one of its vertices is inP_{in} and at least one is in P_{out}. So let $I_{active} \subseteq I$ such that, if $T_{i,j,k,l} \in I$ is active, then $T_{i,j,k,l} \in I_{active}$.

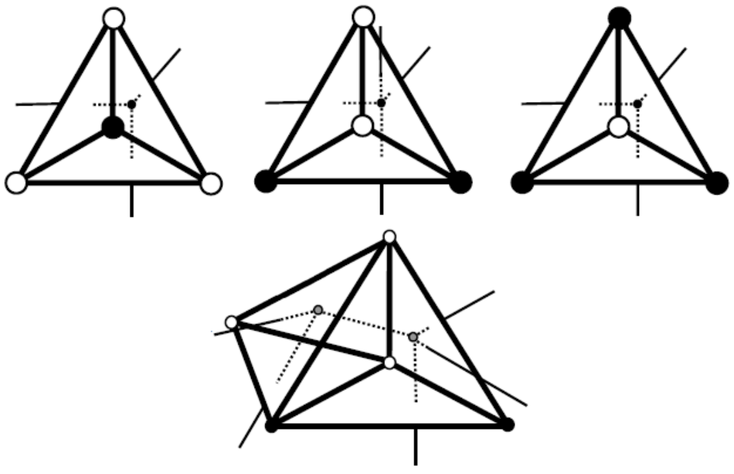
ii.) For each tetrahedron, $T_{i,j,k,l} \in I_{active}$ we define a center point $v_{i,j,k,l}$. There is no specific requirement for the location of $v_{i,j,k,l}$, other than that it is inside of $T_{i,j,k,l}$. For our purposes, at this point we may simply set $v_{i,j,k,l}$ as the centroid of $T_{i,j,k,l}$.

Note: This placement is guaranteed to put the vertex inside of the tetrahedron.

iii.) We now will analyze which tetrahedra are adjacent to one another via common faces. Let's define a face as three points of a tetrahedron, $p_i,\ p_j,\ p_k$, which we write as $F_{i,j,k}$. Let *J* be the set of all tetrahedron faces, such that $F_{i,j,k} \in J$. Furthermore, we define a face as active if and only if at least one of its vertices is in P_{in} and at least one is in P_{out}. So let

$J_{active} \subseteq J$ such that, if $F_{i,j,k} \in J$ is active, then $F_{i,j,k} \in J_{active}$. We define two tetrahedra as adjacent if they share a face.

iv.) Furthermore, we will now define an edge $e_{i,j,k} \in E$, which connects the center points of the two tetrahedra which are adjacent along face $F_{i,j,k} \in J_{active}$.



The three tetrahedra at the top of figure represent the lookup table for the Dual Marching Tetrahedra method. The lower portion shows how two adjacent tetrahedra might connect.

The three cases of DMT in 3D (Nielson, "Dual Marching Tetrahedra"). (Figure 4)

v.) We now must connect the edges together to form the polygons of our final mesh. We can use a similar lookup table to the one used in the Cuberille Method. For each pair of edges leaving $v_{i,j,k,l}$, there are two tetrahedron through which the edges pass. These faces share a tetrahedron edge, which in turn is adjacent to two tetrahedron points. If one of these points is in P_{in}, and one is in P_{out}, we may create a polygon. Now that we have our polygons constructed, we have our final mesh.

CHAPTER 4

CRITERION FOR SMOOTHNESS

While we are seeking to represent smooth analog objects, our data set is of discrete points of density. Thus, even though the resulting mesh of the DMT algorithm has produced a correct enclosing structure, it will not necessarily produce a smooth result. Keeping in mind that we are dealing with polygons and edges, we cannot hope for a continuous result. While a continuous surface has simple measures for smoothness based on curvature, polygons and edges are not so well defined. So, let's take a look at a few ways to define smoothness of such objects.

It is important to note that we define smoothness here so that we have a way to test whether a smoothing algorithm has produced a favorable result. While the human eye may be able to detect that a particular example “looks smoother” than another, ultimately this isn't a reliable measure. Keep in mind that we actually do not use this criterion as a part of our smoothing algorithm itself. We merely use it to verify our results.

3.1 2D Criteria

The 2D case is significantly simpler than the 3D case. The primary idea is to analyze the angles surrounding a vertex in the boundary. For our purposes, we are only looking at each angle individually. Future work could be done to analyze several adjacent angles at once, which would provide a more global smoothness estimate. However, that is outside the scope of this discussion.

There are several tests which we can use. Here we will present two.

a.) Angle Analysis

In a continuous and differentiable line, the angle at any point is always 180°. So, given any subset of points on such a line, the average angle would always be 180°. We can do a similar process for connected line segments, which are connected by vertices. For each vertex connecting two line segments, we have an angle, and so the average of all such angles can be found. We should always select the angle which is 180° or less, also known as the "non-reflex" angle. Since we always selected the non-reflex angle, the maximum average is 180°. So, the closer to 180°, the smoother the result.

Thus, we can define smoothness on a mesh $M = (V, E)$, to be:

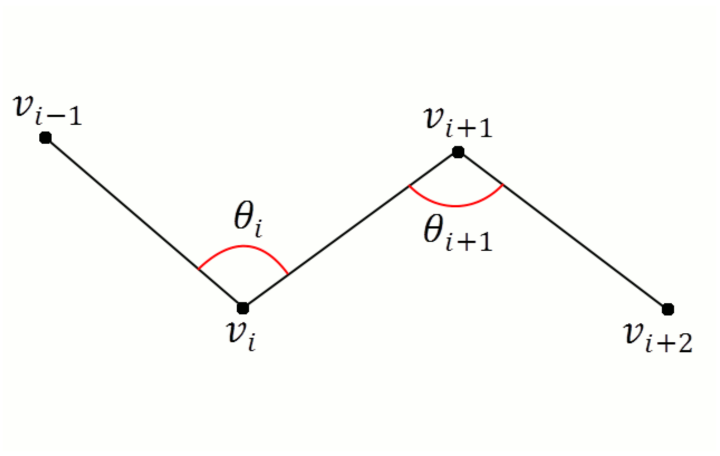
Let V be the set of vertices, and E be the set of edges.

Then $E_v \subset E$ is the set of edges adjacent to $v \in V$.

Then $V' = \{v \mid \forall v \ \in V \ where \ E_v = 2\}$.

Then $smoothness(M) = \frac{\sum_{v \in V'} c(v)}{|V|}$,

where $c(v) = \cos^{-1}\left(\frac{e_0 \cdot e_1}{|e_0||e_1|}\right)$ and $e_0, e_1 \in E_v$.



2D Angle Criterion (Figure 5)

Note that our average is only meaningful if we consistently select the non-reflex angle in our computation. (We may also always select the reflex angle, but for sake of convention, the non-reflex angle is more convenient.) For example, let's say we're averaging the angles of two vertices, both measuring $170°$. The resulting average would then be $170°$. However, let's say that for one we select the non-reflex angle of $170°$and for the other we select the reflex angle of $190°$. The average would then be $180°$, even though our example is clearly not smooth. Thus, we must consistently select the non-reflex angle.

b.) Vector Analysis

This method is similar to the smoothing algorithm itself. It is a more complex method, and in the 2D case does not necessarily give us more information than the previous method. However, due to its relationship to the smoothing algorithm, it is useful to present.

If we look at a vertex with two incident edges, instead of analyzing the angle itself, we can analyze the two edges. Let's say we have a vertex v, and two edges e_1, e_2. We can then construct two vectors of unit length from v along the edges, which specifies two points q_1, q_2. We can then find the midpoint of these points, which is q. We are interested in the distance from v to q. If the two edges are collinear in opposite directions (180°), then clearly the distance will be 0. However, if the two are at any other angle, the distance will be $0 < distance \leq 1$. So, we can see that a smoother boundary will result in a number closer to 0.

Thus, we can define smoothness on a mesh $M = (V, E)$, to be:

Let V be the set of vertices, and E be the set of edges.

Then $E_v \subset E$ is the set of edges adjacent to $v \in V$.

Then $V' = \{v \mid \forall v \ \in V \ where \ |E_v| = 2\}$.

Then, given $v \ \in V'$, we have $E_v = \ \{e_0, e_1\}$.

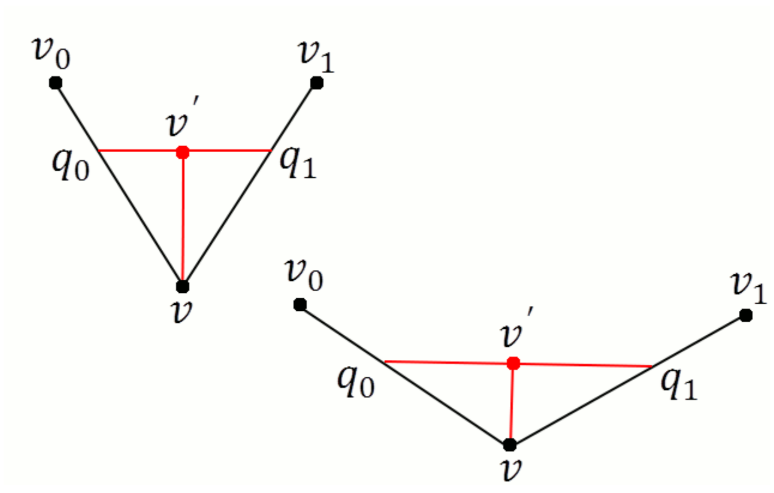
Then $e_0 = (v, v_0)$, $e_1 = (v, v_1)$ where $v_0, v_1 \in V$.

Then let $\widehat{e_0} = \frac{vv_0}{|vv_0|}$, and $\widehat{e_1} = \frac{vv_1}{|vv_1|}$, which are unit length vectors along the adjacent edges.

Use $\widehat{e_0}, \widehat{e_1}$ to create points p_0, p_1.

Then let v' be the midpoint between p_0, p_1.

Then $smoothness(M) = \frac{\sum_{v \in V'} c(v)}{|V|}$, where $c(v) = |vv'|$.



2D Vector Criterion (Figure 6)

c.) Energy Criterion

This method is similar to the 3D method defined by Gibson for Surface Nets (Gibson, 893). It is based on the idea that smoother surfaces will have less surface area. Then, each edge is given an energy, which in this case is equal to the square its length. Then the sum of all of the energies is produced, which provides a rough idea of the surface area of the Surface Net. Here a smoother smaller result means a smoother surface.

It is important to note that the result of this criterion is not scale invariant. If a mesh were scaled to a larger size with no other modifications, it would generate a larger result. For Laplacian smoothing, this would pose a significant problem as the mesh would shrink. However, for an algorithm based on Surface Nets, shrinkage is not a problem, and so this criterion is applicable.

Given a mesh $M = (V, E)$, where V is the set of vertices, E is the set of edges.

Then for each $e \in E$, $e = (v_0, v_1)$, where $v_0, v_1 \in V$.

Then $smoothness(M) = \sum_{e \in E} c(e)$, where $c(e) = |v_0 v_1|^2$.

3.2 3D Criteria

The 3D case is not nearly as simple. At each vertex, we may have between two and four converging faces and edges. Unlike the 2D example, we cannot simply test for the angles converging on a vertex, for instead of dealing with edges alone, we are now dealing with surfaces. There are undoubtedly multiple ways of testing this kind of smoothness, but here we will present two such methods.

a.) Vector Analysis

This method is inspired by the smoothing algorithm which will be presented later. For each vertex, we will have two to four adjacent edges. For each adjacent edge, we may define a point unit distance from the vertex. We then take the centroid of those new points, and have our new point C. If this surface is perfectly flat, C will be at the same location as our vertex.

This method is not particularly meaningful for two edges. For three we can certainly find whether or not a concave surface is defined. For four edges however, we will have a meaningful result if the surface is concave, but the surface may also describe a saddle point. If we have a saddle point, it is possible for C to be located at the vertex even if the geometry is by no means flat.

Just as with the 2D case, a perfectly smooth vertex will result in a distance of 0, and all others will be in the range of $0 \leq distance \leq 1$. So the smoother the surface, the closer the distance will be to 0.

Thus, we can define smoothness on a mesh $M = (V, E, P)$, to be:

Let V be the set of vertices, E be the set of edges, and P be the set of polygons.

Then $E_v \subset E$ is the set of edges adjacent to $v \in V$.

Then $V' = \{v \mid \forall v \in V \; where \; |E_v| \geq 2\}$.

Then, given $v \in V'$, we have $E_v = \{e_0, \dots, e_n\}$.

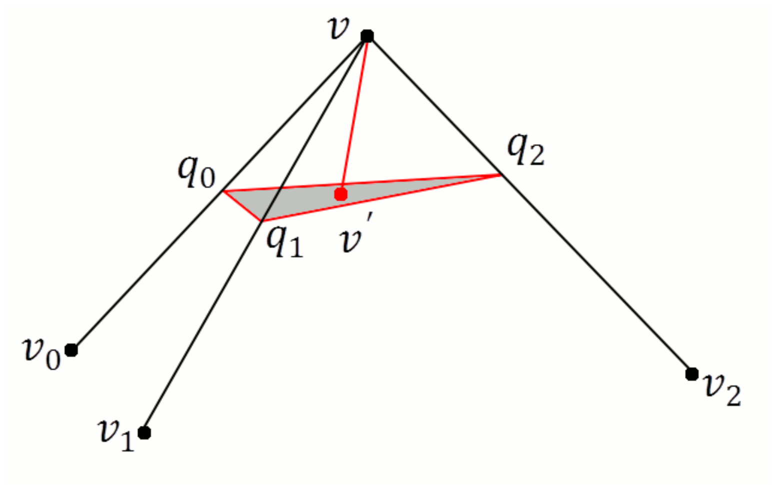
Then $e_i = (v, v_i)$, where $v_i \in V$.

Then let $\widehat{e_i} = \frac{vv_i}{|vv_i|}$, which is a unit length vector along the adjacent edge of $v \in V'$.

Then let v' be the centroid of $\widehat{e_0}, \dots, \widehat{e_n}$, where each vector starts from the position of v.

Then $smoothness(M) = \frac{\sum_{v \in V'} c(v)}{|V|}$, where $c(v) = |vv'|$.

Note, this criterion is very similar to the smoothing algorithm, and is useful to present.



3D Vector Criterion (Figure 7)

b.) Energy Criterion

This method is defined by Gibson for Surface Nets. It is essentially the same as the 2D version (Gibson, 893). Again, a smoother smaller result means a smoother surface.

Given a mesh $M = (V, E, P)$, where V is the set of vertices, E is the set of edges, and P is the set of polygons.

Then for each $e \in E$, $e = (v_0, v_1)$, where $v_0, v_1 \in V$.

Then $smoothness(M) = \sum_{e \in E} c(e)$, where $c(e) = |v_0 v_1|^2$.

CHAPTER 5

SMOOTHING ALGORITHM

5.1 Developing Algorithm

Note, that in both the 2D and 3D case, we are testing the concavity at every vertex. The non-reflex angle between lines is simply a measure of how concave the shape is. We will deal with that concept as we motivate our algorithm.

The mesh built by the DMT is created specifically to contain all of the points of P_{in} and none of P_{out}. As mentioned before, if we were to smooth our mesh once it is build by the DMT, ignoring the data structures and methods used to construct the mesh, we may change shape in such a way which violates the correct point inclusion of the mesh. If we violate the point inclusion, then we do not necessarily correctly represent the object which we sought to visualize. Thus, we need some smoothing technique which modifies our mesh, while keeping the original data in mind.

First, it is important to note that DMT has no specific requirement for the location of $v_{i,j,k,l}$, other than that it must exist inside of $T_{i,j,k,l}$. As long as $v_{i,j,k,l}$ remains inside $T_{i,j,k,l}$, we can clearly see that if the mesh was constructed correctly, we will contain P_{in} and exclude P_{out}. Thus the vertex may be relocated to any place in interior of the tetrahedron.

In our description above, we blindly chose the centroid as the location in each tetrahedron, since it is guaranteed to be on the inside. However, our algorithm will seek to find a location for each $v_{i,j,k,l}$, which provides a smoother mesh as per our smoothness criterion.

We started with the “Surface Nets” method briefly mentioned in the original DMT paper. As mentioned before, the Surface Nets algorithm very similar to the Cuberille method, upon which DMT is based. For the original Surface Nets, all of the voxels are cube shaped, just as they are in the Cuberille. So, we can similarly change the cubes of Surface Nets to tetrahedra. Just as with the Cuberille method, DMT has a node inside every surface tetrahedron, which is connected by an edge to the vertices of its neighboring surface tetrahedra. Thus, our DMT may be taken through the same relaxing processes that the original Surface Nets undergo.

5.2 Statement of Algorithm

Just as we may define DMT for both the 2D and 3D cases, we may define our smoothing algorithm for both as well. In the definition of the 3D case below, simply replace the word tetrahedron with triangle for the 2D case.

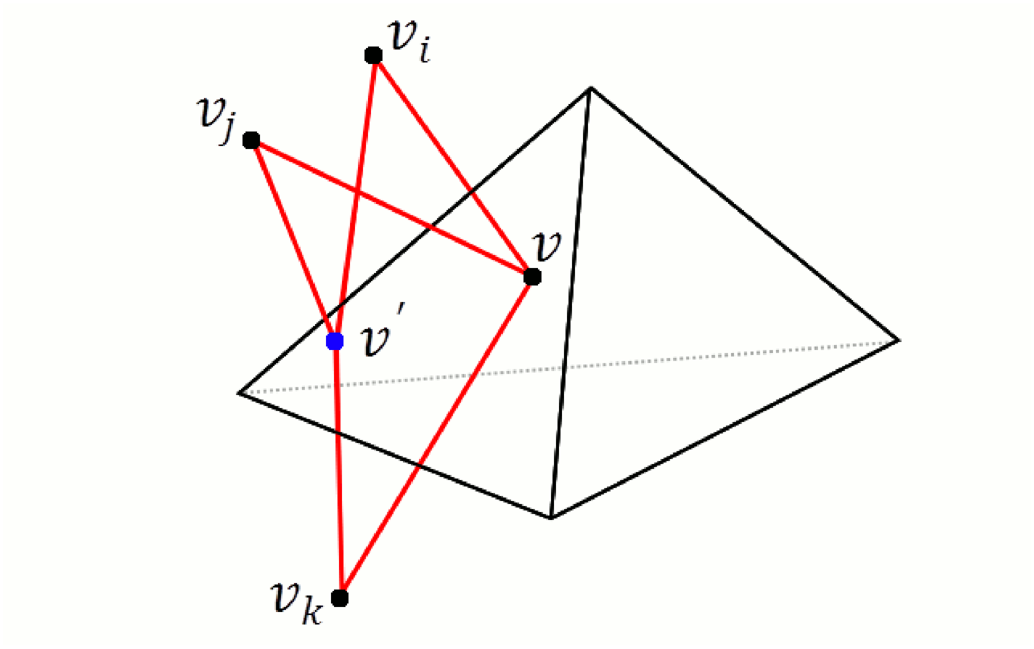
For each tetrahedron $\in I_{active}$, there exists a center vertex v.

Then $V_v \subset V$ is the set of vertices adjacent to $v \in V$.

Then let v' be the centroid of the vertices in V_v.

Then move v to the location of v'.

The new location of v may now be outside of the tetrahedron. If so, it will have to be moved to the location closest to v', such that it is still inside the tetrahedron.



Here is a tetrahedron containing vertex v, which is adjacent to three vertices, v_i, v_j, and v_k. It is then relocated to the centroid of its neighbors, v'.

Smoothing Algorithm (Figure 8)

2D Triangle Bounding.

If the location of $v'$is outside of triangle T, we will have to test it against each side of T to see if it is either "outside" or "on" the side of T, in which case we can say that it "violates" the side.

Case 1: If we violate only one side, we simply project v' onto that side.

Case 2: If we violate two sides, we simply set v' to the intersection of the two sides.

There is a possibility that Case 1 will actually cause Case 2, since projecting to a single side may violate the another side. However, since we projected the point to a location “on” the first side, it now violates two sides. So we may simply run the bounding a second time, and which will project to a point on the triangle, and solve the issue.

3D Tetrahedron Bounding

The 3D case may be handled similarly. If the location of $v'$is outside of tetrahedron T, we will have to test it against each face of T to see if it is either “outside” or “on” the face of T, in which case we can say that it “violates” the face.

Case 1: If we violate only one face, we simply project v' onto that face.

Case 2: If we violate two faces, we must fine the line where the two faces intersect. Then we project v' onto that line.

Case 3: If we violate three faces, we may simply project v' onto the intersection of the three faces.

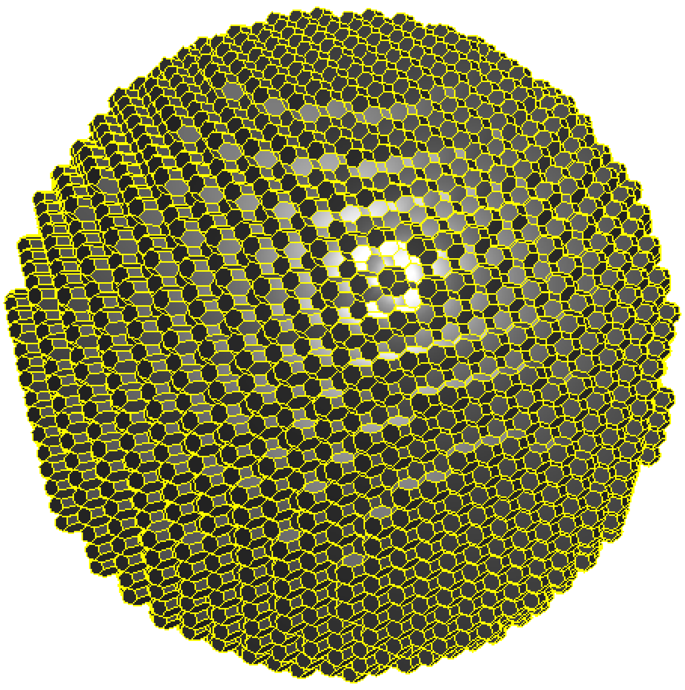
There is a possibility that Case 1 or Case 2 will actually cause Case 2 or Case 3, since projecting to due to one or two violations, may cause violations of other faces. However, since we projected the point to a location “on” the first ace, it still violates that face. So we may simply run the bounding a second time, which will pick up all possible violations, and solve the issue.

CHAPTER 6

DISCUSSION

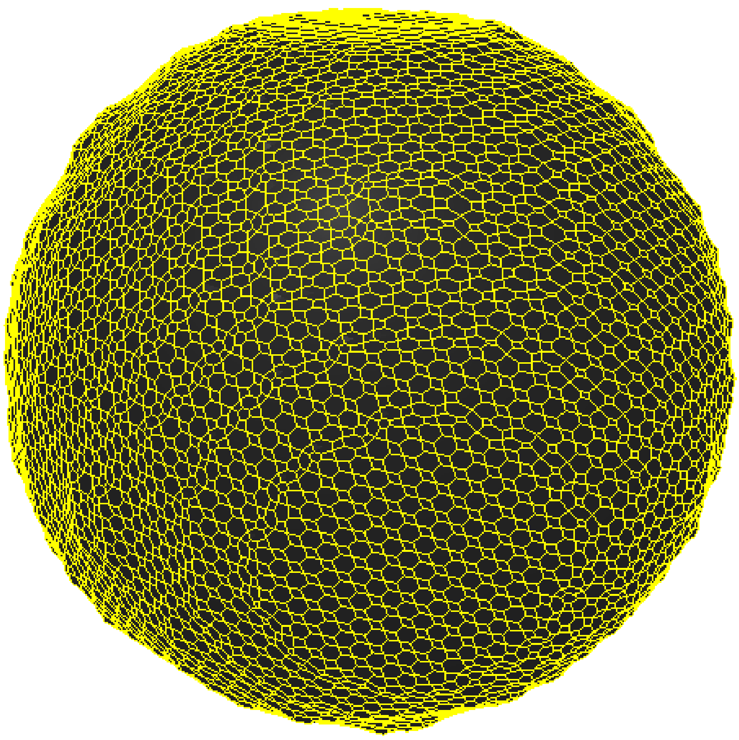
Now that we have our algorithm in place, we can run our smoothness criterion to analyze the result. Drawing inspiration from the Surface Nets algorithm, we may repeat the smoothing process more than one time. Each time the algorithm is run, it will create a smoother result, tending towards some maximum smoothness.

We should note that the maximum smoothness is not necessarily a number which is perfectly smooth by our criterion. If the data set itself is very smooth, say in a high resolution scanning of a circle or sphere, the resulting surface will tend towards a smooth number.



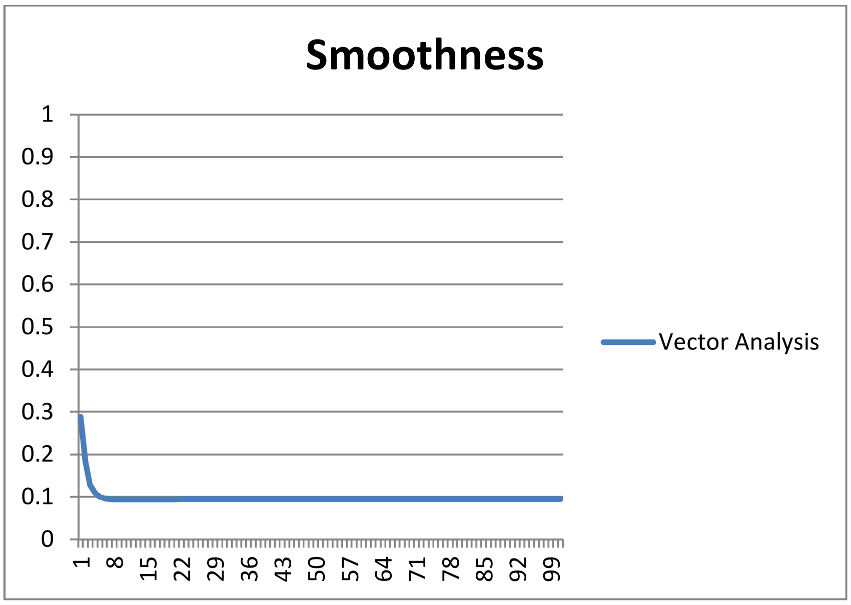
Here is surface obtained from discrete points arranged in a sphere. The grid of data is 32 x 32 x 32.

High Resolution Mesh, smoothed 0 times (Figure 9)



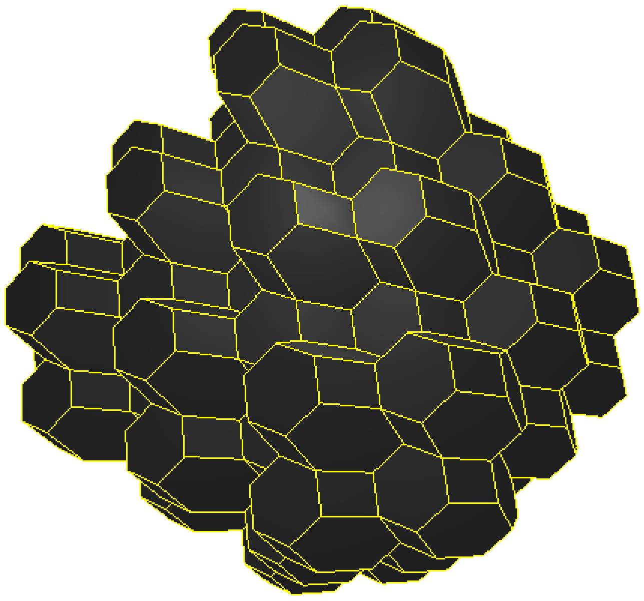
Here we see the same data set, smoothed 100 times. It is visually much smoother, except for some sharp points which can be attributed to the surface settling onto the sharp points of the data set itself.

High Resolution Mesh, smoothed 100 times (Figure 10)



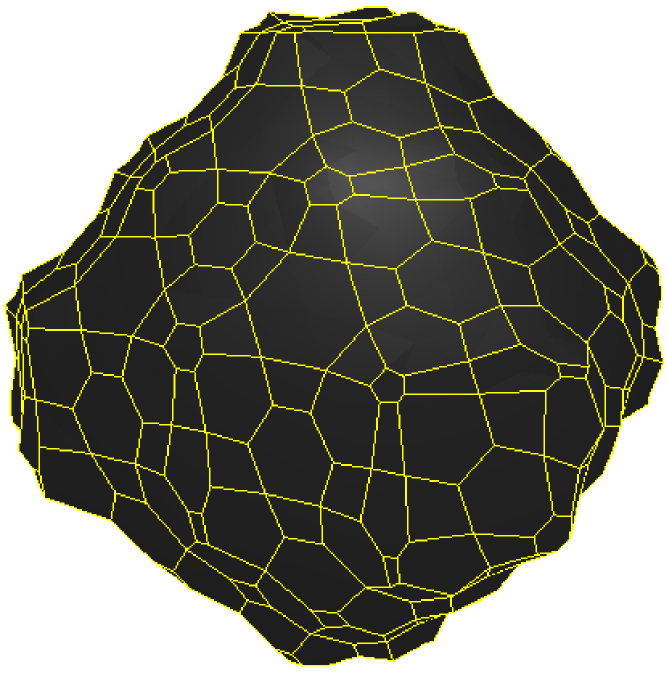
This is a graph of the smoothness criterion over 100 runs. We can see a marked decrease in the smoothness criterion. Here we used the Vector Analysis criterion. A smaller number indicates that our mesh became smoother.

High Resolution Mesh, Vector Analysis Smoothness (Figure 11)



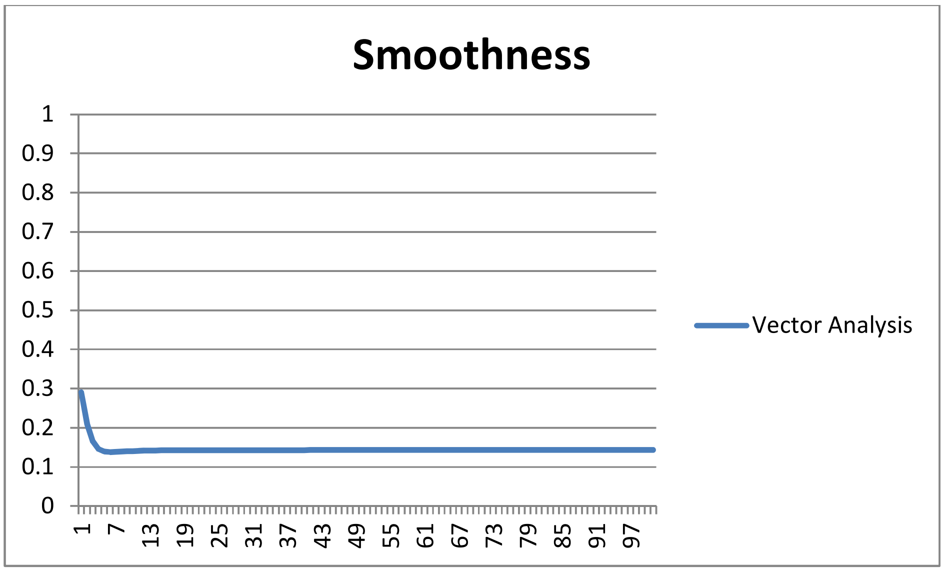
Here is a surface obtained from a lower resolution data set of a sphere. The grid of data is 8 x 8 x 8.

Low Resolution Mesh, smoothed 0 times (Figure 12)



Here is the same data set smoothed 100 times. Once again, it is visually much smoother than before. Similarly, there are sharp points where the data set specifies.

Low Resolution Mesh, smoothed 100 times (Figure 13)

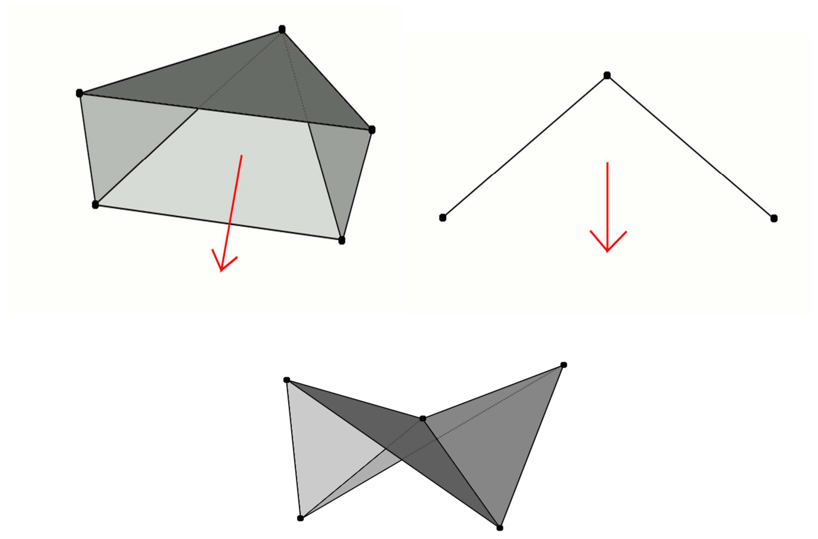


This is a graph of the smoothness criterion over 100 runs. We can again see the smoothness quickly drop, and then grow asymptotically towards some value. Here we again used the Vector Analysis criterion, and so a smaller number indicates an increase in smoothness.

Low Resolution Mesh, Vector Analysis Smoothness (Figure 14)

On the other hand, if our surface is very jagged on a per voxel basis, or simply very low resolution, then the resulting surface naturally will tend towards a much less smooth number.

To explain, we will need a few definitions. Since this algorithm works on a per vertex basis, we need to understand concavity at each vertex. Loosely borrowing definitions from Calculus, we can see that a vertex can be considered "concave inward" if its non-reflex angle is on the inside of the object. Similarly, it can be considered "concave outward" if its non-reflex angle is on the outside of the object. The 3D case works similarly, for each vertex; we may analyze the angles between its adjacent polygons. If all of the angles are non-reflex on the inside, then we can say it is "concave inward". Similarly, if all of the angles are non-reflex on the outside, we can say it is "concave outward". Of course, there are cases in 3D which are ambiguous, such has having a vertex with a “saddle point”, where two edges are concave inward, and two are concave outward.



The top left figure is concavity for a 2D boundary. The top right figure is concavity for a 3D boundary. The bottom figure is a 3D saddle point.

Concavity (Figure 15)

With these definitions in place, we can clearly see that if a vertex is concave inward, it will be moved inward. Likewise, if a vertex is concave outward, it will be moved outward. In the less defined case, the vertex will still be moved to an equi-distant location, which will produce a smoother result.

In the most trivial example, a single “inside” point, surrounded entirely by “outside” points, every single vertex of the result is concave inward. Thus, the vertices will all move inwards during each iteration, eventually reaching the tetrahedron or triangle boundaries. Rather than smoothing, the object simply shrank.

It is also interesting to note, that for the original Surface Nets, it was observed that the smoothness dropped quickly, and then rose asymptotically towards some eventual value. This was interpreted to mean that the sharp points in the eventual result were forming (Gibson, 894). We can see a similar result in the graph of our Low Resolution Mesh.

The complexity of our smoothing algorithm is *O(n)*. We loop through our *n* vertices, which have a maximum of 4 neighbors, which reduces from *O(4n)* to *O(n)*. Of course we may run our algorithm *k* times in order to get the desired result, which would then have the aggregate run at *O(nk)*, but *k* is a defined by the user at run time, and so again, it reduces to *O(n)*.

CHAPTER 7

CONCLUSION

Turning discrete data into continuous surfaces is a non-trivial process. However, when analyzing important data such as that produced by MRI and CT scanners, the clarity of the resulting surface is paramount. Similar methods have existed for earlier algorithms such as Marching Cubes, and Dual Marching Cubes. However, drawing inspiration from the Surface Nets algorithm, we have the ability to produce much smoother results with the Dual Marching Tetrahedra algorithm.

There is some future work to be done incorporating the densities of the mass points from the underlying data set into the smoothing algorithm. When we segment the data points, we normally specify a range of densities where our boundary might lie. Inside this range, we can select a specific density for our boundary. Then, when we analyze the individual tetrahedra, we can interpolate between the densities of the four data points, in order to find some location where the specified density exists. Thus, our vertex should be placed in that location. Now, the vertex is placed according to the underlying data set, and so it is more likely to pick up the characteristics of the original scanned object, whether this it is smooth or jagged. However, this location is ignorant of any smoothing we might wish to perform, and so the results will not necessarily be as smooth as the smoothed version. We can try and unify the two approaches, by finding some blending between the smoothed position, and the weighted position.

REFERENCES

Blandford, D. K., G. E. Blelloch, and C. Kadow. "Engineering a Compact Parallel Delaunay Algorithm in 3D." *Proceedings of the Twenty-second Annual Symposium on Computational Geometry* (2006): 292-300. Print.

Chen, Lih S., Gabor T. Herman, R. Reynolds, and Jayaram Udupa. "Surface Shading in the Cuberille Environment." *IEEE Computer Graphics and Applications* 5.12 (1985): 33-42. Print.

Dyn, N., K. Hormann, S. J. Kim, and D. Levin. "Optimizing 3D Triangulations Using Discrete Curvature Analysis." *Mathematical Methods for Curves and Surfaces: Oslo* 2001 (2000): 135-46. Print.

Field, D. A. "Implementing Watson's Algorithm in Three Dimensions." *Proceedings of the Second Annual Symposium on Computational Geometry* (1986): 246-59. Print.

Gibson, Sarah. "Constrained Elastic Surface Nets: Generating Smooth Surfaces from Binary Segmented Data." Ed. William Wells, Alan Colchester, and Scott Delp. *Medical Image Computing and Computer-Assisted Interventation — MICCAI'98* 1496 (1998): 888-98. Print.

Hoshiko, R., and M. Kawahara. *3-dimensional Mesh Generation Using the Delaunay Method*.

Kim, S. J., C. H. Kim, and D. Levin. "Surface Simplification Using a Discrete Curvature Norm." *Computers & Graphics* 26.5 (2002): 657-63. Print.

Lorensen, William E., and Harvey E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm." ACM SIGGRAPH Computer Graphics 21.4 (1987): 163-69. Print.

Nielson, G. M. "Dual Marching Cubes." *Proceedings of the Conference on Visualization '04* (2004): 489-96. Print.

Nielson, Gregory. "Dual Marching Tetrahedra: Contouring in the Tetrahedronal Environment." Ed. George Bebis, Richard Boyle, Bahram Parvin, Darko Karacin, Paolo Remagnino, Fatih Porikli, Jörg Peters, James Klosowski, Laura Arns, Yu Chun, Theresa-Marie Rhyne, and Laura Monroe. Advances in Visual Computing 5358 (2008): 183-94. Print.

Nielson, Gregory M. "Tools for Triangulations and Tetrahedrizations." *Scientific Visualization: Overviews, Methodologies, and Techniques*. Los Alamitos, CA: IEEE Computer Society, 1997. 429-525. Print.

Okabe, Atsuyuki, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Chichester: Wiley, 2000. Print.

Taubin, G. "Geometric Signal Processing on Polygonal Meshes." *Eurographics State of the Art Reports* 4.3 (2000). Print.