# Delaunay Meshing of Isosurfaces

Tamal K. Dey Joshua A. Levine
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH, 43201, USA
{tamaldey|levinej}@cse.ohio-state.edu

## Abstract

*We present an isosurface meshing algorithm,* DELISO, *based on the Delaunay refinement paradigm. This paradigm has been successfully applied to mesh a variety of domains with guarantees for topology, geometry, mesh gradedness, and triangle shape. A restricted Delaunay triangulation, dual of the intersection between the surface and the three dimensional Voronoi diagram, is often the main ingredient in Delaunay refinement. Computing and storing three dimensional Voronoi/Delaunay diagrams become bottlenecks for Delaunay refinement techniques since isosurface computations generally have large input datasets and output meshes. A highlight of our algorithm is that we find a simple way to recover the restricted Delaunay triangulation of the surface without computing the full 3D structure. We employ techniques for efficient ray tracing of isosurfaces to generate surface sample points, and demonstrate the effectiveness of our implementation using a variety of volume datasets.*

## 1. Introduction

In the domain of implicit functions, many significant research results have been produced in meshing of isosurfaces. One of the earliest known approaches to isosurface polygonalization is credited to Wyvill *et al.* [22] and was followed by the MARCHING CUBES algorithm of Lorensen and Cline [14]. More recently though, there has been a growing concern with designing algorithms that satisfy constraints regarding geometric closeness and topological equivalence between the output triangulation and the isosurface. Not only do we aim to produce a mesh with the same topological and geometric guarantees but also to produce a Delaunay mesh with adaptive density and bounded aspect ratio. Such a mesh is favored in many applications since these qualities lead to reducing discretization error and mesh size. Among the recent works focusing on provable algorithms for meshing surfaces [3, 4, 7, 19] some [4, 7] employ a Delaunay refinement strategy that fits our requirement.

The Delaunay refinement paradigm works on a simple principle: build the Delaunay triangulation over a set of points in the domain and then repeatedly insert additional domain points until certain criteria are satisfied. The main challenge is a proof of termination. For the case of surfaces, one guides the refinement using a subcomplex of the three dimensional Delaunay triangulation called the *restricted Delaunay triangulation*. To provide a topological guarantee between the output mesh and the input surface, a theorem by Edelsbrunner and Shah [12] regarding the *topological ball property* is applied. This theorem says that if every Voronoi face of dimension $k$ which intersects the surface does so in a topological $(k-1)$-ball, then the restricted Delaunay triangulation is homomeomorphic to the surface. Geometric guarantees for the size and shape of restricted Delaunay triangles can also be included.

Despite the benefits of using Delaunay refinement, a practical limitation remains since the computation of a three dimensional Delaunay triangulation and its restricted Delaunay triangulation can be quite expensive. In particular, the cumulative cost of repeated insertions in the 3D structures may become prohibitive. Our main contribution is a technique to reduce the burden of 3D Delaunay triangulations. We observe that the topological ball property, a key to topological guarantee, is satisfied early in the refinement process and that the bulk of the refinement is carried out to capture the geometry. Once the density of sampled points is enough for the topology of the restricted Delaunay triangulation to be correct, we can *discard the Delaunay triangulation entirely* and continue refinement using a more lightweight structure to produce an output faster.

The main flow of our algorithm is based on the technique proposed by Cheng, Dey, Ramos, and Ray [7] for smooth surface meshing which was later adapted in the SURFREMESH algorithm by Dey, Li, and Ray [11] for meshing polygonal surfaces. This algorithm provides guar-

antees for the topology and geometry of the output mesh with respect to the input surface. We show that our two stage technique greatly improves the time and space required for computations.

## 2. Isosurfaces and Delaunay Meshing

In simulation and visualization applications, it is often the case that surfaces are represented in both a parameterized form and an implicit form. For computational purposes, the parameterized form is represented by a piecewise linear surface (a polygonal mesh), while the implicit form is usually represented by a *(scalar) volume dataset*, a collection of points in $\mathbb{R}^3$ each of which has the value of some scalar field $f$ associated with it. Moving from one form to another is often necessary: the implicit form is compact and appropriate for certain applications such as blending, but the parameterized form allows other computations such as static analysis and finite element methods to be performed efficiently.

The transformation from the implicit form to the parametric one is popularly known as the **Isosurface Problem**. Let $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ be a $C^2$-smooth function and $P \subset \mathbb{R}^3$ be a discrete point set. Define the volume dataset $\widehat{f} = \{(p, f(p)) \mid p \in P\}$, a subset of the graph of $f$. We will assume that $\Sigma = f^{-1}(\kappa)$ is a compact surface. The set $\Sigma$ is the *isosurface* of value $\kappa$ for $f$ and $\kappa$ is the *isovalue* on which $\Sigma$ is defined. The isosurface problem asks to find a polygonal mesh for $\Sigma$—a parametric representation of the implicit surface defined by the set $\{x \in \mathbb{R}^3 \mid f(x) = \kappa\}$. Note that one cannot sample $\Sigma$ given only the discrete data $\widehat{f}$. Instead, one interpolates a function $g$ from $\widehat{f}$ and presumes that $\Sigma = f^{-1}(\kappa) = g^{-1}(\kappa)$.

### 2.1. Delaunay vs. Non-Delaunay Isosurfaces

Delaunay refinement is credited to Chew [9] for describing the "furthest point" insertion strategy. Since then, the merits of the Delaunay refinement paradigm have already been shown for meshing a variety of domains including smooth surfaces [4, 7, 11], polyhedral surfaces, and the volumes enclosed by them [6, 15, 18]. Recently Cheng, Dey, and Ramos [5] showed how Delaunay refinement can be used for meshing domains as general as piecewise smooth complexes. Sampling theory in the context of surface reconstruction [1, 10] shows that in surface approximations both point-wise and normal-wise approximation errors depend on the circumradius of the triangles. Since a Delaunay triangulation keeps the circumradii of triangles small, it often provides a good approximation.

MARCHING CUBES is a popular algorithm for extracting isosurface meshes which are not necessarily Delaunay. This algorithm has many advantages; in particular, its speed and simplicity allow for rapid isosurface generation. However, the following issues remain a concern:

1.) Topological and geometric closeness are not guaranteed between the output triangulation and the isosurface $\Sigma$.

2.) The triangulation of $\Sigma$ is not *adaptive* in the sense that the gradedness or distribution of the surface samples is not sensitive to the features of the surface.

3.) No constraints regarding triangle shape exist—this deficiency can lead to numerical error and other simulation issues for finite element methods on the output polygonal mesh.

Variants of the original MARCHING CUBES exist to solve the topological concern [2, 8, 20]. However by adapting the Delaunay refinement approach in [7], our algorithm, DELISO, addresses all three of these concerns simultaneously and efficiently while producing a Delaunay mesh.

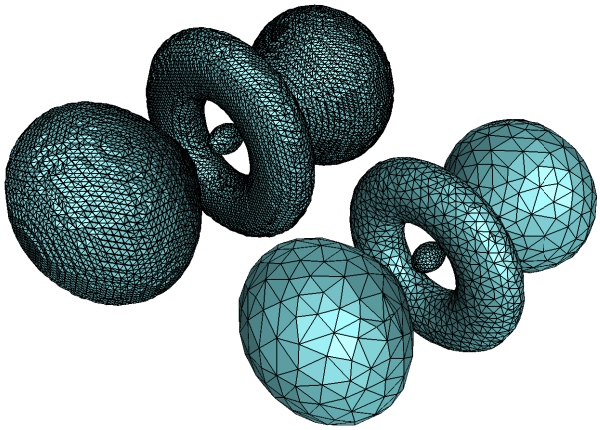

**Figure 1.** ATOM **isosurfaces. Left:** MARCHING CUBES **output. Right:** DELISO **output.**

As motivating examples, consider the isosurfaces generated in Figures 1 and 2. Here we show the isosurfaces of the ATOM and FUEL datasets generated with the isovalues 20.1 and 70.1, respectively. In each figure we have the outputs of both MARCHING CUBES and our isosurface meshing algorithm, DELISO. While both are topologically correct for these cases, only DELISO is provably so. Here the adaptivity of DELISO is apparent: for ATOM the outer spherical regions are meshed with larger, flatter triangles, the center torus is meshed with smaller triangles, and the inner ellipsoid with even smaller triangles. As a result, the MARCHING CUBES version has 22498 vertices compared with only 2089 vertices in the DELISO version. The FUEL dataset similarly has smaller triangles around the throughholes of the surface. In both datasets, there are triangles of arbitrary

skinniness in the MARCHING CUBES output, while in the output of our Delaunay refinement the aspect ratios are kept bounded.

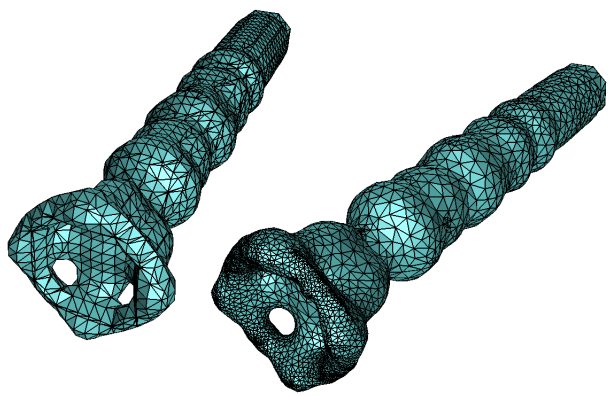

**Figure 2.** FUEL **isosurfaces. Left:** MARCHING CUBES **output. Right:** DELISO **output.**

## 2.2. Restricted Delaunay Refinement

Our goal is to avoid using the 3D Delaunay triangulation for the bulk of the insertions in a Delaunay refinement while maintaining the restricted Delaunay triangulation. For further discussion, we require the following formal definitions.

### 2.2.1. Definitions

Let $S \subset \mathbb{R}^3$ be a finite point set which satisfies general position. We define the *Voronoi cell*, $V_p$, of a point $p \in S$ to be the set $\{x \in \mathbb{R}^3 \mid \forall q \in S, \|p - x\| \leq \|q - x\|\}$. Voronoi cells are convex polyhedron, and for any $p, q \in S$, if $V_p \cap V_q$ is nonempty then it is the set of points which are equidistant from $p$ and $q$. For $2 \leq j \leq 4$, the intersection of $j$ Voronoi cells is called a $(4-j)$-dimensional Voronoi face. The 0-, 1-, and 2-dimensional Voronoi faces are called *Voronoi vertices*, *edges*, and *facets*, respectively. The *Voronoi diagram of* $S$, Vor $S$, is the collection of all Voronoi faces.

The convex hull of $j \leq 4$ points in $S$ is a $(j-1)$-dimensional Delaunay simplex, $\sigma$, if the vertices of $\sigma$ define a $(4-j)$-dimensional Voronoi face, $V_\sigma$, in Vor $S$. We call $\sigma$ and $V_\sigma$ the *dual* of each other. The 1-, 2-, and 3-dimensional Delaunay simplices are called *Delaunay edges*, *triangles*, and *tetrahedra*, respectively. The Delaunay simplices decompose the convex hull of $S$ into the *Delaunay triangulation of* $S$, denoted Del $S$.

Let $S$ be a point set on a surface $\Sigma$. For any Voronoi face $V_\sigma \in$ Vor $S$, the intersection $V_\sigma \cap \Sigma$ is called a *restricted Voronoi face*. The *restricted Delaunay triangulation*, Del $S|_\Sigma$, consists of the dual Delaunay simplices of the restricted Voronoi faces, that is, Del $S|_\Sigma = \{\sigma \in$ Del $S \mid V_\sigma \cap \Sigma \neq \emptyset\}$.

A Voronoi diagram satisfies the *topological ball property* if each $k$ dimensional Voronoi face either does not intersect the surface or intersects it in a $(k-1)$-ball. We know that if a point sample $S$ of a surface $\Sigma$ is sufficiently dense then Vor $S$ satisfies the topological ball property [1, 10]. Edelsbrunner and Shah's theorem [12] says that if Vor $S$ satisfies the topological ball property then Del $S|_\Sigma$ is homeomorphic to $\Sigma$. Together, these facts provide the basis for the topological guarantee of our Delaunay refinement algorithm. By inserting points until the topological ball property is satisfied, we have that $\Sigma$ and Del $S|_\Sigma$ are topologically equivalent.

When the topological ball property is satisfied, each Voronoi edge $V_\sigma$ that intersects the surface does so in a single point $x$. A ball $B_\sigma$ centered at $x$ and circumscribing $\sigma$ is called the *Voronoi ball* of $\sigma$. We say that a triangle $\sigma$ is *encroached* by a point $p$ if $B_\sigma$ contains $p$ in the interior.

We will use the notion of *poles* defined by Amenta and Bern [1] for estimating the scale of the local features as was done in Dey *et al.* [11]. The *positive pole* for a point $q$ is the furthest Voronoi vertex in $V_q$. The *negative pole* is the furthest Voronoi vertex of $V_q$ in the opposite direction. The *pole height*, $h_q$, for $q$ is the distance from $q$ to its negative pole.

### 2.2.2. Avoiding 3D Delaunay

The principal computational bottleneck in working with the restricted Delaunay triangulation is computing and maintaining the three dimensional Delaunay triangulation under point insertions. Our algorithm overcomes this difficulty by splitting the Delaunay refinement algorithm into two stages, where after the first we discard the Delaunay triangulation. For this strategy to work, one needs to have inserted sufficiently many points in the first stage so that the restricted Delaunay triangulation remains homeomorphic to the surface with further insertions. Although we cannot determine this point precisely in the algorithm, we remain satisfied by assuming that the first stage has fulfilled this condition after inserting enough points specified by some user defined parameters. These parameters are chosen experimentally.

Consider an insertion step in Stage 2. Let $p$ be a new point to be inserted in the existing point set $S$. We know that Del$(S \cup \{p\})|_\Sigma$ is homeomorphic to $\Sigma$. Thus Del$(S \cup \{p\})|_\Sigma$ is a piecewise linear manifold, so the union of all triangles incident to any point in Del $(S \cup \{p\})|_\Sigma$ is a topological disk. Let $D$ be the set of triangles incident on $p$ whose underlying space is $\cup D$. The set $D$ consists of the triangles that are in Del $(S \cup \{p\})|_\Sigma$ but not in Del $S|_\Sigma$. This simple observation is crucial for determining the new triangles that are needed to update the restricted Delaunay triangulation upon inserting $p$.

First we determine the triangles that should be deleted from Del $S|_\Sigma$ as a result of inserting $p$. Let $E$ be the set of triangles with underlying space $\cup E$—$E$ is the set of triangles which are encroached by $p$. We know the boundary of $\cup E$ must be same as that of $\cup D$. Since $\cup D$ is a topological disk, its boundary must be a single cycle implying that $\cup E$ is also a topological disk. Once we determine $E$, computing $D$ is trivial as its triangles are computed by connecting $p$ to the boundary edges of $E$. Therefore, the main task in updating Del $S|_\Sigma$ reduces to computing the set $E$. Figure 3 illustrates the technique we use.

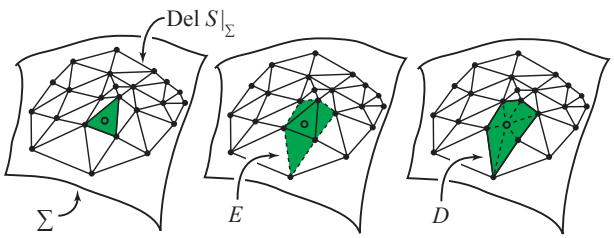


**Figure 3. After Stage 1, we can insert points in Del$S|_\Sigma$ without Del$S$. Left: We refine a triangle $\sigma$ (shaded) by inserting the center of $B_\sigma$. Center: This point encroaches a topological disk $E$. Right: We replace $E$ with $D$ by connecting the boundary of $E$ to the point.**

Since $\cup E$ is a topological disk, we can compute its triangles by a walk in the adjacency structure of Del $S|_\Sigma$. Suppose that we have computed a connected set $E' \subseteq E$. For each triangle $\sigma \in$ Del $S|_\Sigma$ not in $E'$ but sharing an edge with a triangle in $E'$, we check if $\sigma$ is encroached by $p$. If so, we add $\sigma$ to $E'$ and then walk to its neighbors.

To check if a restricted Delaunay triangle $\sigma$ is encroached we need to compute the Voronoi ball $B_\sigma$ and thus the intersection point between $\Sigma$ and the dual Voronoi edge, $V_\sigma$, of $\sigma$. We first compute the geometric dual of $\sigma$, a line $\ell_\sigma$ passing through the circumcenter of $\sigma$ and perpendicular to the plane of $\sigma$. Since the line $\ell_\sigma$ contains $V_\sigma$, we can use it to find the intersection of $V_\sigma$ and $\Sigma$. Let $x$ be the closest point to the circumcenter of $\sigma$ where $\ell_\sigma$ intersects the surface $\Sigma$. Since after Stage 1 the triangles in Del $S|_\Sigma$ approximate the surface to a reasonable level, the intersection of $V_\sigma$ and $\Sigma$ lies close to $\sigma$ meaning that $x$ is the intersection point between $V_\sigma$ and $\Sigma$ as well. Therefore, the ball centered at $x$ which circumscribes $\sigma$ is $B_\sigma$. If $p$ lies in the ball $B_\sigma$, the triangle $\sigma$ is encroached and does not belong to Del $(S \cup \{p\})|_\Sigma$. For improving future checks to determine if $\sigma$ is encroached, we keep the point $x$ stored with $\sigma$ when $\sigma$ is created.

The final piece needed to compute $E$ is an initial triangle in $E'$. In general, this computation would require a point location test. Fortunately, in our case the point $p$ to be inserted is always the center of the Voronoi ball $B_\sigma$ of some triangle $\sigma \in$ Del$S|_\Sigma$. The insertion of $p$ certainly eliminates $\sigma$ from the restricted Delaunay triangulation. Therefore, we can initialize $E'$ with $\sigma$ and then walk through adjacent triangles to determine $E$.

#### 2.2.3. Intersection Search

It is clear that one of the core computations required is the intersection between an arbitrary line or ray and the surface $\Sigma$. By partitioning the volume dataset into voxels we accomplish this task by first collecting the voxels that intersect the line and then determining if the line intersects the surface in each of these voxels. We elaborate on the actual computation of the intersection later in §4.

For the second stage of our algorithm, we seek the closest intersection point of $\ell_\sigma$ with $\Sigma$ for a triangle $\sigma$. To improve this computation, we start traversing voxels at the circumcenter of $\sigma$ and step in both directions along $\ell_\sigma$. By searching in this manner we find the intersection point after traversing only a few voxels.

#### 2.2.4. Timing comparisons

Before describing the algorithm, we provide some examples to illustrate how DELISO improves the computational cost over the standard Delaunay refinement involving full 3D structures. For our experiments, we used a variety of volume datasets generated at various isovalues $\kappa$, Table 1 gives a list of each dataset, the isovalue at which we generated the surface, and the dimensions of the dataset.

| Dataset | $\kappa$ | Dimensions |
|---|---|---|
| FUEL | 70.1 | $64 \times 64 \times 64$ |
| ATOM | 20.1 | $128 \times 128 \times 128$ |
| ENGINE | 40.1 | $256 \times 256 \times 256$ |
| LEG | 22.1 | $341 \times 341 \times 93$ |
| TOOTH | 146.9 | $256 \times 256 \times 161$ |
| CHEST | 82.7 | $384 \times 384 \times 240$ |
| BABY1 | 35.8 | $256 \times 256 \times 98$ |
| BABY2 | 131.6 | $256 \times 256 \times 98$ |
| MONKEY | 40.1 | $256 \times 256 \times 62$ |
| ANEURISM | 124.6 | $256 \times 256 \times 256$ |
| PIG | 120.1 | $512 \times 512 \times 134$ |

**Table 1. Volume datasets.**

Table 2 shows a timing comparison. This table indicates that a significant amount of processing time can be saved by using our technique. The first column of timings were generated by running the first stage of DELISO to a particular threshold (described in the next section) and the second

was generated by running DELISO in a two stage form to the same threshold.

| Dataset | 3D | DELISO | Speedup |
|---|---|---|---|
| ATOM | 16.39 | 5.85 | 2.80 |
| FUEL | 17.73 | 5.18 | 3.43 |
| ENGINE | 311.84 | 101.10 | 3.08 |
| LEG | 441.11 | 165.23 | 2.67 |
| TOOTH | 76.90 | 34.35 | 2.24 |
| CHEST | 1033.56 | 657.34 | 1.57 |
| BABY1 | 485.58 | 204.44 | 2.38 |
| BABY2 | 1134.87 | 468.93 | 2.42 |
| MONKEY | 2570.14 | 791.46 | 3.25 |
| ANEURISM | 663.65 | 262.05 | 2.53 |
| PIG | 614.19 | 288.83 | 2.13 |

**Table 2. Delaunay refinement time comparisons: using a 3D Delaunay triangulation for the entire refinement (3D) vs. our algorithm (DELISO). Times are in seconds.**

## 3. Algorithm Details

Recall that in our two-stage algorithm, we first create a 3D Delaunay triangulation whose restricted Delaunay triangulation is assumed to satisfy the topological ball property with respect to the isosurface in question. Intuitively, after the first stage we have recovered a "rough" version of the surface, but have not satisfied the geometric constraints desired. We next extract the restricted Delaunay triangulation into a polygonal mesh data structure. The second stage uses only this mesh to continue refinement of the restricted Delaunay triangulation until it is geometrically close to the isosurface.

```
DELISO(f̂, κ) {
    Del S_0 ← InitTriangulation()
    Del S_1 ← Recover(Del S_0)
    T_S2 ← Refine(Del S_1)
    return T_S2.
} //end DelIso()
```

**Figure 4. DELISO algorithm.**

In Figure 4 we give pseudocode for our two stage algorithm to mesh the isosurface $\Sigma$ at isovalue $\kappa$ defined by the volume dataset $\widehat{f}$. This algorithm assumes that we have a primitive operation that, given a volume dataset $\widehat{f}$ and isovalue $\kappa$, will compute the set of intersection points between a line segment $r$ and $\Sigma$. InitTriangulation() creates a Delaunay triangulation with a small (random) sample of points $S_0$ on $\Sigma$. In the first stage, Recover(), we take the set $S_0$ and produce a new set $S_1$ whose restricted Delaunay triangulation satisfies the topological ball property. Next, in the second stage, Refine(), we extract Del $S_1|_\Sigma$ into a polygonal mesh data structure, $\mathcal{T}_{S_1}$, and further refine it by inserting additional points to form the set $S_2$ whose restricted Delaunay triangulation (stored in $\mathcal{T}_{S_2}$) is both topologically correct and geometrically close to $\Sigma$.

### 3.1. Isosurface Recovery (Stage 1)

In Figure 5 we show the pseudocode for the Recover() stage. Each step requires some further explanation.

```
Recover(Del S) {
    Del S ← MultiIntersect(Del S)
    Del S ← ExtractManifold(Del S)
    Del S ← ApproximateGeom(Del S, ε, λ, r_min)
    Del S ← ExtractManifold(Del S)
    return Del S.
} //end Recover()
```

**Figure 5. Recover() algorithm.**

MultiIntersect() takes a Delaunay triangulation of a set $S$ and computes the restricted Voronoi face $V_\sigma \cap \Sigma$ of each Delaunay 2-simplex $\sigma \in$ Del $S$. The set $V_\sigma \cap \Sigma$ is simply a set of points. If this set has multiple elements, the point furthest from $\sigma$ is inserted. This process is repeated until no 2-simplices in Del $S$ have restricted Voronoi faces with more than one element.

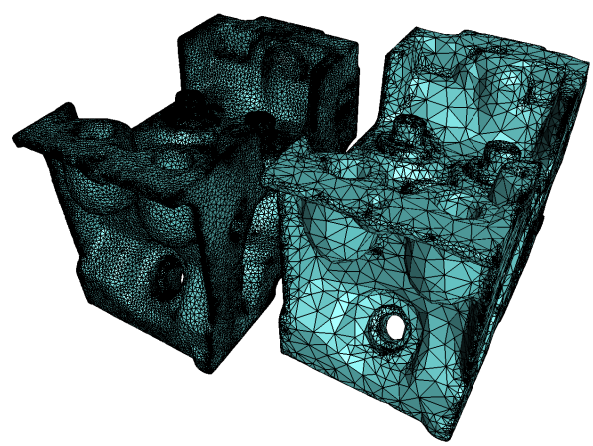

**Figure 6. ENGINE isosurface. Left: Final output of DELISO. Right: Output after Recover().**

ExtractManifold() takes a Delaunay triangulation of a set $S$ and considers the set of Delaunay 2-simplices in

the restricted Delaunay triangulation of $S$ which are adjacent to each 0-simplex $q \in S$. It checks that this set $D_q = \{\sigma \in \text{Del } S|_\Sigma \mid \sigma \cap q \neq \emptyset\}$ forms a topological disk. If it does not, the vertex $V_\sigma \cap \Sigma$ which is furthest from $q$ is inserted. This process is repeated until for each $q \in S$ the set $D_q$ forms a topological disk.

After ExtractManifold() returns, Del $S|_\Sigma$ becomes a manifold, but the topological ball property may still not hold. Luckily, only a minor amount of geometric refinement is typically needed before we safely discard the Delaunay triangulation in our second stage.

ApproximateGeom() takes a Delaunay triangulation of a set $S$ and checks that each 2-simplex $\sigma \in$ Del $S|_\Sigma$ satisfies certain geometric constraints. Let $r$ be the circumradius of $\sigma$, $h$ be the distance from the circumcenter of $\sigma$ to the point in $V_\sigma \cap \Sigma$, and $l$ be the shortest edge length of $\sigma$. We check that:

1. $h/r > \varepsilon$,
2. $r/l > \lambda$, and
3. $r > r_{\min}$.

If $r$ is not too small (condition 3) and either of conditions 1 or 2 holds, we insert the intersection point $V_\sigma \cap \Sigma$. This process is repeated until there is no such 2-simplex $\sigma$ with $r > r_{\min}$ for which condition 1 or 2 holds. The three parameters $\varepsilon$, $\lambda$, and $r_{\min}$ are user inputs. Typical values for each are $\varepsilon = 0.2$, $\lambda = 2.0$, and $r_{\min} = 0.001 * b$ where $b$ is the smallest dimension of the bounding box. These values are scale independent and work well for all models we tested.

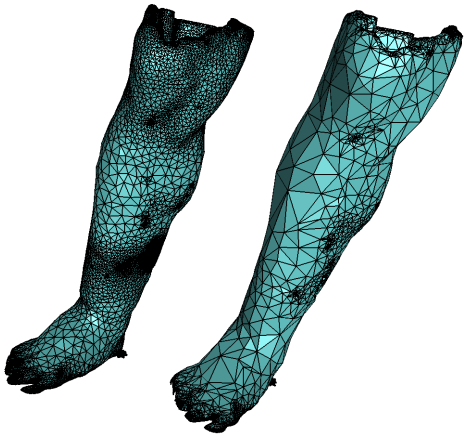

**Figure 7.** LEG **isosurface. Left: Final output of** DELISO**. Right: Output after Recover().**

Each condition is motivated for different reasons. Condition 1 causes the insertion of points which create 2-simplices in Del $S|_\Sigma$ respecting the features of $\Sigma$. Dey *et al.* [11] originally checked the ratio $r/h_q$, the radius over the pole height for a 0-simplex $q$, for capturing geometric features. Instead we use the ratio $h/r$ as it is significantly easier to compute. Condition 2 checks the aspect ratio of each 2-simplex, fixing 2-simplices which are not close to equilateral. Finally, condition 3 prevents triangles from becoming too small—which can cause numeric precision errors.

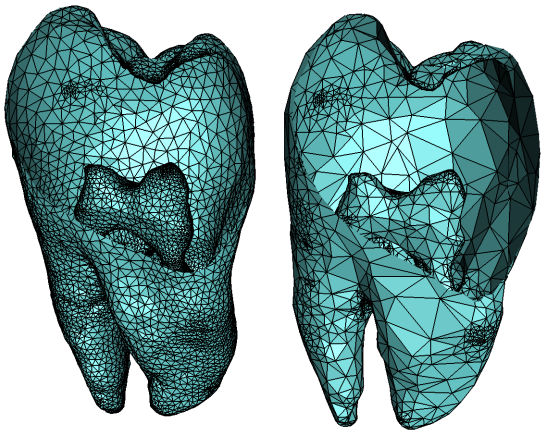

**Figure 8.** TOOTH **isosurface, drawn with a clipping plane to see the inside component of the surface. Left: Final output of** DELISO**. Right: Output after Recover().**

In Figures 6, 7, and 8 we show the output meshes for three datasets, ENGINE, LEG, and TOOTH. In each of these figures the resulting restricted Delaunay triangulation after the Recover() stage is shown on the right while the final output surface is drawn on the left. We observe that the mesh is topologically correct after Recover(), but still needs further refinement to capture the features appropriately.

### 3.2. Isosurface Refinement (Stage 2)

In the second stage of our algorithm, we extract Del $S_1|_\Sigma$ from the result of the first stage into a polygonal mesh structure which is independent from Del$S_1$. We continue our Delaunay refinement algorithm, but we will no longer require the 3D Delaunay triangulation to compute the restricted Delaunay triangulation.

For a point set $S$, define $\mathcal{T}_S = (S, F)$ to be the triangular mesh structure where

$$F \subseteq \{\{a, b, c\} \mid a, b, c \in S\}$$

is the set of facets in $\mathcal{T}_S$. We store adjacency information within the facets as well—each facet $f \in F$ knows the three facets which share an edge with $f$. This data structure is

satisfactory for representing 2-manifolds (potentially with boundary) as triangular meshes. As discussed in §2, for each $f \in F$ we also store the nearest intersection point $x$ to $f$ between $\Sigma$ and the dual line of $f$ to improve future encroachment checks on point insertions.

In Figure 9 we show pseudocode for the second stage of our algorithm. We first compute the pole heights for each vertex in Del $S_1$. These pole heights are used as refinement criteria later in the Refine() step. Next, BuildTriMesh() creates the polygonal mesh structure using Del $S_1$, the output of the first stage. Finally, by using RefineGeom() we refine $\mathcal{T}_{S_1}$ to $\mathcal{T}_{S_2}$.

```
Refine(Del S_1) {
    ComputePoles(Del S_1)
    T_S1 ← BuildTriMesh(Del S_1)
    T_S2 ← RefineGeom(T_S1, ε_1, ε_2, λ, r_min)
    return T_S2.
} //end Refine()
```

**Figure 9. Refine() algorithm.**

With the exception of BuildTriMesh(), the other steps of this stage require some additional clarifications:

ComputePoles() takes as input the Delaunay triangulation, Del $S_1$, and computes the pole height, $h_q$, for each 0-simplex $q \in$ Del $S_1$. For any $q$, we know that $V_q \cap \Sigma$ is a topological disk at this stage since ExtractManifold() guarantees this property. Moreover, this disk separates $V_q$ into exactly two subsets, denoted $V_q^+$ and $V_q^-$, on either side of the disk. Let $q^+$ and $q^-$ be the points which are furthest from $q$ in $V_q^+$ and $V_q^-$, respectively. $q^+$ and $q^-$ are similar to poles defined for smooth surfaces. We define $h_q = \min\{\|q - q^+\|, \|q - q^-\|\}$.

RefineGeom() takes as input the triangular mesh $\mathcal{T}_{S_1}$ output by BuildTriMesh() and generates a new mesh $\mathcal{T}_{S_2}$ by refining each triangle $\sigma$. Let $h_\sigma$ be the pole height of a triangle $\sigma$ defined by averaging the pole heights at the vertices of $\sigma$. We check the following criteria:

1. $h/r > \varepsilon_1$,
2. $r/h_\sigma > \varepsilon_2$,
3. $r/l > \lambda$, and
4. $r > r_{\min}$.

Similar to ApproximateGeom(), we have three types of criteria. First, conditions 1 and 2 are used to capture the features of $\Sigma$. Condition 3 keeps the aspect ratio of the triangles bounded, and condition 4 prevents triangles from becoming too small. If any of conditions 1, 2, or 3 holds, and condition 4 is satisfied, we insert the point $V_\sigma \cap \Sigma$. When inserting, we first compute the disk consisting of encroached facets as described in §2, remove it, and then fill it by connecting its boundary to the vertex we insert. In our experiments, we use the same values for $\lambda$ and $r_{\min}$ as before. We pick $\varepsilon_1 = 0.1$, a reduction from $\varepsilon = 0.2$ in the first stage, and $\varepsilon_2 = 0.2$.

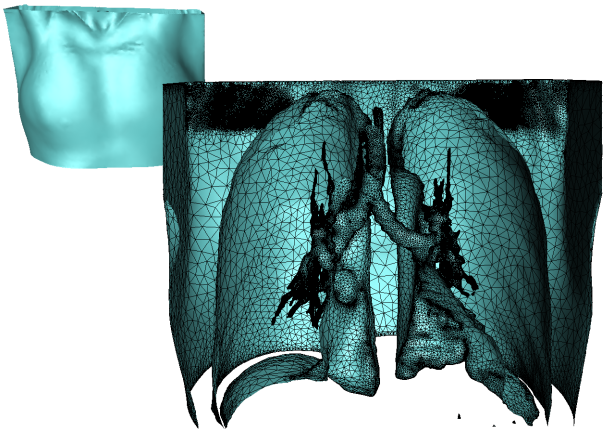

**Figure 10.** DELISO **output for** CHEST**. Back: External component. Front: By clipping the front we see the mesh for the lungs, trachea, and bronchi.**

In practice, we have seen that the value $r/h_\sigma$ is more desirable than $h/r$ as a criterion for geometric refinement. However, pole computations are fairly expensive and require Del $S_1$, so we use $h/r$ in the first stage. Then at the start of Refine() we compute them once before discarding Del $S_1$. When a new point is inserted we estimate its pole height by averaging the pole heights of its adjacent vertices. At this stage since $\mathcal{T}_{S_2}$ is close to $\Sigma$, nearby points have similar pole heights. The averaging step is an effective approximation that allows us to use $r/h_\sigma$ for refinement.

Figures 10 and 11 show some results for two additional datasets (CHEST and BABY). Both show the adaptivity of DELISO by capturing the small features densely and smoothly transitioning to larger triangles in flatter regions. The inner features of CHEST and the tube, mouth, and double sheet of BABY are meshed densely as a result.

Table 3 gives timing results showing the amount of time saved for insertions. For each dataset, we calculate the average insertion times during the Recover() and Refine() stages. In all instances, insertion during Refine() is approximately two orders of magnitude faster. Not shown in this table is where most of the insertion time is spent. For the Recover() stage the majority (80-85%) is spent determining which new Delaunay simplices are restricted Delaunay. The remaining portion is used to compute the new Delaunay triangulation and identify new Delaunay simplices.

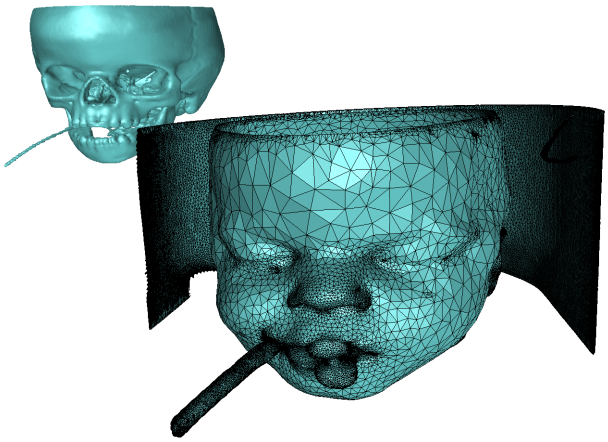

**Figure 11.** DELISO **output for** BABY**. Back: Isosurface for** BABY2**. Front: Mesh of isosurface for** BABY1**.**

| Dataset | $t_{RC}$ | $n_{RC}$ | $t_{RF}$ | $n_{RF}$ |
|---|---|---|---|---|
| FUEL | 1.54e-03 | 1345 | 1.36e-05 | 2936 |
| ATOM | 1.36e-03 | 554 | 1.79e-05 | 1788 |
| ENGINE | 1.53e-03 | 23291 | 1.56e-05 | 84942 |
| LEG | 1.66e-03 | 38889 | 1.70e-05 | 82185 |
| TOOTH | 1.58e-03 | 5416 | 1.54e-05 | 14242 |
| CHEST | 1.82e-03 | 76887 | 1.78e-05 | 280953 |
| BABY1 | 1.56e-03 | 50886 | 1.67e-05 | 217031 |
| BABY2 | 1.71e-03 | 120637 | 1.74e-05 | 392150 |
| MONKEY | 1.60e-03 | 155304 | 1.63e-05 | 423557 |
| ANEURISM | 1.71e-03 | 74173 | 1.57e-05 | 140557 |
| PIG | 1.62e-03 | 52240 | 1.60e-05 | 314368 |

**Table 3. Average insertion times $t_{RC}$ and $t_{RF}$ (in seconds) for inserting $n_{RC}$ and $n_{RF}$ points in the Recover() ($RC$) and Refine() ($RF$) stages, respectively.**

## 4. Intersection Search

We have deferred discussion of how to compute the intersection points between a ray and a volume dataset to this section. Let the line segment $r = \overline{a_0 a_1}$, be parameterized by $r(t) = a + bt$ for $t \in [0, 1]$ where $a = a_0$ and $b = a_1 - a_0$. Given a volume dataset $\widehat{f}$, an isovalue $\kappa$, and $r$, we compute the set of all points in $r \cap \Sigma$ in two steps, shown as the Intersect() algorithm in Figure 12. In Intersect() we first identify which voxels of the volume dataset are potential candidates for intersection and then compute the intersection points in those cells.

```
Intersect(f̂, κ, r) {
    V ← CollectVoxels()
    X ← FindIntersections(V)
    return X.
} //end Intersect()
```

**Figure 12.** INTERSECT **algorithm.**

### 4.1. $k$d-tree Searches

There are a number of different techniques for ray tracing isosurfaces based on $k$d-tree data structures. The goal of these approaches is to rapidly identify which primitive elements (in our case, voxels) are candidates for a more expensive intersection check.

We modify one approach in Havran [13] using a method similar to Wald *et al.* [21] and adapt it to our needs. First, we construct a $k$d-tree on the voxels where at each level we split the set of voxels in half along voxel boundaries in the dimension of the split. At each node we store the maximum and minimum function values ($\kappa_{\min}$ and $\kappa_{\max}$) defined over the portion of the volume represented by that node. These values are computed in a recursive manner.

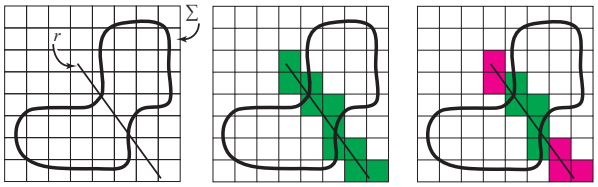


**Figure 13. A $k$d-tree facilitates ray-isosurface intersections. Left: Isosurface $\Sigma$, ray $r$, and the $k$d-tree. Center: Using only the tree and $r$ we identify a set of candidate voxels (in green). Right: Using $\kappa_{\min}$ and $\kappa_{\max}$ we can prune the magenta cells.**

Once we have a $k$d-tree where each node stores this maximum and minimum information, we can collect the set of candidate voxels where $r$ intersects $\Sigma$ in an efficient manner. Figure 13 illustrates this technique in two dimensions. The recursive traversal of Havran begins at the root of the tree and checks if $r$ crosses the split plane, if so it traverses both sides, otherwise it checks whichever side $r$ lies on and continues into the tree. In this manner we prune away just those voxels that the ray does not intersect. However, we can prune additional voxels by checking if $\Sigma$ is within those voxels too. Using the maximum and minimum isovalues, we do an additional check to ensure that the subtree we are traversing has the isosurface in it by checking

$\kappa_{\min} \leq \kappa \leq \kappa_{\max}$. If not, we prune that region even though the $r$ intersects it.

### 4.2. Voxel-Ray Intersection

Once we have collected the set $V$ of candidate voxels in which $\Sigma$ may intersect $r$, we compute the intersection points. We are most interested in finding these intersections exactly, so we solve the system defined by the equations for $r$ and the trilinear surface specified by the eight sample points at the corner of each voxel. This technique is given by Parker *et al.* [16] and reduces to solving the roots of (at most) a cubic polynomial. To solve this polynomial, we use the algorithm of Schwarze [17].

One important enhancement is used here during the second stage, Refine(). As discussed in §2, we can improve searching for the intersection point of $\ell_\sigma$ and $\Sigma$ by stepping along the voxels from the circumcenter of $\sigma$. Since we are searching for the closest intersection point to $\sigma$, we search by starting with a small segment along the dual line of $\sigma$ centered at $\sigma$'s circumcenter. We then perform an intersection check. If we find an intersection we are done, otherwise we increase the size of our segment and repeat, skipping voxels we have already searched. In this manner, we check the closest voxels first—the result is an intersection search with fewer voxel-ray intersection computations.

In Table 4 we show a comparison of the number of voxel-ray intersection computations versus the number of searches at each stage of the algorithm. Since the voxel-ray intersection requires solving a potentially cubic system, this computation is expensive and one would like to minimize it. This table shows that on average we only do one per search, but for those searches where we actually do find an intersection point, we tend to do more. However, in the Refine() stage, we minimize the number of computations by searching for intersection points incrementally along the dual line of each triangle.

## 5. Boundary Issues

One assumption for our algorithm is that the isosurface does not have any boundary. For volume datasets, it is common to have isosurfaces with boundaries since the surface is often truncated by the bounding volume of the dataset. For example, in our experiments only ATOM, ENGINE, and TOOTH did not have boundary curves at the isovalues chosen. Special treatment is required in the Disk() test to handle boundaries, since points on the boundary cannot satisfy this condition.

We make two modifications to adapt the original algorithm to isosurfaces with boundary. First, we precompute the boundary curves and insert them into the initial sample.

| Dataset | $CPI_{RC}$ | $CPI^*_{RC}$ | $CPI_{RF}$ | $CPI^*_{RF}$ |
|---|---|---|---|---|
| FUEL | 1.00 | 3.25 | 1.08 | 1.16 |
| ATOM | 0.72 | 3.41 | 1.30 | 1.45 |
| ENGINE | 0.86 | 3.38 | 1.16 | 1.25 |
| LEG | 1.06 | 2.91 | 1.07 | 1.17 |
| TOOTH | 0.76 | 3.10 | 1.13 | 1.27 |
| CHEST | 1.13 | 2.58 | 1.08 | 1.13 |
| BABY1 | 1.04 | 2.96 | 1.06 | 1.11 |
| BABY2 | 1.08 | 2.72 | 1.05 | 1.11 |
| MONKEY | 1.06 | 2.83 | 1.06 | 1.14 |
| ANEURISM | 0.86 | 2.85 | 1.06 | 1.13 |
| PIG | 1.25 | 3.96 | 1.11 | 1.21 |

**Table 4. Comparison of the number of trilinear computations versus the number of searches for the Recover() ($RC$) and Refine() ($RF$) stages. Shown are the average number of computations per intersection search ($CPI_{RC}$ and $CPI_{RF}$) and the $CPI$'s during only those searches which returned intersection points ($CPI^*_{RC}$ and $CPI^*_{RF}$).**

Second, we modify the Disk() test to allow half disks for points on the boundary.

The insertion of the boundary points is done at the end of the InitTriangulation() step. After computing the random sample of points and inserting them, we next compute the set of boundary curves and insert those. For simplicity, this is done using MARCHING SQUARES on each of the six boundary faces. Upon insertion, we mark these points and allow the Disk() test to be satisfied for these points by either a complete topological disk or a topological half disk. In this manner, we relax the constraints of the algorithm, but only at known boundary points.

## 6. Conclusions and Future Work

We have presented an algorithm, DELISO, for meshing isosurfaces with Delaunay triangles. This algorithm works on the principle of Delaunay refinement with the topological ball property. Although a three dimensional Delaunay triangulation is at the core of this refinement paradigm, we show how the bulk of the computations can be carried out without maintaining the full 3D structure. This improvement eases the computational burden without sacrificing any quality guarantees.

Research is still necessary to make further enhancements to this algorithm. Clearly, the earlier we switch to Refine(), the less time will be required for meshing. However, a certain sampling density is necessary to guarantee the topological ball property at this stage. One would like to understand the precise point where the surface can be extracted with a

provable guarantee that we can continue to update Del $S|_\Sigma$ without Del $S$. Computing ray-volume intersections is also an active field of research which could also improve the implementation of this algorithm.

## Acknowledgements

We acknowledge the support of NSF grants CCF-0430735 and CCF-0635008. All datasets used in this paper are freely available at well known sites, including http://www.volvis.org and http://www9.cs.fau.de/Persons/Roettger/library/. The authors would also like to thank Jason Bryan for assistance using VOLSUITE to generate the MARCHING CUBES isosurfaces.

## References

[1] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *Discrete and Computational Geometry*, 22:481–504, 1999.

[2] P. Bhaniramka, R. Wenger, and R. Crawfis. Isosurface construction in any dimension using convex hulls. *IEEE Transaction on Visualization and Computer Graphics*, 10:130–141, 2004.

[3] J.-D. Boissonnat, D. Cohen-Steiner, and G. Vegter. Isotopic implicit surface meshing. In *STOC '04: Proceedings of the 36th ACM Symposium on Theory of Computing*, pages 301–309. ACM Press, 2004.

[4] J. D. Boissonnat and S. Oudot. Provably good surface sampling and approximation. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 9–18. Eurographics Association, 2003.

[5] S.-W. Cheng, T. K. Dey, and E. A. Ramos. Delaunay refinement for piecewise smooth complexes. In *SODA '07: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1096–1105. ACM Press, 2007.

[6] S.-W. Cheng, T. K. Dey, E. A. Ramos, and T. Ray. Quality meshing for polyhedra with small angles. In *SCG '04: Proceedings of the 20th Symposium on Computational Geometry*, pages 290–299. ACM Press, 2004.

[7] S.-W. Cheng, T. K. Dey, E. A. Ramos, and T. Ray. Sampling and meshing a surface with guaranteed topology and geometry. In *SCG '04: Proceedings of the 20th Symposium on Computational Geometry*, pages 280–289. ACM Press, 2004.

[8] E. Chernyaev. Marching cubes 33: Construction of topologically correct isosurfaces, 1995.

[9] L. P. Chew. Guaranteed-quality mesh generation for curved surfaces. In *SCG '93: Proceedings of the 9th Symposium on Computational Geometry*, pages 274–280. ACM Press, 1993.

[10] T. K. Dey. *Curve and surface reconstruction: algorithms with mathematical analysis*. Cambridge University Press, New York, 2006.

[11] T. K. Dey, G. Li, and T. Ray. Polygonal surface remeshing with Delaunay refinement. In *Proceedings of the 14th International Meshing Roundtable*, pages 343–361, 2005.

[12] H. Edelsbrunner and N. R. Shah. Triangulating topological spaces. In *SCG '94: Proceedings of the 10th Symposium on Computational Geometry*, pages 285–292. ACM Press, 1994.

[13] V. Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, 2000.

[14] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th Conference on Computer Graphics and Interactive Techniques*, volume 21, pages 163–169. ACM Press, Jul 1987.

[15] S. Oudot, L. Rineau, and M. Yvinec. Meshing volumes bounded by smooth surfaces. In *Proceedings of the 14th International Meshing Roundtable*, pages 203–219, 2005.

[16] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive ray tracing for isosurface rendering. In *VIS '98: Proceedings of the Conference on Visualization*, pages 233–238. IEEE Computer Society Press, 1998.

[17] J. Schwarze. Cubic and quartic roots. In A. Glassner, editor, *Graphics gems*, pages 404–407. Academic Press Professional, Inc., San Diego, CA, 1990.

[18] J. R. Shewchuk. Tetrahedral mesh generation by delaunay refinement. In *SCG '98: Proceedings of the 14th Symposium on Computational Geometry*, pages 86–95, 1998.

[19] B. T. Stander and J. C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *SIGGRAPH '97: Proceedings of the 24th Conference on Computer Graphics and Interactive Techniques*, pages 279–286. ACM Press/Addison-Wesley Publishing Co., 1997.

[20] G. Varadhan, S. Krishnan, T. V. N. Sriram, and D. Manocha. Topology preserving surface extraction using adaptive subdivision. In *Eurographics Symposium on Geometry Processing*, pages 241–250, 2004.

[21] I. Wald, H. Friedrich, G. Marmitt, P. Slusallek, and H.-P. Seidel. Faster isosurface ray tracing using implicit kd-trees. *IEEE Transactions on Visualization and Computer Graphics*, 11(5):562–572, 2005.

[22] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, February 1986.