GPU Surface Extraction using the Closest Point Embedding

Mark Kim and Charles Hansen

Scientific Computing and Imaging Institute at the University of Utah, Salt Lake City, USA;
School of Computing at the University of Utah, Salt Lake City, USA

ABSTRACT

Isosurface extraction is a fundamental technique used for both surface reconstruction and mesh generation. One method to extract well-formed isosurfaces is a particle system; unfortunately, particle systems can be slow. In this paper, we introduce an enhanced parallel particle system that uses the closest point embedding as the surface representation to speed-up the particle system for isosurface extraction. The closest point embedding is used in the Closest Point Method (CPM), a technique that uses a standard three dimensional numerical PDE solver on two dimensional embedded surfaces. To fully take advantage of the closest point embedding, it is coupled with a Barnes-Hut tree code on the GPU. This new technique produces well-formed, conformal unstructured triangular and tetrahedral meshes from labeled multi-material volume datasets. Further, this new parallel implementation of the particle system is faster than any known methods for conformal multi-material mesh extraction. The resulting speed-ups gained in this implementation can reduce the time from labeled data to mesh from hours to minutes and benefits users, such as bioengineers, who employ triangular and tetrahedral meshes.

Keywords: Three-Dimensional Graphics and Realism, Surface Reconstruction, Graphics processors, parallel processing, iso-surface extraction, GPU acceleration, scientific visualization

1. INTRODUCTION

Isosurface extraction from three-dimensional scalar volumes is a fundamental technique in visualization. In some cases, the scalar data may be composed of different materials and although the material is stored in a regular grid, the material interfaces generally do not conform to the underlying grid. Recent work by Meyer et al.[1,2] uses a particle-based approach to extract a curvature-dependent, well-formed multimaterial mesh from biological data. This approach uses an energy based system to extract a surface mesh with nearly equilateral triangles. Further, it generates meshes with smaller triangles in areas of high curvature which gives more resolution in areas that need it. Well-formed triangular meshes are a good starting point to generate a tetrahedral mesh that is well suited for finite element simulation.

BioMesh3D[3] is a recent tool based on Meyer's research. However, due to the computational complexity of the particle advection process, users are required to find a balance between the heavy computation required and their needs in terms of the quality of the mesh, quantity of tetrahedrons and the time anticipated to extract the mesh. The excessive computational cost to generate a well-shaped multimaterial mesh has hindered the use of the curvature-dependent particle system by the bioengineering community for numerical simulations.[4] For instance, an attempt was made to extract a mesh from a six material dataset, but was finally stopped after two months because it had yet to finish.[5] Therefore, improving the performance could increase the use of the particle system for multimaterial mesh extraction.

In recent years, advances in computing power have come from an increase in the number of cores. This is particularly true for the graphic processing unit, or GPU, where hundreds of cores are run in a single instruction, multiple thread (SIMT) fashion. To take advantage of this new parallel processing power, efficient parallel algorithms are needed. Kim et al.[6] proposed a dynamic particle system for the GPU to accelerate the particle advection procedure during mesh extraction. This showed up to an order of magnitude speed-up over the CPU implementation for curvature-dependent isosurface extraction. However, it was limited to small volumes due to limited GPU memory size.

The direct adaption of the Meyer particle system to the GPU is not a natural mapping to the SIMT architecture. Kim et al. used a Red-Black update scheme which, coupled with the amount of control flow required, hinders performance on the GPU.[6] Additionally the surface representation, a distance field, requires a reprojection step realized through an iterative

Further author information: (Send correspondence to Mark Kim)
Mark Kim: E-mail: mkim@sci.utah.edu, Telephone: 1 801 587 9459

search process to place particles back onto the surface. This is inefficient on the GPU due to the amount of control flow as well. Therefore, a new approach is to overcome these issues.

In this paper we devise an efficient implementation of a particle system for a GPU-based approach for labeled multi-material mesh extraction. Our contributions are as follows:

- Use of the closest point embedding to define the surfaces in the volume for faster reprojection. (Section 4).
- Barnes-Hut tree code as an acceleration structure for the multimaterial mesh extraction (Section 5).
- New seeding and add/delete algorithms to efficiently place new particles (Section 6).
- The first use of particles for direct visualization of a closest point embedding of a surface.

The remainder of the paper is organized as follows. Section 2 reviews the most relevant work. Section 3 reviews how to represent the interfaces between different materials to place particles on them. Section 4 discusses the closest point embedding for representing the interface while Section 5 describes the Barnes-Hut treecode implementation on the GPU. Section 7 provides the results of two biological datasets using the proposed closest point implementation. Further, the pros and cons of the implementation are discussed and the experimental results are explored. We conclude the paper in Section 8.

2. RELATED WORKS

2.1 Particle Systems on the GPU

Particle systems on the GPU originate from both computer graphics and visualization as well as computational physics. Particle systems for real-time animation and rendering of particles in OpenGL were introduced by Kolb et al.[7] and Kipfer et al.[8] For visualization, Kruger et al. used a particle system for real-time 3D flow visualization on the GPU.[9] Extending the particle system beyond computer graphics, the GPU was used for simulating fluid motion with smooth particle hydrodynamics (SPH).[10] A good overview of state-of-the-art in SPH on the GPU can be found in Goswami et al.[11]

2.2 Particle System for Surface Extraction

To visualize implicit functions, Witkin and Heckbert [12] introduced a particle system to distribute particles evenly over the surface. Particles are placed on the zero-set of the implicit function and follow an energy gradient until a minimum global energy is found. Following the lead of Witkin and Heckbert in the use of particles for visualization, Crossno et al. used a particle system to extract isosurfaces from scalar fields.[13] Meyer et al. employed an energy-based particle system for robust visualization of implicit surfaces[14] and extracting high quality meshes from scalar fields.[1] The use of a compact cotangent energy coupled with curvature dependent particle placement allowed for the extraction of well-formed, locally adaptive meshes. Further, Meyer et al. extended the curvature-dependent particle-based mesh extraction system to extract meshes from multi-material volumes.[2] Finally, Bronson et al. introduced a particle-based system for generating adaptive triangular surfaces and tetrahedral meshes for CAD models.[15] Instead of precomputing feature size, their system adapts to curvature and moves the particles in the parameter space.

While the particle system introduced by Meyer et al. is robust, this robustness comes with an increased computational cost. To speed up the surface extraction, Kim et al. extracted the surface using the GPU.[6] By doing the surface extraction on the GPU, they were able to improve the performance by up to 44 times over the CPU extraction.

2.3 Multi-material Surface Extraction

Bloomenthal and Ferguson[16] were one of the first to extract surfaces from non-binary classification to construct non-manifold surfaces. Their method begins by tetrahedralizing a cubic cell on the surface and from that tetrahedralization constructs a triangulation. Unfortunately, this method can lead to an excessive number of triangles and can generate poorly shaped or degenerate triangles. To improve marching cubes, Nielson and Franke[17] assign each vertex to several different classes instead of the strict binary classification of marching cubes while Hege et al.[18] assigns a probability to the vertices and interpolates. Bonnell et al.[19] use the fractional values of the materials in the cell to create a dual grid. The dual grid has the barycentric coordinates of the fractional values of each material in the original cell at the vertices. On this dual grid,

Algorithm 1 Overview

```
$GenerateDistanceVolumes()$
$GenerateSizingFields()$
$PreviousEnergy = 0$
$CurrentEnergy = 1.0$
$\delta \leftarrow (PrevEnergy - CurrentEnergy)/PrevEnergy$
while $\delta > \delta_{prime}$ do
    $PreviousEnergy = CurrentEnergy$
    $CurrentEnergy = advectParticles()$
    $\delta = (PrevEnergy - CurrentEnergy)/PrevEnergy$
    $AddDeleteParticles()$
end while
$SurfaceExtraction()$
```

the intersections are calculated with the Voronoi cells and then transformed back to the original space and triangulated to create an approximate boundary. Reitinger et al.[20] generalized marching cubes to 8 materials on the vertices, but this leads to increased ambiguities.

These grid-based methods are mostly efficient, but are still constrained to first order approximation whose resolution is tied to the grid rather than the underlying geometry. Recently, Zhang[21] proposed an octree-based approach based on dual-contouring.[22] Another strategy based on Delaunay refinement and was proposed by Oudot et al. [23] and Pons et al.[24] However, two material and three material junctions are not explicitly handled in this work and not represented in the output. Anderson et al.[25] subdivide a cell and randomly assign materials to the subcells so that it is approximately the same volume fraction as the original cell. Then, subcells are randomly swapped to minimize the surface area between materials. The surface, unfortunately, is still blocky at the subcell level. Anderson et al. address this discretization issue by iteratively smoothing the interface.[25] Another volume of fluid method, Prilepov et al. propose a multimaterial interface reconstruction based on fluid percentage.[26]

Meyer et al.[2] proposed a particle-based method to explicitly sample junctions between materials. These patches are not tied to the grid resolution, rather they are based on the underlying surface geometry. Unfortunately, these high quality, Delauney-based meshes take a long time to generate.

2.4 Closest Point Embedding

The closest point method (*CPM*) was introduced by Ruuth and Merriman as a technique for solving PDEs on surfaces.[27] Its usefulness is in its simplicity whereby unmodified three dimensional differential operators replace two dimensional intrinsic surface operators on an embedded surface. Macdonald and Ruuth continued the work with an implicit time step, which replaced the original two-phase explicit time step as well as evolving a level-set on a surface.[28,29] März and Macdonald followed up the works of Macdonald and Ruuth with proofs for the principles of the method.[30] while Tian et al. followed up the level-set on a surface with segmentation on a surface.[31] Hong et al. applied the *CPM* to the level-set equation to simulate fire on an animated surface.[32] Finally, Auer et al. used the closest point method to solve the Navier-Stokes equations on dynamic surfaces.[33]

The use of three dimensional operators on a two dimensional surface would not be possible without the closest point embedding. This embedding is used in this work to reproject particles back on the surface without an iterative search which is important when adapting the particle system to the GPU.

3. SURFACE EXTRACTION

Before the closest point embedding is discussed, the original multimaterial surface extraction of Meyer et al.[2] is reviewed because the closest point embedding is generated from distance fields produced by BioMesh3D. A high level overview of the surface extraction is presented in Algorithm 1. The input is a single, labeled multimaterial volume. The function $GenerateDistanceVolumes()$ returns a distance field volume, DF for every material (Section 3.1). The function $GenerateSizingFields()$ generates a sizing field, h for each material to specify how far a particle, $\mathbf{p}_i$ should be from its

Algorithm 2 GenerateDistanceVolumes()

```
                                        ▷ Input: Labeled Volume, Output: $N$ Signed Distance Fields
    for all material $m_i \in materials$ do
        create a Binary Volume, $BV_i$ of $m_i$
        Smooth the Volume, $BV_i$
        Tighten the Volume, $BV_i$, by user specified radius, $r$
    end for
```

neighbors $\mathbf{p}_j$ to meet local feature size (*LFS*) sampling requirements (Section 3.1). Once the distance and sizing fields are generated, they are used to place the particles on the surface in the function *AdvectParticles*() (Section 3.2). The distance fields and sizing fields represent the implicit surface of the materials as well as the distance of a particle to its neighbors, respectively. To help in the advection process, particles may be added or deleted to place more particles in areas of high curvature and fewer particles in areas of low curvature (Section 6). The particle advection is terminated if $||CurrentEnergy - PrevEnergy||/PrevEnergy < \delta_{prime}$. In practice, we have found $\delta_{prime} = 0.0015$ to be a good choice. Once the energy of the system has plateaued and the particles are evenly spaced, all the particles are transformed into a tetrahedral mesh with Tetgen[34] and then the individual material interfaces are extracted from the tetrahedral mesh as triangular meshes. In practice, BioMesh3D[3] is used to generate the distance and sizing fields as well as to extract the triangular meshes from the tetrahedral mesh. For more detail, we refer the reader to Meyer et al.[1,3]

3.1 Multimaterial Distance Field and Sizing Field

The distance field generation is described in Algorithm 2. The preprocessing step to generate the distance fields generates N distance fields, where N is the number of materials, and each distance field represents the implicit surface of that material. Initially, a labeled volume is split into multiple binary volumes and smoothed with a binary morphology operation[35] which outputs an antialiased grayscale volume. The antialiased grayscale volume is then smoothed with a level set curvature approach called *tightening*.[36] Tightening limits the radius of curvature of the antialiased volume with a user specified radius, r. The tightening procedure generates a signed distance field. A sizing field volume, h_i is generated for every material, m_i and specifies the distance a particle is from its neighbors to meet local feature size (*LFS*) sampling requirements.[1] In practice, the distance and sizing fields are generated with BioMesh3D.[3]

3.2 Particle System

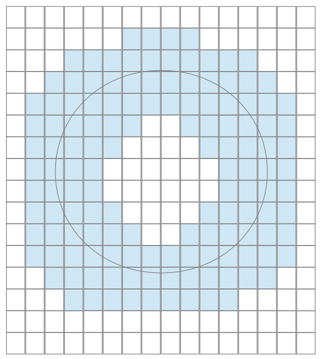
The energy E_i and velocity $\mathbf{v}_i$ is calculated based on the distance from particle $\mathbf{p}_i$ to its neighboring particles $\mathbf{p}_j$ with a cotangent energy function (Eq. 1). The particle $\mathbf{p}_i$ is advected in the tangent space of the surface, in the direction $\mathbf{v}_{ij}$ and then reprojected back onto the surface. The distance, $\mathbf{d}_{ij}$ is the distance between the particles $\mathbf{p}_i$ and $\mathbf{p}_j$, adjusted by their sizing field values, h_i and h_j respectively (Eq. 2). See Kim et al. for more details about the energy equations.[6]

$$E_{ij} = \begin{cases} cot(|\mathbf{d}_{ij}|\frac{\pi}{2}) + |\mathbf{d}_{ij}|\frac{\pi}{2} - \frac{\pi}{2} & |\mathbf{d}_{ij}| \leq 1.0 \\ 0 & |\mathbf{d}_{ij}| > 1.0 \end{cases} \tag{1}$$

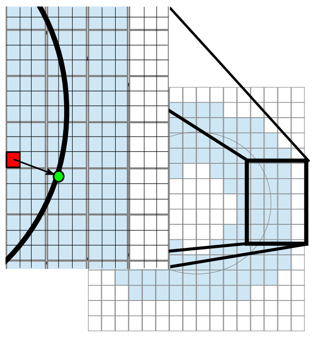
where

$$\mathbf{d}_{ij} = \frac{\mathbf{p}_i - \mathbf{p}_j}{2 \times cos(\frac{\pi}{6}) \times min(h_i, h_j)} \tag{2}$$

Because the particle, $\mathbf{p}_i$ is advected in the tangent space, $\mathbf{p}_i$ needs to be projected back onto the surface. However, the implicit function is represented by multiple materials and therefore the material interfaces need to be defined in terms of the differing materials at their junctions. In the next section, a functional representation of the interfaces is given to build a reprojection equation for projecting particles back onto the surface.



(a) The coarse grid.



(b) The fine grid.

Figure 1: Figures 1(a)-1(b) are examples of the closest point embedding. For all figures, the cells close to the surface are colored blue, while cells far away from the surface are colored white. Figure 1(a) is an example surface, a circle embedded in a coarse grid. Figure 1(b) displays part of the fine level of the surface from 1(a), with spacing $S = 1/3$, and the projection, visualized with an arrow, of the cell in red, $(36,24)$ to the surface location (the green point) $(40.2,23.1)$.

3.3 Multimaterial Surfacing

For multimaterial surfacing, the interface between materials, as well as the gradient near the interface, are required to project particles back onto the interface. In BioMesh3D, the multimaterial implementation is based on functional representation, where the interfaces are modeled by a function.[2,3,37] The functional model is composed of a set of N *indicator functions* $F = \{F_i | f_i : V \longmapsto R\}$ which represents N materials. A material label is assigned to a point $x \in V \iff f_i(x) < f_j(x) \forall i \neq j$. In this multimaterial model, the interface is the set of all points x, where $f_i(x) - f_j(x) = 0$, otherwise known as a *junction*. Generally for three dimensions, there are 3 types of junctions: four material junctions which are single points, three material junctions which are lines and two material junctions which are surfaces. The surface between materials is approximated with a *cell indicator function*, f_i.

To project the particle $\mathbf{p}$ on the surface of a junction between materials i and j,

$$\mathbf{p} \leftarrow \mathbf{p} - \frac{\nabla \tilde{f}_i - \nabla \tilde{f}_j}{|\nabla \tilde{f}_i - \nabla \tilde{f}_j|} \tag{3}$$

where $\nabla \tilde{f}$ is the gradient of the *cell indicator function* at $\mathbf{p}$.The derivation of the *cell indicator function* is beyond the scope of this work. For more details about the functional representation of multiple materials, see Meyer et al.[2]

4. CLOSEST POINT EMBEDDING

A reprojection step (Equation 3) is required to place the particle back on the surface when advecting a particle on an interface (Section 3.2). Unfortunately, this is problematic because in practice the projection operator is an iterative search to find the surface. Since the projection operator is a search, particles cannot be assigned to individual threads in CUDA because the threads would diverge. This reprojection search limits the performance on the GPU. To overcome this limitation, we convert the distance field into a closest point embedding, which can place the particle directly onto the surface. The preprocessing step of the closest point embedding construction is covered in Sec. 4.1. Once the closest point embedding is constructed, it is used during particle advection to place particles back on the surface (Sec. 4.2).

4.1 Constructing the Closest Point Embedding

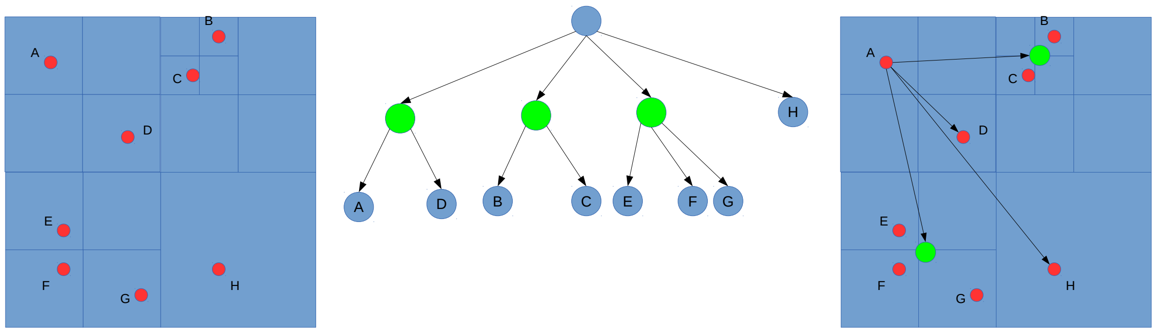
The closest point embedding is reconstructed from the distance field generated in Section 3.1 as a preprocess step. A distance field is generated by BioMesh3D, which stores the distance from a cell to the surface, but only for cells close to the surface. Figure 1(a) is an example of a distance field, where the blue cells are close to the surface and the white cells are outside the narrow band and do not store a distance to the surface. The closest point embedding stores the location on the surface that is nearest to the cell. Using Figure 1(b) as an example, the cell at $(36,24)$ is colored red and the closest location on the surface to the cell is colored green. The value stored in the distance field at cell $(36,24)$ is 4.3, which is the distance to the surface. However, the value stored in the closest point embedding at the cell $(36,24)$ is $(40.2,23.1)$.

A hierarchical grid is constructed to store the closest point embedding, similar to Auer et al.[33] The grid has two levels, a coarse level and a fine level. The coarse level is a three-dimensional grid with the same dimensions as the distance field and each cell, corresponding to the narrow band in the distance field, represents a block of sub-cells for interpolating the closest point position. The fine level is composed of the sub-cells of the blocks that are close to the surface and is stored in a one-dimensional array. An example of the coarse grid is in Figure 1(a).

To construct the closest point embedding, the cells closest to the surface are determined by looking up the corresponding cell values in the distance field. If the value is close to the surface (the blue region in Fig. 1(b)), the cell is processed. For each cell in the one-dimensional fine level, the sub-cell closest point is computed by projecting the cell position onto the surface using the Equation 3. An example is in Figure 1(b). Once the projection is complete, the closest points are stored contiguously in the fine level array. In practice, to ensure the closest point embedding matches the interfaces generated in BioMesh3D, a Catmull-ROM interpolant is used to project the cells onto the surface.[2,3]

4.2 Using the Closest Point Embedding

Once the interfaces are reconstructed into a closest point embedding, a new reprojection step is required to place particles back on the surface after the advection method. To place a particle back onto the surface with closest point embedding, a WENO4 interpolant is used to interpolate the position on the surface.[38] For every particle, $\mathbf{p}_i$ the closest point is retrieved from the closest point embedding data structure based on the position of the particle, in one-dimension. This process is repeated for the three cells surrounding the particle because the WENO4 interpolant requires three neighbors for the parabolic interpolation. These are interpolated to compute the location on the surface, $\mathbf{cp}_i$. The particle, $\mathbf{p}_i$ is placed at the location of the interpolated result, $\mathbf{cp}_i$.



(a) Particles with octree.	(b) Tree representation of 2(a)	(c) Particles and the center-of-mass.

Figure 2: Figures 2(a)-2(c) are an example of a quadtree built with the Barnes-Hut tree code, in two dimensions. Figure 2(a) has eight particles, $A-H$, with the domain subdivided into an octree. Figure 2(b) is the octree from 2(a), visualized as a tree. The blue nodes that are labeled $A-H$ are leafs as these are the quads containing the particles $A-H$ in Fig. 2(a). The nodes colored in green have the center-of-mass of its descendants. Finally, Fig. 2(c), is a spatial visualization of the particles with the domain decomposed into quads. The green points are the center-of-mass positions of the nodes, and the particle A has traversed the tree and calculated its energy from the two green nodes and directly from particles D and H.

5. PARTICLE ADVECTION WITH THE BARNES-HUT TREE CODE

Although speed-ups of up to 44x were achieved with the original isosurface GPU implementation,[6] it is not optimal. In particular, bins are processed in parallel, but the particles in each bin are processed serially. This advection scheme was chosen because each step attempts to maximize the step-size. This maximization step increases the amount of control flow required which is antithetical to the limited branching ability of the GPU. A binning structure is used to reduce the number of neighboring particles required to compute the energy and velocity. The bin size depends on the maximum sizing field value, but if the maximum sizing field value is a significant portion of the total domain then the efficiency of this parallelization strategy is diminished. Therefore, to address these issues and to increase efficiency on the GPU, the Barnes-Hut tree code acceleration structure is used for particle advection.[39]

The Barnes-Hut tree code is better suited for particle advection than the uniforming binning method on the GPU. The Barnes-Hut acceleration structure stores all particles in the domain in an octree such that each leaf in the tree has either zero or one particle. Each node links to its children nodes and also contains a representation of its children with a center-of-mass position and the total mass of all its children particles. In practice, the mass of any particle is set to one.

To construct the tree serially, each particle, $\mathbf{p}_i$, is inserted at the root node. Then the particle descends down the tree and at each node updates the center-of-mass and total mass of the node until a leaf is reached. At the leaf, child nodes are created and the center of mass of the node is updated.

Once the tree is built, it is used to calculate the force or energy of a particle. For each particle, $\mathbf{p}_i$, the tree is traversed searching for every other particle, $\mathbf{p}_j$, to calculate the energy or velocity of the particle, $\mathbf{p}_i$. However, during the tree traversal, if the center-of-mass of a node is sufficiently far away from the particle, $\mathbf{p}_i$, then the center-of-mass of the node is used as a single large particle for the energy or velocity computation of particle $\mathbf{p}_i$ and the children of the node are not traversed. The children are not traversed because the farther a particle or group of particles are away from particle $\mathbf{p}_i$ the less effect they have on the energy or velocity of $\mathbf{p}_i$. The children of the node are approximated with the center-of-mass of the node because the cotangent energy function falls off as the distance between two particles increases. For example, in Figure 2(c), the energy of particle A is computed by traversing the tree. The particle A directly computes the energy from particles D and H, but uses the center-of-mass of particles (B,C) and (E,F,G) instead of traversing the whole tree. This significantly reduces the number of particle-particle energy and velocity computations.

To determine whether a node is “far” away from a particle, a user defined value, θ, is used. This is a threshold on the ratio between the size of the node that a particle, $\mathbf{p}_i$ is in, where the size is the edge length defined by the octree level and the distance from the center-of-mass of the node to $\mathbf{p}_i$. In practice, we set θ to 0.75.

The Barnes-Hut tree code eliminates two problems with the Kim et al. method.[6] First, the tree code eliminates the need to bin the computational space by the maximum sizing field value. Secondly, traversing the tree with the Barnes-Hut tree code on the GPU significantly reduces the control flow. This makes the Barnes-Hut tree code a better algorithm for the GPU than the serial update method. Although all the particles move at the same time with the Barnes-Hut algorithm, if the step is small enough then the particles will still converge to a good solution.

The implementation used in this work to construct and traverse the tree code on the GPU is very similar to Bédorf et al.[39] with two changes. First, the energy and velocity calculations are done with the cotangent energy functions in Section 3.2. The second change is to the velocity calculation. Once the tree code traversal has calculated the velocity for all the particles, $\mathbf{p}_i$, the velocities are multiplied by the sizing field value at the location of the particle, $\mathbf{p}_i$. If the sizing field value is less than zero, i.e. the particles are packed closely because of high curvature, the velocity length is reduced. Likewise, if the sizing field value is large, which indicates an area of low curvature, the velocity length is increased by the sizing field value. By increasing or decreasing the velocity length by the sizing field, the particles will take smaller steps in areas of high curvature and take larger steps in areas of low curvature.

6. INITIAL SEEDING AND ADDING AND DELETING PARTICLES

In previous sections, the energy, velocity, advection and reprojection of the particles using the closest point embedding have been discussed. However, the number of the particles to correctly represent the sampling of the surface has yet to be explored. Unfortunately, this number is not known *a priori*. Therefore, the particle system is initially seeded with enough particles to sufficiently cover the surface and then particles are iteratively advected, added and/or deleted until all particles in the system are close to the ideal energy. The ideal energy is the target energy used to place particles and is computed from the equidistant hexagonal placement of neighboring particles, which is an ideal packing of particles in two-dimensions.[14] If all particles are close to the ideal energy, we consider the surface to be properly sampled. Getting the initial particle count as well as placement of the initial particles close to the final solution helps reduce the number of addition and deletion steps, which decreases the time required for the particle system to converge.

Table 1: The initial and final particle count for the *pig* and *head* datasets, as well as the mean radius ratio of the final mesh.

	CPU			Red-Black Update			Closest Point		
Dataset	Initial #	Final #	Quality	Initial #	Final #	Quality	Initial #	Final #	Quality
pig	151,141	342,231	0.93	151,141	347,906	0.92	286,346	350,498	0.93
head	614,344	1,429,517	0.93	614,344	2,159,347	0.90	1,283,799	2,145,468	0.93

6.1 Initial Seeding

BioMesh3D uses any grid point close to the surface as the initial seeding and then the seeds are projected onto the surface to ensure the surface is fully represented.[3] Unfortunately, this does not take curvature into account. To address this issue, a new seeding strategy was developed where regions of high curvature are seeded with more particles while areas of low curvature are seeded with fewer particles. This is a three step process. First, grid locations close to the surface are added to the seed set. Second, randomly seeded particles are added to the seed set in regions of high curvature. Third, seed particles from the seed set are added to the system only if needed.

Grid points close to the surface are the initial seed set. More particles are added to regions of high curvature, i.e. the sizing field value is less than 1.0. The number of particles added is the inverse of the sizing field value, $1/sizingfield \cdot C$, where C is a user defined constant, to control how many new random seeds are generated. In practice, $C = 2$.

In areas of low curvature (the sizing field value > 1.0), the initial seed set can oversample the surface. To prevent initial oversampling, as the seed set is generated, for each seed, its energy, E, is computed. If the energy is significantly higher than the ideal energy then the seed is not eliminated and not added to the final seed set. In practice, a serial Barnes-Hut tree is generated on the CPU to compute the energy. By eliminating seeds with high energy, the surface is not over sampled with initial particles.

This new seeding approach is closer to the final solution, both in terms of the number and position of particles. This reduces the number of iterations required to achieve an ideal sampling, which in turn decreases the amount of time the particle system takes to converge.

6.2 Adding and Deleting Particles

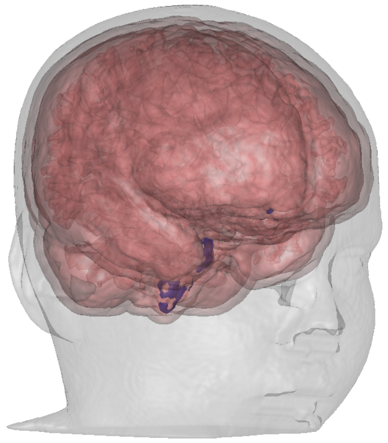
The total number of particles to correctly sample the surface is not known *a priori* and adding and deleting particles based on their energy is important to correctly sample the surface. This two step process is used to speed-up the particle system as well as prevent mass addition and deletion of particles. The first step determines whether adding and deleting is appropriate and the second step adds and deletes particles from the system.

In the first step the energy of the particles is computed and particles with high energy are flagged for deletion while particles with low energy are flagged to have a new particle placed close to them. If the number of added and deleted particles is low then the particle system has the correct number of particles to represent the surface. If the number of added or deleted particles is high, then particles the system continues to the second step. Because this first step is done on the GPU, it can quickly determine whether or not the slower second step should proceed.

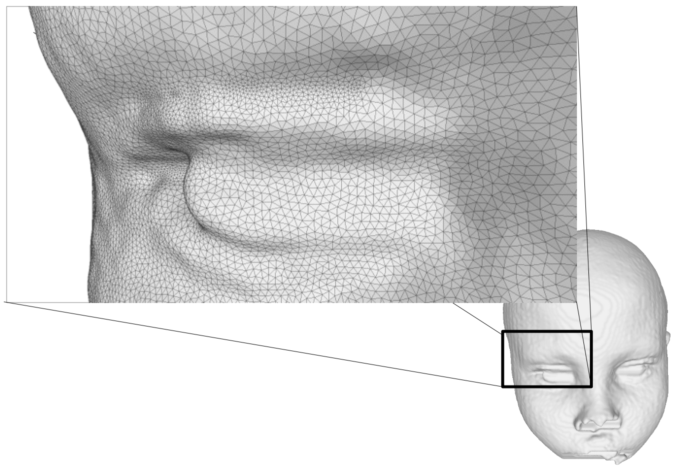
The second step adds and deletes particles from the particle system. Because the first step is done in parallel, i.e. each particle's energy is computed simultaneously, mass addition or deletion of particles can occur. For example, if two particles are very close together, then the energy of both particles would be high and they would both be marked for deletion. However, only one should be deleted while the other one remains in the particle system.

To prevent the mass addition and deletion of the particles, this step is serialized and performed on the CPU. All the particles that were flagged for deletion are removed from the particle system. Then, for each particle that was deleted, $\mathbf{p}_{del}$ is added back into the particle system and its energy is calculated. If the energy is less than 4 times the ideal energy, then $\mathbf{p}_{del}$ is added back to the particle system, otherwise it is permanently removed. By checking the energy as the deleted particles are added back into the tree and only adding the ones that have low energy, mass deletion of particles is prevented.

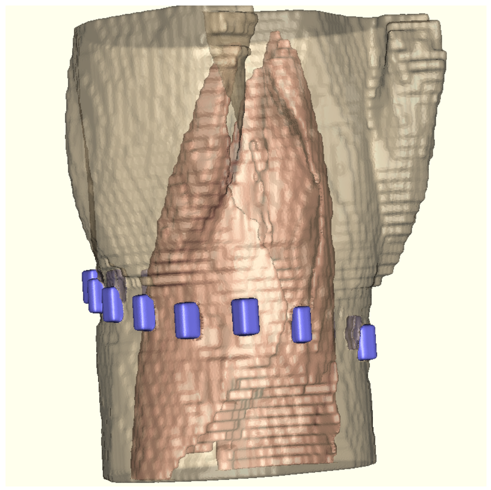
Next, for each particle that was flagged as low energy, $\mathbf{p}_{le}$ a temporary particle, $\mathbf{p}_{tmp}$ is added close to $\mathbf{p}_{le}$ and the energy of $\mathbf{p}_{tmp}$ is computed. If the energy is not too high, then the temporary particle is permanently added to the particle system. Checking the energy prevents mass addition of new particles into the system. In practice, a Barnes-Hut tree is constructed on the CPU to calculate energy for adding and deleting particles.



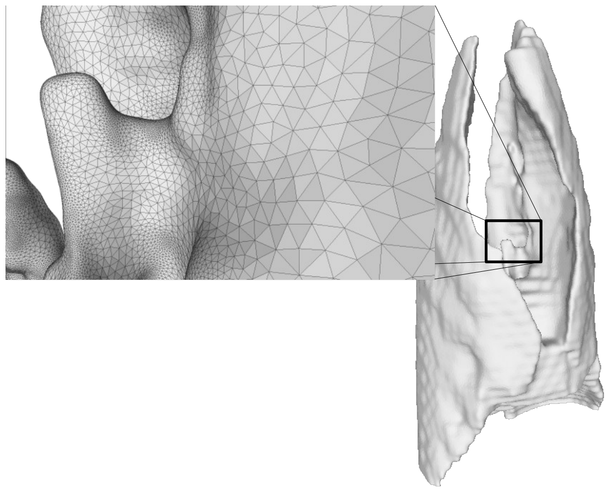
(a) The *head* dataset.



(b) The mesh of the *head* dataset



(c) The *pig* dataset.



(d) The mesh of the *pig* dataset.

Figure 3: The *head* and *pig* datasets. Figure 3(a) shows all the material interfaces of the *head* dataset, and Figure 3(b) shows the triangles of the skin surface. Figure 3(c) shows all the material interfaces of the *pig* dataset, while Fig. 3(d) shows some of the triangles on the "lung" material of the dataset.

Table 2: A comparison of time (in seconds) to complete particle advection for the closest point embedding with the CPU, the red-black implementation and the Barnes Hut tree code. The datasets are the *pig* torso and human *head* volumes.

	Time (secs)			Speed-up		
Dataset	CPU	Red-Black	Barnes Hut	CPU vs RGBS	BH vs CPU	BH vs RGBS
pig	5,056	1,312	472	3.9x	10.7x	2.8x
head	42,725	7,445	1,694	5.7x	25.2x	4.4x

Table 3: Timing results (in seconds) for the *GenerateDistanceVolumes*, *GenerateSizingFields*, closest point grid generation and *SurfaceExtraction* (Sec. 3) for the *pig* and *head* datasets.

	Preprocessing			Postprocessing	
Dataset	Distance Volume	Sizing Field	Closest Point	Extract	Total Time
pig	60	723	6	17	800
head	1,135	3,129	35	70	4,334

7. RESULTS

In this section, the timing results and the quality of the extracted meshes are discussed. All GPU tests were performed on an Intel Xeon E5-2640 with 32GB of RAM with an Nvidia K20 Tesla card with 5GB of RAM using CUDA 5.0. The CPU tests were performed on an Intel Xeon X5550 with 24GB of RAM. Two datasets, a human *head* and *pig* torso are used. The *head* dataset (Figure 3(a)) is a four material volume with a size of $(199, 250, 249)$ and the *pig* dataset is a five material volume with a size of $(136, 136, 136)$ (Figure 3(c)). The initial and final particle count are in Table 1. With the new seeding method, the *pig* dataset begins with 286,346 particles while the *head* dataset begins with 1,283,799 particles. BioMesh3D initial seeding for the *pig* is 151,141 particles, and the *head* dataset begins with 614,344 particles. The *pig* data has approximately 350,000 particles when the particle system finishes while the *head* dataset has approximately 2.1 million particles when the particle system finishes. Both datasets were run with the maximum sizing field value set to 1.0.

7.1 Timing

The timing results of the particle advection for extracting biological multi-material volume data are in Table 2. For this timing comparison, the Kim et al. GPU implementation was extended to extract multiple materials.[6] The new closest point embedding with the Barnes Hut acceleration structure is 2.8 times faster than the red-black update on the GPU for the *pig* and 4.4 times faster for the *head* dataset. Further, the closest point with Barnes-Hut tree code is 10.7 times faster and 25.2 times than the CPU implementations for the *pig* and *head* datasets, respectively. These are significant performance increases over the previous CPU and GPU implementation.

Table 3 contains the timing results for preprocessing the multimaterial volume: generating the distance fields and sizing fields as well as generating the closest point embedding. Further, the timing results for extracting the surface as examined in Sec. 3 is included. The distance and sizing fields generated with BioMesh3D are shared with the CPU, the red-black and closest point implementations. Figure 4 combines the timing results for the distance field, sizing field and surface extraction from Table 2 and 3 into a normalized stacked chart. Each bar is stacked with distance field, sizing field, particle advection and mesh extraction timing results, from bottom to top respectively. The closest point grid generation constitutes less than one percent of the preprocess time and has been omitted from the chart.

With the *pig* dataset, the particle advection section takes 86% of the total time to extract the mesh on the CPU, a significant portion of the total time to extract the mesh. With the red-black implementation, the advection takes 63% of the total time. Although an improvement over the CPU, the particle advection still requires 1.8x more time than generating the sizing field, the segment that takes the second most amount of time. Finally, the closest point embedding with Barnes-Hut tree code is 37% of the total time to extraction the mesh, which is a smaller portion of the total time than the sizing field generation, which is 57% of the total time. This increase in performance is illustrated in Fig. 4. The three bars on the left are the timing results for *pig* dataset for the CPU, the red-black and closest point implementations. These normalized graphs show that particle advection takes a significant portion of the time on the CPU. With the closest point embedding though, particle advection takes less time than the sizing field generation. With the Barnes-Hut tree code, particle advection is no longer the bottleneck for extracting conformal meshes from multimaterial data.

With the *head* dataset, the particle advection takes 91% of the total time to extract the mesh on the CPU. With the red-black implementation, the advection takes 63% of the total time, but still needs 2.4x more time than the segment that takes the second most time, the sizing field generation. Finally, the closest point embedding with Barnes-Hut tree code is 28% of the total time and is almost twice as fast the sizing field generation, which is 54% of the total time. The *head* dataset timing is also in Fig. 4, where the three bars on the right are the timing results for the *head* dataset for the CPU, the red-black and closest point implementations. Like the *pig* dataset, these normalized bars of the *head* dataset illustrate that particle advection takes a significant portion of the time on the CPU. With the closest point embedding though, particle advection takes less time than the sizing field generation. Again, as with the *pig* dataset, particle advection is no longer the bottleneck in the particle-based multimaterial meshing pipeline with the closest point embedding and the Barnes-Hut tree code.

The Barnes-Hut tree code with closest point embedding is up to 4.4x faster than the fastest known GPU particle advection for multimaterial mesh extraction. Further, it is up to 25.2x faster than BioMesh3D. This new technique removes the largest bottleneck of the multimaterial, conformal mesh extraction pipeline and would facilitate the adoption of the particle system pipeline in the biomedical community.[40]

7.2 Quality

To measure the quality of the mesh, the multimaterial triangular mesh is constructed using the particle locations. With the triangular mesh, the inscribed/circumscribed radius ratio of every triangle in the mesh is calculated and averaged over the entire mesh. A numerical value of 0.90 or greater for the radius ratio is considered to be an adequate triangular mesh while a value over 0.92 is considered to be a high quality triangular mesh and a good starting point for generating a good tetrahedral mesh. The *head* and the *pig* dataset have an average inscribed/circumscribed radius ratio of 0.93. These are better than the ratios of the meshes from the red-black update scheme. Further, both the *pig* and *head* dataset average radius ratio are above 0.92 and considered to be high quality triangular meshes.

By eliminating the iterative search for the surface with the closest point embedding and using a flexible acceleration scheme, the Barnes-Hut tree code, the new particle update scheme works better on the GPU. This new technique for particle advection for mesh extraction is faster than the previous GPU implementation, and generates high quality triangular meshes.

8. CONCLUSION

In this paper we have presented a new isosurface extraction algorithm with the closest point embedding. This new technique, coupled with a GPU Barnes-Hut tree code, is used for curvature adaptive, multimaterial mesh extraction from labeled volume data. The closest point embedding is a faster method for the GPU because the reprojection step is no longer an iterative search. Each particle can be assigned to a thread without the need for an iterative search for the surface in the reprojection step. Further, the Barnes-Hut tree code is better suited for particle advection on the GPU because instead of maximizing the step for each particle, small velocity steps are taken to reach an optimal particle configuration on

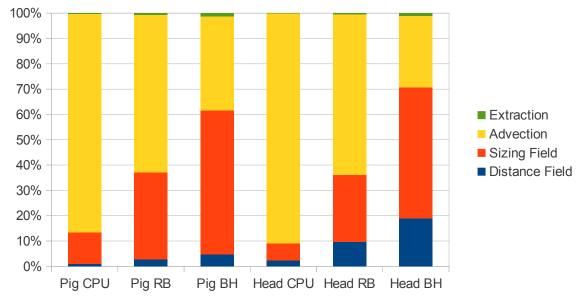


Figure 4: A normalized chart of the full timing results for the *pig* and *head* datasets from Tables 2 and 3. Each bar is the normalized time to extract the mesh (generating the distance field, generating the sizing field, advecting the particles and extracting the mesh) using the CPU, the red-black implementation and the Barnes-Hut tree code.

the surface. These small velocity steps remove much of the control flow that hindered the performance in previous GPU implementations of particle systems for surface extraction.

With the *pig* and *head* test datasets, the closest point embedding with Barnes-Hut tree code is faster than any known particle system for multimaterial mesh extraction. The speed-up is transformative for biomedical work such as Electrical Impedance Tomography (*EIT*) Imaging of the lung.[40]

Looking towards the future, although the closest point embedding generated in this work is a sparse, narrow band around the isosurface, an adaptive closest point embedding would be useful to save more memory on the GPU. Particle advection with the closest point embedding can be extended to other uses such as surface flow visualization.[41]

ACKNOWLEDGMENTS

The authors would like to thank Darrell Swenson for his feedback on BioMesh3D as well as the *head* dataset. This research was supported by the DOE, NNSA, Award DE-NA0002375: (PSAAP) Carbon-Capture Multidisciplinary Simulation Center, the DOE SciDAC Institute of Scalable Data Management Analysis and Visualization DOE DE-SC0007446, NSF ACI-1339881, and NSF IIS-1162013.

REFERENCES

[1] Meyer, M., Kirby, R., and Whitaker, R., "Topology, accuracy, and quality of isosurface meshes using dynamic particles," *IEEE Transactions on Visualization and Computer Graphics (Visualization 2007)* **13**(6), 1704–1711 (2007).

[2] Meyer, M., Whitaker, R., Kirby, R., Ledergerber, C., and Pfister, H., "Particle-based sampling and meshing of surfaces in multimaterial volumes," *IEEE Transactions on Visualization and Computer Graphics (Visualization 2008)* **14**(6), 1539–1546 (2008).

[3] BioMesh3D: Quality Mesh Generator for Biomedical Applications. Scientific Computing and Imaging Institute (SCI).

[4] Swenson, D., Levine, J., Fu, Z., Tate, J., and MacLeod, R., "The effect of non-conformal finite element boundaries on electrical monodomain and bidomain simulations," *Computing in Cardiology* (37), 97–100 (2010).

[5] Swenson, D. private communication (2012).

[6] Kim, M., Chen, G., and Hansen, C., "Dynamic particle system for mesh extraction on the gpu," in [*Proceedings of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units*], *GPGPU-5*, 38–46, ACM, New York, NY, USA (2012).

[7] Kolb, A., Latta, L., and Rezk-Salama, C., "Hardware-based simulation and collision detection for large particle systems," in [*Proc. Graphics Hardware*], 123–131 (2004).

[8] Kipfer, P., Segal, M., and Westermann, R., "Uberflow: a GPU-based particle engine," in [*HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*], 115–122, ACM Press, New York, NY, USA (2004).

[9] Kruger, J., Kipfer, P., Kondratieva, P., and Westermann, R., "A particle system for interactive visualization of 3d flows," *IEEE Transactions on Visualization and Computer Graphics* **11**, 744–756 (November 2005).

[10] Kolb, A. and Cuntz, N., "Dynamic particle coupling for GPU-based fluid simulation," in [*Proc. of the 18th Symposium on Simulation Technique*], 722–727 (2005).

[11] Goswami, P., Schlegel, P., Solenthaler, B., and Pajarola, R., "Interactive SPH simulation and rendering on the GPU," in [*Proceedings ACM SIGGRAPH Eurographics Symposium on Computer Animation*], 55–64 (July 2010).

[12] Witkin, A. and Heckbert, P., "Using particles to sample and control implicit surfaces," *Computer Graphics* , 269–278 (July 1994). Proceedings of SIGGRAPH'94.

[13] Crossno, P. and Angel, E., "Isosurface extraction using particle systems," in [*IEEE Visualization 97*], 495–498 (1997).

[14] Meyer, M., Georgel, P., and Whitaker, R., "Robust particle systems for curvature dependent sampling of implicit surfaces," in [*Proceedings of the International Conference on Shape Modeling and Applications (SMI)*], 124–133 (June 2005).

[15] Bronson, J., Levine, J., and Whitaker, R., "Particle systems for adaptive, isotropic meshing of cad models," *Proceedings of the 19th International Meshing Roundtable* , 279–296 (October 2010).

[16] Bloomenthal, J. and Ferguson, K., "Polygonization of non-manifold implicit surfaces," (1995).

[17] Nielson, G. M. and Franke, R., "Computing the separating surface for segmented data," in [*Proceedings of the 8th conference on Visualization '97*], *VIS '97*, 229–233, IEEE Computer Society Press, Los Alamitos, CA, USA (1997).

[18] Hege, H.-C., Seebass, M., Stalling, D., and Zöckler, M., "A generalized marching cubes algorithm based on non-binary classifications," Tech. Rep. SC-97-05, ZIB, Takustr.7, 14195 Berlin (1997).

[19] Bonnell, K. S., Joy, K. I., Hamann, B., Schikore, D. R., and Duchaineau, M., "Constructing material interfaces from data sets with volume-fraction information," in [*Proceedings of the conference on Visualization '00*], *VIS '00*, 367–372, IEEE Computer Society Press, Los Alamitos, CA, USA (2000).

[20] Reitinger, B., Bornik, E., and Beichel, R., "Constructing smooth nonmanifold meshes of multi-labeled volumetric datasets," in [*In Proc. of WSCG 2005*], 227–234, Press (2005).

[21] Zhang, Y., Hughes, T. J. R., and Bajaj, R. L., "Automatic 3d mesh generation for a domain with multiple materials," in [*In IMR*], 367–386 (2007).

[22] Ju, T., Losasso, F., Schaefer, S., and Warren, J., "Dual contouring of hermite data," *ACM Trans. Graph.* **21**, 339–346 (July 2002).

[23] Oudot, S., Rineau, L., and Yvinec, M., "Meshing volumes bounded by smooth surfaces," in [*Proc. 14th Internat. Meshing Roundtable*], 203–219 (2005).

[24] Pons, J.-P., Sgonne, F., Boissonnat, J.-D., Rineau, L., Yvinec, M., and Keriven, R., "High-quality consistent meshing of multi-label datasets," in [*Information Processing in Medical Imaging*], Karssemeijer, N. and Lelieveldt, B., eds., *Lecture Notes in Computer Science* **4584**, 198–210, Springer Berlin Heidelberg (2007).

[25] Anderson, J. C., Garth, C., Duchaineau, M. A., and Joy, K. I., "Discrete multi-material interface reconstruction for volume fraction data," *Computer Graphics Forum* **27**(3), 1015–1022 (2008).

[26] Prilepov, I., Obermaier, H., Deines, E., Garth, C., and Joy, K., "Cubic gradient-based material interfaces," *Visualization and Computer Graphics, IEEE Transactions on* **19**, 1687–1699 (Oct 2013).

[27] Ruuth, S. J. and Merriman, B., "A simple embedding method for solving partial differential equations on surfaces," *J. Comput. Physics* **227**(3), 1943–1961 (2008).

[28] Macdonald, C. B. and Ruuth, S. J., "The implicit closest point method for the numerical solution of partial differential equations on surfaces," *SIAM J. Scientific Computing* **31**(6), 4330–4350 (2009).

[29] Macdonald, C. B. and Ruuth, S. J., "Level set equations on surfaces via the closest point method," *J. Sci. Comput.* **35**(2-3), 219–240 (2008).

[30] März, T. and Macdonald, C. B., "Calculus on surfaces with general closest point functions," *SIAM J. Numerical Analysis* **50**(6), 3303–3328 (2012).

[31] Tian, L. L., Macdonald, C. B., and Ruuth, S. J., "Segmentation on surfaces with the closest point method," in [*ICIP*], 3009–3012 (2009).

[32] Hong, Y., Zhu, D., Qiu, X., and Wang, Z., "Geometry-based control of fire simulation," *Vis. Comput.* **26**, 1217–1228 (Sept. 2010).

[33] Auer, S., Macdonald, C. B., Treib, M., Schneider, J., and Westermann, R., "Real-time fluid effects on surfaces using the closest point method," *Comput. Graph. Forum* **31**(6), 1909–1923 (2012).

[34] Si, H., *TetGen User Manual* (January 2006).

[35] Gonzalez, R. C. and Woods, R. E., [*Digital Image Processing*], Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd ed. (2001).

[36] Williams, J. and Rossignac, J., "Tightening: curvature-limiting morphological simplification," in [*Proceedings of the 2005 ACM symposium on Solid and physical modeling*], *SPM '05*, 107–112, ACM, New York, NY, USA (2005).

[37] Pasko, A., Adzhiev, V., Sourin, A., and Savchenko, V., "Function representation in geometric modeling: Concepts, implementation and applications," (1995).

[38] Edwards, E. and Bridson, R., "A high-order accurate particle-in-cell method," *International Journal for Numerical Methods in Engineering* **90**(9), 1073–1088 (2012).

[39] Bédorf, J., Gaburov, E., and Zwart, S. P., "A sparse octree gravitational n-body code that runs entirely on the gpu processor," *J. Comput. Physics* **231**(7), 2825–2839 (2012).

[40] Salz, P., Reske, A., Wrigge, H., Scheuermann, G., and Hagen, H., "Improving electrical impedance tomography imaging of the lung with patient-specific 3d models," 49–53, Eurographics Association, Leipzig, Germany (2013).

[41] Li, G.-S., Tricoche, X., Weiskopf, D., and Hansen, C. D., "Flow charts: Visualization of vector fields on arbitrary surfaces," *IEEE Trans. Vis. Comput. Graph.* **14**(5), 1067–1080 (2008).