7

Sparse particle-based multi-material volume meshing for conformal 3D meshes

Daniel F. Malan, Christian Kehl,
Elmar Eisemann, Edward R. Valstar

Abstract

We present a conformal volume mesher that creates tetrahedral meshes from a multi-material image volume. The raison d'être for our mesher is the importance yet difficulty of creating suitable volume meshes in applications such as three dimensional biomechanical finite-element analysis. The input to our method is a segmented multi-material image volume defined on a regular grid such as those generated by Computed Tomography or Magnetic Resonance Imaging. The first step converts the volume to a set of dense topology-preserving surface meshes. During intermediate steps seed particles are defined and optimized so that they will ultimately serve as vertices of the output mesh. The final density of these optimized seed particles can be controlled by the user. The optimized seed particles are inserted as vertices into the dense surface meshes and all other mesh vertices are iteratively removed, until only the seed particles' vertices remain. Our output mesh uses fewer elements and/or obtains lower geometric approximation errors than comparable Delaunay-triangulation-based methods, as those methods need to rely on additional and restrictive sampling criteria

7.1 Introduction

Finite element simulation is useful in many medical contexts, including orthopaedics [Bessho et al., 2009, Reggiani et al., 2007, Schileo et al., 2007, Taddei et al., 2006], oncology [Samani et al., 2001], cardiology [Sankaran et al., 2012] and neurology [Joldes et al., 2010]. Often, the meshes underlying these simulations are derived from segmented image volumes. In many cases, multi-material data is involved, which can describe distinct material regions, e.g., a metal hip prosthesis surrounded by cement embedded in the femur, the latter of which consists of both dense cortical and porous trabecular bone [Andreykiv et al., 2012]. For such cases, multiple sub-meshes need to be derived. These sub-meshes are often connected and interact with each other via shared boundaries. For a faithful simulation of interaction across boundaries, the derived sub-meshes should be conformal to each other, i.e., mesh vertices and edges on shared interface boundaries must coincide.

In addition to mesh conformity, geometric mesh quality plays an important role in finite element simulations. The number of mesh elements, the geometric accuracy of the mesh, as well as the aspect ratios of individual mesh elements influence performance and accuracy of simulations that build upon them [Babuska and Aziz, 1976, Burkhart et al., 2013].

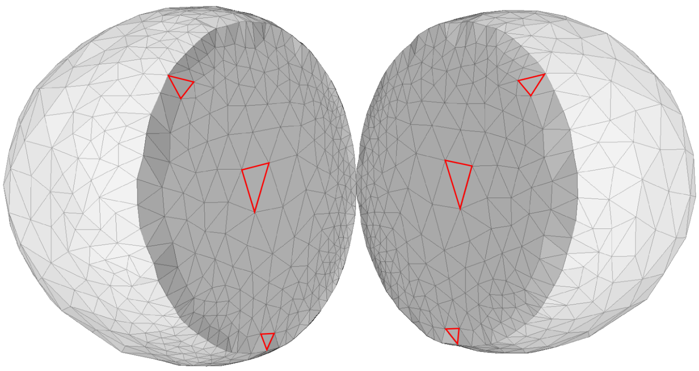


Figure 7.1 – Mesh conformity means that all vertices and all edges on the shared boundary are identically defined: three of the conforming pairs of triangles are highlighted.

In this paper, we propose the Conformal Volume Mesher (CVMesh) a novel meshing algorithm that constructs tetrahedral meshes from multi-material image volumes. It starts by transforming a multi-material segmented image volume into a set of dense initial meshes. Seed particles are then distributed and their positions optimized on the isosurfaces corresponding to these dense meshes using the method of [Meyer et al., 2007, Meyer et al., 2008]. Once optimized, the particles are integrated as vertices into the dense mesh. Afterwards, all original mesh vertices are iteratively removed, keeping only the newly-added vertices. Using this method, a set of water-tight closed surface meshes is maintained for each material in every intermediate step.

Fig. 7.1 shows the meaning of mesh conformity using two partial spheres that touch in a shared plane. This figure emulates Fig.7 in [Meyer et al., 2008], with the notable difference that it shows output of CVMesh after using a segmented voxel grid with two distinct foreground labels, while Fig.7 in [Meyer et al., 2008] used synthetically constructed regular isosurfaces as input.

Our solution endeavours to provide a volumetric mesh with low geometric approximation error while offering direct user control over the number of mesh elements. Further, the definition of the resulting meshes' surface vertices by a unique set of optimized "particles" ensures that shared boundaries are kept conformal.

7.2 Related Work

The problem of generating a volumetric tetrahedral mesh can be reduced to the problem of generating a closed triangle surface mesh, since the latter may be reliably converted into the former by adding interior nodes only. This step does not modify the surface triangles in any way [George et al., 1991].

Several established approaches exist to generate triangular surface meshes from a single material (i.e. binary) image volume. Marching Cubes (MC) [Lorensen and Cline, 1987] is one of the most well-known and widely-used algorithms, as it can quickly generate topologically correct, closed surfaces. Care must however be taken in choosing a reliable MC implementation to avoid the possibility of topological artefacts [Etiene et al., 2012]. The main limitation of MC is its uniformly high triangle density that directly corresponds to the input image voxel resolution. When applied to a high resolution voxel grid such as those generated by Computed Tomography (CT) or Magnetic Resonance Imaging (MRI), the resulting surface meshes must typically be decimated to obtain a tractable number of elements [Garland and Heckbert, 1997, Knapp, 2002].

A multi-material extension of MC exists [Wu and Sullivan, 2003], but as holds for the original MC algorithm, it produces high triangle counts. These dense multi-material meshes are conformal, but standard decimation techniques cannot be directly applied as these operate on each sub-mesh individually and do not guarantee conformity across multi-material interfaces. Native multi-material meshing algorithms based on adaptive decimation exist [Wang, 2007, Kahnt et al., 2011, Young et al., 2008] and form the basis of commercial meshing software such as Simpleware +FE, Visage Amira and Materialise Mimics [Sim, 2013, Ami, 2013, Mat, 2013].

As an alternative to decimating dense precursor meshes, other meshing algorithms rely on direct Delaunay triangulation for multi-material mesh generation [Boltcheva et al., 2009, Kahnt et al., 2011]. These algorithms first compute the desired vertex distribution and then reconstruct the triangulated surface from the vertices. For non-degenerate vertex distributions the Delaunay triangulation is unique, ensuring conformity of vertices and edges on multi-material interfaces. To ensure topologically-correct surfaces, an ε-sampling requirement needs to be satisfied. This requirement states that the maximum Euclidean distance between mesh vertices, expressed as a fraction ε of the local feature size (LFS), should not exceed a certain value [Boissonnat and Oudot, 2005]. The value of $\varepsilon = 0.06$ has theoretically been proven to be sufficient for 3D applications [Amenta and Bern, 1998], while experiments suggest that $\varepsilon = 0.5$ may be a more realistic loose upper bound for practical applications [Meyer et al., 2007].

[Meyer et al., 2007] used a particle optimization algorithm to distribute mesh vertices and have developed their algorithm to enable meshing of multiple materials [Meyer et al., 2008]. The advantage that particle based methods have over direct mesh refinement is that it abstracts the shape that need to be meshed as an implicit surface, which allows for scale invariance. Particles are optimized independent of the underlying image grid in accordance to an energy function. The final particle distribution corresponds to the vertices of the output mesh, which are Delaunay-triangulated to generate the algorithm's output. One advantage of their approach is the slowly varying vertex spacing that is obtained, which results in good triangle aspect ratios. Their particle distribution depends on the meshed object's level-set representation and is largely independent of the underlying image resolution.

Meyer et al.'s algorithm has been integrated into the BioMesh3D [Callahan et al., 2007] package, which is distributed as part of the open-source SciRun software package [Bitter et al., 2007]. While it offers user control over geometric smoothing and triangle count, the ε-sampling requirement remains mandatory just as for all Delaunay triangulation schemes. This requirement leads to a trade-off between the minimum triangle density and the minimum allowable amount of geometric smoothing. We found that surface details are quickly smoothed away and simultaneously the triangle count can remain high, even when using the maximum allowable value of $\varepsilon = 0.5$. Our findings in this regard are supported by the literature [Fayolle and Pasko, 2012]. The approach we propose is not restricted by the strict ε-sampling requirement inherent to Delaunay triangulation schemes, which enables our algorithm to better conserve sharp features. This is possible as violating the ε-sampling requirement allows a more liberal redistribution of vertices with the goal of reducing surface approximation errors.

7.3 Algorithm

Our algorithm follows the general approach of [Meyer et al., 2008], but introduces a modification in the way that the triangle mesh is generated from the optimized particle cloud. This modification allows us to obtain sparser meshes for a given minimum feature size. Not being forced to increase the minimum feature size as a strict function of particle spacing allows lower geometrical errors, lower mesh element counts, or sometimes both.

Our algorithm starts with a multi-material segmented image volume defined

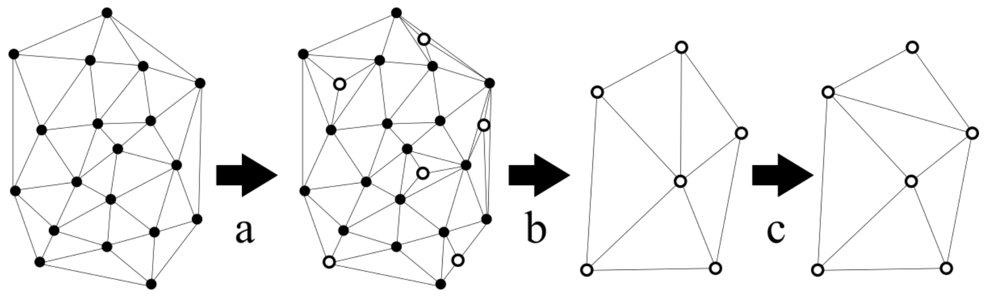


Figure 7.2 – Transformation of a dense input mesh to a topologically identical sparse output mesh by the sequence of a) augmentation, b) decimation and c) edge flipping.

in a regular voxel grid, i.e. each and every voxel has been assigned one and only one categorical label. The number of possible labels has to be at least two – a binary segmentation consisting of foreground and background – but otherwise may be any integer.

Using the same approach of [Meyer et al., 2008], each material label is converted to a grayscale volume whose zero level-set isosurface corresponds to the material's boundary. Each material's level-set surface is then smoothed to remove unwanted noise or spurious details.

The smoothed level-set isosurfaces are used to compute each material's medial axis [Hesselink and Roerdink, 2008]. These medial axes, together with user-definable parameters ε and δ are used to construct a *sizing field* for each material label – we refer the reader to [Meyer et al., 2007] for details on its definition. In our algorithm the value of ε does not need to satisfy the Delaunay ε-sampling requirement [Amenta and Bern, 1998] that would otherwise have been applicable, and may therefore be chosen to exceed a value of 0.5.

The smoothed level-set isosurfaces serve additional purposes - firstly they are each converted to a dense watertight surface mesh via MC. These meshes have topological and geometrical precision, but also a very high number of triangles that make them unsuitable for direct finite element simulation. The final surface meshes that our algorithm will construct will be a refinement of these meshes, but with a different, sparser, distribution of vertices. The distribution of this final set of vertices is determined by the output of an iterative "particle optimization" process, where each "particle" represents a mesh vertex. These particles originate from a collection of "seed particles" that are initially uniformly and densely distributed on the smoothed level-set isosurfaces. Where material labels share a boundary, these seed particles are shared by assigning them to both surfaces simultaneously. Hereby, no vertex duplication or discrepancies occur, which is important to achieve conformity. The particle positions are optimized in an iterative process where they are treated as particles in a mechanical system that have repulsive and attractive forces relative to each other. Particles do not occupy discrete grid positions; each is free to move on the level-set isosurface boundary to which is was assigned. As optimization progresses particles are removed where their density exceeds the requirements set by the sizing field, resulting in a particle count that is typically much lower than the initial number of seed particles.

Once the particle distribution has been optimized, we use these to define the output mesh in a three-step approach to derive watertight meshes that are then converted to tetrahedral meshes.

First, all particle-vertices are iteratively inserted as vertices into the dense MC-generated isosurface meshes, i.e. every particle is inserted as a vertex into every material boundary mesh on which it is located. In the second phase only the newly-added vertices are kept, while all others are iteratively removed. This process is illustrated in Fig. 7.2. For conformity, not only vertices, but also edges need to coincide. Edge conformity is enforced by a final series of edge flips in the constructed meshes.

The topology of the output mesh is not deduced from the vertex distribution but is determined by the dense MC-generated mesh. An important feature of our algorithm is that each vertex insertion or deletion step maintains watertightness and topological correctness of all the meshes that are involved. This fact is crucial, as it allows us to define the final mesh's vertices by a vertex-set that does not respect the ε-sampling requirement. No explicit constraint is placed on the edges that are created each time a vertex is inserted or when a vertex is removed. This freedom allows our algorithm to choose local triangulations that maintain healthy aspect ratios for the newly created triangles, while minimizing the geometric Hausdorff distance to the original dense MC surface mesh.

Algorithm 7.3.1: CVMESH($LabelVolume\ I, r, \varepsilon, \delta$)

```
for each $material\ i \in I$
    do $\phi_i \leftarrow$ level-set isosurface of $i$
       $\tilde{\phi}_i \leftarrow$ smoothed $\phi_i$
       $T_i \leftarrow$ dense triangle surface mesh of $\tilde{\phi}_i$
       $m_i \leftarrow$ medial axis of $\tilde{\phi}_i$
       $s_i \leftarrow$ sizing field as a function of $m_i$, $\varepsilon$ and $\delta$
for all $\tilde{\phi}, s$
    $P_{seed} \leftarrow$ Uniformly distribute seed particles on $\tilde{\phi}$
    $P_{opt} \leftarrow$ Optimize distribution of $P_{seed}$ using $\tilde{\phi}$, $s$
for each $material\ i \in I$
    do $T'_i \leftarrow$ for each $particle\ p \in P_{opt}$
                do if $p$ belongs to $i$
                     then insert $p$ into mesh $T_i$ as vertex
for each augmented boundary mesh $T'_i$
    do $T''_i \leftarrow$ for each $vertex\ v \in T'_i$
                do if $v \notin P$
                     then remove vertex from $T'_i$
Flip edges to ensure edge conformity between meshes
```

7.4 Implementation

CVMesh is implemented as a collection of modules that run in the open-source DeVIDE Runtime Environment [Botha and Post, 2008]. Individual components use a combination of Python and C++, with extensive use of the Visualization Toolkit (VTK) [Schroeder et al., 1996], Teem [Tee, 2013] and Vispack [Ross T. Whitaker, 2013] libraries. The final conversion step from a mesh to a tetrahedral volume is performed using Tetgen [Si, 2006].

The initial input surfaces for each material label are smoothed using the Tightening algorithm of [Williams and Rossignac, 2007]. Dense isosurface meshes are computed using VTK's Marching Cubes algorithm. Computation of the 3D medial axis is a computationally hard problem [Coeurjolly et al., 2008] and can take up an inordinate amount of time to compute. For this we used the Integer Medial Axis (IMA) algorithm of [Hesselink and Roerdink, 2008] that computes a discrete approximation of the medial axis in linear runtime, greatly speeding up the algorithm compared to the brute force sampling approach used by BioMesh3D. To prevent possible artefacts in the output IMA, arising from its discrete nature, we explicitly bound the feature size from be-low by a specified minimum radius of curvature (in practice 0.8 times the voxel size).

The particle positions are optimized using an adapted version of the *particle-system-mm* module [Par, 2014] also used in BioMesh3D. They are then iteratively inserted as vertices into the dense mesh after which all other vertices are iteratively removed. Every vertex that is removed creates a hole, which is immediately closed using a local triangulation scheme, analogous to that of Knapp [Knapp, 2002].

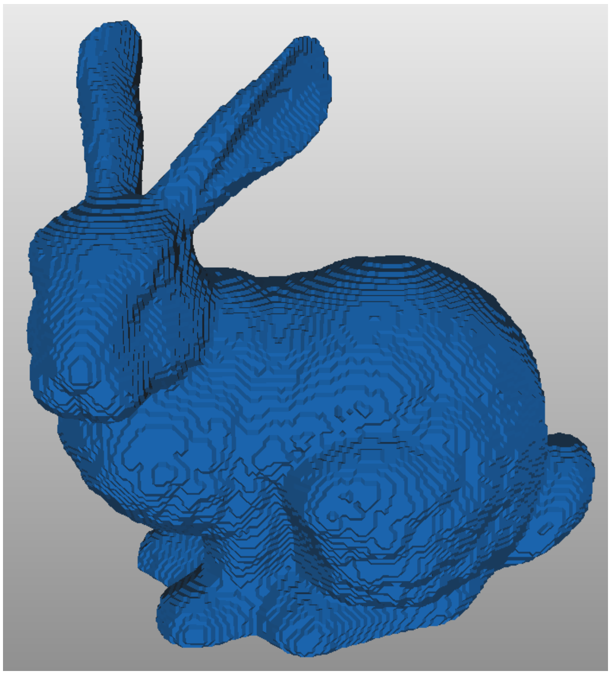
We attempt to avoid the creation of an advancing border separating dense and sparse mesh regions, as such a transition zone that combines short and long edge lengths easily leads to triangles with unfavourable aspect ratios. To that end, vertices are added and removed by placing them in a processing queue with random ordering. At each insertion or deletion we first check for the creation of an undesirable crease or self-intersection, and where this occurs we move the problematic vertex to the back of the processing queue. This process continues until all vertices have been processed.

Conformal edge connectivity across material transitions is enforced by a series of edge flip passes. Each pass ensures that a given pair of meshes have conforming edges on their shared boundary. For each of these passes, one of the material labels is chosen as master, and the other as slave. All co-located vertices on the slave mesh are forced to have an identical edge connectivity as the master mesh. Going through all mesh interactions systematically, at most $(m^2-m)/2$ edge flip passes are needed, where m is the number of materials.

7.5 Results

7.5.1 Bunny (binary image volume)

The first example we present is a 173 x 173 x 181 isotropic voxel volume image obtained by manual segmentation of the Stanford bunny CT dataset [Levoy, 2008]. It is a genus 0 object with two materials (foreground and background). This binary example illustrates differences between our algorithm and that of [Meyer et al., 2007], as implemented in BioMesh3D. In Table 7.1, we show numerical results for Figs. 7.4 and 7.6 that illustrate the advantage of being able to raise ε above 0.5. For $\varepsilon = 1.0$ we achieve a lower element count as well as a lower geometrical error. The former is due to the sparser vertex sampling, while the latter is achieved by not having to use the unique Delaunay triangulation – instead having been chosen more freely to minimize geometrical error.


(a)

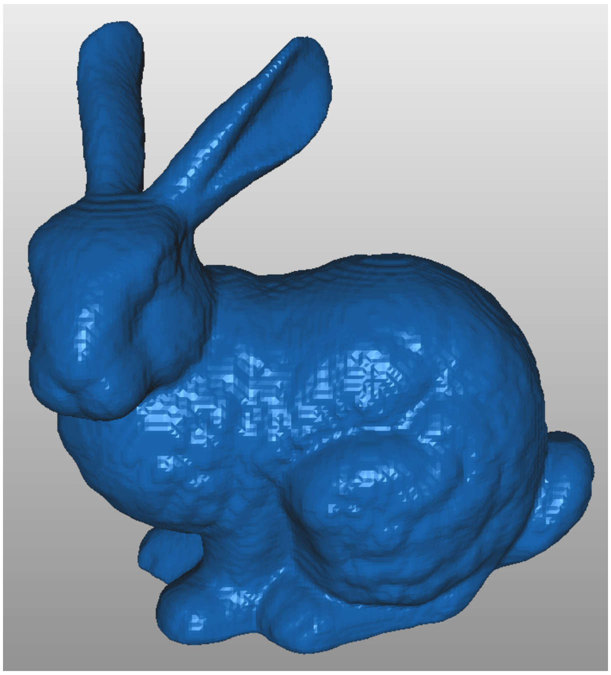

(b)

Figure 7.3 – The Marching Cubes surface of a) the original segmented volume and b) the smoothed level-set isosurface.

In Fig. 7.3a we see the Marching-Cubes isosurface of the Stanford Bunny obtained directly from the CT image volume. Stair-stepping artefacts, a result of the binary segmentation on the isotropic voxel grid, are clearly visible. In Fig. 7.3b, the isosurface is shown after smoothing by limiting the isosurface's radius of curvature to $r = 0.8$ using the "tightening" algorithm [Williams and Rossignac, 2007].

In Fig. 7.4a, we see the dense Delaunay-meshed surface that results from the particle distribution using $\varepsilon = 0.5$ applied to the curvature-limited isosurface of Fig. 7.3b. Areas of high curvature require a very high number of particles, yielding a mesh with a very high element count. Starting with the original isosurface of Fig. 7.3b and, instead using CVMesh's iterative vertex insertion technique, we are able to safely increase ε without smoothing the model. In Fig. 7.4b we show the result using $\varepsilon = 1.0$, leading to fewer unnecessary vertices on fine features. The overall distribution becomes sparser and more uniform, while simultaneously even lowering the geometric error as measured by the Hausdorff distance.

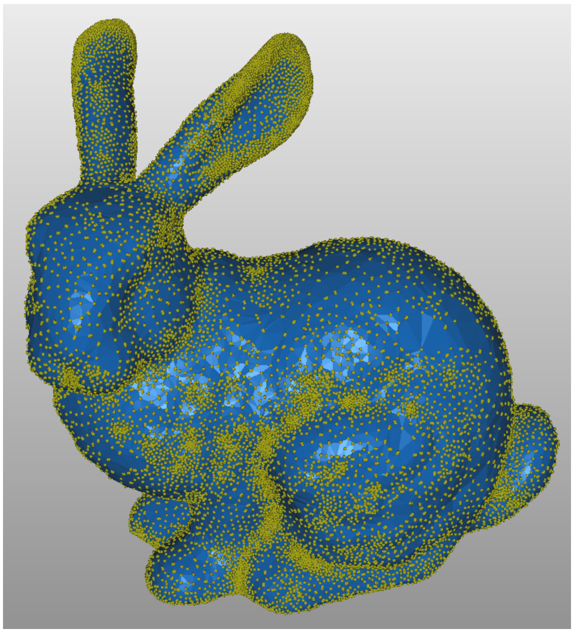
In BioMesh3D, adhering to the ε-sampling of 0.5, we have to increase the radius of curvature in order to reduce the element count, but this leads to a severe detail loss. At the same time, the "tightening" operation cannot lower the curvature in thin regions such as the bunny's ears, maintaining a locally dense vertex distribution. In Fig. 7.5a the radius was increased to r = 15 while ε remained unchanged at 0.5. Increasing ε leads to fewer elements, but also disrespects the ε-sampling requirement, potentially leading to incorrect topology. Fig. 7.5b shows such topological errors at the bunny's ears.

CVMesh can dramatically reduce the number of mesh elements without causing invalid mesh topology. In Fig. 7.6a, the identical vertex distribution that leads to an inconsistent mesh using BioMesh3D is used to achieve a topologically-correct output.

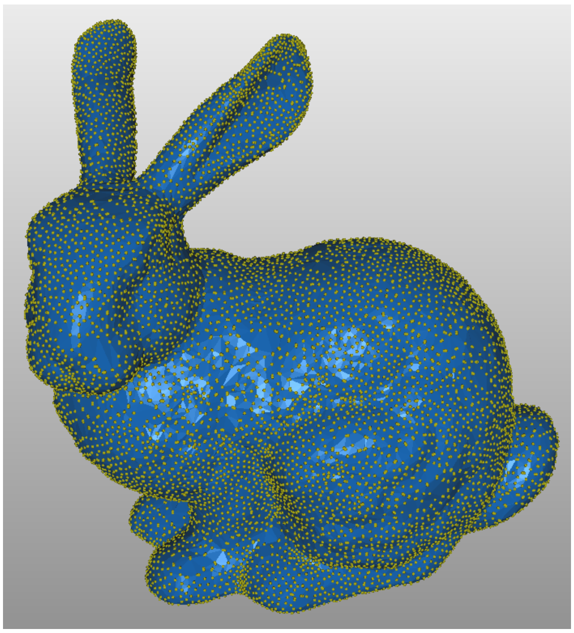
7.5.2 "Tooth" dataset (multiple materials)

The multi-material 103 x 94 x 161 isotropic human CT "Tooth" dataset was cropped from the dataset used in the IEEE Transfer Function Bake-Off [Pfister et al., 2001] and is also freely distributed with BioMesh3D [Bio, 2015]. The dataset consists of four materials: air, dentine, enamel, and root. The results for this dataset are summarized in Table 7.2.

In Fig. 7.7a, we see a sparse vertex distribution (2002, 2686 and 384 particles for the enamel, dentine and root respectively) superimposed on the high-resolution isosurfaces meshes of the three material labels. For this example,

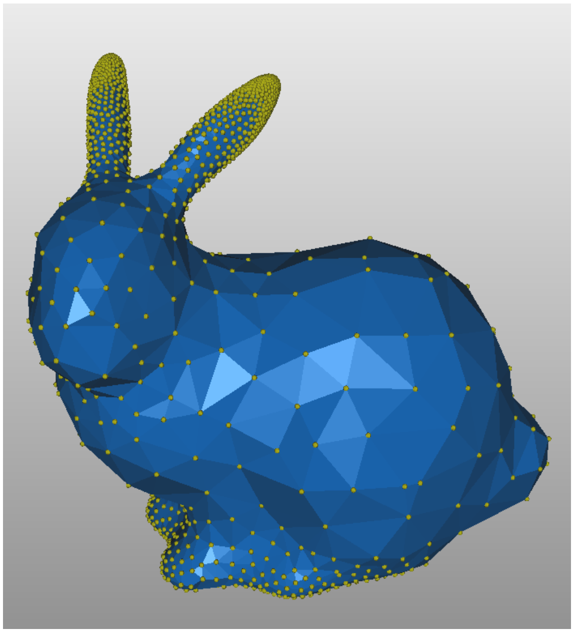


(a)

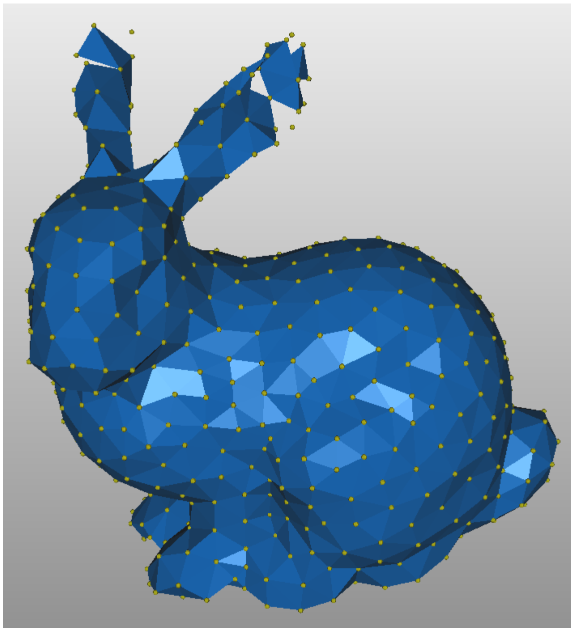


(b)

Figure 7.4 – a) Particle distribution for $\varepsilon = 0.5$. This value of ε is considered a loose upper bound for safe Delaunay meshing [Meyer et al., 2007]. b) Using CVMesh we can safely use a particle distribution of $\varepsilon = 1.0$.

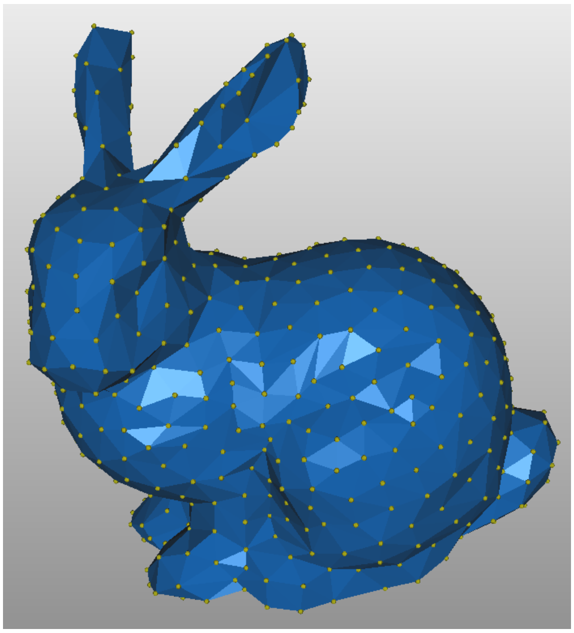


(a)

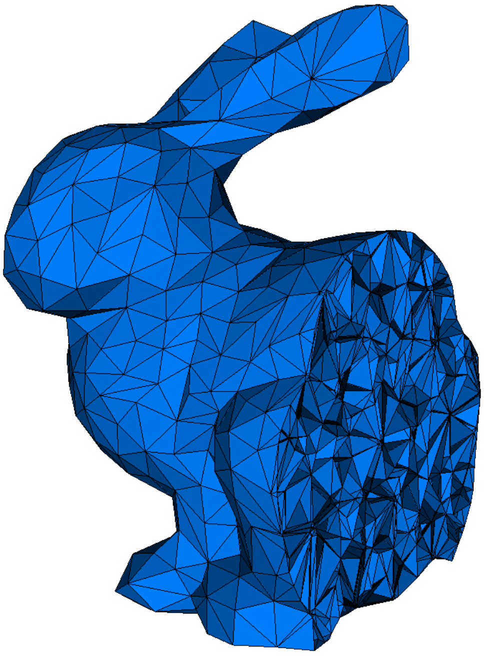


(b)

Figure 7.5 – Reducing vertex count by a) increasing the radius of curvature leads to severe detail loss, while b) only increasing ε preserves more detail, but when Delaunay triangulation is used leads to degenerate mesh reconstruction in thin regions.

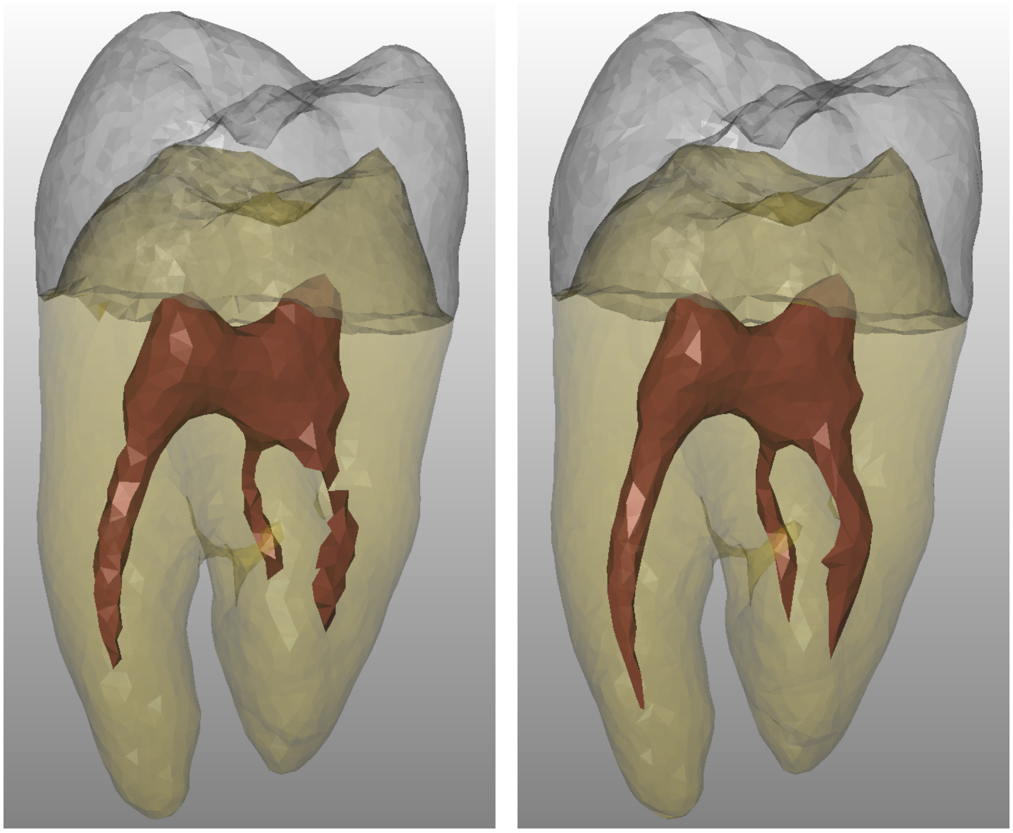


(a)



(b)

Figure 7.6 – a) Topologically correct surface mesh generated using CVMesh's iterative refinement, using the same vertex distribution of Fig. 7.5b. b) Cut view of the tetrahedra-filled mesh.



(a) (b)

Figure 7.7 – For the “Tooth” dataset we see a) If the ε-sampling requirement is not met, Delaunay triangulation creates an invalid triangulation, especially visible in the interior. b) CVMesh creates a topologically correct multi-material mesh with identical vertices as in a.

	r = 0.8 $\varepsilon = 0.5$ (Delaunay)	r = 0.8 $\varepsilon = 1.0$ (CVMesh)	r = 0.8 $\varepsilon = 5.0$ (CVMesh)
# Surface mesh vertices	25,109	15,361	558
# Surface triangles	50,258	30,718	1,170
# Tetrahedra	202,219	**177,552**	**43,596**
Hausdorff distance	0.896	**0.874**	3.75
Mean Hausdorff dist.	0.104	**0.0938**	0.496
Triangle quality: mean	0.79	0.7	0.78
Triangle quality: stddev.	0.21	0.26	0.22
Tetrahedron quality: mean	0.64	0.66	0.73
Tetrahedron quality: stddev.	0.19	0.22	0.18

Table 7.1 – Mesh metrics for the “Bunny” dataset.

traditional decimation techniques cannot be used, as per-material decimation breaks conformity on multi-material interfaces. Fig. 7.7b illustrates the topologically-incorrect surface mesh obtained via BioMesh3D in combination with the sparse vertex distribution. Especially, the tooth’s inner root structure is affected and incorrectly split into two sub-volumes. This is not unexpected, as the used sampling of $\varepsilon = 3.0$ is almost guaranteed to be unsuitable for a Delaunay surface reconstruction. In Fig. 7.7c, we see that CVMesh preserves the correct topology and better approximates the model’s thin extremities.

7.5.3 “Femur” dataset (multiple materials, anisotropic scan)

The “Femur” dataset was obtained for manually segmenting a clinical CT volume comprising 250 x 250 x 500 anisotropic image voxels. The dataset The voxel spacing was $0.4mm \times 0.5mm \times 0.8mm$. As such it was the highest resolution dataset we examined and also had the highest number of materials – six. In addition to the background these six materials (prosthesis, cement, intramedullary canal, trabecular bone, cortical bone and fibrous tissue) were segmented and featured extensive contact areas. Results for this mesh are shown in Fig. 7.8 and Table 7.2.

7.6 Discussion

In summary, CVMesh uses the original mesh to track topology, which enables more aggressive particle-based sampling strategies than in reconstruction by

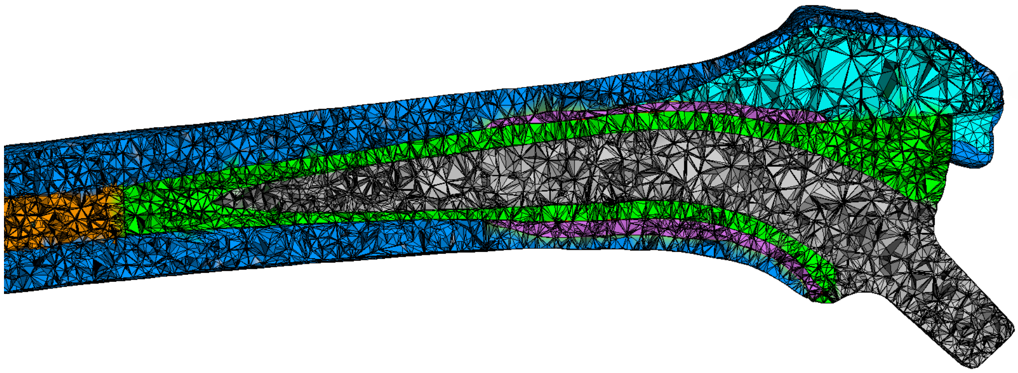


Figure 7.8 – The output obtained for the femur data-set, cut along a plane to show the interior tetrahedra and six distinct material regions.

	"Tooth" dataset r = 0.8 $\varepsilon = 3.0$ (CVMesh)	"Femur" dataset r = 0.8 $\varepsilon = 4.0$ (CVMesh)
# Materials	4	7
# Segmented image voxels	410,826	5,965,683
Voxel spacing	$1 \times 1 \times 1$	$0.5 \times 0.4 \times 0.8$mm
# Surface mesh vertices	3,981	21,393
# Surface triangles	10,124	45,694
# Tetrahedra	65,442	182,493
Hausdorff distance	1.83	3.81
Mean Hausdorff dist.	0.117	0.085
Triangle quality: mean	0.83	0.71
Tetrahedron quality: mean	0.72	0.65

Table 7.2 – Mesh metrics for the "Tooth" and "Femur" datasets.

	“Bunny”	“Tooth”	“Femur”
# Segmented voxels	758,706	410,826	5,965,683
Smoothing (“tightening”)	41 sec	77 sec	7 min
Medial Axis	18 sec	13 sec	72 min
Sizing Field	105 sec	35 sec	21 min
Particle optimization	840 sec	2220 sec	483 min
Vertex insertion	4.8 sec	31 sec	16 min
Vertex decimation	87.7 sec	184 sec	38 min
Edge flipping	2.2 sec	9.7 sec	3 min
Total time (CVMesh)	**18 min**	**43 min**	**640 min**

Table 7.3 – Unoptimized execution times for the “Bunny”, “Tooth” and “Femur” datasets.

Delaunay triangulation. The upshot of the additional freedom in vertex placement is the ability to achieve better geometric approximations with fewer mesh elements.

Specifically, we demonstrate that CVMesh can be used successfully with high values of ε that violate the ε-sampling requirement described by [Amenta and Bern, 1998] and [Meyer et al., 2007]. The upshot of this freedom is that coarser meshes can be used to achieve a given level of geometric accuracy.

CVMesh maintains mesh topology at the end of each vertex addition or vertex removal. Removing a vertex temporarily causes a hole which needs to be closed so that topology is maintained. The mesh patch that is removed along with a vertex – consisting of all triangles connected to the vertex – must be manifold and the resulting hole must also be closeable using another manifold patch, without intersection of any mesh triangles. These assumptions automatically holds when the ε-sampling requirement is respected and generally also holds beyond this limit as shown in the examples of this paper, e.g. at least up to $\varepsilon = 5.0$. As we cannot rely on a theoretical proof of this observation, our re-meshing steps are algorithmically checked for intersections at every mesh modification step.

In its current form, the transformation of a mesh from a the dense MC isosurface to the final particle-defined mesh, as described in Algorithm 7.3.1, is an iterative process that is performed for each material sequentially. This process allows for reordering of vertex operations and closes the holes created by vertex removal so as to minimize geometrical error. The lack of synchronization between the processing of the meshes belonging to different materials mean that adjacent edges are initially not identically defined even though the vertices are. These edge mismatches are solved in the algorithm's final edge flipping step but the algorithm may in future be improved by processing all material meshes in parallel instead of sequentially.

Our algorithm has an inherent limitation in that extremely sparse sampling of a thin curved volume can lead to unavoidable self-intersections as shown in Fig. 7.9. To avoid, or at least detect this we heuristically check for self-intersections during iterative mesh decimation. For the meshing examples in this paper we did not encounter this issue.

In its current implementation CVMesh is still slow for large datasets, as shown in Table 7.3 that shows the runtimes of our experiments on an Intel i7 950 desktop PC running at 3.07 GHz. The algorithm has not been particularly optimized, and especially the vertex insertion, vertex removal and edge flipping steps have been implemented as unoptimized single-threaded Python code. In CVMesh the theoretically challenging task of medial axis computation represents a small fraction of the total algorithm time due to our selection of the efficient IMA algorithm. The majority of CVMesh's run-time is spent on optimizing particle positions, which is an obvious target for future optimization such as massive parallelization on a GPU [Kim et al., 2012].

The minimum number of vertices that is needed to represent any single, connected, genus-0 object is four, in which case the resultant mesh region will be a single tetrahedron. This limitation is relevant whenever a material's segmented label field contains one or more tiny, disconnected, islands that may be insufficiently sampled. To solve this we pre-filter the segmented image to remove small islands, or disallowing particles that insufficiently sample small isolated isosurface islands can prevent this situation from occurring.

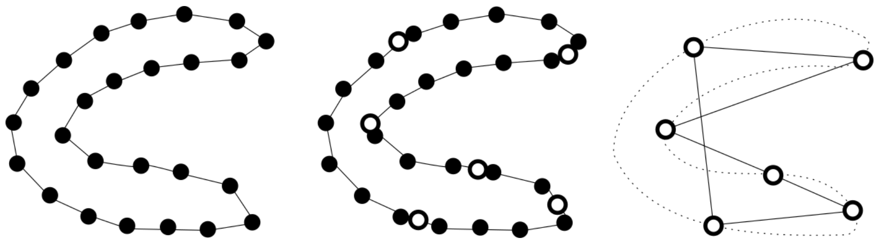


Figure 7.9 – Cross-section illustration of self-intersection that may be caused by an extremely sparse, degenerate, vertex distribution.

The degenerate vertex distribution show in Fig. 7.9 invariably leads to an unacceptable result. As opposed to this inherently degenerate case, transient self-intersections at intermediate mesh refinement steps may be allowed. However we did not allow this in our implementation, as it is difficult to foresee whether self-intersection during an intermediate step will eventually be resolved by additional decimation. Our algorithm therefore permutes the order of vertex processing whenever an operation would have led to self-intersection. If this process fails, the meshing algorithm terminates with the warning that a degenerate vertex distribution was detected – a solution to which would be to specify a smaller value to ε or a larger radius of curvature. This situation was not observed in our experiments.

7.7 Conclusion

We presented a viable approach to multi-material conformal tetrahedral meshing that does not use Delaunay triangulation.

The final density of these optimized seed particles can be controlled by the user, and need not adhere to the Delaunay ε-sampling constraint for the algorithm to succeed. This freedom enables the output mesh to have fewer elements and/or lower geometric approximation errors than comparable Delaunay-triangulation-based methods.

The output of our algorithm is intended to be suitable for finite-element analysis, which is important for applications such as biomechanical simulations. In the examples shown in this paper we were able to generate tetrahedral meshes with a tractable number of mesh elements, while these elements were also well behaved in terms of their aspect ratios and geometric error as measured by their Hausdorff distances relative to the input label field.

Acknowledgements

We would like to thank Charp P. Botha for his contributions to the supervision of this project, as well as Peter Schaafsma who contributed a vital idea to our solution.