2012-ENST-00xx

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « SIGNAL et IMAGES »

Noura FARAJ

Soutenance prévue le 3 juin 2013

Modélisation, Visualisation et Interaction par Maillages avec les Volumes 3D

Directeur de thèse : **Tamy BOUBEKEUR**
Co-encadrement de la thèse : **Isabelle BLOCH**

Jury

M. Loïc BARTHE, Maître de conférence, IRIT-CNRS, Université Paul Sabatier	Rapporteur
M. Mario BOTSCH, Professeur, Bielefeld University	Rapporteur
M. Pierre ALLIEZ, Directeur de recherche, Inria Sophia-Antipolis	Examinateur
M. Stéphane COTIN, Directeur de recherche, Inria Lille	Examinateur
M. Tamy BOUBEKEUR, Maître de conférence, CNRS-LTCI, Télécom ParisTech	Directeur
Mme Isabelle BLOCH, Professeur, CNRS-LTCI, Télécom ParisTech	co-Directrice

TELECOM ParisTech
école de l'Institut Télécom - membre de ParisTech

THÈSE

Contents

Chapter 1

Introduction

Our ability to capture accurately the geometry of real objects opened the way for a wide range of scientific discoveries. Indeed, the acquisition systems reached a precision allowing to capture submillimetric surface details while accounting for their volume and internal materials. For instance, 3D scanners, Magnetic Resonance Imaging (MRI) or Computed Tomography (CT), among others, produce high-resolution multi-material volumetric datasets. These volumes are represented as large, high-resolution voxel grids (also called 3D images since a voxel is the pixel volumetric equivalent) and are typically made of several hundreds of millions of voxels. These grids are used to study physical phenomena (e.g. tornadoes, smoke), to perform physical simulations (e.g. fluids, explosion, car crashes) - see Figure 1.1 - or in a medical context (e.g. prosthesis creation, dosimetry analysis, surgery simulation). The increasing number of generated 3D anatomical images, such as in the context of the Visible Human and Virtual Population projects, allows performing a wide range of medical simulations (Figure 1.2).

In this non-exhaustive list of applications, practitioners often seek for accurate underlying mechanical and physical numerical models. This is indeed only possible when speed is a soft constraint. Thus, the simulations can run offline for hours or days, even weeks in order to get a realistic result. For interactive simulations, speed is a hard constraint, as the user's needs a direct feedback which might come at the expense of precision. To preserve the interactivity the simulation might run with simplified settings, less accurate physical models, or with a fast approximation of models...

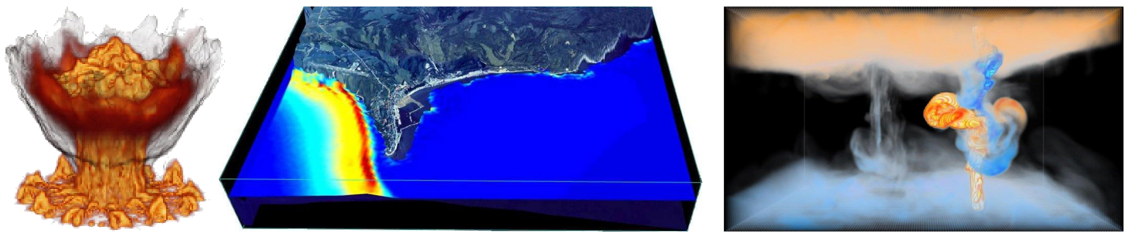


Figure 1.1: Examples of physically based simulations, from left to right: combustion [CS10], tsunami [TS97] and smoke [Moh12] simulations.

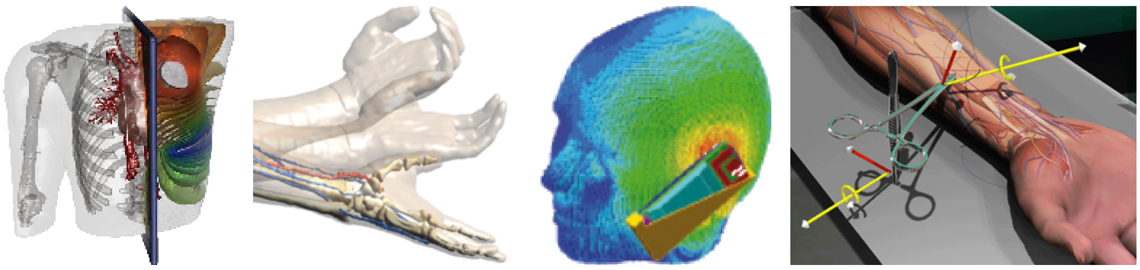


Figure 1.2: Examples of medical simulations, from left to right: cardiac implant electric field simulation in a torso model [Ins09], hand deformation [ITI10], dosimetry analysis and surgery simulation [LLC11].

An alternative solution is to run the simulations on a tetrahedral approximation of the input data instead of directly on such large datasets. Indeed, the number of elements needed to represent the input domain is significantly lower, therefore, allowing the use of Finite Elements Methods (FEM) with a considerably reduced computation cost and memory footprint. The results and performances of these simulations depend on the representation accuracy and on the quality, size and orientation of its elements. In the best scenario, a high-quality adaptive, isotropic mesh can be generated at any resolution depending on the aimed precision and available resources. Only a limited number of methods allow doing so, while accounting for the complex topology of multi-material datasets.

Independently of the chosen representation, the goal of the simulations is to study not only a given phenomenon but also its variability given different settings and configurations. Unfortunately, the acquisition state or posture limits the scope of the analysis since the capture setting is often restricted (e.g. lying down in an MRI machine). Therefore, a pre-processing, such as deformation, is often necessary. This process needs to preserve the features and quality of the input to avoid compromising the simulation. Regrettably, high-quality deformation methods are usually not linear and require costly iterative solving processes, even on small models. Moreover, they are often defined on surface or volume meshes. Although acceptable when a mesh approximation is sufficient, this restriction becomes problematic when targeting the original full resolution voxel dataset.

At each step of the aforementioned meshing and deformation processes, an interactive visualization is mandatory. Displaying both representations of the multi-domain datasets (i. e. high- resolution grids or tetrahedral meshes) is a challenging task not only due to their

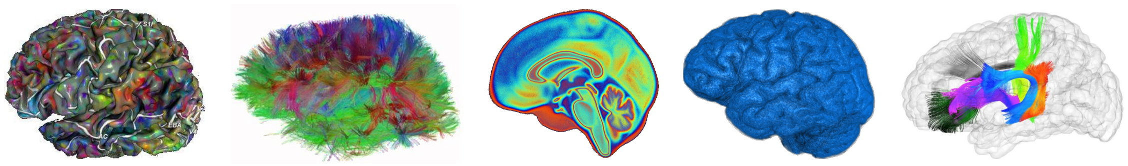


Figure 1.3: **Visualization.** Examples of different visualization strategies on a brain. Image courtesy of, from left to right: functional MRI semantic space [Gao12], Human connectome [oNIaUfBI13], scalar field [LLBP12], outer surface [Web] and microstructures [SEG+12].

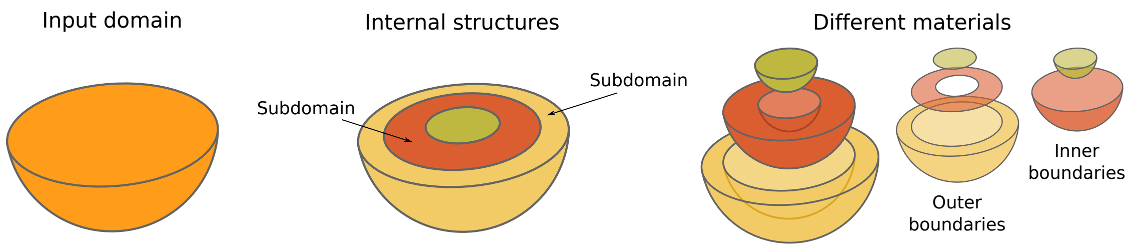


Figure 1.4: Multi-material input. (Left) Input domain, (Center) its internal structures made of different materials, called subdomains, and (Right) Inner and outer boundaries representing the junctions between different materials.

size but also to their topological intrinsic complexity: indeed, displaying all the intricate materials simultaneously might be confusing. To tackle these scalability and clarity issues, only relevant data - i.e. carrying the information necessary to the current process - is displayed (e.g. a slice, boundaries, selected subdomains... Figure 1.3). Extensive work has been done in scientific visualization to identify the user needs and find the adapted strategy.

We have seen that handling interactively these high-resolution grids is a challenging task since scalability, speed and precision are constraints that are difficult to match even on powerful work stations. The main goal of this thesis is to offer methods and tools to mesh and deform 3D volume multi-material datasets using meshes as an interaction primitive. Moreover, we aim at providing tools offering a good balance between quality and accuracy with an efficient display.

In the following, we start by describing the labeled voxel grids and tetrahedral meshes, used to represent multi-material input datasets. Then, we present the applicative context of this thesis.

1.1 Data representation

The input model might be made of different materials, their union represents the input *domain* therefore its materials represent its intricate *subdomains*. The main information is held by the interfaces between materials since they indicate the boundaries of the subdo-

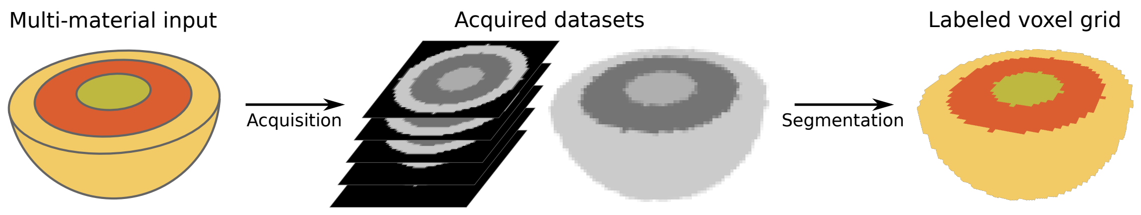


Figure 1.5: Acquisition and segmentation process: (Left) Multi-material input, (Center) 2D gray level acquired slices and the corresponding constructed 3D image represented by a gray level voxel grid (Right) that undergoes a segmentation process.

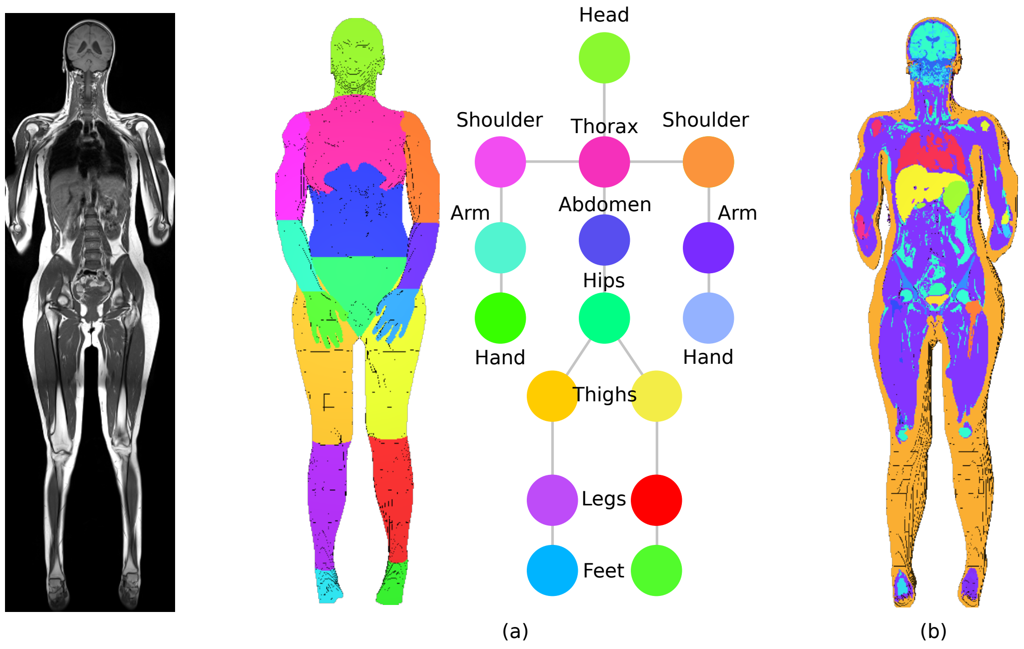


Figure 1.6: Two segmentations of an input whole-body 3D image: (a) body part segmentation and (b) organ segmentation [FAB+11].

mains giving information about their shape and junctions (Figure 1.4). Those boundaries are composed of surface patches (2-dimensional elements at the interface between two materials), possibly feature lines (1-dimensional elements at the meeting of three or more labels), and points (0-dimensional elements meeting points of four or more materials). In the rest of the document, we will refer to the n-dimensional elements as *n-junctions* as noted in [BYB09]. The inner boundaries represent the junctions between the different materials and the outer boundaries their junction with the outer material. Along with the material volumes, the boundaries represent the main characteristics of a multi-domain input.

1.1.1 Image data and segmented volumes

The input domain is acquired using, for instance, MRI or CT scanners generating a collection of 2D gray level slices from which a 3D image is reconstructed and represented by a 3D voxel grid containing the images discrete gray levels (Figure 1.5 center). To identify the different subdomains, the 3D image undergoes a segmentation process - i. e. each voxel is assigned a unique label representing a single material (Figure 1.5 right).

The resulting segmented 3D image is represented by a 3D voxel grid containing the assigned discrete labels. By convention, null values represent the background. Note that different segmentation can be associated with the same acquired dataset (Figure 1.6 (a) represents a body part segmentation and (b) an organ segmentation). We refer to [CPF+95, Maî03] for recent surveys on volumetric segmentation.

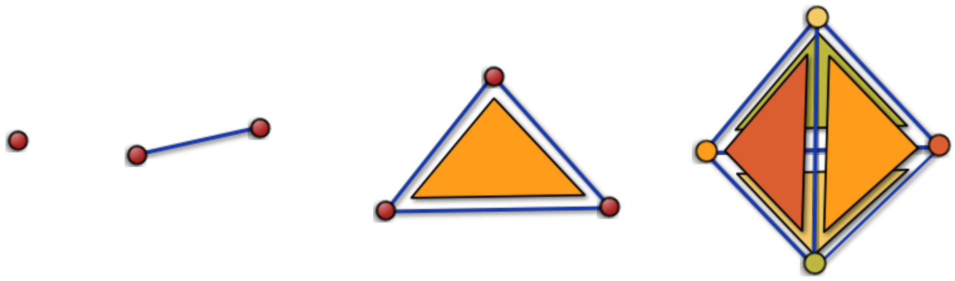


Figure 1.7: Tetrahedral mesh elements, from left to right: a vertex (0-dimensional), an edge (1-dimensional), a triangle (2-dimensional) and a tetrahedron (3-dimensional).

1.1.2 Tetrahedral meshes

A tetrahedral mesh is a discrete representation of the input 3D domain, called a 3D triangulation. It is composed of elements of different dimensions: vertices, edges, facets and tetrahedra (Figure 1.7). A vertex is a 0-dimensional element defined by a point in $\mathbb{R}^3$, an edge is a 1-dimensional element corresponding to the line segment between a pair of connected vertices. A facet is a triangular polygon defined by three connected linearly independent vertices. Finally, a tetrahedron is defined by four linearly independent vertices, and made of the union of four facets, six edges and their embedded space.

To represent the multi-material input domain, the mesh is labeled - i. e. each tetrahedra is assigned a unique label representing a single material -, therefore, each set of tetrahedra with the same label form a submesh that approximates a single subdomain.

The input subdomain's boundary elements are represented using a collection of facets, for the 2-dimensional elements, of edges for 1-dimensional elements, and vertices, for 0-dimensional ones. These can be found at the interfaces between submeshes (Figure 1.8).

As we have previously seen, this representation can be used to perform physically-based simulations but it often needs to undergo a (re)meshing or deformation pre-process. For instance, the input model needs to be set in a physically plausible posture before performing dosimetry analysis.

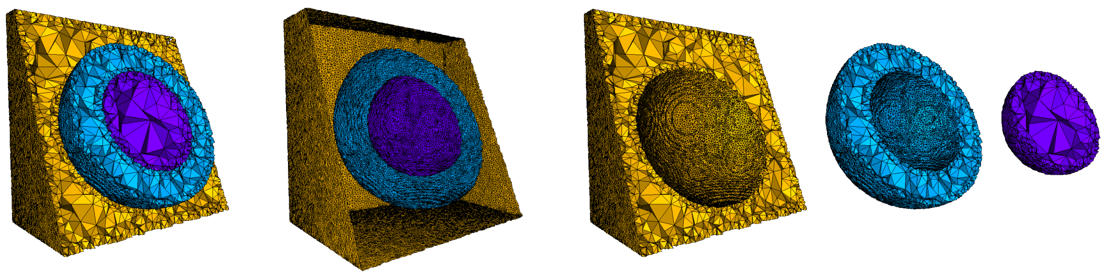


Figure 1.8: Example of multi-material tetrahedral mesh: (Left) cut view, (Center) boundaries, and (Right) each subdomain represented separately by a submesh.

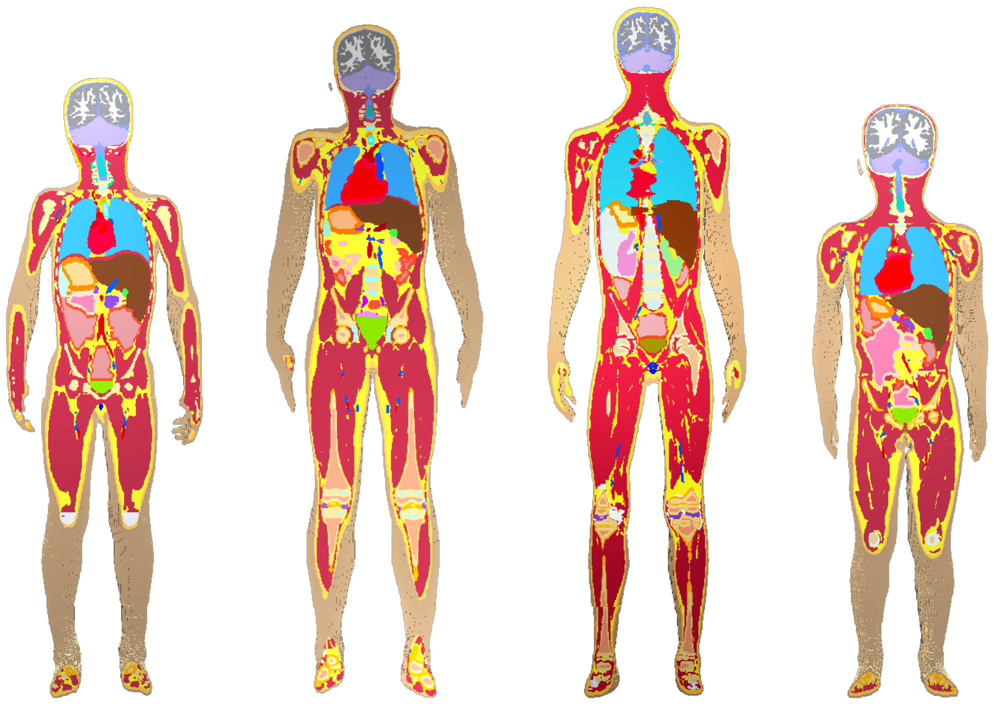


Figure 1.9: Input whole-body segmented voxel grids from the virtual population project.

1.2 Applicative context

This work is part of a project, called KidPocket, of the French research agency to study the potential effects of electromagnetic waves exposure, stemming from emerging technologies, on people's, and more particularly, on children's health, for which we provide modeling tools.

The increasing exposure of human to electromagnetic waves raises a public health concern. Therefore, the actual nature and quantity of the exposure needs to be assessed in order to evaluate the possible impact on human health. To do so, simulations of wave exposure are performed to measure the specific absorption rate (SAR) giving the rate at which energy is absorbed by the body.

1.2.1 Geometric modeling and dosimetry analysis

The specific goal of our project is to measure the exposure to waves from cellphones on children and to study the influence of the posture and morphology on it. These studies are performed using numerical models applied to 3D volumetric datasets representing human models. Only a limited number of scans of children are available and their posture is is often limited to the unavoidable upright acquisition position, lying down on their back or front. These postures are not natural for cellphone use and reduce the scope of the analysis.

This project suggests solutions and is divided into three parts:

1. generate new children whole-body models and offer deformation tools to change their posture,
2. study the exposure associated with the use of new telecommunication technologies,
3. develop methods and tools to evaluate the uncertainty associated with the evaluation of the local and global exposure.

This project includes numerous partners: Télécom Bretagne, Orange Labs, Phimeca Engineering, Université Pierre et Marie Curie, Université Paris-Est Marne la Vallée, National Institute of Medical Research (INSERM), Inria Sophia Antipolis team project NACHOS and Télécom ParisTech.

The work done is this thesis contributes to the first task by providing tetrahedral meshes representing anatomical models of children and scalable high-quality deformation tools. These tools allow us to interactively pose whole-body high-resolution voxel grids before performing SAR measures. Therefore, our methods will be mainly applied but not limited to segmented medical images.

1.2.2 Available datasets

The whole-body MRI of children, made available by our partners, have been acquired from collaborating hospitals. Depending on the patient, around 32 coronal slices were acquired with a slice thickness of $6mm$. The reconstructed voxel size for all images is $1.3 \times 1.3 \times 7.2\ mm^3$. This strong anisotropy causes the data to exhibit a lot of partial volume effects. Due to the use of multiple coils, the images actually result from the composition of 4 or 5 images (depending on the patient's height), and some artifacts may appear such as missing parts due to field size or lower intensity at the transition between two images. To tackle the problems emerging from the resolution and the position of the patient (e. g. often the patient had his hands leaning on his thigh or the arms stuck to the thorax in the armpit region – Figure 1.6), a body part segmentation is performed, as described in [FAB+11]. Additionally, we use whole-body children models from the Virtual population project [CKH+10]. These models, composed of 1mm cubic voxels, are highly detailed with up to 84 represented organs (Figure 1.9). Furthermore, to demonstrate the scalability of our approach, we also use the Visible Human dataset [SASW96].

1.3 Contributions

The core idea of this thesis is to combine the two major alternative representations for volume datasets, namely *voxel grids* and *tetrahedral meshes*, in an unified framework allowing to process and edit interactively massive datasets. In particular, we will show that enhancing a high-resolution voxel grid with a task-specific adaptive volume mesh yields a flexible algorithmic ecosystem for volume meshing, freeform volume deformation and interactive volume rendering. The main contributions of this thesis are:

- an iterative remeshing algorithm for 3D triangulations providing high-quality meshes at the desired resolution while preserving features such as multi-material boundaries. Our method can also be applied on segmented voxel grids by initializing a trivial high-resolution tetrahedral mesh generated from it.
- an interactive 3-scale freeform deformation system, called *VoxMorph*, using a motion adaptive tetrahedral mesh as a mid-scale deformation structure, for both full resolution voxel grids and meshed approximations, which ensures limited distortions and mass loss.
- an extension of the VoxMorph system toward even higher resolution models, with improved accuracy and robustness.

We complete our framework by providing an interactive visualization of the high-resolution voxel grids taking advantage of the time-varying tetrahedral mesh used for the deformation.

1.4 Thesis organization

After this introduction, Chapter 2 makes an overview of the existing tetrahedral meshing and remeshing approaches, presents our method as well as an evaluation of the generated meshes' quality. Chapter 3 gives an overview of the existing interactive deformation methods, presents our VoxMorph system and evaluates it. Chapter 4 presents a more robust and scalable version of this system and an electro-magnetic wave exposure study for human-waves interaction simulations done with the help of our partners. Chapter 5 presents the additional elements to obtain an iterative and understandable visualization of the high-resolution voxel grids.

Finally, Chapter 6 gives a general conclusion on the contributions of this thesis and Chapter 7 an overview of the research avenue it has opened.

Chapter 2

Volume data modeling

In this chapter, we focus on the generation of multi-material meshes suitable for simulations. The mesh must conform to the domain inner- and outer-boundaries while removing the acquisition noise. Furthermore, the use of FEM involves quality and size criteria since poorly shaped tetrahedra induce instability in the numerical models and a large number of elements make interactive simulations impossible (Section 2.1.1). These constraints are difficult to match directly during the meshing process (Section 2.1.2), hence, a quality improvement post-process is often necessary. Furthermore, in order to meet material, precision and computation time constraints the simulations might be run at a different resolution making a simplification and refinement processes desirable. Unfortunately, the resulting mesh often fails to meet the quality requirements. Therefore, remeshing is preferred, since the domain is re-sampled to combine the size and quality constraints (Section 2.1.3). Furthermore, a fast remeshing method allows the user to explore the parameter space easily, and to navigate between different resolutions. Additionally, this process can be applied on any multi-material tetrahedral mesh and allows to extend the scope of usable datasets.

We propose an iterative isotropic remeshing method conforming to multi-material boundaries using well-known local topological operations. We start by describing the topological and quality constraints and we present an overview of existing tetrahedral mesh generation (Section 2.1), simplification and improvement methods. Next, we describe our method (Sections 2.2 and 2.3) and evaluate the results using different quality criteria (Section 2.4). Finally, we compare our model to state-of-the-art methods and explain in which cases our approach is preferable.

2.1 Background

In this section, we start by presenting the properties and criteria our model must meet in order to produce meshes that are suitable for simulations, then, we give a quick overview of existing methods.

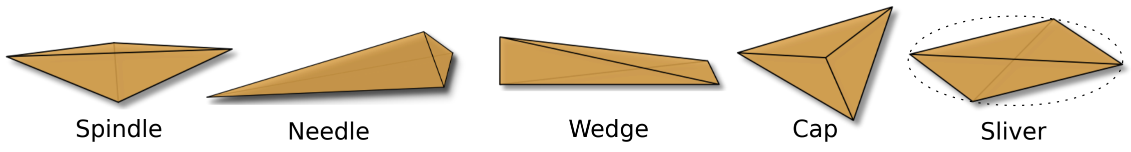


Figure 2.1: **Ill-shaped tetrahedra.** Different types of tetrahedra degeneracy. The most common one is a sliver, i.e. almost flat but with edges of similar length (last example).

2.1.1 Requirements and quality evaluation

Multi-material settings involve additional constraints for tetrahedral mesh generation and processing, that will be described in the following.

Topology criteria

As we have seen in the previous chapter, multi-material domains and their intricate subdomains may have complex topologies that need to be represented in the output mesh. The topological information is held by the boundaries through n-junctions, which need to be handled explicitly, in order to sustain the materials volume and connectivity. Nevertheless, the input data are subject to acquisition and discretization noise. Therefore, a balance between the preservation of the input mesh or image features and the regularization needs to be found since it is necessary to ensure smooth boundaries for visualization and stability purposes. This problem can be modeled as an energy minimization with two terms: one to conform to the data and the other for regularization. This energy is noted as $E = E_{data} + E_{regularization}$.

Tetrahedra quality constraints

Apart from boundary conforming criteria, the FEM methods stability, convergence and accuracy depend on the size and shape of its tetrahedra [She02]. Tetrahedra with very small dihedral angles cause negative volumes under small perturbations, and large angles strongly increase the simulation errors. One single inverted or poorly shaped tetrahedron is enough to compromise the entire simulation. Figure 2.1 illustrates the different degenerate cases. In 2D, the quality of the triangle shape is evaluated using the ratio of the shortest and longest edge, which gives a bound on the minimum angle but this property is no longer true in 3D. A poorly shaped tetrahedron can have edges with similar length. This is the case for the most common type of degeneracy, called *slivers*, which are almost flat tetrahedra. A better way to evaluate a tetrahedron's quality is to evaluate its minimal dihedral angle. Note that a regular tetrahedron has dihedral angles equal to 70.5 degrees and a sliver has two large angles and the others two are almost null.

Element size and distribution

The maximum stability is obtained when the simulations are run on uniform meshes but these are composed of very small elements to represent the boundaries accurately. There-

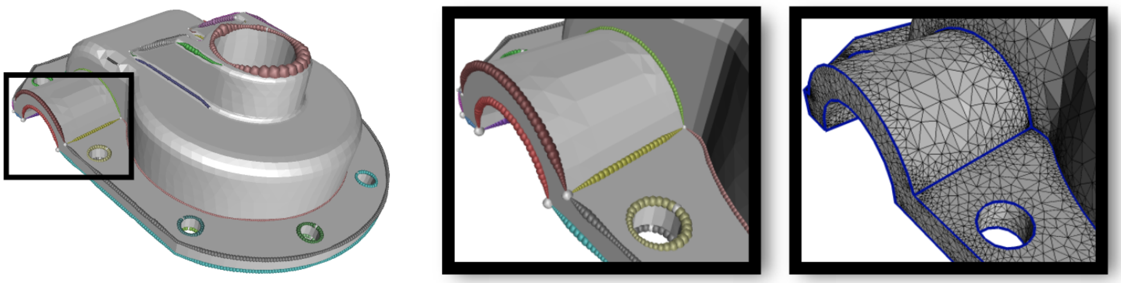


Figure 2.2: **Piecewise smooth complex**: the colored spheres represent the protecting balls of varying size around its edges. Illustration taken from DelPSC [DL09].

fore, meeting both topological and uniformity constraints might generate meshes with a prohibitive size. A trade-off can be found by running the simulation on adaptive meshes with refined tetrahedra only where needed: near the junctions or in regions of interest. For instance, for a fracture simulation the mesh is sampled more densely near the impact point. The size of the element must vary smoothly to avoid poorly shaped tetrahedra [Rup95].

The generation and simplification of tetrahedral meshes have been active fields of research due to the wide range of applications. We present here a non exhaustive overview of existing methods and focus on the ones that are able to handle multi-material inputs.

2.1.2 Generation of tetrahedral meshes

We can group the methods in three main categories: Delaunay-based methods, Lattice-based methods and variational methods.

Delaunay refinement process

The 3D Delaunay triangulation is a popular kind of tetrahedral mesh due to its empty sphere property specifying that every tetrahedron does not have any vertex in its circumscribing sphere. These triangulations are uniquely defined except in degenerate cases where five points are co-spherical. A Delaunay triangulation is built using a set of points distributed over the input domain followed by a Delaunay refinement process [Che93, Rup95, She98] where *Steiner* points are added to the triangulation until the input approximation, element shape and quality criteria, are met. This process has been proposed first for domains bounded by a smooth surface [BO05a, CD03], and further extended for piecewise smooth boundaries [CDL07]. In order to handle sharp features in the input domain, the authors proposed to build a Piecewise Smooth Complex (PSC) composed of surface patches, curves - meeting of surface patches with arbitrary angles - and points - meeting of curves. Protecting balls of varying size are defined around the edges to preserve the features of this complex during the refinement process [CDR07, DL09]. Figure 2.2 illustrates this method called DelPSC.
These principles have been extended to handle multi-material domains in [PSB$^+$07], and to preserve their 1- and 0-junctions using similar protecting balls [BYB09]. Most recently,

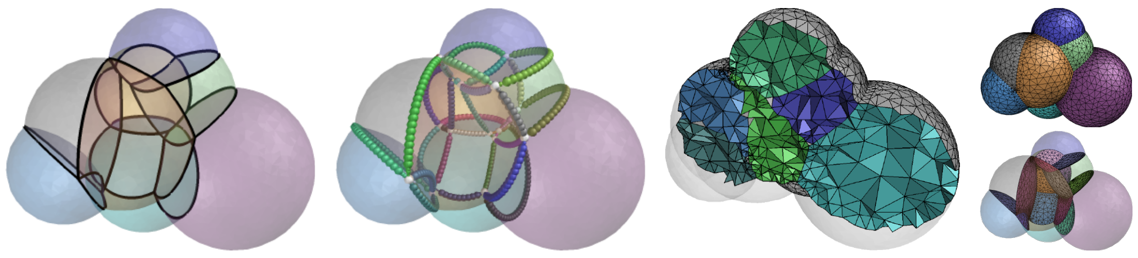


Figure 2.3: (Left) A set of curves are extracted where three or more labeled regions intersect. (Middle) these curves are protected with an initial set of balls that may get refined by the algorithm later. (Right) resulting mesh generated by converting these balls to weighted points and refinement fills in samples for the interface surfaces [DJL12].

Dey et al. proposed a PSC composed of multi-material junctions [DJL12] to represent junctions meeting at arbitrary angle accurately (Figure 2.3). We use a similar complex to identify the features to preserve.
While these methods allow to generate high-quality meshes, the construction of the PSC and the implementation of the protecting balls paradigm are not trivial in a multi-material setting.

Lattice-based meshing

Lattice-based meshing approaches are build upon the Marching Cubes algorithm [LC87] applied to multi-material boundaries [WS03]. An initial high-resolution regular mesh can be generated from the volume data, where the elements are split according to precomputed boundary configurations until the input domain is represented well (Figure 2.4). However, those methods tend to produce dense meshes and ill-shaped tetrahedra near the boundaries. *Isosurface stuffing* uses a similar paradigm to represent single material domains but ensures theoretical bounds for the tetrahedra dihedral angles [LS07]. Inspired by this strategy, Bronson et al. were able to offer similar guarantee for labeled volume data [BLW13]. Nonetheless, the resolution depends on the input grid which prevents multi-resolution meshing and generates cumbersome meshes.

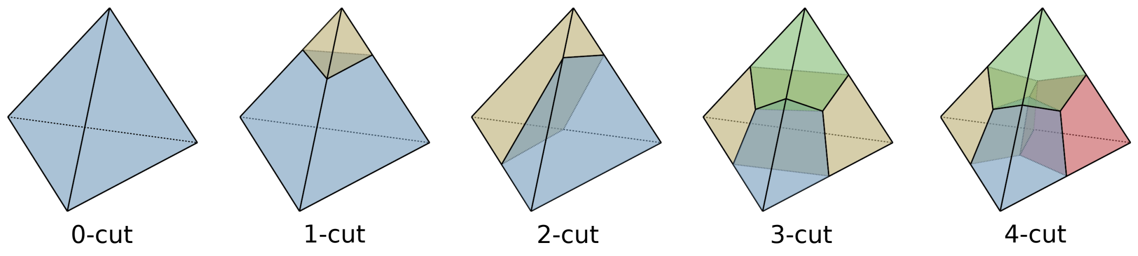


Figure 2.4: The 5 unique interface topologies determined by the number of cut-points present on a lattice tetrahedron. Image courtesy of [BLW13].

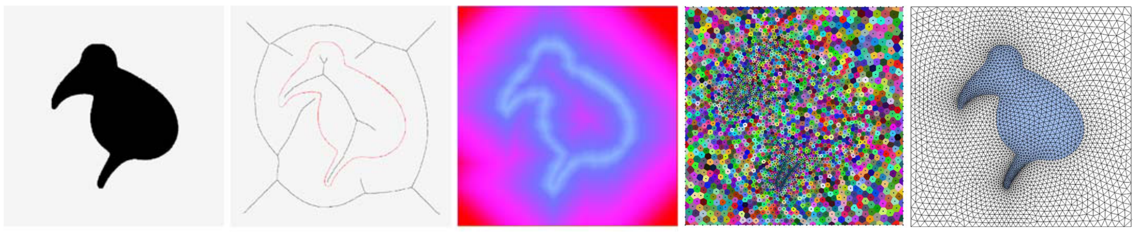


Figure 2.5: A 2D example of [DVS+09], from left to right: original image, medial axis of the shape, a density function generated using the medial axis, an approximated Centroidal Voronoi Diagram built using the density function and the object boundaries dualized into the resulting mesh.

Variational methods

The main idea of variational approaches is to insert a set of particles in the input domain and compute their distribution using a non-linear energy optimization. This energy is composed of a data term, to conform to the input domain boundaries, and a quality term, to get an uniform point distribution. The output mesh is a triangulation generated from the resulting point distribution [CA97]. A similar process can be performed using a mesh as an input: the mesh vertices positions are optimized to fit the input domain [GS11]. Dardenne et al. proposed to distribute the points using a density function computed from the medial axis of the input voxel grid. The mesh is built using an approximate Centroidal Voronoi Diagram taking the 2-junctions into account [DVS+09]. In order to represent the 1- and 0-junctions, Meyer et al. distribute points in a hierarchical manner starting by the 0-junctions and finishing by the volume [MWK+08]. These methods provide high-quality results but depend strongly on the initial setting and are computationally expensive.

2.1.3 Remeshing and quality improvement

Since feature preservation and quality constraints are tedious to combine, ill-shaped tetrahedra are likely to be generated during the meshing process, hence, a quality improvement step is often necessary.

Improvement

Chen proposed the Optimal Delaunay Triangulation (ODT) approach, that improves the Delaunay triangulation quality by reallocating the inside vertices positions to minimize the interpolation error defined as the mesh quality [Che04, CX04]. This process is performed using a unified functional optimization and generates isotropic meshes. Note that this method was adapted for tetrahedral meshing of 3D domains in [ACSYD05]. Tournois et al. extended this energy to optimize the boundary vertices' positions and proposed a method that couples refinement and optimization strategies to guide the insertion of Steiner points and obtain high-quality meshes directly [TWAD09]. This method called Natural ODT (NODT) fails to handle input meshes with sharp creases. Furthermore, Delaunay-based refinement processes do not prevent the apparition of slivers - meeting the

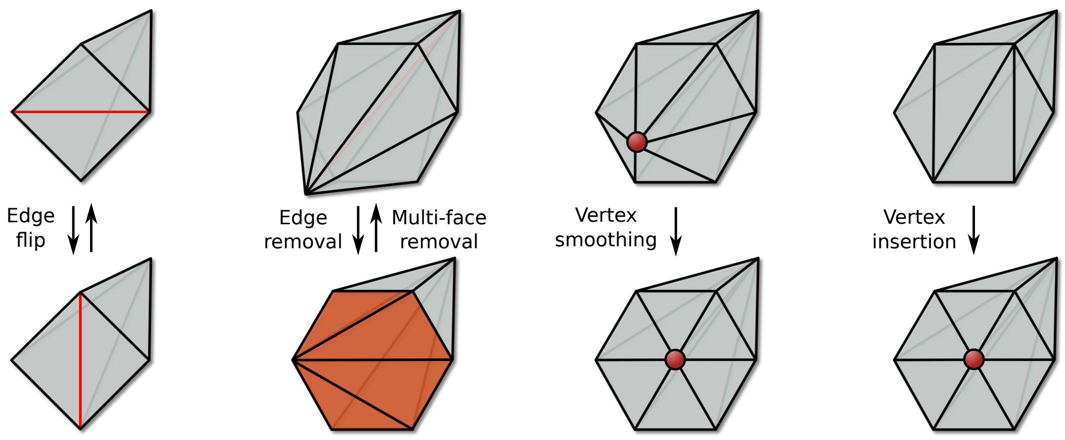


Figure 2.6: Local topological operations performed to improve the tetrahedra quality.

Delaunay constraints but with a poor quality - making a post-processing step inevitable. A first solution, called sliver *exudation*, turns the triangulation into a weighted Delaunay triangulation [CDE+00]. An alternative approach is to *perturb* slivers through vertex reallocation and Delaunay connectivity update [TSA09].

In a more general setting, the quality can be improved using optimization-based smoothing and local topological operations such as edge flip or removal as proposed in [FOg97]. This method was further improved by Klingner et al. through additional operations such as vertex insertion and multi-face removal combined with a roll back mechanism to cancel operations that decreases the quality [KS07]. Figure 2.6 illustrates the optimization operations.

In order to cover a large range of resolutions and mesh more densely regions of interest, both simplification and refinement schemes, generating high-quality meshes, are necessary.

Refinement

Except for Delaunay refinement processes, only a limited number of triangle mesh refinement techniques have been extended to tetrahedral meshes, and only a few of them take multi-material features into account. Burkhart et al. proposed a topology-based refinement operator inspired by the $\sqrt{3}$-subdivision scheme [BHU10]. This method combines the subdivision process with edge flip and vertex smoothing operations in order to maintain the mesh's quality and preserve features, in a single material setting.

Simplification

For visualization purposes, the simplification of tetrahedral meshes has been studied extensively, especially methods based on edge contraction [GZ05, ILGS06] inspired by surface simplification techniques [Hop96, GH97]. These methods successively contract edges until the desired resolution or the maximum tolerated error is reached. Each edge collapse oper-

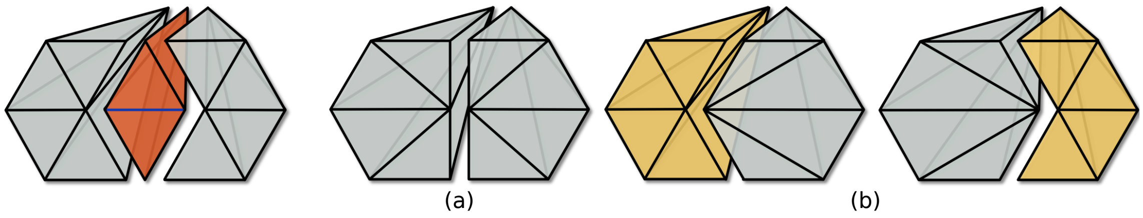


Figure 2.7: **Edge collapse operation.** A collapse removes red tetrahedra and one of the edge vertices. Depending on the heuristic: (a) mid-point collapse, both vertices and their incident tetrahedra are affected, or (b) collapse toward one of the vertices, only the tetrahedra incident to the moved vertex are affected.

ation removes one vertex and the tetrahedra around the edge (Figure 2.7). The decimation process is guided by the minimization of an error metric combined with heuristics allowing to meet local topological and geometrical criteria. For instance, an edge can be collapsed “entirely” at its mid-point, and then all the elements incident to both edge vertices are affected, or toward one of the edge vertices, and only the elements incident to the removed vertex are affected (Figure 2.7, yellow tetrahedra remain unchanged). Indeed, naive operations may induce tetrahedra intersection or inversion and can even change the topology of the domains.

We can prevent the inversion of tetrahedra easily by discarding all operations that would generate tetrahedra with negative volumes. On the other hand, checking for intersection of the mesh’s outer boundary tetrahedra is computationally expensive. Since these problems occur in concave regions only, Kraus et al. proposed to fill the mesh’s convex hull with so-called *imaginary tetrahedra* [KE02]. Therefore, preventing the inversion of these tetrahedra - that are part of the triangulation but not of the represented domain (Figure 2.8) - ensures that no self-intersections occur. Nevertheless, the local criteria proposed are not sufficient to preserve the mesh topology and the convex hull.

For triangle meshes, Hoppe proposed robust heuristics to perform feature-aware collapse operations allowing to represent the surface mesh substructure, such as polylines and boundaries, accurately [Hop96]. For tetrahedral meshes, Dey et al. first introduced *link conditions* allowing to detect and discard the operations that would change the mesh topology [DEGN99]. The authors demonstrate the validity of their approach for 3D complexes without borders, by building an extended complex on which the operations are performed. To do so, the outer boundary facets of the mesh are linked to a dummy vertex, hence, creating an extended complex without borders. This method was extended to handle multi-material meshes using additional link conditions to preserve 1- and 0-features [TNB10, VBHH11]. Nevertheless, quality is not the primary concern of these methods, therefore the simplified meshes are not suitable for simulations.

To meet the quality and size constraints simultaneously, a remeshing process is preferred since it allows the user to navigate between different resolutions easily.

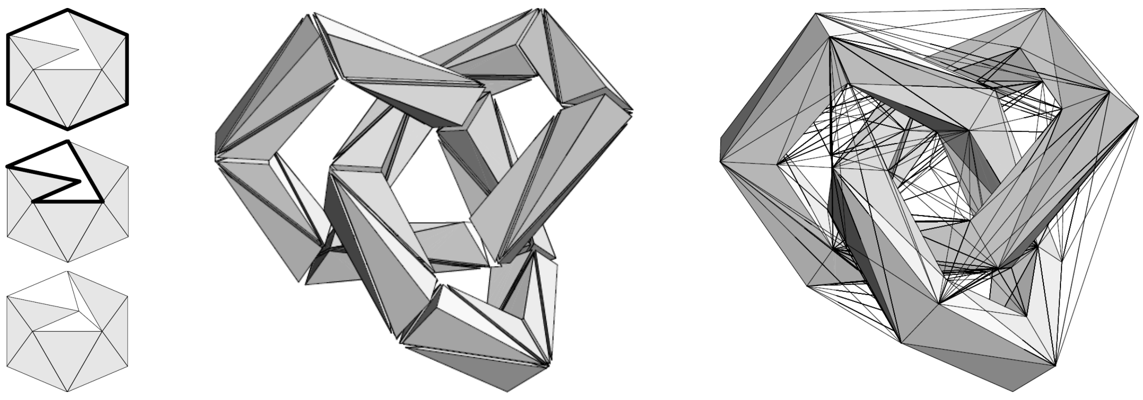


Figure 2.8: (Left) 2D example, first, the convex hull is computed then the concave parts are identified and triangulated. (Right) Non convex triangulation and the generated imaginary tetrahedra in wire-frame [KE02]

Remeshing

An efficient approach for remeshing is to interleave the simplification/refinement process with local operations that improve the mesh quality. In this line of work, Cutler et al. proposed a method to generate high-quality multi-material tetrahedral meshes at different resolutions by performing local topological operations such as elements swapping, vertex smoothing and vertex addition to meet the quality and edge length criteria [CDM04]. While this method allows to preserve boundary surfaces using a volume-based error metric, it does not take into account the 1- and 0-junctions. Furthermore, the operation minimizing the error is performed, which is computationally expensive since the cost of each operation needs to be evaluated at each step.

Similar efficiency and feature preservation issues were tackled for the remeshing of triangular meshes. Following [KBpS00, VRpS03], Botsch and Kobbelt proposed an iterative remeshing method to generate isotropic high-quality triangular meshes using simple local operations performed in a predefined order [BK04]. The following operations are performed iteratively for a target edge length. First, the edge lengths are equalized using edge split and collapse operations. Second, the valences of the vertices are regularized by flipping edges. Finally, the vertex distribution is improved using tangential smoothing, as described in [Bot05]. These operations are performed successively regardless of the local quality. The authors claim that 5-10 iterations are sufficient to generate high-quality remeshed surfaces. To improve the mesh isotropy, the Voronoi area of the vertices are equalized using area weighted smoothing. As a final step, the resulting vertices are projected back onto the input surface in order to represent the input shape accurately. Furthermore, this method allows to preserve detected or user defined 1- and 0-features, by adding simple heuristics that do not have a significant effect on the performance.

As we have seen in this section, no existing method allows us to generate high-quality meshes at different resolutions efficiently, while preserving the multi-material features of different dimensions. Most existing meshing processes allow to generate a mesh within several minutes or hours, and in some cases to refine this mesh (Delaunay triangulation), but not to simplify it. Simplification methods can preserve these features but do not meet the quality requirements.

We propose a remeshing method inspired by Botsch et al. [BK04] that performs local topological operations in a predefined order. To avoid self-intersections robustly, we add imaginary tetrahedra to the triangulation [KE02]. To unify the heuristics, we perform all operations on an extended complex by linking the outer facets of the triangulation to a dummy vertex [DEGN99]. We propose feature-aware operations using heuristics stemming from link conditions [VBHH11].

2.2 Multi-Material Adaptive (re)mesher

In this section, we present our iterative Multi-Material Adaptive (M-MAd) (re)mesher algorithm for 3D triangulations, providing high-quality meshes at the desired resolution while preserving features such as multi-material boundaries. Our method can also be applied on segmented voxel grids by initializing a trivial high-resolution tetrahedral mesh generated from the input grid. Our algorithm allows to generate uniform meshes as well as adaptive ones for which the spatially-varying resolution is driven by a sizing field based on a distance field. In particular, this adaptive version provides elements with increasing size when located away from the boundaries in order to minimize the number of elements, while preserving accurate boundaries, therefore resulting in an isotropic adaptive mesh. We propose to use a *feature complex*, inspired by [AA09] and similar to the one proposed in [DJL12] to define feature-dependent topological operations coupled with hierarchical smoothing operations using a Moving Least Squares (MLS) representation of the boundaries. We propose to preserve additional features which are either provided by the user as a set of polylines, or detected on the domain boundary. As opposed to Delaunay-based methods, our approach allows to navigate easily in the parameter space, by ruling the target edge length interactively. Furthermore, any type of multi-material mesh can be processed by our method.

2.2.1 Overview

Given a target length l, our remeshing method can be summarized as follows:

Pre-processing : detect the boundaries and features to preserve, add a layer of imaginary tetrahedra to prevent self-intersections, and build the extended complex by connecting the boundary facets to a dummy vertex,

1 **split** any edge longer than $e_{\max}$,

2 **collapse** any edge shorter than $e_{\min}$,

3 **flip** edges to minimize the average valence (on the boundaries) and to optimize locally the dihedral angle distribution (in the volume),

4 **filter** to relocate vertices, taking features into account,

5 go to step **1** unless the target resolution is reached,

Post-processing : optimize to remove slivers and improve the mesh quality.

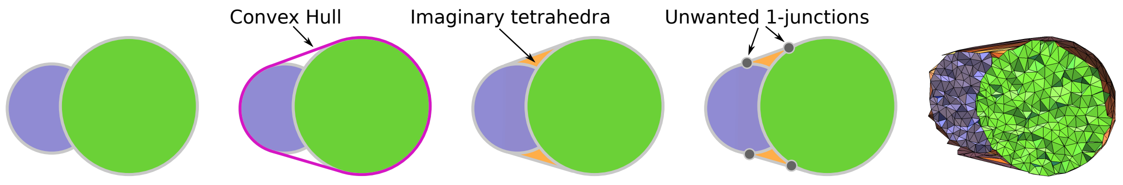


Figure 2.9: Directly filling the space delimited by the convex hull and the mesh boundaries with imaginary tetrahedra can create unwanted features.

For a target edge length, we use constant values $e_{\max} = 4l/3$ and $e_{\min} = 4l/5$, which are the optimal thresholds to obtain a uniform mesh, since splitting an edge that verifies $|e_{max} - l| > |\frac{1}{2}e_{max} - l|$ and collapsing an edge that verifies $|e_{min} - l| > |\frac{3}{2}e_{min} - l|$ reduces the deviation from the target length, see [Bot05] for more details. Optionally, we tailor l in a spatially varying fashion w.r.t. a sizing field capturing the distance to the boundary. In our experiments, performing the operation in this order gave the best results. Note that, when the input mesh has the aimed resolution and is processed for regularization and quality improvement purposes, the target length is set to a value slightly lower than the current average edge length to introduce perturbations in the optimization [BK04].

2.2.2 Representation

We note the set of labels associated with either an input multi-material 3D triangulation or a voxel grid as $\mathcal{L} = \{l_n\}_{n \in I_{\mathcal{L}}} \subset \mathbb{Z}$. By convention, null values represent the background, i.e. the parts of the data that do not belong to the represented domain.

Labeled tetrahedral mesh

The mesh is noted $M = \{V, E, T\}$ with $V = \{v_i\}_{i \in I_V} \subset \mathbb{R}^3$ its vertices, $E = \{e_{ij}\}$ its edges connecting adjacent vertices v_i and v_j and $T = \{t_k\}_{k \in I_T}$, its tetrahedra indexed over V. We call the triangular faces of the tetrahedra *facets*. We note $T_1(v_i)$ (resp. $F_1(v_i)$) the set

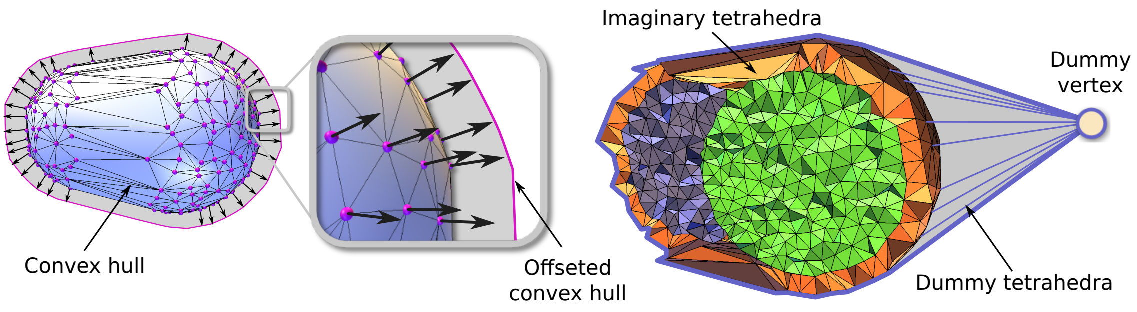


Figure 2.10: **Pre-processing.** Addition of a layer of imaginary tetrahedra to the input triangulation and extension of the complex through the addition of dummy tetrahedra subsequent to the connection the outer facets to the dummy vertex.

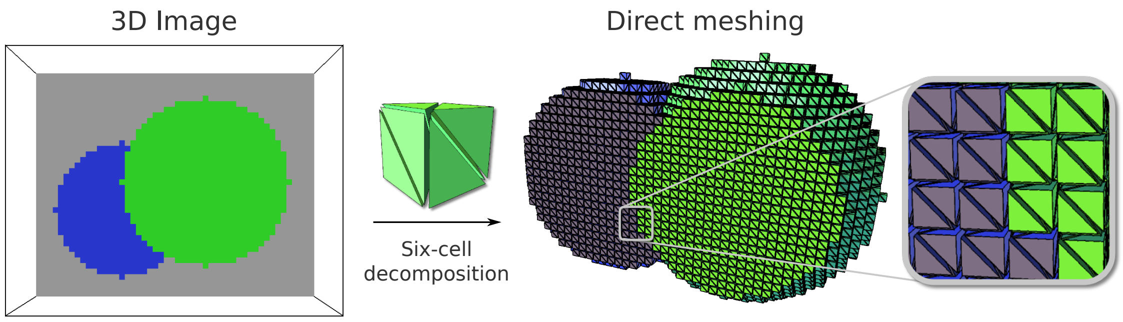


Figure 2.11: Segmented input voxel grid (Left) and resulting multi-domain tetrahedral mesh (Right) with its six-cells decomposition.

of tetrahedra (resp. facets) incident to a vertex v_i and $T_1(e_{ij})$ (resp. $F_1(e_{ij})$) the set of tetrahedra (resp. facets) around an edge e_{ij}.

The input mesh is typically composed of n subdomains, with $L(t_k) = l_i$ denoting the label associated with a given tetrahedron t_k. We add a special imaginary subdomain which ensures that the remeshing process will not introduce self-intersections of the represented domain [KE02]. Since these elements will be processed like any other subdomain, naively filling the convex hull of the input vertices generates unwanted features (Figure 2.10). To tackle this issue, the input mesh is embedded into an inflated convex hull to ensure that the input domain is surrounded by at least one layer of tetrahedra (Figure 2.10, left). To do so, we duplicate the elements of V laying on the convex hull and displace them in their outer normal direction before triangulating them, using a Restricted Delaunay Triangulation accounting for the original mesh. The displacement factor is typically set to 4% of the domain's bounding box.

To unify the representation of inter-domain boundaries and 3D surfaces, we add another fictitious subdomain by connecting the outer boundary facets (i.e., of the offseted convex hull) to a dummy vertex. These *dummy* tetrahedra allow us to process the mesh's outer boundary like any other. The additional *imaginary* tetrahedra (Figure 2.10, orange elements) have a null label $L(t_k) = 0$ (i.e., background), the *dummy* tetrahedra have a negative label $L(t_k) = -1$, while all other are marked with a positive label, and are later called *domain* tetrahedra.

Labeled voxel grid input

We can also initialize M directly from a segmented 3D voxel grid (e.g., MRI) defined by a function that associates each voxel grid in $\mathbb{N}^3$ with its labels. In order to preserve all boundaries, we define M through a six-cell decomposition of all the voxels contained in the bounding box of the domain (Figure 2.11). Again, we unify internal and external boundaries by, first, inflating the bounding box and associating a null label to the so-defined imaginary tetrahedra and, secondly, connecting their outer facet to a dummy vertex to generate dummy tetrahedra to which we associate the label -1. The resulting high-resolution mesh M reproduces the grid topology with aliased boundaries.

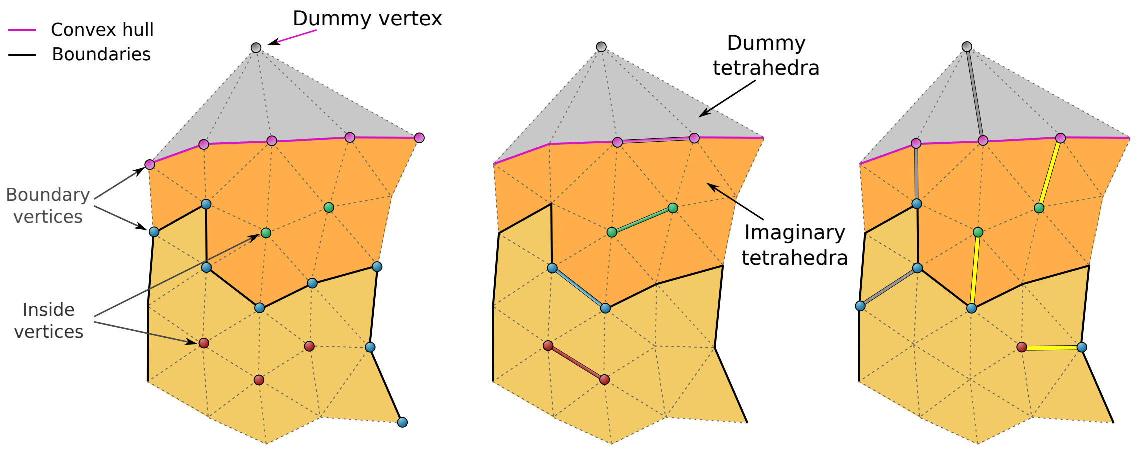


Figure 2.12: **Classification.** (Left) Vertex types, (Center) Boundary and inside edges and (Right) Mixed and critical edges.

2.2.3 Elements notations

We define here the simplices notation used in the remainder of the document (Figure 2.12).

Facets shared by two tetrahedra that belong to different subdomains are *boundary facets*. The vertices (resp. edges) of those facets are *boundary vertices* (resp. *boundary edges*). For each vertex v_i, we note $S(v_i) \subset L$ the set of labels incident to v_i:

$$S(v_i) = \{L(t_k)_{t_k \in T_1(v_i)}\}. \tag{2.1}$$

Similarly, we define $S(e_{ij})$ for an edge. *Inside* (resp. *boundary*) vertices verify $|S(v_i)| = 1$ (resp. $|S(v_i)| > 1$), except for the dummy vertex. We identify four types of edges:

- *inside edges* connect two inside vertices (Figure 2.12, green and red),
- *mixed edges* connect an inside vertex and a boundary vertex (Figure 2.12, yellow),
- *boundary* edges connect two boundary vertices and have incident boundary facets (Figure 2.12, pink and blue),
- all other edges are *critical* edges (Figure 2.12, gray).

2.2.4 Feature detection

In order to conform to the input boundaries and perform feature-aware operations, we identify feature elements of different dimensions that together form a *feature complex* similar to a point-sampled cell complex [AA06] or a PSC [DJL12].

Feature elements. The *boundary facets* are the complex elements of dimension 2. The *feature edges*, which are boundary edges at the intersection of three or more labels ($|S(e_{ij})| > 2$) are the elements of dimension 1. Vertices with three or more incident labels

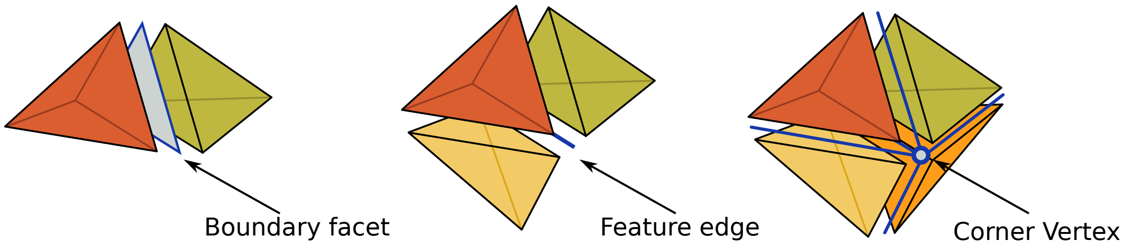


Figure 2.13: **Feature elements.** *(Left)* **Boundary facet**: facet between tetrahedra of different labels, *(Center)* **feature edge**: edge between three or more labels and *(Right)* **corner vertex**: vertex at the intersection of four or more labels.

($|S(v_i)| > 3$) and at least three feature edges in their one-ring are *corner vertices* (Figure 2.13) and are elements of dimension 0.

The input domain's n-junctions - with n the dimension in the feature complex - are represented in the mesh through the feature complex (see Figure 2.14) as follows:

- **2-junctions** are sets of connected boundary facets located at the interface between two subdomains. Each 2-junction forms a triangle surface patch, where its orientation is deduced from one of its incident subdomains and delimited by 1-junctions (if present in the data).
- **1-junctions** are sets of connected feature edges sharing the same set of incident subdomains. Each 1-junction forms a polyline at the intersection of 2-junctions with different pairs of subdomains.
- **0-junctions** are corner vertices and are located at the intersection of 1-junctions.

We refer to boundary vertices as vertices that lie on 2-junctions but which do not belong to a 1- or 0-junction as a *surface vertex*. The vertices that lie on 1-junctions and that are not corner vertices are referred to as *feature vertices*. All vertex types are summarized in Table 2.1. Additionally, boundary edges linking two surface vertices are called *surface edges*. We do not need to build the feature complex since the elements of the feature complex are already present in the input mesh.

Type of v_i	$\lvert S(v_i)\rvert$	# of incident feature edges
Inside	1	-
Surface	2	-
Feature	>2	1 or 2
Corner	>3	>2

Table 2.1: **Vertex types**. The number of subdomain indices and feature edges incident to a vertex give its type.

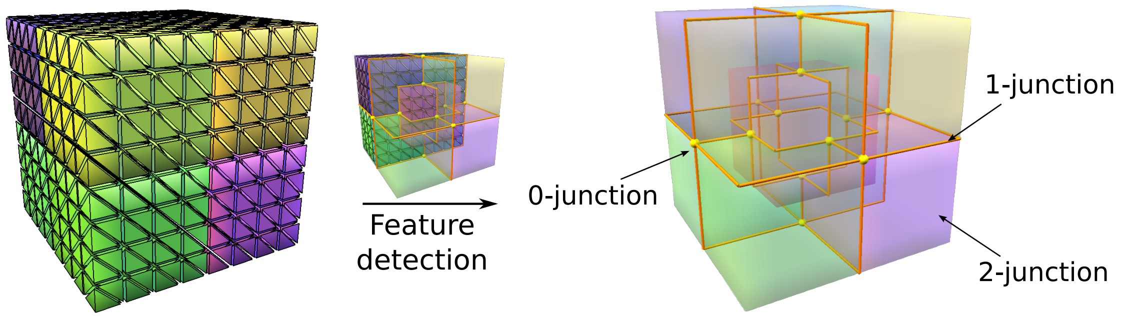


Figure 2.14: **Feature detection.** The surface patches represent 2-junctions, the polylines 1-junctions and the spheres 0-junctions.

2.2.5 Smooth Modeling

We model piecewise smooth boundaries by fitting 2-junctions with MLS surfaces. These meshless surfaces are defined using a local projection operator allowing to evaluate the position of a point on a local surface reconstructed from unorganized point samples. The smoothing power depends on the weight function used to combine the employed samples to compute the projection and on a scaling factor defining the size of the explored neighborhood to find these samples.

Different types of MLS surfaces can be used depending on the user needs. Alexa et al. [ABCO+01] first provided a definition of smooth manifold surface from a set of points, also called Point Set Surface (PSS), based on the work presented in [Lev03]. The projection on the surface is recovered by locally fitting a plane near the evaluation point using a non linear minimization of the square distance to the samples. This definition was proven equivalent to a simple weighted local combination of positions and normals used to define the projection plane [AA04]. The projection can become unstable under low sampling rates and in the presence of high curvature. To tackle this issue Guennebaud and Gross proposed Algebraic Point Set Surfaces (APSS) fitting algebraic spheres instead of

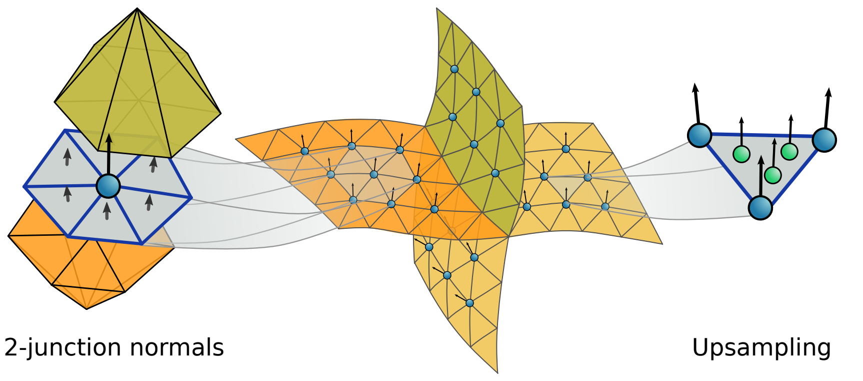


Figure 2.15: **Smooth interfaces.** (Left) Consistently oriented normals, (Right) surface patch up-sampling and consistently oriented normals used to build the MLS surface representation of 2-junctions.

planes [GG07]. Alexa and Adamson proposed Hermite Point Set Surfaces (HPSS) which definition avoids shrinking of the input model. Instead of projecting the point on a single plane, it is projected on a plane per considered neighboring sample. Then the position is computed using a weighted averaging of these projections resulting in a Hermite combination [AA09]. These definitions generate smooth surfaces, therefore, tend to remove local features. To tackle this issue, a feature preserving Robust Implicit Moving Least Square (RIMLS) operator was proposed in [OGG]. Its non-linear kernel regression makes it more robust to outliers hence easier to identity and protect features. Most recently, Guillemot et al. proposed Non-Local Point Set Surfaces (NLPSS) using similarity measures to propose a non-local point set definition possibly extending any PSS definition [GAB12]. The non-local point set provides additional information that allows us to recover missing data and enhance the representation by increasing the signal to noise ratio. We refer to [CWL$^+$08] for a recent survey on MLS surfaces.

Any of these aforementioned definitions can be used to represent the input domain boundaries. We define each MLS surface using a dense point sampling of the 2-junctions, excluding feature vertices, together with their consistently oriented vertex normals (Figure 2.15 left). To do so, the input surface patches are up-sampled in an area weighted fashion (Figure 2.15 green vertices).

The MLS representation allows us to remove the input noise and acts as a regularizer at each step of our remeshing algorithm. Furthermore, since this representation is independent from the current mesh state the user can easily navigate in the parameter space for instance go from a low resolution mesh to a high-resolution one without starting over.

2.3 Operations

Special care needs to be taken when processing boundaries since the used local operations, described in this section, do not result in the preservation of the feature's topology. Therefore, following [CDM04] and [TNB10, VBHH11], we define feature-aware heuristics and conditions depending on the feature complex hierarchy.

2.3.1 Classification

The main idea is that only interior vertices should be allowed to be relocated freely in the volume, while n-junction vertices should be moved along their n-junction, in order to preserve the latter (the same idea was used in 2D for the tangential Laplacian [Bot05], resulting in the preservation of the surface features).
To fit these constraints, we define three different conditions between the pairs of connected vertices leading to different heuristics: *similarity*, *inclusion* and *exclusion* (Table 2.2).

- The similarity condition is met for v_i and v_j, that are not corner vertices, if $|S(v_i)| = |S(v_j)|$ and $|S(e_{ij})| = |S(v_i)|$ - i.e. for inside vertices, surface vertices linked by a surface edge, feature vertices linked by a surface edge and feature vertices linked by a feature edge.

Condition	Type of v_i	Type of v_j	Type of e_{ij}
Similarity	inside	inside	inside
Similarity	surface	surface	surface
Similarity	feature	feature	feature
Inclusion	inside	boundary	mixed
Inclusion	surface	feature or corner	surface
Inclusion	feature	corner	feature
Exclusion	boundary	boundary	critical (i.e. not boundary)
Exclusion	feature	feature or corner	surface
Exclusion	corner	corner	-

Table 2.2: **Conditions.** All operations depend on the type of the edge and its vertices. Here, we define conditions to match in order to preserve the subdomain's topology.

- The inclusion condition if $|S(v_i)| > |S(v_j)|$ and $|S(e_{ij})| = |S(v_j)|$ and v_j is not a corner vertex - i.e. for mixed edges, edges linking a surface vertex and a feature or corner vertex and feature edges linking a feature and a corner vertex.
- Otherwise, the exclusion condition is met - i.e. critical edges, edges between feature or corner vertices that do not belong to the same 1-junction or linking two 0-junctions.

The pair of vertices meeting the similarity condition affect each other. For the ones meeting the inclusion condition, only the vertex with the highest dimensionality in the feature complex, or the one not belonging to it, will be affected. And finally, for the exclusion condition, none of the vertices will be affected since it would create undesired merging of vertices that belong to different 2- or 1-junctions and fail to preserve small local features of the subdomains. Table 2.3 summarizes the conditions and the derived heuristics. In

Condition	$|S(v_i)|$	$|S(v_j)|$	$|S(e_{ij})|$	$\# f_i$	$\# f_j$	Update v_i	Update v_j
Similarity	1	1	1	-	-	yes	yes
Similarity	2	2	2	-	-	yes	yes
Similarity	$n_i > 2$	$= n_i$	$= n_i$	<3	<3	yes	yes
Inclusion	1	>1	1	-	-	yes	no
Inclusion	2	>2	2	-	-	yes	no
Inclusion	n_i>2	$n_j > n_i$	n_i	<3	-	yes	no
Exclusion	n_i	n_j	$\neq min(n_i, n_j)$	-	-	no	no

Table 2.3: **Heuristics.** Summary of our feature aware heuristics. We can easily detect if the vertices will be affected during the current operation depending on their number of incident subdomains and of incident feature edges noted $\# f_i$, i.e. on their dimension in the feature complex.

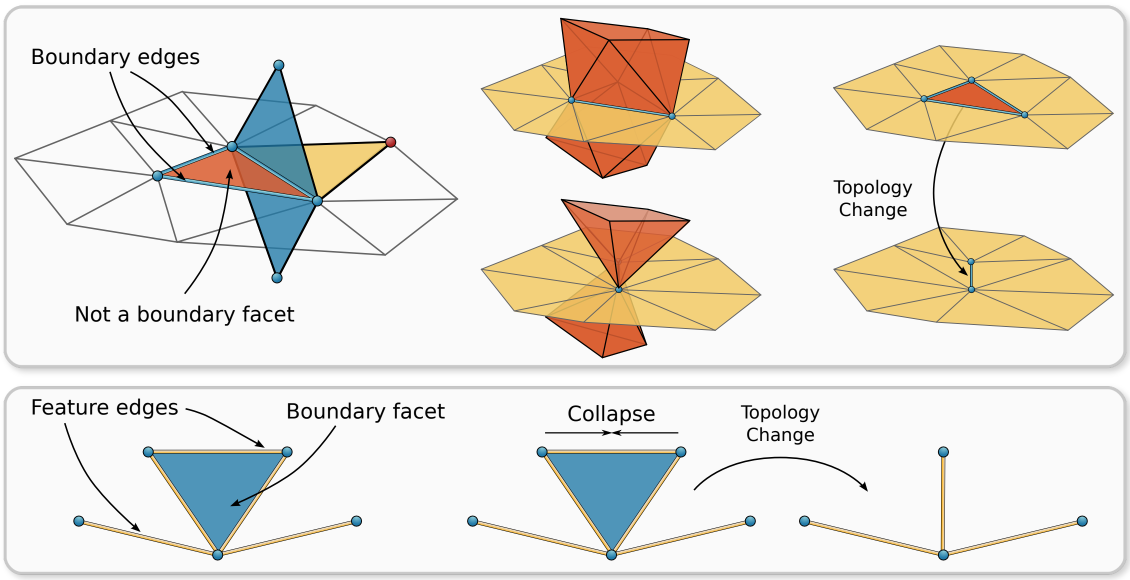


Figure 2.16: **Topology exceptions.** Additional tests to preserve the topology of the 1- and 2-junctions in small tubular regions. (Top row) Example of a collapse operation creating a pinched boundary. Therefore, the operation is discarded if the processed boundary edge has an incident facet that is not on a boundary (red) but all three of its edges are on a boundary (blue). (Bottom row) Example of a collapse operation changing the topology of a 1-junction. Therefore, the operation is discarded if the processed feature edge has an incident boundary facet (blue) and all three of its edges are on a 1-junction (yellow).

practice, except for corner vertices, a vertex v_i will only be affected by the vertices v_j of its one-ring if $|S(v_i)| \geq |S(v_j)|$ and $|S(e_{ij})| = |S(v_j)|$.

Unfortunately, these conditions do not prevent a change of topology in the configurations illustrated in Figure 2.16. The three edges around the red facet meet the similarity condition but a collapse operation would change the topology of the subdomains, creating a pinched boundary. Similarly, the collapse of any of the three feature edges around the blue facet would change the topology of the 1-junction. Therefore, we perform two additional *topology tests* for boundary and feature edges to detect these configurations. For the first topology exception, none of the vertices are updated if at least one facet around the edge is not a boundary facet (red facet in Figure 2.16) and its three edges are boundary edges (blue edges in Figure 2.16). For the second one, none of the vertices are updated if at least one facet around the edge is a boundary facet (blue facet in Figure 2.16) and its three edges are feature edges (yellow edges in Figure 2.16).

Now that we have defined the feature-aware heuristics, we describe our remeshing process in the following.

2.3.2 Split edges

In the first step of our iterative procedure, all the edges with a length superior to $e_{\max}$ are split i.e. a vertex v_k is added at their mid-point dividing every tetrahedron around

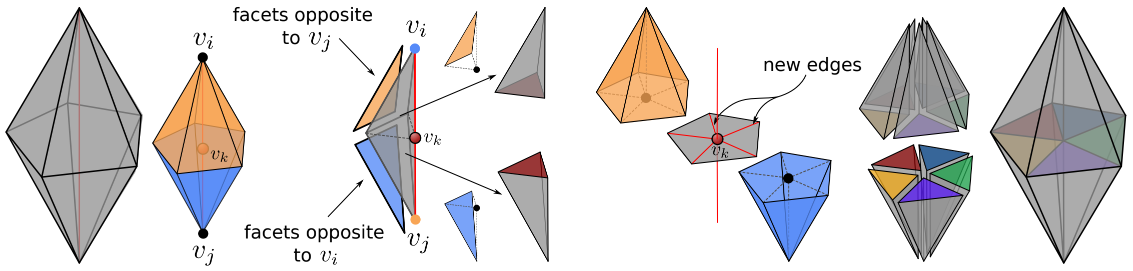


Figure 2.17: **Split.** (Left) A vertex v_k is added at the edge's mid-point. (Middle left) New tetrahedra are made of the facets opposite to the edge's vertices (yellow and blue) and the new vertex (red). They are assigned the same label as the one they are subdividing. (Middle right) New edges incident to v_k in red. (Right) Resulting local triangulation with every tetrahedron around the initial edge divided into two.

the edge into two (Figure 2.17). Note that the two new tetrahedra are assigned the same label as the one they are subdividing. The set of labels of the added vertex v_k is the set of labels of the tetrahedra around the current edge, i.e. $S(v_k) = S(e_{ij})$. As only insertions are performed at this step, no particular feature preservation rules are required.

2.3.3 Collapse edges

Now that the long edges have been split, we remove short edges is order to get a uniform edge length close to the target one. To do so, we collect all the edges to collapse with a length smaller than e_{min}, and proceed to perform the possible collapses. Indeed, not all edges can be collapsed without modifying the topology of the subdomains, inverting tetrahedra or even resulting in an invalid data structure [Hop96, DEGN99].

Feature preserving collapse. We start by evaluating which condition the current edge meets to define which type of collapse to perform according to Table 2.3:

- similarity: mid-point collapse (both vertices are affected),
- inclusion: toward the vertex with the highest number of subdomain - i.e. with the lowest dimension in the feature complex (only the vertex with the smallest number of subdomain is affected),
- exclusion: no collapse, since a contraction would change the topology of the subdomains (none of the vertices or tetrahedra are affected).

The tetrahedra incident to the affected vertex, or vertices, are set to be updated except the ones incident to the current edge, which are set to be removed (Figure 2.18). In order to ensure the validity of the operation, we perform the following tests:

1. all the updated tetrahedra have a positive volume,
2. all the new edges are shorter than e_{max},

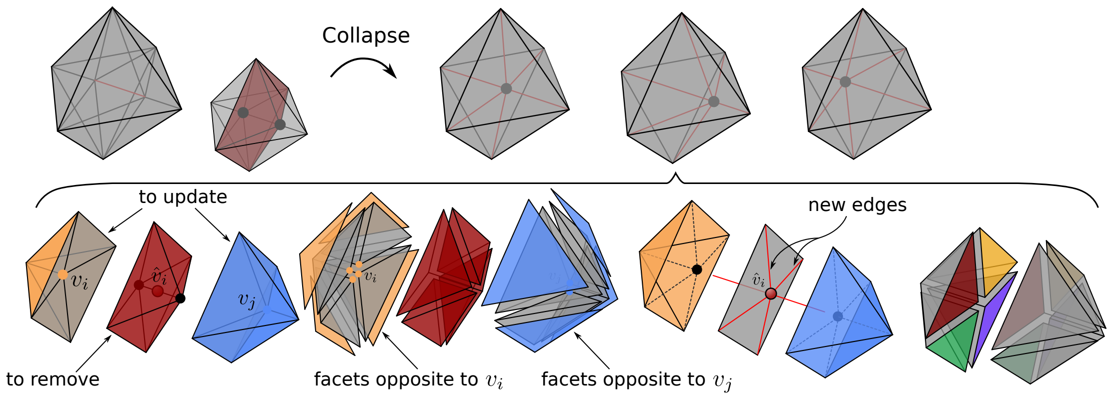


Figure 2.18: **Collapse.** (Top) Three types of collapse: mid-point, toward v_j and toward v_i. (Bottom) Example of mid-point collapse: (Left) tetrahedra around the edge to collapse are set to be removed, the remaining tetrahedra incident to the edge's vertices to be updated, (Middle left) updated tetrahedra correspond to the ones formed by the facets opposite to the edge's vertices (yellow and blue) and a vertex at the edge mid-point (red circle), (Middle right) new edges and updated edges incident to v_k in red and (Right) resulting local triangulation.

3. none of its incident non boundary facets have three boundary edges,

4. none of its incident boundary facets have three feature edges.

The third test (resp. fourth) - i.e. topology tests - is performed for boundary (resp. feature) edges only and avoids the occurrence of *pinched* boundaries (resp. 1-junction topology changes) in tubular regions, as illustrated in Figure 2.16. If these four constraints are met the operation is performed. Some of the new edges, incident to the remaining vertex, may be shorter than $e_{\min}$. Therefore, we evaluate their length and set the ones that do not respect the length criteria to be collapsed. This process is repeated until all edges are either smaller than $e_{\min}$ or impossible to collapse.

Once the overall edge length is close enough to the target one, the local connectivity is changed in order to minimize the average valence and optimize locally the dihedral angle distribution, using an edge flip operation described in the following.

2.3.4 Flip edges

Flipping an edge of a tetrahedral mesh induces more changes than for a triangular mesh. Indeed, the operation changes the number of tetrahedra adjacent to the edge, except when there are exactly two or four adjacent tetrahedra. It removes an edge and replaces it with facets. The flip operation, also called *edge removal*, has been first proposed in [dlG95] and further studied in [FOg97, KS07], as a mesh improvement strategy. For each edge e_{ij}, we explore the space of possible flip operations and perform the one that offers the best quality for $T_1(e_{ij})$, i. e. that maximizes the worst dihedral angle, for inside edges, and averages the boundary vertices valences, for boundary edges. In the mean time, the operation must

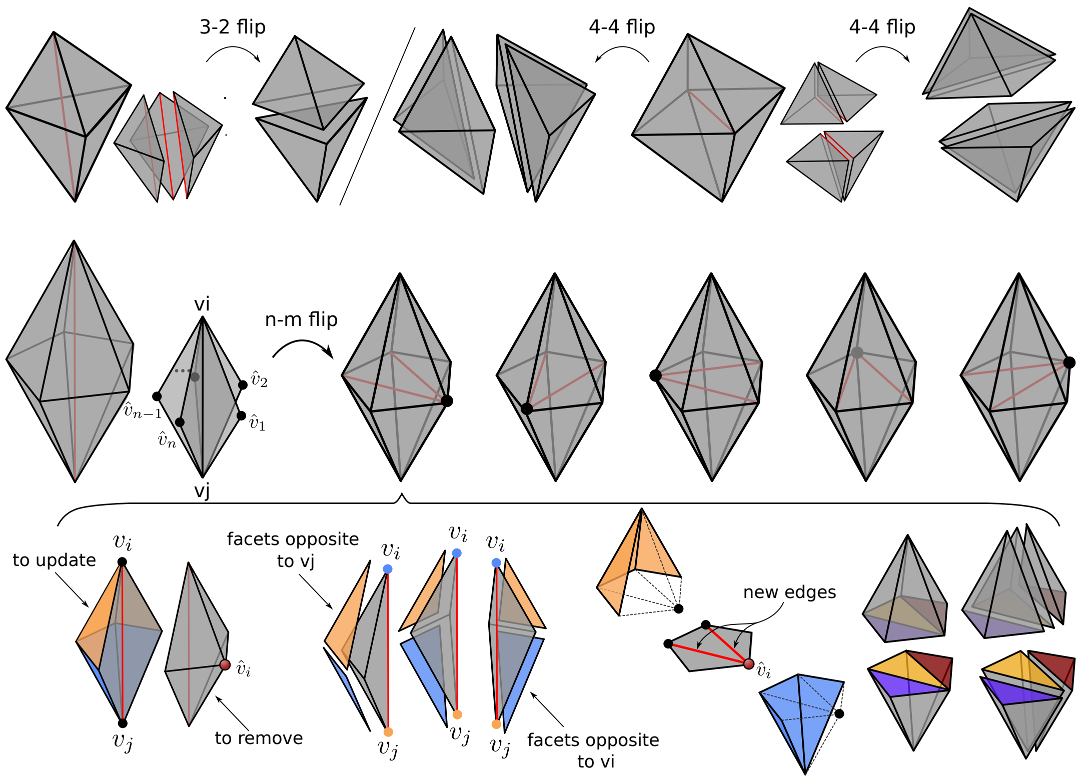


Figure 2.19: **Flip of non-boundary edges.** (Top left) 3-2 flip, (Top right) 4-4 flip and (Bottom) n-m flip.

not generate any inverted tetrahedra or edges that already exist. In the following, we first describe the edge flip operations without feature preservation constraints, then the flip of boundary edges.

Flip operations. The flip operation depends on the number of tetrahedra around the edge, therefore, we note *n-m flip* the flip operation with n, the number of input tetrahedra, and m, the resulting number of tetrahedra. We note $\hat{V}_1(e_{ij}) = \{\hat{v}_1, ..., \hat{v}_n\}$ the set of vertices around the edge e_{ij} in counter clockwise order, i.e. the vertices $\hat{v}_i$ of the facets $F_1(e_{ij})$ around the edge that are different than v_i and v_j (Figure 2.19).

If $n = 3$, the only possible configuration removing one tetrahedra is the 3-2 flip (Figure 2.19 top left). The current edge is removed and replaced by a facet formed by the three vertices of $\hat{V}_1(e_{ij})$.

If $n = 4$, there are two possible configurations, both being 4-4 flips (Figure 2.19 top right). The current edge is replaced either by an edge between $\hat{v}_1$ and $\hat{v}_3$ or between $\hat{v}_2$ and $\hat{v}_4$ depending on the flip criterion.

If $n > 4$, there are n possible configurations, being n-m flips, all with $m = 2n - 4$ (Figure 2.19 bottom). The current edge is removed and $n - 2$ facets are added. Since all new facets are incident to $\hat{v}_i$, we call it a flip *toward* the vertex $\hat{v}_i$. We evaluate the validity and the quality of all possible configurations - i.e. toward each of the vertices around the edge -

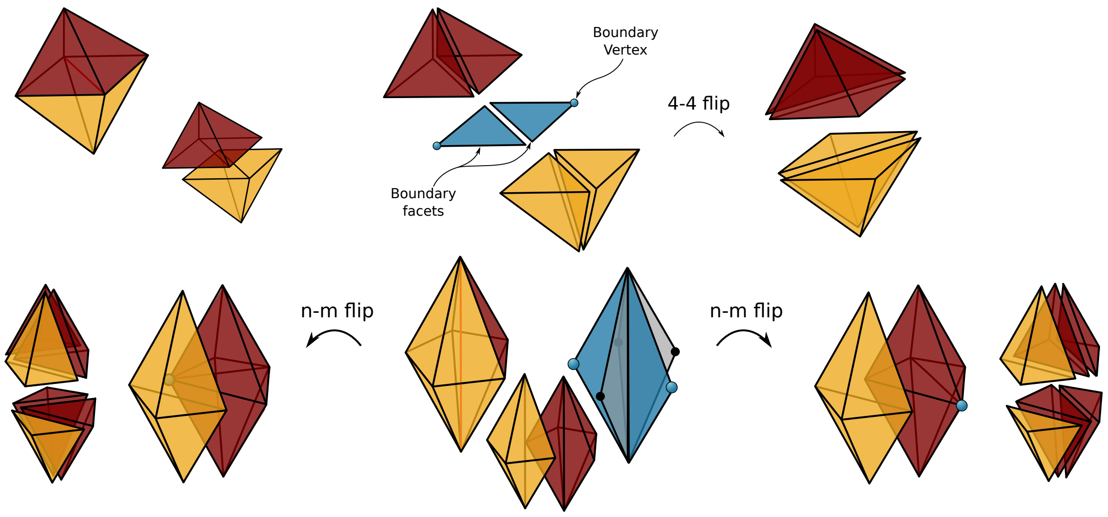


Figure 2.20: **Flip of boundary edges.** (Top) 4-4 flip (Bottom) n-m flip

and perform the one that maximizes the quality of the worst tetrahedron. For each $\hat{v}_i$, the new tetrahedra are generated using the vertex $\hat{v}_i$ and the facets opposite to v_i and v_j that are not incident to $\hat{v}_i$ (Figure 2.19, yellow and blue facets). The new edges are the ones between the vertices $\hat{v}_i$ and $\hat{v}_j$ such as $\{\hat{v}_j \in \hat{V}_1(e_{ij}) | i + 2 \leq j \leq i + n - 2\}$ (Figure 2.19, red).

Boundary edge flips. Only flips of surface edges minimizing the average boundary vertices valence's deviation from 6 for surface vertices and 4 for feature vertices are performed. Feature edges are not flipped since processing them would fail to preserve 1-junctions. These edges have exactly two incident boundary facets, hence, to ensure the preservation of the 2-junctions, the edges are flipped toward one of the two boundary vertices of these facets enforcing a boundary edge in the new configuration (Figure 2.20). The 3-2 flip of boundary edges are not allowed.

2.3.5 Filtering

Now that the connectivity have been updated to locally optimize the dihedral angle distribution, the vertices are reallocated to improve the vertex distribution. Each vertex is smoothed by averaging the subset of its one-ring vertices that meet the similarity or inclusion condition, according to Table 2.3 and, for boundary vertices, that verify the topology tests. We first smooth the feature vertices, then the surface vertices using the smoothed feature vertices and, finally, the inside vertices using the smoothed boundary vertices (Figure 2.21). This hierarchy allows us to prevent the inversion of tetrahedra.

Feature vertices. The feature vertices lie at the intersection of three or more 2-junctions. For each adjacent 2-junction, we perform a tangential Laplacian smoothing using the positions of its neighbors on the related 1-junction and the vertex normal corresponding to

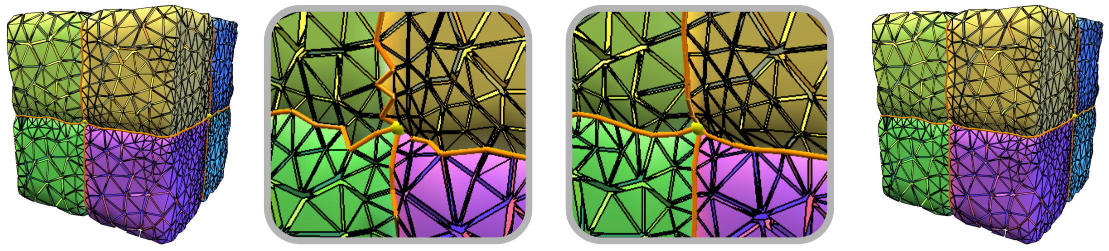


Figure 2.21: **Feature preserving smoothing.** Using the feature complex, a feature preserving hierarchical smoothing is performed.

the current 2-junction, then project the result on its MLS surface. The smoothed position is then obtained by averaging the projected points.

Surface vertices. First, we perform a tangential Laplacian smoothing by considering the surface, updated feature and corner vertices of its one ring, that verify the similarity or inclusion condition and the topology tests, and the corresponding vertex normals, which are computed using its adjacent boundary facets. The smoothed position is then projected onto the MLS surface modeling the related 2-junction.

Inside vertices. Finally, the vertices that do not belong to the feature complex are smoothed by performing a Laplacian smoothing using all its adjacent vertices, since they all verify the similarity and inclusion conditions.

2.3.6 Quality improvement

After a few iterations (typically 5-10), the aimed resolution is reached and the overall mesh quality is improved. We observed that performing a few cycles of smooth and flip operations drastically improves the quality of the output mesh.

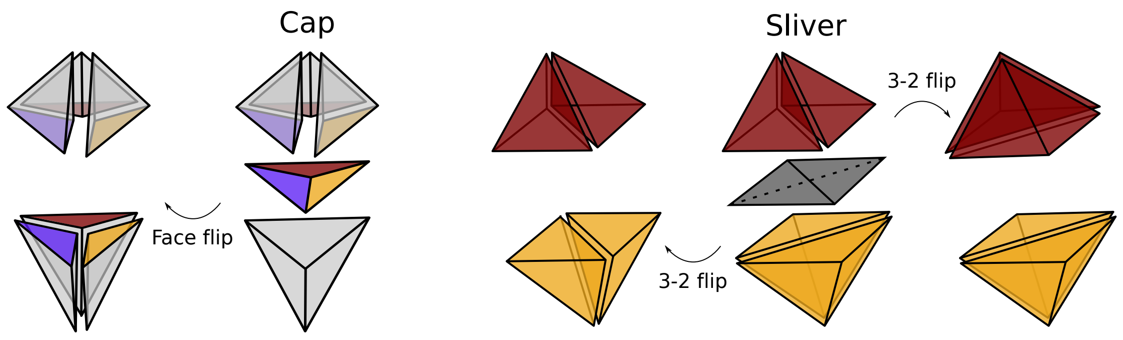


Figure 2.22: **Sliver removal.** Optimization step to remove the remaining few poorly-shaped tetrahedra: (Left) face flip, for cap tetrahedra, or (Right) edge removal for other kinds of degeneracy.

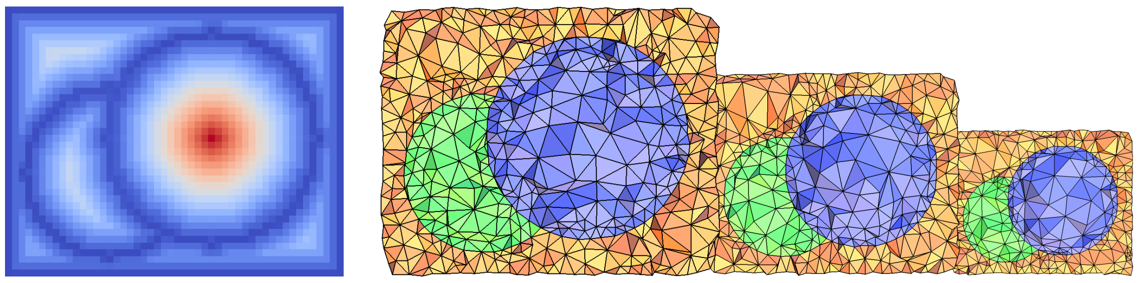


Figure 2.23: **Adaptivity.** (Left) Sizing field rising from the distance to the 2-junctions and tailoring the target edge length. (Right) Resulting adaptive meshes at three different resolutions.

A final optimization process might be necessary to remove slivers or poorly shaped tetrahedra. For all the tetrahedra that do not meet the quality criteria, we perform the local operation that improves the quality measure the most. If the current tetrahedron is a *cap*, a face flip is performed, which is the inverse of the 2-3 flip (Figure 2.22, left). In the case of a sliver, we perform an n-m flip on one of the two edges that are shared by the pairs of facets that have the largest dihedral angle, and that improves best the quality measure. Commonly, a small number of tetrahedra do not meet the quality criteria, and only a few iterations are required (Figure 2.22 right).

2.3.7 Adaptive sizing field

Our method allows us to consider spatially varying edge length targets, and in this section we present results based on sizing fields that we derive automatically from the input geometry. In order to get graded meshes of reduced size, we propose to compute the sizing field based on a distance field from the boundaries. This distance field is computed by first discretizing the space inside the triangulation offseted bounding box resulting in a voxel grid of a user defined resolution. In a second step, each voxel is assigned the smallest distance from its center to the set of boundary facets of the input model. These distances are efficiently computed using an acceleration structure detailed Section 2.4. Finally, the grid values are normalized. Given a target length for the boundary edges l_B and another one for the inside edges l_I, the objective length l_{ij} for the current edge e_{ij} is given by:

$$l_{ij} = (l_I - l_B)D(m_{ij}) + l_B \tag{2.2}$$

with m_{ij} the edge mid-point and $D : \mathbb{R}^3 \rightarrow \mathbb{R}^+$ the normalized distance field from the boundaries.

2.3.8 Additional feature preservation

The preserved 1- and 0- features stem from the multi-material junctions. Nevertheless, the input domain's sharp features are not taken into account. In the following, we present how to preserve additional features by either segmenting the imaginary tetrahedra, hence creating multi-material junctions, or by explicitly tagging features edges and corner vertices to be preserved during the remeshing process.

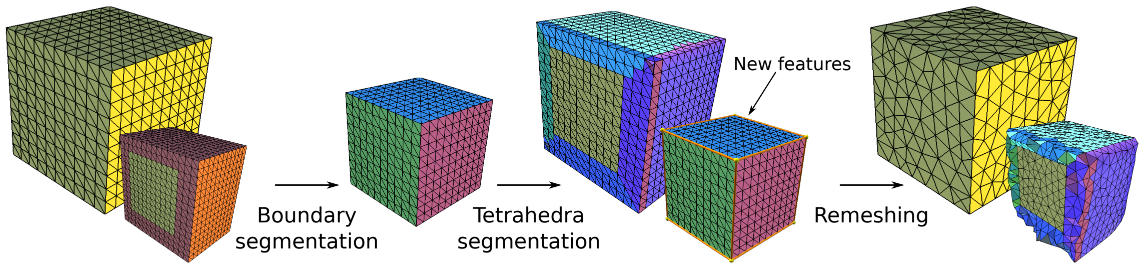


Figure 2.24: **Surface feature preservation.** (Left) Input mesh: domain tetrahedra in yellow and imaginary tetrahedra in orange, (Middle left) outer boundary segmentation using VSA, (Middle right) segmentation of the imaginary tetrahedra implicitly creating 1- and 0-junctions representing the closed sharp feature of the outer boundary, (Right) output mesh with preserved sharp features.

Closed features. We propose to detect closed features on the outer boundary of the represented domain, and to add them to the feature complex by segmenting the imaginary tetrahedra. To do so, the outer boundary facets - with one incident imaginary tetrahedron - are clustered in surface patches with similar normals delimited by closed feature lines using the Variational Shape Approximation (VSA) method [CSAD04].

Detected feature lines represent sharp creases of the input domain. In order to preserve them during the remeshing process, we add them to the feature complex by propagating the facet's segmentation to the imaginary tetrahedra. The imaginary tetrahedra are now labeled therefore implicitly creating 1-junctions where the detected close features lie (Figure 2.24). In concave regions, facets belonging to two different regions might be connected by critical edges (Figure 2.25). In that case performing a straightforward flood-filling propagating the facet segmentation to the imaginary tetrahedra induces unconnected components, conflict regions and sparse 1- and 0-junctions created on the boundaries (Figure 2.25). To overcome this problem, we split imaginary critical edges as a pre-process.

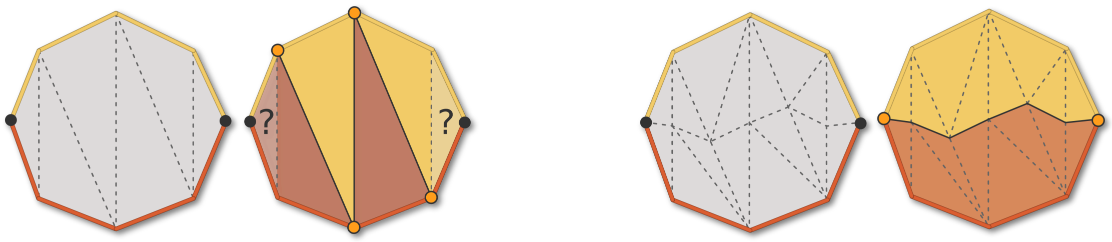


Figure 2.25: **Surface feature preservation.** 2D illustration of the problem in concave regions: (Left) imaginary tetrahedra in gray, segmented boundary facets (yellow and red lines), detected feature at the junction of segmented regions (black circles). Straightforward propagation resulting in conflict regions, disconnected components, unwanted 1-junctions (orange circles) and the target feature is not created. (Right) Imaginary critical edges split, then segmentation propagation.

Marked features. The described feature detection and preservation method is limited to closed junctions, nevertheless, not all relevant feature lines are closed, for instance, the curvature based feature lines. Hence, we propose to preserve user provided polylines. To do so, we add each polyline as a distinct 1-junction to the feature complex, their end points and intersections as corner vertices. The features are explicitly tagged and not implicitly defined by the multi-material junctions hence when splitting a feature edge, the two new edges and the inserted vertex are tagged as features. The definition of conditions (Table 2.2) still holds. Table 2.4 summarizes the heuristics specific to this case.

Condition	$\# f_i$	$\# f_j$	Is e_{ij} feature	Update v_i	Update v_j
Similarity	2	2	yes	yes	yes
Inclusion	0	>=1	no	yes	no
Inclusion	2	1 or >2	yes	yes	no
Exclusion	>0	>0	no	no	no
Exclusion	1 or >2	1 or >2	-	no	no

Table 2.4: **Heuristics.** Summary of our feature aware heuristics for tagged features. We can easily detect if the vertices will be affected during the current operation depending on their number of incident feature edges noted $\# f_i$, i.e. on their dimension in the feature complex.

The feature polylines are smoothed by the remeshing process. Therefore, if the user wants to preserve the exact input feature vertices, we set them as corner vertices.

2.4 Results and comparisons

Performances were measured on an Intel Core2 Duo (single thread) at 2.4 GHz with 8GB of main memory. We used the Computational Geometry Algorithms Library (CGAL) 3D triangulation code as an underlying mesh structure and its Axis-Aligned Bounding Box (AABB) tree implementation to efficiently compute the adaptive sizing field. This tree is built using the input mesh's boundary facets and is used as an acceleration structure to compute the distance field.

Since the conditions are evaluated using local adjacency information, we do not need to build the feature complex explicitly in contrast to [DJL12]. We only store the list of labels incident to each vertex and construct the MLS representation of the 2-junctions. Note that we tagged the elements of the user provided features.

The 2-junctions are identified by grouping the connected boundary facets delimited by 1-junctions. These surface patches are up-sampled and the MLS surfaces are created using the resulting point sets. In the shown results, we use a classical PSS definition to represent aliased boundaries coming from the six cell decomposition and an HPSS otherwise. Any other definition can be used without any change to our methodology. For instance, in the case of input meshes with limited noise and relevant features, one might prefer a representation handling sharp features [OGG].

Model	Input	Output	Timing
Spheres	50x50x50 voxels	346747 tet.	3min34
		42535 tet.	2min30
		2378 tet.	33s
	3999 tet.	699037 tet.	4min10
		17292 tet.	1min41
		2924 tet.	6s
Segmented sphere	50x50x50 voxels	58583 tet.	1min12
		7270 tet.	10s
		2725 tet.	4s
Fertility	130586 tet.	1181721 tet.	6min51
Hand	68x71x112 voxels	594983 tet.	12m30
Liver	117678 tet.	474342 tet.	9min40

Table 2.5: **Performance table.**

Our implementation is robust to poorly shaped tetrahedra with zero or negative volume. Furthermore, any type of triangulation can be remeshed by our method allowing us to process a wide range of input. We demonstrate the validity of our approach on both synthetic and acquired segmented voxel grids.

As explained in the previous section, our heuristics depend of the element's dimension in the feature complex. For instance, we start by flipping the boundary edges that improve the average valence of the boundary vertices then flip the inside edge that locally maximizes the minimal dihedral angle. For surface meshes, simply optimizing the vertices valences result in well shaped triangles and offers high-quality meshes. In our experiments, averaging the valences fails to provide similar results for tetrahedral meshes. Indeed, the flip operation is more delicate in 3D especially in a multi-material setting imposing additional constraints. We observed that using only a valence-based cost does not allow us to recover from tetrahedra configurations with poorly shaped elements. We found a compromise by first improving the valence of the boundary vertices and then locally improving the minimal dihedral angles. Note that we use the signed dihedral angles to remove inverted tetrahedra. Figure 2.26 shows the results obtained on multi-material embedded spheres using a valence-based flip cost on the first row, an angle-based cost on the second row and our solution in the last row. The target volume valence is 10 for hull vertices and 12 for all the others. Boundary vertices have multiple valences to optimize: one per 2-junction. The target valence is 6 (resp. 4) for surface (resp. feature) vertices. We measured the mean deviation of the dihedral angle from 70.5 degrees, of the boundary facets' angles from 60 degrees and the relative mean deviation from the target edge length for the first at the tenth iteration for a target length of $\frac{1}{2}l_0$ with l_0 the input average edge length. The histograms represent, from left to right, the dihedral angles distribution, the boundary facets' angles distribution and the edge lengths.

After the first iteration a high number of flat tetrahedra appear due to the MLS projection of the aliased boundary, and we can observe that the valence-based flip does not

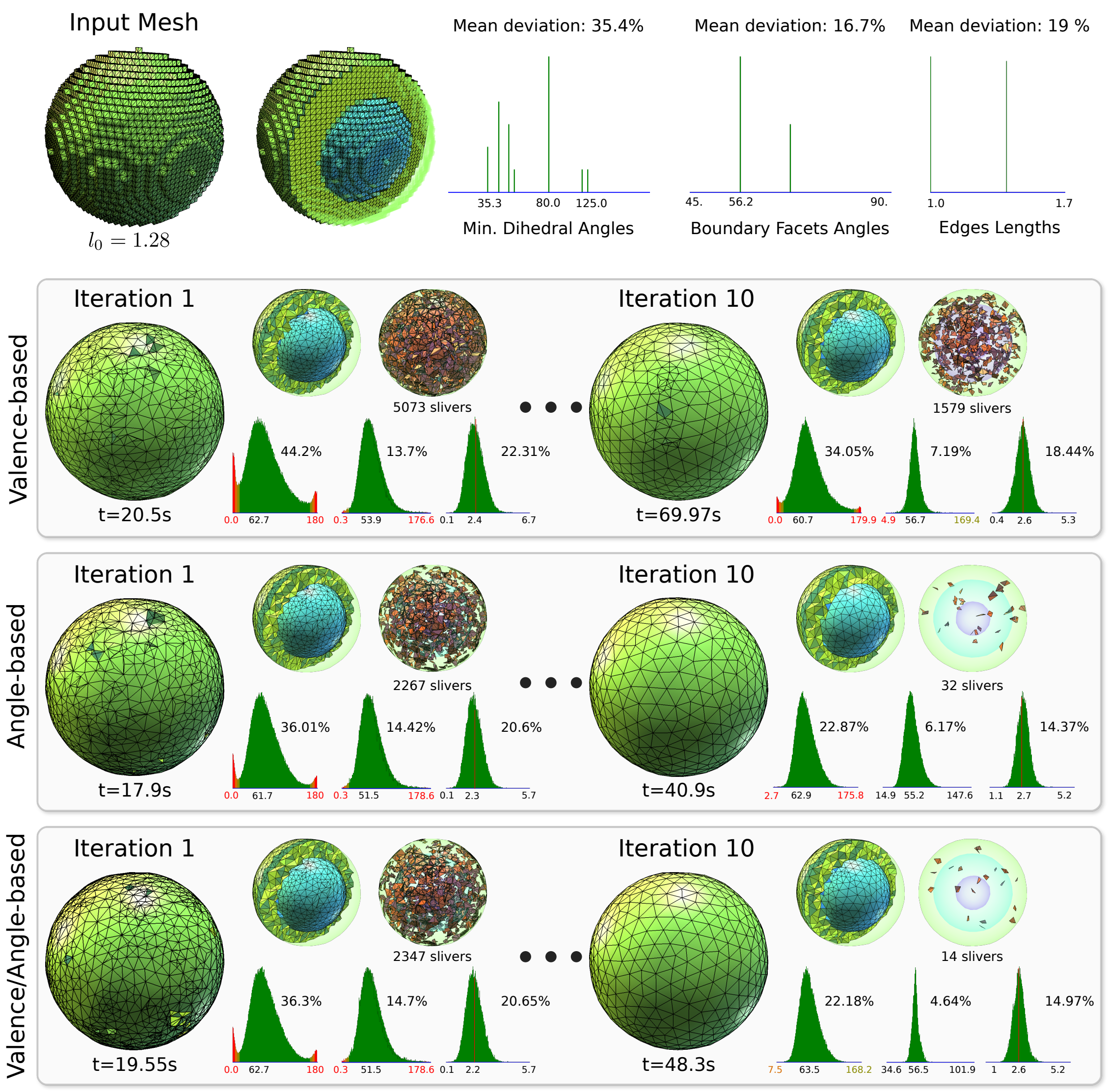


Figure 2.26: **Flip comparison.** Ten remeshing iterations of an aliased multi-material six-cell decomposition using different flip costs: (First row) valence-based flip with a target valence per 2-junction of 6 and 4 for, respectively, surface and feature vertices and a target volume valence of 10 for hull vertices and 12 for all the others, (Second row) angle-based flip - i.e. the flip that maximizes the minimal dihedral angle is performed, and (Last row) valence-based for the boundary edges and angle-based for all the others. Note that up to two configurations exist for boundary conforming flip operation and the one that minimizes the volume cost function is performed. We measured the mean deviation of the dihedral angle from 70.5 degrees (34.05%, 22.87% and 22.18%), of the boundary facets' angles from 60 degrees (7.19%, 6.17% and 4.64%) and the relative mean deviation from the target edge length (18.44%, 14.97% and 14.37%). The histograms represent, from left to right, the dihedral angles distribution, the boundary facets' angles distribution and the edge lengths. The number of displayed slivers with dihedral angles under 15 degrees are respectively 5073, 2267 and 2347 for the first iteration and 1579, 32 and 14 for the tenth.

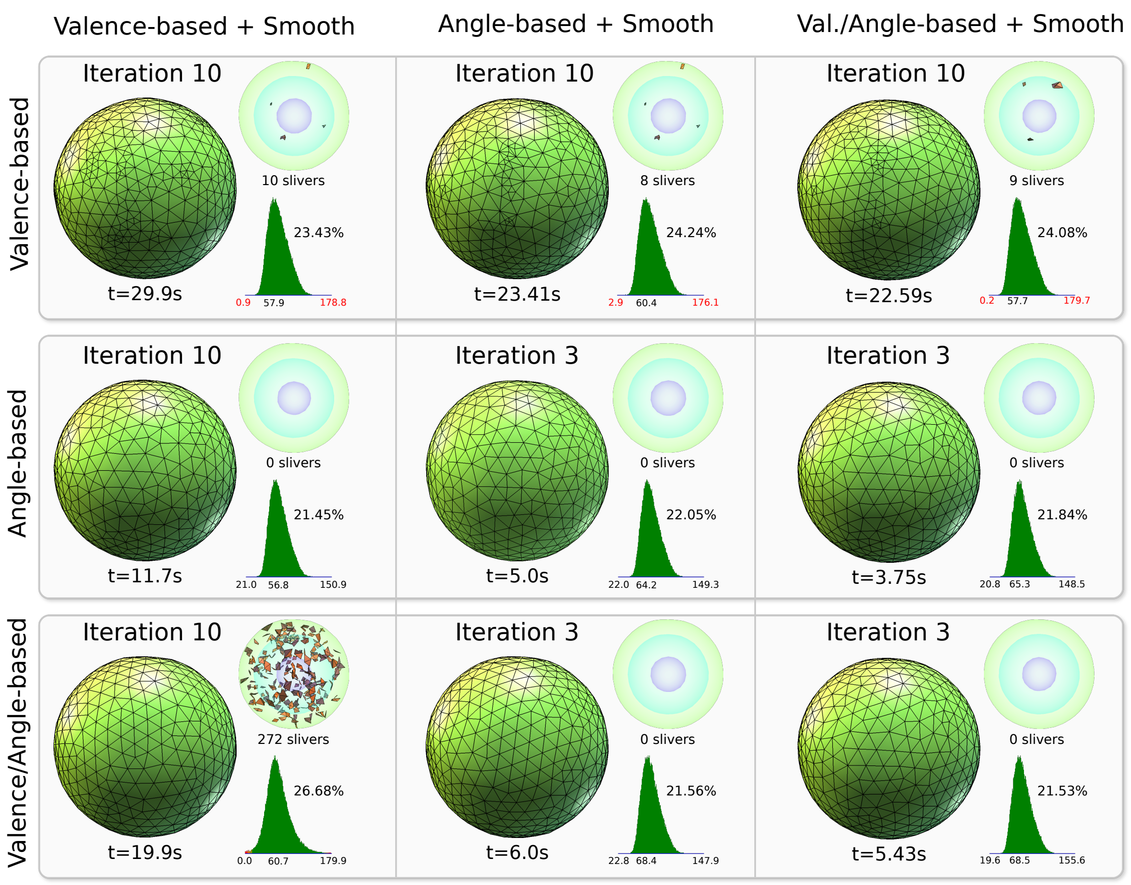


Figure 2.27: **Flip-smooth comparison.** A maximum of ten iterations of flip-smooth using valence- (resp angle- and valence/angle-) based flip for the first (resp. second and third) column. Input meshes are generated in ten iterations using: (First row) valence-based flip, (Second row) angle-based flip and (Last row) valence/angle-based flip. Distribution of dihedral angles, mean deviation from 70,5 degrees and sliver tetrahedra with dihedral angles under 15 degrees.

recover from such a configuration since thousands of slivers remain after ten iterations (Figure 2.26). Nevertheless, the angle-based flip performs well, and combining the valence and angle-based approaches yield the best results offering high-quality boundary surface patch, with angles between 34.6 and 101.9 degrees in the example Figure 2.26, and a good dihedral angle distribution with a limited number of slivers.

When the desired resolution and global high quality is reached, we have observed that performing a few cycles of flip-smooth is often enough to remove the remaining few poorly shaped tetrahedra. Figure 2.27 illustrates a comparison on a maximum of ten iterations using the three different types for this stage on the three resulting meshes of Figure 2.26. We observe that the valence-based flip does not reliably improves the mesh quality. Both angle- and valence/angle-based flips allow us to generate high-quality results.

We evaluate the validity of our approach by performing each operation independently and combine them together. Figure 2.28 shows, in the first row, the regularization of the edges

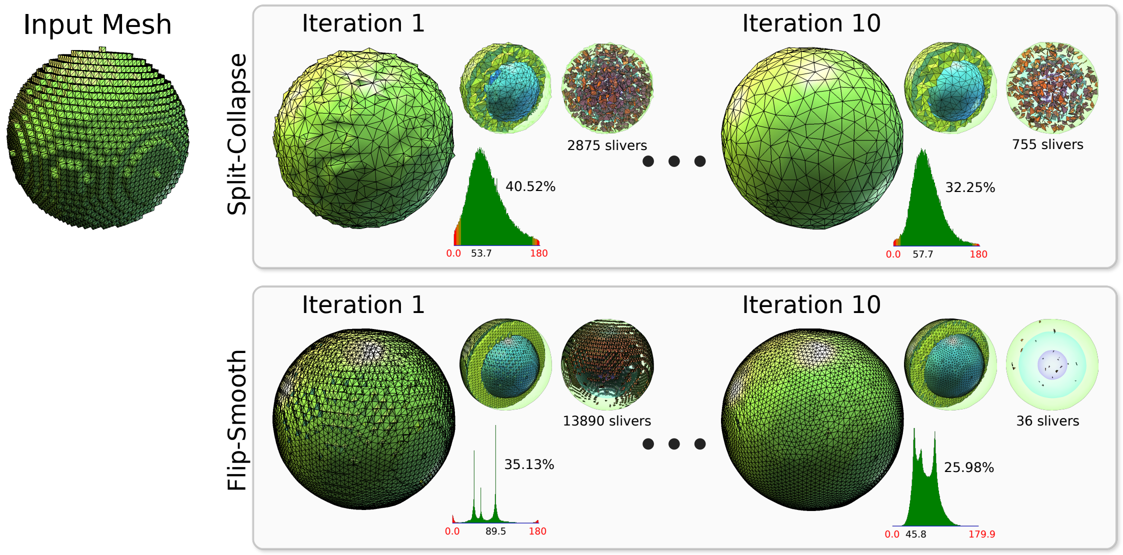


Figure 2.28: **Separate operations.** (First row) Split-Collapse operations with a target length of $2l_0$, (Second row) Flip-Smooth operations. Distribution of dihedral angles, mean deviation from 70,5 degrees and sliver tetrahedra with dihedral angles under 15 degrees. Both angle- and valence/angle-based flips allow to generate high-quality results while a valence-based approach can decrease the input mesh quality.

lengths alone and, in the second row, the flip-smooth cycles alone. We can observe that the high quality of the mesh stems from the simultaneous refinement and connectivity improvement (Last row Figure 2.26).

Figure 2.29 illustrates the successive steps of our method on a segmented sphere with 0- and 1-junctions with a target edge length of $2l_0$. The final result is obtained in five iterations and two flip-smooth cycles. The MLS representation of the surface patches allows to smooth noisy boundaries and features. Furthermore, it allows to refine as well

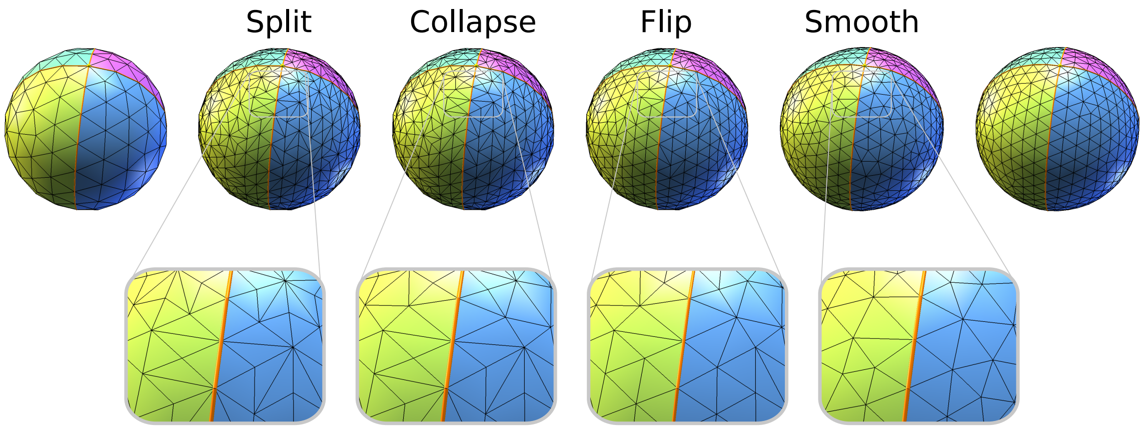


Figure 2.29: **Step-by-step.** Successive operations of our method performed on a segmented sphere with a target edge length of $0.5l_0$.

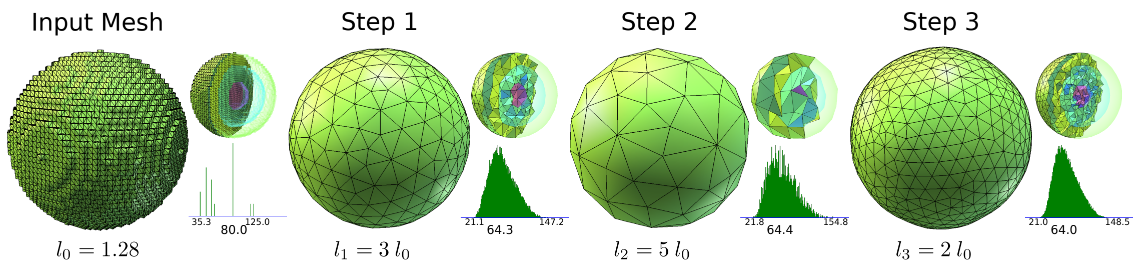


Figure 2.30: **Parameter space exploration.** High-quality meshes generated for each target length, in 5 iterations plus 2-3 flip and smooth cycles, with dihedral angles between $[21.1, 147.2]$, $[21.8, 154.8]$ and $[21.0, 148.5]$ for step 1, 2 and 3 respectively.

as simplify the input mesh while preserving the boundaries' shape since the representation is independent from the current mesh state. Figure 2.30 illustrates an interaction session where the user navigates between different resolutions. High-quality meshes are generated, within seconds, for three different target lengths, in 5 iterations plus 2-3 flip and smooth cycles. Even though no sliver removal step was performed, the resulting meshes all have dihedral above 21 degrees.

Since our approach is targeted but not limited to multi-material input domains, we evaluate our approach on single material input meshes such as the fertility model (Figure 2.31), then we compare our method with state-of-the-art single meshing techniques. In Figure 2.32, the meshes of the first row are generated from the same Sphere surface mesh using the same size and quality parameters. The first mesh is generated by a Delaunay Refinement process and the second by using DelPSC [CDR07]. The meshes of the second row are generated using the Refinement mesh as an input. We performed our remeshing process, 5 iterations and 3 flip-smooth cycles, in 20 seconds and performed an ODT [Che04, ACSYD05] optimization with the same time limit to get a fair comparison. We can see that our method does not produce slivers since the smallest dihedral angle is 23.2 degrees, whereas other methods need an additional optimization step. Note that on a similar example, NODT [TWAD09] performs better than ODT but also produces slivers. Our method prevents the apparition of degenerated tetrahedra with a large longest to shortest edge ratio, such as needles and wedges, since the edge lengths are equalized, most of the slivers and spindles are removed

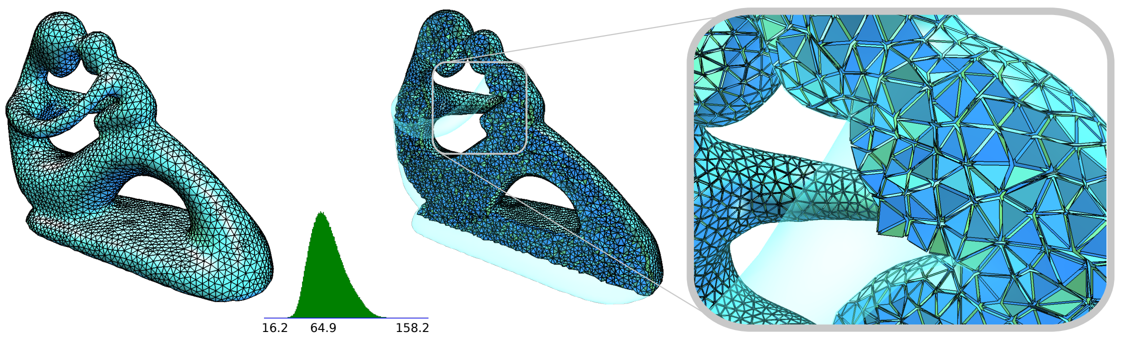


Figure 2.31: **Fertility.** High-quality isotropic remeshing.

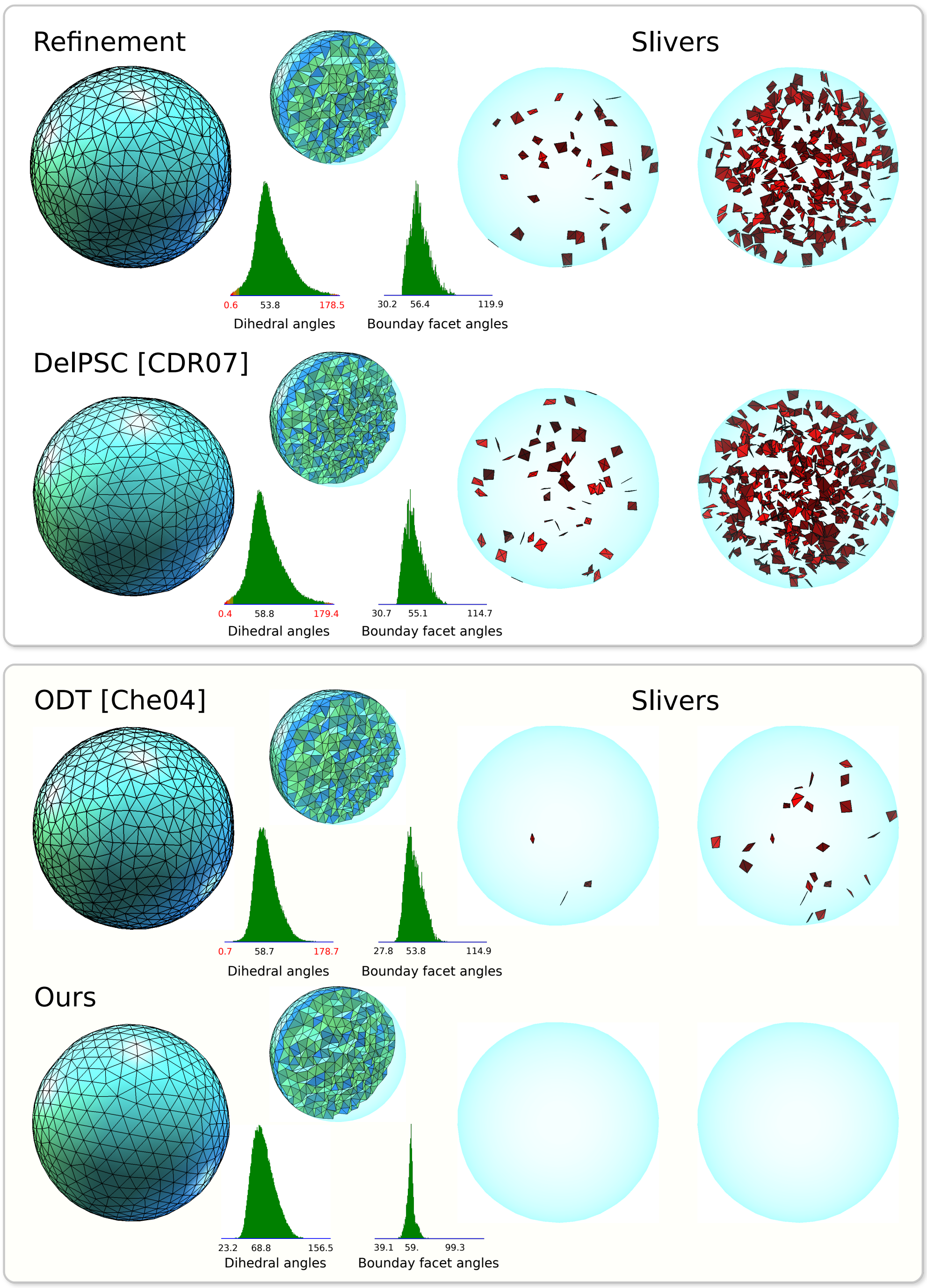


Figure 2.32: **Comparison.** (First row) Mesh generated from the same Sphere surface mesh using the same size and quality parameters. The first mesh is generated by a Delaunay Refinement process and the second by using DelPSC [CDR07]. (Second row) Meshes generated using the Refinement mesh as an input.

on account of the angle-based flip and the remaining few are taken out as a final process, along with the cap tetrahedra (Figure 2.1). Note that this final step is often unnecessary.

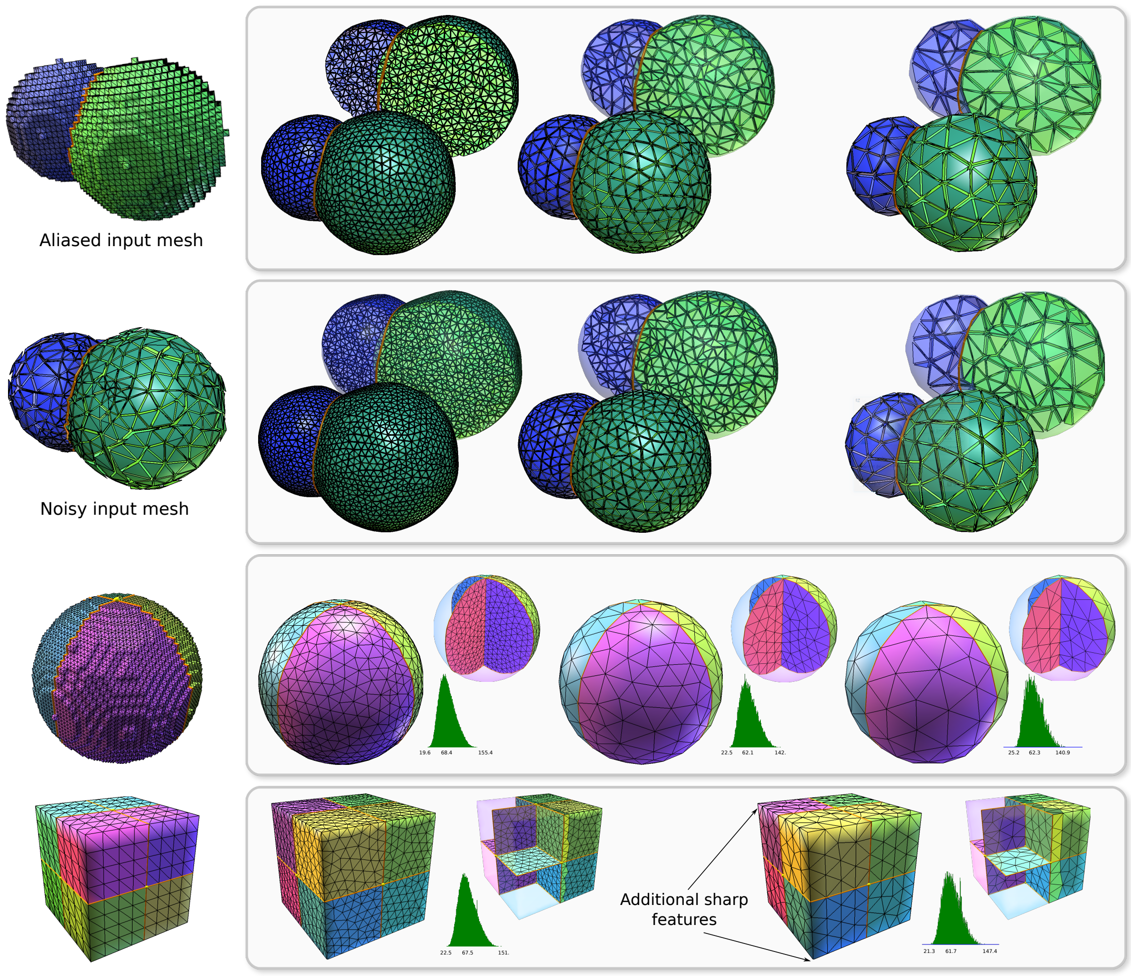


Figure 2.33: **Synthetic results.** (Re)meshing synthetic data at different resolutions using our method. The last raw illustrates the addition of closed features to the feature complex which are detected on the mesh outer boundary.

We present additional synthetic results (Figure 2.33) and apply our method on segmented medical images (Figures 2.34 and 2.35). Table 2.4 shows the performances for the (re)meshing of the presented various input data sets. Note that the performances strongly depend on the input and the edge aimed length.

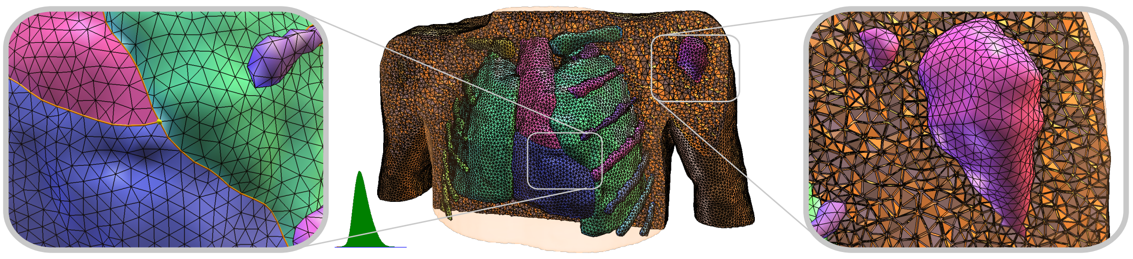


Figure 2.34: **Result.** Remeshing of a torso model with our algorithm.

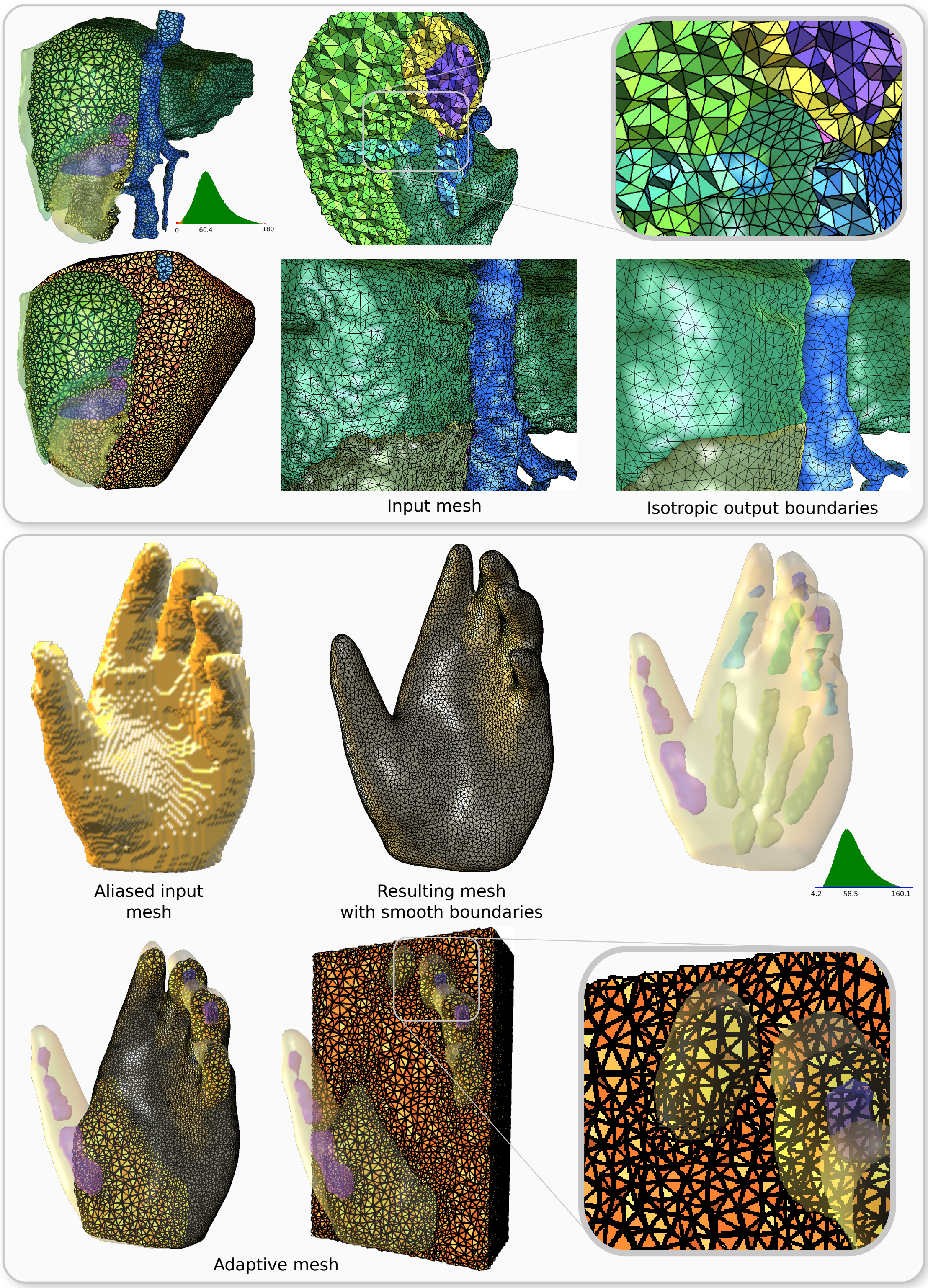

0.
60.4
180
Input mesh
Isotropic output boundaries
Aliased input mesh
Resulting mesh with smooth boundaries
4.2
58.5
160.1
Adaptive mesh

Figure 2.35: **Result.** (Re)meshing of medical models with our algorithm.

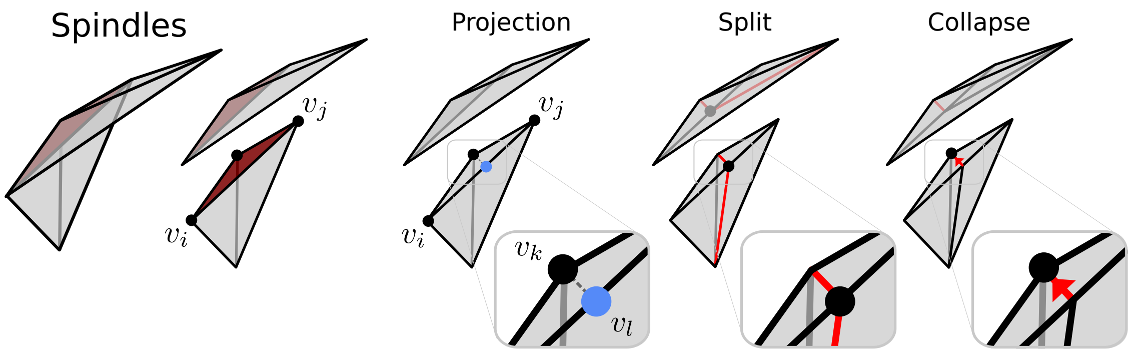


Figure 2.36: **Spindle removal.**

2.5 Possible improvement

In this chapter, we presented our M-MAd mesher, an iterative (re)meshing approach for multi-domain tetrahedral meshes based on local operations simultaneously refining or simplifying the tetrahedra while improving the quality through local topology changes and point reallocations. Additionally, our framework allows to preserve features detected on the outer boundary of the domain as well as user defined features.

Our method provides high-quality meshes at different resolutions using well known simple local topological operators and is general enough to be applied to any structured mesh. As a result, our high-quality adaptive (re)mesher can compete with state-of-the-art methods [DJL12] but is free from the computation of a Delaunay triangulation/Voronoï diagram, has lower memory consumption and is significantly easier to implement.

Our feature-aware operations strictly preserve the input topology. This property may cause problems for noisy datasets, for instance with small groups of isolated tetrahedra or non-manifold junctions. Poorly shaped tetrahedra are generated when the target edge length is significantly larger than the size of the feature to preserve.

To tackle this issue, part of these feature preserving conditions can be relaxed or the target edge length can be changed during a post-processing stage. When the target resolution is reached and the overall quality of the mesh meets the user's constraints, we can point to the user the parts of the model that could not be further processed or improved because of topology constraints, and we remesh these regions locally with adapted parameters. If the user releases the topology constraints, the operations are performed without the feature preserving heuristics - i.e. single material case - otherwise the size constraint is released and the target edge length is defined using a sizing field based on the local feature size. An alternative is to use a dynamic sizing field, adjusting the local target size during the remeshing process as proposed in [TWAD09].
Note that to avoid part of this problem when dealing with a segmented voxel grid, a pre-processing step can be performed to remove isolated voxels, small features and merge too close together corner vertices in order to get a cleaner input [DJL12].

In some cases, our method can fail to remove chains of ill-shaped tetrahedra.We observed that these configurations often remain because the collapse operations are prevented in order to avoid inversion of tetrahedra, and the flip operation fails to remove spindle. Two adjoint spindles can be eliminated simultaneously, as illustrated in Figure 2.36, using a split and a collapse operation. A vertex v_p is inserted at the projected position of v_l on e_{ij}, then the created short edge e_{kl} is collapsed toward v_k. In this case, we could remove the spindles while releasing the positive volume and topology constraints to reduce the number of poorly shaped element possibly allowing our process to recover from such complex configurations. These processes must be combined with a roll back mechanism in order to cancel the operations that did not improve the quality. More generally, groups of poorly shaped tetrahedra could be removed using a feature preserving version of the vertex insertion strategy (also combined with a roll back mechanism) proposed in a single material setting [KS07].

If an aggressive simplification is performed, the size of the neighborhood search to perform the MLS projection may become too small. To tackle this issue, we can dynamically adjust this size based on the target edge length.

We can reduce the memory footprint of our meshing process by using an adaptive version of our six-cell decomposition. Our method is well suited to improve the quality of the mesh generated with lattice based methods [BLW13].

Since we use local topological operations only, our remeshing method can be applied on a portion of the mesh using only local neighborhood information. Therefore, based on the streaming algorithm for compressing tetrahedral volume meshes proposed [ILGS06], an extension of our method to model that do not fit in main memory seems foreseeable.

Chapter 3

Volume data deformation

In the previous chapter, we presented a (re)meshing method allowing us to generate high-quality tetrahedral meshes suited for physically-based simulations. Unfortunately, these simulations are limited by the acquisition posture. While meshes can be postured using high-quality deformation methods, these methods are not applicable to voxel grids. Furthermore, they are usually not linear and require costly iterative solving processes, even on small models. In many scenarios, offline deformation computation is not an option – since the user may need to explore the space of possible shapes interactively. In this chapter, we focus on interactive deformation of voxel grids providing high-quality results suitable for physical simulation such as SAR measures on real datasets (i.e., segmented full human body images as in Figure 3.1) for various poses or as a plausible input for heavier, physics-based deformation frameworks. Such deformation methods must be scalable, interactive and easy to use for unexperienced users while preserving the model features (Section 3.1.1). Since these constraints are tedious to combine when processing directly the high-resolution model, the deformation process is often decomposed in order to act at different scales - i.e. on simpler representations (Section 3.1.2).

We propose an interactive 3-scale freeform deformation tool, targeted, but not limited, to high-resolution voxel grids, which ensures limited distortion and mass loss. We start by describing the constraints the deformation method must meet and we present an overview of existing interactive deformation methods (Section 3.1). Then, we describe the developed VoxMorph method in Section 3.2, its implementation in Section 3.3 and demonstrate its validity on several whole-body voxel grids.

3.1 Background

In this section, we start by presenting the constraints our deformation system must meet. Then, we give a quick overview of the existing methods and we detail the ones that are instrumental to ours.

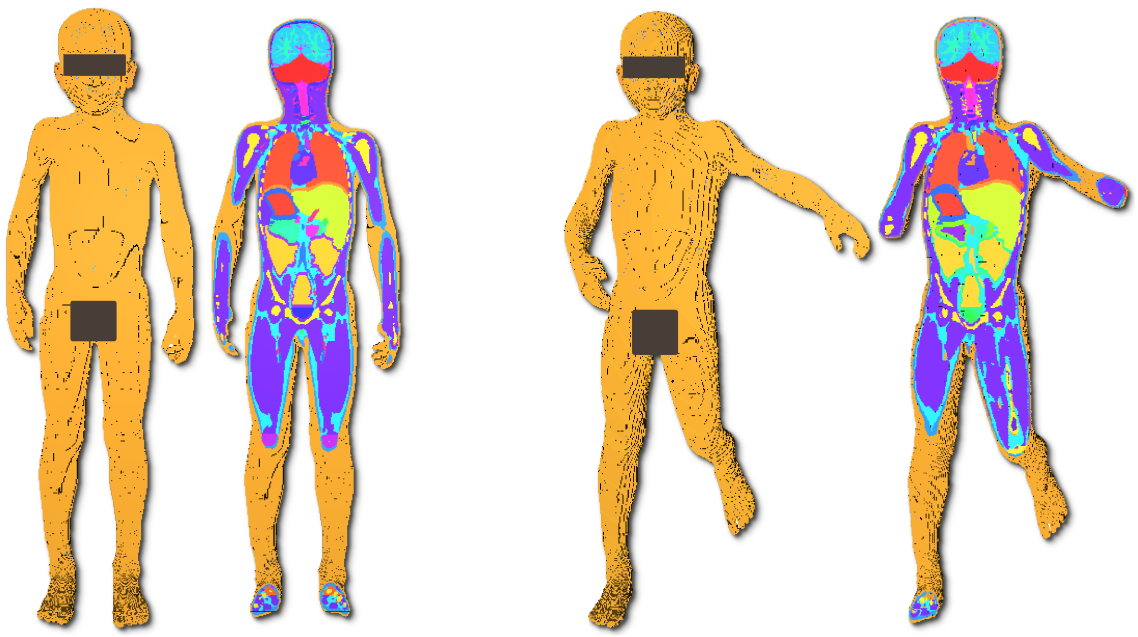


Figure 3.1: Example of a segmented high-resolution voxel grid deformation using our system.

3.1.1 Requirements

SAR measures are performed on high-resolution voxel grids representing whole-body postured models. Interactive deformation generating physically plausible posture and models suitable for simulations, imposes to simultaneously meet speed and scalability while offering intuitive user control.

Feature preservation

The deformation must be of high-quality in order to preserve the model features such as the local curvature, the subdomains' aspect and connectivity. Since physically-based deformations lack generality, we propose to perform physically-plausible deformations using geometrical assumptions only. These deformations mimic physical behavior but are not as computationally involved and offer more freedom during the modeling process. A conformal deformation preserves the input model's angles by inducing the same stretch in all directions, hence preserving features. Such guarantees are difficult to meet for volumetric methods, since they are limited to Möbius transformations, as stated by Liouville [Lio50]. In light of that, we settle for quasi-conformal deformations offering a bound on the induced anisotropic stretch. Additionally, the deformation must vary smoothly on the model since quick variations generate discontinuities.

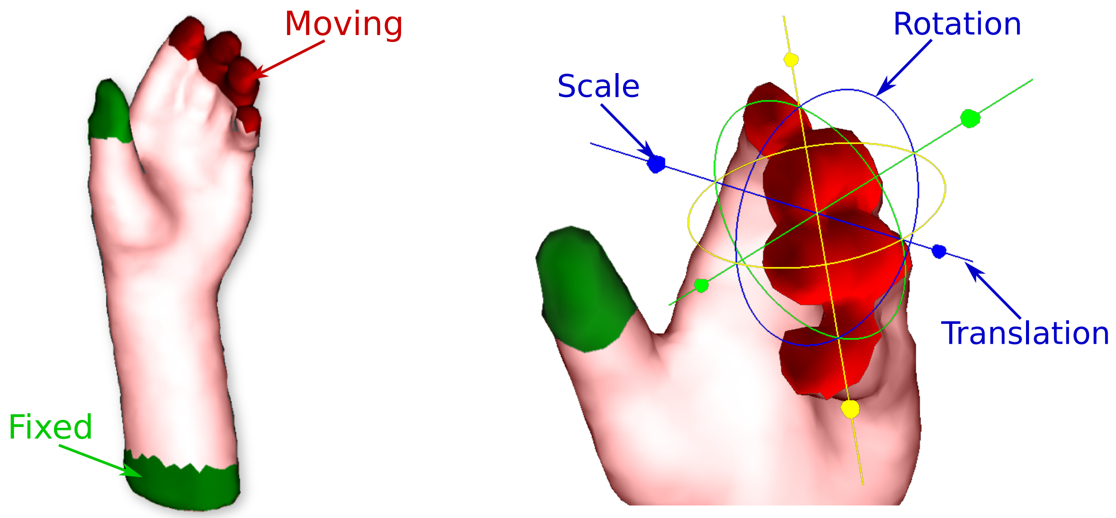


Figure 3.2: **Handle-based deformation:** the user performs deformations through positional constraints on a subset of vertices. The fixed constraints are represented in green and the moving ones, called handles in red. These handles undergo rigid deformations (rotation, scale or translation) and the free part of the model is updated accordingly through a minimization of some stretching/bending energies.

Scalability and interactivity

The model itself has a large memory footprint. Indeed, high-resolution whole-body voxel grids are typically made of several hundreds of millions of voxels, and some models may not even fit in memory. Therefore, the deformation method must have a small memory overhead.
To generate a large number of postures efficiently, for instance for epidemiology studies, the deformation needs to be interactive. Furthermore, to reach the desired posture the user needs an interactive feedback during the process.

User-friendly

Since our application is targeted to unexperienced users, the interaction should remain as intuitive intuitive as possible. Therefore, the user must be presented only with a few parameters to set and the interaction primitives must be simple.

Due to the wide range of applications, the deformation of surface models have been an active field of research. Nevertheless, only a limited number of the proposed methods allow deforming volumetric datasets. In the following, we propose a quick overview of the existing interactive methods.

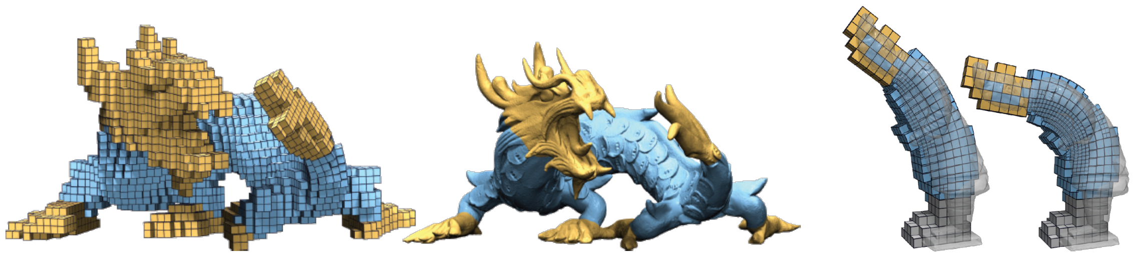


Figure 3.3: The models are deformed by optimizing the positions of the set of cells to meet the user's constraints indicated in yellow and gray. The cells are refined where the deformation error is high, as shown on the right. Images taken from Botsch et al. [BPWG07].

3.1.2 Interactive shape deformation

Interactive freeform deformation (FFD) methods can be classified into two main categories: surface deformation and space deformation.

Surface deformation

Linear variational surface deformation methods offer a flexible handle-based FFD framework. The user manipulates a typically sparse set of positional and/or rotational constraints (called handles) and the surface is updated accordingly. Figure 3.2 illustrates fixed and moving constraints and displays the manipulator used in our system.

The linear surface editing methods proposed over the last few years, use different representations in order to provide interactive deformation while coping with linearization artifacts. For instance, Yu et al. extended a gradient-based approach from image to surface editing [YZX$^+$04]. The deformation can be performed using a Laplacian-based representation of the surface, by encoding each vertex relative to its neighborhood, combined with an implicit transformation optimization [SCOL$^+$04]. Alternatively, Lipman et al. proposed a differential deformation approach based on a local frame-based representation of the mesh for which the local transformations for each frame are optimized to preserve the frame adjacency information [LSLCO05]. Although they are efficient for simple deformations, linear methods cannot cope with large shape modifications [BS08], and introduce many visible artifacts.

Non-linear variational approaches have been introduced to offer better detail and volume preservation at the cost of scalability and, often, implementation easiness. For instance, the PriMo system [BPGK06], taking its inspiration from thin shells and plates, offers physically-plausible deformations. The input high-resolution surface is embedded into prisms, and the rigid deformation is guided by the optimization of local and global shape matching energies. This method is robust, allows for physical surface behavior using intuitive parameters, and can perform large complex deformations. Nevertheless, it is complex to implement. The authors proposed an extension to rigid cells [BPWG07], where the mesh is embedded into a volumetric structure made of hexahedral cells that are refined in regions of high-deformation error (Figure 3.3). The As-Rigid-As-Possible (ARAP) surface

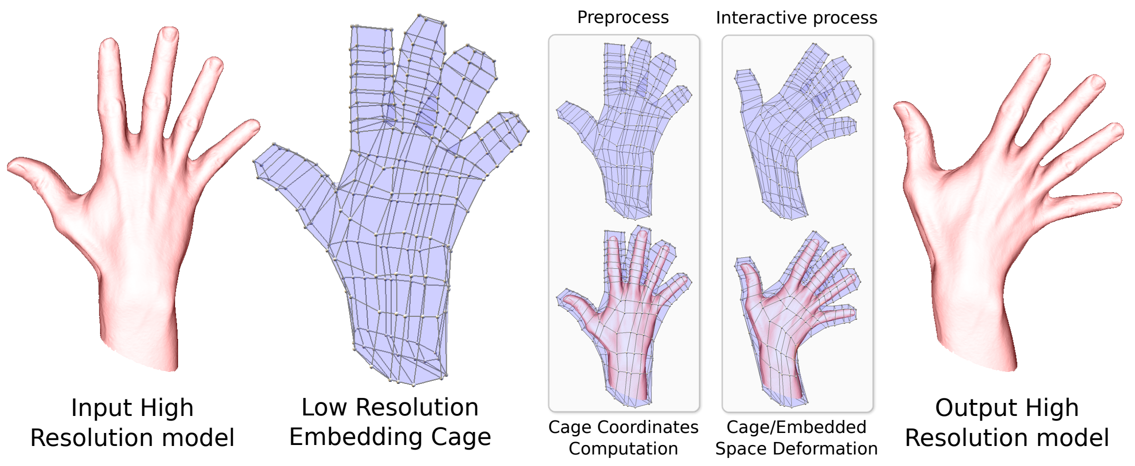

Preprocess
Interactive process
Input High Resolution model
Low Resolution Embedding Cage
Cage Coordinates Computation
Cage/Embedded Space Deformation
Output High Resolution model

Figure 3.4: **Cage example:** the input high-resolution model is embedded into a low vertex count mesh called a cage. As a pre-processing, the input mesh vertices positions are expressed with reference to the cage, and when the cage vertices positions are changed, the high-resolution mesh is updated interactively.

deformation technique, detailed in Section 3.1.3, provides high-quality deformations but is not scalable [SA07].

Space deformation

The idea of defining the deformation of an object by the transformation of its embedding space has been introduced by Sederberg et al. [SP86]. Without any constraint on the actual model representation (e.g., mesh, point set), such techniques suppose the preliminary creation of a coarse, closed polygonal surface mesh – called *cage* – surrounding the input object. Initially, all the points of the model are expressed as a weighted combination of the cage vertices. These weights are called *coordinates* with reference to the cage. At runtime, when a user moves the cage vertices, all the points are updated by multiplying their cage coordinates by the current cage vertex positions (Figure 3.4). Good coordinate

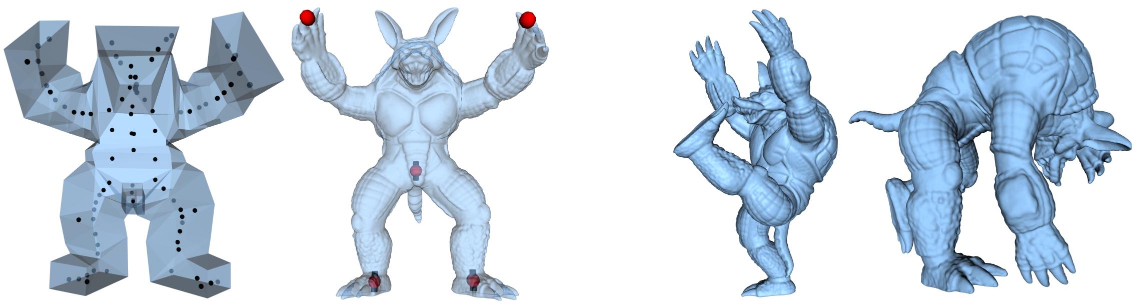


Figure 3.5: Deformations of the Armadillo model using variational harmonic maps [BCWG09]. (Left) Cage and anchor locations along with the original pose and constraints. (Right) Deformed poses.

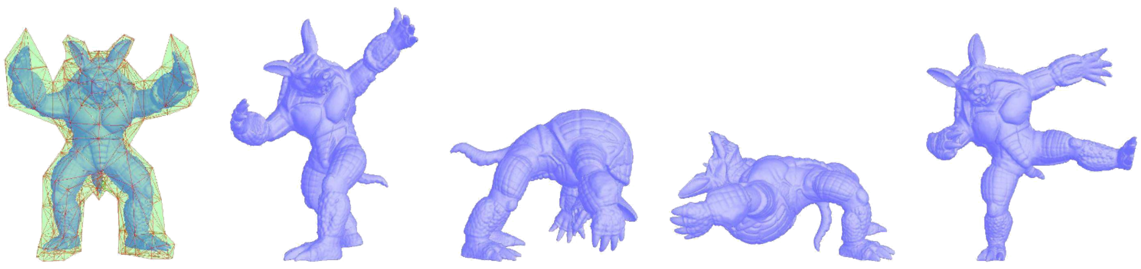


Figure 3.6: Deformation results of Armadillo [ZLXP09]. The leftmost is the original Armadillo and its tetrahedral control mesh, the others are results with different poses.

systems ensure at least smoothness and thus allow users to control a complex shape by only editing its cage. Among the popular cage coordinate systems, we can cite the *Mean Value Coordinates* (MVC) [JSW05] which smoothly interpolate vertex data (e.g., positions) using arbitrary, closed manifold cages, but may not preserve the model features; the *Harmonic Coordinates* [JMD$^+$07] which are defined as the solution of a Laplace equation with boundary conditions on the cage itself; the *Green Coordinates* (GC) [LLCO08] which offer shape-preservation by considering both cage vertex positions and cage face normals. Note also that cage coordinates can be combined with a skeleton to control the deformation using a sparse set of constraints [BCWG09] (Figure 3.5).

Other methods use a tetrahedral mesh as a deformation control structure that deforms its embedded space accordingly. Song et al. propose to do so using modified barycentric interpolation [HCLB09]. However, using simple barycentric coordinates to deform the tetrahedra inner space may generate discontinuities when the deformation is not rigid and the control mesh is too coarse. In order to obtain a high-quality deformation, a non-linear method can be applied on the control mesh [SL07, ZLXP09] (Figure 3.6). These methods are not scalable and they are used on very coarse tetrahedral meshes only in practice.

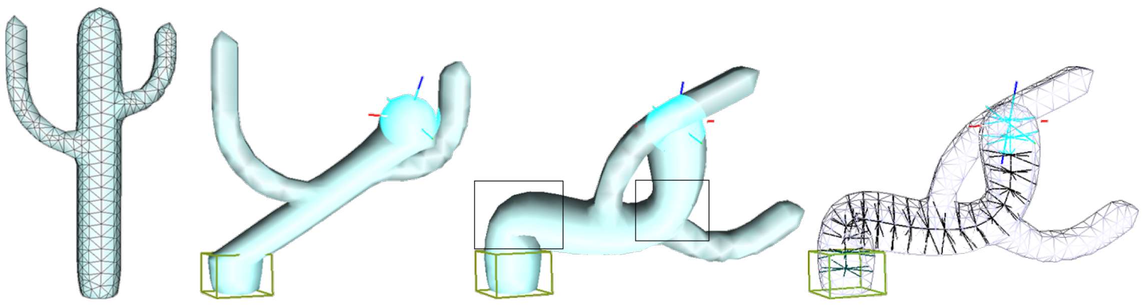


Figure 3.7: Dancing cactus using skeleton information with ARAP method (100 iterations), the skeleton edges are represented on the rightmost image. The deformations are achieved by anchoring the bottom and translating the top [ZNM10].

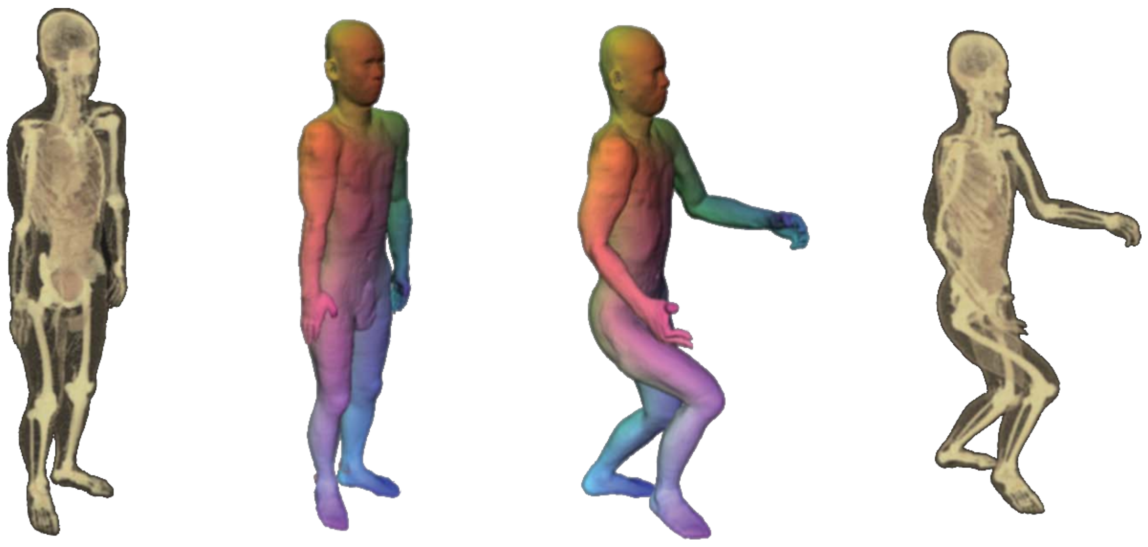


Figure 3.8: Whole-body voxel grid deformation [NW09]. (Left) Input 3D medical image, (Center) surface representation postured using a skeleton-based deformation and (Right) resulting postured 3D image. We can see deformation artifacts on the hand and the upper part of the leg.

Combined deformation

Recent works have tackled the problems stemming from both categories by combining surface and space deformation. For instance, Borosan et al. [BHZN10] proposed to perform surface deformation by using ARAP on a subsampled model before transferring the resulting deformation to the full resolution one using MVC, taking advantage of the fact that MVC are defined outside the cage as well. Similarly, Zhang et al. [ZNM10] proposed to combine an ARAP deformation and a skeleton to better preserve the volume (Figure 3.7). Unfortunately, such solutions do not scale above a few tens of thousands of samples whereas the typical size of voxel grids usually exceeds hundreds of millions of samples.

Voxel grid deformation

Looking at the particular case of voxel grids, most existing techniques make use of the original space deformation technique by Sederberg et al. [SP86] to achieve plausible poses of 3D segmented medical images in order to perform SAR analysis for instance. Nagaoka et al. [NW08] represent the segmented voxel grid as a hexahedral mesh that is rasterized after the cage-based deformation. Skeleton-based volume deformation methods have also been used in the context of 3D medical images [NW09]: a mesh representing the outer boundary of the voxel grid is deformed using a manually defined skeleton and the final 3D image is then computed using a mapping and a volume filling algorithm (Figure 3.8). An alternative method based on a dummy model deformation was proposed by Gao et al. [GMMW11]. In both cases, an interface allowing the user to specify the articulation angles is used. Unfortunately, both methods involve a tedious model-specific manual pre-processing to get plausible deformations. In fact, most medical image deformations used in today's simulations are still performed by cutting and pasting limbs and organs with a tedious manual adjustment at the junctions [NZZ+10].

So far, no deformation method allows to meet the scalability, interactivity and feature preservation constraints directly.

3.1.3 Instrumental methods

In the following, we focus on two recent techniques which are instrumental to our approach.

As-Rigid-As Possible FFD

ARAP systems are defined on a surface through an iterative process. In the first step, rotation matrices are estimated for each vertex according to the geometry of its 1-neighborhood. In the second step, new vertex positions are found according to the rotation matrices defined on the vertices and the vertex positional constraints. This process is iterated until convergence.

This deformation technique is based on the principle that mesh deformations must be locally as-rigid-as possible and smooth. For each vertex i of the cage P a cell C_i is defined, covering its incident faces. To find the as-rigid-as possible deformation that transforms this cell in C'_i, of the deformed mesh P', the rotation R_i minimizing the following energy E is computed:

$$E(C_i, C'_i) = \sum_{j \in \mathcal{N}(i)} w_{ij} \left\| (v'_i - v'_j) - R_i(v_i - v_j) \right\|^2 \tag{3.1}$$

where $\mathcal{N}(i)$ is the 1-neighborhood of i and w_{ij} the edge cotangent weight. The system of equations to solve:

$$\sum_{j \in \mathcal{N}(i)} w_{ij}(v'_i - v'_j) = \sum_{j \in \mathcal{N}(i)} \frac{w_{ij}}{2}(R_i + R_j)(v_i - v_j) \tag{3.2}$$

can be expressed using matrices $\mathbf{L}\mathbf{v}' = \mathbf{b}$ with $\mathbf{L}$ corresponding to the Laplace-Beltrami operator. The user defined constraints are added to the system by fixing the positions $v'_j = c_k, k \in \mathcal{F}$ with $\mathcal{F}$ the set of indices of the constrained vertices. This equation is pre-factorized and the edges weights pre-computed. Therefore, given an initial guess for the v'_i positions, the rotation matrices R_i are computed. The positions are then defined using the resulting rotation matrices. This process is repeated until the shape approaches the desired one.

Quasi-conformal FFD

The following notation are taken from the original GC paper [LLCO08]. The majority of the cage coordinates express a point η inside the cage P as an affine sum of the cage vertices noted $\mathbb{V} = \{v_i\}_{i \in I_{\mathbb{V}}} \subset \mathbb{R}^3$:

$$\eta = F(\eta; P) = \sum_{i \in I_{\mathbb{V}}} \varphi_i(\eta) v_i \tag{3.3}$$

where $\varphi_i(.)$ are the cage coordinates. The deformation induced by the deformed cage P' is defined by:

$$\eta \mapsto F(\eta; P') = \sum_{i \in I_{\mathbb{V}}} \varphi_i(\eta) v'_i \tag{3.4}$$

with $\mathbb{V}' = \{v'_i\}_{i \in I_{\mathbb{V}}}$ the vertices of the deformed cage. This expression does not allow to preserve the details of the model because the transformation can induce shear and anisotropic scaling.

GC express the position of a 3D point η inside a triangular mesh P as a linear combination of its vertex positions and its triangle normals:

$$\eta = F(\eta; P) = \sum_{i \in I_{\mathbb{V}}} \varphi_i(\eta) v_i + \sum_{j \in I_{\mathbb{T}}} \psi_j(\eta) n(t_j)$$

with $\mathbb{V} = \{v_i\}_{i \in I_{\mathbb{V}}} \subset \mathbb{R}^3$, the cage vertices, and $\mathbb{T} = \{t_j\}_{j \in I_{\mathbb{T}}}$, its triangular faces and $n(t_j)$ the outward face normal of t_j. $\varphi_i(.)$ and $\psi_i(.)$ are the cage coordinates. The deformation induced by the deformed cage P' – with $\mathbb{V}' = \{v'_i\}_{i \in I_{\mathbb{V}}}$ its vertices and $\mathbb{T}' = \{t'_j\}_{j \in I_{\mathbb{T}'}}$ its faces – is defined as:

$$\eta \rightarrow F(\eta; P') = \sum_{i \in I_{\mathbb{V}}} \varphi_i(\eta) v'_i + \sum_{j \in I_{\mathbb{T}}} \psi_j(\eta) s_j n(t'_j)$$

with $\{s_j\}_{j \in I_{\mathbb{T}}}$ the *stretch factor* of t_j insuring scale invariance.

In the following, we present our interactive deformation system, called *VoxMorph* [Far12], that allows us to perform quasi-conformal deformations of high-resolution voxel grids with a low computation time. We introduce a deformation adaptive linearization scheme, that reduces the full processing time by several orders of magnitude, allowing for the typical try-and-test loop unavoidable in any modeling session. Furthermore, our system behaves intuitively by preserving local geometric features and global shape structures, while providing a real time feedback to control the deformation.

3.2 VoxMorph

From the user's point of view, VoxMorph is a *cage-based* freeform deformation system, exploiting a simple (coarse) cage embedding the voxel model. This cage acts as the unique deformation control interface and is classically represented as a closed polygonal surface mesh. The shape can be edited by moving selected handles on the cage and freezing others.

From a technical point of view, VoxMorph is a **3-scale** deformation algorithm using suitable deformation methods at different scales. At coarse scale, a non-linear variational method allows to control the cage's shape using a few vertex constraints only while solving for all others in an *as-rigid-as possible* fashion. Such a behavior is known to be the most intuitive in interactive shape deformation [ACOL00, IMH05]. As the cage vertex count is typically low (e.g., a few hundreds), real time performances are preserved. At medium scale, a linear quasi-conformal space deformation method is used to transfer the cage modification to its inner space while locally bounding the variation of angles and distances. Here, the deformation can be evaluated smoothly on several tens of thousands

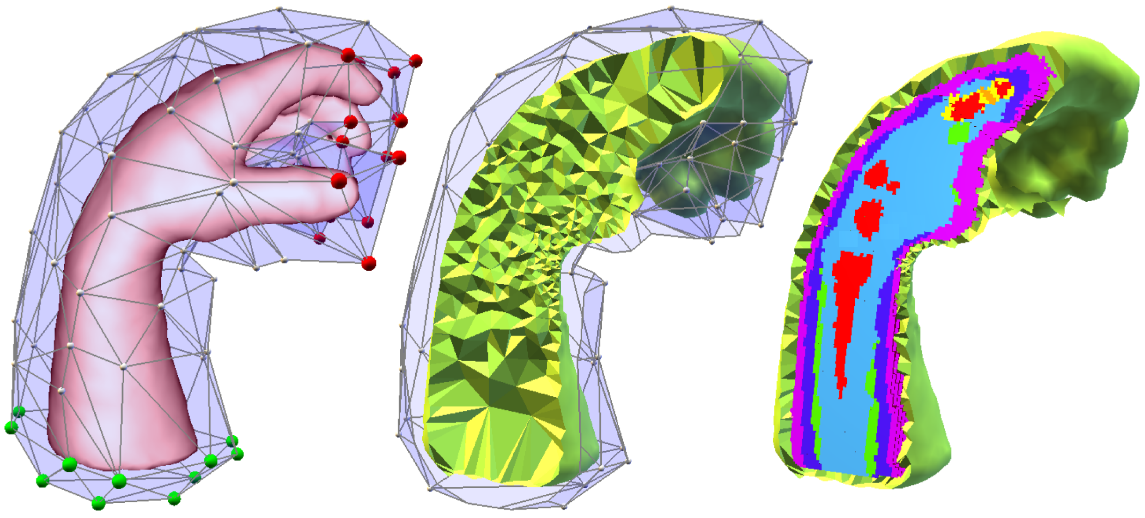


Figure 3.9: **A 3-scale deformation pipeline:** the input voxel grid deformation process is controlled using a coarse cage with non-linear ARAP deformation (left). When reaching a satisfying shape, the deformation is first transferred to a deformation-adaptive mid-resolution volume mesh using GC (middle). The full resolution voxel grid is finally rasterized in its deformed space using the so-defined per-simplex linear deformation (right).

of points interactively. At fine scale, all input voxels need to be deformed but computing space deformation coordinates for each single voxel is prohibitive. Therefore we propose a new motion-adaptive linear approximation scheme, by the means of a tetrahedral *transfer mesh* deformed at medium scale, and defining a linear (barycentric) deformation on a per-tetrahedron basis. This adaptive approximation provides a fast rasterization process for the input full resolution model in the transformed space defined by the cage deformation (Figure 3.9). Although we illustrate this chapter using regular voxel grids, our method is cage-based and therefore compatible with any discrete volumetric data (e.g., adaptive, point-sampled and even meshes).

3.2.1 Overview

VoxMorph is a freeform deformation system for voxel grids based on a 3-scale deformation procedure:

1. a non-linear variational deformation [SA07], operated by the user on a coarse cage embedding the model;

2. a linear global space deformation transferring the shape defined at lower scale in a quasi-conformal way [LLCO08] to a mid-resolution visualization surface for interactive rendering;

3. a new local linear deformation technique transferring the shape modification from a mid-scale tetrahedral deformation-adaptive mesh, undergoing the two previous steps, to the full resolution voxel grid in an efficient 3D rasterization post-process.

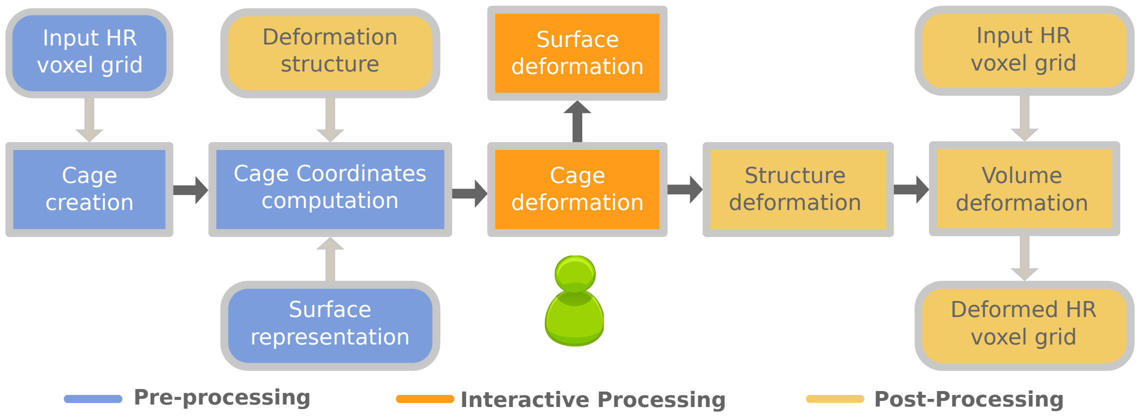


Figure 3.10: **Pipeline decomposition:** the cage/envelope creation process and cage coordinates computation are performed as a pre-processing (blue), the user interaction with the cage and the envelope is performed interactively (orange), and the deformation of the transfer mesh and the HR voxel grid as a post-processing (yellow).

In order to preserve the interactivity of the deformation, we generate the different representations and compute the cage coordinates as a pre-processing step and perform the full resolution deformation as a post-processing step - i. e. when the desired posture is reached (Figure 3.10). The resulting voxel grid contains a deformed model that can be used for simulation, visualization and virtual reality.

3.2.2 Pre-processing

Our system runs on a 3D voxel grid $\mathbb{G}$ (e.g., medical image), which defines a function $I : \mathbb{N}^3 \rightarrow E$ with E being defined on a discrete (e.g., post-segmentation) scale. By convention, null values represent the background.

In order to provide the user with a high-resolution visual feedback during interactive deformation, we start by extracting the boundary of the foreground volume (e.g., corresponding for instance to the skin of a full body medical dataset). To do so, we generate a binary grid $\mathbb{B}$ from $\mathbb{G}$ with $\mathbb{B}[g] = 0$ for null voxels and $\mathbb{B}[g] = 1$ otherwise. We extract a smooth mesh approximation of the interface between the two labels using a restricted Delaunay triangulation (RDT) with adaptive refinement as proposed by Boissonnat et al. [BO05b]. We call the resulting visualization surface *envelope* (see Figure 3.11) and will update it at each frame during interaction by applying the first two scales of our approach. Note that this envelope has typically 3 orders of magnitude less samples than the input high-resolution voxel grid. Although the cage may be designed manually, in our system we propose an optional automatic generation process (see Figure 3.12). We reuse $\mathbb{B}$ and apply a supervised dilation to it to prevent artifacts occurring when the cage is too close to the data. The interface between the resulting domains and the background is meshed again using a RDT before being simplified to a prescribed resolution (typically few tens or hundreds of vertices) using QEM simplification [GH98].

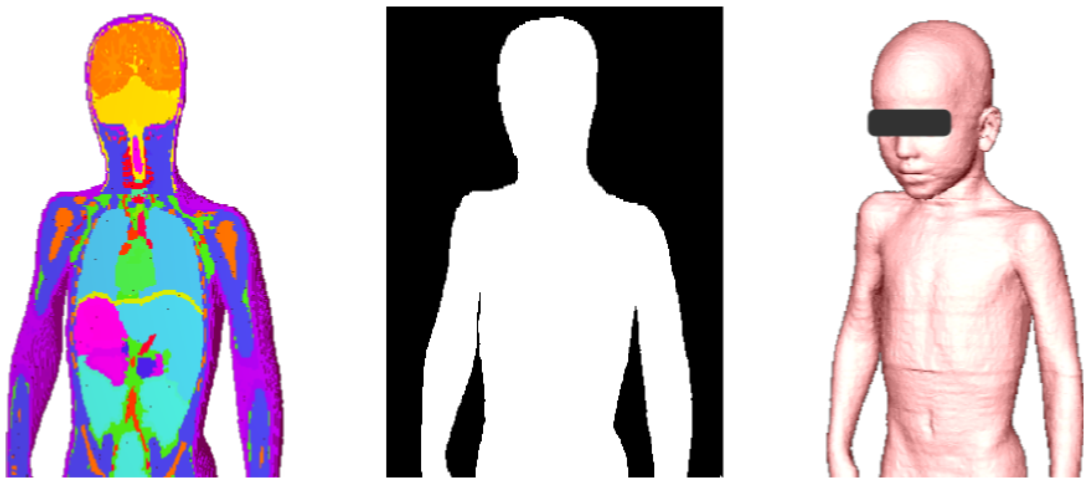


Figure 3.11: **Visualization envelope:** Starting from the input voxel grid (left) a binary grid captures the foreground/background interface (middle) from which a surface mesh is sampled for interactive visual feedback (right).

Lastly, we compute the GC encoding (see Section 3.1.3) of the envelope relative to the cage. At each frame during the modeling session, we transmit the cage deformation to the envelope in a quasi-conformal way using GC deformation. Here, the intermediate envelope resolution complies with GC complexity to preserve interactivity at this stage. One interesting property of GC, compared to other coordinate systems (e.g., MVC), is their good space separation property: nearby components can undergo very different deformations as long as cage faces can be located between them (see Figure 3.14 left).

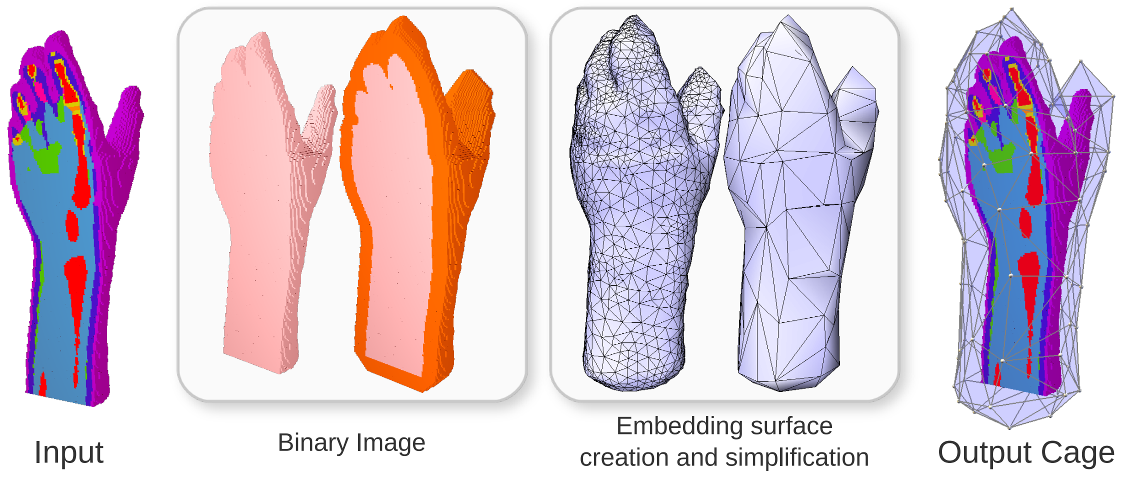


Figure 3.12: **Deformation cage:** a dilated version of the foreground/background binary grid (middle left) is meshed at its boundary; the resulting surface is simplified (middle right) and used as the interaction cage (right).

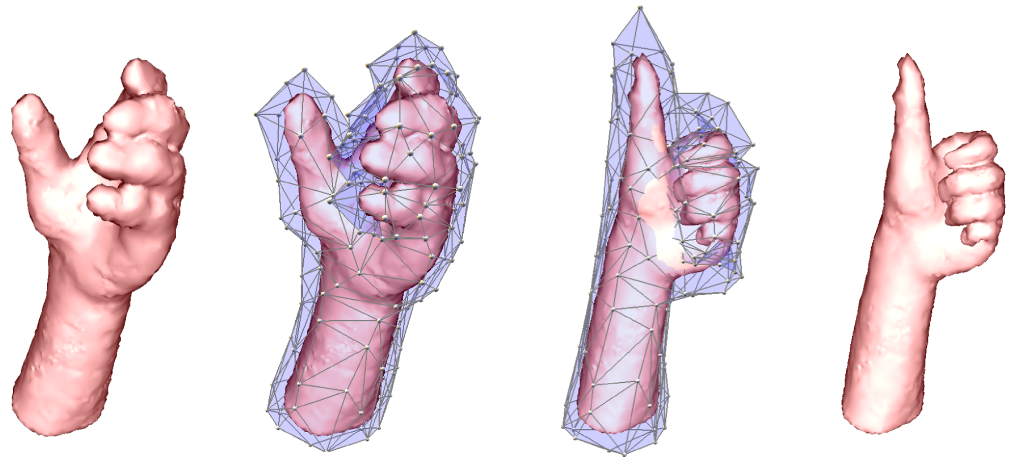


Figure 3.13: **Interactive FFD:** ARAP deformation performed on the cage, with GC deformation transfer to the visualization envelope.

3.2.3 Interactive deformation

During interaction, the user controls the deformation by manipulating cage vertices only. Indeed, moving each cage vertex independently is a tedious task, even for experienced users, and the result is often unpleasant. Therefore, we provide the user with a higher level mechanism by letting her specify a few positional constraints on the cage and solve for all the others using ARAP (see Section 3.1.3). In particular, the user can specify fixed constraints and new positions to reach (green and red spheres respectively in Figure 3.9)

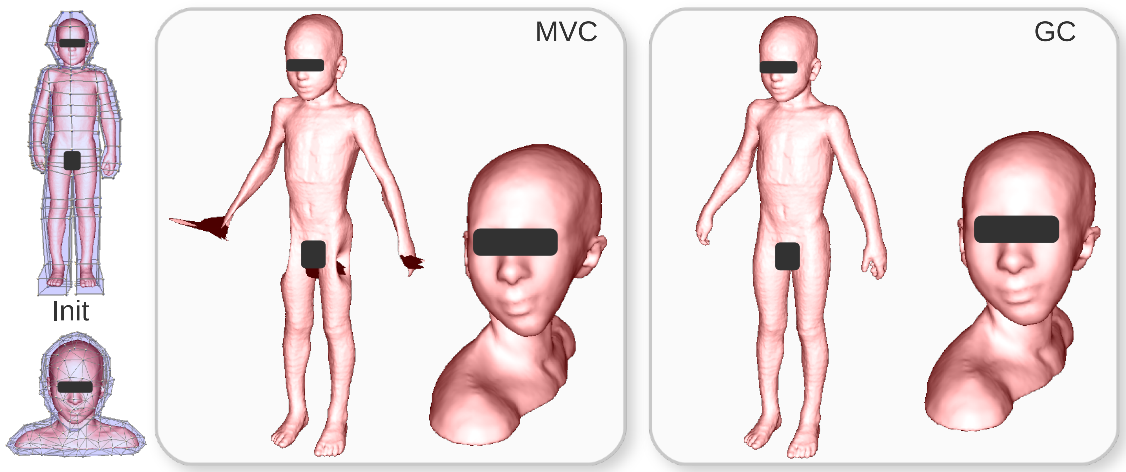


Figure 3.14: **Comparison:** Mean Value (MVC) versus Green (GC) coordinates in our application context. The cage proximity between the arm and the hip leads to a strong distortion when deforming using the MVC whereas GC offer a good separation of the space. Interpolating the coarse deformation with MVC exhibits ARAP distortions which are compensated by GC.

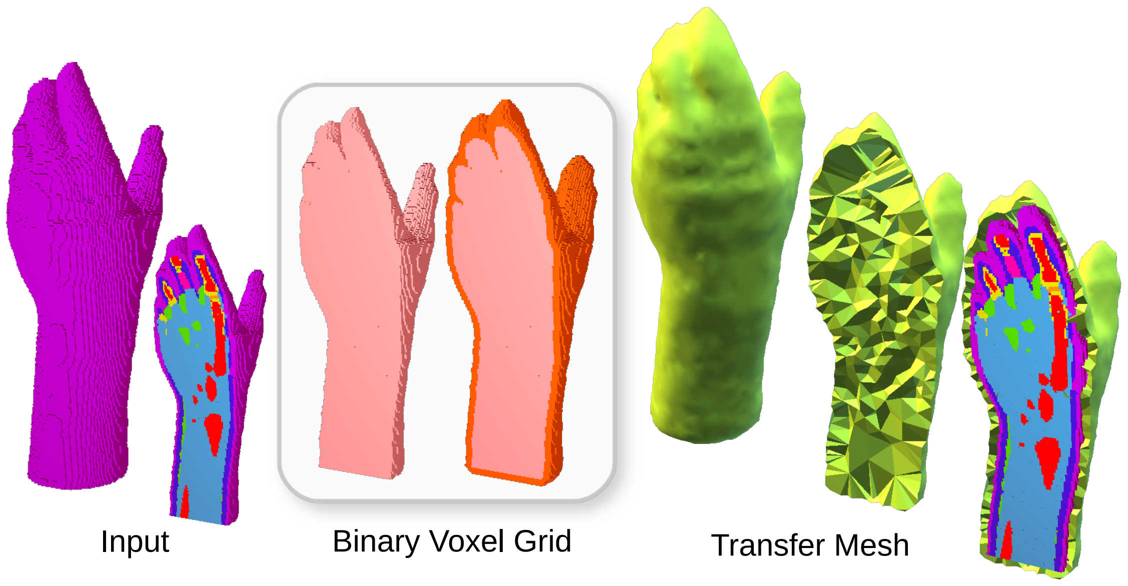


Figure 3.15: **Transfer mesh**: an adaptive Delaunay tetrahedral mesh undergoes the ARAP/GC deformation defined interactively and transmits it linearly to the input voxel grid for a limited per-voxel computational cost.

for an arbitrary subset of the cage vertices and the entire cage will undergo a plausible motion fitting the constraints (example Figure 3.13). The low number of cage vertices allows solving for ARAP interactively. The ARAP system can be disabled at any time by the user to allow per-vertex modeling.

Furthermore, the quasi-conformal property of GC compensates for part of the distortions that can still appear with ARAP (see Figure 3.14 right). The quasi-conformal nature of the deformation is needed in the context of medical image deformation, as the result is as close as possible to what can be obtained with physics-based deformations, using only a linear scheme. GC are the only linear coordinates that ensure this property.

3.2.4 Full resolution deformation

While the combination of ARAP and GC can cope with tens of thousands of samples, the voxel grid $\mathbb{G}$ is several orders of magnitude larger. Not only does this prevent us from using ARAP/GC interactively, but also the computation of GC (with $O(|P|)$ complexity for a single sample to deform) is prohibitive, even as a post-processing performed once the cage has reached a satisfactory shape. For instance, the full deformation of the complete voxel grids can take up to 5 hours and 20 minutes on the examples we provide in Section 3.3.

Therefore, we propose a *third* deformation scale, performed after the interactive FFD session, which is local and linear, and allows us to quickly rasterize all input voxels in the deformed space. As ARAP/GC deformations are smooth, they can be approximated, up to a target level of accuracy, using a linear operator. We propose to use an adaptive tetrahedral mesh – the *transfer mesh* (TM) – generated in the inner volume of the cage,

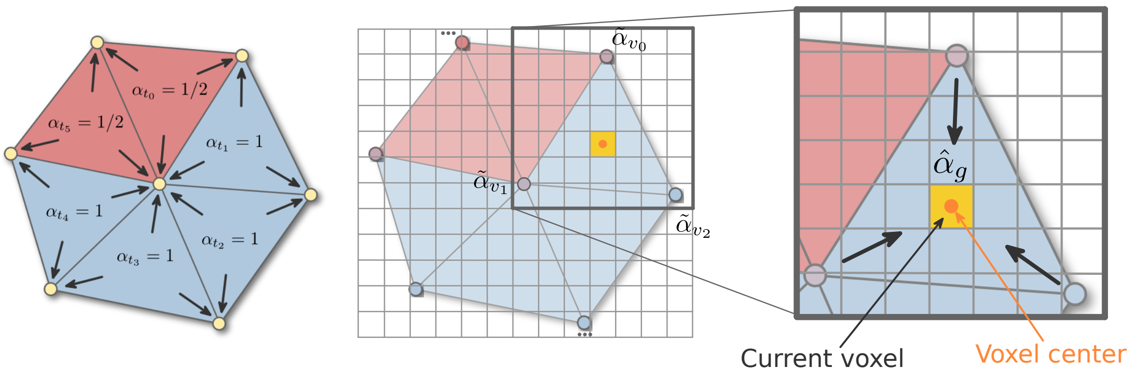


Figure 3.16: **Per-voxel resizing factor:** a per-tetrahedron resizing factor is computed using the per-tetrahedron error (red tetrahedra have an error higher than the threshold and a resizing factor of 0.5). A smooth per-vertex value is computed by averaging the one-neighbors tetrahedra values. Finally, a per-voxel value is computed using a barycentric interpolation. This value is used to update the sizing field.

and which both captures the ARAP/GC deformations and transfers it linearly, on a per-tetrahedron basis, to its inner space, providing quick deformation queries for any fine voxel.

The TM ensures the correct rasterization of the target grid, without introducing holes in the result. Furthermore, it captures the volume deformation in real-time and allows the user to track volumetric measures (distortion, stretch, ...) interactively. This feature is essential for the user to control the quality of the on-going deformation.

The key observation here is that linear deformation transfer reproduces rigid motions effectively. Therefore, the TM structure should adapt locally to quasi-rigid deformations, i.e., each tetrahedron should be small enough to bound non-rigidity in its motion. To do so, we start by generating an initial TM to sample a deformation error metric before refining the mesh structure adaptively using a spatially varying sizing field. This leads to a deformation-adaptive TM which is used in a 3D rasterization process to output a deformed voxel grid.

Initialization

We generate the initial TM by reusing $\mathbb{B}$. This time, we define the TM as an adaptive Delaunay *tetrahedrization* restricted to the binary volume, subject to a *sizing field* grid [CGA] $\mathbb{F}$ with the same resolution as $\mathbb{G}$ so that $\mathbb{F}[g]$ prescribes a target tetrahedron size in the vicinity of voxel g (see Figure 3.15). We initialize $\mathbb{F}$ to a constant value trading intuitively speed for accuracy (typically 5% of the voxel grid's diagonal in our experiments). The resulting TM is noted $M = \{V, T\}$ with $V = \{v_i\}_{i \in I_V} \subset \mathbb{R}^3$ its vertices, and $T = \{t_j\}_{j \in I_T}$ its tetrahedra indexed over V. The TM that is deformed using ARAP/GC is noted $M' = \{V', T'\}$.

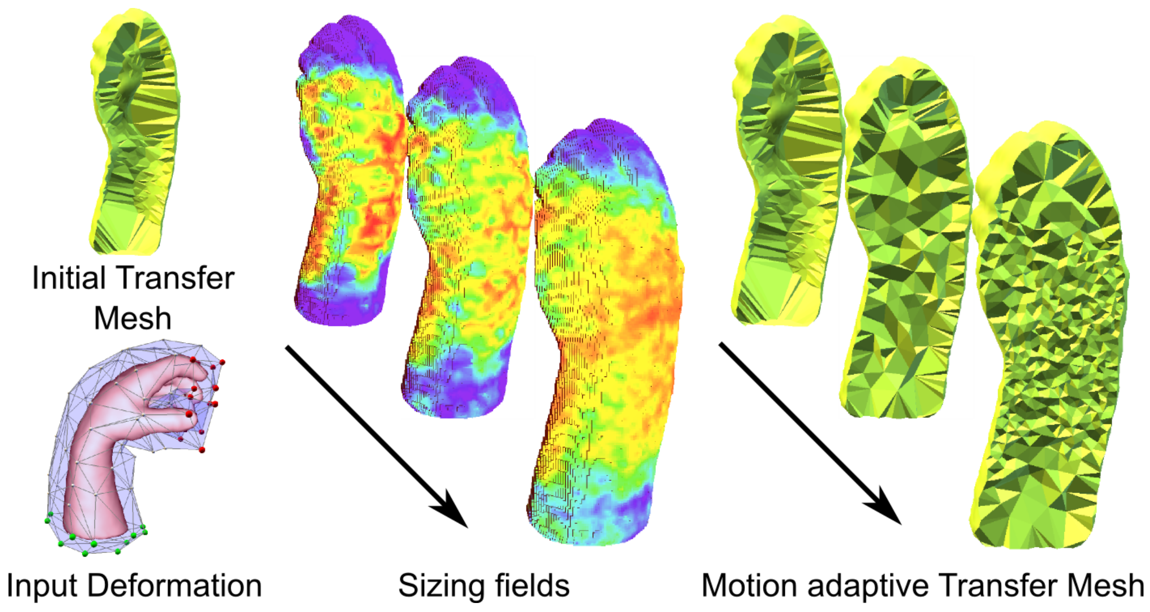


Figure 3.17: **Adaptive linear deformation transfer:** the Transfer Mesh (Right) is generated from a deformation-adaptive sizing field (Left).

Deformation-adaptive meshing

Each tetrahedron t_i of M defines a basis B_{t_i} which is transformed into B'_{t_i} by the ARAP/GC deformation. With $B'_{t_i} = B_{t_i} D_{t_i}$, the basis change matrix D_{t_i} is defined as:

$$D_{t_i} = B'_{t_i} B^{-1}_{t_i}.$$

Quick local variation of D_{t_i} in the volume indicates that a linear interpolation would provide only poor approximation quality. Therefore, we refine the TM iteratively using an adaptive meshing strategy (see Figure 3.17). First, we compute ΔB_t the Laplacian of the basis change weighted by the tetrahedra volume and note its norm $\mathcal{L}_{t_i}$:

$$\mathcal{L}_{t_i} = \frac{\left\| \sum_{j \in \mathcal{N}(t_i)} (D_{t_j} - D_{t_i}) |t_j| \right\|_f}{\sum_{j \in \mathcal{N}(t_i)} |t_j|}$$

with $\mathcal{N}(t_i)$ the 1-neighborhood of t_i, $|t_j|$ the volume of the neighbor j and $||.||_f$ the Frobenius norm. We measure a per-tetrahedron resizing factor α_{t_i} from $\mathcal{L}_{t_i}$ and a threshold h as:

$$\alpha_{t_i} = \begin{cases} 1 & \text{if } \ \mathcal{L}_{t_i} < h \\ 1/2 & \text{otherwise} \end{cases}$$

Second, we compute a per-vertex smooth value:

$$\tilde{\alpha}_{v_j} = \frac{\sum_{t_i \in \mathcal{N}(v_j)} \alpha_{t_i}}{card(\mathcal{N}(v_j))}$$

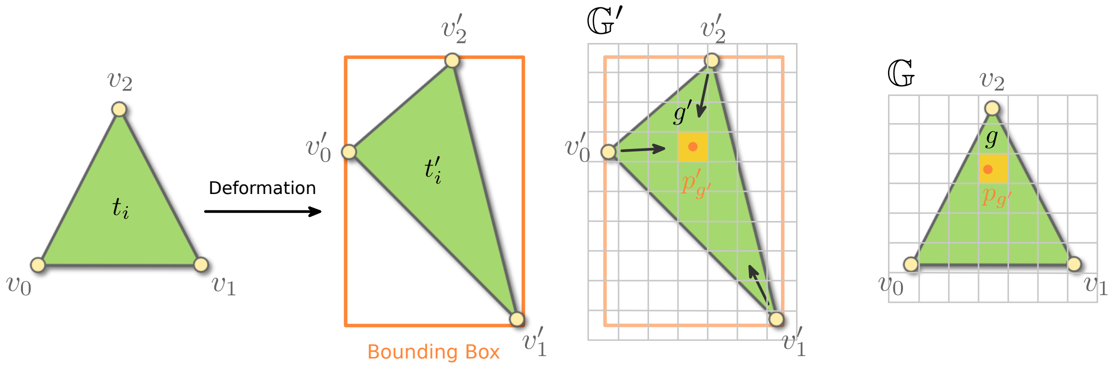


Figure 3.18: **Barycentric rasterization:** for each tetrahedron of TM, the bounding box is computed (orange) as well as the barycentric coordinates of the center of all the deformed grid voxels contained in the deformed tetrahedron. Using these coordinates and the initial TM vertex positions, we find their initial positions and project them in the initial grid to get the labels to assign to the final grid.

with $\mathcal{N}(v_j)$ the incident simplices of v_j. Third, we compute a per-voxel value $\hat{\alpha}_g$ using a barycentric interpolation:

$$\hat{\alpha}_g = \sum_{v_k \in t_g}^{3} \lambda_{v_k} \tilde{\alpha}_k$$

with λ_{v_k} the barycentric coordinates of the center of g in the tetrahedron t_g intersecting it (Figure 3.16). Note that the interpolation of resizing factors is critical to avoid discretization artifacts during remeshing. Lastly, we update $\mathbb{F}$:

$$\mathbb{F}[g]_{updated} = \mathbb{F}[g]\hat{\alpha}_g$$

and update the TM accordingly. This process is iterated until $\mathbb{F}$ converges or a prescribed TM resolution is reached.

Barycentric rasterization

The final step of this third-scale deformation consists in computing the actual output grid $\mathbb{G}'$. We start by computing the bounding box of the deformed cage and create $\mathbb{G}'$ with a voxel size similar to $\mathbb{G}$. Then, we iterate over the TM tetrahedra, and assign a label to the voxels contained inside it. To do so, we compute the bounding box of each tetrahedra t'_i, then we iterate over the voxels in it: for each voxel g' that has not been visited, and is in t'_i, we compute its center's barycentric coordinates, noted $\lambda_{v'_k}$, in t'_i, such that:

$$p'_{g'} = \sum_{v'_k \in t'_i}^{3} \lambda_{v'_k} p_{v'_k}$$

with $p'_{g'}$ the voxel center's position and $p_{v'_k}$ the positions of t'_i's vertices. Then, we multiply them with the coordinates of the corresponding tetrahedron t_i, noted p_{v_k}, in M to obtain a 3D position in the initial space, noted $p_{g'}$, as follows:

$$p_{g'} = \sum_{v_k \in t_i}^{3} \lambda_{v'_k} p_{v'_k}.$$

The corresponding voxel g in the initial grid is found by projecting the resulting position in $\mathbb{G}$. Finally, we set $\mathbb{G}'[g'] = \mathbb{G}[g]$. The overall rasterization process is illustrated in Figure 3.18. Once all the tetrahedra have been processed, the voxels that have not been

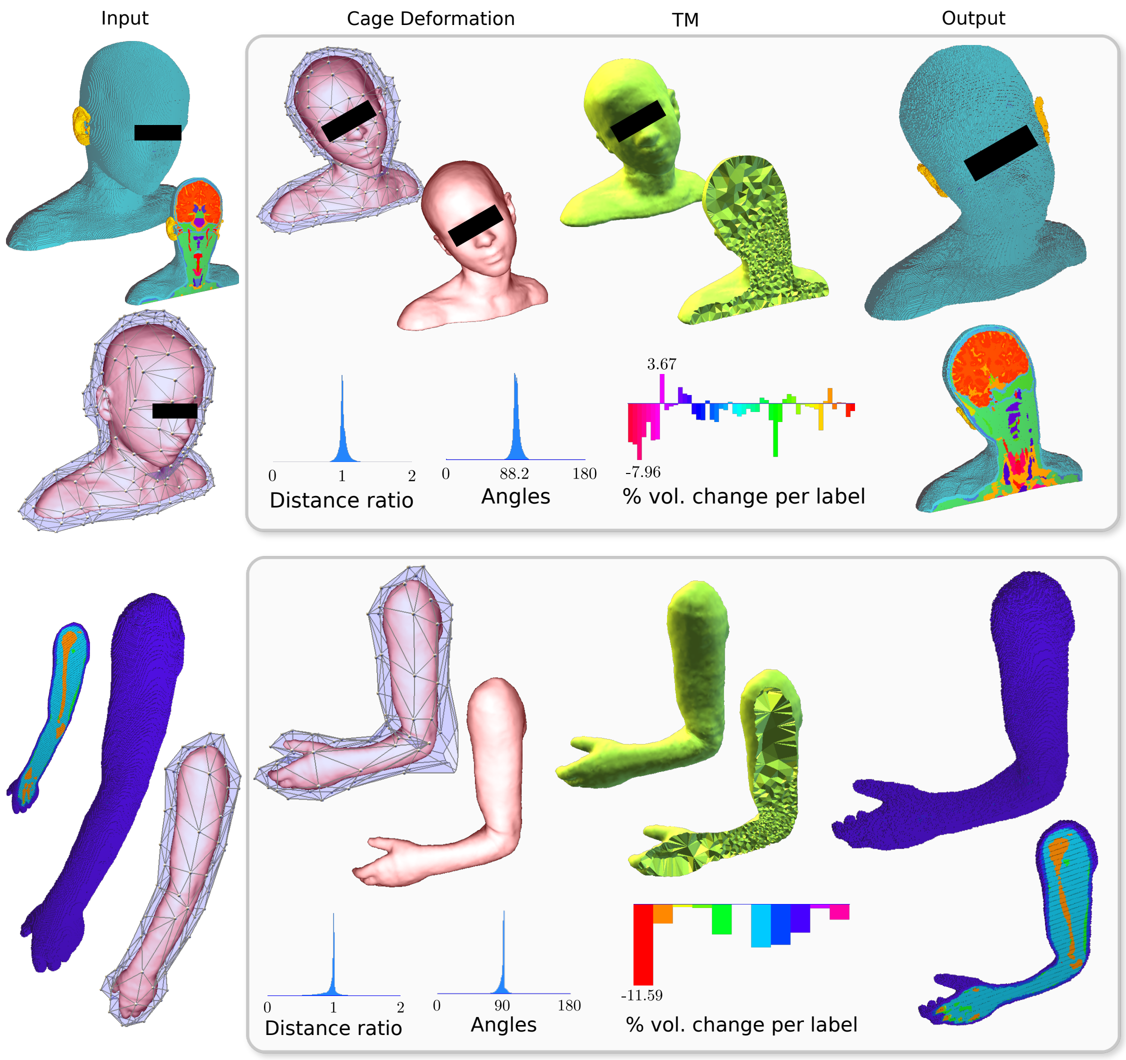


Figure 3.19: Deformation of voxel grids using VoxMorph. The *distance ratio* and *angles* are calculated on the dual of the voxel grid. The error histograms indicate that the deformations are mostly conformal, and induce neither stretch nor angle distortions. The labels that endure large volume changes are the ones with an originally very small volume: these variations come from unavoidable rasterization artifacts.

visited are set to 0, since they are not in the TM (i.e., background).

Finally, note that the construction of the TM we presented is generic and is not restricted to a particular underlying deformation scheme.

3.3 Implementation and results

We implemented our system in C++ with OpenGL for rendering. We used CHOLMOD [CDHR08] as a Cholesky solver and the CGAL library [CGA] for tetrahedral mesh generation, surface meshing and simplification. Performances were measured on an Intel Core2 Duo at 2.4 GHz with 8GB of main memory.

In Figure 3.19, we illustrate the deformation of two voxel grids representing a head model in the top row and an arm in the bottom raw. Figures 3.21 and 3.20 illustrate deformations of whole-body high-resolution voxel grids. The blue histograms represent the deviation from a locally rigid deformation: the first one shows angle distribution of genuine orthogonal pairs after deformation (no distortion at 90°) and the second the stretch (no distortion at 1). We can see that the resulting deformations are close to locally rigid ones, even under strong motions, and that the stretch is minimized. Multi-color histograms show the volume change on a per-label basis after deformation. The global volume change is bounded, in the worst case, to 11.6% for all the experiments we conducted. These measures validate our 3-scale approach, in particular the new adaptive fine-scale stage. Table 3.1 summarizes the performance of our system on the models from Figure 3.19 (and others). On the largest model, a full resolution deformation can still be obtained in less than 8 minutes, which is more than 50 times faster than a full GC deformation.

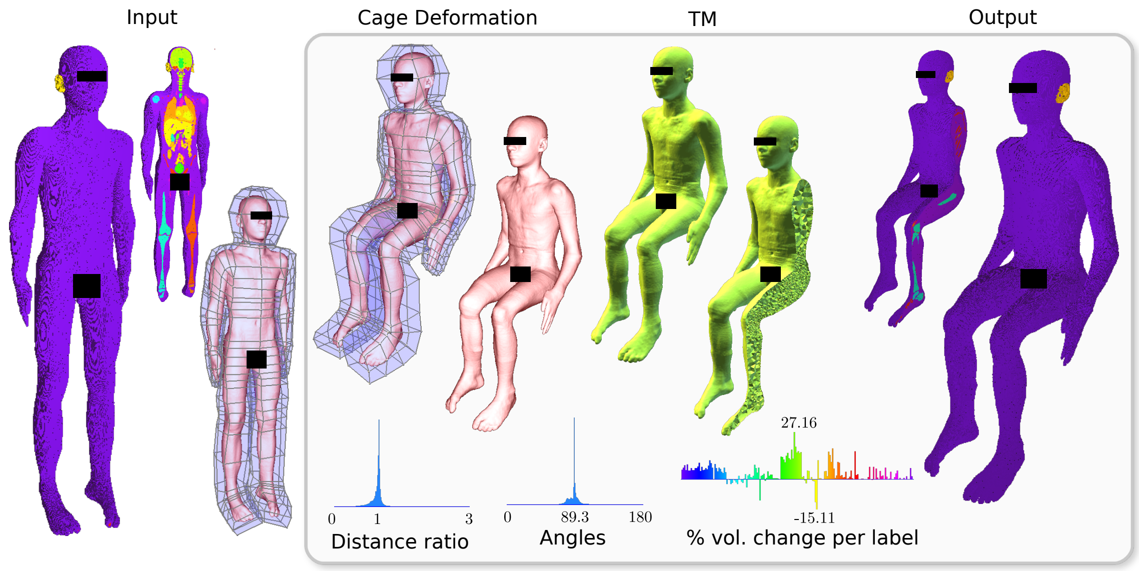


Figure 3.20: **Louis model.** Deformation of a whole-body voxel grid made of 412 million voxels.

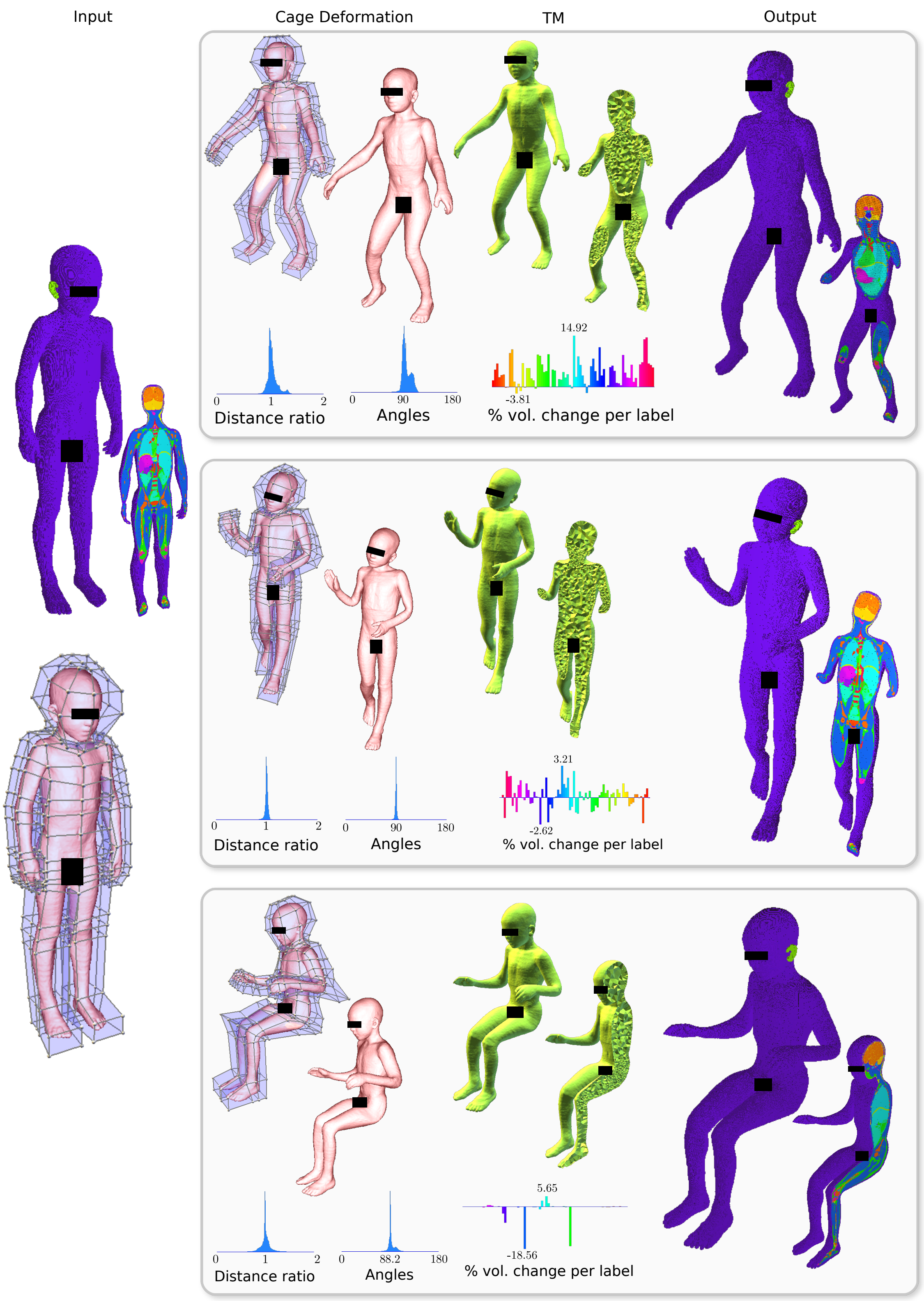


Figure 3.21: **Thelonious model.** Deformation of a whole-body voxel grids.

Model	Voxels	CV	EV	FPS	TV	CT
Hand	2 985k	263	2 545	50	9 879	28s
Head	4 792k	263	15 628	7.8	31 672	1m47
Arm	8 990k	95	6 264	28	15 314	56s
Louis	412M	512	44 748	3.	168 712	7m55
Thelonious	122M	303	61 347	3.	78 789	4m17

Table 3.1: **Performance table**: with CV (resp. EV and TV) the cage (resp. envelope and transfer mesh) vertex count, FPS the framerate during interaction and CT the final full deformation post-processing time on an Intel Core2 Duo at 2.4 GHz with 8GB of main memory.

3.4 Towards a more robust and scalable pipeline

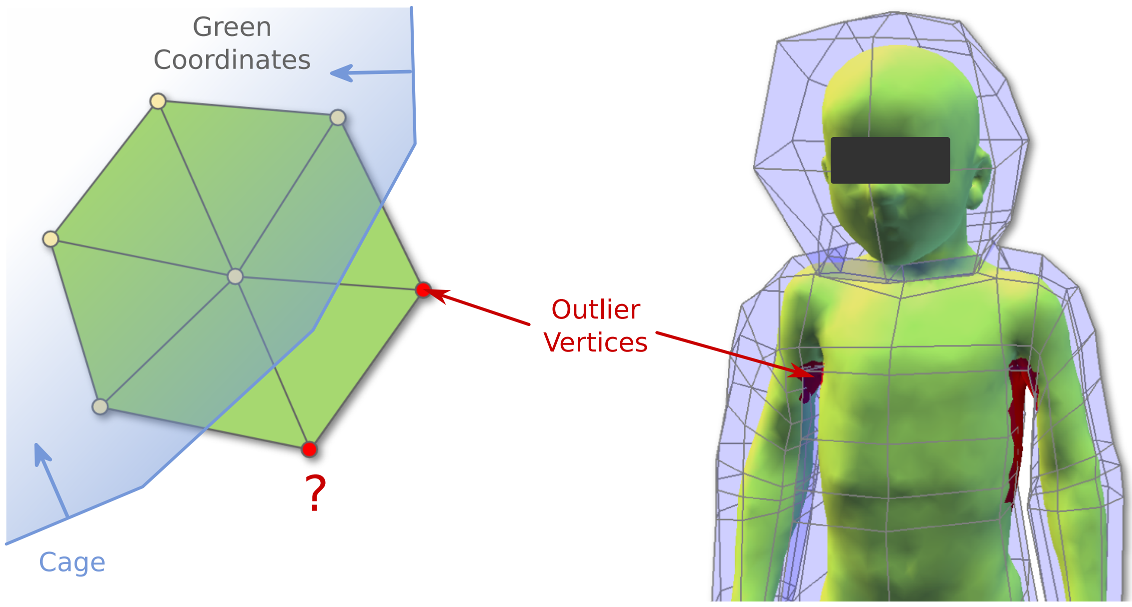
Our deformation pipeline successfully generates high-quality deformations of high-resolution voxel models without resorting to full resolution meshing. As we target general shape modeling scenarios, our system allows for arbitrary deformations. Similarly, while bending an arm is easier with a skeleton, editing the morphology of a face is easier with our system. In fact, cage-based systems such as ours are typically complementary to a skeleton-based one. Here again, a 3-scale approach could be developed. Although existing spatial or surface deformation methods could be used to replace the two first scales of our pipeline, our particular combination of ARAP and GC seems to outperform consistently other choices in the numerous experiments we conducted.

The (optional) automatic cage creation process raises numerous interesting questions. For instance, how to handle two distinct components that are too close: the dilation step may not lead to a cage allowing the user to manipulate them independently. Also, the QEM simplification does not guarantee to still bound the volume of interest. Fortunately, both cases can be fixed by adjusting interactively the dilation factor, imposing manual editing of the cage structure only in very rare cases.

Currently, our system is limited to voxel grids fitting in memory. Scaling up could be achieved by using compressed representations or out-of-core/streaming environments.

Since we use GC, the model needs to be strictly embedded into the cage otherwise outliers are generated for which the deformation is unknown (Figure 3.22).

In the next chapter, we propose a user interface to supervise the cage generation process, as well as a novel tetrahedral variational deformation method to extend cage-based deformations to the outside of the cage model. We also deal with larger datasets using an out-of-core method.



[redacted]

Figure 3.22: **Limitation:** since we use GC, the entire model has to fit in the controlling cage, and when the cage fails to meet this condition, usually in regions where the limbs are close together, the deformation of the vertices outside the cage, called outlier vertices (Red), is unknown. We tackle this issue in the next chapter.

CHAPTER 4

Robust and scalable volume data deformation

The VoxMorph system, presented in the previous chapter, has so far three main limitations: first, the entire model has to fit in the controlling cage **strictly**, which makes its construction very difficult either in an automatic or a supervised context [LLCO08]. Second, when transferring the deformation from the coarse scales to the finer ones, the adaptive metric plays a critical role in the deformation quality which can be improved. Lastly, although VoxMorph is able to deform large images, it remains restricted to in-core datasets limiting the scope of usable data since the increasing accuracy of modern 3D acquisition systems generate huge high-definition datasets.

In this chapter, we make the following contributions. First, we increase the flexibility of the system by proposing a limb separation interface, and deform parts of the model that are outside the cage using a new variational approach on our transfer mesh. Secondly, to improve the accuracy, we propose a new error metric that avoids refining too deeply the transmission structure between mid and high-resolution data. Thirdly, we propose an out-of-core implementation in order to scale up to datasets that do not fit in memory. As a result, our new system can handle a larger range of datasets. We apply our pipeline by performing digital dosimetry analysis for radio-wave effects studies on high-resolution segmented 3D medical images posed with our approach. All the representations in this pipeline are generated with our system.

4.1 Image data and segmented volumes

The input of our system are medical images and the output are images containing interactively deformed models. These images are segmented and the derived models are used to perform SAR analysis as discussed in the introduction of this thesis (see Chapter 1). As in the previous chapter, we note $\mathbb{G}$ the grid representing the organ segmentation and we use an additional grid, noted $\mathbb{S}$, representing a body part segmentation. This second grid is used to separate unwanted links between body parts. If none is available or necessary (i. e. no limb separation needed), we use a binary voxel grid with $\mathbb{S}[g] = 0$ for null voxels and $\mathbb{S}[g] = 1$ otherwise. Note that the user can create a rough segmentation interactively.

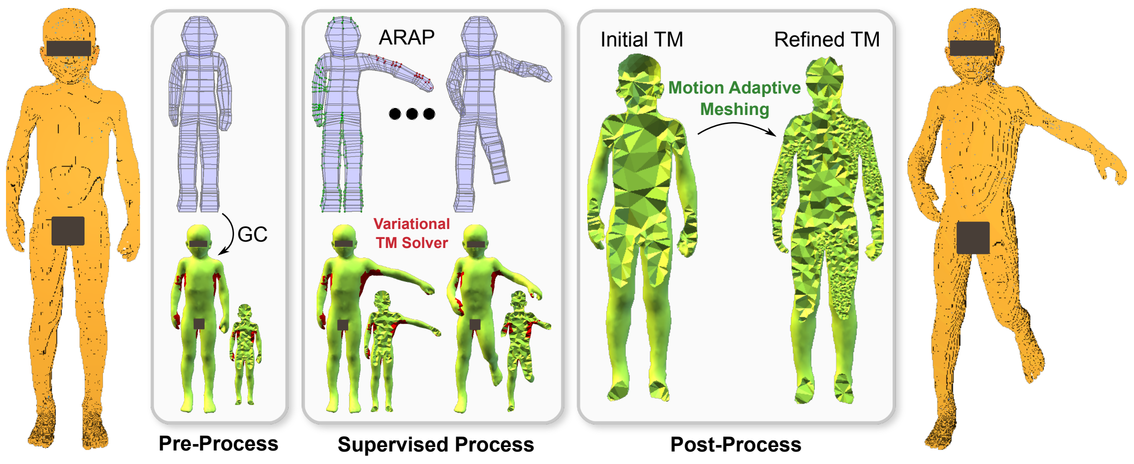


Figure 4.1: **Pipeline overview:** the input is a segmented voxel grid. We construct a cage and a tetrahedral mesh (i.e. the transfer mesh TM) with segmentation-aware connectivity from it. The user manipulates the cage – deforming the TM vertices that are inside it (in green) using cage coordinates – and the tetrahedral solver computes the position of the outlier vertices (in red) interactively. The deformation is transferred from the TM to the grid in real-time. The adaptive resolution of the TM is driven by the deformation.

4.2 A robust and scalable deformation method

The cage can cross the model very often, excluding part of the voxels from the space deformation; we tackle this problem by designing a new variational optimization technique, which conveys the inner space deformation to outer voxels. To improve the quality of the deformation, we propose a new error metric to guide the motion adaptive refinement process. Finally, to make the pipeline scalable, we propose an out-of-core deformation process for the high-resolution voxel grid. Our general pipeline is illustrated in Figure 4.1.

4.2.1 Robust cages

To initialize the system, the user supervises a morphological cage creation process acting on $\mathbb{S}$. In the VoxMorph system, a uniform dilation of $\mathbb{S}$ is performed. Then, a high-resolution 2D restricted Delaunay triangulation (RDT) is generated, capturing the interface between the resulting grid and the background, which is simplified to a prescribed resolution (typically a few tens or hundreds of vertices). This process can create unwanted links between limbs (e.g. armpit, hand and hip) and therefore generate a cage with an incorrect topology. In our system, we propose a limb separation supervised process by offering the possibility to create inset cages: the user selects a region where two limbs need to be separated, the voxels located at their interface are eroded automatically while others are adaptively dilated (Figure 4.2). Finally, the aforementioned surface creation and simplification process is applied.

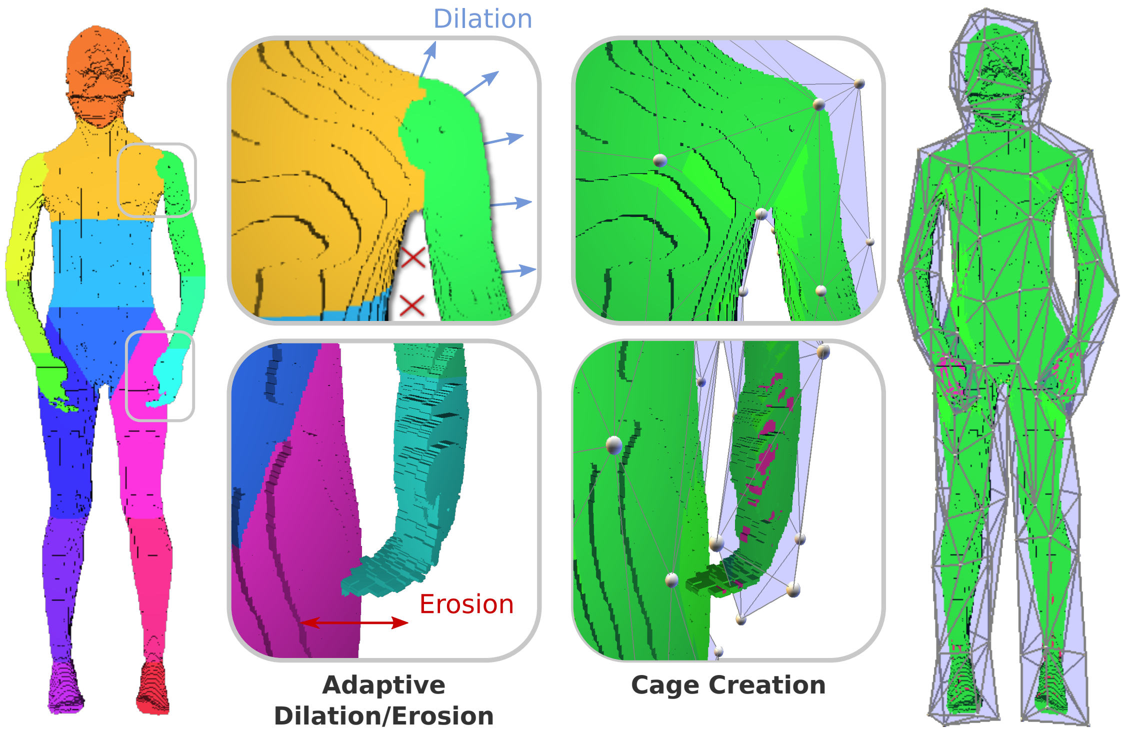


Figure 4.2: **Robust cage creation:** voxels, located at the interface between labels to separate, are eroded in user defined regions while adaptively dilating elsewhere in order to generate an inset cage with a topology different from the input model's allowing limb separation.

4.2.2 Accurate motion-adaptive structure

GC require the cage to enclose the model to be deformed, making its creation extremely difficult even for expert users, as well as for automatic methods. As explained in the previous chapter, the full deformation of $\mathbb{G}$ using GC can be prohibitive (e.g., several hours for the models we present in Section 4.3). Lipman et al. [LLCO08] explained how to extend linearly the coordinates through a set of faces of the cage, making the use of partial cages possible. In their setup, the system identifies the set of coordinates that are *valid*, and the values for the other coordinates are found by inverting a linear system. This approach requires the user to specify manually the faces that need to be extended. More importantly, we need the extension of the deformation to help separating *distinct glued limbs*, and therefore it cannot be driven by a *space separation method* similar to the use of a cage, as points at similar positions (we duplicate vertices in our case) have to be placed at different positions. Consequently, the direct use of the extension of the coordinates through facets of the cage is impossible in our context.

In our VoxMorph system the scalability issue is solved using a volumetric tetrahedral structure based on $\mathbb{G}$: the *Transfer Mesh* (TM), which is built to enclose the volume to process and has a resolution that is adapted to the deformation. We propose to apply a deformation propagation method performed on this structure to solve the robustness problem. We remind the notations used in the previous chapter and detail the additional ones. We denote $TM = \{V, E, T\}$ with $V = \{v_i\} \subset \mathbb{R}^3$ its vertices, $E = \{e_{ij}\}$ its edges connecting adjacent vertices v_i and v_j, and $T = \{t_k\}$ its tetrahedra.

All TM vertices that are inside the cage are deformed using GC, and all others' positions are recovered by minimizing a *rigidity energy* on the TM (Figure 4.5). The space deformation is transmitted to the voxel grid using barycentric interpolation, which guarantees *consistent rasterization* of the target voxel grid, and allows real-time deformations of $\mathbb{G}$.

Finally, to cope with the approximations introduced by the linearization scheme, we propose a new error metric for the iterative refinement strategy of the TM, speeding-up the refinement process.

In the following, $T_1(v_i)$ denotes the set of tetrahedra adjacent to a vertex v_i, $T_1(e_{ij})$ the set of tetrahedra adjacent to an edge e_{ij}, and $T_1(t)$ the set of tetrahedra sharing a face with a tetrahedron t. We note B_t the 3×3 transformation matrix of each tetrahedron t, uniquely defined by the transformation of its 4 vertices, $|t|$ its volume before deformation, and ΔB_t the local change of the transformation matrices expressed as:

$$\Delta B_t = B_t - \frac{\sum_{t_n \in T_1(t)} |t_n| \cdot B_{t_n}}{\sum_{t_n \in T_1(t)} |t_n|}. \tag{4.1}$$

We model the energy to minimize as:

$$e_{rigid} = \sum_t |t| \cdot ||\Delta B_t||^2. \tag{4.2}$$

using the Frobenius norm.

Set up

We construct the TM as a restricted adaptive Delaunay multi-material *tetrahedrization* generated from $\mathbb{S}$, the limb segmentation associated with $\mathbb{G}$, and constrained by a *sizing field* [CGA], which allows us to control the spatially varying tetrahedron size explicitly. Depending on the segmentation, the resulting mesh can be composed of several subdomains, therefore a label l_k is associated with each tetrahedron.

Label separation

We use the limb segmentation to split the TM in user-defined regions in order to remove unwanted limb connections. The user draws a selection area and provides a pair of labels to be separated. In this area, we duplicate the vertices of the TM that belong to a face common to two tetrahedra to be separated, i. e. one labeled in the first subdomain and the other in the second. We add the new vertices to the mesh, and re-index the tetrahedra in the second subdomain over them, consequently cutting the mesh (Figure 4.3).

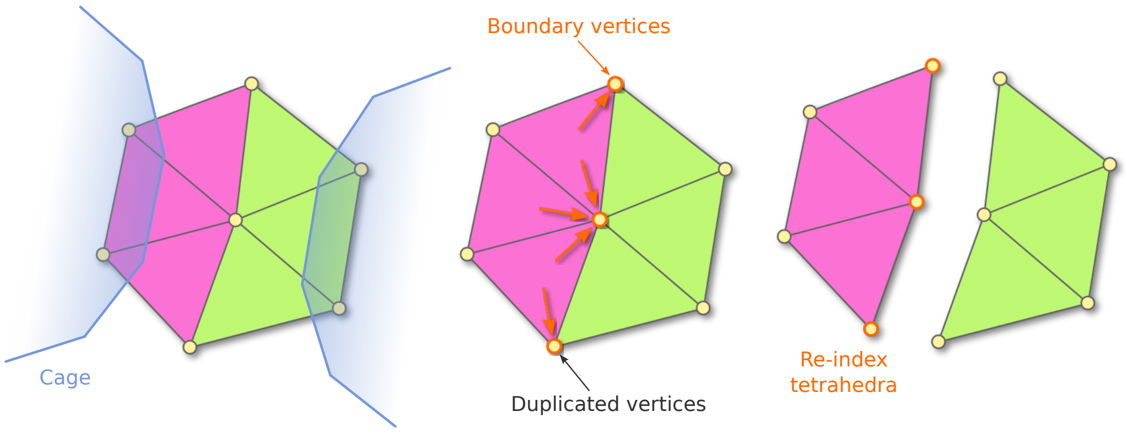


Figure 4.3: **Tetrahedral mesh cutting:** in user defined regions, we duplicate the vertices of the TM that belong to a face common to two tetrahedra to be separated. We add the new vertices to the mesh, and re-index the tetrahedra of one of the subdomain over them.

Improving robustness with a tetrahedral solver

The minimization of e_{rigid} is performed in two steps. We note T_c the tetrahedra of the TM whose four vertices are inside the cage (blue in Figure 4.4 center), and V_c the vertices of TM that are inside the cage (Figure 4.4 right).

In the first step, we compute B_{t_c} for all $t_c \in T_c$ (these are deformed using GC) and set them as constraints, that we note $\tilde{B}_{t_c}$. We then set ΔB_{t_u} to be the zero 3×3-matrix for all others - i.e. tetrahedra with at least one outlier vertex noted t_u (red in Figure 4.4).

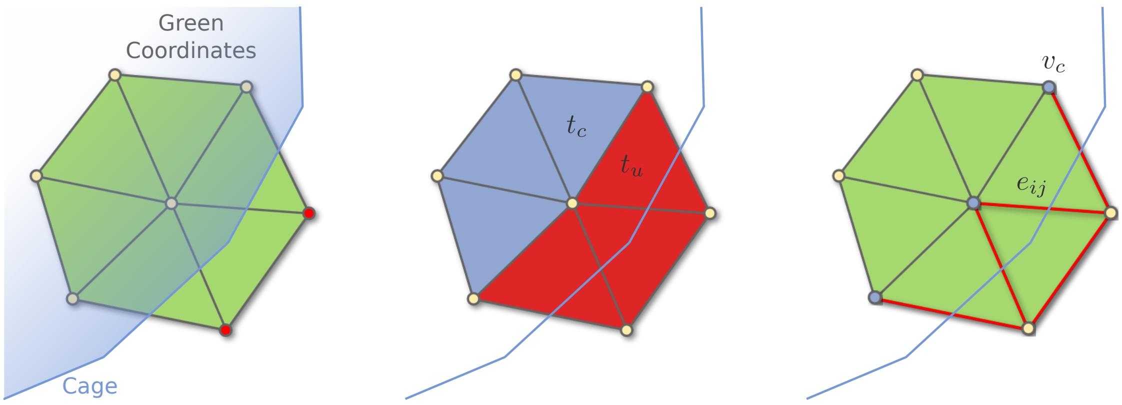


Figure 4.4: **Deformation propagation setup:** (Left) outlier vertices in red. (Center) The tetrahedra for which the transformation is unknown are marked in red and the others, which are set as constraints, in blue. (Right) The blue spheres are the constraint vertices and the red edges, used to recover the deformed outlier vertices positions, are incident to at least one outlier vertex.

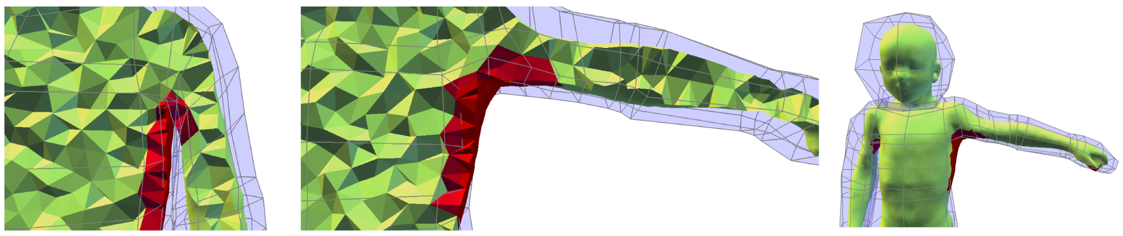


Figure 4.5: **Deformation of the outlier vertices:** the green tetrahedra are deformed using GC, the red tetrahedra's geometry – partially located outside the cage – is recovered by minimizing e_{rigid}.

This is done by solving the following linear system in the least squares sense, with B_{t_u} its unknowns:

$$\begin{cases} \sqrt{|t_c|} \cdot B_{t_c} & = \sqrt{|t_c|} \cdot \tilde{B}_{t_c} \quad \forall t_c \in T_c \\ \sqrt{|t_u|} \cdot \Delta B_{t_u} & = 0_{33} \qquad\qquad \forall t_u \in T \setminus T_c \end{cases} \tag{4.3}$$

The result is a set of transformation matrices B_t for all $t \in T \setminus T_c$.

In the second step, we recover the vertex positions of the TM, using the transformation matrices of the tetrahedra. We add the positions of all vertices in V_c as constraints, and set the edges to be the initial ones transformed by B_t. This is done by solving the following linear system in the least squares sense:

$$\begin{cases} \sqrt{\sum_{t \in T_1(v_c)} |t|} \cdot v_c & = \sqrt{\sum_{t \in T_1(v_c)} |t|} \cdot \tilde{v}_c \\ \sqrt{\sum_{t \in T_1(e_{ij})} |t|} \cdot (v_i - v_j) & = \frac{\sum_{t \in T_1(e_{ij})} |t| \cdot B_t \cdot (v_i^{init} - v_j^{init})}{\sqrt{\sum_{t \in T_1(e_{ij})} |t|}} \end{cases} \tag{4.4}$$

$\forall v_c \in V_c$ and $\forall e_{ij} \in E$. The result is a set of positions for all vertices of the TM that were outside the cage during the embedding (Figure 4.5). The linear systems are factorized at the creation of the TM and solved efficiently at each frame using CHOLMOD, a linear algebra library [CDHR08].

Improving the quality with a new error-metric

To cope with the piecewise linear approximation introduced by the TM, we make the sizing field adaptive to the deformation by refining the TM iteratively.

At each step, we define an error e_t for each tetrahedron t as

$$e_t = ||\Delta B_t||, \tag{4.5}$$

and an error e_v for each vertex v as

$$e_v = \frac{1}{\sum_{t \in T_1(v)} |t|} \sum_{t \in T_1(v)} |t| e_t \tag{4.6}$$

the mean of the errors of its adjacent tetrahedra. We then obtain a smooth *error metric* e_g on the initial grid by rasterizing each tetrahedron, filling up the grid with the barycentric

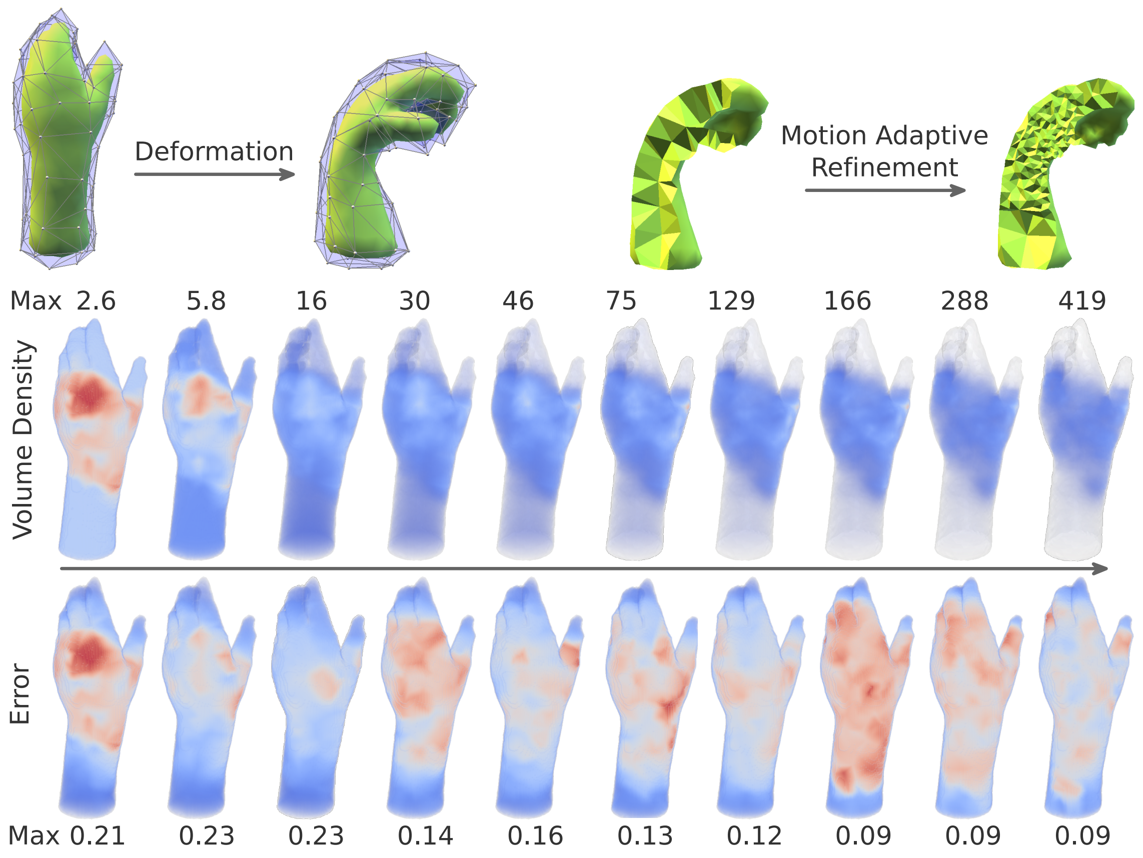


Figure 4.6: **Error measures:** at each iteration, the volume density (top) of the Transfer Mesh is refined, so that the error due to the linearization of the deformation (bottom) decreases and eventually becomes uniform.

interpolation of its vertices' errors. The sizing field is updated according to the error metric with the following rule:

$$\mathbb{F}[g]_{updated} = \mathbb{F}[g]\sqrt[3]{\frac{e_{mean}}{\max(e_g, e_{mean})}}. \tag{4.7}$$

with e_{mean} the grid mean error. This process allows us to obtain a uniform error on the grid at convergence (Figure 4.6). This new metric allows us to reduce the final resolution by about 40% on average compared to the original VoxMorph system.

4.2.3 Scalable out-of-core deformation upsampling

In the previous version of our system, the deformation of the grid is performed in-core by computing the barycentric coordinates of the center of all the deformed grid voxels contained in the deformed tetrahedra. Using these coordinates and the initial TM vertex positions, we find their initial positions and project them in the initial grid to get the labels to be assigned to the final grid. In order to process out-of-core datasets, we propose to perform the entire pipeline on a low resolution version of the grid, which is down-sampled on the fly, and to apply an out-of-core rasterization of the high-resolution one (Figure 4.7).

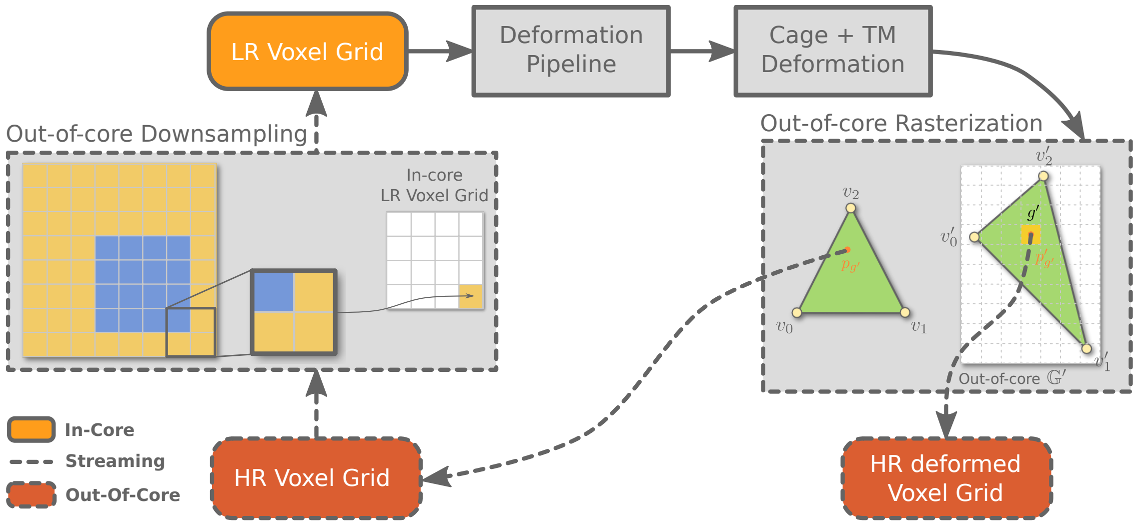


Figure 4.7: **Out-of-core pipeline:** the deformation pipeline is applied on a low resolution version of the input high-resolution voxel grid which is never loaded in main memory. The final result is obtained by performing an out-of-core rasterization.

Note that the grid down-sampling, illustrated in Figure 4.7, does not need to be precise since the final goal is to deform the high-resolution grid. We implemented a streaming process to perform the per-voxel deformation, therefore, both the initial and deformed high-resolution voxel grids do not need to be loaded in memory. We use the TM of the low resolution image to assign the label to the high-resolution one. To do so, we apply the previously described process to all the voxels of the deformed high-resolution grid contained in the deformed tetrahedra and fetch in memory the voxels values where the positions project and directly write the result to disk (Figure 4.7).

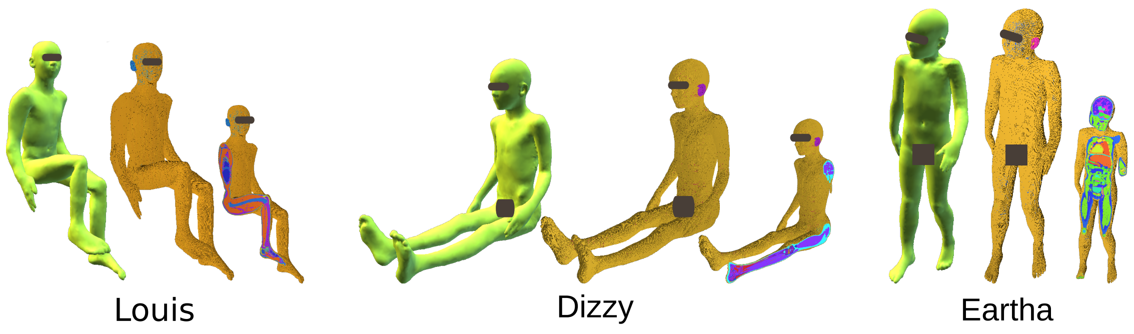


Figure 4.8: **Results:** 3 other HD 3D images deformed with our system.

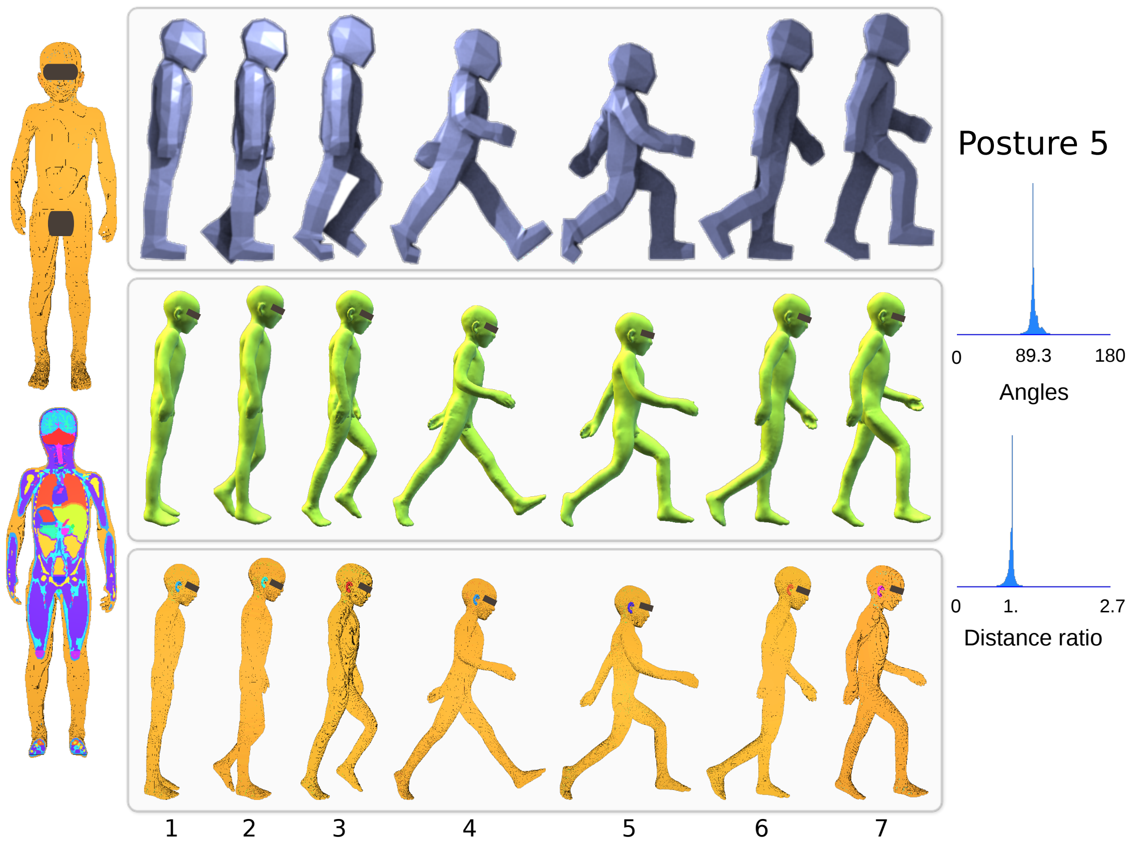


Figure 4.9: **Whole-body deformations:** 7 poses simulating a walk animation of the Thelonious model [CKH+10]. The first row shows the cage controlled by the user, the second row the automatic motion adaptive TM and the last row the resulting, full resolution segmented voxel grids after deformation. The deviation from the quasi-conformal deformation is computed for the fifth position which has the highest volume change.

4.3 Results

We demonstrate the validity of our method by performing high-quality deformations on several high-resolution whole-body voxel grids and perform a wave exposure simulation on some of the resulting grids.

4.3.1 Deformation evaluation

We implemented our system in C++ with OpenGL for rendering. We used CHOLMOD [CDHR08] as a Cholesky solver, GSL [Con10] for the SVD and the CGAL library [CGA] for tetrahedral mesh generation, surface meshing and simplification. Performances were measured on an Intel Core2 Duo at 2.4 GHz with 8GB of main memory.

Figures 4.8, 4.9 and 4.10 illustrate the results obtained with our method for 6 high-resolution whole-body segmented images. The 7 postures of the Thelonious model have been interactively designed using our system and are further used to perform dosimetry

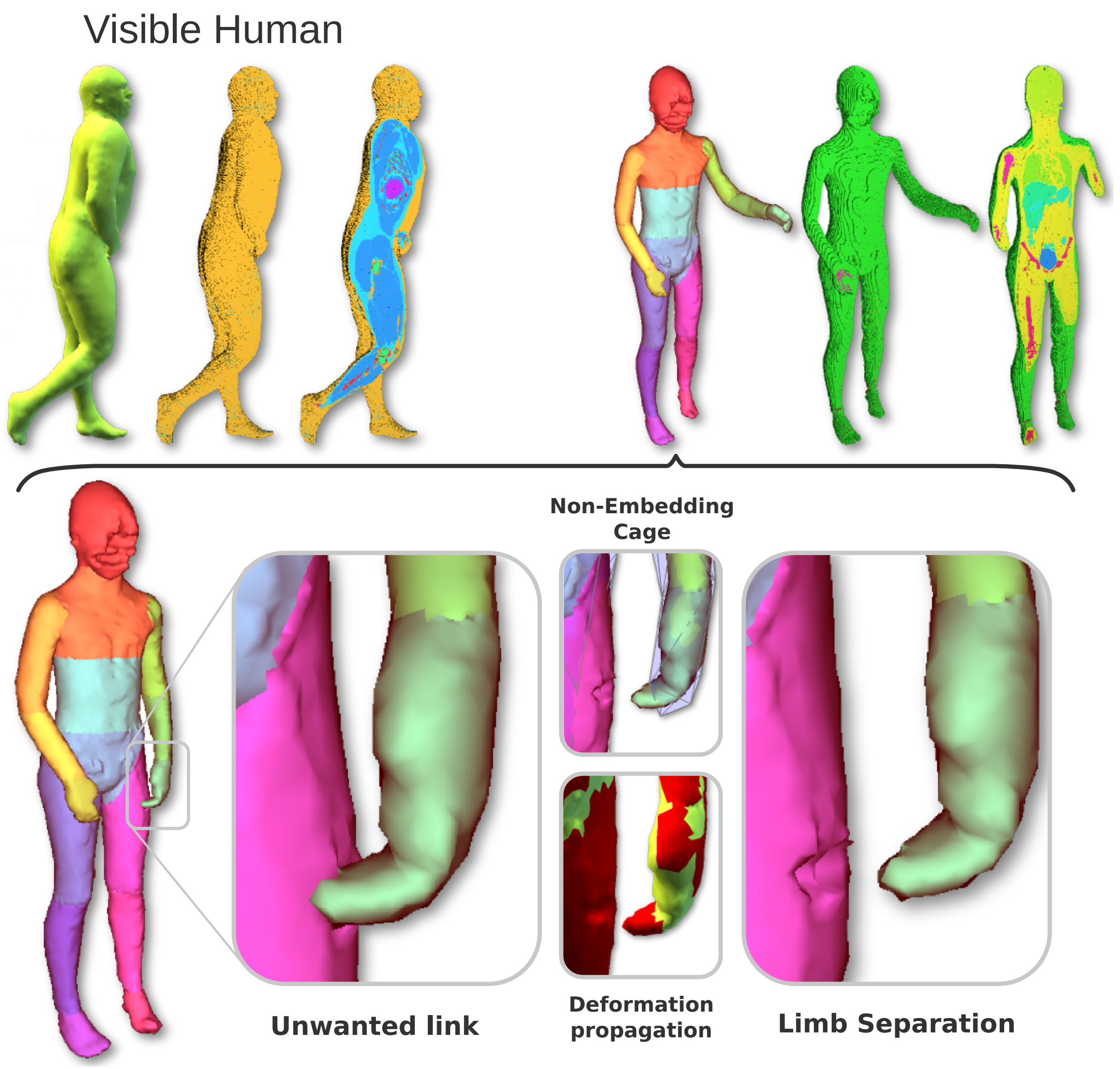


Figure 4.10: (Left) The Visible Human deformation was performed out-of-core and (Right) limb separation.

analysis. The highest weight variation is -7% for Posture 5, which is still acceptable. For this posture, which is supposedly the worst case, the blue histograms illustrate the quasi-conformal nature of the resulting deformation. For all other postures, the difference is less than 1%. Figure 4.10 illustrates the out-of-core deformation of the Visible Human model and an example of our limb separation approach. Table 1 summarizes various performance measures of our system on models ranging from 122 to 260 million voxels in core and one out-of-core deformation resulting in a grid of 862 million voxels. Note that, in all cases, the framerate (FPS) is limited by the rendering capabilities and not by our deformation workload. All these models contain outliers, and therefore the tetrahedral solver was required to deform them.

Model	Voxels	CV	FPS	TV	Outliers	CT
Thelonious	122M	303	11.	36 157	393	2m31
Eartha	243M	479	9.	33 308	1890	3m10
Louis	260M	512	7.	27 588	2	4m03
Dizzy	141M	479	10.	29 226	532	2m38
LR Visible Human	108M	152	11.	17 036	1317	1m28
HR Visible Human	*374M*	-	-	-	-	*11m33*

Table 4.1: **Performance table**: with CV (resp. TV) the cage (resp. transfer mesh) vertex count, FPS the framerate during interaction and CT the final full deformation pre- and post-process time. The last row shows the out-of-core processing time to deform the High-Resolution Visible Human model, using the Low-Resolution version for the user interaction process, into a grid of 862 million voxels.

4.3.2 Application to dosimetry analysis

With the help of our partners at Orange Labs, we demonstrate the validity of our approach by simulating human exposure to waves on a whole-body model in different postures in order to evalute the influence of the model's position on the SAR.

Materials and numerical simulation conditions

A far-field exposure of the walking Thelonious, composed of more than 70 different tissues with a 2 mm resolution, was simulated using an incident plane wave polarized vertically, with a left side incidence, emitting at the frequency of 2100 MHz. The whole-body exposure of Thelonious while walking was evaluated and the SAR calculated for 7 different postures (see Figure 4.11) using the well-known Finite Difference Time-Domain (FDTD) method [TH00]. The SAR, expressed in W/kg, quantifies the exposure to electromagnetic fields and represents the power absorbed per mass unit of tissue.

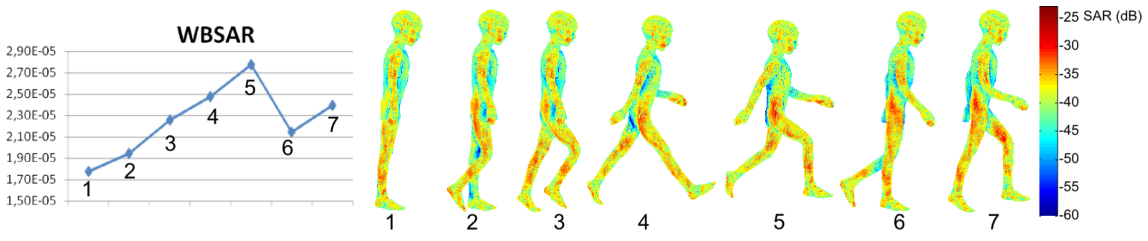


Figure 4.11: **Dosimetry simulation.**(Left) WBSAR calculated for E = 1 V/m. (Right) SAR distribution evolution while Thelonious is walking.

Radiofrequence exposure variations of a walking child

Figure 4.11 shows that the localization of the maximum SAR depends on the posture. For each model, the whole-body averaged SAR (WBSAR) was calculated in order to analyze the influence of the posture. The whole-body averaged SAR is equal to the whole power absorbed by the numerical model divided by the body weight. This was computed for an incident field E = 1 V/m (which corresponds to a Surface Power Density of 2.65 e-3 W/m2). Figure 4.11 plots the variations of the WBSAR with the posture of Thelonious. We can observe, as expected, that the WBSAR is proportional to the cross-section: from Posture 1 to Posture 5 the cross-section increases and so does the WBSAR. Then it decreases from Posture 5 to Posture 6 and increases again at Posture 7.

4.4 Conclusion

We extended our cage-based 3D image deformation pipeline VoxMorph, allowing to deform high-resolution whole-body anatomically correct models interactively. We made the pipeline robust by relaxing the constraint of enclosing cages and proposed a separation scheme. We improved the quality of the deformation by proposing a new error-metric for the refinement process of the motion adaptive structure. Finally, we solved the scalability issue by proposing an out-of-core deformation scheme. As a result, our system extended the scope of usable datasets, to deform in-core high-resolution voxel grids of over 260 million voxels within less than 4 minutes and to perform out-of-core deformations of the 3D images that do not fit in memory, e.g. 862 million voxels. The resulting deformed images are well suited for physical simulations, as well as variability and uncertainty studies.

It is currently intensively used by physicists for editing voxelized body models used in human-waves interactions studies and digital dosimetry simulations, known to be computed better on regular grids. This initial non-expert user feedback indicates that our system can be instrumental in other scenarios.

The rasterization step can be made more robust by performing the described process for more than one sample per voxel. Indeed, we can easily upsample each voxel at this stage and take the resulting dominant label with only little computation overhead since the rasterization process takes only a few milliseconds.

Our deformation propagation process can be used as an interactive tetrahedral mesh deformation method where the handles can be set as our system's constraints and the free parts of the models as the unknown variables.

An interesting avenue for further research is to refine the mesh in regions where rigidity could be enforced, in our case for bones. As a second step the deformation could be propagated to the remaining tetrahedra using our propagation method.

If physically-based deformation is mandatory (e.g., posing applications), an interesting venue for future work would be to combine our 3-scale scalable approach with recent, computationally demanding, material-aware methods [FGBP11].

Finally, to improve performances, we could use our remeshing method to meet the sizing field constraint and obtain smoother boundaries for the multi-material mesh cutting.

We mainly used our system for medical images but it can also be used to deform other kinds of volume data. For instance, it could be used to deform high-resolution tetrahedral meshes by embedding the input mesh into the TM.

Chapter 5

Volume data visualization

We complete our volume processing and editing pipeline with a visualization component since it is the remaining bottleneck of our deformation methods presented in the previous chapters. We focus on the interactive rendering of high-resolution voxel grids. Indeed, when the embedding tetrahedral mesh is deformed, the resulting voxel grid is obtained in milliseconds, but the update of the display can take minutes. This latency is due to the fact that streaming such a considerable amount of data to the Graphics Processing Unit (GPU) takes time. To avoid this costly process, we make use of the time-varying embedding Transfer Mesh (TM) - i.e. the deformation structure described in Chapter 3 - to render efficiently the voxel grid using barycentric interpolation (Figure 5.1). Our method takes advantage of the programmable graphics pipeline by combining a rasterization process, applied to the tetrahedral mesh on the Central Processing Unit (CPU) side, and a ray casting process performed on a 3D texture containing the voxel grid values which undergoes the so-defined deformation on the GPU side. Note, that the rendering time depends more on the screen resolution than on the 3D image resolution, and that the output visualized by the user describes faithfully the output of our deformation framework.

We start by presenting a quick overview of the interactive volume rendering methods using ray casting (Section 5.1) casting, followed by a description of our framework (Section 5.2) and demonstrate its validity on several whole-body voxel grids (Section 5.3).

5.1 Background

In this section, we present the properties our method must meet, and we give a quick overview of existing methods.

5.1.1 Requirements

The visualization process must be scalable, interactive and easy to use. Indeed, the high-resolution whole-body voxel grids to render are made of hundreds of millions of elements. Furthermore, the user needs a direct feedback to efficiently navigate through the volume, allowing for the typical try-and-test loop unavoidable in any modeling session. Finally, the

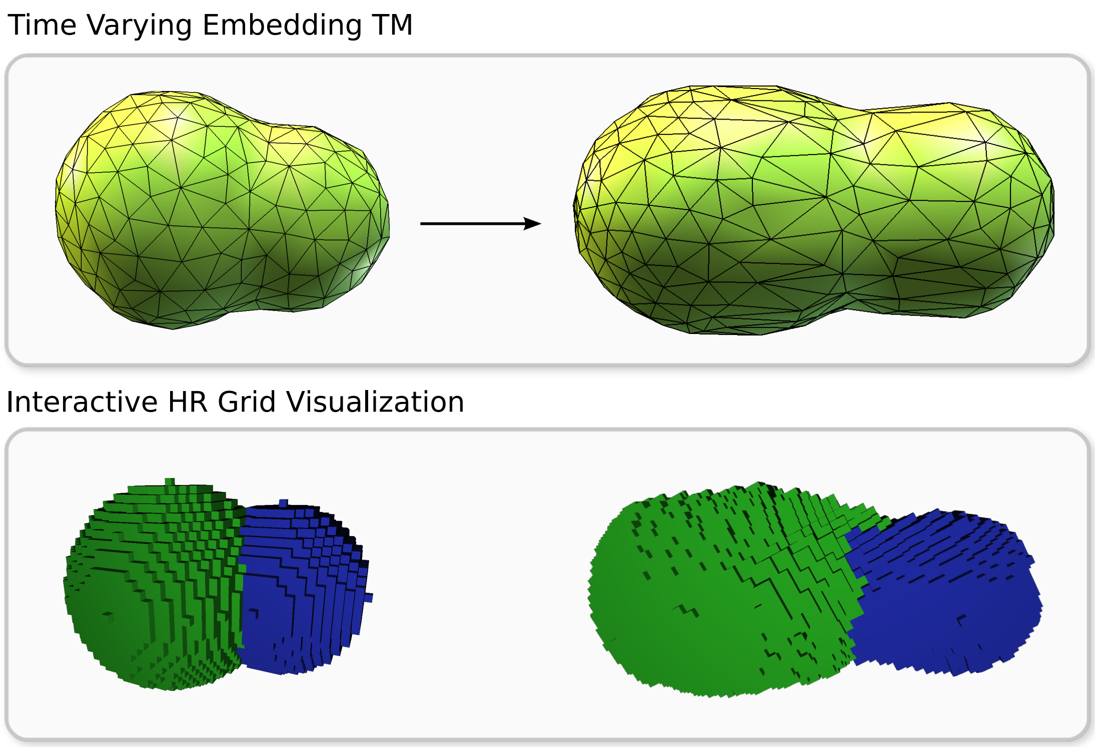


Figure 5.1: The first row displays the time-varying transfer mesh used to render the segmented voxel grids represented in the bottom row.

method must handle time-varying datasets interactively. Our goal is to represent the actual voxels of the grid, in order to visualize their structure to display information that is suitable to the users. As a matter of fact, the users of our deformation methods perform physical simulations on such segmented grids, hence, they need an easy-to-read information about the different subdomains. Displaying the voxels allows them to perceive the thickness of the material, the component connectivity or easily spot the outliers... Therefore, we do not want to perform a volume rendering process but render the voxels with their actual face normals.

The efficient display of volumetric datasets has been an active and wide field of research. We present here a non-exhaustive overview of interactive methods.

5.1.2 Volume data rendering

Given a discrete scalar field stored in a voxel grid, the main goal of most volume rendering methods is either to display an iso-surface representing the distribution of a given value, to render the complete volume as a semi-transparent medium or to combine both. To do so, a large number of methods have been proposed. We present here the principle of ray casting, along with GPU-based implementations.

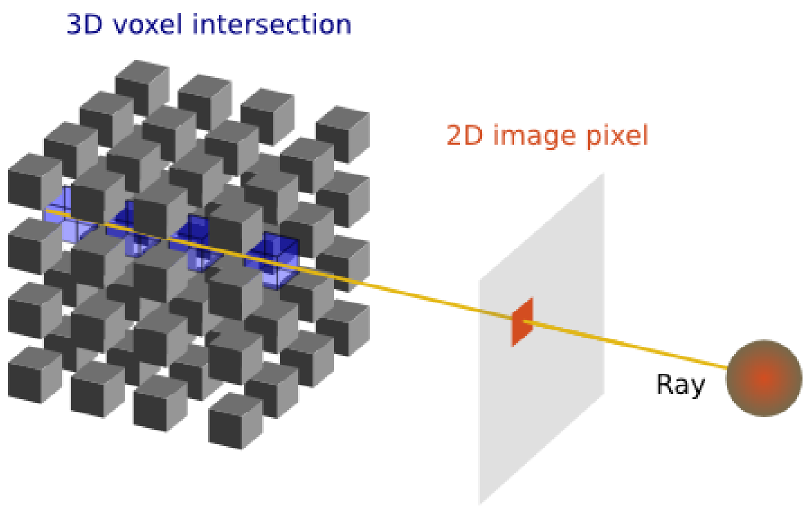


Figure 5.2: **Volume ray casting:** the contributions of the 3D voxel grid primitives (in blue) found along the ray are accumulated to compute the color of the pixel traversed by the ray (in red).

Ray casting

Ray tracing [App68] is a popular rendering technique used to simulate the physical behavior of light, and allows to generate high-quality realistic images. The core idea is to cast *primary* rays from the camera center through each pixel of the result image to compute the pixel's color using visibility and light contributions. *Secondary* rays are used to compute additional effects such as shadows or indirect lighting. Ray casting is based on a similar principle but uses a single primary ray per pixel. Ray casting methods allow to render the volumetric datasets as a semi-transparent medium using an approximation of the volume rendering integral; in that case, the contributions of primitives found along the ray are accumulated to compute the final pixel color (Figure 5.2). The first step is to find the input intersection between the ray and the grid bounding box, and then march along the ray in the grid until a termination condition is met or the point where the ray exits the bounding box is reached. In the case of iso-surface rendering, the ray is simply stopped when a voxel with the target iso-value is hit. For direct volume rendering, the ray traversal is stopped when the accumulated opacity has reached a given threshold. Numerous methods were proposed to perform this process interactively [GBKG04], and most of these propose a GPU-based ray casting [KW03a, Sch05] since the ray traversal for different pixels are independent and can be performed in parallel, as presented in the following section.

GPU-based implementations

We start by giving a quick description of the GPU rendering pipeline, illustrated in Figure 5.3, with its fixed stages (in orange) and its programmable stages (in green), before describing the principle of GPU-based ray casting. The 3D graphics application interacts with the hardware using an Application Programming Interface (Figure 5.3 blue) such as OpenGL. The primitives, typically triangles, are sent from the CPU to the GPU as

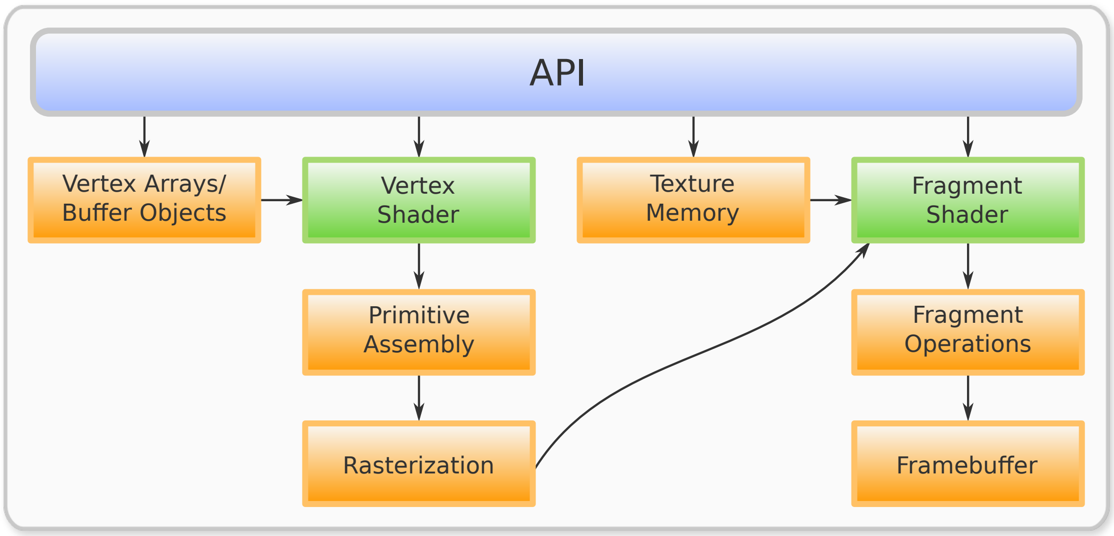


Figure 5.3: **Simplified GPU rendering pipeline:** the fixed stages are represented in orange and the programmable ones in green.

a stream of vertices along with their geometric information: positions, normals, colors and/or texture coordinates. Primitives are then converted into fragments during the rasterization stage and the vertices attributes are interpolated to obtain per fragment values. The fragment shader modifies fragments attributes and has access to texture data than can be applied at this stage. The color of each pixel is computed individually at this stage that is programmable. Therefore, the ray casting operations are performed in a single pass.

To give access to the input regular voxel grid to the fragment shader, it is stored directly on the GPU memory as a 3D texture, and the voxel values are fetched using points along the ray as texture coordinates. The rays are cast from the view point through the current pixel and the grid traversal starts at the bounding box entry point. This point can be computed directly in the fragment shader but it can lead to a significant overload [RGW+03]. To tackle this issue, some methods rasterize the front faces of the grid's bounding box, hence, getting the entry and exit point as fragment positions [KW03b]. The ray might traverse a considerable amount of background voxel, therefore, Kratz et al. proposed to rasterize an embedding surface representing more closely the input data, therefore reducing the processing time [KHF+06]. Note, that our method is inspired by this line of work.

For more information we refer the reader to recent books and surveys on GPU-based volume rendering [SCCB05, Wei06] and interactive ray tracing methods [MFS06].

As we have seen in this section, existing volume rendering methods allow to generate high-quality images representing the data as a semi-transparent medium or as iso-surfaces interactively. These methods display the volume or a smooth surface, where we want to represent the actual voxels of the grid in particular, we want the voxel structure to be clearly visible using the voxel's face normals for shading.

We propose a voxel grid rendering method based on a ray casting process performed inside each element of an embedding tetrahedral mesh that we describe in the following section.

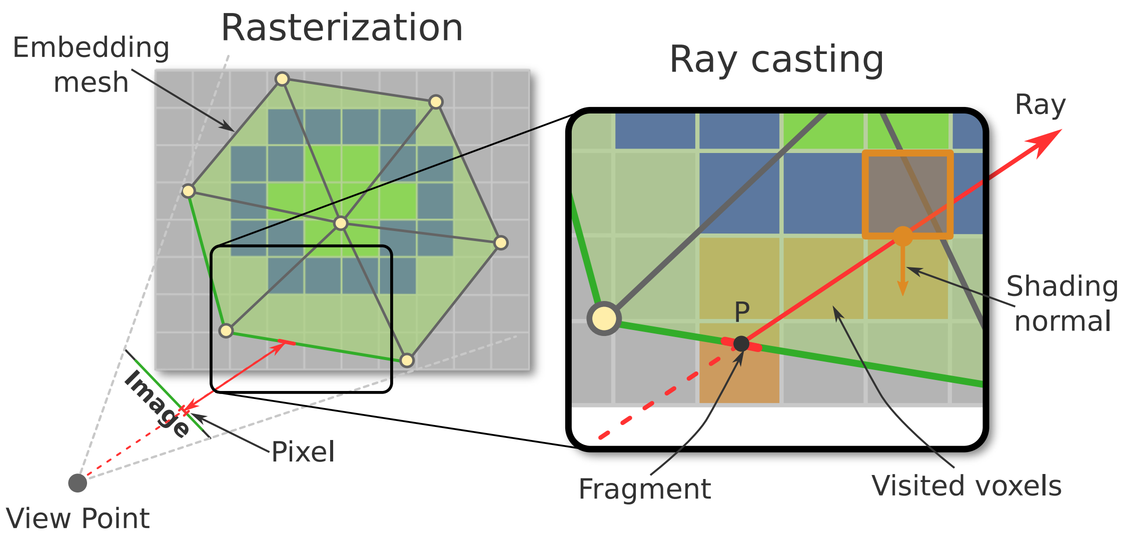


Figure 5.4: **Overview:** the input voxel grid data is embedded in the TM, which is rasterized, and the resulting fragments are used as starting points for a ray traversal in the grid. The first intersected voxel that does not belong to the background is rendered using its intersected face normal.

5.2 Voxel grid visualization

We propose a GPU-based rendering method for the visualization of high-resolution segmented voxel grids handling efficiently time-varying inputs. We load the voxel grid into a 3D texture in the GPU memory, and we use the embedding TM to guide a ray casting process performed in the fragment shader. For a given fragment, the pixel color is computed using the first intersection point with a non-background voxel - i.e. with a positive subdomain value - along with its intersected face normal. The grid values, contained in the aforementioned 3D texture, are accessed using a barycentric interpolation of the vertices' texture coordinates of the embedding tetrahedra. The TM vertices positions are updated on the CPU side and the deformed grid is displayed interactively without having to update the 3D texture. Finally, the user can navigate through the volume using axis-aligned cutting planes. Note, that we do not use the TM as an approximation structure for visualization, and that the resulting visualization represents faithfully the output of the deformation pipeline introduced in Chapter 3.

5.2.1 Overview

In this section, we present our interactive voxel grid rendering illustrated in Figure 5.4 and detailed in Algorithm 1.

On the CPU side, we start by computing the texture coordinates for the tetrahedra of TM, and storing the facet neighbors of each tetrahedron in a texture. Then, the TM tetrahedra are rendered using instancing and the programmable rendering pipeline (Figure 5.4 center).

Algorithm 1: Overview

```
Data: High-resolution voxel grid and a time-varying embedding tetrahedral mesh.
Initialization: Load the voxel grid in a 3D texture on the GPU and the tetrahedra
neighborhood information in a 2D texture
Rasterization: draw tetrahedra instances with corresponding coordinates
foreach fragment do
    trace a ray in the grid
    find initial intersection point
    while no intersection and still within the fragment's tetrahedron do
        find corresponding voxel
        compute the voxel center
        if center not in tetrahedron then
            find tetrahedron containing the voxel center
        end
        compute center texture coordinates
        fetch voxel color
        if intersected voxel color is black then
            find next intersection point with the grid
        else
            intersection found
            shade intersection position with the intersected voxel face normal
            return
        end
    end
end
```

On the GPU side, for each fragment, we cast a ray from its position, noted P, in the direction from the view point to P. The voxels inside the current instance that are intersected by the ray are visited until a non-background voxel is found or the ray exits the current tetrahedron (Figure 5.4 right). In order to get a consistent color per voxel in the deformed grid, we fetch the color for its center from the texture. Note, that this 3D point might not be located in the current tetrahedron. To cope with this issue, we find the tetrahedron that contains it by visiting iteratively the neighboring tetrahedron whose shared triangle faces the voxel center the most, resulting in an efficient depth-first traversal. If the color is not black (i.e. background) the shading of the fragment is computed using the intersection point and the associated intersected face normal.

5.2.2 CPU pre-processing

Some costly operations depend only on the tetrahedra geometry. We detail in the following which computations can be performed in a pre-processing step, and which ones require an update at each frame.

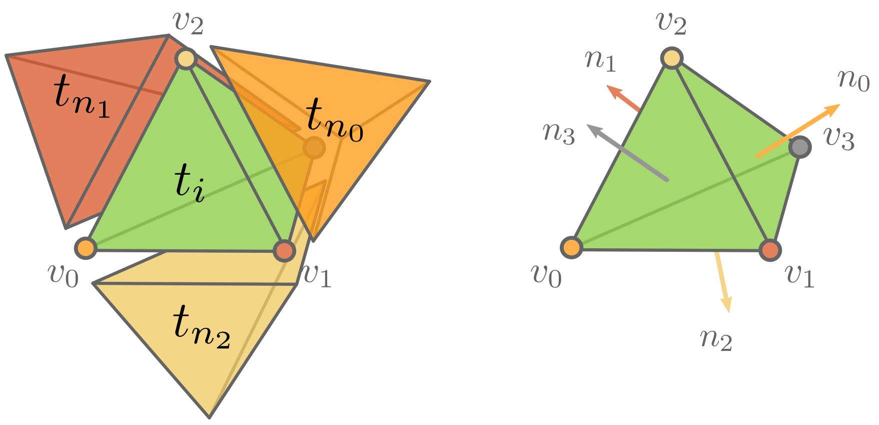


Figure 5.5: (Left) Neighborhood of the tetrahedron in green: the neighbor t_{n_k} is the tetrahedron opposite to the vertex v_k sharing a facet with t_i. (Right) Facet normals: n_k is the normal to the facet opposite to the vertex v_k.

Texture coordinates computation

We compute the texture coordinates of the tetrahedra vertices in order to access the voxel grid values contained in a 3D texture. The texture coordinates of the voxel grid's bounding box are $(0, 0, 0)$ for the bottom left corner and $(1., 1., 1.)$ for the top right corner. The TM is embedding the non-background voxels but is embedded inside this bounding box. Hence, the vertices texture coordinates are given by the vertices positions with reference to the bounding box.

This step is performed only once, at the creation of the initial (non-deformed) TM.

Pre-computation

The texture coordinates of a given point in the fragment shader is obtained by barycentric interpolation of the corresponding vertices of the tetrahedron containing this point. Hence, we need to compute efficiently the barycentric coordinates inside the tetrahedron on the GPU. Note that the barycentric coordinates are also used to determine if a point is located inside a tetrahedron. These can be computed using two factors α_k and δ_k, such that $\lambda_k(p) = \delta_k \alpha_k$, $k \in [0, 3]$, with:

$$\alpha_k = (p - v_{k+1}).n_k \tag{5.1}$$

and

$$\delta_k = \frac{1}{(v_k - v_{k+1}).n_k} \tag{5.2}$$

with p the current point position, v_k the vertices positions, n_k the normal of the facet opposite to v_k - i.e. that do not contain v_k - and $\lambda_k(p)$ the resulting coordinates of p. Part of this computation is constant over the tetrahedron - i.e. the δ_k depend on tetrahedron attributes (vertex positions and facet normals) only and not on the current point. Therefore, we pre-compute the four δ_k for each tetrahedron and store them in a 2D texture in order to reduce the number of operations to perform on the GPU. Additionally,

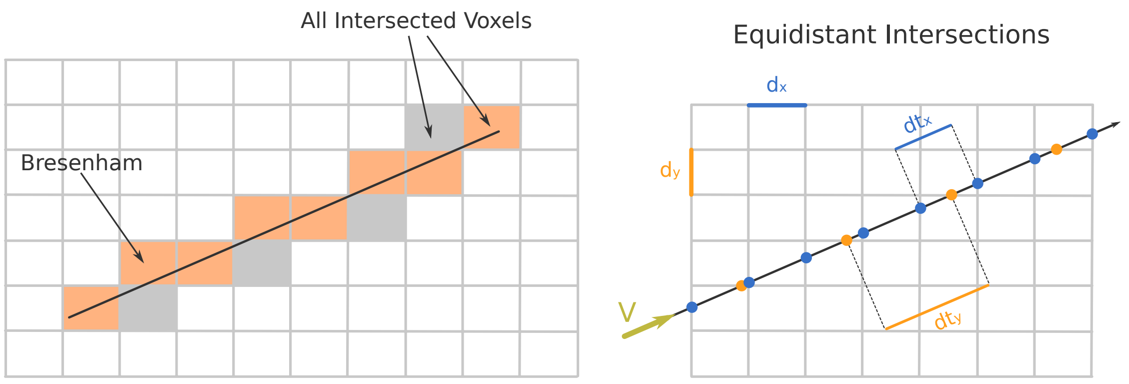


Figure 5.6: **Efficient ray casting:** (Left) to render the voxel grid accurately, all the voxels intersected by the ray must be processed. A straightforward rasterization of the line, such as Bresenham in orange, is not sufficient, hence, we use a 3D DDA method instead which stems from the fact that ray/voxel intersections are equidistant in a regular grid efficiency (Right).

for each tetrahedron, we store the (at most) four neighbors with which it shares a facet, and its facet normals (Figure 5.5). Once the pre-computations have been performed and the necessary information for the ray casting process have been loaded in the GPU memory, the tetrahedra triangles are rendered using the graphics pipeline.

5.2.3 GPU ray casting

We use the fragments resulting from the rasterization of the TM as a starting point for the ray traversal. This traversal is stopped when a valid intersection is found or the next intersection point is outside the current tetrahedron.

Overview

All the voxels intersected by the ray need to be processed, therefore a simple line rasterization method, such as Bresenham, is not enough. We use a 3D Digital Differential Analyzer (DDA) algorithm which takes advantage of the fact that the ray/voxel intersections are equidistant in a regular grid to efficiently compute the intersection point, its associated voxel and the intersected face normal (Figure 5.6). We express the ray as $r(t) = P + t.V$ with P the fragment position, V the normalized ray direction from the view point through P and $t \in \mathbb{R}$ the variable parameter used to walk on the ray.

Initialization

As a first step, we project the fragment position noted P in the grid in order to find the corresponding initial voxel, noted $g = (i, j, k)$. We note t_{min} the current minimum t, which is initialized at 0 since we start casting the ray from P, and compute the t_{next_i} with $i \in \{x, y, z\}$ corresponding to the next intersections in each axis-aligned direction

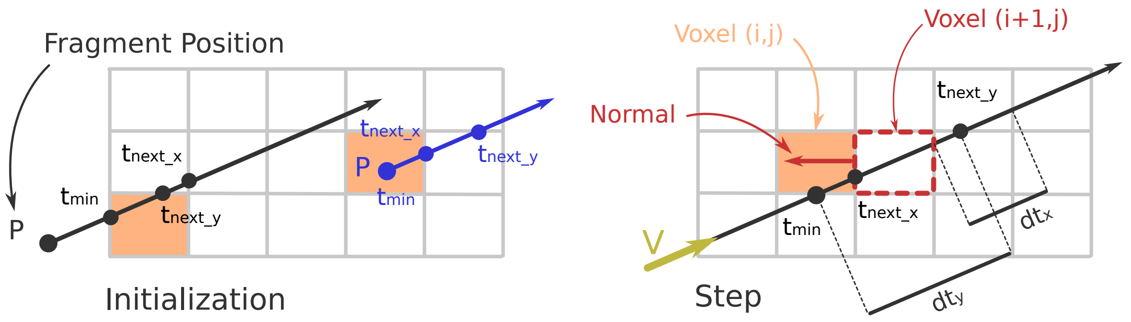


Figure 5.7: (Left) **Initialization:** the ray origin is outside the grid (black) or in the grid (blue) which is always true in our case. t_{min} is set to 0 in order to start the traversal at P. The corresponding initial voxel is in orange and the t_{next_i} are initialized to the first intersection point in each axis-aligned direction. (Right) **Ray traversal step:** the next voxel is explored using the minimum t_{next_i} value. We store the corresponding face normal for the shading, and the voxel index to fetch its color.

(Figure 5.7 left). We also compute the ray steps dt_i for each axis-aligned direction i and set them to $dt_i = d_i/V_i$, with d_i the voxel size, and V_i the coordinates of the ray direction. We also compute the voxel steps dg_i allowing to find the next voxel depending on the ray direction, such that $dg_i = 1$ if V_i is positive and $dg_i = -1$ otherwise.

Algorithm 2: Find next intersection

```
Input: ray $r$, ray steps $dt_i$, voxel steps $dg_i$.
Initialization: current $t_{min}$, $t_{next_x}$, $t_{next_y}$, $t_{next_z}$ and voxel $g = (i, j, k)$
if $t_{next_x} < t_{next_y}$ and $t_{next_x} < t_{next_z}$ then
    $i \leftarrow i + dg_x$
    $t_{min} \leftarrow t_{next_x}$
    $t_{next_x} \leftarrow t_{next_x} + dt_x$
    $n \leftarrow (-dg_x, 0, 0)$
end
else if $t_{next_y} < t_{next_x}$ and $t_{next_y} < t_{next_z}$ then
    $j \leftarrow j + dg_y$
    $t_{min} \leftarrow t_{next_y}$
    $t_{next_y} \leftarrow t_{next_y} + dt_y$
    $n \leftarrow (0, -dg_y, 0)$
end
else
    $k \leftarrow k + dg_z$
    $t_{min} \leftarrow t_{next_z}$
    $t_{next_z} \leftarrow t_{next_z} + dt_z$
    $n \leftarrow (0, 0, -dg_z)$
end
```

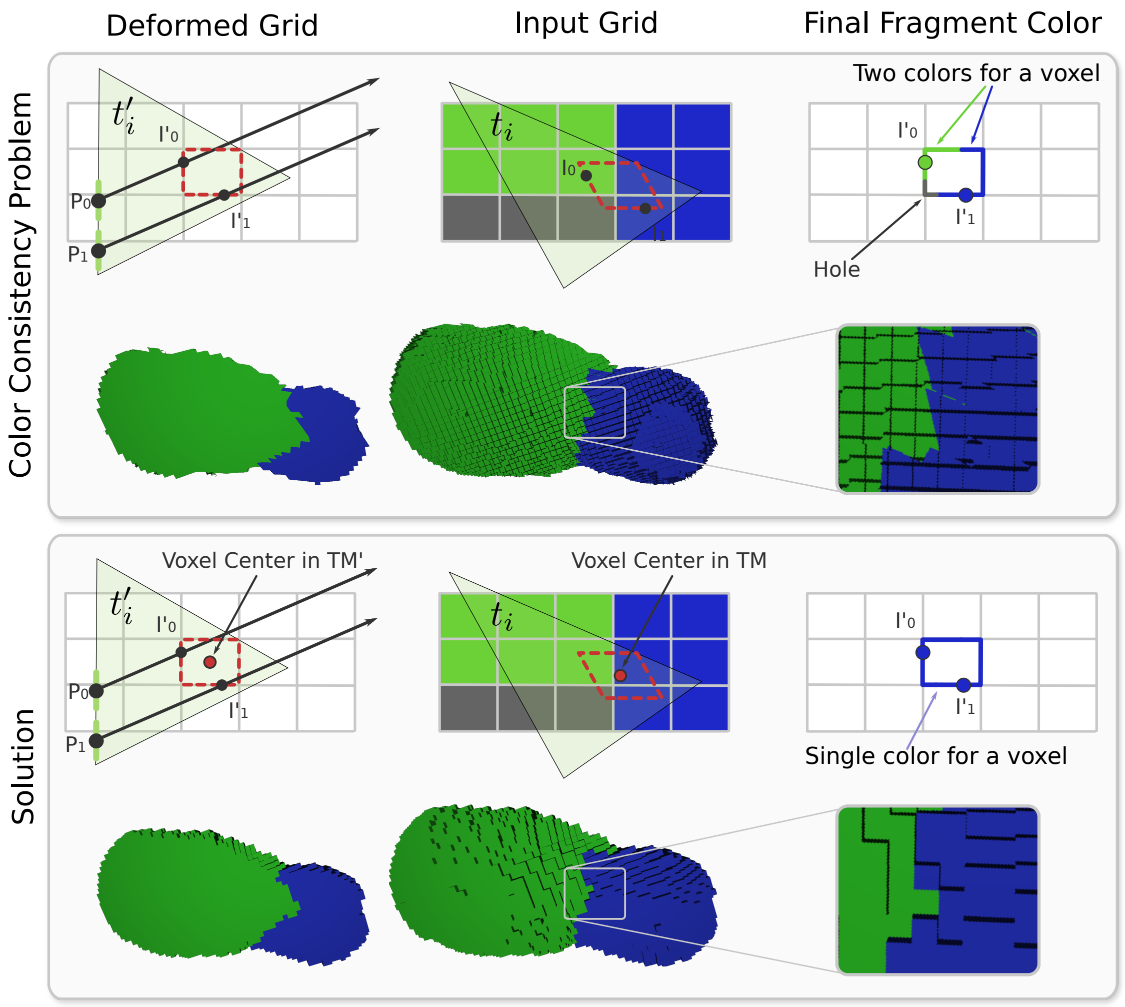


Figure 5.8: **Voxel value:** (Top row) color consistency problem: different parts of a single voxel of the deformed grid are projected in different voxels in the initial grid, resulting in multiple colors per voxel and holes. (Bottom row) This problem is solved by computing the color for the voxel center only and apply it to the entire voxel - i.e. to all the intersection points with this voxel.

Ray traversal

We iterate on the ray by setting t_{min} to the smallest t_{next_i} - i.e. taking the next closest intersection point -, we store the corresponding face normal n and update the voxel value accordingly. This process is illustrated on the right of Figure 5.7 and detailed in Algorithm 2. Then, if the intersection point lies inside the current tetrahedron, we fetch the subdomain index associated with its corresponding voxel center in a 3D texture. If the voxel does not belong to the background, we use the current position and the face normal, noted n, to compute the shading and get the pixel color.

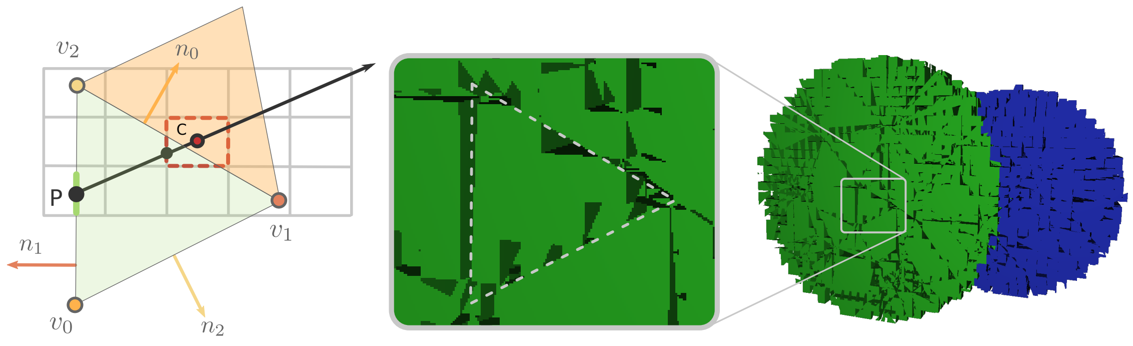


Figure 5.9: The intersection point is not located in the same tetrahedron as the corresponding voxel center, resulting in cut voxels. A depth-first search is performed using the point location w.r.t. the planes defined by the tetrahedra's facets, by searching the neighbor associated with the facet which normal points the most to the voxel center.

Voxel value

To get the voxel value corresponding to a point, we compute its 3D texture coordinates using a barycentric interpolation of the vertex texture coordinates of the tetrahedron in which it is located. We perform this process on the voxel center in order to get a consistent color for each resulting voxel. Indeed, for the deformed grid different parts of a single voxel might be projected in different voxels in the initial initial grid resulting in voxels displayed with different colors and even holes (Figure 5.8).

Neighborhood search

The voxel center might be outside the current tetrahedron containing the intersection point (Figure 5.9). Therefore, we perform an efficient depth-first search using its local neighborhood information in order to find the tetrahedron in which the point is located. For the current tetrahedron, we compute the $\alpha_k = (p - v_{k+1}).n_k$ for each of its vertices and explore the neighbor k for which α_k is maximum. This value allows to determine on which side of the planes, defined by the tetrahedron facets, the point is located, hence, which neighbor is more likely to contain it. We iterate until the tetrahedron containing the point has been located or a given number of iterations has been performed. Note, that the α_k are the values used to compute the barycentric coordinates.

Cutting planes

In order to navigate through the volume, four cutting planes, three axis-aligned and one following a frame, can be manipulated interactively. The voxels that have their center on the averted side of one of the planes are treated as background voxel during the ray traversal that continues until a visible voxel is found.

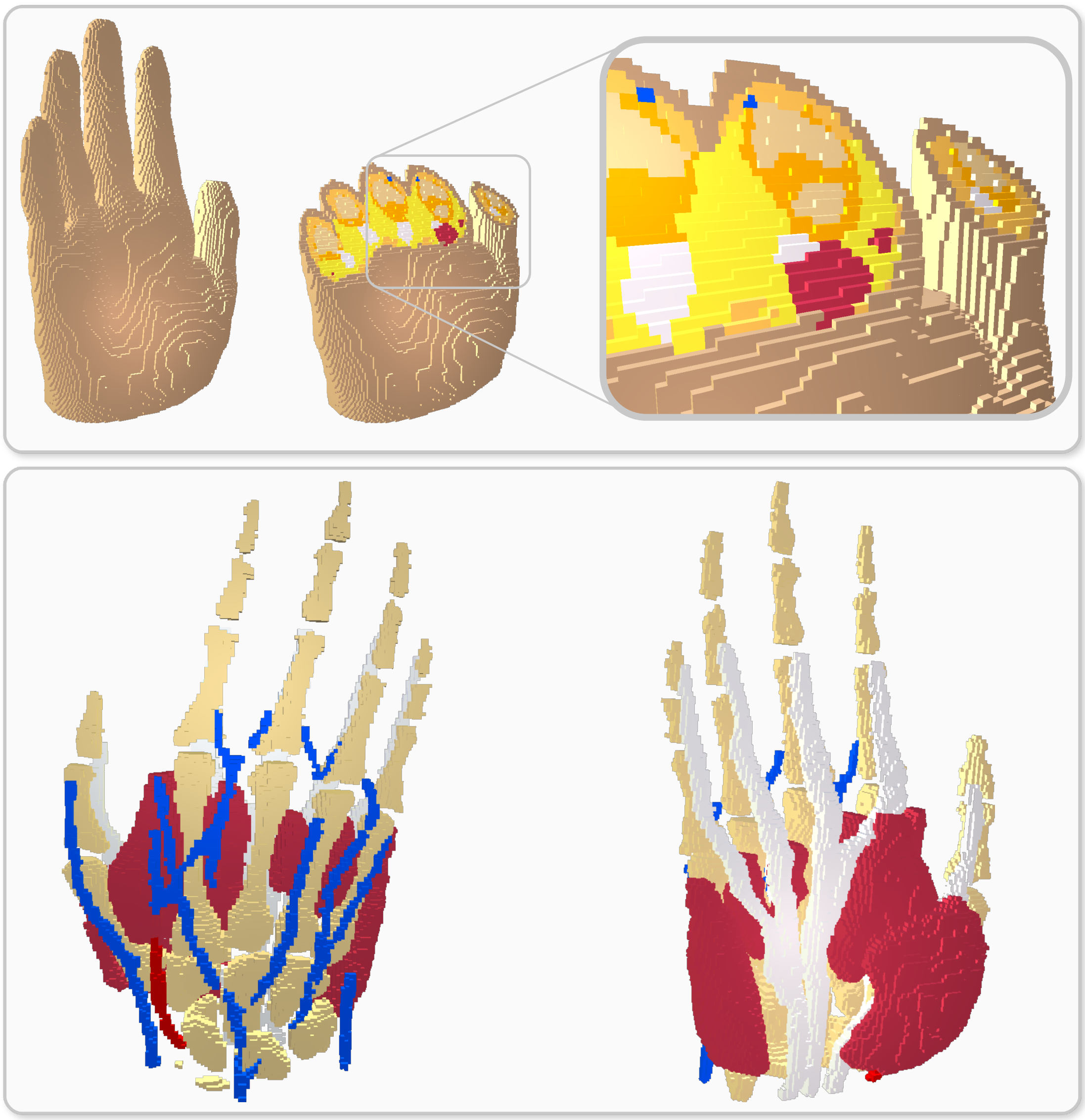


Figure 5.10: **Hand model.** Interactive renderings of a voxel grid representing a hand.

5.3 Results

We implemented our system in C++ using OpenGL for rendering and OpenGL Shading Language for our shaders. Performances were measured on an Intel Core2 Quad at 3 GHz with a nVidia GTX 480 GPU.

Figure 5.10 illustrates the voxel rendering performed interactively and the use of the non-axis aligned cutting plane. Figures 5.11, 5.12 and 5.13 illustrate the results obtained interactively on whole-body voxel grids with our method. The tetrahedral meshes used to render the voxel grids, displayed in the first rows, result from deformations performed using our method described in the previous chapters. The neighborhood information and texture coordinates are computed and stored for the initial mesh only. To display the deformed grid, the mesh vertices positions are changed and the pre-computed parts of the barycentric coordinates are updated accordingly.

Table 5.1 summarizes the performances of our system on the models presented in this section. We can see that the largest model, i.e. Visible Human, is displayed interactively, although no hierarchical data structure is used.

Model	# voxels	# TM tetrahedra	FPS
Spheres	125 000	2 057	53
Hand	1.41M	8 510	70
Thelonious	122M	71 361	45
Louis	260M	144 371	22
Visible Human	374M	145 359	12

Table 5.1: **Performance table**: with the number of voxels, the transfer mesh tetrahedra count and FPS the framerate during interaction.

5.4 Conclusion

Our method successfully displays high-resolution voxel grids interactively using a time-varying embedding tetrahedral mesh. Hence, when plugged in our deformation method, it provides an interactive visualization of the deformed high-resolution grid. The proposed method is well suited to our context since it allows to render the actual voxels. Additionally, the user can navigate through the volume interactively using cutting planes.

We can cast additional rays to add visual effects such as indirect lighting, which gives further information about the volume data, or render given subdomains as transparent medium. Furthermore, to improve the perception of the volume efficiently, screen-space effects can be added, such as ambient occlusion.

An alternative method would be to render the boundary facets of the mesh only, and cast the ray in the grid while using the depth-first search to get the traversed tetrahedra.

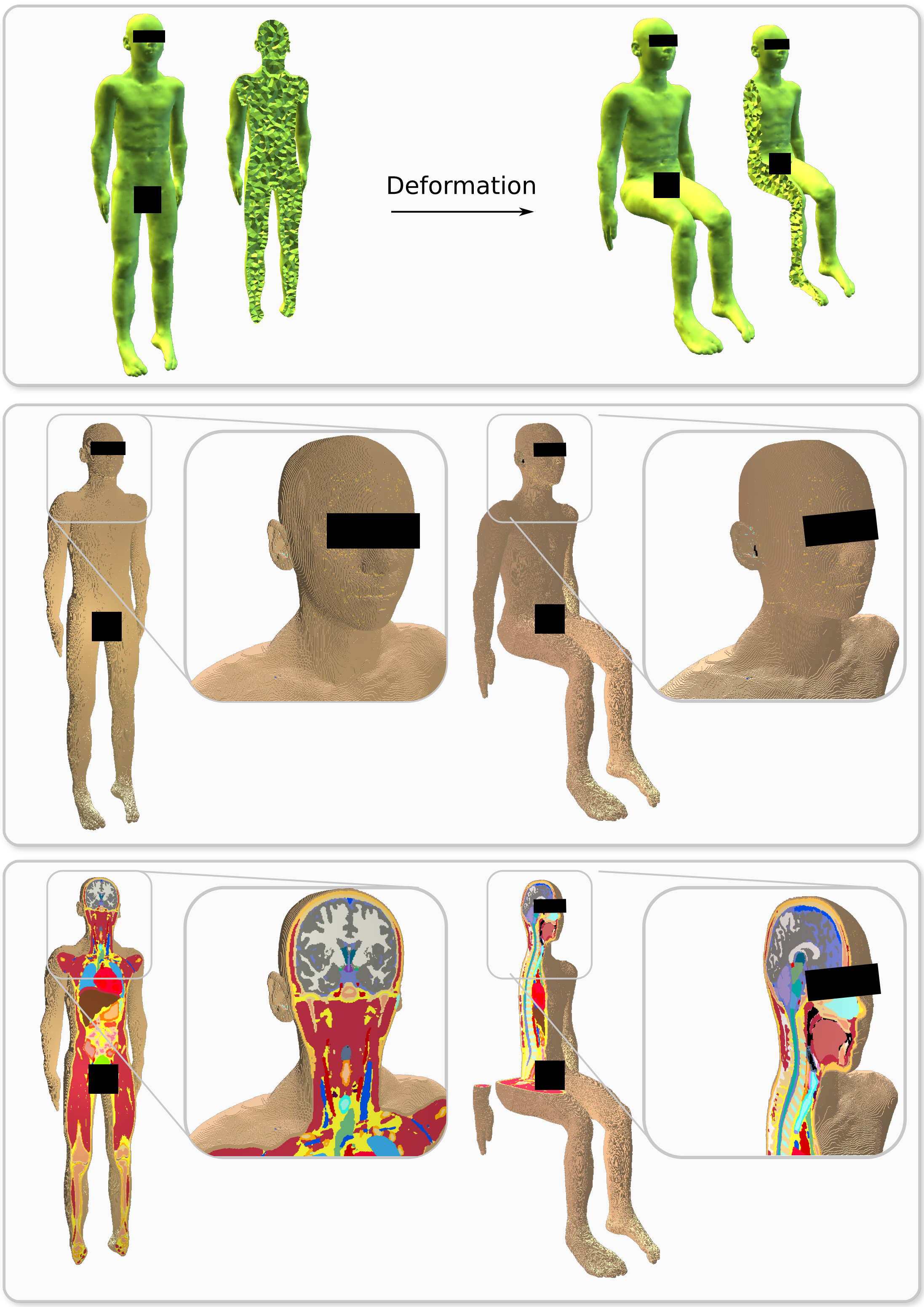


Figure 5.11: **Louis model.** Interactive renderings of a whole-body voxel grid. The first row displays the time-varying embedding tetrahedral mesh used to perform the renderings of the bottom rows.

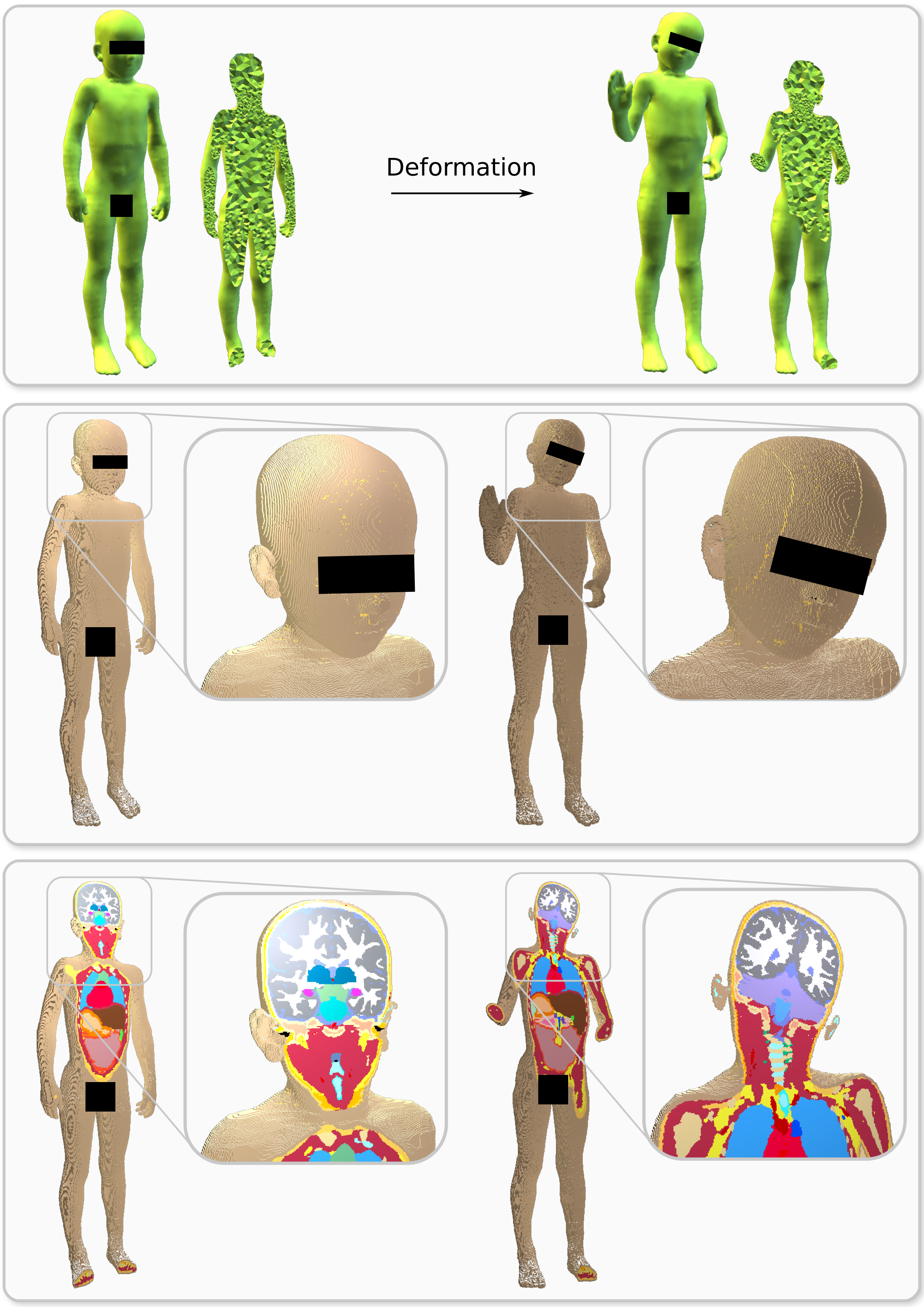


Figure 5.12: **Thelonious model.** Interactive renderings of a whole-body voxel grid.

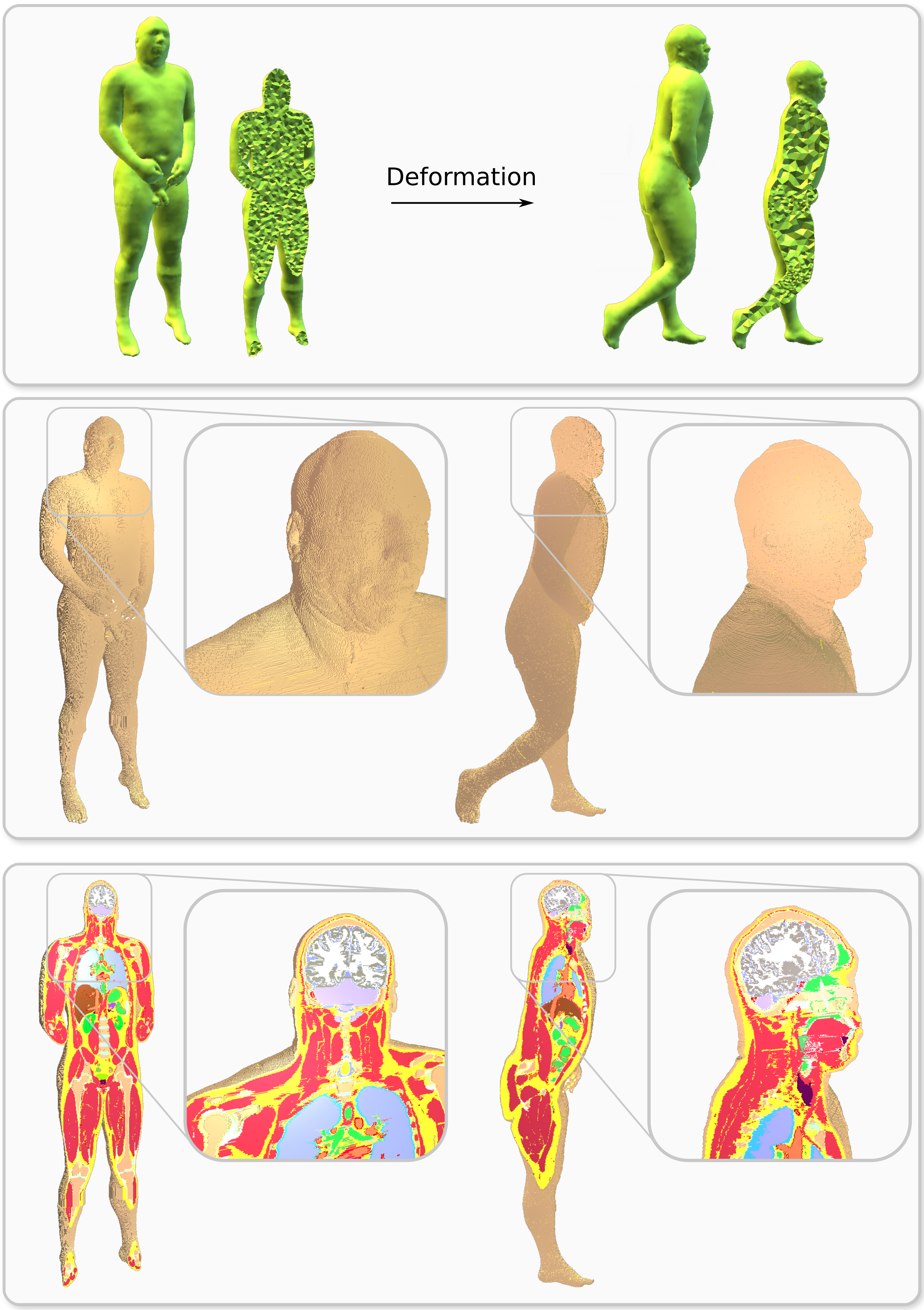


Figure 5.13: **Visible Human model.** Interactive renderings of a large whole-body voxel grid.

CHAPTER 6

CONCLUSION

The topic of this thesis was to model, interact with and visualize interactively volume datasets, while providing high-quality results that are suitable for physical simulations. The two main representations used to handle such datasets, namely voxel grids and tetrahedral meshes, are usually opposed in computer graphics to perform such task. In this work, we have shown that using them together provides versatile methods allowing to process, modify and visualize volume models efficiently. Essential features of our systems stem from this combination: preservation of the input data, adaptivity (computing power, memory footprint) and interactivity even on large high-resolution models.

Modeling Starting from a voxel grid, a significant number of simulation frameworks use a multi-domain tetrahedral mesh as an underlying representation for scalability and interactivity issues, making a meshing pre-processing step necessary. The generated meshes must meet sizing and quality constraints while representing the subdomain interfaces accurately. We proposed M-MAd: an iterative remeshing approach for tetrahedral meshes compatible with segmented voxel grids, based on local operations, simultaneously refining or simplifying the created tetrahedra. Our algorithm generates high-quality adaptive, isotropic volume meshes, preserving boundaries between domains and features detected on the outer boundary of the domain as well as user-defined features. To do so, we model the input data as a *point-sampled cell complex* capturing the boundaries between the different subdomains and perform feature-aware local remeshing operations ordered with respect to a quality metric. In order to generate fine elements only where needed, we control the local remeshing process with a *sizing field* tailoring the tetrahedra's size in a spatially-varying manner. Additionally, our method can be used as a remesher to adjust the elements size and/or quality of an existing multi-material tetrahedral mesh in order to meet the constraints imposed by the simulations frameworks. As a result, our (re)mesher generates high-quality adaptive meshes and is free from any computation of a Delaunay triangulation/Voronoï diagram. In contrast to such approaches, our method has lower memory footprint and is significantly easier to implement, and it allows us to navigate bottom-up and top-down between different resolutions easily by re-using the current mesh instead of re-computing it from scratch.

Deformation The medical images are acquired in a certain state/posture; while in a number of application scenarios, the physical simulations need to be performed on more natural poses (sitting down, walking, using a cell phone...). Because it is impossible to capture the whole variety of all potential configurations, a deformation pre-process is often necessary. The deformation must be scalable, generate high-quality results, in order to provide models that are suitable for simulations, while preserving the interactivity constraint. Indeed, the user needs an interactive feedback to reach the aimed physically plausible posture. In many cases the voxel grids are meshed to perform deformations meeting these constraints. In this second part of the thesis, we showed that an adaptive tetrahedral mesh, the so-called Transfer Mesh, can be used as a deformation structure to interact with the voxel grid containing the information to preserve. As opposed to the meshes described in the previous part, the tetrahedral mesh does not represent the input model features but is an approximation model of the deformation. In our 3-scale interactive freeform deformation pipeline for voxel grids, this structure is deformed using a cage for high-level deformation control. Our system combines a high-quality non-linear deformation at coarse scale - i.e. applied to the cage -, a quasi-conformal space deformation at mid-scale and a new adaptive local linear deformation transfer structure - i.e. the Transfer Mesh - at fine-scale. The two first scales are applied interactively on a visualization envelope, while the complete full resolution deformation is computed as a post-process after the interactive session, resulting in a high-resolution voxel grid containing the deformed model. All the described representations are generated with our system. To cope with the fact that generating a fully-embedding cage is not always possible, we proposed a tetrahedral variational deformation method to extend cage-based deformations to the outside of the cage model, thus allowing the use of inset cages. Furthermore, our system extends the scope of usable datasets by performing necessary topology changes and handling out-of-core datasets. Finally, we validated our pipeline by performing digital dosimetry analysis for radio-wave effects studies on high-resolution segmented 3D medical images posed with our approach.

Visualization To explore the volumetric dataset and evaluate the deformation result, the user needs an efficient visualization of the voxel grid. We propose to take advantage of the time-varying embedding transfer mesh - i.e. the mid-scale structure used in our deformation pipeline - to render voxel grids efficiently, using barycentric interpolation. Indeed, the remaining bottleneck of our deformation pipeline is the visualization of the resulting voxel grid since streaming the full resolution dataset to the GPU takes a considerable amount of time. We load the initial grid into a 3D texture in GPU memory only once and associate texture coordinates to the tetrahedral mesh to access the voxel labels. Then, this mesh is rasterized, and we perform a ray casting process starting from the resulting fragments into the deformed grid.

Altogether, our contributions tackle the different problematics arising from our applicative context. From these complementary technical contributions the idea of a combined voxel/tetrahedral volume representation clearly arises as a versatile hybrid shape model suitable for a number of major graphics applications in the context of medical data. Such a combined representation can be further developed to tackle other problems efficiently.

Chapter 7

Perspectives

In this thesis, we worked on different topics stemming from the interaction with high-resolution volume datasets. Beyond the proposed technical contributions, a number of issues arised in the course of this research work.

During the modeling process, we noticed that the exact preservation of the topology of the input mesh or voxel grid is not always desirable. Indeed, the input dataset might be subject to acquisition or meshing noise, resulting in isolated small groups of tetrahedra. On the other hand, such a group of tetrahedra might represent a small subdomain that is relevant to the user. An interesting avenue of research is to identify these components during the remeshing process, let the user decide interactively if the signal needs to be preserved and, if so, adapt the target edge length accordingly. Performing this operation as a pre-process only is not enough, since depending on the overall mesh resolution the user might want to release the topology constraints. Indeed, when the subdomain becomes significantly smaller than its surrounding elements the strict preservation of the topology may result in poorly shaped elements. An additional feature of the system would be to restore the subdomain when the current resolution allows it. In our system, we can recover from low resolutions using the MLS representation by ensuring that at least one tetrahedron of each connected component remains.

While our operations ordering provides high-quality results, one might wonder about a statistical approach that would allows to reach a linear time complexity. Indeed, the different tetrahedra configurations can be listed, and a statistical analysis could be performed to determine which sequence of operations (split, collapse, smooth or flip) is the most likely to improve the quality of the element. We believe that studying sequences of operations would give better results than the behavior of isolated ones, and eventually allow to avoid local minima of the energy we minimize.

The automatic creation of deformation cages remains an open problem. More than often, the cage is generated using a simplified version of an embedding high-resolution surface mesh, which is prone to generate self-intersecting or non-embedding cages even with an adaptive offset. While we tackle this issue by releasing the embedding constraint, this is not always an option. The main difficulties stem from the input model posture where parts of the model are spatially close but have a large geodesic distance. An interesting avenue of research is to detect these configurations automatically and generate a posture for which the cage creation process is simplified. Applying this process in an interactive and hierarchic fashion would allow for a wide range of applications. For example, such a system would simplify the typical texturing process by unfolding the model in regions of interest and allow the artist to paint directly the pattern on the mesh, instead of a set of distorted 2D image to be mapped.

Last, in term of medical modeling application, the naive simplification of an embedding surface mesh during the cage creation process does not account for the fact that some regions will undergo stronger deformations, i. e. near articulations in this context. The generated cage may be difficult to use as an interaction primitive since the vertex positions are not optimized to allow the desired deformations. An interesting avenue of research is to identify automatically these articulations on the medical image and use this information to guide the simplification and allocate more vertices near the regions of interest. Alternatively, a set of optimal cages adapted to the different human morphologies (women, men, children...), with the necessary control vertices, could be created manually optionally with a skeleton controlling its deformation, casting the cage generation problem to a shape retrieval and retargeting problem.

Appendix A

Publications

- **Robust and Scalable Interactive Freeform Modeling of High Definition Medical Images.**
Noura Faraj, Jean-Marc Thiery, Isabelle Bloch, Nadege Varsier, Joe Wiart and Tamy Boubekeur
MICCAI Workshop on Mesh Processing in Medical Image Analysis 2012 - Springer Lecture Notes in Computer Science.

- **VoxMorph: 3-Scale Freeform Deformation of Large Voxel Grids.**
Noura Faraj, Jean-Marc Thiery and Tamy Boubekeur
Shape Modeling International 2012 - Computer & Graphics Journal 2012.

- **Parameterizing Animated Lines for Stylized Rendering.**
Bert Buchholz, Tamy Boubekeur, Sylvain Paris, Noura Faraj and Elmar Eisemann
ACM SIGGRAPH 2011 - Talk Program.

- **Spatio-temporal Analysis for Parameterizing Animated Lines.**
Bert Buchholz, Noura Faraj, Sylvain Paris, Elmar Eisemann and Tamy Boubekeur
Non-Photorealistic Animation and Rendering 2011.

- **VoxMorph.**
Noura Faraj, Jean-Marc Thiery et Tamy Boubekeur
Journées de l'Association Française d'Informatique Graphique 2011.

- **Fetus RF exposure analysis. Preliminary results based on three realistic 3D digital models.**
Joe Wiart, Soichi Watanabe, Isabelle Bloch, Jérémie Anquez, Juan Pablo De La Plata Alcade, Elsa Angelini, Tamy Boubekeur, Noura Faraj, Christian Person Yenny Constanza Pinto Ballestreros, Nadège Varsier, Thierry Kientega , Marjorie Jala, Hadjem Abdelhamid, Azzedine Gati, Man-Faï Wong, Emmanuelle Conil, Bruno Sudret, Tomoaki Nagaoka, Wake Kanako, Osamu Fujiwara, Akimasa Hirata, Jianqing Wang, Saito Kazuyuki, Masaharu Takahashi and Koichi Ito
33rd Annual Meeting of the Bioelectromagnetics Society 2011.

Appendix B

Aknowledgements

We take this opportunity to thank our partners and the following organizations for providing us with segmented volume datasets:

- IT'IS Foundation for the Virtual Population project,
- U.S. National Library of Medicine for the Visible Human project,
- AimShape Network.

Additionally, we thank Professor Catherine Adamsbaum for her collaboration for the acquisition of whole-body MRI of children.

We are especially grateful to the developers of CGAL that put together a constantly evolving library that we have extensively used for this work.

This work was funded by ANR KidPocket and ANR FETUS projects.

Bibliography

[AA04] Anders Adamson and Marc Alexa, *Approximating bounded, non-orientable surfaces from points*, Shape Modeling International, 2004, pp. 243–252.

[AA06] Anders Adamson and Marc Alexa, *Point-sampled cell complexes*, ACM Transactions on Graphics (TOG), vol. 25, 2006, pp. 671–680.

[AA09] Marc Alexa and Anders Adamson, *Interpolatory point set surfaces-convexity and hermite data*, ACM Transactions on Graphics (TOG) **28** (2009), 20:1–20:10.

[ABCO$^+$01] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva, *Point set surfaces*, Proceedings of the conference on Visualization '01, VIS '01, 2001, pp. 21–28.

[ACOL00] Marc Alexa, Daniel Cohen-Or, and David Levin, *As-rigid-as-possible shape interpolation*, SIGGRAPH, 2000, pp. 157–164.

[ACSYD05] Pierre Alliez, David Cohen-Steiner, Mariette Yvinec, and Mathieu Desbrun, *Variational tetrahedral meshing*, ACM Transactions on Graphics (TOG) (2005), 617–625.

[App68] Arthur Appel, *Some techniques for shading machine renderings of solids*, Proceedings of the April 30–May 2, 1968, spring joint computer conference, ACM, 1968, pp. 37–45.

[BCWG09] Mirela Ben-Chen, Ofir Weber, and Craig Gotsman, *Variational harmonic maps for space deformation*, ACM Transactions on Graphics **28** (2009), 34:1–34:11.

[BHU10] D. Burkhart, B. Hamann, and G. Umlauf, *Adaptive and feature-preserving subdivision for high-quality tetrahedral meshes*, Computer Graphics Forum **29** (2010), 117–127.

[BHZN10] Péter Borosán, Reid Howard, Shaoting Zhang, and Andrew Neal, *Hybrid mesh editing*, Eurographics, 2010.

[BK04] Mario Botsch and Leif Kobbelt, *A remeshing approach to multiresolution modeling*, Symposium on Geometry Processing, 2004, pp. 185–192.

[BLW13] Jonathan Bronson, Joshua Levine, and Ross Whitaker, *Lattice cleaving: Conforming tetrahedral meshes of multimaterial domains with bounded quality*, 21st International Meshing Roundtable (Xiangmin Jiao and Jean-Christophe Weill, eds.), 2013, pp. 191–209.

[BO05a] Jean-Daniel Boissonnat and Steve Oudot, *Provably good sampling and meshing of surfaces*, Graphical Models **67** (2005), 405 – 451.

[BO05b] Jean-Daniel Boissonnat and Steve Oudot, *Provably good sampling and meshing of surfaces*, Graphical Models **67** (2005), 405–451.

[Bot05] Mario Botsch, *High quality surface generation and efficient multiresolution editing based on triangle meshes*, Ph.D. thesis, 2005, pp. 1–174.

[BPGK06] Mario Botsch, Mark Pauly, Markus Gross, and Leif Kobbelt, *Primo: coupled prisms for intuitive surface modeling*, Symposium on Geometry Processing, 2006, pp. 11–20.

[BPWG07] Mario Botsch, Mark Pauly, Martin Wicke, and Markus Gross, *Adaptive space deformations based on rigid cells*, Computer Graphics Forum **26** (2007), 339–347.

[BS08] Mario Botsch and Olga Sorkine, *On linear variational surface deformation methods*, IEEE TVCG **14** (2008), 213–230.

[BYB09] Dobrina Boltcheva, Mariette Yvinec, and Jean-Daniel Boissonnat, *Feature preserving Delaunay mesh generation from 3d multi-material images*, Symposium on Geometry Processing, 2009, pp. 1455–1464.

[CA97] Patricia Crossno and Edward Angel, *Isosurface extraction using particle systems*, IEEE Visualization, 1997, pp. 495–498.

[CD03] Siu-Wing Cheng and Tamal K. Dey, *Quality meshing with weighted Delaunay refinement*, SIAM Journal on Computing **33** (2003), 69–93.

[CDE$^+$00] Siu-Wing Cheng, Tamal K. Dey, Herbert Edelsbrunner, Michael A. Facello, and Shang-Hua Teng, *Silver exudation*, J. ACM **47** (2000), 883–904.

[CDHR08] Yanqing Chen, Timothy A Davis, William W Hager, and Sivasankaran Rajamanickam, *Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate*, ACM Transactions on Mathematical Software **35** (2008), 1–14.

[CDL07] Siu-Wing Cheng, Tamal K. Dey, and A. Levine, *A practical Delaunay meshing algorithm for a large class of domains*, Proceedings of the 16th International Meshing Roundtable, 2007, pp. 477–494.

[CDM04] B. Cutler, J. Dorsey, and L. McMillan, *Simplification and improvement of tetrahedral models for simulation*, Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing, 2004, pp. 93–102.

[CDR07] Siu-wing Cheng, Tamal K. Dey, and Edgar A. Ramos, *Delaunay refinement for piecewise smooth complexes*, 18th Annual ACM-SIAM Symposium Discrete Algorithms, 2007, pp. 1096–1105.

[CGA] CGAL, CGAL *Computational Geometry Algorithms Library*, http://www.cgal.org.

[Che93] L. Paul Chew, *Guaranteed-quality mesh generation for curved surfaces*, Ninth annual symposium on Computational geometry, 1993, pp. 274–280.

[Che04] Long Chen, *Mesh smoothing schemes based on optimal Delaunay triangulations*, 13th International Meshing Roundtable, 2004, pp. 109–120.

[CKH$^+$10] Andreas Christ, Wolfgang Kainz, Eckhart G Hahn, Katharina Honegger, Marcel Zefferer, Esra Neufeld, Wolfgang Rascher, Rolf Janka, Werner Bautz, Ji Chen, Berthold Kiefer, Peter Schmitt, Hans-Peter Hollenbach, Jianxiang Shen, Michael Oberle, Dominik Szczerba, Anthony Kam, Joshua W Guag, and Niels Kuster, *The virtual family- development of surface-based anatomical models of two adults and two children for dosimetric simulations*, Physics in Medicine and Biology **55** (2010), N23.

[Con10] GSL Project Contributors, *GSL - GNU scientific library - GNU project - free software foundation (FSF)*, http://www.gnu.org/software/gsl/, 2010.

[CPF$^+$95] Jean Pierre Cocquerez, Sylvie Philipp-Foliguet, et al., *Analyse d'images: filtrage et segmentation*.

[CS10] C-Safe, *Fire simulation, C-Safe: Center for the simulation of accidental fires and explosions*, 2010, http://www.csafe.utah.edu/.

[CSAD04] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun, *Variational shape approximation*, ACM Transactions on Graphics (TOG) **23** (2004), 905–914.

[CWL$^+$08] Z.-Q. Cheng, Y.-Z. Wang, B. Li, K. Xu, G. Dang, and S.-Y. Jin, *A survey of methods for moving least squares surfaces*, Fifth Eurographics / IEEE VGTC conference on Point-Based Graphics, 2008, pp. 9–23.

[CX04] Long Chen and Jinchao Xu, *Optimal Delaunay triangulations*, Journal of Computational Mathematics-International Edition- (2004), 299–308.

[DEGN99] Tamal Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry Nekhayev, *Topology preserving edge contraction*, Publ. Inst. Math.(Beograd)(NS) **66** (1999), 23–45.

[DJL12] Tamal K. Dey, Firdaus Janoos, and Joshua A. Levine, *Meshing interfaces of multi-label data with Delaunay refinement*, Engineering with Computers **28** (2012), 71–82.

[DL09] Tamal Dey and Joshua Levine, *Delaunay meshing of piecewise smooth complexes without expensive predicates*, Algorithms **2** (2009), 1327–1349.

[dlG95] E Brière de l'Isle and Paul-Louis George., *Optimization of tetrahedral meshes*, Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations, IMA Volumes in Mathematics and its Applications, 1995, pp. 97–128.

[DVS$^+$09] J. Dardenne, S. Valette, N. Siauve, N. Burais, and R. Prost, *Variational tetrahedral mesh generation from discrete volume data*, The Visual Computer **25** (2009), 401–410.

[FAB$^+$11] G. Fouquier, J. Anquez, I. Bloch, C. Falip, and C. Adamsbaum, *Subcutaneous adipose tissue segmentation in whole-body MRI of children*, CIARP 2011, vol. LNCS 7042, 2011, pp. 97–104.

[Far12] *Voxmorph: 3-scale freeform deformation of large voxel grids*, Computers & Graphics **36** (2012), 562 – 568, Shape Modeling International (SMI).

[FGBP11] François Faure, Benjamin Gilles, Guillaume Bousquet, and Dinesh K. Pai, *Sparse Meshless Models of Complex Deformable Solids*, SIGGRAPH, 2011, pp. 73:1–73:10.

[FOg97] Lori A. Freitag and Carl Ollivier-gooch, *Tetrahedral mesh improvement using swapping and smoothing*, International Journal for Numerical Mthods In Engineering **40** (1997), 3979–4002.

[GAB12] Thierry Guillemot, Andrès Almansa, and Tamy Boubekeur, *Non local point set surfaces*, International Conference on 3D Imaging, Moldeing, Processing, Visualization and Transmission (3DIMPVT), 2012, pp. 324–331.

[Gao12] James Gao, *The Functional Magnetic Resonance Imaging Semantic Space WebGL App*, 2012, http://gallantlab.org/semanticmovies/.

[GBKG04] Soren Grimm, Stefan Bruckner, Armin Kanitsar, and Eduard Groller, *Memory efficient acceleration structures and techniques for cpu-based volume raycasting of large data*, Volume Visualization and Graphics, 2004 IEEE Symposium on, IEEE, 2004, pp. 1–8.

[GG07] Gaël Guennebaud and Markus Gross, *Algebraic point set surfaces*, ACM Transactions on Graphics (TOG) **26** (2007).

[GH97] Michael Garland and Paul S. Heckbert, *Surface simplification using quadric error metrics*, 24th annual conference on Computer graphics and interactive techniques, 1997, pp. 209–216.

[GH98] Michael Garland and Paul S. Heckbert, *Simplifying surfaces with color and texture using quadric error metrics.*, IEEE Vis., 1998, pp. 263–269.

[GMMW11] J. Gao, I. Munteanu, W. F. O. Müller, and T. Weiland, *Generation of postured voxel-based human models for the study of step voltage excited by lightning current*, Advances in Radio Science **9** (2011), 99–105.

[GS11] O. Goksel and S.E. Salcudean, *Image-based variational meshing*, Medical Imaging, IEEE Transactions on **30** (2011), 11 –21.

[GZ05] Michael Garland and Yuan Zhou, *Quadric-based simplification in any dimension*, ACM Transactions on Graphics (TOG) **24** (2005), 209–239.

[HCLB09] Jin Huang, Lu Chen, Xinguo Liu, and Hujun Bao, *Efficient mesh deformation using tetrahedron control mesh*, Computer Aided Geometric Design **26** (2009), 617–626.

[Hop96] Hugues Hoppe, *Progressive meshes*, 23rd annual conference on Computer graphics and interactive techniques, 1996, pp. 99–108.

[ILGS06] Martin Isenburg, Peter Lindstrom, Stefan Gumhold, and Jonathan Shewchuk, *Streaming compression of tetrahedral volume meshes*, Proceedings of Graphics Interface, 2006.

[IMH05] Takeo Igarashi, Tomer Moscovich, and John F. Hughes, *As-rigid-as-possible shape manipulation*, ACM Transactions on Graphics (TOG) **24** (2005), 1134–1141.

[Ins09] Scientific Computing Imaging Institute, *Simulation of electric field generated by an implanted cardiac defibrillator in a torso model*, 2009, http://www.sci.utah.edu/gallery2/v/cibc/.

[ITI10] ITIS, *Posable models*, 2010, http://www.itis.ethz.ch/services/anatomical-models/posable-models/.

[JMD$^+$07] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki, *Harmonic coordinates for character articulation*, ACM Transactions on Graphics (TOG) **26** (2007), 71.

[JSW05] Tao Ju, Scott Schaefer, and Joe Warren, *Mean value coordinates for closed triangular meshes*, ACM Transactions on Graphics (TOG) **24** (2005), 561–566.

[KBpS00] Leif Kobbelt, Thilo Bareuther, and Hans peter Seidel, *Multiresolution shape deformations for meshes with dynamic vertex connectivity*, 2000, pp. 249–260.

[KE02] Martin Kraus and Thomas Ertl, *Simplification of nonconvex tetrahedral meshes*, Hierarchical and Geometrical Methods in Scientific Visualization, 2002, pp. 185–196.

[KHF$^+$06] Andrea Kratz, Markus Hadwiger, Anton Fuhrmann, Rainer Splechtna, and Katja Bühler, *Gpu-based high-quality volume rendering for virtual environments*, International Workshop on Augmented Environments for Medical Imaging and Computer Aided Surgery (AMI-ARCS), vol. 2006, 2006.

[KS07] Bryan Matthew Klingner and Jonathan Richard Shewchuk, *Agressive tetrahedral mesh improvement*, Proceedings of the 16th International Meshing Roundtable, 2007, pp. 3–23.

[KW03a] J. Kruger and R. Westermann, *Acceleration techniques for gpu-based volume rendering*, Visualization, 2003. VIS 2003. IEEE, 2003, pp. 287–292.

[KW03b] Jens Kruger and Rüdiger Westermann, *Acceleration techniques for gpu-based volume rendering*, Proceedings of the 14th IEEE Visualization 2003 (VIS'03), IEEE Computer Society, 2003, p. 38.

[LC87] William E. Lorensen and Harvey E. Cline, *Marching cubes: A high resolution 3d surface construction algorithm*, no. 4.

[Lev03] D. Levin, *Mesh-independent surface interpolation*, Geometric Modeling for Scientific Visualization (2003), 37–49.

[Lio50] J Liouville, *Extension au cas des trois dimensions de la question du tracé géographique*, Application de l'Analyse á la Géométrie (1850), 609–616.

[LLBP12] Shusen Liu, Joshua A Levine, Peer-Timo Bremer, and Valerio Pascucci, *Gaussian mixture model based volume visualization*, Large Data Analysis and Visualization (LDAV), 2012, pp. 73–77.

[LLC11] SimQuest LLC, *Simquest open-incision surgical simulation*, 2011, http://www.medgadget.com/2011/03/simquest_openincision_surgical_simulation.

[LLCO08] Yaron Lipman, David Levin, and Daniel Cohen-Or, *Green coordinates*, ACM Transactions on Graphics (TOG) **27** (2008), 78:1–78:10.

[LS07] François Labelle and Jonathan Richard Shewchuk, *Isosurface stuffing: fast tetrahedral meshes with good dihedral angles*, ACM Transactions on Graphics (TOG) **26** (2007).

[LSLCO05] Yaron Lipman, Olga Sorkine, David Levin, and Daniel Cohen-Or, *Linear rotation-invariant coordinates for meshes*, ACM Transactions on Graphics (TOG) **24** (2005), 479–487.

[Maî03] Henri Maître, *Le traitement des images*, Hermès science, 2003.

[MFS06] Gerd Marmitt, Heiko Friedrich, and Philipp Slusallek, *Interactive volume rendering with ray tracing*, Eurographics State of the Art Reports (2006), 115–136.

[Moh12] Zakiuddin Shehzan Ayub Mohammed, *Smoke simulation*, 2012, http://www.colorseffectscode.com/Projects/SmokeSimulation.html.

[MWK+08] M. Meyer, R. Whitaker, R.M. Kirby, C. Ledergerber, and H. Pfister, *Particle-based sampling and meshing of surfaces in multimaterial volumes*, Visualization and Computer Graphics, IEEE Transactions on **14** (2008), 1539 –1546.

[NW08] Tomoaki Nagaoka and Soichi Watanabe, *Postured voxel-based human models for electromagnetic dosimetry*, Physics in Medicine and Biology **53** (2008), no. 24, 7047.

[NW09] T. Nagaoka and S. Watanabe, *Voxel-based variable posture models of human anatomy*, no. 12, 2015–2025.

[NZZ+10] Yong Hum Na, Binquan Zhang, Juying Zhang, Peter F Caracappa, and X George Xu, *Deformable adult human phantoms for radiation protection dosimetry: anthropometric data representing size distributions of adult worker populations and software algorithms*, Physics in Medicine and Biology **55** (2010), no. 13, 3789.

[OGG] Cengiz Oztireli, Gaël Guennebaud, and Markus Gross, *Feature Preserving Point Set Surfaces based on Non-Linear Kernel Regression*, Computer Graphics Forum **28**, 493–501.

[oNIaUfBI13] Laboratory of Neuro Imaging at UCLA and Martinos Center for Biomedical Imaging, *Human Connectome Project*, 2013, http://www.humanconnectomeproject.org/.

[PSB$^+$07] J.-P. Pons, F. Ségonne, J.-D. Boissonnat, L. Rineau, M. Yvinec, and R. Keriven, *High-quality consistent meshing of multi-label datasets*, 20th international conference on Information processing in medical imaging, 2007, pp. 198–210.

[RGW$^+$03] Stefan Roettger, Stefan Guthe, Daniel Weiskopf, Thomas Ertl, and Wolfgang Strasser, *Smart hardware-accelerated volume rendering*, Proceedings of the symposium on Data visualisation 2003 (Aire-la-Ville, Switzerland, Switzerland), VISSYM '03, Eurographics Association, 2003, pp. 231–238.

[Rup95] Jim Ruppert, *A delaunay refinement algorithm for quality 2-dimensional mesh generation*, J. Algorithms **18** (1995), 548–585.

[SA07] Olga Sorkine and Marc Alexa, *As-rigid-as-possible surface modeling*, Symposium on Geometry Processing, 2007, pp. 109–116.

[SASW96] Victor Spitzer, Michael J Ackerman, Ann L Scherzinger, and David Whitlock, *The visible human male: a technical report*, Journal of the American Medical Informatics Association **3** (1996), 118–130.

[SCCB05] Cláudio Teixeira Silva, Joao Luiz Dihl Comba, Steven Paul Callahan, and Fabio Fedrizzi Bernardon, *A survey of gpu-based volume rendering of unstructured grids*, Revista de informática teórica e aplicada. Porto Alegre, RS. Vol. 12, n. 2 (out. 2005), p. 9-29 (2005).

[Sch05] Henning Scharsach, *Advanced gpu raycasting*, In Proceedings of CESCG 2005, 2005, pp. 69–76.

[SCOL$^+$04] Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian Rössl, and H-P Seidel, *Laplacian surface editing*, Symposium on Geometry Processing, 2004, pp. 175–184.

[SEG$^+$12] Sarah J Short, Jed T Elison, Barbara Davis Goldman, Martin Styner, Hongbin Gu, Mark Connelly, Eric Maltbie, Sandra Woolson, Weili Lin, Guido Gerig, et al., *Associations between white matter microstructure and infants' working memory*, NeuroImage (2012).

[She98] Jonathan Richard Shewchuk, *Tetrahedral mesh generation by Delaunay refinement*, Fourteenth annual symposium on Computational geometry, 1998, pp. 86–95.

[She02] Jonathan Richard Shewchuk, *What is a good linear element? - interpolation, conditioning, and quality measures*, 11th International Meshing Roundtable, 2002, pp. 115–126.

[SL07] Wenhao Song and Ligang Liu, *Stretch-based tetrahedral mesh manipulation*, Proceedings of Graphics Interface 2007, ACM, 2007, pp. 319–325.

[SP86] Thomas W. Sederberg and Scott R. Parry, *Free-form deformation of solid geometric models*, ACM Siggraph Computer Graphics, 1986, pp. 151–160.

[TH00] Allen Taflove and Susan C Hagness, *Computational electrodynamics*, vol. 160, Artech house BostonLondon, 2000.

[TNB10] Dilip Mathew Thomas, Vijay Natarajan, and Georges-Pierre Bonneau, *Link Conditions for Simplifying Meshes with Embedded Structures*, Transactions on Visualization and Computer Graphics (2010), 1007–1019.

[TS97] Vasily V Titov and Costas Emmanuel Synolakis, *Extreme inundation flows during the hokkaido-nansei-oki tsunami*, Geophysical Research Letters **24** (1997), 1315–1318.

[TSA09] Jane Tournois, Rahul Srinivasan, and Pierre Alliez, *Perturbing Slivers in 3D Delaunay Meshes*, 18th International Meshing Roundtable, 2009, pp. 157–173.

[TWAD09] Jane Tournois, Camille Wormser, Pierre Alliez, and Mathieu Desbrun, *Interleaving Delaunay refinement and optimization for practical isotropic tetrahedron mesh generation*, ACM Transactions on Graphics (TOG) **28** (2009), 75:1–75:9.

[VBHH11] Fabien Vivodtzev, Georges-Pierre Bonneau, Stefanie Hahmann, and Hans Hagen, *Substructure topology preserving simplification of tetrahedral meshes*, Topological Methods in Data Analysis and Visualization, 2011, pp. 55–66.

[VRpS03] Jens Vorsatz, Christian Rössl, and Hans peter Seidel, *Dynamic remeshing and applications*, Department, Stony Brook University. Her, 2003, pp. 167–175.

[Web] Web3d, *Basic brain, shaded: X3d specification volume visualization example*, http://web3d.org.

[Wei06] Daniel Weiskopf, *Gpu-based interactive visualization techniques*, Springer, 2006.

[WS03] Ziji Wu and John M. Sullivan, *Multiple material marching cubes algorithm*, International Journal for Numerical Methods in Engineering **58** (2003), 189–207.

[YZX$^+$04] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum, *Mesh editing with poisson-based gradient field manipulation*, ACM Transactions on Graphics (TOG), vol. 23, 2004, pp. 644–651.

[ZLXP09] Yong Zhao, Xinguo Liu, Chunxia Xiao, and Qunsheng Peng, *A unified shape editing framework based on tetrahedral control mesh*, Computer Animation and Virtual Worlds **20** (2009), no. 2-3, 301–310.

[ZNM10] Shaoting Zhang, Andrew Nealen, and Dimitris Metaxas, *Skeleton based as-rigid-as-possible volume modeling*, Eurographics Association (2010), 21–24.