# A parallel resampling method for interactive deformation of volumetric models

Alejandro Rodríguez[a], Alejandro León[a], Domingo Martín[a], Miguel A. Otaduy[b]

[a] *University of Granada, Granada, Spain*
[b] *URJC, Madrid*

**Abstract**

In this work, we propose a method to interactively deform high-resolution volumetric datasets, such as those obtained through medical imaging. Interactive deformation enables the visualization of these datasets in full detail using state-of-the-art volume rendering techniques as they are dynamically modified. Our approach relies on resampling the original dataset to a target regular grid, following a 3D rasterization technique. We employ an implicit auxiliary mesh to execute resampling, which allows us to decouple mapping of the deformation field to the volume from actual resampling. In this way, our method is practically independent of the deformation method of choice, as well as of the resolution of the deformation meshes. We show how our method lends itself nicely to an efficient, massively parallel implementation on GPUs, and we demonstrate its application on several high-resolution datasets and deformation models.

*Keywords:*
Volume data, volume deformation, 3D rasterization

## 1. Introduction

The tissue distribution of biological forms is often captured and visualized using volumetric representations, most notably in medical applications. Biological tissue is soft and deformable for the most part, hence applications dealing with biological tissue require methods to deform such volumetric representations. Some common examples in the medical field include image registration, intra-operative navigation, or surgical planning. Nowadays practical implementations of these applications are often limited to rigid data or non-interactive solutions, due to the difficulty to deform volumetric representations in an interactive manner.

Regardless of the particular deformation method of choice, one major challenge for the development of applications using deformed volume data is the visualization of dynamic volumetric representations. Traditionally, there are mainly two strategies to address this problem: (i) Segment and mesh the original volume data, and resort to visualization of surface meshes. This strategy fails to capture the full detail in the original volume data, and it does not allow interactive modification of the visualized isosurfaces as offered by volume rendering techniques. (ii) Execute volume rendering using unstructured meshes. Despite achieving interactive framerates, as demonstrated by Georgii et al. [1], the cost of unstructured volume rendering grows with mesh resolution. In addition, as remarked by Correa et al. [2], advanced lighting such as gradient-based lighting, which is an important feature in medical applications, becomes complex even in the case of static unstructured meshes.

In our work, we follow a different strategy. We propose to deform the original volume data by dynamically resampling it onto a regular grid, and then apply regular volume rendering. Of course, this strategy is not new to computer graphics, as it constitutes the fundamental pipeline of raster graphics with deferred shading. This strategy has already been applied to volume data deformation, by other authors, e.g. Schulze et al. [3] and Gascón et al. [4], but we achieve more than one order of magnitude speed-up compared to previous methods. As a result, we can deform and render high-resolution volumetric objects with high-resolution deformable meshes at interactive rates.

In our method, the input volumetric dataset is transformed into an implicit sampling mesh of the same resolution. At runtime, two steps are performed. First, after the deformation stage, the deformation field is applied to the positions of the nodes of the sampling mesh. And second, the deformed sampling mesh is resampled onto the target regular grid. Both steps are massively parallelized in our GPU implementation. With our two-step method, we decouple the resampling from the evaluation of deformations, and due to the implicit and regular nature of the sampling mesh, we eliminate the need for any type of indirection or auxiliary data structure, and thus we maximize throughput, achieving better performance than previous methods. Actual volume rendering is decoupled into yet a third step, which is executed efficiently on the target regular grid.

The decoupling of deformation mapping, resampling, and volume rendering has several advantages over previous approaches. In contrast to methods based on unstructured volume rendering, one major advantage is the possibility to adopt advanced lighting models efficiently, such as gradient-based volume lighting.

*Email address:* `alejandrora@ugr.es` (Alejandro Rodríguez)

Figure 1: A cutting operation is applied to a sheep heart dataset consisting of 12.4 million voxels, deformed using a Mass-Spring model. Our proposed resampling algorithm allows us to deform and cut the model interactively. After each simulation step, our method outputs a resampled regular grid of the same resolution as the input, which is visualized with a standard Ray-casting algorithm, revealing internal features due to the open cut. Resampling of this high-resolution dataset is executed in just 26 ms per frame.

Moreover, the size of the visualization viewport and the resolution of the deformation model affect performance separately, and there is no extra penalty in combining a large viewport with a high-resolution deformation model. When compared with all previous methods, one general advantage of ours is that it can be coupled with any deformation technique by means of a mapping of the underlying deformation structure to the implicit sampling mesh. And yet a final but important advantage is that the performance of the resampling algorithm is independent of the resolution of the underlying deformation structure, thus allowing interactive visualization when using deformation meshes several orders of magnitude denser than previous methods. This feature becomes critical to support deformation algorithms that support high-resolution meshes, such as GPU-parallel soft-tissue simulation algorithms based on implicit FEM [5] or Mass-Spring [6], coarsening algorithms that govern a heterogeneous high-resolution tetrahedral mesh through an efficient homogenized low-resolution simulation [7, 8, 9], or the ChainMail algorithm [3, 10], which can handle interactively models with millions of nodes.

In our examples we have tested diverse models such as the Finite Element Method, the Mass-Spring Model and the ChainMail algorithm, and we also include extensions to handle topological changes. We show that we can successfully deform and visualize a volume with 12.4 million voxels, using a mass-spring model with 2.4 million tetrahedra, at a rate of more than 20 fps (Fig. 1).

The remainder of this paper is organized as follows: Section 2 reviews related work. Our parallel resampling method is described in Section 3, including the definition of the implicit sampling mesh, the mapping process and the resampling algorithm, together with implementation details. Section 4 presents several results, coupling our pipeline with several deformation methods. Several experiments to analyze the properties of the method are also presented. Finally, our conclusions are listed in Section 5.

## 2. Previous works

Existing techniques for visualizing deformable volumetric models can be grouped into three main strategies.

A first group of approaches relies on the concept of spatial deformation. Instead of applying a deformation to the model, an inverse deformation map is applied to the rays that traverse the space containing the volume. Rezk-Salama et al. [11] proposed a spatial deformation technique dividing the space into a hierarchical set of deforming sub-cubes, thus defining a piecewise approximated inverse deformation field. Westermann et al. [12] proposed a set of local and global free-form space deformations, applying the inverse deformations through tesselated slicing planes. H. Chen et al. [13] applied Ray-casting to volumetric models deformed through a free-form deformation (FFD) approach using an inverse ray deformation technique. M. Chen et al. [14] introduced the concept of Spatial Transfer Functions as a tool to define FFD operations. Correa et al. [15] presented a set of FFD spatial operators enforcing an alignment with the features present in the models to generate medical illustrations. These spatial deformation schemes enable interactive frame rates, but at the price of low-resolution, non-physically based deformation.

A second group of approaches deforms an unstructured volumetric mesh, and then executes volume rendering on this unstructured mesh. Classic and current unstructured volume rendering techniques aim for an accurate visualization of non-regular volumetric models that can be deformed over time (see the works of Miranda et al. [16] and Okuyan et at. [17] for recent GPU-based unstructured volume rendering techniques). However, a direct application to physically deformed medical volumes impedes interactive frame rates under high-resolution deformation structures, since medical models may contain several orders of magnitude more elements than those handled by these techniques interactively. Inheriting many of the ideas of these approaches, Georgii et al. [1] proposed a system to perform unstructured volume rendering of tetrahedral meshes deformed by physical simulation schemes using the GPU rendering pipeline,

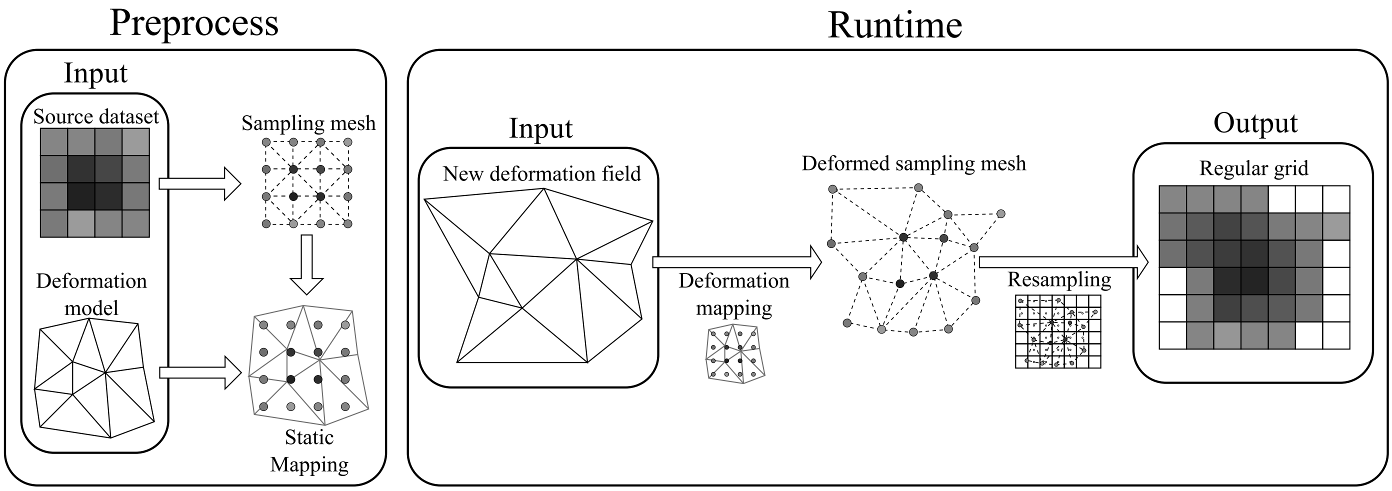


Figure 2: An overview of our method. In a preprocessing step, the sampling mesh is created using the input dataset, and coupled with the underlying deformation structure, generating static mapping information. After every deformation step, the resulting deformation is mapped to the sampling mesh, which is then resampled to a regular grid.

also allowing the use of 3D texture mapping to increase the detail inside a tetrahedron. While the method achieves interactive framerates for relatively large models (i.e., 100 ms per frame for a tetrahedral mesh of 190,000 elements using a 512x512 viewport), its cost grows bilinearly with mesh resolution and viewport size. Nakao et al. [18] proposed the use of a dynamically refined proxy mesh to adapt the mesh complexity to the deformations applied to the model. This scheme handles large volumes and allows topological changes in the model but, as the authors point out, the performance of the method depends on the number of nodes of the deformable model, and can only support a few hundred interactively.

A third group of approaches, motivated by the superior performance of direct volume rendering (DVR) techniques over the unstructured volume rendering techniques, as well as the more advanced shading and illumination techniques available under DVR, performs a resampling of the deformed volume onto a regular grid which is later fed as input to a standard DVR pipeline. Schulze et al. [3] proposed a resampling algorithm based on a nearest neighbor search to relocate and interpolate the deformed voxels. However, the algorithm is designed to perform resampling only on small parts of the volume, and performance degrades badly if the deformed volume is large. In addition, the resampling process is highly dependent on the ChainMail deformation technique. Gascón et al. [4] proposed a GPU-based tetrahedral mesh rasterization algorithm with 3D texture mapping. After each simulation step, the deformed tetrahedra are rasterized onto a regular grid, and target voxels are mapped to the original volume for a texture lookup. The parallel implementation of the rasterization process achieves interactive frame rates for high-resolution volumes, but performance degrades under high-resolution meshes. In addition, the method only supports deformation techniques based on tetrahedral meshes.

Our method also falls in this group, as it performs a parallel resampling of the original volume data set onto a regular grid. However, thanks to the use of an intermediate implicit sampling mesh that decouples the computation of the deformation from the rasterization, we avoid costly indirections during the actual rasterization, and the performance is independent of the resolution of the underlying deformation structure.

## 3. Volume resampling pipeline

The key to the efficiency of our volume resampling method is the use of an implicit sampling mesh that effectively decouples the data structure of the particular deformation model of choice from the actual resampling of the volume. Fig. 2 outlines the full resampling pipeline.

In a preprocessing step, the sampling mesh is generated from the input dataset and it is coupled with the deformation method. At runtime, two steps are performed after each deformation stage on a massively parallel manner on the GPU: the mapping of the deformation to the sampling mesh, and the resampling of the deformed sampling mesh.

This section starts with a description of the implicit sampling mesh, and continues with descriptions of the deformation mapping and the actual resampling.

### *3.1. Implicit sampling mesh*

Given a regular grid as input dataset, we define a regular mesh of the same resolution that carries in its vertices the data associated to grid points. This regular mesh retains absolutely all the information present in the original dataset. When the mesh is deformed, a continuous field can be reconstructed on the entire volume occupied by the mesh through interpolation of the original data values inside each mesh element.

Specifically, given an input dataset stored as a regular grid of $l \times m \times n$ voxels, we create an array of $l \times m \times n$ vertices, each of them associated to one voxel. Each vertex is assigned the data value of its corresponding voxel, and is placed at a position in

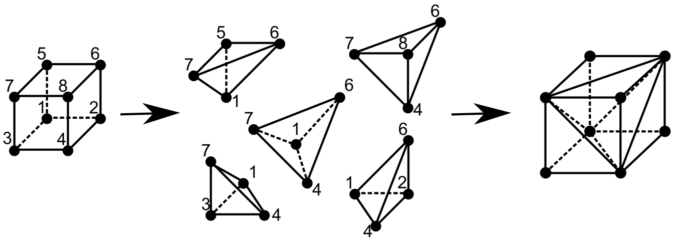


Figure 3: A 5T tetrahedral decomposition generates five adjoining tetrahedra covering the same volume as the original hexahedron.

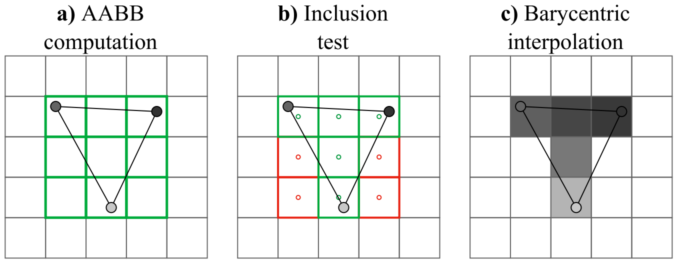


Figure 4: Steps of the resampling process on a 2D example. a) The AABB of the triangle is computed. b) An Inclusion test is performed for each voxel in the AABB, using the barycentric coordinates of its center. c) Output data values are assigned to the voxels lying inside the triangle through barycentric interpolation.

space given by its indices and the spacing of the model in each dimension.

Given the initial layout of the vertices, every eight adjacent vertices define a hexahedron. We implicitly decompose each hexahedron into five adjoining tetrahedra, following a 5T decomposition, as shown in Fig. 3.

Each hexahedron can be decomposed into five adjoining tetrahedra (5T decomposition) partitioning the entire volume of the hexahedron as shown in Fig. 3. We transform the hexahedral mesh into an adjoining tetrahedral mesh by applying mirrored 5T decompositions to adjacent hexahedra [19], using the shared face as a mirror plane.

The proposed decomposition scheme produces a mesh that is continuous and complete. These properties guarantee that, as long as the input deformation is self-intersection free, every point inside the mesh is bounded always by one and only one tetrahedron.

Thanks to the regularity of the mesh, we can infer the tetrahedra incident on each vertex simply from its indices. As a result, the sampling mesh is only implicitly defined, without the need to explicitly build it or store it. Therefore, the implicit sampling mesh does not add any overhead to the storage of the original dataset aside from the vertex positions, and it is visited on-the-fly during the resampling step.

We have considered other options for the definition of the implicit sampling mesh, in particular the hexahedral mesh defined inherently by the grid. However, we have opted for our tetrahedral mesh because inclusion tests and interpolation functions are simpler (non-planar faces are avoided), hence more efficient at run-time. Despite having more elements than the hexahedral mesh, the implicit definition of our tetrahedral mesh avoids storage penalties.

### 3.2. Deformation mapping

The sampling mesh acts as an intermediate representation between each particular deformation method and the resampling process. The deformation method produces a deformation field that can be evaluated at any location in space. In order to carry out this evaluation efficiently, we set a static mapping between the deformation method and the sampling mesh.

In a preprocessing step, for each vertex of the sampling mesh, we store static mapping information, i.e., appropriate pointers to the elements of the deformation method, as well as static weights or coefficients needed for evaluating the deformation field. The particular pointers, weights and/or coefficients stored per vertex depend on the particular deformation method of choice.

At runtime, once the deformation model is updated, the mapping process is executed to define the updated positions of the sampling mesh vertices, according to the new deformation field. This mapping is executed on a massively parallel manner on the GPU, and the actual functions to be evaluated depend again on the particular deformation method of choice.

In Section 4 we show examples with different deformation methods.

### 3.3. Volumetric resampling

At runtime, once the deformation is mapped onto the vertices of the sampling mesh, we execute the resampling of the original dataset onto a target regular grid. This process is executed in parallel for all the tetrahedra of the sampling mesh, and each tetrahedron contains all the information needed in the process, i.e., the target positions of its four vertices and the original data values to be interpolated.

For each tetrahedron of the sampling mesh, we perform the following process. First, we select candidate target voxels by computing an axis-aligned bounding box (AABB) of the vertices of the tetrahedron. Then, we traverse all the candidate voxels, and compute their barycentric coordinates. For voxels that lie inside the tetrahedron, we compute the output value through barycentric interpolation of the data values stored in the four vertices of the tetrahedron. A simplified 2D version of the resampling process is shown in Fig. 4.

#### 3.3.1. GPU implementation

This algorithm is well suited for massive GPU parallelization. We have implemented it as a single GPU kernel that runs in parallel on the hexahedral decomposition of the sampling mesh, thus each thread visits the five tetrahedra defined implicitly on each hexahedron.

A high-level pseudo-code of the resampling kernel is outlined in Code 1. Each thread loads the eight vertices of a hexahedron (lines 7-14), labeled as indicated in Fig. 3. Then, to ensure that the complete mesh remains adjoining, the vertices are reordered as necessary, thus the labels of the vertices are exchanged (lines 15-23) depending on the parity of the indices of the first vertex (vertex 1 in Fig. 3). Note that this operation

```
1 kernel_resample(tex3D outGrid)
2   int x,y,z;
3   x = getThreadIndexX();
4   y = getThreadIndexY();
5   z = getThreadIndexZ();
6   vertex v1, v2, v3, v4, v5, v6, v7, v8;
7   v1 = getVertex(x, y, z);
8   v2 = getVertex(x+1, y, z);
9   v3 = getVertex(x, y, z+1);
10  v4 = getVertex(x+1, y, z+1);
11  v5 = getVertex(x, y+1, z);
12  v6 = getVertex(x+1, y+1, z);
13  v7 = getVertex(x, y+1, z+1);
14  v8 = getVertex(x+1 ,y+1, z+1);
15  IF (x % 2 == 1)
16    swap(v1, v2);swap(v3, v4);swap(v5, v6);swap(v7, v8);
17  ENDIF
18  IF (y % 2 == 1)
19    swap(v1, v5);swap(v2, v6);swap(v3, v7);swap(v4, v8);
20  ENDIF
21  IF (z % 2 == 1)
22    swap(v1, v3);swap(v2, v4);swap(v5, v7);swap(v6, v8);
23  ENDIF
24  SampleTetrahedron (v1, v3, v4, v7, outGrid);
25  SampleTetrahedron (v7, v8, v4, v6, outGrid);
26  SampleTetrahedron (v4, v2, v1, v6, outGrid);
27  SampleTetrahedron (v1, v5, v7, v6, outGrid);
28  SampleTetrahedron (v7, v4, v6, v1, outGrid);
29 END
30
31 SampleTetrahedron (vertex v1, v2, v3, v4, Tex3D outGrid)
32  aabb boundingBox = outGrid.computeAABB(v1, v2, v3, v4);
33  FOREACH (voxel IN boundingBox)
34    float4 baryCoords = computeBaryCoords(voxel.center, v1,
       v2, v3, v4);
35    IF (centerLiesInsideTetrahedron(baryCoords))
36      char newValue = interpolateValue(baryCoords, v1, v2, v3,
       v4);
37      setValue(voxel, dataValue);
38    ENDIF
39  ENDFOREACH
40 END
```

Code 1: Pseudo-code of the resampling GPU kernel. Each thread handles one hexahedron of the sampling mesh, the eight vertices of the hexahedron are loaded. After that, the corresponding tetrahedra are deduced on-the-fly and sampled onto the output grid.

is handled at runtime, without ever storing the tetrahedral mesh explicitly. Lastly, each tetrahedron is actually sampled on the target grid (lines 24-28) as explained earlier.

Thanks to the regular and structured nature of our sampling mesh, consecutive threads access consecutive array positions and thus we achieve coalesced global memory read operations (lines 7-14). This coalesced access scheme is achieved independently of the current configuration of the mesh since it depends on the GPU thread ids alone. For this same reason, the access to vertex-based data (such as deformed positions and volume data) is easily optimized using shared memory since neighboring threads belonging to the same thread warp access neighboring vertices. Moreover, the implicit definition of the mesh eliminates the need for any indirection scheme for the topology definition, saving both memory requirements and global and shared memory accesses.

## 4. Results and discussion

We have tested our volume deformation method with several different deformation models. In the following subsections we discuss implementation details for each deformation model, along with performance data. We refer the reader to the video provided as additional material for more results. We also present the results of several tests performed in order to analyze the properties and limitations of our method.

We have implemented our algorithm using OpenCL 1.2, running on an Intel Core i5-3570 machine with 8 GB RAM, equipped with an AMD Radeon R9 270X with 2 GB of video memory GDDR5.

### *4.1. ChainMail deformation*

A parallel version of the ChainMail algorithm, similar to the one proposed by Rößler [20] has been implemented. The ChainMail algorithm, introduced by Gibson [21], applies elastic and plastic deformations to the model at the same resolution of the input dataset by defining geometric constraints between neighbor elements of the model. Heterogeneous deformations can be achieved by defining different restrictions to the elements, as explained in [22].

Since the ChainMail algorithm works at the resolution of the input dataset, in this case the vertices of the sampling mesh are co-located with the ChainMail elements and the mapping is straightforward.

We have integrated the possibility to execute simple topological changes on the ChainMail model by implementing cutting and carving operations, following the approach in [23]. To apply the topological changes to the volume dataset for visualization purposes, we obtain acceptable results simply by deleting tetrahedra that are cut or carved, thanks to the high resolution of the implicit sampling mesh. We add a *bitfield* char value to each hexahedron to flag individual tetrahedra as active or inactive, and we check this bitfield as part of the resampling kernel in Code 1. When a new cutting/carving operation is applied to the model, we perform an intersection test with the cut surface (or carving volume) for each tetrahedron, and

Figure 5: A foot dataset consisting of 16.7 million voxels is deformed using a corotational finite element method using a mesh of 16,000 tetrahedra. Our resampling pipeline generates a resampled regular grid in 39 ms, which is then visualized with a standard Ray-casting algorithm.

flag new inactive tetrahedra accordingly. With simple cut surfaces or carving volumes, a brute-force GPU parallelization of per-tetrahedron tests turned out to be fast enough. We refer the reader to the survey by Wu et al. [24] for more information on advanced cutting methods.

Fig. 6 shows example applications of our method on two medical datasets. The knee model with 13.2 million voxels is deformed at 35 ms per frame. 11 ms are devoted to ChainMail deformation, 0.5 ms to map the deformation to the sampling mesh, and 23.5 ms to resample the volume. Cutting operations are performed in less than 21 ms, and they affect performance only at frames when the cut is actually executed, not in subsequent frames. The head model with 6.6 million voxels is deformed at 16 ms per frame. Carving operations are performed in less than 8 ms, and they also affect performance only while carving is executed.

### *4.2. Mass-Spring deformation*

We have also tested our algorithm together with a dynamic simulation based on the mass-spring model, using tetrahedral meshes and explicit integration, parallelized on the GPU as proposed by Georgii et al. [25]. To implement the mapping from the mass-spring model to the implicit sampling mesh, as a preprocessing step we identify for each implicit node the mass-spring tetrahedron that contains it as well as its barycentric coordinates in the tetrahedron. At runtime we simply perform a barycentric combination of mass-spring node positions, which is trivially parallelized on the GPU.

We have also integrated simple cutting operations on the mass-spring model, separating adjacent tetrahedra by their shared face. To apply the cutting operations on the volume dataset, we follow the same approach as for the ChainMail model described above, using a bitfield of active tetrahedra.

Fig. 1 shows an example of a heart simulated with the mass-spring model that is deformed, cut, and resampled using our approach. With a volume dataset of 12.4 million voxels and a mass-spring model of 2.5 million tetrahedra, full volume deformation takes only 45.7 ms per frame. Dynamic deformations using explicit integration take 19 ms, deformation map-

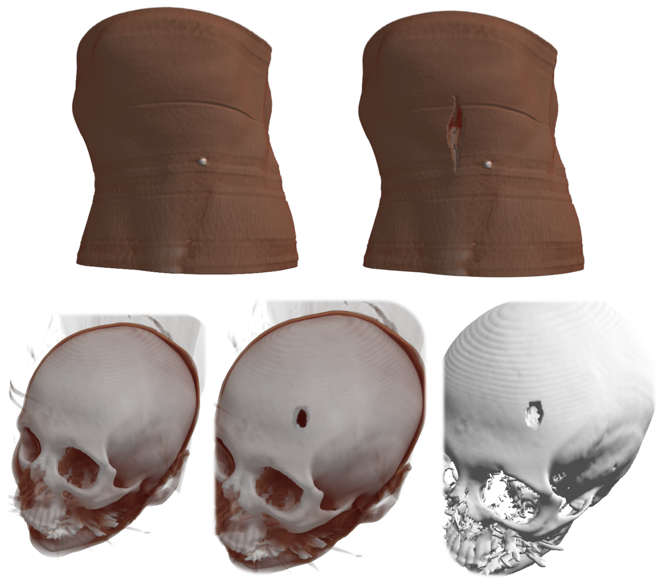

Figure 6: Interactive deformations of medical datasets using the ChainMail model.**Top:** A cutting operation is applied to a knee model consisting of 13.2 million voxels, followed by a deformation to visualize internal structures. Our pipeline resamples the volume in 24 ms. **Bottom:** A carving operation is applied to a head model consisting of 6.6 million voxels. Our pipeline resamples the volume in 11 ms. The output regular grid can be visualized with different direct volume rendering techniques, such as a standard Ray-casting volume rendering technique (top, bottom-left and bottom-center) or a Ray-casting-based isosurface extraction algorithm (bottom-right).

Table 1: Evaluation of the cost of the two steps of our algorithm (deformation mapping and resampling) for different deformation methods and deformation mesh resolutions. The cost of ray-casting the resampled model for a 700 × 700 viewport with 500 samples per ray is also shown. Finally, the size of the sampling mesh and its required GPU memory are also shown.

| **Deformation algorithm** | **Deformation nodes** | **Deformation mapping** | **Parallel resampling** | **Ray-casting** | **Sampling mesh size (tetrahedra)** | **Sampling mesh memory** |
|---|---|---|---|---|---|---|
| ChainMail | 13,200,705 | 0.4 ms | 23.3 ms | 8.2 ms | 65,153,280 | 188.84 MB |
| Mass-Spring | 2,000 | 5.7 ms | 24.2 ms | 7.5 ms | 65,153,280 | 390.27 MB |
| Mass-Spring | 512,000 | 7.4 ms | 23.7 ms | 8.3 ms | 65,153,280 | 390.27 MB |
| FEM | 125 | 5.9 ms | 24.1 ms | 7.3 ms | 65,153,280 | 390.27 MB |
| FEM | 3,200 | 5.9 ms | 23.9 ms | 7.2 ms | 65,153,280 | 390.27 MB |

ping takes 5.3 ms, and actual resampling takes 21.4 ms. Cuts applied to the model are computed in less than 50 ms.

### 4.3. FEM deformation

Finally, we have tested our algorithm with a corotational finite element method (FEM) [26], using a quasi-static solver with tetrahedral elements.

The deformation field is mapped to the sampling mesh using the exact same approach as for the mass-spring model described above. Fig. 5. shows an interactive deformation of a foot.

### 4.4. Performance analysis

We have carried out several tests to analyze the performance and scalability of our algorithm. The factors that we analyze are: the deformation method and its resolution, the resolution of the input volume dataset, the application of large deformations, and the size of the viewport. We also analyze preprocessing and memory costs.

#### 4.4.1. Deformation methods and their resolution

We have compared performance on the same input volume dataset for the three deformation methods listed earlier in this section. We have used the knee MRI dataset shown on the top row of Fig. 6, with a resolution of $189\times305\times229 = 13,200,705$ voxels (leading to a sampling mesh of approximately 65 million tetrahedra). We have tested the ChainMail model at the same resolution as the volume, and the Mass-Spring and FEM models on two different resolutions each. Table 1 reports the resolutions of the deformation models, the time spent on mapping the deformation to the sampling mesh, and the time spent on actual resampling. Timings were averaged over several runs of the algorithm.

As the results indicate, the cost of deformation mapping is notably lower than the cost of resampling. Furthermore, the cost of resampling is practically independent of the deformation model and its underlying resolution. This result was expected, as our sampling mesh succeeds to decouple deformation mapping from resampling. The cost of deformation mapping is also fairly insensitive to the resolution of the deformation mesh.

With the ChainMail model, deformation mapping is negligible. With the Mass-Spring and FEM models it grows slightly with the resolution of the deformation model, but even with a

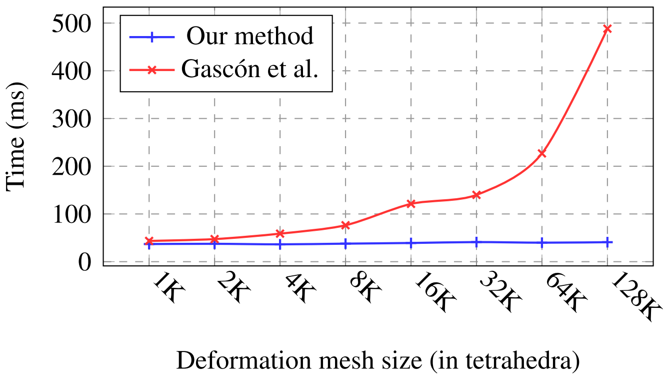


Figure 7: Scalability comparison of our method and the one by Gascón et al. [4] w.r.t. the resolution of the deformation mesh.

Mass-Spring model with over half a million nodes the cost remains low. It is important to note that we used much coarser meshes with the quasi-static FEM solver running on the CPU because with high-resolution meshes the actual deformation solver becomes the bottleneck. This was not the case with our Mass-Spring solver because of explicit time integration.

For the case of tetrahedral deformation meshes (either with the Mass-Spring or FEM models), we also performed a more extensive scalability analysis as a function of mesh resolution. And we compared the results of our algorithm to the results of the rasterization algorithm by Gascón et al. [4]. For this purpose, we used the foot dataset shown in Fig. 5. Fig. 7 shows scalability plots with our method and with the method of Gascón et al. Ours is fairly independent of mesh resolution, while theirs is largely penalized under high-resolution meshes. The reason is that their algorithm uses indirection mechanisms that depend on the deformation mesh during resampling. Ours, instead, takes advantage of the implicit sampling mesh to decouple deformation mapping and resampling, avoiding costly indirections.

#### 4.4.2. Volume resolution

In order to study the scalability of our resampling algorithm w.r.t. the resolution of the input volume dataset, we have measured the resampling time for several datasets. For this purpose, we have generated synthetically homogeneous data cubes of varying size. We have tested the three deformation meth-

Table 2: Performance results of the large deformation test shown in Fig. 9. The table show resampling times for the five proposed scenarios, samples per tetrahedron (average, minimum, and maximum), resampling throughput measured as samples per millisecond, grid size, and required memory.

| **Deformation** | **Resampling time** | **Average samples per tetrahedron (min, max)** | **Samples per millisecond** | **Output grid size** | **Output grid memory** |
|---|---|---|---|---|---|
| Undeformed | 14.6 ms | 1.07 (1, 3) | 2,670,286 | $256 \times 256 \times 113$ | 14.2 MB |
| Local deformation | 14.8 ms | 1.08 (1, 12) | 2,667,913 | $256 \times 256 \times 113$ | 14.2 MB |
| Rotated | 23.6 ms | 1.83 (2, 6) | 2,821,122 | $360 \times 372 \times 216$ | 55.2 MB |
| Spatially varying scaling | 42.7 ms | 3.25 (1, 8) | 2,769,767 | $384 \times 384 \times 170$ | 47.8 MB |
| Scaling 1.5x | 51.2 ms | 4.21 (3, 5) | 2,997,839 | $384 \times 384 \times 170$ | 47.8 MB |

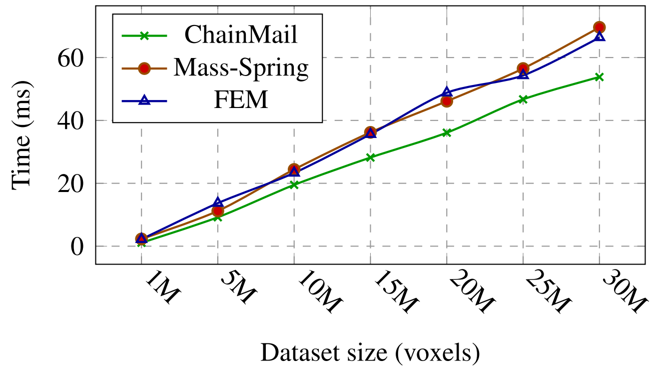


Figure 8: Scalability of our resampling pipeline w.r.t. the size of the input volume dataset. The three deformation methods have been applied to each tested dataset.

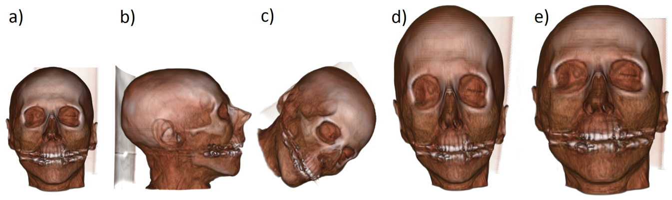


Figure 9: a) A head dataset consisting of 7.4 million voxels undergoes different large deformations. The model is b) deformed by a localized load on the nose, c) rotated 45 degrees on each axis, d) deformed using spatially varying scaling, and e) deformed using uniform scaling with a factor of 1.5.

ods on each dataset, using a mesh with 512,000 nodes for the Mass-Spring model and a mesh with 3,200 nodes for the FEM simulation.

As shown in Fig. 8, our resampling pipeline (including times of both deformation mapping and resampling) exhibits a linear growth with respect to the size of the input dataset for the three tested deformation methods. Notice that when coupled with the ChainMail algorithm, the growth rate is slightly lower due to the simpler mapping scheme.

### 4.4.3. *Performance under large deformations*

We have conducted another experiment to analyze the behavior of our resampling algorithm when large deformations are applied to the model. For this purpose, we have used a head dataset consisting of $256 \times 256 \times 113$ voxels (Fig. 9.a) ), and we have analyzed different deformed scenarios with respect to the undeformed configuration. The first scenario comprises a localized load on the nose (Fig. 9.b). The second scenario consists of a rotation of 45 degrees on each axis to maximize the misalignment of the sampling grid and the output grid (Fig. 9.c), increasing the size of the axis-aligned bounding box of the dataset by a factor of 3.9. The third scenario consists of a spatially varying scaled model, using a scaling factor going from 1.0 at the bottom to 1.5 at the top of the head (Fig. 9.d). The last scenario consists of a uniform scaling of the model by a factor of 1.5 (Fig. 9.e). In the last two cases, the volume of the axis-aligned bounding box of the dataset grows by a factor of 3.375.

Table 2 analyzes the cost of resampling for this experiment. Note that the amount of deformation affects only the cost of resampling, and the costs of deformation mapping and volume rendering are equal in all five cases. The localized load produces the highest peak in maximum samples per element for the elements affected but, due to its local scope, the performance of the algorithm is barely affected. The rotated configuration maximizes the misalignment of the sampling and the output grids, increasing the samples tested per element, and the resampling time grows by a factor of 1.61.

As expected, for the uniform scaling, the cost grows roughly linearly with the output volume. Both under uniform and spatially varying scaling, the size of the dataset grows from 7.4 million to 25.1 million voxels, but under spatially varying scaling the deformed model does not cover the entire grid. For this reason, it yields a slightly lower resampling cost. It is particularly interesting to analyze the throughput of the resampling algorithm, measured in terms of the grid samples handled per millisecond, and the uniform scaling case yields a higher throughput. The reason is that, under spatially varying scaling, the deformed tetrahedra do not have equal volumes. These size differences, together with the misalignment of the sampling and the output grids produced by the inhomogeneous scaling, lead to imbalanced workloads for the GPU threads, reducing the degree of parallelism. Notice how, for this example, the number of samples per tetrahedron may vary by a factor of 8. We can conclude that the performance of our method may become suboptimal under an extreme imbalance in the sizes of deformed tetrahedra, due to reduced parallelism.

### 4.4.4. *Viewport size*

Thanks to the decoupling of the resampling and the actual visualization, in our algorithm, the size of the viewport affects only the performance of visualization, not the resampling step.

This is different from the behavior of unstructured volume rendering methods, where the size of the viewport affects the cost of all major steps, as analyzed by Okuyan et al. [17]. In addition, unstructured volume rendering methods pay a performance penalty when they use both high-resolution meshes and large viewports. This is not the case with our algorithm, because mesh resolution and viewport size affect the cost of disjoint steps.

In most of our experiments, we have used a viewport of $700 \times 700$ pixels, with 500 samples per ray. Under this resolution, the cost of ray casting was of 7.4 ms per frame on average, and went up to 24.1 ms per frame on average with the addition of on-the-fly gradient-based lighting. With a viewport of $1920 \times 1080$ pixels, the average cost per frame of ray casting was 40.3 ms, and with gradient-based lighting it rose to 107.7 ms. All the images in the paper were generated using gradient-based lighting.

#### *4.4.5. Preprocessing*

Although less relevant than the runtime cost, the preprocessing cost of our method is low. It grows linearly with both the size of the volume dataset and the resolution of the deformation method. In the most demanding scenario we have tested, a volume with 30 million voxels and a Mass-Spring mesh with 512,000 nodes, the preprocessing step took less than 15 seconds.

#### *4.4.6. Memory*

Last, our method is limited by GPU memory, as memory requirements grow linearly with the size of the dataset. In single precision, each voxel requires a total of up to 31 bytes: 2 to store its data value, 12 for its deformed position, 16 for indices and barycentric weights of deformation nodes to implement the deformation mapping, and 1 to flag topological changes. All in all, our algorithm requires 29.5 MB of memory per million voxels in the original dataset. For the ChainMail algorithm, the memory requirements are lower, as shown in Table 1, because there is no need to store deformation mapping information.

The GPU memory required by the output regular grid is comparatively lower, with only 2 bytes per voxel in our examples. This amounts to roughly 1.9 MB per million voxels.

If the memory limit is met and a higher visual detail is desired, a higher resolution volumetric model could be mapped to the sampling mesh by applying 3D texturing to its tetrahedra, as in [4], instead of using direct interpolation of the data values stored at their vertices.

## 5. Conclusions and future work

We have presented an algorithm to interactively resample deformable volumetric models onto a regular grid, which can be fed as input to standard direct volume rendering techniques. Our algorithm relies on an implicit sampling mesh, making the resampling process independent of the underlying deformation method.

This independence has been demonstrated in our experiments, and it grants two major advantages over previous approaches: First, a great variety of deformation methods can be coupled with our algorithm by means of a deformation mapping scheme. We have demonstrated the coupling with three deformation methods in this paper. Second, the cost of the resampling step is independent of the deformation method and its resolution, thus allowing the interactive visualization of datasets deformed using dense meshes.

All the stages of our implementation run in parallel using the GPU, and the execution time scales linearly with respect to the size of the input volume dataset. However, the amount of required dedicated memory also scales linearly with respect to the size of the dataset, hence for very large datasets it is possible to reach the memory limits of commodity GPUs.

Our algorithm admits several lines of future work to further enhance its performance and accuracy. They include the use of our regular sampling mesh at medium-high resolution combined with 3D texture mapping, adaptive resampling only in regions where deformation exceeds a threshold, more accurate handling of topological changes, or integration with advanced illumination techniques.

## References


[1] Georgii J, Westermann R. A generic and scalable pipeline for gpu tetrahedral grid rendering. Visualization and Computer Graphics, IEEE Transactions on 2006;12(5):1345–52.

[2] Correa CD, Hero R, Ma KL. A comparison of gradient estimation methods for volume rendering on unstructured meshes. Visualization and Computer Graphics, IEEE Transactions on 2011;17(3):305–19.

[3] Schulze F, Bühler K, Hadwiger M. Interactive deformation and visualization of large volume datasets. In: GRAPP (AS/IE). Citeseer; 2007, p. 39–46.

[4] Gascon J, Espadero JM, Perez AG, Torres R, Otaduy MA. Fast deformation of volume data using tetrahedral mesh rasterization. In: Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation. ACM; 2013, p. 181–5.

[5] Allard J, Courtecuisse H, Faure F. Implicit fem solver on gpu for interactive deformation simulation. GPU Computing Gems Jade Edition 2011;:281–94.

[6] Etheredge C. A parallel mass-spring model for soft tissue simulation with haptic rendering in cuda. In: 15th Twente Student Conference. 2011,.

[7] Kharevych L, Mullen P, Owhadi H, Desbrun M. Numerical coarsening of inhomogeneous elastic materials. ACM Transactions on Graphics (TOG) 2009;28(3):51.

[8] Torres R, Espadero JM, Calvo FA, Otaduy MA. Interactive deformation of heterogeneous volume data. In: Biomedical Simulation. Springer; 2014, p. 131–40.

[9] Nesme M, Kry PG, Jeřábková L, Faure F. Preserving topology and elasticity for embedded deformable models. ACM Transactions on Graphics (TOG) 2009;28(3):52.

[10] Fortmeier D, Mastmeyer A, Handels H. Image-based palpation simulation with soft tissue deformations using chainmail on the gpu. In: Bildverarbeitung für die Medizin 2013. Springer; 2013, p. 140–5.

[11] Rezk-Salama C, Scheuering M, Soza G, Greiner G. Fast volumetric deformation on general purpose hardware. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware. ACM; 2001, p. 17–24.

[12] Westermann R, Rezk-Salama C. Real-time volume deformations. In: Computer Graphics Forum; vol. 20. Wiley Online Library; 2001, p. 443–51.

[13] Chen H, Hesser J, Männer R. Ray casting free-form deformed-volume objects. The Journal of Visualization and Computer Animation 2003;14(2):61–72.

[14] Chen M, Silver D, Winter AS, Singh V, Cornea N. Spatial transfer functions: a unified approach to specifying deformation in volume modeling and animation. In: Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics. ACM; 2003, p. 35–44.

[15] Correa C, Silver D, Chen M. Feature aligned volume manipulation for illustration and visualization. Visualization and Computer Graphics, IEEE Transactions on 2006;12(5):1069–76.

[16] Miranda FM, Celes W. Volume rendering of unstructured hexahedral meshes. The Visual Computer 2012;28(10):1005–14.

[17] Okuyan E, Güdükbay U. Direct volume rendering of unstructured tetrahedral meshes using cuda and openmp. The Journal of Supercomputing 2014;67(2):324–44.

[18] Nakao M, Minato K. Physics-based interactive volume manipulation for sharing surgical process. Information Technology in Biomedicine, IEEE Transactions on 2010;14(3):809–16.

[19] Hacon D, Tomei C. Tetrahedral decompositions of hexahedral meshes. European Journal of Combinatorics 1989;10(5):435–43.

[20] Rössler F, Wolff T, Ertl T. Direct gpu-based volume deformation. Proceedings of Curac 2008;:65–8.

[21] Gibson SF. 3d chainmail: a fast algorithm for deforming volumetric objects. In: Proceedings of the 1997 symposium on Interactive 3D graphics. ACM; 1997, p. 149–ff.

[22] Schill MA, Gibson SF, Bender HJ, Männer R. Biomechanical simulation of the vitreous humor in the eye using an enhanced chainmail algorithm. In: Medical Image Computing and Computer-Assisted Intervention, MICCAI'98. Springer; 1998, p. 679–87.

[23] Frisken-Gibson SF. Using linked volumes to model object collisions, deformation, cutting, carving, and joining. Visualization and Computer Graphics, IEEE Transactions on 1999;5(4):333–48.

[24] Wu J, Westermann R, Dick C. A survey of physically based simulation of cuts in deformable bodies. In: Computer Graphics Forum (to appear). Wiley Online Library; 2015,.

[25] Georgii J, Echtler F, Westermann R. Interactive simulation of deformable bodies on gpus. In: SimVis. 2005, p. 247–58.

[26] Müller M, Gross M. Interactive virtual materials. In: Proceedings of Graphics Interface 2004. Canadian Human-Computer Communications Society; 2004, p. 239–46.