Published in final edited form as:

Int J Numer Method Biomed Eng. 2013 January ; 29(1): 1–16. doi:10.1002/cnm.2489.

An efficient algorithm for mapping imaging data to 3D unstructured grids in computational biomechanics

Daniel R. Einstein^{1,*†}, Andrew P. Kuprat¹, Xiangmin Jiao², James P. Carson¹, David M. Einstein³, Richard E. Jacob¹, and Richard A. Corley¹

¹Systems Toxicology, Pacific Northwest National Laboratory, Richland, WA, U.S.A

²Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY, U.S.A

³Computational Mathematics, University of Massachusetts Lowell, Lowell, MA, U.S.A

SUMMARY

Geometries for organ scale and multiscale simulations of organ function are now routinely derived from imaging data. However, medical images may also contain spatially heterogeneous information other than geometry that are relevant to such simulations either as initial conditions or in the form of model parameters. In this manuscript, we present an algorithm for the efficient and robust mapping of such data to imaging-based unstructured polyhedral grids in parallel. We then illustrate the application of our mapping algorithm to three different mapping problems: (i) the mapping of MRI diffusion tensor data to an unstructured ventricular grid; (ii) the mapping of serial cyrosection histology data to an unstructured mouse brain grid; and (iii) the mapping of computed tomography-derived volumetric strain data to an unstructured multiscale lung grid. Execution times and parallel performance are reported for each case.

Keywords

multiscale modeling; imaging; computed tomography; MRI; histology

1. INTRODUCTION

The spatial arrangement of cells working synergistically to carry out a higher-level function is the very definition of an organ. Cells are the main actors in biology. It is critical, therefore, to be able to relate computed fields—be they mechanical, electrical, or chemical—to the spatial organization of cells and their associated products and structures. Collagen fiber architecture, for example, is a profound determinate of connective tissue mechanics [1, 2]; myofiber architecture strongly influences cardiac function in health and disease [3]; conductive tissues in the heart determine activation patterns [4]; phenotypes of endothelial cells in the respiratory airways vary according to place [5]. These are all examples of

*Correspondence to: Daniel R. Einstein, Biological Monitoring and Modeling, Pacific Northwest National Laboratory, Richland WA, 99352, U.S.A.

†daniel.einstein@pnl.gov

observable spatial patternings that are essential associates with biomedical computations to either establish subject-specific boundary conditions or to parametrize the computation [6, 7].

The problem of mapping volumetric fields from one unstructured grid to another unstructured grid has been extensively studied (e.g., [8–11]). Simply applying these methods to the image-to-grid problem would result in undue computational complexity. However, to date, most approaches to mapping imaging data to unstructured grids have largely been manual [5], ad hoc [12], or confined to a special case [13]. In fact, lack of automated methods has led to the necessity of simplifying assumptions, such that imaging data, when it is available, has often not been used to its fullest potential in computational biomechanics.

These mappings have an importance beyond computation. For example, one might be interested in the spatial colocation of nanoparticles with inflammatory markers for studies of toxicology [14] or drug design [15], for the association of gene expression with brain function and disease [16], for the association of inflammatory markers with a computed shear stress field [17], or for epithelial type with soluble gas concentration [5]. Spatial correlation and data mapping also play an important role in understanding biological variations and morphogenesis.

From a computational perspective, imaging data are piecewise constant fields. For example, *parenchymal strain* is a volume field computed from a nonlinear warp, *cell type* and *gene expression strength* are volume fields determined from reconstructed serial histology; magnetic resonance diffusion tensor data consists of volume tensor fields of *fiber angles*, *fractional anisotropy*, and *apparent diffusion*. These fields are piecewise constant because there is a single value associated with each voxel. Although this data is inherently three dimensional (3D), it exists on a voxelated Cartesian grid, rather than the typically unstructured hexahedral, tetrahedral, or hybrid grid of finite-element models. Here, we focus on the problem of transferring or mapping these fields between images and computational grids. We formulate the problem in such a way that the data could derive from any volumetric imaging data (magnetic resonance, computed tomography, positron emission tomography, optical projection tomography) or indeed any volumetric histology data. That is, we presume no a priori information about either the grid or the image, beyond the obvious constraint that they represent the same geometry. To our knowledge, ours is the first presentation of a general solution to the image-to-grid problem.

2. THEORY

In this section, we present our method for mapping imaging data to an arbitrary computational grid. The approach consists of establishing an efficient, automatic, geometric relationship between a source 3D image (IM) and an arbitrary target unstructured grid (UG) of polyhedra. Here, we assume that the target grid and image occupy approximately the same geometrical space. Thus, we assume that any linear or nonlinear warping necessary to bring image and grid into approximate alignment has been performed upstream of the mapping. We do not assume that the target grid is completely embedded or ‘surrounded’ by the image. Thus, the approach is applicable to images that represent a subset of the grid

geometry or visa-versa. Clearly, there can be no assumption that the image and target grids exactly or even closely share a boundary. Geometry in an image is implicit, whereas the geometry of a computational grid is explicit. Typically, Marching Cubes [18] is applied to the image, to extract the boundaries of the object, resulting in a staircased geometry. Although more advanced methods can produce a relatively smooth boundary [19] from the image, construction of the target grid will typically include several iterations of mesh adaptivity and smoothing. Thus, in the best possible case, there will be a difference between the boundaries of IM and UG that is on the order of the dimensions of one voxel. These differences are to be expected.

2.1. Mapping

In brief, the approach consists of computing a sparse matrix of intersection volumes V_{ij} between the j th voxel of the image and the i th element of the computational mesh in $\mathcal{O}(N)$ time, where N is the number of nonzero V_{ij} . To efficiently find corresponding local voxel neighborhoods of a given mesh element, we first identify those voxels that occupy the same physical space as the minimum bounding box of the polyhedra one at a time. This neighborhood is winnowed to just those voxels that actually intersect the polyhedron by comparing the element coordinates against the planar half spaces represented by the faces of each voxel in the neighborhood. For those voxels that do intersect with the mesh element, the polyhedron of intersection between the voxel half spaces and the mesh element are efficiently computed and added to V_{ij} . For images that hold multiple fields, for example, components of velocity or fiber angle, this operation needs to be only performed once. Thereafter, with the sparse matrix of overlap volumes known, a sparse matrix–vector multiply can be rapidly performed once for each field. It is important to note that the algorithm requires no a priori mapping between the image and the grid. With general biomedical geometries, such an a priori mapping would be clearly impractical if not impossible.

Let $f^{IM}(\mathbf{x})$ be the imaging-based field, where $\mathbf{x} \in \Omega_{IM} = \cup_{j=1}^{n_{IM}} IM_j$ is the imaging domain defined by the voxels IM_j . For now, let us assume that f is a piecewise constant field of floating point values. Special treatment is required when dealing with integer-valued fields such as cell type, which will be discussed next. We wish to create a similar piecewise constant field, $f^{UG}(\mathbf{x})$, on the domain $\mathbf{x} \in \Omega_{UG} = \cup_{i=1}^{n_{UG}} UG_i$, defined by the polyhedra of the computational grid. In general, we are only interested in a subset of IM , roughly corresponding to the geometry of UG . That subset, which we denote the *preimage*, is of course unknown at the outset. In the presentation, for simplicity, we drop any explicit reference to the subset and let the IM stand for the *preimage*. Let Ψ_{IM_j} be the characteristic function for voxel IM_j , which is defined by

$$\Psi_{IM_j}(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} \in IM_j \\ 0, & \mathbf{x} \notin IM_j \end{cases} \quad (1)$$

Similarly, let Ψ_{UG_i} be the characteristic function for the computational element UG_i . In this notation, $f^{IM}(\mathbf{x}) = \sum_j f_j^{IM} \Psi_{IM_j}(\mathbf{x})$ and $f^{UG}(\mathbf{x}) = \sum_i f_i^{UG} \Psi_{UG_i}(\mathbf{x})$. This conveniently defines $f^{IM}(X) \equiv 0, X \notin \Omega_{IM}$ and $f^{UG}(X) \equiv 0, X \notin \Omega_{UG}$. For our map to be exactly conservative,

$$\int_{\Omega_{UG}} f^{UG}(\mathbf{x}) dV = \int_{\Omega_{IM}} f^{IM}(\mathbf{x}) dV. \quad (2)$$

Clearly, if $f^{UG}(\mathbf{x}) = f^{IM}(\mathbf{x}) \forall \mathbf{x}$, this would be satisfied. However, this equality will never be satisfied because an image has a stepped, voxelated boundary, whereas that of the unstructured grid will be typically smooth. Nevertheless, we can force $f^{UG} = f^{IM}$ in the weak sense by requiring

$$\int_{\Omega_{UG}} f^{UG} \Psi_{UG_i} dV = \int_{\Omega_{IM}} f^{IM} \Psi_{UG_i} dV, \quad 1 \leq i \leq n_{UG}, \quad (3)$$

where n_{UG} equations will fix the n_{UG} unknown coefficients f_i^{UG} . From the previous equation, we have

$$\int_{\Omega_{UG}} \sum_k f_k^{UG} \Psi_{UG_k} \Psi_{UG_i} dV = \int_{\Omega_{IM}} \sum_j f_j^{IM} \Psi_{IM_j} \Psi_{UG_i} dV.$$

So,

$$|UG_i| f_i^{UG} = \sum_j |UG_i \cap IM_j| f_j^{IM},$$

where $|IM_j|$ denotes the volume of IM_j (which are all the same), and similarly, $|UG_i|$ denotes the volume of UG_i (which in general are not all the same), and $|UG_i \cap IM_j|$ denotes the volume of the intersection of polyhedron UG_i with voxel IM_j . This implies that we should set

$$f_i^{UG} = \frac{1}{|UG_i|} \sum_j |UG_i \cap IM_j| f_j^{IM} \quad (4)$$

to satisfy Equation (3) for all UG_i . In addition, summing Equation (3) over $1 \leq i \leq n_{UG}$ implies Equation (2). As long as UG is completely embedded in IM —which is a starting assumption because we are letting IM stand for the preimage—and as long as the volumes of

all UG can be exactly computed—as will be the case if all faces of $UG_i \forall i$ are planar—Equation (4) will be exact and will not only be conservative but will also accurately preserve the bounds on the field.

For discrete fields, such as cell type, we have to take a different approach because the result for f_i^{UG} cannot be an average of field values of overlapping voxels. Instead, let $y = f_j^{IM}$ be the integer (or discrete) values in the image; we define the value f_i^{UG} for an element i in the target mesh to be

$$f_i^{UG} \leftarrow \arg \max_y \left(\sum_{j \in f^{-1}(y)} V_{ij} \right). \quad (5)$$

Here, $f^{-1}(y)$ denotes the *preimage* of y , which is the set of voxels j such that $f_j^{IM} = y$. So, f_i^{UG} is set to be the value y whose *preimage* maximally intersects with the i th element of the target mesh.

2.2. Mapping algorithms

An efficient algorithm to map cell-based fields between images and unstructured meshes requires efficient evaluation of the approximate volumes in Equation (4). This task is naturally broken up into two parts:

- **Compute intersections:** Find out for which i, j we have $V_{ij} > 0$. V_{ij} is a sparse matrix: it has far fewer than $n_{UG} \cdot n_{IM}$ nonzero entries. Looping over all $n_{UG} \cdot n_{IM}$ entries would be fatally inefficient and is unnecessary. Instead, we use the structured nature of the Cartesian image to find an approximate neighborhood N_{IM} , a subset of which overlaps with element UG_i . In the case where $|UG_i| \gg |IM_j|$, we determine the inner ball to UG_i and record the intersection volume as the volume of the voxel. We then compute the intersection volumes, $V_{ij} \approx |UG_i \cap IM_j|$, for the remaining voxels in the neighborhood. These intersection volumes will be exact if the elements are planar and approximate if they are nonplanar. Our approach has complexity $\mathcal{O}(N)$ where N is the number of faces corresponding to the nonzero V_{ij} .
- **Mapping Fields:** By using the computed intersection volumes, $V_{ij} \approx |UG_i \cap IM_j|$, map f^{IM} onto f^{UG} . If the f^{IM} is continuous, we apply the mapping given by Equation (4). If f^{IM} is discrete, for example cell type, we apply the mapping given by Equation (5).

2.2.1. Compute Intersections—By having separated the geometric problem of computing V_{ij} from the actual mapping or transfer of the field from the image to the mesh, we focus on the principle challenge of computing the intersection volumes. Because of the structured nature of the image, image object (IM) is simply the image metadata. The mesh

object (UG) is a combination of data and metadata and follows the CFD General Notation System [20] standard for element connectivity. UG may be a single element type, ‘mixed’ type consisting of a hybrid of standard linear element types (tetrahedra, pyramids, prisms, hexahedra), or a polyhedral type consisting of arbitrary polyhedra, with the restriction that all polyhedra must be convex for the mapping to be accurate. The polyhedral type ($Poly$) is instantiated as needed for computing the intersection volumes V_{ij} .

Algorithm 1 gives the ‘outer’ algorithm for determining the overlap volumes of IM . We loop through elements UG_i in UG and for each one call `GETVolsAroundMeshElt` that determines the overlap volumes for UG_i . The overlap volumes V_{ij} and corresponding voxels j are returned as lists, which are appended to $VolList$ in Algorithm 1. $VolList$ consists of the 3-tuples (i, j, V_{ij}) which are the end product of the algorithm. Algorithm 2 begins by calling Algorithm 3 that planarizes any nonplanar faces of UG_i and instantiates it as a $Poly$ type object ($\mathcal{P}_{elt}$) that assumes planar faces. Algorithm 2 then continues by calling `VolmPoly3D` to compute the volume of planarized UG_i .

Algorithm 1

ComputeIntVols

```

1:  ComputeIntVols (image  $IM$ , mean object  $UG$ )
     $IM$ : an image object;  $UG$ : the target mesh object;
    Output  $VolList$ : a list of triplets  $(i, j, V_{ij})$  that for all elements  $UG_i$  in  $UM$  and all voxels in  $IM$  with non-zero
    overlap give overlap volumes  $V_{ij} \approx \text{volume}(UG_i \cap IM_j)$ ;
2:  for  $i \leftarrow 1$  to  $n_{UG}$ /number of elements in  $UG$ 
3:    [ $intVoxels$ ,  $intVols$ ]  $\leftarrow$  GetVolsAroundMeshElt( $UG_i$ )
4:    for  $K \leftarrow 1$  to  $\text{length}(intVoxels)$ 
5:      append  $(i, intVoxels(k), intVols(k))$  to  $VolList$ 
6:    end for
7:  end for
8:  return  $VolList$ ,

```

Algorithm 2

GetVolsAroundMeshElt

```

1:  GetVolsAroundMeshElt( $UG_i$ )
    Output: A list of intersected voxels ( $intVoxels$ ) and corresponding list of intersection volumes ( $intVols$ );
2:   $\mathcal{P}_{elt} \leftarrow \text{PLANARIZATION}(UG_i)$ ; //Instantiate  $Poly$  object  $\mathcal{P}_{elt}$  which is  $UG_i$  with non-planar faces planarized if
    necessary
3:   $volMeshElt \leftarrow \text{VolmPoly3D}(\mathcal{P}_{elt})$ ; //find the volume of  $UG_i$ 
4:  find  $N_{IM}$  the neighborhood of voxels that overlaps with the Cartesian bounding box of  $UG_i$  (This is
    straightforward since voxels are identical and are aligned with Cartesian axes.)
5:  if  $N_{IM}$  consists of only one voxel
6:     $intVoxels(1) \leftarrow voxel$ ;  $intVols(1) \leftarrow volMeshElt$ ;
7:  return
8:  end if

```

```

9:  if  $volMeshElt \gg$  voxel volume
10:    use linear programming to find inscribed ball
11:  for each  $voxel$  inside ball
12:    append  $voxel$  to  $intVoxels$ ;
13:    append scalar equal to voxel volume to  $intVols$ ;
14:    remove  $voxel$  from  $N_M$ 
15:  end for
16: end if
17: #OMP PARALLEL FOR
18: for  $j \leftarrow 1$  to  $size(N_M)$ 
19:    $voxel \leftarrow N_M(j)$ ;
20:    $\mathcal{P}_{int} \leftarrow \mathcal{P}_{elt}$ 
21:    $EmptyIntersection \leftarrow false$ 
   COMMENT: intersect  $\mathcal{P}_{int}$  with the half-spaces  $D_{face}$  of  $voxel$ 
   (where  $voxel = \cap_{face=1}^6 D_{face}$ )
22:   for  $face \leftarrow 1$  to 6;
23:     if  $D_{face} \cap \mathcal{P}_{int} = \mathcal{P}_{int}$ 
24:       continue with next iteration on  $face$ ;
25:     else if  $D_{face} \cap \mathcal{P}_{int} = \emptyset$ ;
26:        $EmptyIntersection \leftarrow true$ ;
27:       break;
   COMMENT:  $PlanePolyInt3D$  computes intersection of the polyhedron
    $\mathcal{P}_{int}$  with the half-space bounded by the plane of the voxel face,
    $\mathcal{P}_{tmp} = \mathcal{P}_{int} \cap D_{face}$ ,  $\mathcal{P}_{tmp}$  is a new truncated  $Poly$  object.
28:    $\mathcal{P}_{tmp} \leftarrow PlanePolyInt3D(D_{face}, \mathcal{P}_{int})$ ;
29:   if  $\mathcal{P}_{tmp} = \emptyset$ 
30:      $EmptyIntersection \leftarrow true$ ;
31:     break;
32:   else;
33:      $\mathcal{P}_{int} \leftarrow \mathcal{P}_{tmp}$  //update the polyhedron with the latest truncation
34:
35:   if  $\neg EmptyIntersection$ 
36:      $volIntPoly \leftarrow VolmPoly3D(\mathcal{P}_{int})$ ;
37:     append  $voxel$  to  $intVoxels$ ; append scalar  $volIntPoly$  to  $intVols$ ;
38: #END OMP PARALLEL FOR

```

Algorithm 3

Planarize_poly

```

1: Planarize_poly ( $UG_i$ )
   Output: a  $Poly$  object  $\mathcal{P}_{elt}$  equal to  $UG_i$  with non-planar faces planarized if necessary

```

```

2:  if  $UG_i$  known to have all planar faces (e.g. tetrahedron)
3:       $\mathcal{P}_{elt} \leftarrow \text{Load\_Poly}(UG_i)$  // Instantiates Poly object
4:      return
5:  end if
6:   $\mathcal{P}_{elt} \leftarrow \text{Load\_Hex}(BB^+(UG_i))$  // Instantiate a Poly object that is a hexahedron slightly larger than the Cartesian
    bounding box of  $UG_i$ 
7:  for each face in  $\mathcal{P}_{elt}$ 
8:      if number of points in face is 3
9:          compute plane  $\mathcal{D}_{face}$  from points;
10:     else if number of points in face is 4
11:         construct plane  $\mathcal{D}_{face}$  from the edge midpoints;
12:     else;
13:         construct least squares plane  $\mathcal{D}_{face}$  from face points
14:      $\mathcal{P}_{elt} \leftarrow \text{PlanePolyInt3D}(\mathcal{D}_{face}, \mathcal{P}_{elt});$ 

```

By using the fact that voxels are identical and are aligned with Cartesian axes, we compute the neighborhood of voxels N_{IM} that overlap with the Cartesian bounding box of UG_i . If N_{IM} consists of a single voxel, then UG_i is entirely within that voxel, and we return information on the single overlap, which is the identity of the overlapping voxel and the volume of overlap. If N_{IM} is more than one voxel and the volume of UG_i is much greater than the voxel volume, say by two orders of magnitude, we use linear programming in the C Double Description (CDD) Library [21] to calculate the Chebychev center and radius of the inner ball of UG_i [22]. Voxels that are entirely inside the inscribed ball are omitted from N_{IM} because their intersection volumes are trivially equal to the voxel volume (Algorithm 2, lines 9–16).

Depending on the shape and orientation of UG_i , all or few of the remaining candidate voxels will intersect with $\mathcal{P}_{elt}$. For example, if UG_i is a tetrahedron on average roughly 1/5 of the candidate voxels, minus those inside the inner ball, will intersect with it. We wish to further winnow these null voxels quickly and efficiently. We therefore determine if all of the nodes of UG_i are on the wrong side of any voxel half space D_{face} . Those voxels that remain in the list are guaranteed to intersect with UG_i . For these voxels, $\mathcal{P}_{int}$ is copied from $\mathcal{P}_{elt}$ and is successively updated by PlanePolyInt3D , which determines that portion of $\mathcal{P}_{int}$ that lies behind the voxel half space. After $\mathcal{P}_{int}$ is chiseled by all six faces of the voxel, VolmPoly3D then computes and records its volume in VolList . This is volume $V_{ij} \approx |UG_i \cap IM_j|$.

To planarize UG_i prior to computing the intersection (Algorithm 3), a Cartesian bounding box $\mathcal{P}_{hex}$ that is slightly larger than the minimum bounding box of $\mathcal{P}_{elt}$ is intersected with the planarized faces of $\mathcal{P}_{elt}$. If UG_i is a tetrahedron, clearly no planarization is needed. For all other polyhedra, the plane is constructed from the three points of a triangular face, the midpoints of the face edges of a quadrilateral face, or the least squares plane of a face with more than four points. The construction of a plane from the midpoint of the edges of a quadrilateral face is possible if the faces is assumed bilinear because in this case, the midpoints all lie on a single plane. Although it would be possible to determine element-by-element if planarization is needed, in a biological mesh, few, if any, nontetrahedral elements

will be planar. We therefore planarize all nontetrahedral elements. It is also important to note that in the case of nonconvex polyhedra, planarization can introduce an error. However, except in the most pathological cases, these errors will be on the order of the error introduced by the planarization of convex polyhedra. Moreover, severely nonconvex elements are bad engineering practice and introduce a much more serious discretization error and therefore should be avoided. In fact, depending on the application and the element design, planarization can be considered optional.

Algorithm 4

PlanePolyInt3D

```

1: Plane_polyInt3D ( $\mathcal{D}_{face}, \mathcal{P}_{in}$ )
   Output:  $\mathcal{P}_{out}$  is a version of  $\mathcal{P}_{in}$  that has been clipped by  $\mathcal{D}_{face}$ 
2: store corners in  $\mathcal{P}_{in}$  that are on or behind  $\mathcal{D}_{face}$  in  $\mathcal{P}_{out}$ 
3: for each edge in  $\mathcal{P}_{in}$ 
4:   if both corners of edge are behind  $\mathcal{D}_{face}$ 
5:     store existing corners in  $\mathcal{P}_{out}$ 
6:   else if only one is behind  $\mathcal{D}_{face}$ 
7:     store corner in  $\mathcal{P}_{out}$ 
8:     compute intersection of edge with  $\mathcal{D}_{face}$ 
9:     store both new corner and new edge in  $\mathcal{P}_{out}$ 
10: for each face in  $\mathcal{P}_{in}$ 
11:   if all or part of a face in  $\mathcal{P}_{in}$  is behind  $\mathcal{D}_{face}$ 
12:     assign partial face to  $\mathcal{P}_{out}$ 
13:     assign new edges to existing faces
14:   create new face in  $\mathcal{P}_{out}$  on  $\mathcal{D}_{face}$  with new edges
15: return  $\mathcal{P}_{out}$ 

```

The remaining algorithms to be described are VolmPoly3D and PlanePolyInt3D. VolmPoly3D takes advantage of the fact that *Poly*-type polyhedron is assumed to have planar faces and uses a well-known geometric formula to compute the volume of the planar polyhedron. PlanePolyInt3D, however, is considerably more involved, and its efficient implementation is important to the efficient implementation of the overall algorithm. However, as can be seen in Algorithm 4, it basically amounts to determining first which vertices are behind the half plane and then intersecting their edges. Finally, the faces of intersection are reconstructed.

2.3. Parallel implementation

Because Algorithm 1 considers one element at a time from UG , and because Algorithm 2 generates a list of potential candidates in IM from a simple hash operation, the overall approach can clearly be made parallel. The structure of the overall algorithm suggests a possible nested approach with the main loop in Algorithm 1 as the outer level and the nested loop indicated in Algorithm 2 with an OpenMP pragma as the inner level. In this article, for simplicity, we will only show the latter of these two, as indicated in Algorithm 2. We chose

OpenMP for the parallel implementation rather than message passing because the application is likely to be on multicore workstations and because, as we will show, the execution time for practical-sized problems is reasonable. However, it is clear that more aggressive attempts at parallelism are possible.

3. EXAMPLES AND DISCUSSION

In this section, we demonstrate the application of our algorithm on a series of mapping problems. We first consider an artificial symmetric image pattern with spherical meshes of various sizes. Following that, we consider the (i) mapping of cardiac diffusion tensor data to an unstructured tetrahedral grid of a heart; (ii) the mapping of mouse brain data to a tetrahedral grid of the mouse brain; and (iii) the mapping of computed volumetric strain, or regional ventilation, to a multiscale grid of the rat lung. Results are presented for both serial and parallel implementations. All computations were performed on a Dell Precision T7500n workstation with 2 Quad Core 2.40 GHz Intel Xeon Processors with 24 GB shared memory.

3.1. Sample sphere

The focus of this sample problem is to illustrate the performance of the algorithm on a synthetic image (Figure 1) whose features and symmetry are easily recognized. Table I reports the computation times for four different unstructured meshes: three tetrahedral meshes of order-of-magnitude size differences and one hybrid prism-tetrahedral mesh. All times are for computations on a single thread.

Note that the integrals for the mesh Equation (3) are slightly smaller than those of the overlapped voxels in the image. This is expected and indeed necessary because the overlap voxels of the image are voxelated, not boundary fitted, and will, of necessity, extend beyond the geometry represented by the unstructured grid. Table I also reports the sizes of the grid cells as well as the cells and faces in the sparse matrix. Clearly, the geometric fidelity of the mapped field is a function of the size and orientation of the grid cells with respect to the image features and gradients. The subject of how to construct appropriate meshes to capture interesting image features is beyond the scope of this work.

3.2. Cardiac diffusion tensor data

Cardiac diffusion tensor imaging (DTI) is a high-resolution noninvasive MRI modality for measurement of myocardial (and brain) fiber structure and geometry [23]. The method is based on the assumption that water diffusion is highest along the axis of myocardial fiber, and therefore, the primary eigenvector of the diffusion tensor coincides with the local fiber orientation. Because the myocardium is anisotropic, computational models of the ventricular mechanics must include the local fiber angle [4]. Historically, computational studies of ventricular mechanics have imposed idealizations of the fiber angle orientation and assumed that the fiber angles range from 60° at the epicardium to 60° at the endocardium [12]. Mapping the measured DTI data directly from the imaging data to the computational grid represents a compelling alternative because it enables the true spatial variation in fiber angles to govern ventricular mechanics. Once mapped, these quantities can serve the role of initializing a computational analysis or validating computational predictions. The fiber

angle, for example, must be assigned for each computational cell to prescribe local material anisotropy.

Moreover, accounting for the fiber angle or principle eigenvector alone leaves much mechanical information out. The second and third eigenvectors and the relative magnitudes of the three eigenvalues also have mechanical meaning. Myofibers, such as collagen, tends to be splayed. In other words, at a material point, the orientation is strongly in one direction, but fibers in the sheet and so-called cross-fiber directions exist as well. Thus, the constitutive behavior of myocardium is not purely transversely isotropic. Indeed, the relative magnitude of the three eigenvalues of the diffusion tensor vary spatially. This behavior typically gets absorbed into material parameters of phenomenological models of myocardial stress–strain behavior, but we have shown that it can be used to parametrize the anisotropy matrix for a hyperelastic model of ventricular contraction in a spatial heterogeneous fashion [6].

All *ex vivo* DTI imaging data was graciously provided by Dr. Edward Hsu of the University of Utah. The data consisted of an intensity image to be segmented for the image geometry (Figure 2(A)), the nine components of the three eigenvectors of the diffusion tensor, and the three eigenvalues, for a total of 13 images with a resolution of $96 \times 96 \times 128$ for each image. The intensity image was automatically segmented by applying the Power Watershed algorithm [24], with two seed points, one for the tissue and one for the background (Figure 2(A)). A triangulated surface was extracted by applying a variant of the Marching Cubes algorithm [18] to the segmented image. The triangulated mesh was adapted to the gradient-limited feature size [25], and subsequently, a layered tetrahedral mesh was generated in BioGeom (simtk.org/home/biogeom).

The geometric mapping between the images and the unstructured grid (Figure 2(B)) was computed for a single image and applied to the remaining 11. This is one of the advantages of creating a geometric mapping—it only needs to be done once. Table II reports the overlap statistics. Figure 3 reports the wall clock time for 1, 2, 4, and 8 threads. Figure 2(C) and (D) shows the first eigenvector, the most commonly recognized component, of the mapped diffusion tensor data.

3.3. Mouse brain gene expression data

In this example, we map murine cell density data from serial histology to an unstructured grid. The original data consisted a stack of 350 nissl-stained images acquired by cyrosectioning coronally a single frozen adult mouse brain. Each image was size 850×670 pixels at a resolution of $15 \mu\text{m}$ per pixel. A total of 175 slices each $25 \mu\text{m}$ thick were collected and stained according to a protocol described in [26]. The serial sections were reconstructed into a 3D volumetric representation of the cell density for the mouse brain from serial sections by applying a warp filtering described in [27]. Subsequently, the reconstructed volume was resampled to have an isotropic resolution. Specifically, the reconstructed image was interleaved with interpolated slices. Segmentation and unstructured grid generation were similar to the methods described previously.

Cell density was mapped onto an anisotropic tetrahedralized representation of the brain volume (Figure 4). Colors correspond to local the density of cells demonstrating high levels

of probed gene transcript. The signal seen here is predominately in the Purkinje cells of the cerebellum and the dentate gyrus of the hippocampus. Table III reports the overlap statistics. Figure 5 reports the wall clock time for 1, 2, 4, and 8 threads.

Anisotropic and nonuniform meshing allows finer representation of anatomical detail in regions where the signal changes rapidly over space. This allows important details to be retained while minimizing computational resources for large datasets. The result in this case is a grid in which the maximum cell volume is four orders of magnitude larger in terms of volume than the voxel volume. To make the processing of the intersection volumes more efficient where the grid size is much larger than the voxel size, we determine the maximum-inscribed ball. Solution of the maximum-inscribed ball for general convex polyhedra is efficiently computed with the double description method implemented in the CDD/CDD+ library [21, 22]. This approach keeps the processing time acceptably low (Figure 5), despite the vast number of overlap faces in the sparse matrix (Table III).

Clearly, processing times are a function of the relative size and orientations of the computational grid and the image voxel. In the second example (Section 3.2), the voxels and the computational cells were of the same order of magnitude, requiring that the intersection volumes of each overlap voxel be computed. In the case of the mouse brain data (Section 3.3), the relative size of voxel and computational cell varied greatly, and the intersection volumes of many voxels were trivially computed. This example also illustrates how care must be taken when constructing the grid to assure the resolution of the desired image features. The decision concerning which image features to support in the computation is largely a function of the application.

3.4. Multiscale rat lung

Historically, computational models of the respiratory system have been unable to account for mechanical variation in the lung. However, local deviations from nominal heterogeneity and compliance in disease states such as emphysema and fibrosis have both important clinical and pathological implications [28]. Moreover, by altering site-specific flow, these deviations are likely to influence the metabolism and therefore toxicity of soluble gasses and particulates [5]. Recently, estimations of regional ventilation based on differences in multidetector row computed tomography, a volumetric field, have been incorporated into multiscale models of lung respiration [29–31]. In this example, we take a philosophically similar approach. In contrast, however, we map 3D volumetric strain from a target image to a reference image, with the goal of prescribing initial and boundary conditions for a multiscale model of rodent.

In brief, 3D deformation fields were calculated between a pair of micro CT images from a live mechanically ventilated rat [32] by nonlinearly warping a target image to a reference image [33, 34]. A total of 11 images were acquired over the breathing cycle. Here, for simplicity, we focus on two images, one at $t = 0$ ms and one at $t = 80$ ms, corresponding to a change in input pressure of $\approx 2\text{cmH}_2\text{O}$. From these deformation maps, 3D volumetric strain was computed by solving for nonlinear strain on the basis of a finite-element discretization. A Eulerian strain description was chosen such that all strain values would be consistently referred to the reference configuration [35]. Figure 6 shows the computed field. The

parenchymal volume was segmented from the live image by applying the power watershed algorithm to the image. The triangulated surface was then tessellated with polyhedra suitable for simulation with the finite-volume method.

Table IV reports the overlap statistics. The mesh consisted of 150,778 polyhedra with 873,084 points and 1,023,846 faces. This example shows the performance of the algorithm with arbitrary polyhedra. Figure 7 reports the wall clock time for 1, 2, 4, and 8 threads. Note that even though the number of overlap faces is fewer than in the previous example, the problem is more computationally expensive. Because the size of the polyhedra in this example is all on the same order of magnitude with respect to the image voxels, each intersection pair must be computed, and there is no significant savings from solving the linear programming problem to find the maximally inscribed ball. Nevertheless, given the number of faces in the problem, the wall clock time is reasonable.

4. LIMITATIONS AND FUTURE WORK

Because organs are by definition spatially heterogeneous arrangements of cells, mechanical properties with efficient continuum representations such as tissue elasticity or mass transfer emerge in a spatially heterogeneous fashion from the cellular and extracellular constituents of tissue. We have presented an efficient and robust method for mapping volume fields on the basis of cells or their intermediate scale properties or extracellular constituents from images to general imaging-derived computational grids. However, in some applications, a volume-to-surface mapping may be more appropriate. Some examples are the association of the expression of inflammatory markers with a computed shear stress field [17] or epithelial type with soluble gas concentration [5]. Because all imaging data are inherently volumetric, however, the mapping of volume fields is a necessary precursor to a volume-to-surface mapping.

Another limitation of the presented algorithm is that the approach to parallelism relies on OpenMP and thus is only applicable to multithreaded and shared memory machines. More aggressive parallelism, including the use of domain decomposition and message passing are of course possible. Such an approach will be attractive as the size of our images grows to be multiple terabytes. Nevertheless, the presented algorithm is efficient on common workstations and, moreover, inherently parallel. Thus, no redesign would be necessary to pursue such strategies.

On a related note, more work needs to be performed to improve and speed up the reconstruction and processing, including segmentation and registration, of huge images, particularly with regard to images derived from serial histology where resolutions can be on the subcellular scale. Our work in this area is ongoing.

4.1. Extension to mapping to piecewise linear fields

A more important limitation of the presented method is that the piecewise constant image function f^M is mapped to a piecewise constant function f^{UG} on the mesh. If element size in UG is large with respect to variations in f^M , the piecewise interpolation could be rough. In the case where the mapped field is an integer type, such as cell type, this misrepresentation

is inevitable, and the only recourse is to modify the mesh density in areas of rapid variations. However, in the case of a continuous field, a piecewise linear interpolation could be more accurate and appropriate. Next, we discuss, but do not implement, the case where the function f^{UG} is to be represented as a piecewise linear function on an unstructured mesh with image function f^M , still assumed piecewise constant.

Let $f^{UG} = \sum_j f_m^{UG} \Phi_{UG_m}$, where the function Φ_{UG_m} is the familiar piecewise linear *hat* function, which is one at node m in the unstructured mesh and is zero at all other nodes. Similar to Equation (3), we can again force $f^{UG} = f^M$ in the weak sense by requiring

$$\int_{\Omega_{UG}} f^{UG} \Phi_{UG_m} dV = \int_{\Omega_{IM}} f^{IM} \Phi_{UG_m} dV, \quad 1 \leq m \leq m_{UG}, \quad (6)$$

where m_{UG} equations will fix the m_{UG} unknown coefficients f_m^{UG} , where m ranges over the m_{UG} nodes in the unstructured mesh. From the above equation, we have

$$\int_{\Omega_{UG}} \sum_p f_p^{UG} \Phi_{UG_p} \Phi_{UG_m} dV = \int_{\Omega_{IM}} \sum_j f_j^{IM} \Psi_{IM_j} \Phi_{UG_m} dV.$$

This leads to a linear system

$$M_{mp} f_p^{UG} = L_{mj} f_j^{IM}, \quad (7)$$

where $M_{mp} = \int_{\Omega_{UG}} \Phi_{UG_m} \Phi_{UG_p} dV$ is the sparse *mass matrix* of inner products of piecewise linear basis functions over the unstructured mesh and where $L_{mj} = \int_{\Omega_{IM}} \Phi_{UG_m} \Psi_{IM_j} dV$ is the sparse matrix consisting of the integral of the products of the unstructured piecewise linear Φ_{UG_m} and the image piecewise constant Ψ_{IM_j} . From this, we see that computation of L_{mj} would be very similar to computation of V_{ij} in Algorithm 2 because it requires finding the voxels in the image that overlap a given element in the unstructured mesh.

In contrast to the piecewise constant case, however, the target in the unstructured mesh would not be element m but rather any element i that has node m as a vertex. As overlapping volumes between the image voxels and unstructured grid are found, it would then be necessary to integrate a known linear function Φ_{UG_m} over the intersection volume between element i and voxel j . This is nearly as straightforward as simply finding the intersection volume, as was done in the piecewise constant case, because it is straightforward to decompose the intersection polyhedron into constituent tetrahedra. Specifically, the intersection polyhedron can be decomposed into $2E$ tetrahedra, where E is the number of edges, simply by connecting the vertices of each edge to one centroid of an adjacent face and to the centroid of the polyhedron. With the polyhedron decomposed into tetrahedra, the

integration of a linear function is easily *exactly* computable [36]. Thus, computation of L_{mj} would be reduced to a minor modification of Algorithms 1 and 2. In fact, the outer loop in Algorithm 1 could still run over the elements i in the unstructured grid. In this case, we would compute the contribution to L_{mj} for each node m that belongs to element i within the outer loop, such that after the loop is finished, all the summed contributions would give L_{mj} .

The mass matrix M_{mp} in the unstructured mesh is readily computed as per standard finite-element techniques, and the system (7) would be solved as a sparse matrix system by standard techniques. It is anticipated that solution of the sparse system would be cheaper than computation of the overlap matrix L_{mj} . However, by lumping the mass matrix, one could eliminate the cost of the sparse system solution altogether. Unfortunately, mass matrix lumping introduces a diffusion effect on mapped quantities [9] and thus is not recommended.

5. CONCLUSIONS

We have presented an efficient and general method for mapping volume fields from imaging data to organ scale and multiscale computational grids. The method is general in the sense that no a priori information about the computational grid or the image is required for the mapping. In Section 2, we demonstrated that the resulting mapping is conservative and preserves the minima and maxima of the field. We illustrated this with cardiac diffusion tensor data, where the mapped normalized eigenvectors were shown to range between -1 and 1 . This example also illustrated that once the geometric mapping is established, all 13 components of the dataset could be mapped efficiently with the same mapping. We then illustrated the parallel performance of the method in three relevant areas of multiscale modeling: the heart, the brain, and the lung.

Acknowledgments

We gratefully acknowledge Dr. Edward Hsu for the cardiac DTI data. We also thank Stewart J. Mosso for providing the FORTRAN 90 subroutines PLANEPLYINT3D and VOLMPOLY3D. This work was financially supported by the National Institutes of Health Bioengineering Research Partnership Grant R01-HL073598 (Richard A. Corley, PI) and by DOE LDRD 90001 (Kerstin Kleese-Van Dam, PI).

References

1. Weis SM, Emery JL, Becker D, McBride DJ, Omens JH, McCulloch AD. Myocardial mechanics and collagen structure in the osteogenesis imperfecta murine (oim). *Circulation Research*. 2000; 87(8):663–669. [PubMed: 11029401]
2. Sacks MS, Chuong CJ, More R. Collagen fiber architecture of bovine pericardium. *American Society for Artificial Internal Organs Journal*. 1994; 40(3):632–637.
3. Hsu EW, Healy LJ, Einstein DR, Kuprat AP. Imaging-based assessment and modeling of the structures of the myocardium. In: Guccione JM, Kassab GS, Ratcliffe MB, editors *Computational Cardiovascular Mechanics – Modeling and Applications in Heart Failure*. Springer; New York: 2009. 23–40. Published, 2009
4. Glass L, Hunter PJ, McCulloch AD. *Theory of Heart: Biomechanics, Biophysics, and Nonlinear Dynamics of Cardiac Function*. Springer-Verlag; New York: 1991.
5. Carey SA, Minard KR, Trease LL, Wagner JG, Garcia GJM, Ballinger CA, Kimbell JS, Plopper CG, Corley RA, Postlethwait EM, et al. Three-dimensional mapping of ozone-induced injury in the nasal

airways of monkeys using magnetic resonance imaging and morphometric techniques. *Toxicologic Pathology*. 2007; 35(1):27–40. [PubMed: 17325970]

6. Einstein DR, Del Pin F, Jiao X, Kuprat AP, Carson JP, Kunzelman KS, Cochran RP, Guccione JM, Ratcliffe MB. Fluid-structure interactions of the mitral valve and left heart: comprehensive strategies, past, present and future. *International Journal for Numerical Methods in Engineering*. 2010; 26(348–380):348–380. [PubMed: 20454531]
7. Tawhai M, Bischoff J, Einstein D, Erdemir A, Guess T, Reinbolt J. Multiscale modeling in computational biomechanics. *IEEE Engineering in Medicine and Biology Magazine*. 2009; 28(3): 41–49.
8. Jaiman RK, Jiao X, Geubelle PH, Loth E. Conservative load transfer along curved fluid-solid interface with non-matching meshes. *Journal of Computational Physics*. 2006; 218(1):371–397.
9. Farrell PE, Maddison JR. Conservative interpolation between volume meshes by local galerkin projection. *Computer Methods in Applied Mechanics and Engineering*. 2009; 198(33–36):2632–2642.
10. Kuprat A, Mosso S. Technical Report LA-UR-055084. Los Alamos National Laboratory; 2005. Efficient algorithms for mapping cell quantities between overlapped 3D unstructured meshes.
11. Menon S, Schmidt DP. Conservative interpolation on unstructured polyhedral meshes: an extension of the supermesh approach to cell-centered finite-volume variables. *Computer Methods in Applied Mechanics and Engineering*. 2011; 200(41–44):2797–2804.
12. Sun K, Zhang Z, Suzuki T, Wenk JF, Stander N, Einstein DR, Saloner D, Wallace AW, Guccione JM, Ratcliffe MB. Dor procedure for dyskinetic anteroapical myocardial infarction fails to improve contractility in the border zone. *Journal of Thoracic and Cardiovascular Surgery*. 2010; 140(1):233–239. [PubMed: 20299030]
13. Wang Y, Gu X, Thompson PM, Yau S. Medical Imaging Computing and Computer Assisted Intervention (MICCAI). St. Malo, France: Sep, 2004 3D harmonic mapping and tetrahedral meshing of brain imaging data.
14. Jacobsen NR, Møller P, Jensen KA, Vogel U, Ladefoged O, Loft S, Wallin H. Lung inflammation and genotoxicity following pulmonary exposure to nanoparticles in apoe^{-/-} mice. *Part Fibre Toxicol*. 2009; 12(6):2–12.
15. Suri SS, Fenniri H, Singh B. Nanotechnology-based drug delivery systems. *Journal of Occupational Medicine and Toxicology*. 2007; 1(2):16–22.
16. Carson JP, Eichele G, Chiu W. A method for automated detection of gene expression required for the establishment of a digital transcriptome-wide gene expression atlas. *Journal of Microscopy*. 2005; 217(3):275–281. [PubMed: 15725131]
17. Sucosky P, Balachandran K, Elhammali A, Jo H, Yoganathan AP. Altered shear stress stimulates upregulation of endothelial vcam-1 and icam-1 in a bmp-4- and tgf- β 1-dependent pathway. *Arteriosclerosis, Thrombosis, and Vascular Biology*. 2009; 29:254–260.
18. Lorensen WE, Cline HE. Marching cubes: a high resolution 3d surface construction algorithm. *Computer Graphics*. 1987; 21:163–169.
19. Chernyaev EV. Technical Report Cern CN 95–17. CERN; 1995. Marching cubes 33: construction of topologically correct isosurfaces.
20. CGNS Team. Technical Report AIAA R-101A-2005. The American Institute of Aeronautics and Astronautics; 2005. AIAA recommended practice for the cfd general notation system—standard interface data structures.
21. Fukuda K. [Accessed May, 2011] CDD/CDD+ library. (Available from: www.ifor.math.ethz.ch/~fukuda/cdd_home/)
22. Fukuda K, Prodon A. Double description method revisited. In: Deza M, Euler R, Manoussakis I, editors *Combinatorics and Computer Science*. Vol. 1120. Springer; Berlin/Heidelberg: 1996. 91–111. *Lecture Notes in Computer Science*
23. Jiang K, Pandya Y, Smithies O, Hsu EW. Three-dimensional diffusion tensor microscopy of fixed mouse hearts. *Magnetic Resonance in Medicine*. 2004; 52(3):453–460. [PubMed: 15334561]
24. Couprie C, Grady L, Najman L, Talbot H. Power watershed: a unifying graph-based optimization framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2011; 33(7):1384–1399. [PubMed: 21079274]

25. Kuprat AP, Einstein DR. An anisotropic scale-invariant unstructured mesh generator suitable for volumetric imaging data. 2009; 228:619–640.
26. Carson JP, Thaller C, Eichele G. A transcriptome atlas of the mouse brain at cellular resolution. *Current Opinion in Neurobiology*. 2002; 12(5):562–565. [PubMed: 12367636]
27. Ju T, Warren J, Carson J, Bello M, Kakadiaris I, Chiu W, Thaller C, Eichele G. 3D volume reconstruction of a mouse brain from histological sections using warp filtering. *Journal of Neuroscience Methods*. 2006; 156(1–2):84–100. [PubMed: 16580732]
28. Robertson HT, Krueger MA, Lamm WJ, Glenny RW. High-resolution spatial measurements of ventilation-perfusion heterogeneity in rats. *Journal of Applied Physiology*. 2010; 108:1395–1401. [PubMed: 20203067]
29. Yin Y, Choi J, Hoffman E, Tawhai MH, Lin CL. Simulation of pulmonary air flow with a subject-specific boundary condition. *Journal of Biomechanics*. 2010; 43(11):2159–2163. [PubMed: 20483412]
30. Tawhai M, Burrowes K. Multi-scale models of the lung airways and vascular system. *Advances in Experimental Medicine and Biology*. 2008; 605:190–194. [PubMed: 18085270]
31. Lin CL, Tawhai M, McLennan G, Hoffman E. Computational fluid dynamics. *IEEE Engineering in Medicine and Biology Magazine*. 2009; 28(3):25–33.
32. Jacob RE, Lamm WJ. Stable small animal ventilation for dynamic lung imaging to support computational fluid dynamics models. *PLoS One*. 2011; 6(11):e27577.doi: 10.1371/journal.pone.0027577 [PubMed: 22087338]
33. Shackleford JA, Kandasamy N, Sharp GC. On developing b-spline registration algorithms for multi-core processors. *Physics in Medicine and Biology*. 2010; 55(21):6329–6351. [PubMed: 20938071]
34. Sharp G. [Accessed June 2011] Plastimatch. 2011. (Available from: <http://plastimatch.org/>)
35. Freed A, Einstein D. Hypo-elastic model for lung parenchyma; *Biomechanics and Modeling in Mechanobiology*. 1–17.
36. Brenner S, Scott L. *The Mathematical Theory of Finite Element Methods*. Springer-Verlag; New York: 1994.

Figure 1.

(A) cross-section of image; (B) grid superimposed on image for size comparison; (C) cross-sections of a tetrahedral grid at two-order of magnitude resolutions and a hybrid prism-tetrahedral grid. The image consisted of 1 million voxels.

Figure 2.

Mapping of cardiac diffusion tensor data. (A) Intensity data from which geometry was extracted. Arrows show the two seed points for automatic segmentation. (B) Cut-away of the resulting layered unstructured grid. (C) Vectors of the first eigenvector of the diffusion tensor mapped to the unstructured grid. Magnitudes are the first eigenvalue. (D) Stream traces of the first eigenvector of the diffusion tensor mapped to the unstructured grid. Magnitudes are the first eigenvalue.

Figure 3.
Parallel performance for the mapping of 12 image components to the unstructured grid of the heart.

Figure 4.

Mapping nissl stain and cell density data from serial histology of the mouse brain. (A) Reconstructed image volume of the nissl-stained mouse brain. Inset is a digital image of one nissl-stained slide. (B) Reconstructed image and mesh surface. (C) Mapped nissl stain of the dentate gyrus region of the hippocampus on a cut-away of the tetrahedral grid. (D) Mapped cell density signal of the dentate gyrus region of the hippocampus on a cut-away of the tetrahedral grid. Colors correspond to local density of cells. The signal seen here is predominately in the Purkinje cells of the cerebellum and the dentate gyrus of the hippocampus.

Figure 5.
Parallel performance for the mapping of the cell density data from the reconstructed serial cyrohistology data.

Figure 6.

Mapping of Eulerian volumetric strain ($\frac{dv}{dV}$), computed from live images of a mechanically ventilated rat. A total of 11 images were acquired during the breathing cycle. Displacement images were computed from nonlinear warps between images. A finite element-based calculation was used to determine volumetric strain of the image and mapped to a polyhedra volume mesh. Polyhedra are more accurate and more efficient than tetrahedra in a finite-volume calculation. (A) Superposition of image and polyhedral mesh. (B) Mapped field with detail showing the edges of the polyhedra. (C) Vertical cut-away through the mesh showing the nonuniform nature of lung expansion.

Figure 7.
Parallel performance for the mapping of the of volumetric strain computed from live computer tomography-imaging of a Sprague–Dawley rat to a finite-volume mesh consisting of arbitrary polyhedra.

Table I

Output from the mapping illustrated in Figure 1.

	Min volume	Max volume	Min value	Max value	Overlap cells	Overlap faces	Integral	Time (s)
Image	1.29E-04	1.29E-04	0.0	52.7	N/A	N/A	81.54	N/A
Grid 1	2.48E-04	1.90E-02	13.1	31.0	133,794	810,140	75,080	6.2
Grid 2	8.97E-05	2.51E-03	13.1	31.0	208,293	1,273,382	75,080	8.4
Grid 3	6.93E-06	1.27E-03	12.83	32.1	593,179	3,789,234	75,080	18.6
Grid 4	1.83E-06	1.96E-03	12.87	32.1	456,067	2,981,122	75,080	10.9

Table II

Output from the mapping illustrated in Figure 2.

	Min volume	Max volume	Min value	Max value	Overlap cells	Overlap faces
Image	1.0	1.0	-1.0	1.0	N/A	N/A
Grid	3.6E-05	12.42	-1.0	1.0	2,308,550	14,588,192

Minimum and maximum values are for all of the components of the three eigenvectors.

Table III

Output from the mapping illustrated in Figure 4.

	Min volume	Max volume	Min value	Max value	Overlap cells	Overlap faces
Image	1.0	1.0	0	255	N/A	N/A
Grid	0.0	2.74E+04	0	255	175,140,543	1,068,054,494

Note the vast difference in scale between the maximum volume in the grid and the volume of the voxel. Efficient processing of the voxels contained in the maximum-inscribed ball determined by the linear programming solution to the Chebychev center keeps the problem tractable.

Table IV

Output from the mapping illustrated in Figure 6.

	Min volume	Max volume	Min value	Max value	Overlap cells	Overlap faces
Image	1.0	1.0	9.4036E-01	1.0381E+00	N/A	N/A
Grid	4.432E-04	6.496E+01	9.4032E-01	1.0381E+00	15,774,955	98,076,230

Note that the size of the vast majority of the polyhedra are on the same order of magnitude as that of the voxel. Thus, the solution of the nonlinear programming problem to find the maximum-inscribed ball does not afford the same economy as seen in Figure 4. Nevertheless, the computational efficiency is reasonable.