Tetrahedral Mesh Modeling of Density Data for Anatomical Atlases and Intensity-Based Registration

Jianhua Yao, Russell Taylor

Computer Science Department, The Johns Hopkins University, Baltimore, MD

Abstract. In this paper, we present the first phase of our effort to build a bone density atlas. We adopted a tetrahedral mesh structure to represent anatomical structures. We propose an efficient and automatic algorithm to construct the tetrahedral mesh from contours in CT images corresponding to the outer bone surfaces and boundaries between compact bone, spongy bone, and medullary cavity. We approximate bone density variations by means of continuous density functions in each tetrahedron of the mesh. Currently, our density functions are second degree polynomial functions expressed in terms of barycentric coordinates associated with each tetrahedron. We apply our density model to efficiently generate Digitally Reconstructed Radiographs. These results are immediately applicable as means of speeding up 2D-3D and 3D-3D intensity based registration and will be incorporated into our future work on construction of atlases and deformable intensity-based registration.

1. Introduction and Background

One of the most critical research problems in the analysis of 3D medical images is the development of methods for storing, approximating and analyzing image data sets efficiently. Many groups are developing electronic atlases for use as a reference database for consulting and teaching, for deformable registration-assisted segmentation of medical images and for use in surgical planning. Researchers at INRIA have built atlases based on surface models and crest lines [1-3]. Cutting *et al* [4] have built similar atlases of the skull. These atlases are surface models with some landmarks and crest lines and don't contain any volumetric density information. Chen [5] and Guimond [6] have built an average brain atlas based on statistical data of the voxel intensity values from a large group of MRI images. Their atlases only have intensity information and don't describe the structural information of the anatomy. Pizer *et al* have proposed a medial model representation called M-rep to represent the shapes of 3D medical objects [7, 8]. A CSG scheme allows M-reps to describe complicate shapes. One advantage of this approach is that it is easy to deform medial atoms to accommodate shape variations. One drawback is that the current representational scheme is primarily useful for exterior shapes, and more work will be needed to extend this work to volumetric properties.[1]

One goal of our current research is to construct a deformable density atlas for bone anatomy and to apply the atlas to different applications. A related goal is to provide an efficient representation for 2D-3D and 3D-3D intensity based registration.

[1] The authors have had very useful discussions with Dr. Pizer about the issues involved, and are looking forward to exploring these issues with the UNC group.

S.L. Delp, A.M. DiGioia, and B. Jaramaz (Eds.): MICCAI 2000, LNCS 1935, pp. 531-540, 2000.

Our intent is to use intensity-based deformable registration methods both in the construction of the atlas and in exploiting it for patient-specific procedure planning and intraoperative guidance. It may be possible in many cases to dispense with patient CT in favor of radiographs. The atlas will provide a basis for representing "generic" information about surgical plans and procedures. The atlas will provide infrastructure for study of anatomical variation. The techniques should be extendable to a wide variety of bony anatomies. It may also provide an initial coordinate system for correlating surgical actions with results, as well as an aid in postoperative assessment.

This atlas should include: 1) model representations of "normal" 3D CT densities and segmented surface meshes; 2) 3D parameterization of surface shape & volumetric properties; and 3) statistical characterization of variability of parameters.

We adopt a tetrahedral mesh model to represent volumetric properties for several reasons. Tetrahedra provide great flexibility and other representations can be converted into tetrahedral meshes relatively easily. Tetrahedra are easy to deform and are easy to compute with. It is convenient to assign properties and functions to the vertices and tetrahedra. Computational steps such as interpolation, integration, and differentiation are fast and often can be done in closed form. Finite element analysis is often conveniently performed on tetrahedral meshes.

This paper reports current progress in building suitable tetrahedral density models from patient CT scans and illustrates the usefulness of this representation in performing an important calculation in 2D-3D registration. In section 2, we elaborate the method to construct the tetrahedral mesh from bone contours. We outline the key steps of the method, including contour extraction, tetrahedron tiling, branching and continuity preserving. In section 3, we present the method to compute and assign the density function to the tetrahedron and evaluation the accuracy of the density function. Then in section 4, we show the results using our density model to generate Digitally Reconstructed Radiographs (DRRs). Finally section 5 discusses current status and future work.

2. Construction of Tetrahedral Mesh Models from Contours

There are several techniques to construct the tetrahedral meshes. The easiest way first divides the 3D space into cubicle voxels that can be easily divided into four tetrahedra [9]. The drawback of this method is that the mesh generated is far too dense and doesn't capture any shape property of the model. A variation of this method first merges similar voxels into larger cubes or uses oct-tree techniques to subdivide the space, then performs the tetrahedronization. 3D Delaunay triangulation is another method. But this method is time consuming and requires some post-processing since the basic algorithm produces a mesh of the convex hull of the anatomical object. Boissonnat *et al* [10] proposed a method to construct tetrahedral mesh from contours on cross section. They computed 2D Delaunay triangulation on each section, then tiled tetrahedral mesh between sections and removed external. They also solved complex branching problem by contour splitting. But they didn't consider the intercrossing and continuity between tetrahedra, so their tetrahedral mesh may not be valid for volumetric analysis.

Our tetrahedral mesh construction algorithm is derived from the surface mesh reconstruction algorithm of Meyers, *et al* [11]. The tetrahedral mesh is constructed slice by slice and layer by layer based on bone contours extracted in a separate

algorithm. The algorithm is straightforward and fast. The running time is *O(n)*, where *n* is the total number of vertices in the model. We only sketch the algorithm here. More details can be found in our research report [12].

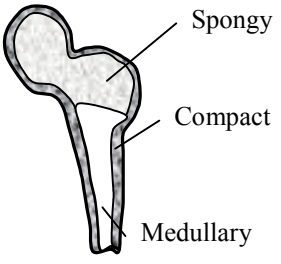
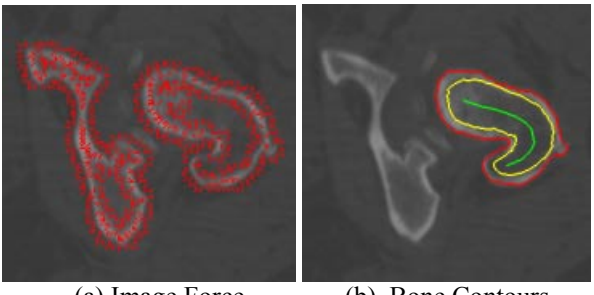


Figure 1. Bone Structure



(a) Image Force (b) Bone Contours

Figure 2. Contour Extraction

2.1 Contour Extraction

Bones contain two basic types of osseous tissue: compact and spongy bone [13]. Compact bone is dense and looks smooth and homogeneous. Spongy bone is composed of little beams and has a good deal of open space. The shaft of long bones is constructed of a relatively thick collar of compact bone that surrounds a medullary cavity. Short, irregular and flat bones share a simple design: they consist of thin plates of compact bone on outside and spongy bone within. Figure 1 illustrates this structure.

We extract both the outer contour and the inner contour of the bone. If the shell between contours is too thin, only outer contour is extracted[2]. In our algorithm, we apply a deformable "snake" contour model to extract the bone contours [14]. The forces that drive the deformable contour model can be expressed as: $F = F_{internal} + F_{image} + F_{external}$, where $F_{internal}$ is the spline force of the contour. F_{image} is the image force generated from the iso-value. And $F_{external}$ is an external force to allow the contour shrink or expand relative to the medial axis of the contour. Provided the initial value of the contour, the algorithm can automatically converge to the contour of the bone. The algorithm can also be applied repeatedly using the outer contour as the initial value to find the inner contour. It should be mentioned that the users only need to provide the initial contour for the first slice. For any following slice, the contour generated on previous slice can be used as the initial value. So our method is highly automatic. Figure 2a shows the image force used in our algorithm to extract the inner and outer contour of the bone.

The tetrahedral model is a solid model. So after obtaining the contour of the bone, we also compute the medial axis of the inner contour (or of the outer contour if no inner contour has been extracted) and treat it as the innermost layer. Currently, we use Lee's method to extract the medial axis [15]. Hence we can tile the medial axis and the bone contour to build a solid model.

The contours are fitted into B-spline curves and re-sampled at desired resolution for the follow up tiling. During sampling, the segments with larger curvature are

[2] In the current implementation, this decision is made manually on a slice by slice basis, but it will be easy to automate.

sampled at higher resolution, while those with smaller curvature at lower resolution. Figure 2b shows the contours and the medial axis extracted by our algorithm.

2.2 Data Structure of Tetrahedral Mesh

We chose a simple data structure to represent the tetrahedral mesh, consisting of a list of vertex coordinates and a list of tetrahedra. Each tetrahedron contains links that reference its four vertices and the four face neighbors. The face neighbors of a tetrahedron are those tetrahedra share a common face with this tetrahedron. This data structure is easy to maintain and update. Other information such as the density function is also stored in the tetrahedron.

2.3 Tiling

Tiling is the essential step in building the tetrahedral mesh. The idea is to divide the space between adjacent slices into tetrahedra by the aid of the bone contours, then connect the tetrahedra into a mesh. The tiling operations are based on local information, so the algorithms are fairly efficient even on large data set. The tiling can be expressed as the traversal of four ordered contours on two adjacent slices (two on each slice). At each step, one or two contours advance. Once the contours advance, new tetrahedra are generated. The tiling continues until all contours are traversed.

2.3.1 Tiling Patterns: There are 32 distinct tiling patterns for each single step [12]. There are two categories of the tiling patterns: advance-one-contour and advance-two-contours. Figure 3 shows two tiling pattern examples. In order to understand the tiling pattern, we give some definitions. In Figure 3, the tiling happens between adjacent slice N and slice N+1. The current vertices of the contours are called the front vertices. The next vertices of the front vertices are called the candidate vertices for next advance. In Figure 3a, *a1, b1, c1, d1* are the front vertices and *a2 b2 c2 d2* are the candidate vertices of contour *a b c d* respectively. The front faces are those triangles composed by the front vertices. The front faces are the connections between the tetrahedra generated in last tiling and those to be generated in current tiling. In Figure 3a, triangle *a1b1c1* and triangle *b1c1d1* are the front faces. The pivot vertices are those vertices chose for current tiling. We can have one or two pivot vertices in each tiling. The selection of pivot vertices determines the tiling pattern. In Figure 3a, *b2* and *d2* are the pivot vertices. And in Figure 3b, a2 is the pivot vertex. In general, the pivot vertices and the front faces together decide the tiling pattern. In Figure 3a, the tiling pattern produces three new tetrahedra. They are *b2a1b1c1*, *b2b1c1d1* and *b2c1d1d2* (we denote the tetrahedron by its four vertices).

There are totally 32 tiling patterns according to the selection of pivot vertices and the fact of the front faces. Among these, $\binom{4}{2}\times 2\times 2 = 24$ cases are advance-two-contours patterns and $\binom{4}{1}\times 2 = 8$ cases are advance-one-contour patterns.

2.3.2 Metric Functions: As shown in last section, we have 32 tiling patterns. In order to choose the best pattern for tiling, a metric function must be evaluated on each candidate pattern. And we find out that most metric functions used in surface tiling

are also good for tetrahedron tiling [11]. The metric function used in our algorithm is a combination of minimizing span length, matching direction and matching normalized arc length. Details of the metric functions are in our research report [12].

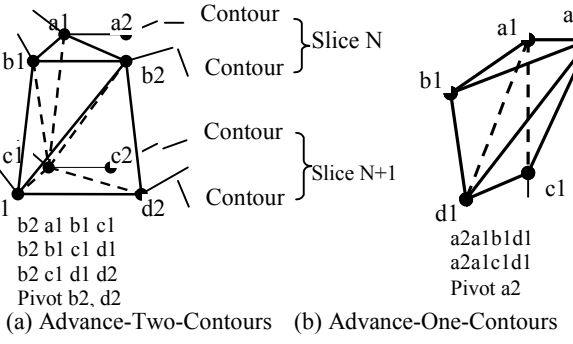


Figure 3. Tiling between slices

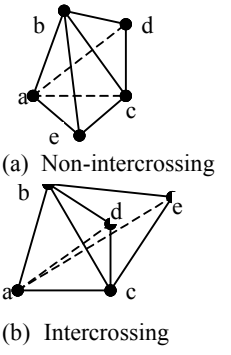


Figure 4. Intercrossing between tetrahedra

2.3.3 Constraints: The tiling problem seems very unconstrained due to the 32 tiling patterns we mentioned in section 2.3.1. But in order to form a reasonable tetrahedral mesh, some constraints must be imposed.

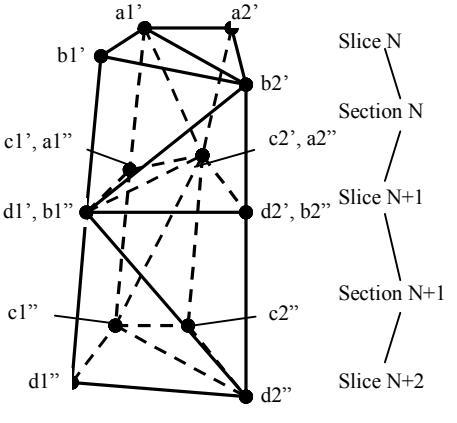


Figure 5. Continuity constraint between slices
Traversal sequence of section N is *b2c2a2d2*, and that of section N+1 is *d2a2b2c2*. (For clarity, all internal crossing edge are omitted)

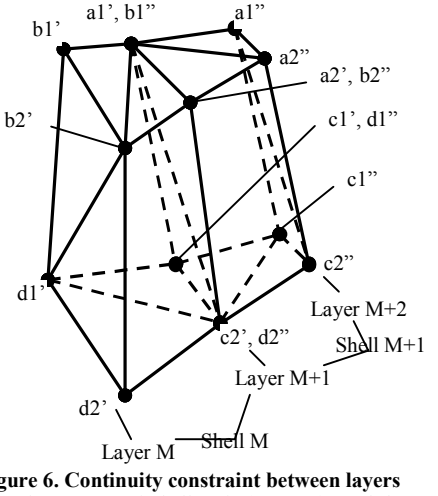


Figure 6. Continuity constraint between layers
Traval sequence of Shell M is *b2c2a2d2*, and that of Shell M+1 is *d2c2a2b2*. (For clarity, all internal crossing edge are omitted)

2.3.3.1 **No Intercrossing between Tetrahedra:** This constraint is obvious for a valid subdivision of the volume. The case that causes intercrossing is shown on figure 4. Figure 4.a shows a non-intercrossing case. But in Figure 4.b, line *ae* pierces triangle *bcd* and causes the intercrossing of two tetrahedra *abcd* and *abce*. So when we select a tiling pattern, those newly generated tetrahedra shouldn't intercross with each other and shouldn't intercross with the old tetrahedra.

2.3.3.2 **Continuity Constraint between Slices:** On a valid tetrahedral mesh, all triangular faces except those on the boundary should have two neighbor tetrahedra. Because we build the mesh slice by slice, the triangle faces between sections should be shared by two sections and a continuity constraint should be imposed. To solve this problem, we record the traversal sequence as an ordered array. Each time the tiling procedure advances to new pivot vertices, the contour number of the pivot vertices will be added to traversal sequence. So the traversal sequence records the tiling procedure. For example in Figure 5, the traversal sequence of *Section N* in that segment is *bcad*, which means the pivot vertices are in the order of *b2c2a2d2*. Furthermore we define the sub traversal sequence *ab* of the traversal sequence as a subsequence which contains all and only the entries of *a b* in the entire traversal sequence. For example, sub traversal sequence *ab* of the above traversal sequence is *ba*. Similarly we can define the sub traversal sequence *cd, ac*, etc. And we can see that the subsequence *cd* also records the tiling pattern of the triangle faces on *slice N+1*. Because *Section N+1* and *Section N* share *slice N+1*, the tiling pattern on *slice N+1* should be preserved while tiling *Section N+1*. This means that the sub traversal sequence *cd* of *Section N* should be identical to the sub traversal sequence *ab* of *Section N+1*. Here we should notice that the contour *c'* in *Section N* becomes the contour *a"* in *Section N+1*, while contour *d'* becomes contour *b"*.

2.3.3.3 Continuity Constraint between Layers: Similar to the continuity constraint between slices, the continuity constraint between layer should also be imposed during the tiling procedure. We build the tetrahedral mesh layer by layer. We call the space between *layer M* and *layer M+1 Shell M*. *Shell M* and *Shell M+1* share *layer M+1*. So the sub traversal sequence *ac* of *Shell M* should be identical to the sub traversal sequence *bd* of *Shell M+1*. We also should notice that the contour *a'* in *Shell M* becomes the contour *b"* in *Shell M+1*, while contour *c'* becomes contour *d"*. Figure 6 illustrates the continuity constraint between layers.

2.4 Contour Correspondence between Slices

The corresponding problem arises whenever there are multiple contours (the outer contours) on one slice. Currently in our problem, the bone model is usually simple and doesn't have a lot of branches. So the correspondence is not very complicate. We solved the correspondence problem by simply examining the overlap and distance between contours on adjacent slices.

2.5 Branching Problems: Problems associated with branching structures have been studied intensively in surface mesh construction [11]. Some of the techniques are suitable for tetrahedral mesh construction. There are two kinds of branching problems: branching between layers and branching between slices.

2.5.1 Branching between Layers: The layer branching occurs when the numbers of layers of corresponding contours between adjacent slices are different. The branching between layers can be easily converted to the tiling of three contours (one on one slice, two on the other slice), which is a special case of the tiling of four contours. Branching between layers usually happens at the boundary between spongy bone (2 layers) and compact bone (3 layers).

2.5.2 Branching between Slices: The branching between slices occurs when the numbers of contours between adjacent slices are different. We adopt the methods used in the surface mesh construction [11]. We construct a composite contour that connects the adjacent contours at the closest points. (Figure7a). This composite contour is then tiled with the single contour from the adjacent slice. In Figure 7, we show the case of *1:2* branching. This method can handle more general cases, i.e. the *m:n* branching cases. In the general case, the method can produce composite contours for both slices and then conduct the standard tiling procedure.

2.6 Results

Figure 8 shows wire frame renderings of two tetrahedral mesh models produced by our method. Figure 8.a is a femur model. Figure 8.b is a half pelvis model. Table 1 lists some facts about these two models.

Model	Num of Vertices	Num of Tetrahedra	Num of Slices	Num of Voxels inside	Volume (mm^3)	Area (mm^2)
Femur	1400	7815	46	859503	144720	17593
Pelvis	3887	18066	89	873494	497664	83453

Table 1. Facts about two tetrahedral model

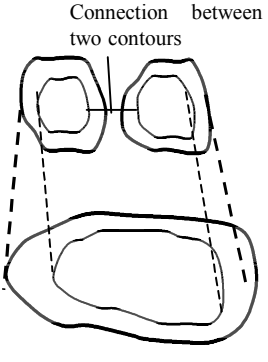
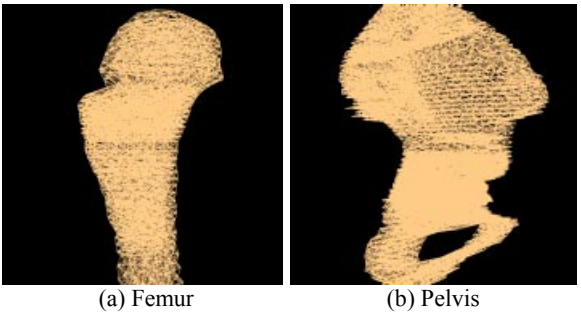


Figure 7. Branching between slices



(a) Femur (b) Pelvis

Figure 8. Tetrahedral Mesh Model

3. Density Function

We assign an analytical density function to every tetrahedron instead of storing the density value of every pixel in the model. The advantage of such a representation is that it is in explicit form and is a continuous function in 3D space. So it is easy to compute the integral, to differentiate, to interpolate, and to deform.

We build our density function in the barycentric coordinate base. The definition of barycentric coordinate in a tetrahedron is as following. For any point K inside a tetrahedron *ABCD*, there exists four masses w_A, w_B, w_C, and w_D such that, if placed at the corresponding vertices of the tetrahedron, their center of gravity (barycenter) will coincide with the point K. (w_A, w_B, w_C, w_D) is called the barycentric coordinate of K. Furthermore barycentric coordinates are in a form of homogeneous coordinates where $w_A+w_B+w_C+w_D=1$, and the barycentric coordinate is defined uniquely for every point.

Currently we define the density function as a quadratic polynomial. For each tetrahedron, we first get a sample of the pixels inside the tetrahedron, and obtain the pixel density via the CT data set, then fit a polynomial function of the density in the barycentric coordinates of those sampled pixels. After getting the density function, we can compute the density of any point inside the tetrahedron by their barycentric coordinate. In our initial experience quadratic polynomials have worked well, although we are considering the use of higher order polynomials in barycentric Bernstein form with adaptively simplified tetrahedral meshes.

In order to test the accuracy of the density function, we randomly chose sample points $\{p_i\}$ inside the tetrahedron. The number of sample points of each tetrahedron for testing is the same as the number used to generate the density function. Then we compute the densities $\{c_i\}$ of $\{p_i\}$ by interpolation using CT data set and compute the densities $\{d_i\}$ of $\{p_i\}$ using the density function of the tetrahedron. Table 2 lists the comparison of $\{d_i\}$ and $\{c_i\}$. We present the results in three bone categories: compact bone, spongy bone and medullary cavity. We got the results from the femur density model we built in last section. From the results we can learn that the density distributions in compact bone and medullary cavity area are more homogeneous.

	Avg Density $Avg(c_i)$	Avg Density Diff $Avg(\lvert c_i-d_i\rvert)$	Std Dev $(\lvert c_i-d_i\rvert)$	Avg Density Diff $Avg(\lvert c_i-d_i\rvert/c_i)$	Max Density Diff $\text{Max}(\lvert c_i-d_i\rvert)$
Compact Bone	105.98	1.869	1.440	1.9%	4.803
Spongy Bone	77.9	2.309	2.046	2.4%	7.437
Medullary Cavity	70.956	1.783	1.354	1.3%	4.325

Table 2. Accuracy of Density Function

The other advantage of our density model is its efficiency in storage. We store a density function for each tetrahedron instead of storing the density value for every voxel inside the tetrahedron. For quadratic polynomial in 3D space, we need 10 parameters to describe it. Table 3 gives the comparison of storage usage in tetrahedral density model and regular CT image (here we only count those voxels inside the tetrahedral mesh). From the table we can see that the larger the tetrahedron, the more efficient in storage the density model can be. A hierarchical structure will further improve the storage efficiency.

	Num of Tetra	Avg Volume (mm^3)	Avg Num of voxels inside	Storage per tetra (bytes)	Avg Storage in CT image (bytes)
Compact Bone	5110	13.958	75.6	Min 72	151
Spongy Bone	2504	26.745	181.1	Max 84	362
Medullary Cavity	164	34.228	200.4	Avg 75	401

Table 3. Storage of Tetrahedral Density Model

4. Computing DRRs from Tetrahedral Mesh Density Model

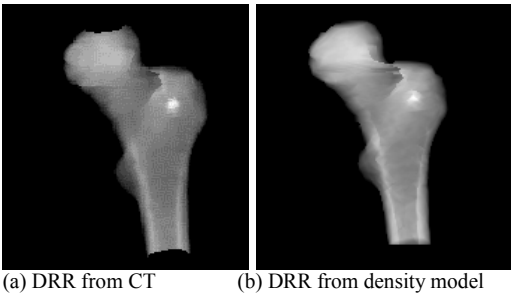
We can employ our tetrahedral mesh density model to support fast computation of DRRs. When a ray passes through a 3D data set, it may go though hundreds of voxels but only a few tetrahedra. Because the density function is in an explicit form, the integral along a line can be computed in close form. Furthermore the neighborhood

information is stored in the mesh, so it is fast to get the next tetrahedron hit by the ray from current hitting tetrahedron. And it is also fast to find the entry point of the casting ray from the neighborhood information. In contrast generating DRR from CT data require computing partial voxel crossing, which is a time consuming operation. So we have an efficient way to generate the DRR from our tetrahedral density model, which is an important technique in 2D-3D intensity-based registration. The computation can be even faster if we have a hierarchical model to represent different level of details.

	Running time	Avg. elems Passed through	Avg Density	Avg Density Diff $Avg(\lvert c_i-d_i\rvert/c_i)$	Std Dev of Density Diff $(\lvert c_i-d_i\rvert/c_i)$
CT Data set	45 s	65.6 voxels	142.5	4.9%	3.7%
Density Model	14 s	18.3 tetras	142.3		

Table 4. Comparison of DRR using CT data set and Tetrahedral Density Model

Figure 9a is a DRR generated from a CT data using the ray casting algorithm. Figure 9b is a DRR generated from our tetrahedral density model. The image size is 512*512. Table 4 shows the comparison of the two DRRs in Figure 9. We compare the pixel values $\{c_i\}$ of the DRR generated from CT data and the pixel values $\{d_i\}$ of the DRR generated from density model (We only compare those non-zero values). We find out that the running time for an un-optimized implementation using density model is only about 1/3 of the running time using CT data set. Figure 10 illustrates the tetrahedra passed by a ray.



(a) DRR from CT (b) DRR from density model

Figure 9. DRR of femur

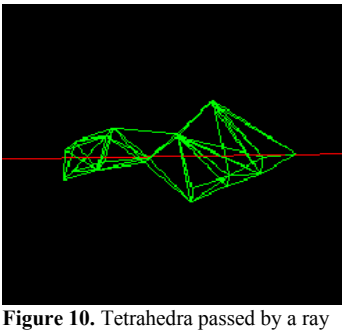


Figure 10. Tetrahedra passed by a ray

5. Discussions and Future Work

In this paper we have presented the first phase of our density atlas research. We proposed a very efficient and automatic method to construct the tetrahedral mesh from bone contours. We assigned a density function to the tetrahedron and showed that it is reasonably accurate. We also presented the fast generation of DRRs from our density mesh, which is an essential technique for 2D-3D intensity based registration and volume visualization.

We will continue our work on building the density atlas. We are investigating the technique to simplify the tetrahedral mesh based on face collapsing and build a hierarchical data structure to represent multiple levels of details of the density model

[9]. We will also investigate the deformation rule of the tetrahedral mesh and make the density atlas deformable. We will build an average density model from a large group of data by incorporating the statistical information in the model. And we will apply finite element technique to study the biomedical and bio-mechanical properties of the anatomy. Ultimately we will apply the density atlas to various application such us 2D-3D registration, surgical planning etc.

Acknowledgements

This work was partially funded by NSF Engineering Research Center grant EEC9731478. We thank Integrated Surgical System (Davis, CA) for providing the femur data and the Shadyside Hospital for the pelvis data. We specially thank Stephen Pizer, Sandy Wells, Branislav Jaramaz, Robert VanVorhis, Jerry Prince, and Chengyang Xu for their useful suggestions. Some of their suggestions have been included in this paper; more of them are still under investigation.

References

[1] G. Subsol, J.-P. Thirion, and N. Ayache, "First Steps Towards Automatic Building of Anatomical Atlas," INRIA, France 2216, March, 1994.

[2] G. Subsol, J.-P. Thirion, and N. Ayache, "A General Scheme for Automatically Building 3D Morphometric Anatomical Atlas: Application to a Skull Atlas," INRIA 2586, 1995.

[3] J. Declerck, G. Subsol, J.-P. Thirion, and N. Ayache, "Automatic Retrieval of Anatomical Structures in 3D medical Images," INRIA, France 2485, Feb. 1995.

[4] C. B. Cutting, F. L. Bookstein, and R. H. Taylor, "Applications of Simulation, Morphometrics and Robotics in Craniofacial Surgery," in *Computer-Integrated Surgery*, R. H. Taylor, *et al*, Eds. Cambridge, Mass.: MIT Press, 1996, pp. 641-662.

[5] M. Chen, T. Kanade, *et al*, "3-D Deformable Registration of Medical Images Using a Statistical Atlas," Carnegie Mellon University, Pittsburgh, PA CMU-RI-TR-98-35, December, 1998.

[6] A. Guimond, J. Meunier, and J.-P. Thirion, "Average Brain Models: A Convergence Study," INRIA, France 3731, July, 1999.

[7] S. M. Pizer, A. L. Thall, and D. T. Chen, "M-Reps: A New Object Representation for Graphics," Univ. of North Carolina, Chapel Hill 1999.

[8] S. M. Pizer, D. S. Fritsch, *et al*, "Segmentation, Registration, and Measurement of Shape Variation via Image Object Shape," *IEEE Trans. Medical Imaging*, 1999.

[9] I. J. Trotts, B. Hamann, and K. I. Joy, "Simplification of Tetrahedral Meshes with Error Bounds," *IEEE Transactions On Visualization and Computer Graphics*, vol. 5, pp 224-237, 1999.

[10] J.-D. Boissonnat and B. Geiger, "Three dimensional reconstruction of complex shapes based on the Delaunay triangulation," INRIA, France 1697, May, 1992.

[11] D. Meyers, S. Skinner, and K. Sloan, "Surfaces from Contours," *ACM Transactions on Graphics*, vol. 11, pp. 228-258, 1992.

[12] J. Yao, "Building Tetrahedral Mesh Model from Bone Contours," ERC, Johns Hopkins Univ. ERC-2000-01, 2000.

[13] E. N. Marieb, *Human Anatomy and Physiology*, second edition, 1992.

[14] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active Contour Models," *International Journal of Computer Vision*, pp. 321-331, 1988.

[15] D. T. Lee, "Medial Axis Transformation of a Planar Shape," *IEEE Transactions On Pattern Analysis and Machine Intelligence*, vol. 4, pp. 363-369, 1982.