Construction of a human torso model from magnetic resonance images for problems in computational electrocardiography.

*Robert S. MacLeod** (macleod@cvrti.utah.edu),
Christopher R. Johnson† (crj@cs.utah.edu),
*Phil R. Ershler** (ershler@cvrti.utah.edu)

UUCS-94-017

*Nora Eccles Harrison
Cardiovascular Research and Training Institute
and
†Department of Computer Science
at the
University of Utah
Salt Lake City, UT 84112 USA

September 26, 1994

Contents

Abstract

Applying mathematical models to real situations often requires the use of discrete geometrical models of the solution domain. In some cases destructive measurement of the objects under examination is acceptable, but in biomedical applications the measurements come from imaging techniques such as X-ray, computer tomography (CT), or magnetic resonance imagining (MRI). A necessary early step in the modeling process is then to extract from these images the measurements (locations and distances) that form the basis of the geometrical model.

In this paper we describe the construction of a geometrical model of the human thorax based on the high resolution MRI scan of a single subject. We outline the scanning procedure, the image collection and conversion to computerized image files, the segmentation of the images into boundary nodes, and the connection of these nodes into surface, and then volume, meshes. Included are brief descriptions of the tools developed at the CVRTI for this project, as well as our experiences in creating and using them. The result of this work was a pair of models at two different levels of spatial resolution, which set new standards in the area of bioelectric field modeling and the application of these models has been described previously [1, 2].

1 Introduction

The development of numerical modeling tools which work on complex, irregular, two and three-dimensional domains has opened many fields previously dominated by direct experimental research to the application of simulation and modeling. While leadership in this advance has come largely from the areas of mechanical and electrical engineering, biomedical researchers have also been advocates of these techniques. For the medical researcher, the attraction goes beyond the cost issues of most engineering disciplines to the important ethical constraints of experiments involving both humans and animals. As a result, the biomedical research community, while historically not as mathematically sophisticated as other research groups, has been quick to apply emerging methodologies to their studies.

This paper describes just such an application of modeling techniques to biomedical, in this case electrocardiographic, research. The focus of the work is to develop computer models which, most generally phrased, will predict electrical activity in the heart from electric potential (voltage) measurements on the surface of the body, the essence of the medical diagnostic technique known as electrocardiography. In contrast to the largely qualitative or descriptive approach employed in the interpretation of the electrocardiogram (ECG), we seek a quantitative description of the electric field within, and on, the torso. Due to the complexity of the equations describing such relationships, any modeling approach requires a discrete description of the geometry of the heart and torso. The specific aim of this paper is to describe the creation of just such a model of the human thorax geometry for studies of electrocardiographic field problems.

2 MRI Recording

While destructive modeling of objects is an acceptable technique in some areas of modeling, gathering anatomical information from living beings is often more effectively done using (relatively) noninvasive techniques. For the reconstruction of three dimensional objects, a scanning modality is required. In medical applications, such scanning techniques make use of, for example, X-ray, for computer tomography (CT) scans; radionuclide emission, for gamma scans or autoradiography; ultrasound, for sonography; and magnetic spin resonance, for magnetic resonance imaging (MRI). Each of these modalities exhibits limits in spatial and temporal resolution, as well as a variable degree of side effects [3], which make the appropriate choice a function of many, often contradictory, factors.

For this project, we selected MRI as the imagine methodology best suited to our primary goals which were to achieve maximal spatial resolution with minimal side effects. We were fortunate to have the support of the University of Utah Health Sciences Center and Dr. Dennis Parker to overcome a significant shortcoming of MRI — the cost. Below is a description of all phases of the recording and digitalization of the Utah human torso models from a sequence of MRI images

2.1 Recording of MRI

Recording was performed at the LDS Hospital Magnetic Resonance Imaging Center by Dr. Dennis Parker. The subject, a healthy, 19-year old volunteer, was recorded on August 31, 1990 with Robert MacLeod and Christopher Johnson assisting. Spacing between the images was 5 mm and all images from slices near the heart were gated with the R wave of the ECG.

2.2 Scaling of the Images:

The scaling of the images themselves was determined by the field of view (FOV) set by the operator of the MRI equipment. Every image contained 256 pixels by 256 pixels of data and these covered the selected field of view. The conversion to length units occured through simple division, for example, with a FOV = 48, the scaling factor is FOV/256 [cm/pixel].

2.3 File naming conventions

Images recorded from this subject were saved in separate files (one per image) when they were transferred to the Vax and unpacked from the Data General format. The file names were set up originally as `RCLMR.F001B` to `RCLMR.F143` from tape 1, and `RCLMR.F001B` to `RCLMR.F096B` from tape 2. During conversion (using *CONVERT*) from the large image format to the 256 by 256 format, the files names were also converted and encoded with some information about the image data. The filename format was *xx***t***n***s***mmm_ppp***.mri**, where *xx* were the initials of the subject, *n* was the tape number, *mmm* was the distance of the slice about the arbitrary zero-level set at the time of the recording, and *ppp* was the running number of the image in the set. The complete list thus ran from `rlt1s020_001.mri` to `rlt2s615_120.mri`, that is, 120 images running from 20 mm to 615 mm above the zero level.

3 Digitization of Slices

From the Vax, the image files were transferred to a Mac II via FTP with no conversion. Each file contained a single image, stored as a single 65536-byte record. Each byte contained the gray value for a single pixel in a 256 by 256 pixel image. The first byte in this file type was at the top left-hand corner of the image; the rest of the image followed sequentially from this point, left-to-right, top-to-bottom.

Digitization of the images was performed manually using the program *New Digitize* on the Mac II. This program has controls (pull-down menus) to read the data files and display them as 256 by 256 or 512 by 512 pixel images, in white-on-black or black-on-white shades. The user could also add constant values to all gray values to brighten the image and enhance dark sections.

Once the image was displayed on the screen, the user selected which surface was to be digitized and proceeded around the image, marking each point with the mouse. The outer, or body surface was the default starting surface and each selected point was marked with a red "+" sign. Additional surfaces which were digitized include the fat layer (boundary between fat and muscle), the inner boundary of the muscle, the epicardial, left ventricular and right ventricular endocardial, and both lung surfaces. Phantom electrodes placed on the subject could be digitized using sets of two points aligned with the corners of the electrode image. *The order in which points are digitized was up to the user but was normally in a clockwise direction around the surface!* Note also that the standard view of an MRI image is from the feet of the patient towards the head. At the time of processing, there was no facility to edit digitized points except to clear a whole set of points from one surface and re-digitize them all. Once the points were selected, the user stored the data in a text file for later processing.

3.1 Raw points file format

The raw points files contained the digitized point locations from the *New Digitize* program. The format of the file was quite simple: each point was stored as a coordinate pair, delimited by a single comma. The first value of each pair represented the row number of the selected pixel, the second the column number, both relative to the top left corner of the image. Points from a single surface were stored together, prefixed by a single line containing the number and type of points which followed. For example:

```
5 body surface points
163,298
165,307
169,313
173,320
177,326
6 fat points
169,301
172,308
176,314
180,322
185,329
191,332
0 epicardial points
0 rv endo points
0 lv endo points
0 lung points
0 electrode points
```

NOTE: This format was altered somewhat in November., 1990. As described above the "fat" points actually included both the points which separate the fat from the muscle and those which define the inner boundary of the muscle layer. This was changed so that the "fat" layer included *only those points which separate the fat and muscle regions*. The points which define the inner boundary of the muscle layer(s) were then to be found in a second "muscle" layer. This change was reflected both in the digitization software (*New Digitize*, as well as data previously stored, which was converted using the program *mrisort*. An example of data in the new format is given below:

```
7 body surface points
173,305
174,314
176,321
```

```
180,328
184,334
189,339
5 fat points
181,309
181,320
186,327
194,331
199,336
0 epicardial points
0 rv endo points
0 lv endo points
0 left lung points
0 right lung points
8 muscle points
192,308
198,316
206,323
217,330
226,337
230,342
240,341
247,337
0 electrode points
```

3.2 Conversion of Units

The units returned by *NewDigitize* for the digitized points were based on the pixel row and column marked by the user with the mouse, but were *shifted* to facilitate display on the Macintosh screen. To convert the pixel location back to the 256 X 256 pixel format of the original MRI data image, the following conversion had to be applied:

```
mri_row_num = (mac_row_num - 35) / 2
mri_col_num = (mac_col_num - 50) / 2
```

In order to combine digitized data and the MRI image in the same display, a further conversion was necessary to replace the row/column addressing scheme with a Cartesian co-ordinate system. See discussion of the program *mri2ps* in section 3.6.1 and *mrimetric* in section 3.8 below.

3.3 Conversion of MRI images and transfer to the SGI

Since we used a Silicon Graphics (SGI) workstation for the bulk of the further image processing, we had to move both the image files and the digitized values from the Macintosh or Vax. The digitized values in the raw data file were ASCII format and hence presented no problems, however, the image files were binary and had to be first converted from a single 65536-byte records to something more amenable to transfer over the TCP/IP-based link. This was made necessary by a limitation of the program *TRANSL8*, which we used to convert binary files between VMS (RMS) format and UNIX. We chose to break the (for *TRANSL8* oversized) 64 kByte single-record files into a sequence of 256-byte records and wrote the program *CONVERT.EXE* on the Vax 750 to perform the conversion.

Another feature of *CONVERT* was the option to have amplitude histogram calculated from the gray values of each image. The histogram had 100 bins which were divided equally over the full range of the gray values (maximum 0-255). The results were stored in ASCII files as pairs of values: the first number on each line was the gray value at the centre of the bin and the second number was the number of pixels found in the bin. The output name of the histogram data defaulted to the same as the image file with the extension ".histo". The point of constructing histograms of the gray values was to guide the modest image processing steps which could be performed before displaying the images. While the gray values were normalized to span the full range of 0-255 before being stored as Vax files, the number of pixels with gray values in the upper (white) end of the spectrum was often very small. By either adding a constant to the gray values (to lighten the image) or scaling the gray values by a factor (increasing the contrast), saturation of these relatively few light pixels could be tolerated in return for improved image quality.

3.4 Transfer of Image Data

The program *TRANSL8*, which performed the transfer of the converted image data files, ran on both the Vax and the SGI. In order to direct the conversion of the VMS binary format to that required by UNIX, a format file had to exist on each system. For the transfer of 256-byte records of image data we used the same format file (`MRI.FSF`) on both systems:

```
# Transfer the MRI data files, packed a single buffer of bytes,
# across from VMS to UNIX.
access sequential
form unformatted
record *(256A1)
```

The commands necessary to perform the transfer with *TRANSL8* were similar to normal FTP commands: starting from the Vax, the user started *TRANSL8*, then issues the `OPEN` command to attach to the SGI:

```
V750>transl8

TRANSL8, TCP/IP Version 2.00
Unpublished - All rights reserved under the copyright laws
      of the United States
Copyright (c) 1989,1990 Accelr8 Technology Corporation

t8> open sgi
Trying to connect to 'sgi' ... OK
Waiting for Transl8 server response ... OK
>> sgi Login (macleod) : macleod
>> Password required :
macleod Login Successful.
t8>
```

From the `t8>` prompt, the user issued `put` and `get` commands as in ftp with the difference that the format file had to be specified if it did not have the same filename (with the `.fsf` extension) as the data file. The format of the put command was:

```
put sourcefile [targetfile] -s sourceformatfile -t targetformatfile
```

eg.

```
put rlt1s020_001.mri -s mri.fsf -t mri.fsf
```

which would transfer the file `rtl1s020_001.mri`, keeping the target filename the same, under control of the format file `mir.fsf`, which was available in the current directory of both the source and target computers.

3.4.1 Automatic login

It was possible to set up a file in the home directory of the VMS account which facilitated automatic login on the target system. The file had to be called `FTP.INIT`, and contained entries in the following format:

```
set systemname /username:yourusername /password:yourpasword
```

simply starting *TRANSL8* with the name of the system in the command line initiated the full login on the target machine, *eg.*

```
vax4000>transl8 sgi

TRANSL8, TCP/IP Version 2.00
Unpublished - All rights reserved under the copyright laws of the United
States
Copyright (c) 1989,1990 Accelr8 Technology Corporation

Trying to connect to 'sgi' ... OK
Waiting for Transl8 server response ... OK
macleod Login Successful.
t8>
```

3.4.2 Transferring large numbers of file

To copy more then a few files in this manner proved very tedious and time-consuming. To expedite the process for the MRI data, we used DCL command files to run *TRANSL8*. The file contained all the necessary login and file transfer commands, for example:

```
transl8 sgi
cd /usr/people/macleod/mri/rcldata
put   rlt1s020_001.mri   -s mri.fsf -t mri.fsf
put   rlt1s025_002.mri   -s mri.fsf -t mri.fsf
put   rlt1s030_003.mri   -s mri.fsf -t mri.fsf
put   rlt1s035_004.mri   -s mri.fsf -t mri.fsf
put   rlt1s040_005.mri   -s mri.fsf -t mri.fsf
put   rlt1s045_006.mri   -s mri.fsf -t mri.fsf
.
.
.
quit
```

The `cd` command ensured that the data files arrived in the desired subdirectory and the `.fsf` files were both in the same directories as the data files.

3.5 Accessing MRI files under UNIX

Once the image files had been converted and transferred from the VAX/VMS realm to Unix, accessing them was quite straightforward. All extra control data was stripped away during the *TRANSL8* process so that what remained was just a stream of bytes that could be read using, for example, unformatted Fortran read statements. Note, however, that the structure of the data was still in 256, 256-byte records, each record representing a row of the image, running from left to right, and rows running from top to bottom.

The Fortran routine `ReadMriImage` was what we normally used to read the data and the essential lines of code are:

```
C**********************************************************************
      IMPLICIT NONE
      INTEGER i,j,k, error, tlen, luin,
     +     Trulen
      BYTE datin(256,*)
      CHARACTER mrinfilename*(*), inchar*1
C**********************************************************************

C*** Open the file with the MRI data

      OPEN (UNIT = luin, FILE=mrinfilename, FORM='unformatted',
     +     ACCESS = 'sequential', STATUS = 'old', READONLY,
     +     IOSTAT = error)
      IF (error .NE\@. 0) THEN
         tlen = TruLen(mrinfilename)
         WRITE(*,'(/a,a/ a,i6 )')
     +        ' Error opening file ',mrinfilename(1:tlen),
     +        ' is ', error
         GO TO 999
      END IF

      DO i=1,256
         READ(luin) (datin(i,j), j=1,256)
      END DO
      CLOSE (luin)

 999  CONTINUE
      RETURN
      END
```

3.6 Postscript display – the *mri2ps* program

With the data transferred to the SGI computer, we needed programs to construct image copy in the Postscript (Adobe Systems) format. The program to convert mri images to postscript files was called *mri2ps* and ran interactively to create a single output file for each MRI image. An option of the *mri2ps* program allowed the user to select two image-enhancement manipulations of the images:

1. add a constant, user-specified value is added to all pixel gray-values, thus lightening the entire image.
2. use an amplitude histogram of the pixel data, which the program generates, to equalize the data and enhance contrast.

The latter maneuver is based on the finding that the largest numbers of pixels are found in the bins at the dark end of the gray scale, with an almost monotonic decrease in the number of pixels per bin as the scale becomes lighter. With the data divided into 100 bins spanning the 0-255 range, *mri2ps* permitted the user to specify what percentage of the maximum frequency to use as a cut off; the remaining pixel values were then "stretched" to fit the 256-value dynamic range. The necessary steps were as follows:

1. Perform histogram on all gray values in the image and determine the frequency count in the largest bin, f_{max}
2. Ask user what percentage (P_{cutoff} of the largest frequency to use as cutoff, and compute $f_{cutoff} = f_{max} * P_{cutoff}$
3. Assuming a monotonic decrease in bin frequency as gray value goes from the largest bin towards the white end of the scale, find the bin with a frequency just larger then f_{cutoff}, bin_{cutoff}
4. Find the gray value in bin_{cutoff}, $gray_{cutoff}$ and compute a scaling factor based on this value to apply to all other gray values: $scale = 255/gray_{cutoff}$
5. Apply *scale* to all gray values in the image, applying a cutoff of 255 to the results.

The result of this "stretching" was to increase the overall brightness of the image without noticeably enhancing the dark (black) pixels, *i.e.* the contrast was increased. Larger values of the cutoff percentage brought about a greater degree of contrast enhancement, but also drove more of the light pixels into (white) saturation. Typical values of acceptable contrast enhancement for this dataset lay in the range of 0.01 to 2 %.

3.6.1 Including digitized data: Co-ordinate Transformation

In addition to displaying the mri image data, *mri2ps* provided the option of including a plot of the surface boundary data, as entered by the user in *NewDigitize*, over the image. This required the matching of image and raw data coordinate system since the original MRI data was addressed in row/column format and the digitized points were in similar format but shifted and doubled to a 512 row by 512 column format. The necessary conversion is described by

$$xpoint = ydigpoint - 50.ypoint = 477. - xdigpoint, \tag{1}$$

where *xpoint*, *ypoint* describes the location of the point in Cartesian space and *xdigpoint*, *ydigpoint* the same point in the original digitized space.

See section 3.8 on the *mrimetric* program, which takes care of converting units of the digitized surface data from pixel addresses to shifted, corrected millimeters for more details.

3.7 From raw digitized to interpolated points

3.8 Scaling of points: *mrimetric*

The original digitized points were in pixel rows and column format as they came from *New Digitize* but had to be converted into some absolute units of length, based on a standard co-ordinate system before they could be used for creating the model. The standard units for all our models were chosen to be millimeters and the scaling from pixels to millimeters was provided by the field of view of the MRI imaging equipment, as described is section 2.1. Our standard co-ordinate system, hereafter referred to as "body" co-ordinates, is aligned such that the z-axis runs caudal to cranially through the approximate centre of the body. The x-axis is directed toward the left arm, the y-axis, by default in a right-hand co-ordinate system, toward the back. The $z = 0$ origin for this dataset was chosen at the level of the lowest slice so that all slices had positive z-values.

The program *mrimetric* performed the conversion of the digitized data from rows and columns to body co-ordinates. Below is a summary of the various functions required for this conversion.

3.8.1 Checking for doubled points

During the digitization process, accidental double-clicking at a point produced doubled entries in the resulting data file. A subroutine in the *mrimetric* program called `Check_For_Doubles` checked each point location against the remaining points and discarded any point within some tolerance.

3.9 Transforming the data

Transformation of the data was a two-part operation. The points first had to be centered about an origin (corresponding to the central axis of the body) and then converted from pixel units to absolute length units (millimeters). We first attempted to determine the location of the origin from calculations of centre-of-mass of all the points in the torso surface of each layer. However, because the sample points entered during the digitization with *New Digitize* are not regularly spaced, the resulting predictions were inevitably biased and could not be used. Instead, we first performed a preliminary interpolation of the data and used these points to compute unbiased (or less biased, at least) centers of mass for each layer. From these values, pairs of x- and y-shifts could be tabulated for all the layers; these were then stored in a scaling file and read by *mrimetric* at execution time. The user could also enter the shifts manually or have them computed from centre-of-mass values at run time.

The scaling files were simple ASCII files containing records such as:

```
rlt1s020_001.dat      48     0      rlz000_001.dat  277.71  234.30
rlt1s025_002.dat      48     5      rlz005_002.dat  278.12  234.30
rlt1s030_003.dat      48     10     rlz010_003.dat  277.62  234.30
```

The first name in each line was the raw digitized data filename, the next value was the field of view in centimeters, the third was the actual z-value in millimeters, the fourth was the output filename for the scaled, shifted data, the last two were the x-shift and y-shift, respectively, in millimeters. Note that these values already included the offsets which arise from the digitization process in *New Digitize* (see section 3.6.1)

The second part of transformation of the data involved converting the pixel locations (row, column co-ordinates) into millimeters. The conversion factor followed directly from the value for the field of view of the MRI-recording and the number of pixels in the image (512 by 512 in our case) as follows:

```
xval = column_val * pixel_to_mm - xshift
```

and

```
yval = row_val * pixel_to_mm - yshift.
```

where

```
pixel_to_mm = field_of_view [cm] / 51.2
```

3.10 Checking for closed surfaces

Subsequent stages of developing the geometric model were greatly simplified if surfaces in each layer could be treated as closed. While this condition represented something of an approximation on some layers, we considered the compromise to be well justified by the considerable savings in time in generating the model and the fact that absolute accuracy remained an unrealistic, and essentially unnecessary, goal for this project. Since this was an afterthought and not a criteria during the digitization process, it was necessary to detect and attempt to correct unclosed surfaces during the conversion of data from pixels to millimeters.

To check and, if necessary, close surfaces, we wrote the subroutines `Close_Surfaces` and `Check_for_Closed`. The aim of these routines was to flag discontinuities in the digitized data by monitoring the distance between successive points in a surface. When the distance between two points exceeded a pre-defined value (typically 40 mm) the program logged the location and point number as a discontinuity and passed this information to a routine for correction. Correction consisted of trying to use the next outer-most surface as a template, from which we could construct a parallel set of points to bridge the discontinuity. The procedure proved to be somewhat limited in cases where the template surface curved more than 10–20 degrees over the discontinuity. Even the worst case, however, yielded a set of points which we either accepted or manually adjusted to generate closed, relatively smooth and non-intersecting surfaces.

3.11 Interpolation: spline representation of the data points

We used the raw points digitized from each image to construct parametric, bi-cubic spline equations for the outline of each surface, and then regenerated node points at a constant (and adjustable) spatial separation as a form of spatial filtering or interpolation. This allowed us to control the internodal separation, and hence the resolution of the final model, as well as smooth the inevitably jagged hand-digitized point set. From each image, the result was a set of evenly spaced, closed loops, one for each surface, which we refer to as a 'layer'. The format of such later files is described in section 6.7.

The points generated by the spline fit formed the basis for all further work with the model. This was also the stage of the process at which a manual pass was made to detect errant points and ensure spatial continuity between layers of the geometry. A program for this called *EDTRISLICE* was written on the Vax using the Tektronix Plot10 library (for use with the Macintosh via Versaterm). The user could select the .slice files to read, and then view the entire surface, or select a single layer to edit. To help direct the editing, outlines of the layers above and below the actual edit layer were overlaid (in different colours), while only points in the middle layer could be moved. There was no facility for adding points, nor changing their order, just moving existing ones. A pass through every surface of every layer of the model was usually required before the set of model points was complete.

4 From points to surfaces

4.1 Background

In order to apply the boundary element method (BEM) to realistic geometric models, and also to visualize recorded or computed potentials, it is necessary to connect the points into planar polygons. Triangular elements make an excellent choice for irregular surfaces such as those in the human body. While it is possible to define criteria for optimal triangulation [4, 5], we know of no fully automatic scheme for the generation of triangles from points scattered in three-dimensional. Fortunately, in this case we can take advantage of the fact that all points from the layers described here are constrained to parallel planes, which are regularly spaced. As a result, the general triangulation problem reduces to a much simpler task of connecting pairs rows of points in successive layers, essentially a "lacing" process.

4.1.1 Lacing triangles

Algorithms for performing lacing operations of datapoints on parallel planes have been described[6, 7, 8] and shown to work even in situations in which the surface bifurcates, as in, for example, the structure of blood vessels[7, 8].

In order to apply these techniques to the construction of a model of the human torso, we combined the points from adjacent layers into three-dimensional "slices", which were the input for our lacing programs. Triangulation of each slice involved the following steps:

1. Determine an anatomically based starting point and locate the nearest node from each layer of the slice to that starting point. We typically chose the anterior midline of the body for a starting location.
2. Order the points for each surface in each layer in such a way that they form a continuous sequence, commencing at the starting point and proceeding in a common direction

(we chose counterclockwise, as viewed cranio-caudally) around the surface. This was typically performed during the cubic spline fitting of the sampled points.

3. Determine a starting point on one of the layers, and find the nearest point in the second layer; join these to define the first connective segment.

4. Proceeding to the next point in each layer, determine which of the two possible pairs of triangles that can be formed by joining them with the endpoints of the first segment yield the shortest diagonal (and hence the most optimal triangulation in terms of maximizing the size of the smallest angle in each triangle[5]).

5. With this pair of triangles defined, a new connective segment is formed, and the previous step can be repeated until all points have been included, and the surface completely triangulated.

Figure 1 shows this process in a simple case with 10 points in each layer of a slice. The starting point is at point 1, which is connected to point 11. Testing of the two possible diagonals (from points 1 to 12 or points 2 to 11), shows that the latter produces triangles with larger minimum angles and hence is the proper choice. A second triangle, consisting of points 2, 11, and 12 is also immediately constructed and the segment between points 2 and 12 becomes the new starting point for further lacing.

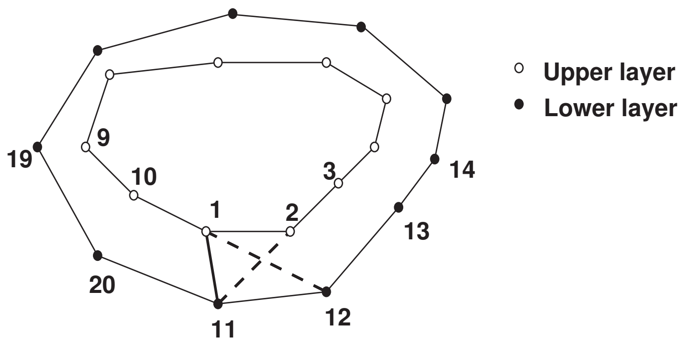

Figure 1: Example of lacing two layers to triangulate the surface enclosing the resulting slice. Points from the upper layer are shown as empty circles, those from the lower as filled circles.

While this basic algorithm worked *most* of the time, there were a number of situations in which additional mechanisms were required. In some cases of extremely irregular surfaces, the lacing scheme had to be augmented with some checks for triangles which are too large, or had to be formed, not between nodes on different layers, but in the plane of one of the layers. Errors frequently occurred in regions of significant concavity, in which triangle segments ran outside the hull of the body, as, for example, in the crescent-shaped regions of the lower lungs. Simple examples of these situations are given in Figure 2. In panel **A** the upper layer describes a smaller surface than the lower layer, and a single point (point P in the figure) is shared by seven triangles, instead of two or three, as for all the other points. Panel **B** shows a situation in which erroneous triangles (shown in thicker, dashed lines) are constructed across a region which is outside the actual surface. Such triangles must be detected and removed, ideally without user input.

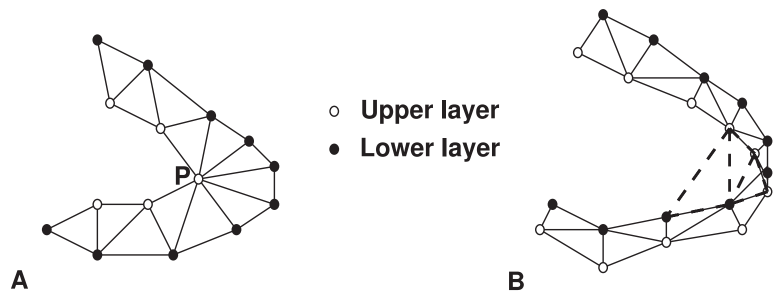

Figure 2: Two examples of special cases in lacing triangles. In **A**, the two layers join to form a very flat surface since one layer does not lie directly over its neighbor. In panel **B**, a sharply curved surface can result in erroneous triangles which lie outside the actual surface (shown as dashed lines.)

4.1.2 Inside/Outside algorithm

Determining when a triangle lay outside the surface was one of several related problems that arose during the model construction (and will be described in more detail below), in which the basic question was whether a point was *inside* or *outside* a surface. To solve this, we again make use of the fact that all the points in tomographic data lay in discrete planes. If a point lies inside the curve outlining a surface on the plane (the "surface curve"), then the algebraic sum of all the angles about the point which are formed between adjacent points on the curve, should be equal to 2π. For a point outside the curve (and hence the surface partially defined by the curve), the sum of the angles is zero. Using this fact, and provided that the points have already been ordered in a strict sequence of first-order neighbors, we can construct a set of tests with which the location of a point, relative to a surface, can be determined. This same concept can be extended directly to three dimensions if, instead of planar angles, the sum of the *solid angles* about a point is computed. However, this calculation requires knowledge of the outward normals on an already complete surface tessellation and is therefore unsuited to this application. (Note that once a triangularized geometry is complete, a computation of the sum of the solid angles is an excellent means of checking for closure and consistency of the outward normals.)

Figure 3 shows the principle behind the inside/outside algorithm, in which the sum of all angles θ_i over the ordered set of nodes about the point P is equal to 2π if P is inside the curve, and 0 if P is outside the curve.

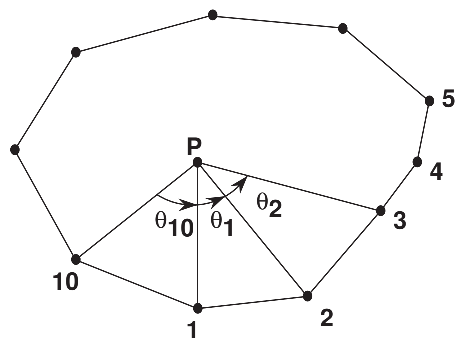

Figure 3: The inside/outside algorithm, used to determine if point P lies inside our outside the outline formed by joining the order set of nodes i. If P is inside, $\sum_{i=1}^{i=N} \theta_i = 2\pi$; if P is outside, the sum is equal to zero.

To apply this technique to the problem of surface triangulation by lacing, we seek to detect not whether points, but triangle connectivities lie outside the surface. Hence we must derive one or more points from the proposed triangle and determine from their position relative to the surface whether or not the triangle itself lies outside. We found that checking the location of the triangle centroid projected on both layers (and eliminating the triangle if the centroid lies outside either) was adequate for most purposes.

4.1.3 Completing the surface model

Once laced into multiple slices, the points and triangle connectivities had to be concatenated to make a complete surface description. We performed this operation on a surface-by-surface basis so that the end result was a set of points and connectivities for each of the surfaces in the complete model. This allowed for easy selection of the surfaces that were to be included in a computation and/or visualization. Before they were ready to use, however, there were two final steps in the model construction process.

The first involved connecting the points which formed the 'ends' or 'caps' of the model, the first and last layer of each surface. We selected two approaches for this, either by triangulating the points in the layer, or by adding a grid of points to fill in the region defined by the points in the end layers, and then triangulating the resulting augmented layer in two dimensions. The choice of method depended on the size of the opening at each end and the desired density and spatial resolution of the model we were constructing. The triangulation of the resulting two-dimensional point set was a straightforward matter and numerous programs exist for this purpose, some even in the public domain (*e.g., voronoi* from the netlib collection, available via electronic mail from netlib@ornl.gov). What is usually missing from these algorithms, however, is a provision for concave curves, in which case erroneous triangles are constructed which lie outside the curve. To remedy this, we implemented a check of all the triangles formed by the two-dimensional triangulation using the inside/outside test described above, applied to the centroid of each triangle.

The final step in constructing a surface model of the geometry involved setting the direction of the outward normal vectors to each of the triangular elements. Finding a normal is easily done by computing the cross product of any two sides of the triangle. Ambiguity remains, however, in the final choice of which of two opposite directions is outward, and which inward. This can be resolved in most cases by applying, once again, a test to the endpoints of both candidate normal vectors to determine which lies inside, and which outside the surface. For the models described here, added normal vectors in both directions to each of the three vertices of the triangle, then tested the endpoints relative to the surface to which the vertex belonged, until we achieved an unambiguous result of one endpoint found inside, the other outside the surface. The encoding of this information into the model files lay in the order in which the points of each triangle were stored and we used the standard convention of ordering points counterclockwise when viewed from the outside of the surface. As a final, complete check of the integrity of the model, we selected a point inside each surface, and computed the total solid angle about that point. If the surface description is complete (no holes), and all the outward normals are correct, the result must be 4π.

4.2 Programs for Surface Segmentation

4.2.1 *trisurf*

The program which performs lacing and final construction of the model was called *trisurf* and is written in Fortran for use under Unix. The *trisurf* program combined a number of modules necessary to create surface descriptions from a set of points. The important assumption underlying the data structures of *trisurf* was that the **points were layered in the z-axis, and had been pre-sorted according to z-value**. Once this condition was met, *trisurf* could take these points and create a fully triangularized surface-based model.

4.2.2 *map3d*

The program used to visualize geometry and scalar data associated with geometry is *map3d*. This program also permitted the editing of triangular meshes, at least in terms of the connectivity of the nodes of the model. There is a separate documentation for *map3d*, which should be available from the same source as this document.

5 Volume Segmentation

5.1 Background

To apply three-dimensional finite element methods to a problem the solution domain must be discretized into a mesh of volume elements. Many problem domains can be described in terms of analytic functions, and automated mesh generation schemes will convert them into regularly spaced, or *structured* grids. Models which are based on a set of fixed, irregularly spaced sample points, on the other hand, are known as *unstructured.* While structured meshes are easier to construct and require less computer memory (since they can be described parametrically), they are poorly suited to irregularly-shaped objects, and especially poorly suited to follow boundaries between regions within the domain (*e.g.,* the organs in a model of the human body). There are, however, techniques by which a set of three-dimensional points can be joined in some optimal sense to form a mesh of volume elements and many of these techniques are based on an extension of the Delaunay criteria to three dimensions [4, 9].

5.1.1 Hybrid meshing approach

Our approach was hybrid in nature, a combination of a regular grid applied within the irregular surface outlines from the digitized MRI images. Starting with the same point set used for surface triangulation, we first added a planar grid of points to each of the layers, then removed all points that were within some tolerance of an existing surface boundary. The spacing of the grid determined the overall density of the mesh. We then concatenated adjacent pairs of grided layers to form three-dimensional slices, which were the input data to a program which automatically generated optimal (in the maximum smallest angle sense of Delaunay) tetrahedral elements. Subsequent quality control of the tetrahedralization consisted of removing tetrahedra which were flat, too large (and hence contained other smaller tetrahedra), or which spanned concave regions of the outer boundaries of the domain. To detect elements which lay outside the domain the location of the midpoint of each side of the tetrahedron was tested against the surface enclosing the domain, using, once again, the inside/outside check (see section 4.1.2); if any part of a tetrahedra lay outside the domain it was removed.

5.1.2 Assigning conductivities

One of the major advantages of the finite element technique over, for example, the boundary element method, is that the effect of inhomogeneities, even those of an anisotropic nature, can be included directly in the formulation of the problem. In the case of electrical activity in the human torso, there exist different tissue types which possess different passive conductivity characteristics, and hence affect the distribution of electric potential and current in the body. The associated step in constructing our finite element model of the human torso was, therefore, to identify those elements which lay in certain regions of the body, and to assign each a conductivity tensor. We located the elements by projecting the centroid of the tetrahedra onto the nearest surface and then subjecting that point to a chain of inside/outside tests on each of the boundary surfaces of the model until its relative position was determined. For example, a point which lay inside the body surface but outside the boundary between subcutaneous fat and skeletal muscle, was part of the fat layer; a point which was inside the body surface, the fat/muscle boundary, and the inner boundary of the muscle, but outside of all other surfaces, was not part of any distinct tissue group, and was assigned a conductivity equal to the mean for the thorax. We did not assign numerical conductivity values until execution of the finite element solver, employing a look-up table based on group numbers assigned to each tissue region.

5.1.3 Sorting slices

The final step in the three-dimensional model construction was the concatenation of the tetrahedralized slices into a complete geometrical description of points, connectivities, and tissue regions. In order to facilitate transfer of the simulation results back to the surface description of the model, we kept tables which described the location of each point, in terms of both the volume and surface point numbers.

6 File Format Summary

6.1 Raw MRI image files

This is the format of the images as they came from the digital tapes produced on the MRI computer.

Format:	binary
Data type:	byte
Record length:	65536 bytes
# of records:	1

Order of data storage: first byte is the upper left pixel of the image; following bytes run left to right and top to bottom of the image. Each pixel is specified by a single byte (256 gray values).

6.2 Converted MRI image files

These files are the output of the CONVERT program, which transformed the single-record raw MRI files into 256, 256-byte records. Each record was equivalent to one line of the image.

Format:	binary
Data type:	byte
Record length:	256 bytes
# of records:	256

Order of storage: each record represents one line of the image; the first record is the top row and each row runs from left to right across the image.

6.3 *CONVERT* Histogram files

These were the optional output of the CONVERT program, the histogram of the gray values in each image.

Format:	ASCII
Data type:	character-coded real and integer
Record length:	variable
# of records	100 (number of bins in histogram)

Record format: 2 character strings, separated by one or more spaces: the first contains the gray value at the centre of the bin, the second contains the number of pixels in that bin

6.4 *New Digitize* data files = raw MRI data files

These were the output files from the MRI digitizing program *New Digitize*. They contained the locations in the image of the sampled points, organized into separate surfaces. The filenames were set as `rltXsYYY_ZZZ.dat`, where `rlt` identified the subject, `X` was the number of tape from which the data was transferred from the MRI machine to the Vax, `YYY` was the level of the slice in millimeters, relative to the lowest layer *recorded*, and `ZZZ` was the running number of the slice. For example, `rlt1s020_001.dat` was from tape 1, at 20 millimeters from the lowest recorded slice, and was the first layer in the series. (Note: the reason this was the first slice, although there were slices recorded lower on the torso, was because these lowest slices were discarded before digitization with *New Digitize*).

Format:	ASCII
Data type:	character and character-coded integer
Record length:	variable
# of records:	variable

Record format: Records are of 2 types: surface header records and pixel co-ordinate records. Surface header records are single lines containing the type of surface described (body surface, lungs, lv and rv endocardial, epicardial, fat, and electrodes) and the number of points which follow to describe the boundary. Following each header record is a sequence of co-ordinate pairs, separated from each other by a single comma (no space). The first number of each pair is the row location, the second number is the column location of each pixel, relative to the top left corner of the image. Data ranges from 0 to 600 along both axis.

6.5 Postscript files

These were "standard" postscript files, divided into four sections: 1) a header or prolog, and contains no executable statements, only text comment lines which are human-readable; 2) a setup section of dictionaries for commonly used commands an character sets; 3) the image data in pixel (image) format [10, 11]; 4) a block of text information for the plot (filename, date stamp and image processing values applied to the image data), followed by the vector draw instructions for the plot of the digitized values (optional).

6.6 Scaled MRI data files

The input to the *mrimetric* program were raw MRI data files and the output were files in the scaled MRI data format. The layout of these files was almost identical to the raw

MRI data files described above, with the co-ordinates converted from pixel addresses to units of millimeters. The origin of each layer was located on the z-axis of the torso, as set in *mrimetric* . Filenames were set to `rlzxxx_yyy.dat`, where `xxx` was the z-value of the layer, in millimeters, relative to the lowest level actually used in the model, and `yyy` was the running number of the layer. Thus `rlz005_001.dat` contained data from the second layer, at a level of $z = 5$ mm.

Format:	ASCII
Data type:	character and character-coded integer
Record length:	variable
# of records:	variable

Record format: Records are of 2 types: surface header records and pixel co-ordinate records. Surface header records are single lines containing the type of surface described (body surface, fat, muscle, epicardium, left lung, right lung, lv and rv endocardium, and electrodes) and the number of points which follow to describe the boundary. Following each header record is a sequence of co-ordinate pairs, separated by a single comma (no space). The first number of each pair is the x co-ordinate, the second number is the y co-ordinate, relative to a central origin; both numbers are in units of millimeters.

6.7 Interpolated layer files (`.torso`, `.fat`, `.muscle`, etc.)

A converted MRI data file was the input to the interpolation program (on the Vax), whose output was an interpolated layer file. For each single MRI data file (containing points for all the surfaces in one layer) the interpolation program sorted the data into individual surfaces and output a separate file for each. Hence, from a file like `rlz125_026.dat` came files for each of the surfaces in that layer: `rlz125_026.torso`, `rlz125_026.fat`, and `rlz125_026.muscle`, etc. The format of these files was even simpler than that of the MRI files, just lists of x, y, and z co-ordinates.

Format:	ASCII
Data type:	character-coded integer
Record length:	variable
# of records:	variable

Record format: Each record consists of a set of x, y, and z co-ordinates in a list-directed real format, separated by multiple blanks. There is no indication of how many points are in the files so reading must continue until an end-of-file is hit.

6.8 Slice Files (`.slice`)

The slice files followed very simply from the interpolated layer files and were constructed by concatenating the contents of two successive layer files (typically using the VMS copy command). Filenames included the subject *eg.* `rl`, the slice number, and the surface, with the extension `.slice`. For example `RLZ_FAT.SLICE97` contained the points in the fat surfaces from slice 97, which were the same points as in fat layers 97 and 98.

Format:	ASCII
Data type:	character-coded integer
Record length:	variable
# of records:	variable

Record format: Each record consists of a set of x, y, and z co-ordinates in a list-directed real format, separated by multiple blanks. There is no indication of how many points are in the files so reading must continue until an end-of-file is hit. There are two sections to the files: the first contains the points from the lower *i.e.* smaller values of z) layer of the slice; the second set of points are form the upper layer of the slice.

6.9 `.pts`, `.fac`, and `.tetra` files

This file type is the one we used more often to store geometry in this project. This is not a single file type, but more accurately a family of ASCII files which each contain different components of the geometry. A file with the extension .pts contains the locations of the nodes of the geometry, with a single point described on each line by its x, y, and z Cartesian coordinate in floating point format. The connectivities of the nodes into triangles are described in files with the .fac extension, each line containing three integer values which are indices to points in a .pts file. Index values begin at 1 (not zero). For tetrahedral elements, we have files with the .tetra extension, each line containing a set of four indices to nodes in a .pts file. All of the .pts, .fac, and .tetra files can also have an optional identifying integer value appended to each line of the file, which defines the "group" number for the node or element. Group numbers can be freely assigned as we have established no fixed conventions.

6.9.1 `.pts` files

Format:	ASCII
Data type:	reals coded as ASCII strings
Record length:	3 or 4 reals
# of records:	one for every point

Record format: Each record contains an ASCII coded set of co-ordinates in three-dimensional space, stored as X, Y, and Z, respectively. In a new form of the `.pts` file, a forth entry has been added to each record, a scalar which is associated with the point it shares the record with. These values are normally the "group numbers", which are determined as part of finding the conductivities of the elements of the finite element model.

6.9.2 `.fac` files

Format:	ASCII
Data type:	integers coded as ASCII strings
Record length:	3 or 4 integers
# of records:	one for every triangle

Record format: Each record contains an ASCII coded set of pointers to points in a `.pts` file (usually of the same name with the `.pts` extension). These three points, when connected, form a triangle on the surface of the model somewhere. A scalar value can be appended to each record as a freely defined value associated with the triangle.

6.9.3 `.tetra` files

Format:	ASCII
Data type:	integers coded as ASCII strings
Record length:	4 or 5 integers
# of records:	one for every tetrahedron

Record format: Each record contains an ASCII coded set of pointers to points in a `.pts` file (usually of the same name with the `.pts` extension). These four points, when connected, form a tetrahedron in the volume of the model. A scalar value can be appended to each record as a freely defined value associated with the triangle.

7 Summary

We have constructed two torso models from a single set of MRI scans, one which uses all the images and maintains an internodal spacing of about 5 mm (high resolution model), the other utilizing every second MRI scan, with a mean internodal spacing of 11 mm (medium resolution). The models contain 42,000 (11,000) surface nodes joined to form 85,000 (19,000) triangles and 82,000 (22,000) total nodes to form 500,000 (130,000) tetrahedra, where the first numbers are for the high resolution and the number in parentheses are for the medium resolution model.

8 Glossary of Terms

Below is a list of the commonly used terms in this documentation (and the research it describes).

Layer: refers to a set of points which all lie in the same plane and describe a set of outlines in cross section. Typical examples of this would be the digitized datapoints from a single MRI image.

Slice: is a three-dimensional structure, composed of a pair of adjacent layers. Exact usage here is somewhat vague when we combine more than 2 layers (is this still a slice, or a set of slices?) but normally we intend a slice to represent a pair of layers.

Grid: is a set of regularly spaced points in a single plane. Grids are typically added to layers to fill in the space between the surface outlines, *eg.*to close a surface at the top and bottom. Grids are also added to each layer of a model in order to tetrahedralize it for use with the finite element solver.

Triangularization: is the act (art?) of connecting points into triangular surface elements. This can be done in two and three dimensions, and can be 'optimal' in some sense of creating elements as close to equilateral as possible. In the case of triangulating surfaces based on tomographic data, as described here, we have a somewhat constrained triangulation we refer to as *lacing* since we simply linked pairs of point sequences from adjacent layers.

References

[1] C.R. Johnson, R.S. MacLeod, and P.R. Ershler. A computer model for the study of electrical current flow in the human thorax. *Computers in Biology and Medicine*, 22:305–323, 1992.

[2] C.R. Johnson, R.S. MacLeod, and M.A. Matheson. Computer simulations reveal complexity of electrical activity in the human thorax. *Computers in Physics.*, 6(3):230–237, May/June 1992.

[3] M.R. Stytz, G. Frieder, and O. Frieder. Three-dimensional medical imaging: algorithms and computer systems. *ACM Trans. Comp. Surveys.*, 23(4):421–499, 1991.

[4] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In F.K. Hwang and D.Z. Du, editors, *Computing in Euclidean Geomtry*. World Scientific, 1992.

[5] D.T. Lee and B.J. Schachter. Two algorithms for constructing a Delaunay triangulation. *Int. J. Comp. Inf. Sci.*, 9:219–242, 1980.

[6] H. Christiansen. MOSAIC triangulation algorithm. In *MOVIE.BYU Program Manual*. Engineering Computer Graphics Laboratory, Brigham Young University, Provo, Utah, 1987.

[7] I. Vesely, B. Eickmeier, and G. Campbell. Automated 3-D reconstruction of vascular structures from high defninition casts. *IEEE Trans Biomed Eng*, BME-38:1123–1129, 1991.

[8] R.R. Mercer, G.M. McCauley, and S. Anjilvel. Approximation of surfaces in quantitative 3-D reconstructions system. *IEEE Trans Biomed Eng*, BME-37:1136–1146, 1990.

[9] A. Bowyer. Computing dirichlet tesselations. *Computer J*, 24:162–166, 1981.

[10] Adobe System Incorporated. *Postscript Language Tutorial and Cookbook*. Addison-Wesley, 1985.

[11] Adobe System Incorporated. *Postscript Language Reference Manual*. Addison-Wesley, second edition, 1990.