

TR-H-262

**MRI Toolbox: A MATLAB-Based System for
Manipulation of MRI Data**

Mark K.TIEDE

1999.1.8

ATR人間情報通信研究所

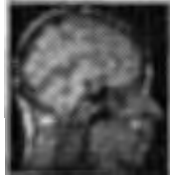
〒619-0288 京都府相楽郡精華町光台2-2 TEL: 0774-95-1011

ATR Human Information Processing Research Laboratories

2-2, Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0288, Japan

Telephone: +81-774-95-1011

Fax : +81-774-95-1008



MRI Toolbox

A MATLAB-based system for manipulation of MRI data

Mark K. Tiede

ATR Human Information Processing Laboratories

Summary: The MRI Toolbox provides functions for parsing scanner-generated MRI data files, supports a custom class `mrivol` facilitating manipulation of MRI volumes at variable data resolutions, includes a GUI for interactive viewing, measuring, and oblique reslicing of data, and offers a range of user-extensible analysis routines.

Dependencies: The tools require Matlab version 5.1 or later, and have been tested on Mac, Wintel, and Unix (SGI and Sun) platforms. Compiled MEX files are used to improve response where appropriate (source included); M-file equivalents permit the tools to function in their absence. Certain analysis routines depend upon the availability of the Image Processing Toolbox.

January 1999

1. The *mrivol* custom class

1.1 Motivation

MR images are typically reconstructed from an excited volume of interest (a 'slice') collapsed across depth to form the output image. Volumetric data is obtained by collecting parallel slices at repeated offsets through the region of interest:



Figure 1: Sagittally-oriented volumetric MRI

If the goal is measurement of the MR-sampled volume the resulting image sets are difficult to work with for several reasons. For instance, the number of imaged slices bracketing a volume typically ranges from 20 to 50, requiring substantial book-keeping to correctly track load order, orientation, *et cetera*. In addition, as a consequence of the collapse across depth the effective resolution of MRI volumes is less along the axis of acquisition, so separate scaling factors must be maintained for both the image plane and its normal. Annotations made during data collection (e.g. target articulation) must be maintained as separate, potentially confusable records. Clearly it is advantageous to store all information associated with a volume (data,

parameters, and annotations) as a single object, preferably one derived automatically from the original scanner-generated file.

The Matlab (MathWorks, Inc.) environment has several advantages for MRI analysis, including its support for multi-dimensional arrays, user-extensible filter and analysis functions, and plotting capabilities consistent across platforms. A straightforward representation of a MRI volume within Matlab is possible by simply mapping each image slice to a three-dimensional array: [height x width x image#]. However, Matlab performs all arithmetic using an eight byte (double precision) format, so the memory requirements for a typical [256 x 256 x 30] volume can rapidly become overwhelming. One workaround suggested and to some degree supported by MathWorks is to store the data in 'uint8' (byte) format, especially if its intensity range lies within 8 bits. The potential intensity range of MRI data is 15 bits, however, with vocal tract data typically reaching 12 for extreme values.

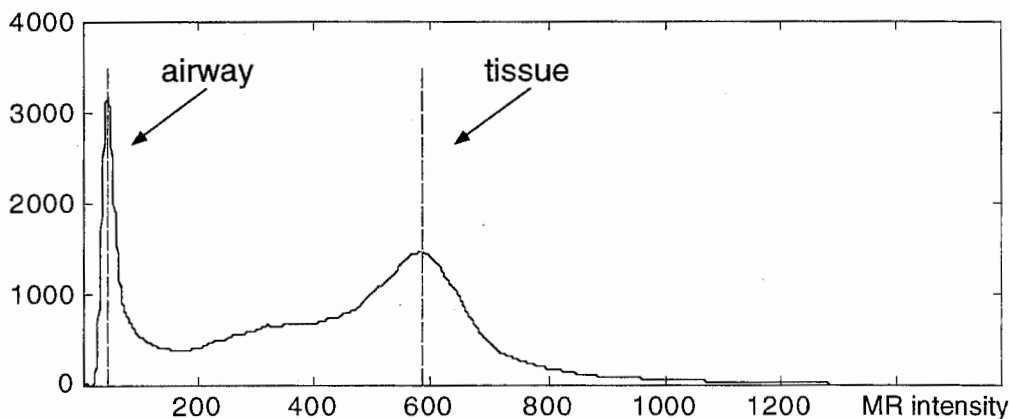


Figure 2: Trimmed histogram of vocal tract volume data

Simple min/max scaling of such data to 8 bit precision results in the useful range of the data being squashed into the low end of the byte range. Scaling using a standardized cutoff (say 1000 given the above histogram) gives more useful results and permits consistent comparison across volumes. However, while generally satisfactory for viewing volumes, the loss of precision inherent in any scaling is a problem for analysis, especially in automated or semi-automated feature detection tasks. Ideally then routines for manipulation of MRI data should be flexible with respect to storage requirements, operating transparently on data mapped to different precisions.

Within the MRI Toolbox, the desiderata of unified data stored with flexible precision and its associated parameters and annotations are encompassed through the use of a custom storage class called `mrivol`. Objects of this type encode all information associated with a volume, but can be manipulated as simple variables. Support exists for creating `mrivol` objects directly from scanner format files, or they may be generated from sets of separate images or as the output of toolbox manipulations (e.g. reorientation or resizing). Overloaded methods function as wrappers to mainstream Matlab and Image Processing Toolbox (IPT) procedures. From the user's perspective the `mrivol` class permits volume data from contrasting conditions (say */i/* vs. */a/*) to be named, stored, and otherwise manipulated as a single object, with component parts (e.g. a specific slice, scaling factors, etc.) accessed as necessary.

1.2 Anatomy

Here is an example `mrivol` object:

```
mt_aa_c =
  COR mrivol object: [256 x 204 x 30] (int16)
  dims = [257.092 x 204.665 x 116] (mm)
  mpp = 1.0082
  isi = 4
  range= [0 2227]
  info =
      ID: 'B32186.0'
      Subject: 'M.Tiede'
      Comments: 'AA'
      Date: '96-08-17'
      Time: '15:52:55'
  Orientation: 'COR'
      LeftSide: 'RIGHT'
      BottomSide: 'FEET'
      NumImages: 30
      SideLength: 256
      Thickness: 4
      Pitch: 4
      FOV: 258.1000
      AcqTime: 139
  ImagingData: 'SE/256, TR=920, TE=15, NEX=1'
  LoadOrder: 'posterior ==> anterior'
```

In this case `mt_aa_c` is an `mrivol` class variable composed of 30 coronally oriented image slices stored using signed two byte precision (`int16`). Each image is [256 x 204] pixels in size with a mm per pixel scaling factor (`mpp`) along the image plane of 1.0082. The inter-slice-interval (`isi`) specifying the mm offset between slices along the axis of acquisition is here 4 mm. By applying these to the pixel/slice dimensions

of the volume isotropic mm dimensions (`dims`) are obtained. The intensity range for this volume is [0 2227]. The `info` structure can consist of anything the user wants; in this case it has been initialized to annotations derived from the header of the Shimadzu format data file from which `mt_aa_c` was created.

Each of these components may be accessed individually. For example,

```
>> mt_aa_c.mpp                % access the 'mpp' field
ans =
    1.0082

>> mt_aa_c.type              % access the orientation 'type'
ans =
COR

>> mt_aa_c.isi = 5;          % change the 'isi' value to 5 mm
```

The `range` and `dims` fields are read-only. The `info` field may be accessed as a complete structure, or by individual member (if not found a new `info.element` member is created). For example,

```
>> mt_aa_c.src = 'shimadzu';
```

adds a new field `src` to the `info` structure with value 'shimadzu.'

Image data may be accessed in one of several ways. The upper leftmost pixel of the first image of the volume may be accessed by `mt_aa_c(1,1,1)`. Similarly a range of data may be obtained using standard Matlab array addressing as in `mt_aa_c(10:50,5,[1 3])`, which would return rows 10 through 50 of the fifth column from volume slices 1 and 3. Since it is often convenient to access a single image slice, `mrivol` supports a non-standard use of the curly brace notation in which all rows and columns of the slices specified within braces are returned. For example, `mt_aa_c{1:5}` returns the first five slices of the volume, and is equivalent to `mt_aa_c(:,:,1:5)`. The entire volume may also be accessed using the `mt_aa_c.data` syntax. Each of these access methods return data in the same format as stored (`int16` in this case). Explicitly typecasting a format type as in `double(mt_aa_c)` for instance returns the specified data in the appropriate format.

1.3 Methods

An `mrivol` object is created using the following syntax:

```
>> m = mrivol(data, type, mpp, isi, info);
```

where `data` is image volume data [`height x width x slices`], `type` is one of SAGittal, CORonal, TRaNsverse, or OBLique, `mpp` and `isi` give the mm/pixel and inter-slice-interval, and `info` is an arbitrary annotation object. Any or all of these may be empty. Data may be given in one of `uint8`, `uint16`, `int16`, or `double` formats.

Ordinarily users will create `mrivol` objects using a function designed to parse scanner-generated data files, as in

```
>> mt_aa_c = LoadSMT('B32186.0');
```

Currently data loading functions exist for Shimadzu format and individual raw or supported format image data files (a DICOM parser is under development).

Overloaded methods for the `mrivol` class include the following:

```
>> length(mt_aa_c)           % returns number of slices
ans =
    30

>> size(mt_aa_c)            % returns pixel/slice dimensions
ans =
    256    204    30
```

and the various type converters (`uint8`, `double`, etc.). Image Processing Toolbox functions with overloaded methods include `im2bw`, `imcrop`, `imhist`, `imrotate`, `montage`, and `movie`.

The `mrivol` class also supports several special purpose methods, of which three of the most useful are `resize` (to up/down sample the image data while preserving mm scaling), `reslice` (to resample the image data in either standard or arbitrary orientations), and `slicer` (the core procedure for oblique slice resampling). For example,

```
>> mt_aa_x2_s = resize(reslice(mt_aa_c, 'SAG'), 2, 'cubic');
```

produces a new sagittally oriented volume from the original coronal dataset with twice the number of image pixels per mm.

Although MRI data is intensity-based, a common analytic procedure is convert thresholded data to binary-valued form (e.g. air/tissue boundary extraction). The `threshold` method performs exactly this function, converting the resulting data to `uint8` format automatically. Because intermediate interpolation of the resulting

binary data is problematic, the MRI Toolbox supports the shape-based interpolation algorithm of Raya and Udupa (1990), through the `ShapeInterp` and `DistMap` methods. In this approach binary data are characterized by the shortest distance from an edge (i.e., a 1/0 juncture), and these distance values (rather than the original binary data) provide the basis for interpolation. The results (thresholded to give output binary images) track intermediate shapes with much less distortion than simple interpolation techniques.

2. IRMA: *mrivol* object GUI

2.1 Overview

Visual inspection is an essential step in the analysis of any MRI data. Within the MRI toolbox the `mrivol` class is complemented by the **IRMA*** interactive viewing tool, which is intended to facilitate such inspection. IRMA provides a graphical interface for display, measurement and animation of the slices comprising a volume. It permits simple interactive control of displayed size and brightness, and supports extraction of oblique views through the volume with arbitrary orientation. IRMA also provides support for user-extensible analysis procedures triggered by `MouseDown` events within the displayed volume.

IRMA is invoked from the command line specifying the name of an `mrivol` object, with an optional colormap argument:

```
>> irma(mt_aa_c, gray(256));
```

More than one IRMA tool may be open simultaneously. As it is often useful to compare equivalent images from contrasting conditions IRMA supports ‘synchronizing’ open viewers so that all increment their displayed slices simultaneously.

There are three main windows associated with each IRMA viewing tool: a ‘Viewer’ control window for interactive control of display scaling and animation, a ‘Display’ window that presents the image data, and a ‘Slicer’ window for specifying the parameters for oblique reslicing. Each of these windows support additional functions via menu options and accelerator key shortcuts. While IRMA windows are hidden from mainstream Matlab graphics calls, images may be exported to a standalone window object supported by the MRI toolbox (called `smartwin`) that supports measurement, resizing, and annotation, and is accessible to Matlab functions for printing, saving, et cetera.

* IRMA: Imagerie par Résonance Magnétique (Animée). So OK you try making a name out of MRI...

2.2 Viewer window

The IRMA viewer window provides interactive control of display parameters for the loaded volume.

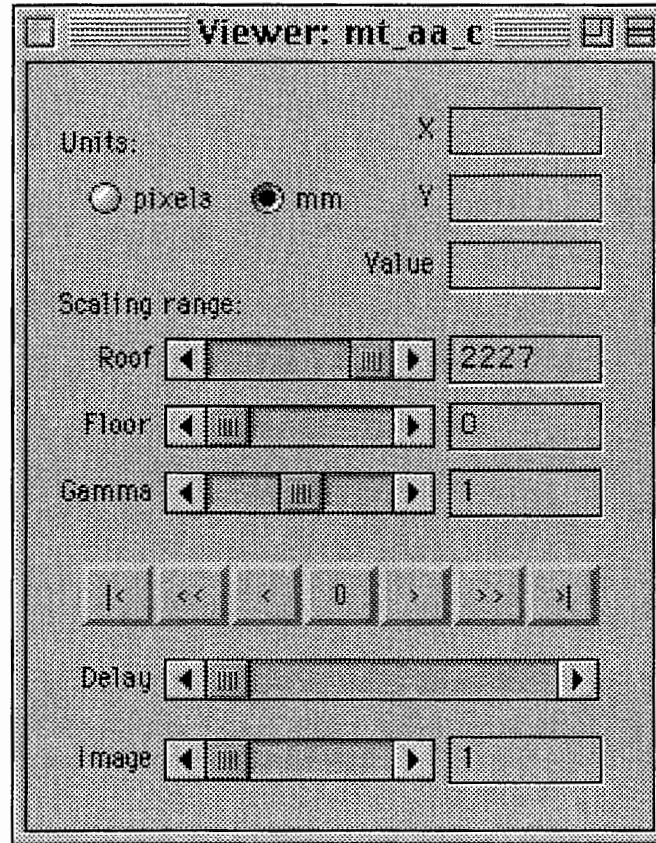


Figure 3: IRMA Viewer control window

The `Units` switch toggles between pixel and mm axis units (if the `mrivol mpp` field is empty, only pixel units are possible). The `Scaling range` controls are used to map the intensity range of the volume to the displayed colormap; values may be either selected by slider or entered explicitly. The pushbuttons control animation: the left and rightmost buttons reset to the first/last image of the volume, “<<” and “>>” initiate reverse and forwards animation, “<” and “>” step one image backwards or forwards, and the “0” button halts ongoing animation. The displayed image may also be selected using the `Image` slider or entry field. Animation speed is hardware and platform dependent, however the `Delay` slider control permits interpolation of an interval between updates if the refresh rate is too fast. The `x`, `y`, and intensity `Value` fields are updated continuously when the cursor is moved over the (active) `Display` window.

When either the Viewer control or Display windows are active, IRMA supports certain functions by a menu interface (depending on the platform, this may be either attached to each window directly, or appended to the global Matlab menubar). Most of these functions may also be invoked using the associated accelerator key, as in “D” (suitably modified) to select “Duplicate Window.”

VIEWER		Movement	
About IRMA...		Reset to Head	⌘H
Reset Default Position	⌘R	Cycle Backwards	⌘A
Show Reference Grid	⌘G	Step Backwards	⌘B
Interpolate Slices	⌘I	Stop Cycling	⌘X
Mouse Down Behavior	▶	Step Forwards	⌘F
Duplicate Window	⌘D	Cycle Forwards	⌘Z
Oblique Slicer	⌘S	Reset to Tail	⌘T
Export Data...	⌘E	Step Synchronize	⌘Y
Close IRMA	⌘W		

Figure 4: IRMA Viewer and Display Menu options

Among these functions the `About` entry presents built-in help for IRMA using the `helpwin` mechanism. `Reset Default Position` repositions the Viewer and Display windows in their original size and location. `Show Reference Grid` toggles the superimposition of dotted reference lines upon the currently displayed image. When `Interpolate Slices` is toggled intermediate images are interpolated at 1mm increments when stepping between actual volume slices (typically separated by 3 - 5mm `isi` offsets). `Mouse Down Behavior` supports selection of user-defined routines triggered by a mouse click over the displayed image (see section 2.6). `Duplicate Window` creates a copy of the currently displayed image in a separate stand-alone `smartwin` object (discussed below).

The Slicer window controlling oblique slicing parameters is not initially visible; the `Oblique Slicer` menu option is used to make it active. It is possible to create a new displayed object as a function of IRMA-based analysis (e.g. `Extract Sections`), and the resulting `mrivol` can be exported to the base workspace using the `Export Data` menu option. `Close IRMA` provides a means of deleting the currently active viewer.

The Movement menu options shadow the Viewer control window animation buttons (the associated accelerator keys provide efficient control of animation from the keyboard). In addition, the `Step Synchronize` option when toggled causes a single step forward or back in the current IRMA object to be echoed by all the currently active IRMA viewers (however active animation is not tracked).

2.3 Display window

The IRMA display window displays the volume image slice at the current Z offset (if the `Interpolate Slices` option is active the intermediate interpolation is displayed). The intensity range of the volume mapped to the active color map is displayed in the rectangular bar to the right.

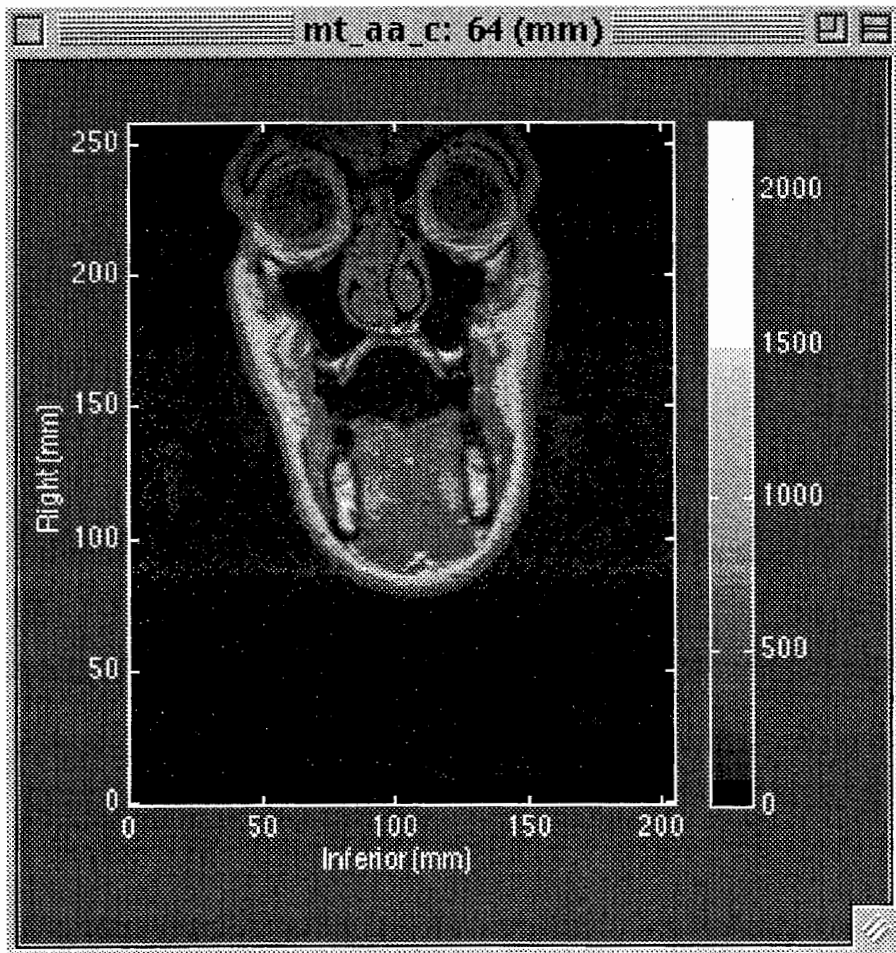


Figure 5: IRMA Display window

The axis units are controlled by the `Units` setting of the Viewer control. Fast (pixel doubling) resizing is supported by resizing the window (use the `Reset Default Position` menu option to restore the default size). As the cursor is moved over the

(active) Display window, the current X and Y offset values in axis units are displayed within the Viewer control, along with the intensity value of the nearest voxel.

IRMA provides support for identifying and labelling orientation, not just of the original volume, but of oblique slices through it. Volume data are displayed using the following orientations:

```
1st image ==> last image
    right --> left           (Sagittal)
posterior --> anterior      (Coronal)
    inferior --> superior    (Transverse)
```

IRMA thus presents sagittal data viewed from subject right, coronal data viewed from subject front, and transverse data viewed from underneath (looking up).

IRMA's mouseDown behavior (i.e. a mouse click with the cursor over the displayed image) depends on the current `Units` type. When `pixels` are selected, any mouseDown in the Display window is passed to the mainstream Matlab `zoom` function, permitting rapid enlargement of an area of interest. When `mm` units are selected, in default of a user-selected procedure the clicked location (X, Y, Z) is echoed to the command line (a right button or otherwise modified click leaves a plotted crosshair at the clicked location). When a user procedure is active (e.g. `iu_threshold` supporting seed-based binary thresholding) the mouseDown is passed to it for processing. Regardless of user procedure, a double-click in the window border surrounding the image causes the displayed image to be refreshed.

2.4 Slicer window

Oblique reslicing through an `mrivol` dataset is separately controlled through an additional window. Initially this 'Slicer' control window is not visible; selecting `Oblique Slicer` from the Viewer window menu hides the Viewer control window and displays the slicing controls. The Viewer controls are restored when the Slicer window is itself closed, either by depressing the "Done" pushbutton or by selecting the platform-dependent window close option. Slicing requires that the displayed `mrivol` have both `mpp` and `isi` fields defined (to permit isotropic scaling).

At the top of the Slicer window is a wireframe representation of the displayed volume showing the current slice plane. Its orientation may be changed by clicking and dragging within the window. Use the “reset” button to restore the original orientation. This display is informational only; use the controls beneath it to change the location of the slice plane relative to the volume.

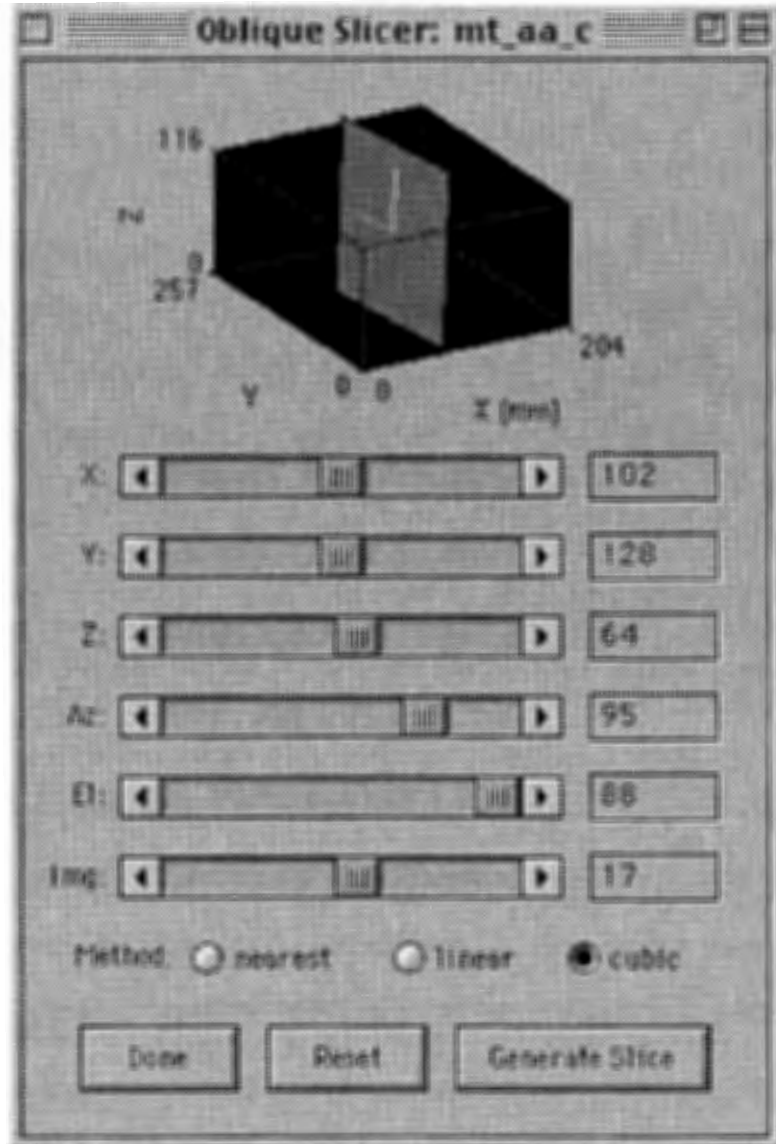


Figure 6: IRMA Oblique Slicer control window

The slice plane orientation is controlled by setting an offset point within the volume, and azimuth and elevation angles defining a plane through it. Volume coordinates are in mm relative to the lower left corner of the first slice of the volume. The azimuth and elevation angles are specified CCW from the horizontal in degrees. As the Z value changes, the associated image slice (interpolated if necessary) is shown in the

Display window, and the current (X, Y) offset values are updated to reflect its projection onto that slice. It is also possible to select an image slice explicitly using the `Img` control.

While the Slicer window is active a reference line is superimposed on the image shown in the Display window corresponding to the current offset and azimuth orientation of the slice plane through that image slice. The offset can be set directly from the image by clicking and dragging on the midline circle; similarly the azimuth can be changed by dragging the circle on the line/frame intersection.

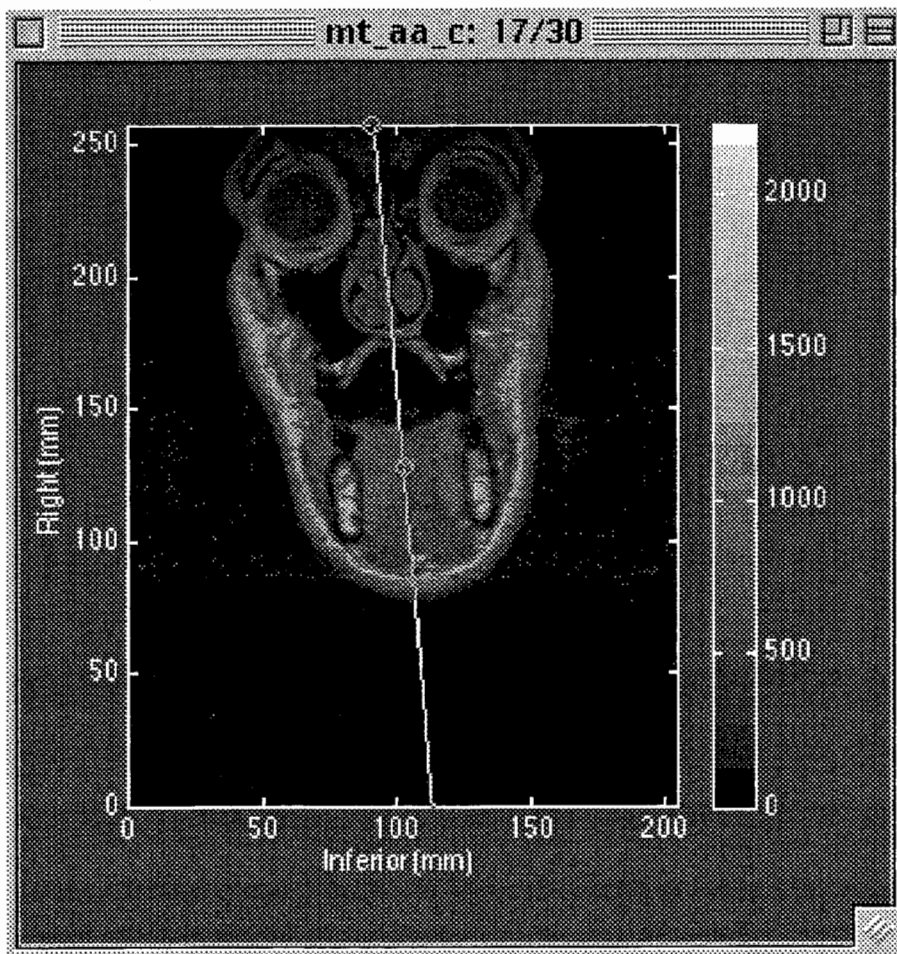


Figure 7: Display window with active slicer reference

Once the parameters for slicing have been established, depressing the “Generate Slice” pushbutton computes the interpolated oblique image and displays the result in a stand-alone `smartwin` object. The maximum dimension of the resulting image is determined by the norm of the volume dimensions, however blank rows and columns are trimmed from the result. Three interpolation methods are supported:

nearest neighbor, trilinear interpolation, and tricubic (Catmull-Rom spline-based; Arata 1995). The active `method` is selected by the corresponding radiobutton control within the Slicer window.

The Slicer window supports a separate set of menu options, all with associated accelerator keys. The `Next Image` and `Previous Image` entries provide a means of nudging the currently displayed image. `Pop Display` is useful for finding an occluded Display window in a crowded screen display.

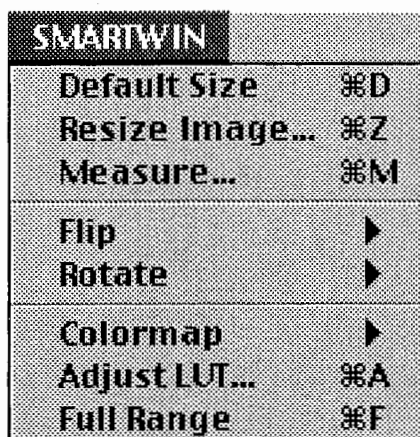
SLICER	
Next Image	⌘F
Previous Image	⌘B
Pop Display	⌘D
Invert Orientation	⌘I
Reset	⌘R
Generate	⌘G
Extract Sections	⌘E
Close SLICER	⌘W

Figure 8: IRMA Slicer Menu options

`Invert Orientation` rotates the current azimuth by 180° and inverts the sign of the current elevation. The `Reset` entry restores the default slice orientation of mid-image offset, and 90° azimuth and elevation (it also restores the default wireframe volume perspective. `Generate` is another way to initiate slice computation. `Extract Sections` is used to initiate the process of vocal tract cross-section extraction (discussed below).

2.5 smartwin objects

The MRI Toolbox makes extensive use of a specialized image display function called `smartwin` that supports intensity scaling, reorientation, resizing, and various other options in a window accessible to mainstream Matlab calls (for printing, annotating, et cetera).



SMARTWIN	
Default Size	⌘D
Resize Image...	⌘Z
Measure...	⌘M
Flip	▶
Rotate	▶
Colormap	▶
Adjust LUT...	⌘A
Full Range	⌘F

Figure 9: Smartwin Menu options

A `smartwin` may be resized in the usual way using fast pixel doubling, after which the `Default Size` option can be used to restore the original scaling. The `Resize Image` option brings up a dialog supporting (optionally filtered) high quality resizing using bilinear or bicubic interpolation. `Measure` uses the IPT function `getline` to obtain the coordinates of clicked locations on the image, reporting X and Y positions, segment lengths, angles, and total length at the command line in the units of the figure. The `Flip` and `Rotate` options can be used to reorient the image while preserving axis labelling.

A `smartwin` object also supports three ways to change the displayed intensity mapping. The `Colormap` itself may be set to one of 16 built-in models. `Adjust LUT` permits control of the display gamma, and the minimum and maximum intensity values mapped to the range of the current colormap. Finally, the `Full Range` option sets the minimum and maximum values to correspond to those of the image.

2.6 IRMA user procedures

With `mm` selected as the current units type, a mouse click with the cursor positioned over the Display window image causes the current location (X, Y, Z) to be echoed to the command line (a right button or otherwise modified click leaves a

plotted crosshair at the clicked location). User extensions to this default behavior are supported through the mechanism of customizable mouse-down procedures, which are installed by choosing `Select` from the `Mouse Down Behavior` menu option. After configuration mouseDown events are passed to the active user procedure for specialized processing. For example, `iu_boundary` is a sample user procedure supporting air/tissue boundary detection:

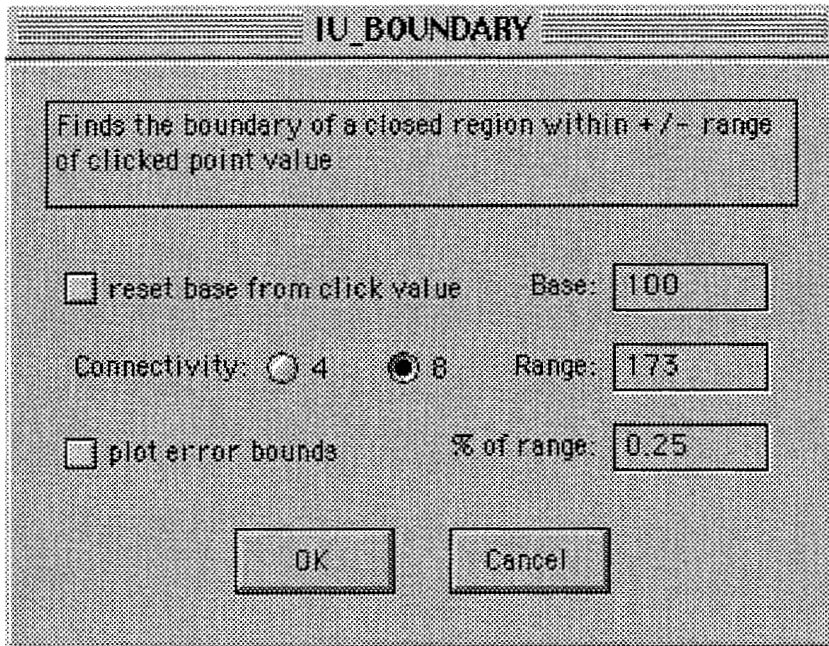


Figure 10: `iu_boundary` user procedure configuration dialog

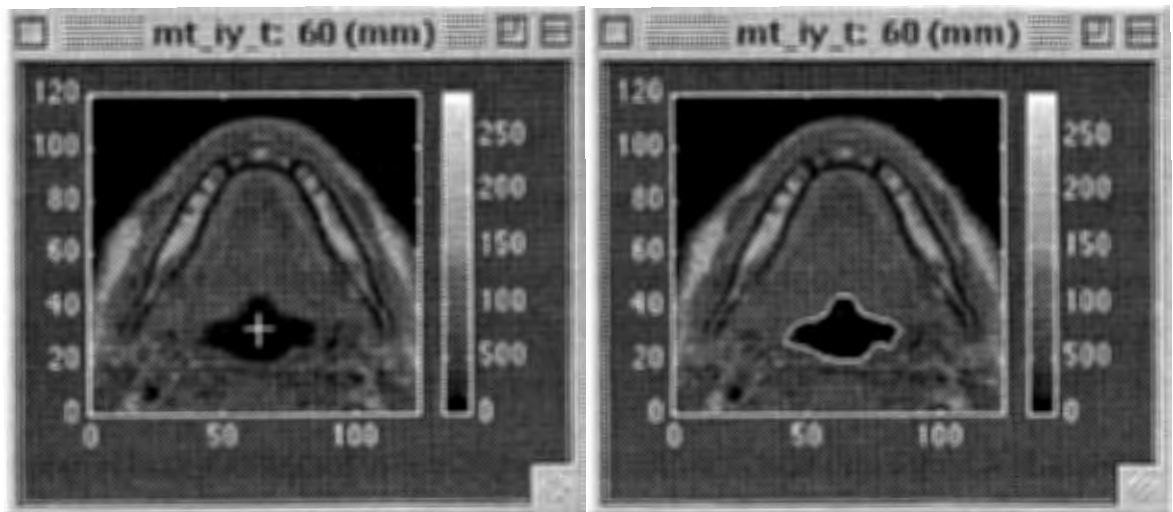


Figure 11: application of `iu_boundary`; clicking in airway (left image) results in detected boundary (right image)

IRMA provides three command line options for interacting with user procedures:

```
data = irma('get')      % retrieve user proc data
params = irma('show')   % show current user proc parameters
irma('set', params)     % replace current with specified params
```

Each user procedure is free to customize the internal structure of both the `data` and `params` (configuration state) variables maintained by IRMA. The `iu_boundary` procedure for example returns the X, Y, Z coordinates of the detected outline in response to `irma('get')`.

2.7 Vocal tract cross-section extraction

The MRI toolbox was developed to address problems encountered in MRI-based analysis of vocal tract shapes characteristic of sustained vowels (Tiede & Yehia 1996). An essential part of this analysis is the extraction of vocal tract cross-sections approximately orthogonal to the tract, whose orientation is established through an extension of the ‘semi-polar’ grid originally proposed by Heinz & Stevens (1964). To permit consistent grid scaling across subjects, IRMA includes support for deriving the extraction grid automatically from four physiologically based landmarks identified on a midsagittal projection through the volume of interest (for details and motivation of this approach see Tiede 1999). As currently implemented cross-section extraction is supported for coronal and transverse volumes only.

The first step towards establishing the extraction grid with this procedure is to establish the orientation (offset, azimuth and elevation) of the midsagittal projection. Because the orientation is used as the basis for cross-section extraction along the grid this provides a mechanism for correction of subject head misalignment with respect to the sampling plane. Once determined, select `Extract Sections` from the `Slicer` window menu; this opens the midsagittal projection through the volume in a `smartwin` object, and displays a grid initialization dialog (click `OK` to continue):

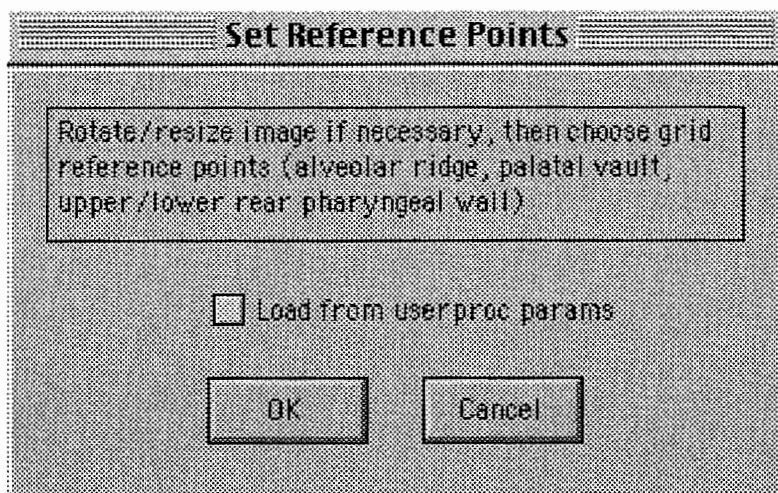


Figure 12: grid initialization dialog

The grid layout is determined by interactively identifying in turn the location of the four landmarks on the midsagittal projection: the base of the alveolar ridge, the highest point of the palatal vault, the point on the rear pharyngeal wall adjoining the anterior apex of the second vertebra, and the meeting of the rear pharyngeal wall with the arytenoid cartilage. Move the mouse over the image location corresponding to each landmark and click; an asterisk will appear at the selected location. The `delete` key can be used to remove a misplaced landmark.

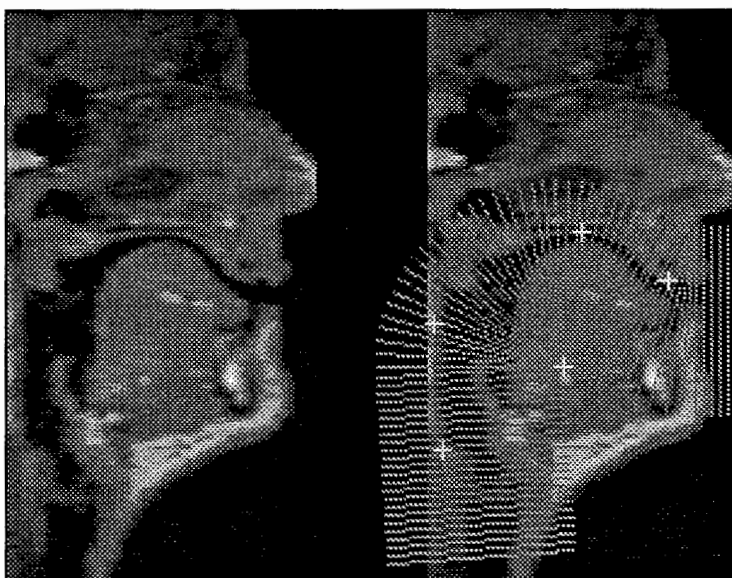


Figure 13: application of cross-section extraction to /i/; midsagittal projection (left) with superimposed grid (right)

Once the four landmarks have been chosen press the `Enter` key to initiate cross-section grid extraction. The derived grid is superimposed on the midsagittal

projection, and the results of section extraction (a derived `mrivol` object) are displayed in a new IRMA viewer. The derived `mrivol` uses the `info` field to store details of the extraction parameters used, and includes the midsagittal projection image, which is displayed and synchronized by IRMA to show the offset of each displayed cross-section along the tract.

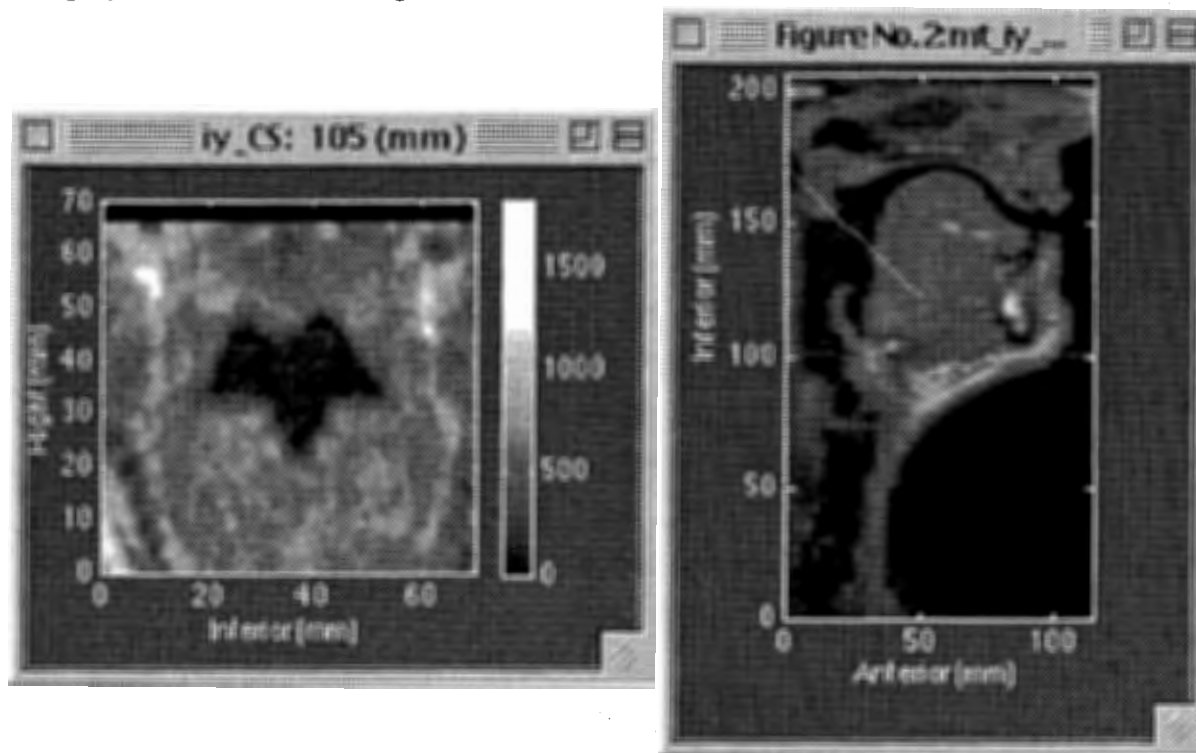


Figure 14: derived cross-section `mrivol` (left) with attached midsagittal projection (right) showing offset along the tract

If the `Load from userproc` checkbox of the initialization dialog is selected a previously established set of landmarks can be used to re-establish the grid, bypassing the interactive selection step. The landmark coordinates are stored as the `userproc params` value after each successful extraction and as such they may be retrieved or reset using the command line `get/set` mechanism described in section 2.6.

3. MRI Toolbox Reference

This section provides descriptions of each toolbox function, showing purpose, argument structure, and examples. A list of the main MRI toolbox functions with one line descriptions can be obtained by entering `help mri` from the command line prompt. Each function of the toolbox is self-documenting, so that entering `help <function>` provides detailed help for that function.

3.1 General

3.1.1 **convert** - filtered data format conversion

```
usage: outData = convert(inData, newFormat, clip)
```

supported input formats: uint8, uint16, int16, double

newFormat can be one of

```
uint8    - input data scaled to 256 level range
uint16   - input data is range checked for 0 .. 65535 range
int16    - input data is range checked for +/- 32767 range
double   - equivalent to double(inData)
```

optional `clip` argument is meaningful only for conversion to `uint8`;
specify 1 to clip rather than scale data to 256 level range (default = 0)

input `double` format data is rounded for non-double output

Examples:

Scale two byte `int16` data to one byte (256 level) data:

```
>> byteData = convert(int16data, 'uint8');
```

Notes:

`convert` is implemented as a compiled MEX file for efficiency.

3.1.2 **DistMap** - apply Raya-Udupa distance transform to binary image

```
usage: dt = DistMap(bw)
```

Given binary (`uint8`) thresholded image `bw` this procedure computes its Raya/Udupa distance transform `dt`: each output pixel receives as its value the shortest distance to the boundary between the 0 and 1 regions of `bw`. The output image is in `uint8` format. Points on the boundary receive output values of 128, points within the boundary receive values 129 .. 254, points outside the boundary receive values 1 .. 127. Distances are clipped to 1 (max external distance) and 254 (max internal distance).

Examples:

Display the distance map of binary image `bw`:

```
>> smartwin(DistMap(bw));
```

Notes:

Implements the algorithm described in Raya & Udupa (1990). This function is supported as a MEX file only. See also `mrivol/distmap`, `ShapeInterp`.

3.1.3 FindOutline - determine perimeter coordinates

usage: `xy = FindOutline(bw)`

Given a (single, connected) perimeter contained in binary (uint8) thresholded image `bw`, returns the set of ordered perimeter coordinates in `xy` (in ULC-based image units).

Examples:

Find the outline of the first-detected contiguous region found within binary image `bw`:

```
>> labelledImg = bwlabel(bw);          % IPT region labelling function
>> xy = FindOutline(labelledImg == 1); % find 1st detected region
```

Notes:

See also `seedfill`.

3.1.4 hist - compute and display smoothed histogram

usage: `[bins, sBins] = hist(img, nBins, cutoff)`

This procedure computes the histogram of intensity image `img` and displays it with a Butterworth lowpass filtered overlay. The zero bin is not displayed. The `nBins` (number of histogram bins) argument defaults to the image intensity range. The `cutoff` argument for low pass smoothing must lie within the range $0 \leq \text{cutoff} < 1$. The default is `.1`; specify `0` to disable smoothing. The function optionally returns bin counts for unsmoothed (`bins`) and smoothed (`sBins`) histograms.

Examples:

Display the 100-bin histogram for `img` with default (`.1`) smoothing:

```
>> hist(img, 100);
```

Find the bin counts for default number of bins and `.3` smoothing:

```
>> [bins, sBins] = hist(img, [], .3);
```

Notes:

This function uses the IPT function `imhist`. See also `mrivol/imhist`.

3.1.5 implot - scaled image plotting

usage: `[fh, ih] = implot(img, mag, map, range)`

This procedure plots `img` within a new figure window the same size as the image. Intensity images [`rows x cols x 1`] are plotted using specified color map (default = `gray`). Three plane RGB images [`rows x cols x 3`] are plotted as truecolor. The optional magnification factor `mag` (default = `1`) performs bicubic interpolation with anti-alias lowpass filtering applied when downsampling. The optional `range` argument is passed to `imagesc` for intensity scaling (default = full range). The function optionally returns handles to the created figure (`fh`) and image (`ih`).

Examples:

Plot the 10th slice of `mrivol vol`:

```
>> implot(vol{10});
```

Plot it at halfsize with colormap `bone`:

```
>> implot(vol{10}, .5, bone(64));
```

Notes:

Resizing is disabled for figures created with `implot`. The Matlab zoom function is enabled by default. See also `smartwin`, `imagesc`, `zoom`.

3.1.6 **irma** - interactive *mrivol* object viewing tool

usage: `irma(vol, map, name)`

Initialize with *mrivol* object `vol`. Optional colormap `map` argument defaults to `gray(256)`. The display name argument defaults to the `vol` variable name, but any string may be specified.

Additional command line options include:

```
>> irma - pops all active IRMA viewers
>> irma abort - closes all active IRMA viewers

>> vol = irma('vol') - returns displayed mrivol object
>> data = irma('get') - returns contents of user procedure data
>> params = irma('show') - returns current user procedure params
>> irma('set', params) - replaces current user procedure parameters
```

When the Slicer window is available the following option supports resetting slicing parameters from a structure:

```
>> irma('slcp', info)
```

where `info` has fields

```
AZ - azimuth (degrees)
EL - elevation (degrees)
OFFSET - [X Y Z] volume offset (mm)
REFPTS - [4 x X, Y] extraction grid landmarks (relative to midsagittal projection)
```

IRMA uses the volume type to establish labelled orientations if possible:

```
1st image ==> last image
right --> left
posterior --> anterior
inferior --> superior
```

The standard IRMA display therefore presents sagittal data viewed from subject right, coronal data viewed from subject front, and transverse data viewed from underneath (looking up). Use `reslice` to resample a volume to other orientations.

Examples:

Display *mrivol* `foo` with colormap `bone` and name `vocal tract`:

```
>> irma(foo, bone(64), 'vocal tract');
```

The same thing with the default (grays) colormap:

```
>> irma(foo, [], 'vocal tract');
```

3.1.7 **iu_*** - irma mouseDown user procedures

usage: `data = iu_*(action, varargin)`

IRMA mouseDown procedures must support at least two action handlers:

```
CONFIG - process any configuration necessary
DOWN - called on mouseDown over image in Display window
```

The CONFIG handler is called when the userProc is installed, or when Reconfigure is selected from the Mouse Down Behavior menu.

Notes:

By convention IRMA user procedures are prefixed by `iu_` (the interactive procedure selection mask matches `iu_*`). Several example procedures exist, with the `iu_template` procedure in particular providing a starting point for user customization.

3.1.8 LoadImg - load MRI data from supported image format files

```
usage: vol = LoadImg(fileMask, fmt, type, mpp, isi, info)
```

This procedure loads image data from an `imread` supported format and returns `mrivol` object `vol`. User is first prompted to select one file from the set matching `fileMask` using an interactive dialog box. `type`, `mpp`, `isi`, and `info` arguments are optional and passed directly to the `mrivol` constructor. Working assumptions: image files are named in slice acquisition order.

Examples:

Load a volume from files matching `IY*.tif`, specifying `type`, `mpp` and `isi` scaling parameters:

```
>> iy = LoadImg('IY*.tif', 'COR', 1.04, 3);
```

Notes:

See also `LoadRAW`, `LoadSMT`, `mrivol`.

3.1.9 LoadRaw - load 16 bit raw format MRI data

```
usage: vol = LoadRaw(fileMask, type, mpp, isi, info)
```

This procedure loads raw format (headerless, uncompressed) image data and returns `mrivol` object `vol`. User is first prompted to select one file from the set matching `fileMask` using an interactive dialog box. `type`, `mpp`, `isi`, and `info` arguments are optional and passed directly to the `mrivol` constructor. Working assumptions: data files are named in slice acquisition order; data are row-wise 256 x 256 16bit pixel format

Examples:

Load a volume from files matching `IY*.dat`, specifying `type`, `mpp` and `isi` scaling parameters:

```
>> iy = LoadRaw('IY*.dat', 'COR', 1.04, 3);
```

Notes:

See also `LoadImg`, `LoadSMT`, `mrivol`.

3.1.10 LoadSMT - load and annotate MRI volume from Shimadzu format file

```
usage: vol = LoadSMT(fileName)
```

This procedure parses a Shimadzu format SMT file and returns the annotated `mrivol` object `vol`. Scaling factors and orientation are determined from header values; relevant imaging parameters are returned as members of the `info` field. `fileName` is assumed to hold the path to a Shimadzu format SMT file. If `fileName` is empty the file may be selected using an interactive dialog; if `fileName` contains the "*" wildcard it is interpreted as a mask for the file selection dialog. Image data are returned in `int16` format.

Examples:

Load a volume interactively, resize and display it:

```
>> irma(resize(LoadSMT, 2), bone(64), 'coronal iy');
```

Load a volume interactively using file selection mask `B*.SMT`:

```
>> vol = LoadSMT('B*.SMT');
```

Notes:

See also `LoadRaw`, `mrivol`, `smt2mat`, `SMTread`.

3.1.11 **match** - pattern matching using convolution

usage: `[ci, xy, v, best] = match(img, subimg, tol)`

Returns `ci`, the cross-correlation of `subimg` with `img` derived by convolution. Use this function to facilitate matching between a template and a base image. `tol` is an optional argument specifying the cutoff for returned values of match peaks as a percentage of the maximum correlation value (default = .9). Locations are in pixel coordinates relative to image ULC and are returned in `xy` [`nVals` × `X, Y`] along with their associated magnitudes `v` [`nVals` × 1]. The index of the best fit (i.e. `xy` location associated with `max(v)`) is returned in `best`.

Examples:

Compute and plot the cross-correlation with default tolerance:

```
>> imshow(match(img, subimg));
```

Find and plot the location of the best fit of `subimg` within `img` at 98% tolerance:

```
>> [ci, xy, v, best] = match(img, subimg, .98);
>> imshow(img); hold on;
>> plot(xy(best,1), xy(best,2), '+');
```

Notes:

Works best with thresholded or semi-thresholded data.

3.1.12 **resize** - resize intensity image

usage: `img = resize(img, scale, method)`

Given a single plane intensity image in double format this procedure returns the result resized by `scale` using the specified interpolation method. `scale` can be any number > 0. Three methods are supported: nearest (nearest neighbor), linear (bilinear interpolation), and cubic (bicubic interpolation). If unspecified method defaults to cubic.

Examples:

Plot image at twice size using bilinear interpolation method:

```
>> smartwin(resize(img, 2, 'linear'));
```

Reduce image size by half with preliminary lowpass anti-alias filtering using default (cubic) method:

```
>> [ih, iw] = size(img); mih = floor(ih*.5); miw = floor(iw*.5);
>> h = fir1(10, mih/ih)' * fir1(10, miw/iw);
>> img = resize(filter2(h, img), .5);
```

Notes:

`resize` is implemented as a compiled MEX file for efficiency. See also `mrivol/resize`.

3.1.13 **rotate** - rotate image

usage: `rotImg = rotate(img, theta, origin, method)`

This function rotates `img` through angle `theta` measured CCW in degrees. The rotation pivot is specified by the `origin` argument in pixel units relative to the image ULC (default is image center). The interpolation method is passed directly to `interp2` (default is '*linear'). If no output argument is specified the rotated image is plotted with superimposed image bounds and origin.

Examples:

Plot image rotated 30° around its center with superimposed reference bounds and bicubic interpolation:

```
>> rotate(img, 30, [], '*cubic');
```

Rotate image -30° around origin [50 50] using default method:

```
>> rotImg = rotate(img, -30, [50 50]);
```

Notes:

See `interp2` for supported interpolation methods.

3.1.14 **seedfill** - find filled region containing seed within threshold limits

```
usage: [img_out,xy] = seedfill(xv,yv,img_in,levels,seed,connect)
```

This function supports thresholded region detection within intensity image `img_in`. `xv,yv` are optional arguments specifying the image coordinate system (by default uses pixel based coordinates with ULC origin). The thresholding boundaries are specified by the `levels` argument. If scalar image pixels with values `<levels` are mapped to 0, pixels `>=levels` are mapped to 1. `levels` may also specify a `[min max]` vector giving the range of pixel intensities to map to 1. The `seed` argument is an `[X Y]` vector giving the starting point for connected region detection; if unspecified may be selected from the plotted image interactively. The `connect` argument specifies neighborhood connectivity for fill operations (default = 8). Returns `img_out`, a binary image corresponding to `img_in` whose non-zero elements give the detected contiguous region. Optionally returns `xy`, the perimeter coordinates of the detected region.

Examples:

Find and plot region between intensity levels [50 100] using an interactively chosen seed:

```
>> [img_out, xy] = seedfill(img_in, [50 100]);
>> imshow(img_out); hold on; % plot binary image
>> plot(xy(:,1), xy(:,2)); % plot perimeter
```

Notes:

Requires the IPT.

3.1.15 **ShapeInterp** - shape based interpolation using Raya-Udupa algorithm

```
usage: bw = ShapeInterp(bw0, bw1, ni)
```

Given binary (uint8) thresholded images `bw1` and `bw2` this procedure interpolates `ni` intermediate images (default = 1) returned in `bw` (`bw0` and `bw1` are the first, last images of the returned array).

Examples:

Interpolate 3 intermediate images between original 2 for a resulting 5 image array:

```
>> bw = ShapeInterp(bw0, bw1, 3); % size(bw,3) == 5
```

Notes:

Implements the algorithm described in Raya & Udupa (1990). This function is supported as a MEX file only. See also `DistMap`, `mrivol/ShapeInterp`.

3.1.16 **smartwin** - enhanced image display window

```
usage: fh = smartwin(img, ...)
```

This procedure creates a new figure displaying the specified monochrome intensity image at its optimal size, and attaches a menu supporting various kinds of subsequent manipulation. Supported initialization `name:value` argument pairs include:

```
'map' - colormap (default gray)
'range' - intensity scaling range ([min max], default is full image range)
'mpp' - mm per pixel (for mm scaling; default is pixel coordinates)
'xlab' - {bottom top} orientation labels (default none)
'ylab' - {left right} orientation labels (default none)
'orient' - orientation (0, 1, 2, 3; default 0); displayed image is rotated CW 90° * orient
'title' - figure title
```

Unless `mpp` is specified, window uses default ULC origin for image coordinates; if `mpp` is given origin is image LLC. `smartwin` optionally returns handle `fh` of the created figure; in this case the figure `'visible'` property is set to `'off'` to support further pre-display initialization. By manipulating the displayed intensity range `smartwin` supports interactive thresholding; to export the image mapped onto the current display intensity range use

```
>> [img, map] = smartwin('EXPORT');
```

This format also optionally returns the colormap in use.

Examples:

Display & export an image using the bone colormap with clipped intensity range:

```
>> smartwin(img, 'map', bone(256), 'range', [50 250]);  
>> clippedImg = smartwin('EXPORT');
```

Display a mm/pixel scaled image:

```
>> smartwin(img, 'mpp', 1.014); % 1.014mm/pixel
```

Display a labelled image:

```
>> smartwin(img, 'xlab', {'bottom' 'top'}, 'ylab', {'left' 'right'});
```

Display a titled image rotated 270° CW:

```
>> smartwin(img, 'title', 'Rotated Image', 'orient', 3);
```

Notes:

See also `imshow`.

3.1.17 **superimpose** - superimpose one image on another interactively

usage: `superimpose(img1, img2, mode, ...)`

Plots the smaller image superimposed on the larger; the superimposed region may then be dragged to a new location using the mouse. Optional mode argument specifies how regions are combined; can be one of

'ADD'	- addition
'SUB'	- subtraction
'COPY'	- displace previous contents
'AND'	- bitwise AND
'OR'	- bitwise OR
'XOR'	- bitwise XOR (default)

The current combination mode can be changed using

```
>> superimpose('MODE', newMode)
```

entered from the command line. Any additional initialization arguments passed directly to `smartwin`.

Examples:

Superimposition with default (XOR) combination mode:

```
>> superimpose(bigImg, littleImg);
```

Change combination mode to AND:

```
>> superimpose('MODE', 'AND')
```

Notes:

See also `smartwin`, `match`.

3.1.18 **threshold** - binary threshold intensity image

usage: `img = threshold(img, levels)`

If `levels` is scalar it is interpreted as a cutoff value (values < cutoff returned as 0); otherwise each row of `levels` specifies a [min max] pair bracketing an intensity region to be mapped to output "1"

Examples:

Threshold image using two selection ranges:

```
>> binImg = threshold(img, [100 200 ; 400 500]);
```

Notes:

`smartwin` can be used to perform this function interactively.

3.2 *mrivol* methods

This section provides descriptions of toolbox functions specialized for *mrivol* objects. Several functions that overload the default behavior of mainstream functions are not documented here if their behavior is transparent. These include support for typing (`uint8`, `int16`, `uint16`, `double`; returns image data as an array in requested format), `size` (returns the pixel dimensions and number of slices), `length` (number of slices); `DistMap`, `resize`, `ShapeInterp`, and `threshold` (return results applied to all volume images). Use `help mrivol/<function>` to see help for overloaded functions.

3.2.1 **area** - area function estimation

```
usage: A = area(vol, sections)
```

Computes and displays area function for binary thresholded cross-sectional *vol* for specified sections (default is all sections). Returns area *A* in units of volume.

Examples:

Display area function for sections 10 – 50:

```
>> area(vol, [10:50]);
```

Notes:

Requires the IPT. See also `threshold`.

3.2.2 **FindOutline** - determine perimeter coordinates for thresholded *mrivol*

```
usage: ot1 = FindOutline(m)
```

Given a binary thresholded *mrivol* object *m* returns the set of perimeter coordinates associated with each section as a cell array (assumes one object per section). Each cell contains an array with dimensions `[nCoords x X, Y, Z]` where *X, Y* are mm coordinates relative to the slice LRC and *Z* is the mm offset into volume.

Notes:

Requires the IPT. See also `threshold` and the mainstream `FindOutline`.

3.2.3 **imcrop** - selects a subvolume cropped from input volume images

Interactive version:

```
usage: [m_out, cropRect] = imcrop(m_in, imgNum);
```

Displays image `imgNum` from *mrivol* `m_in` (default = 1), then awaits mouse-driven selection of the cropping rectangle. Optionally returns the cropping rectangle used. Non-interactive version:

```
m_out = imcrop(m_in, cropRect);
```

where `cropRect` specifies the cropping rectangle bounds in pixels.

Notes:

Requires the IPT.

3.2.4 **imhist** - builds and displays volume histogram

usage: `pv = imhist(m, full)`

Identifies (from smoothed histogram) bimodal peaks and the trough between them, returned as `pv` [`lower_peak` `trough` `upper_peak`]. By default the lowest (0) bin is ignored; specify non-zero `full` to override this behavior and plot all.

Notes:

Requires the IPT.

3.2.5 **imwrite** - saves volume images as separate files

usage: `imwrite(m, scale, fname, fmt, ...)`

Creates files for each `mrivol` `m` image plane named `fname_n.fmt`, where `n = 1 : length(m)`, using `scale` to map data to an 8 bit intensity range. The only required argument is `m`; if unspecified the filename is selected by interactive dialog, `fmt` defaults to `tiff` (no compression), and `scale` defaults to `m.RANGE`. Additional arguments passed to mainstream `imwrite`.

Examples:

Save volume images as jpeg files overriding default scaling (filename selected interactively):

```
>> imwrite(vol, [50 1000], [], 'jpeg'); % empty filename arg
```

Notes:

Complements `LoadImg`.

3.2.6 **montage** - display volume slices as a single image

usage: `montage(m)`

Passes the reshaped contents of `m`. `DATA` to the IPT `montage` function.

Notes:

See also the mainstream IPT `montage`.

3.2.7 **movie** - creates a Matlab movie from the volume image slices

usage: `movie(m, ...)`

Creates (and optionally returns) a Matlab format movie from `mrivol` object `m` image data, then 'plays' it. The remaining arguments are passed to the built-in `movie` function.

Examples:

Create volume slice movie; play it 3 times at 2 fps:

```
>> movie(vol, 3, 2);
```

Notes:

See also `irma`.

3.2.8 **mrivol** - construct `mrivol` object

usage: `m = mrivol(data, type, mpp, isi, info)`

where

`data` - volume image data [height x width x depth]

optional data descriptors are

`type` - SAG | COR | TRN | OBL

`mpp` - mm/pixel mapping factor

`isi` - interslice interval (mm)

`info` - arbitrary annotation field

Examples:

Create an `mrivol` object from a coronal image array:

```
>> info = struct('SUBJECT', 'MT', 'UTTERANCE', 'AE');  
>> m = mrivol(corData, 'COR', 1, 3, info);
```

Notes:

Image data given in IPT image deck form (height x width x 1 x nImages) is flattened automatically by `mrivol`. See also the various `Load*` functions.

3.2.9 `reslice` - resample `mrivol` object

This function supports various ways of resampling `mrivol` data. All require that `data`, `mpp`, and `isi` be specified for the input volume, and reorientations require that the input type be one of {SAG | COR | TRN}. In the first form optimized manipulations are selected by keyword:

```
usage1: m_out = reslice(m_in, option, isi, method, crop)
```

where `option` is one of

```
'SAG'      - convert to sagittally oriented volume  
'COR'      - convert to coronally oriented volume  
'TRN'      - convert to transverse oriented volume  
'RESAMPLE' - resample using new isi
```

`m_in` is the input `mrivol` object; `isi` is the output inter-slice-interval (mm, default = 1); `method` is the resampling method (default = cubic); `crop` is an optional argument that if non-zero causes 0-valued rows and columns to be removed from the resulting slices (default = 1).

The second form supports generalized translation and rotation:

```
usage2: m_out = reslice(m_in, az, el, origin, isi, method, crop, type)
```

where `az` is the reorientation azimuth (degrees, default = 0); `el` is the reorientation elevation (degrees, default = 0), `origin` is the reorientation pivot (mm, default = `m_in` volume center); `type` is the output volume type (default = 'OBL'); remaining arguments as above. Binary thresholded data are interpolated using the Raya-Udupa distance transform (see `DistMap`).

Examples:

Convert an `mrivol` to a sagittally oriented dataset using 3mm `isi` with default method and cropping:

```
>> sagVol = reslice(vol, 'SAG', 3);
```

Resample volume correcting for subject head tilt (default `isi`, trilinear interpolation, no cropping):

```
>> corVol = reslice(vol, -5, 2, [110 100 50], [], 'linear', 0, 'COR');
```

Notes:

See also `slicer`.

3.2.10 `seedfill` - threshold volume starting from seed voxel

```
m_out = seedfill(m_in, levels, seed)
```

Given `seed` location [X, Y, Z] in volume mm coordinates returns connected `levels` thresholded output volume in `uint8` (logical) form.

Examples:

Assuming `seed` is a voxel within `levels`, find thresholded voxels connected through volume:

```
>> filledVol = seedfill(vol, [50 320], [110 100 20]);
```

Notes:

Requires the IPT.

3.2.11 slicer - extract oblique slice from `mrivol` object

This function supports two ways to specify an oblique resampling plane. In the first form the slice bounds are determined from an offset point within the volume, and azimuth and elevation angles:

```
usage1: [img, isect] = slicer(m, az, el, offset, method, crop)
```

Azimuth `az` is the counterclockwise angle in the `xy` plane measured from the positive `x` axis in degrees. Elevation `el` is the angle from the `xy` plane. `offset [x y z]` defines a point in `mrivol` `m` mm coordinates, where `x` increases from left to right, `y` increases from bottom to top, and `z` increases from first image to last. The origin is the lower left corner of the first image of the volume. The default `offset` is the center of the volume. `method` is an optional argument specifying the interpolation method; one of {'nearest' | 'linear' | 'cubic' (default)}. `crop` is an optional argument that if non-zero causes 0-valued rows and columns to be removed from the generated slice (default = 1).

The resulting oblique slice intersects the volume along the plane through `offset` oriented at the azimuth and elevation angles (relative to the volume). Output image size is determined by the norm of the dimensions of `m`.

In the second form the oblique slice bounds are specified explicitly:

```
usage2: img = slicer(m, bounds, method, crop)
```

`bounds` defines the initial and terminal edges of the oblique slice in the form of XYZ point pairs (0-based volume mm coordinates, [4 x 3]), where the point ordering determines the volume sweep direction. `bounds` can be exterior to the volume, but must be rectangular to avoid distortion.

Returns the oblique image slice (double format); pixels on the extracted image external to the volume are returned as 0. Binary thresholded data are interpolated using the Raya-Udupa distance transform (see `DistMap`). Optionally returns initial, terminal intersection coordinates `isect` of oblique slice.

Examples:

Extract the midsagittal projection through a coronally oriented volume (default center volume offset):

```
>> sagImg = slicer(corVol, 90, 90); % tricubic, cropping
```

Extract a coronal projection through a transverse oriented volume (default interpolation, no cropping):

```
>> [corImg, isect] = slicer(trnVol, 5, 88, [105 98 15], [], 0);
```

Same thing using intersection bounds returned above:

```
>> corImg = slicer(corVol, isect, 'cubic', 0);
```

Notes:

See also `irma`, `reslice`.

References

- Arata, L. (1995) "Tri-cubic Interpolation," in *Graphics Gems V* (A. Paeth, ed.), New York: Academic Press, pp. 107-110.
- Heinz, J.M., and Stevens, K.N. (1964) "Derivation of area functions and acoustic spectra from cineradiographic films of speech," *Quarterly Progress Report*, Research Laboratory of Electronics, MIT, **74**, pp. 192-198.
- Raya, S.P., and Udupa, J.K. (1990) "Shape-based interpolation of multidimensional objects," *IEEE Transactions on Medical Imaging*, **9**, pp. 32-42.
- Tiede, M. K. (1999) "An MRI-based morphological approach to vocal tract area function estimation," Ph.D. dissertation, Linguistics, Yale University.
- Tiede, M.K., and Yehia, H. (1996) "A shape-based approach to vocal tract area function estimation," *Proc. ASA-ASJ 3rd Joint Meeting*, pp. 861-866.