A Model for Arbitrary Plane Imaging, or the Brain in Pain Falls Mainly on the Plane

Jeff Miller
Dylan Helliwell
Thaddeus Ladd
Harvey Mudd College
1250 N. Dartmouth Ave.
Claremont, CA 91711
{ jmiller, dhelliwe, tladd } @math.hmc.edu

Advisor: Michael Moody

Summary

We present an algorithm for imaging arbitrary oblique slices of a three-dimensional density function, based on a rectilinear array of uniformly sampled MRI data.

We

- develop a linear interpolation scheme to determine densities of points in the image plane,
- incorporate a discrete convolution filter to compensate for unwanted blurring caused by the interpolation, and
- provide an edge-detecting component based on finite differencing.

The resulting algorithm is sufficiently fast for use on personal computers and allows control of parameters by the user.

We exhibit the results of testing the algorithm on simulated MRI scans of a typical human brain and on contrived data structures designed to test the limitations of the model. Filtering distortions and inaccurate modeling due to interpolation appear in certain extreme scenarios. Nonetheless, we find that our algorithm is suitable for use in real-world medical imaging.

The UMAP Journal 19 (3) (1998) 223–236.

Constructing the Model

Our model consists of four main parts:

- First, we develop a technique for positioning a plane anywhere in $\mathbb{R}^3$.
- Then we interpolate data from the region in $\mathbb{R}^3$ that contains data onto the plane.
- Next, we use a sharpening technique to remove extra blur caused by the interpolation.
- Finally, we construct a difference array and use it to create a line drawing representing edges in the image.

Assumptions

- **Density variations in the source object are reasonably well behaved and continuous.** Discontinuities such as sharp edges will be approximated in the model but only if they are isolated on a scale of several array elements. Similarly, erratic behavior and wild fluctuations can be accurately modeled only if they exist on a scale of several pixels. The model should image the source of the data array, not the array itself; but the accuracy of the oblique slice images depends on the accuracy of the data in the array.
- **The data array represents isotropically spaced samples.** The array $A(i, j, k)$ contains discretized samples from a continuous three-dimensional space, for which we use coordinates (x, y, z). The component $A(i, j, k)$ represents a density f at some point (x_i, y_j, z_k). We assume that the source was uniformly sampled, so that

$$x_i = i\delta x, \qquad y_j = j\delta y, \qquad z_k = k\delta z,$$

  where δx, δy, and δz are constant distances between samples (typically close to 1 mm). We also assume that these distances are all equal (if not, then we rescale the coordinate system to compensate).
- **The destination computing platform is a typical contemporary personal computer.** Thus, input data arrays may not be larger than the memory of a typical PC. We assume arrays of up to $256 \times 256 \times 256$ integer elements (16 MB in size) each in $[0, 255]$. Also, we gauge computing time by typical PC processor speeds.

Plane-Array Intersection

To represent the slice of the object on the computer screen, we establish a mapping between the three-space of the object and the plane of the monitor.

We represent an arbitrary plane in $\mathbb{R}^3$ in terms of the angles that it makes with the xy-plane and the z-axis, together with a displacement of the origin. The map $T : \mathbb{R}^2 \to \mathbb{R}^3$ given by

$$T(u,v) = R \begin{pmatrix} u \\ v \\ 0 \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix}$$

transforms a point in $\mathbb{R}^2$ into a point on the plane, where the point (x_0, y_0, z_0) is a displacement of the origin and R is the rotation matrix

$$R = \begin{pmatrix} \cos\phi\cos\theta & -\sin\theta & \sin\phi\cos\theta \\ \cos\phi\sin\theta & \cos\theta & \sin\phi\sin\theta \\ -\sin\phi & 0 & \cos\phi \end{pmatrix}.$$

The angles ϕ and θ are the polar and azimuthal angles in spherical coordinates for the vector normal to the plane.

Interpolation

To represent the image, we seek a regularly spaced array of discretized points (u_p, v_q) corresponding to the pixels of a computer monitor. Since the points $T(u_p, v_q)$ need not coincide with the points (x_i, y_j, z_k) for the data, we need to be able to approximate density values at arbitrary points in $\mathbb{R}^3$. Thus, we interpolate the data from nearby points whose values are given by A.

With a slight abuse of notation, let $g(x, y, z)$ be the gray-scale value of the image at (x, y, z), so that $g\big(T(u,v)\big) = g(u,v)$ and $g(x_i, y_j, z_k) = A(i,j,k)$.

From the numerous techniques for interpolation, we seek an algorithm that will smoothly approximate the density without being computationally intractable.

Nearest-Neighbor Approach

Let (x^*, y^*, z^*) be the point for which we want to know the density. This point is contained in a cubic cell, of size $\delta x \times \delta y \times \delta z$, that has corners of known density given by the array A. From these eight corners, we simply find the point (x_a, y_b, z_c) that is closest to (x^*, y^*, z^*) and set $g(x^*, y^*, z^*) = A(a,b,c)$.

3-D Linear Interpolation

We also develop a technique that we call *3-D linear interpolation*. For this method, we hope to find a smooth continuation of the data within the cubic cell, starting from the density values of the corners of the cubic cell containing (x^*, y^*, z^*). We base our approach on solving the Laplace equation

$$\Delta g = \frac{\partial^2 g}{\partial x_1^2} + \cdots + \frac{\partial^2 g}{\partial x_n^2} = 0$$

successively in one, two, and three dimensions.

We choose two adjacent corners of the cell and solve the one-dimensional Laplace equation, using the densities at the corners as boundary values. This gives the smoothest function between the two corners—a straight line. We then solve the two-dimensional Laplace equation on the faces of the cubic cell, using the straight lines as boundary conditions. Finally, we fill the cube with the three-dimensional solution to the Laplace equation, using as boundary conditions the values on the faces.

The details are not difficult. Denote the points in the cubic cell as in **Figure 1**. The values along edges are constructed by simple linear interpolation; for example, the values along the lower left edge of the cube in **Figure 1** are given by

$$\begin{aligned} g(x^*, y_j, z_k) &= g(x_i, y_j, z_k) + \frac{g(x_{i+1}, y_j, z_k) - g(x_i, y_j, z_k)}{x_{i+1} - x_i} \, (x^* - x_i) \\ &= A(i, j, k) + \frac{A(i+1, j, k) - A(i, j, k)}{x_{i+1} - x_i} \, (x^* - x_i). \end{aligned}$$

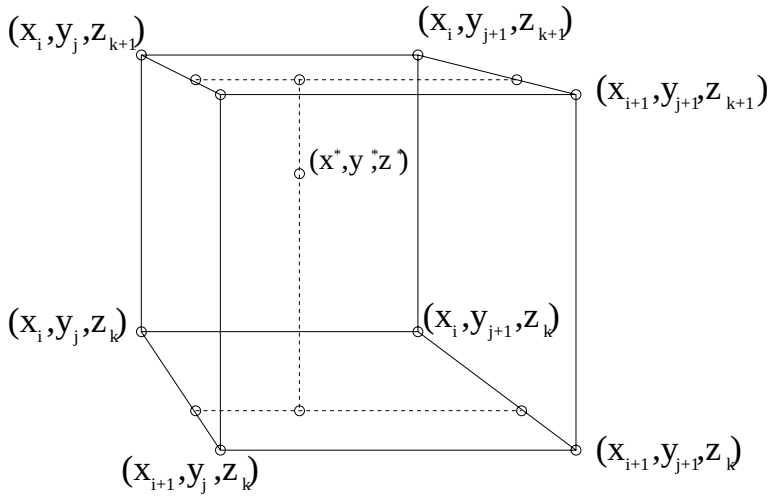

Figure 1. A cubic cell demonstrating the notation for the 3-D interpolation scheme.

Similarly, we find

- values $g(x^*, y_{j+1}, z_k)$ along the lower right edge in terms of $A(i, j+1, k)$ and $A(i+1, j+1, k)$,
- values $g(x^*, y_j, z_{k+1})$ along the upper left edge in terms of $A(i, j, k+1)$ and $A(i+1, j, k+1)$, and
- values $g(x^*, y_{j+1}, z_{k+1})$ along the upper right edge in terms of $A(i, j+1, k+1)$ and $A(i+1, j+1, k+1)$.

We continue to use linear interpolation to get the value $g(x^*, y^*, z_k)$ on the bottom face in terms of the value $g(x^*, y_j, z_k)$ on the lower left edge and the

value $g(x^*, y_{j+1}, z_k)$ on the lower right edge, as well as the value $g(x^*, y^*, z_{k+1})$ on the upper face in terms of the value $g(x^*, y_j, z_{k+1})$ on the upper left edge and the value $g(x^*, y_{j+1}, z_{k+1})$ on the upper right edge.

As a last step, we use linear interpolation yet again to obtain the value $g(x^*, y^*, z^*)$ in terms of the value $g(x^*, y^*, z_k)$ on the lower face and the value $g(x^*, y^*, z_{k+1})$ on the upper face. The result is a unique value for $g(x^*, y^*, z^*)$, in terms of the eight closest corners, which does not depend on the order of interpolation. [EDITOR'S NOTE: We omit the authors' proof of this fact, which they arrive at by explicit calculation of $g(x^*, y^*, z^*)$ and the observation that the result is symmetric in x, y, and z.] In addition,

$$\frac{\partial^2 g}{\partial x^2}(x^*, y^*, z^*) = \frac{\partial^2 g}{\partial y^2}(x^*, y^*, z^*) = \frac{\partial^2 g}{\partial z^2}(x^*, y^*, z^*) = 0,$$

which means that Laplace equation is satisfied in the cubic cell. The uniqueness theorem for the Laplace equation with Dirichlet boundary conditions implies that only one solution may be obtained by this method.

Other Techniques

We considered a three-dimensional spline, in which a cubic interpolation is chosen to make first derivatives continuous. Unfortunately, the complexity and the computing time of this technique made it intractable. We also looked at spatially weighted averaging techniques. However, in applying this method to a simple two-dimensional case, we found gross discrepancies with the true image (a simple linear ramp was altered to look like a series of wavy steps).

Ultimately, we found:

- The nearest-neighbor technique is the most useful for cursory image analysis, and
- 3-D linear interpolation is the most efficient method for a more realistic image.

Image Sharpening

Interpolation inevitably blurs, or *low-pass filters*, the actual image f. Hence, we add a stage to the algorithm that sharpens, or *high-pass filters*, the recorded image g. We considered various techniques for sharpening.

Revert from Boundaries

One approach to sharpen is to detect the location of edges or boundaries in the image and then revert to a nearest-neighbor pixel determination near those locations. We discovered that this approach has the adverse affect of increasing graininess and pixelation.

Point-Spread Function

Another approach, following Andrews and Hunt [1977], is to assume that the recorded image is the actual image convolved with a *point-spread function* (PSF), denoted $h(x, y)$. Thus,

$$g(u,v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x-\xi, y-\eta) f(\xi,\eta)\, d\xi d\eta. \qquad \textbf{(1)}$$

The actual image is then obtained by deconvolution with a discrete Fourier transform. The PSF may be calculated a priori or measured a posteriori.

Alternatively, the discretized nature of the data and the linear nature of the interpolation procedure lead us to recast **(1)** into the matrix equation

$$g(u_p, v_q) = \sum_{m=1}^{N} \sum_{n=1}^{N} a(p,m) b(q,n) f(u_m, v_n),$$

where $a(p, m)$ is an $N \times N$ matrix that blurs the columns of the digitized plane image and $b(q, n)$ is an $N \times N$ matrix that blurs the rows. The "blurring" matrices may be approximated with components near unity on the leading diagonals and components equal to some small "mixing" parameter on the adjacent off-diagonals. The image f may then be restored by inverting these matrices.

Ultimately, we deemed this and the Fourier PSF approach to be too computationally expensive.

Convolution Filter

Our favored technique, following Rosenfeld and Kak [1982], is to use a convolution filter. This approach rests on the assumption that the blurring occurred as a diffusion process. If the actual image f is an initial condition to the diffusion equation

$$\kappa \nabla^2 g = \frac{\partial g}{\partial t},$$

then by expanding a time-dependent $g(u, v; t)$ about a small value τ of time we obtain

$$\begin{aligned} f(u,v) = g(u,v;0) &= g(u,v;\tau) - \tau \frac{dg}{dt}(u,v;\tau) + O\left(\tau^2\right) \\ &= g - \kappa\tau\nabla^2 g + O\left(\tau^2\right). \end{aligned}$$

Thus, f may be restored by subtracting the Laplacian of g from g. This technique, commonly called *unsharp masking*, is especially appealing in our model; since we chose an interpolation scheme that forces interpolated regions of the image g to satisfy Laplace's equation, $\nabla^2 g = 0$.

In practice, the Laplacian is approximated using finite differences. Define

$$\Delta_u g(u_p, v_q) = g(u_p, v_q) - g(u_{p-1}, v_q),$$
$$\Delta_v g(u_p, v_q) = g(u_p, v_q) - g(u_p, v_{q-1}).$$

Higher-order difference operators are defined by repeated first differencing, as in

$$\Delta_u^2 g(u_p, v_q) = \Delta_u g(u_{p+1}, v_q) - \Delta_u g(u_p, v_q),$$

leading to

$$\begin{aligned}\nabla^2 g &= \Delta_u^2 g(u_p, v_q) + \Delta_v^2 g(u_p, v_q) \qquad \textbf{(2)}\\ &= g(u_{p+1}, v_q) + g(u_{p-1}, v_q) + g(u_p, v_{q+1}) + g(u_p, v_{q-1}) - 4g(u_p, v_q).\end{aligned}$$

Applying the Laplacian operator to an entire matrix may be viewed as a discrete analog of convolution. That is, we can find the Laplacian of a component $g(u_p, u_q)$ of the image matrix by multiplying component-wise each value in the 3×3 neighborhood around the component with the "mask" matrix

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

and then summing all of the components of the resulting 3×3 matrix.

In light of **(2)**, we wish to convolve with the mask

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} - \alpha \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -\alpha & 0 \\ -\alpha & 1+4\alpha & -\alpha \\ 0 & -\alpha & 0 \end{pmatrix},$$

which subtracts the Laplacian times a control parameter α (analogous to $\kappa\tau$) from the function itself.

Natural extensions of this technique are to use higher-order approximations of the Laplacian operator (thus convolving with a larger mask) or to enhance the mask with some sort of high-pass filter. Considerations of time and computational complexity prevented our development of such techniques.

Edge Detection

To make boundaries more visible in our images, it is useful to detect edges and generate corresponding line drawings. To do this, we use a variation of the finite differences method already discussed. At each point (x_i, y_j, z_k), we evaluate

$$\begin{aligned}\Delta_{2x}(i,j,k) &= A(i-1,j,k) - A(i+1,j,k)\\ \Delta_{2y}(i,j,k) &= A(i,j-1,k) - A(i,j+1,k)\\ \Delta_{2z}(i,j,k) &= A(i,j,k-1) - A(i,j,k+1).\end{aligned}$$

This set of definitions centers the difference around the point in question.

We construct a new array Γ, where

$$\Gamma(i,j,k) = \max\{\Delta_{2x}(i,j,k),\ \Delta_{2y}(i,j,k),\ \Delta_{2z}(i,j,k)\},$$

which gives a measure of how fast the values of A are changing through a given point.

This technique has many strong points:

- The values for Γ are easy to compute.
- The method does not bias the type of edge encountered (straight, curved, diagonal, etc.).
- The values of Γ remain within the range of the original grayscale range [Rosenfeld and Kak 1982].

We then apply our interpolation techniques to a plane passing though the box of data given by Γ rather than by A. Once we have this new two-dimensional image, we convert it into a binary image by applying a threshold condition: Every point with an interpolated value above the threshold value is made black, while every point with an interpolated value below the threshold value is made white. This converts regions where the difference values are high into black regions against a white background. Since edges will exhibit large differences in density, the black regions will thus represent edges.

Analysis of the Model

We implemented our plane-imaging algorithm on a Unix graphics workstation and analyzed the algorithm's behavior on several different data sets. One data set is a simulated MRI scan of a human brain, an example of a data volume that the model would expect to receive in a real-world medical imaging environment. We created several other contrived data sets to test our algorithms on known structures and thereby expose limitations of the model.

Computer Implementation

We built our model as an interactive graphical application using the C++ language and the OpenGL 3-D graphics library. We designed this program to possess many of the features that a plane-imaging system used in an actual medical situation would contain. It takes as input an arbitrarily-sized 3-D block of byte values (0–255) representing the density array A. The program presents two display windows to the user:

- The first window is a view of the (x, y, z) coordinate space, showing wire-frame representations of both the input data array and the projection plane.

- The second window shows the scanned image that lies on the plane as generated by our plane-imaging algorithm.

The user can use the keyboard and mouse to move the imaging plane to different positions (x_0, y_0, z_0) and angles (ϕ, θ) within A, viewing in real time how the projected image changes. We used this program to generate all of the figures in this paper.

The coded algorithms for interpolating in A and for creating the sharpened image in the projection plane are all straightforward translations of the mathematical expositions given earlier. We imparted some extra intelligence to the imaging algorithm so that it can determine in which part of the (u, v) plane the source data lie (see **Appendix**).

The user can control all parameters of the model, including the sharpening control factor and the edge-detection threshold. Different interpolation techniques may be selected, and the sharpening and edge-detecting filters may be toggled as well. The program executes quickly enough; but since we paid little attention to creating optimized algorithms, there is much much potential for speeding up operations such as the sharpening filter.

Results on Brain MRI Data

The principal test data set for our model is a simulated human brain MRI volume containing 181 slices, each 181×217 pixels. These data are the output of a highly realistic MRI computer simulation [Cocosko et al. 1997] and are thus likely to reflect data from an actual MRI scanner that would be of diagnostic interest to a user.

Exploration of Obliquely Oriented Structures

Figures 2–5 show sample output of our plane-imaging algorithm on the brain MRI data set, for several different plane orientations, both orthogonal and oblique. Viewing such output dynamically on a graphical computer display would allow a doctor to explore structures in the brain that lie on planes in any possible position and orientation.

Fine Structure

The brain MRI data set demonstrates the merits of applying the sharpening filter. **Figure 6** displays two brain images; the one on the left has been sharpened and the one on the right has not. The sharpened image lacks the overall blurriness of the unsharpened image. It also displays more clearly the fine structure in the brain that would be of interest to a surgeon planning a minimally invasive procedure.

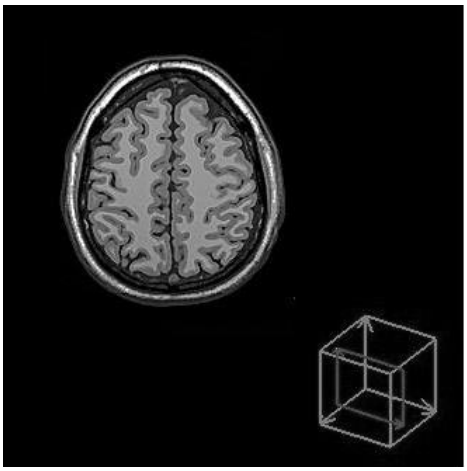

Figure 2. Image centered at $(91, 108, 120)$ with $(\phi, \theta) = (0, 0)$. The imaging plane is orthogonal to the z-axis.

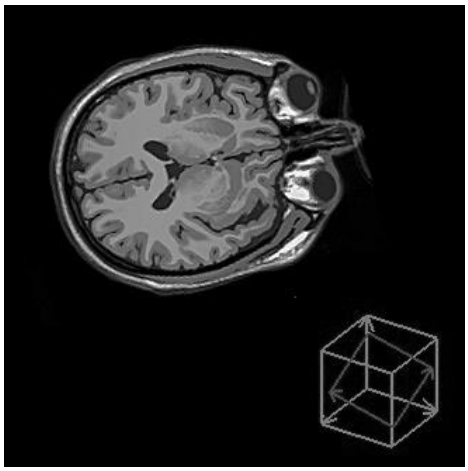

Figure 3. Image centered at $(91, 108, 120)$ with $(\phi, \theta) = (35, 75)$. The imaging plane lies obliquely within the data volume.

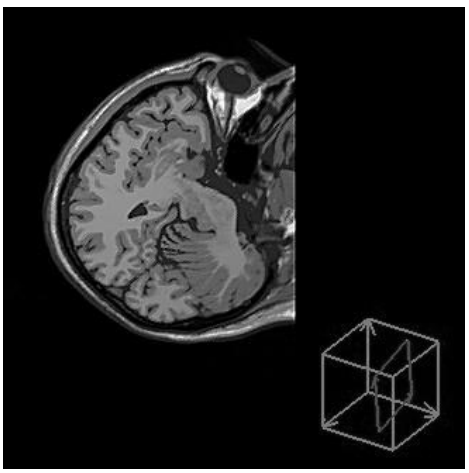

Figure 4. Image centered at $(110, 100, 70)$ with $(\phi, \theta) = (130, -30)$.

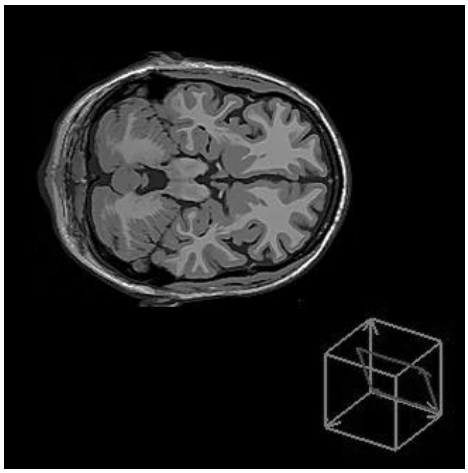

Figure 5. Image centered at $(100, 100, 48)$ with $(\phi, \theta) = (-20, 90)$.

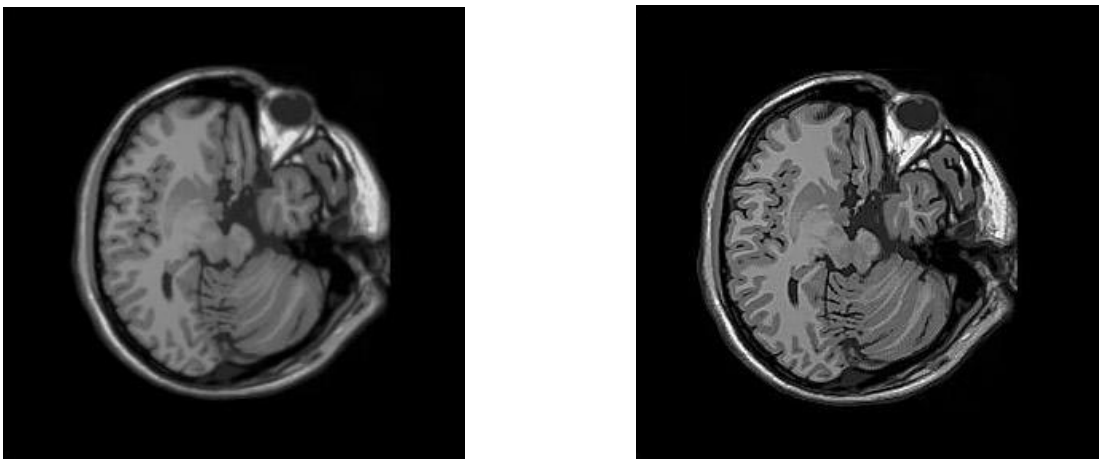

Figure 6. Comparison of two oblique plane scans of the brain MRI data. The image on the right has been passed through the sharpening filter, while the image on the left has not. The imaging plane is centered at $(85, 118, 58)$ with $(\phi, \theta) = (30, 0)$.

Line Drawings

The edge-detection algorithm that we described can be used to generate black-and-white line drawings of the kind found in an anatomical atlas (see **Figure 7**). Such drawings are useful for seeing clear boundaries of structures in the brain.

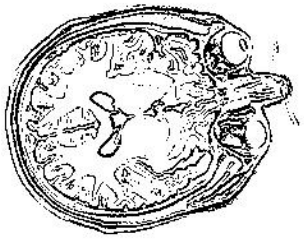

Figure 7. Black-and-white line drawing of an oblique plane through the brain MRI data. Lines were drawn using edge detection. The imaging plane is the same as in **Figure 3**.

Results on Contrived Data Sets

A question that arises about our model is whether or not the sharpening filter is removing the blurriness introduced by the interpolation algorithm or just the blurriness inherent in the data array. To attempt to answer this question, we examined the operation of our model on a data set with perfectly discrete boundaries—a data set consisting of "slabs," that is, parallel, evenly spaced planes of some nominal depth containing maximum intensity pixels. **Figure 8** shows the close-up results of passing an imaging plane through this volume at an angle of 35° to the slabs (an arbitrary choice). Comparing the clarity of the

slab edges in the two different images demonstrates that our sharpening filter performs well in removing effects of interpolation, provided that boundaries cross the image plane at a sufficiently large angle. When the imaging plane is at a small angle to the slabs, as in **Figure 9**, we see blurred edges even in a sharpened image—an inevitable consequence of our 3-D interpolation scheme.

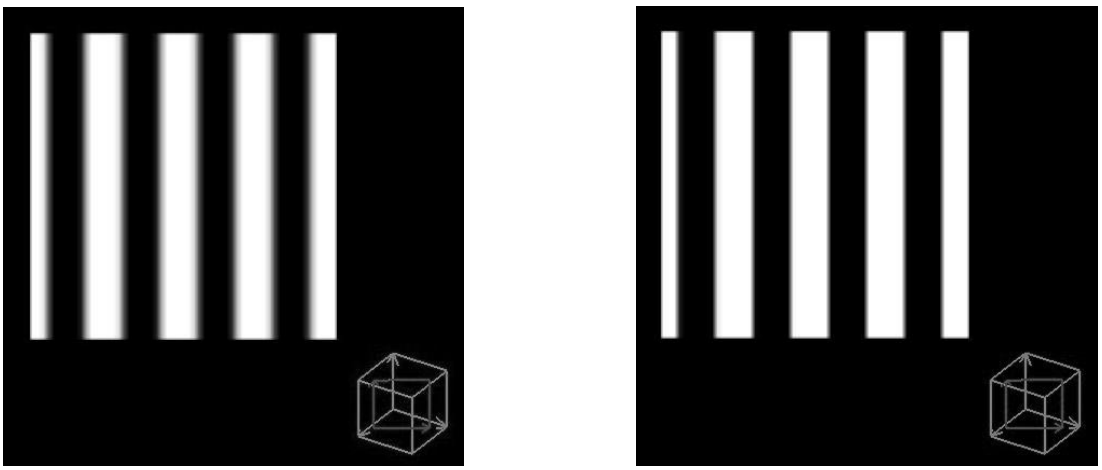

Figure 8. Comparison of two oblique plane scans of the "slabs" data. The angle of incidence between the imaging plane and the slab visible is 35°. The image on the right has been passed through the sharpening filter, while the image on the left has not; resulting images are close-up views.

The values in the data set can be viewed as the discrete extreme of behaviors that our model can expect to encounter. To examine the opposite continuous extreme, we created a data set whose intensity does not vary in a single xy-plane but instead varies as a cubic function of z. **Figure 10** shows the projected image from a plane passing through the data volume at angle of 35° to the xy-plane. The interpolation algorithm in our model linearizes much of the nonlinearly varying data, and subsequently the sharpening filter introduces distortion.

Limitations of the Model

Our model has several limitations:

- Arrays of non-uniformly or anisotropically sampled data are not considered.
- Objects with planar edges parallel or nearly parallel to the projection plane are imaged inaccurately.
- Interpolation and smoothing generate minor distortions that, given more time and work, may be alleviated by the use of higher order schemes. Using additional contrived data structures as algorithm input may further illuminate the causes of such distortions.

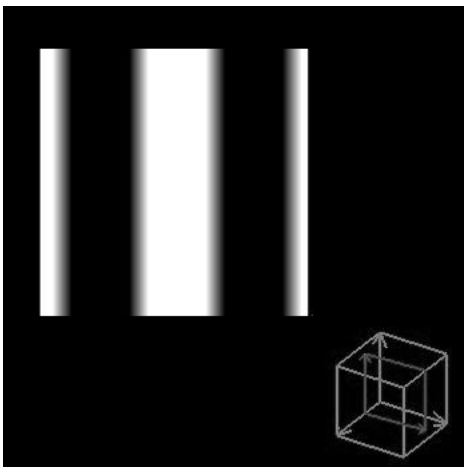

Figure 9. A nearly parallel oblique plane scan of the slabs data (angle of incidence is 5°) with sharpening filter applied. Edge ramping effects are visible.

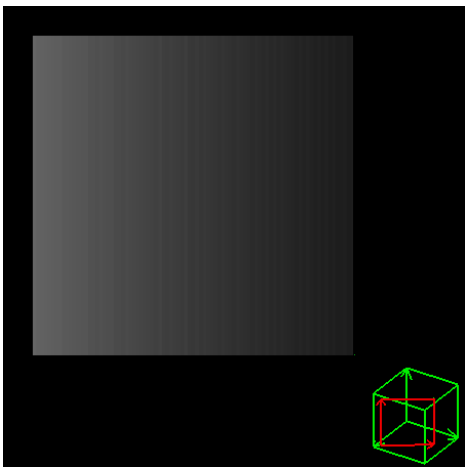

Figure 10. Oblique plane scan of a continuously varying data volume. Some linearization effects of the sharpening filter are visible.

Conclusions

Our computer implementation provides convincing pictures that illustrate the ability of our model to depict density variations in oblique planes. Our algorithm surpasses most existing algorithms by allowing any planar orientation, by generating images quickly, and by including sharpening and edge-detecting filters. The results of testing our model on simulated brain MRI images demonstrates its applicability to real-world medical imaging.

Appendix: Bounds of the Imaging Plane

For computational optimization of our plane-imaging algorithm, we derive bounds for the region of intersection of the plane and the box of data. This allows our algorithm to iterate over the smallest (u, v) region necessary to ensure that all of the intersected data have been obtained.

The transformation $T(u, v)$ gives the three equations

$$\begin{array}{rcrcrcl} x & = & \cos\phi\cos\theta\, u & - & \sin\theta\, v & + & x_0 \\ y & = & \cos\phi\sin\theta\, u & + & \cos\theta\, v & + & y_0 \\ z & = & -\sin\phi\, u & & & + & z_0. \end{array}$$

We can rewrite these to get

$$\begin{array}{rcrcr} x - x_0 & = & \cos\phi\cos\theta\, u & - & \sin\theta\, v \\ y - y_0 & = & \cos\phi\sin\theta\, u & + & \cos\theta\, v \\ z - z_0 & = & -\sin\phi\, u & & . \end{array}$$

We want to know when the plane crosses an edge of the box of data in $\mathbb{R}^3$. This

edge will be constant in two of the three variables (x, y, z). We can plug these two values into the appropriate equations above and solve for u and v.

For instance, suppose we want to know at what point (u, v) the plane intersects one of the edges of the data box that is parallel to the z-axis. In this case, we know what x and y are since we know the size and position of the box in $\mathbb{R}^3$. Using the equations for $x - x_0$ and $y - y_0$, we have:

$$\begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix} = \begin{pmatrix} \cos\phi\cos\theta & -\sin\theta \\ \cos\phi\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}.$$

Notice that if $\phi = \pi n + \frac{\pi}{2}n$, where $n \in \mathbb{Z}$, this transformation does not have an inverse. In this case, the plane is parallel to the z-axis, and hence any vertical edge of the data box. If $\phi \neq \pi n + \frac{\pi}{2}$, then we can invert the above transformation to obtain

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{\cos\phi} \begin{pmatrix} \cos\theta & \sin\theta \\ -\cos\phi\sin\theta & \cos\phi\cos\theta \end{pmatrix} \begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix}.$$

We can perform similar operations for the other two directions in $\mathbb{R}^3$ with similar restrictions on ϕ and θ. We thus obtain 12 points in the uv-plane, or fewer if the plane is parallel to one of the axes. With these 12 values, we simply choose the largest and smallest values for u and v to get a rectangle that tightly bounds the intersection of the plane and the data box.

References

Andrews, H.C., and B.R. Hunt. 1977. *Digital Image Restoration.* Englewood Cliffs, NJ: Prentice-Hall.

Cocosco, Chris A., Vasken Kollokian, Remi K.-S. Kwan, and Alan C. Evans. 1997. BrainWeb: Simulated Brain Database. `http://www.bic.mni.mcgill.ca/brainweb/` .

Rosenfeld, Azriel, and Avinash C. Kak. 1982. *Digital Picture Processing.* 2 vols. San Diego, CA: Academic Press.

Russ, John C. 1995. *The Image Processing Handbook.* Boca Raton, FL: CRC Press.