A Tricubic Interpolation Algorithm for MRI Image Cross Sections

Paul Cantrell
Nick Weininger
Tamás Németh-Csőri
Macalester College
St. Paul, MN 55105

Advisor: Karla V. Ballman

Introduction

We designed and implemented a program capable of:

- taking in a large three-dimensional array of one-byte grayscale voxels (volume "pixels"), the output from an MRI machine;
- slicing through that array along an arbitrary plane;
- and using interpolation to produce an image of the cross section described by the plane.

We allow the user to select the plane of cross section by specifying three points that should be in the plane, or by specifying one point and two angles. We account for the possibility of voxels of unequal size in different dimensions, but presume they are evenly spaced in each dimension. We then use a tricubic interpolation algorithm to produce a cross-sectional image. This method is our extension of bicubic interpolation, an algorithm used widely with two-dimensional images. We chose the tricubic method because it offers an optimal balance of accuracy and computational speed. Finally, we allow the user to "stain," or color, important portions of the data.

We tested the program on simple geometric figures to verify its correctness. We then tested it on actual MRI image slices of four brains, with very satisfactory results. We found that important image features were preserved well and that image staining was useful in visualization. The interpolation algorithm runs in linear time; it produces an image from a $256 \times 256 \times 256$ data volume in a few seconds.

The UMAP Journal 19 (3) (1998) 237–253.

Finally, we constructed several data sets that point out the limitations of our algorithm and of the problem itself. These limitations concern the behavior of our algorithm in areas of maximal uncertainty, farthest from the sample points.

Design Considerations

Typical Uses of MRI Images

As described in Rodriguez [1995], MRI scans of various parts of the body are used to diagnose a wide range of disorders. One of the most common uses is the detection of abnormal bodies in the brain, such as tumors, cysts, and hematomas. Because of variations in appearance of both healthy tissues and tumors, it is critical that the sharpness or fuzziness of boundaries, as well as the general shape and brightness of regions, should be preserved when taking cross sections.

An analysis program should rely on intuitive spatial understanding and also provide a straightforward way for a user to specify a volume for highlighting in subsequent cross sections.

Characteristics of Image Data

Data Size

The usual data size for one MRI slice is 256×256 grayscale pixels. In order to get data covering an entire 3-D object, multiple slices are required. The time taken for each scanning slice is dependent on a parameter to the scanning process called *repetition time*; a typical slice might take several minutes to scan, though multiple slices may sometimes be scanned simultaneously [Hornak 1997]. Since the amount of time that patients can spend immobile in the machine is limited, the number of slices that can be taken is small compared to the slice resolution. The database for our real-world test data [Johnson and Becker 1997] typically took 25–60 slices to scan an entire brain.

This means that the actual volume of space represented by each voxel is likely not to be a cube. Instead, it will be a rectangular prism, significantly longer in one dimension than in the other two; the algorithm will need to take this fact into account so as not to produce distorted output. Furthermore, if a voxel is much longer along one axis than along the others, much more interpolation in that dimension will be required, so images taken in planes parallel to that axis may be especially inaccurate.

Data Artifacts

Many different types of artifacts may be present in MRI image data; some are results of incorrect operation or configuration of the machine, while others

are products of the physical properties of the scanning process [Ballinger 1997; Hornak 1997].

Since most of these types of artifacts reflect problems with the machine's configuration that may produce misleading images, it is important that they be preserved in cross section, so that the MRI operator can see them and recalibrate the machine appropriately.

Sampling Characteristics

The manifestation of all of these image characteristics in data is fundamentally tied to the properties of discrete sampling. We can classify data in which discrete sample points describe a continuous function (as our data do) as either *undersampled*, *oversampled*, or *critically sampled* (see **Figure 1**), depending on how the sampling resolution corresponds to the actual detail in the image.

a. Undersampled. **b.** Oversampled. **c.** Critically sampled.

Figure 1. Typical data sampling characteristics for images.

- Oversampled data: The sample grid is finer than the image detail. Such images tend to look very blurry, and neighboring grid points tend to vary only slightly and contain essentially redundant information. This high level of detail lends these images to accurate interpolation and enhancement.
- Undersampled data: The image contains detail finer than the sample grid and there is little correlation between neighboring pixels, especially at the edges of objects in the image. If the actual sample area for each pixel is smaller than the sample area that the pixel represents, the image may be characterized by jagged edges and sharp contrasts. Such images make interpolation and enhancement a matter of heuristics and guesswork.
- Critically sampled data lies at the border of undersampling and oversampling, and MRI data fall into this category. As with oversampled data, the edges of boundaries tend to be unaliased (smooth), and the image may even appear slightly blurry; however, as with undersampled data, the detail at the pixel level is important, and interpolation possibilities are limited.

Interpolation Algorithms

Our input data come as a set of image values taken at discrete points, but the cross sections that we want to take may not pass exactly through any of these points. Therefore, we need a way to estimate image values at arbitrary points based on the image values at the sample points. That is, based on our array of samples $A_{i,j,k}$, we want an interpolating function $f : \ ^3 \rightarrow \ $ such that

$$f(i,j,k) = A_{i,j,k}$$

when i, j, and k are integers, and such that f takes on reasonable values for nonintegral i, j, k. (This stipulation that the interpolating function match the sample points is reasonable, as MRI images tend to be very clean and have a high signal-to-noise ratio.)

In choosing an interpolating function, we had to make a trade-off between accuracy of image production and running time, limited by the typically critically sampled nature of MRI data. We chose a cubic method, which we found to be surprisingly fast and quite accurate for actual MRI data.

Tricubic Interpolation

Cubic interpolation is a special case of Lagrange interpolation, which is a simple method of finding the unique polynomial of degree $(n-1)$ that passes through n data points [Mnuk 1997].

We consider first the one-dimensional case. Cubic interpolation begins with the four sample points closest to the target point x—its two nearest neighbors on either side, $\lfloor x \rfloor$, $\lfloor x \rfloor - 1$, $\lceil x \rceil$, and $\lceil x \rceil + 1$—and fits a cubic function $p : \ \rightarrow$ to them; $p(x)$ gives the interpolated value at x (see **Figure 2**). Note that the particular f described by these four points around x gives the values only for the region between the middle two. Thus, the function f that interpolates the whole image is a piecewise composite of many different cubic functions.

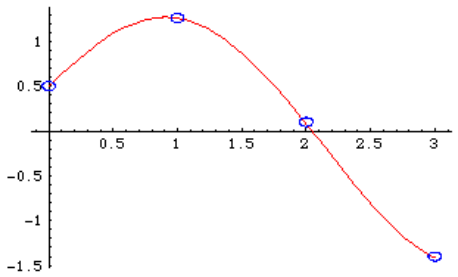


Figure 2. One-dimensional cubic interpolation.

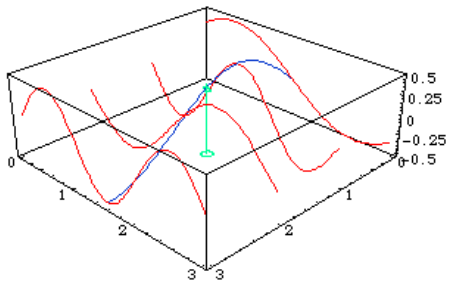


Figure 3. Two-dimensional cubic interpolation.

This procedure generalizes nicely to multiple dimensions. It does not require, as one might expect, the construction of an elaborate multivariate poly-

nomial or the solution of a large system of equations; in fact, it is sufficient to perform the interpolation in each dimension consecutively.

It is perhaps easiest to visualize this process in two dimensions first [Makivic 1996]. As shown in **Figure 3**, we separate the 16 points surrounding the target point into four lines of four points each. We do a one-dimensional cubic interpolation along each of these lines and evaluate the resulting cubics at points along a perpendicular line containing the target point. We then use *these* four evaluated points to interpolate another cubic that we can evaluate at the target point.

We can then extend this to three dimensions in the obvious way: Split the 64 points into four planes of 16 points each. In each of these planes, perform the two-dimensional process to get four interpolation points along a line through the target point. Finally, perform an interpolation to get a function value for our target point. This requires a total of 21 one-dimensional interpolations, five for each plane plus the final one. The process is illustrated in **Figure 4**. It runs in time linear in the total number of voxels in the volume.

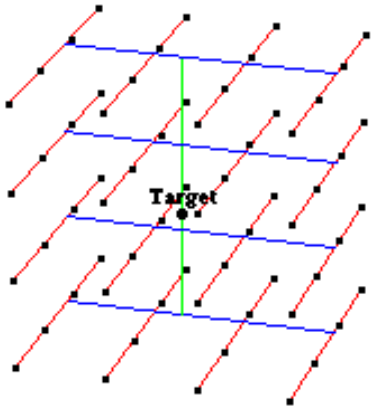


Figure 4. Schematic of three-dimensional cubic interpolation.

One key question arises: Does it matter how we choose the planes, and the lines within each plane? It turns out that the final value obtained at the target point is independent of the order in which the dimensions are chosen for the interpolation. [EDITOR'S NOTE: We omit the authors' proof.]

Cubic interpolation is particularly appropriate to critically sampled data. It relies on some correlation and continuity between neighboring points, producing smooth curves that fit slightly smoothed object edges very well without introducing artificial detail not present in the original image. However, it does not oversmooth or mangle detail past the single-pixel level, and it introduces minimal artifacts. Although cubic interpolation performs poorly on undersampled or jagged data, it is appropriate for MRI data.

Furthermore, because it involves only simple arithmetic and is linear in the size of the data set, bicubic interpolation is fast enough to produce typical MRI images in close to real time without a high-end workstation. The results of a two-dimensional bicubic enlargement are in **Figure 5a**.

a. Bicubic. **b.** Nearest-neighbor. **c.** Bilinear.

Figure 5. Enlargements by various interpolation algorithms.

Alternative Interpolation Methods

We considered and rejected several alternative interpolation methods.

Nearest-Neighbor Interpolation

In nearest-neighbor interpolation, the image value at an arbitrary point is the value of the nearest sample point. This is a very fast algorithm—it requires only a single rounding operator for each dimension.

Nearest-neighbor interpolation is often appropriate to undersampled data because it preserves the jaggedness of such data and does not attempt presume that image fades smoothly across the sharp edges. It would probably be the most appropriate method for cross sections of black-and-white line art medical diagrams.

However, for these same reasons, it performs very poorly on critically sampled data, and its rounding can actually locally distort image proportions where a smoother interpolation would preserve them (**Figure 5b**).

Linear Interpolation

Linear interpolation takes the image value at a point to be the average of the image values of all its known neighbors, weighted by how far away they are. This is also a very simple and fast algorithm. However, it is little more than a blurring of the nearest-neighbor method and shares many of its problems. In particular, it leaves object edges jagged and uneven, even when they are not aliased, and tends to blur excessively (**Figure 5c**).

Convolution-Based Methods

There is a wide variety of much more intricate interpolation methods based on the convolution of the image matrix, including Fourier-based methods, CMB interpolation, and Wiener enhancement.

Although they perform extraordinarily well, for several reasons they are inappropriate to the task at hand. These methods are primarily targeted at oversampled data and tend to act more as de-blurring algorithms than interpolators. Furthermore, since they require taking the convolution, they are not linear in the data set size and tend to be quite slow (Mahan [1996] describes a run of dozens of hours to enhance a small image of Saturn).

Since MRI data tend not to be particularly blurry, and since enlargement is not our goal, these computationally expensive algorithms are simply overkill. Furthermore, especially in critically sampled data, they are likely to produce artifacts with visually striking large-scale structure, which could be misleading to a reader of the image and lead to a misdiagnosis.

Image Enhancement

Either during or after interpolation, we have the option of enhancing the image to sharpen blurred regions, enhance edges, or otherwise bring out details. However, we found that the tricubic method performed well enough that most of these methods were either inappropriate or unnecessary. The human eye is extremely adept at interpolation of obscured detail, and tricubic interpolation tends to capitalize on this by producing blurry but suggestive output in regions of uncertainty. The enhancement algorithms that we examined revealed no details that the eye could not already interpolate. Given the dangers in introducing artificial detail in medical imaging, we decided to leave our tricubic method unenhanced.

Anti-Aliasing

For sharp or undersampled data, it can be beneficial to blur high-contrast edges slightly. However, this is counterproductive in our case; the detail in our images is significant, but the edges of objects are not generally aliased.

Sharpening

Traditional sharpening algorithms work by moving a pixel's value away from the average of its neighbors, perhaps weighted so that the sharpening will be localized to the edges of objects.

A major problem with this sort of sharpening is that it can produce jagged edges and exaggerate the effects of noise in data (**Figure 6**). Since tricubic interpolation can produce blurry output in heavily interpolated regions, such sharpening could be of use. However, we found that it made little visible difference in cross sections of real data and revealed no significant new detail.

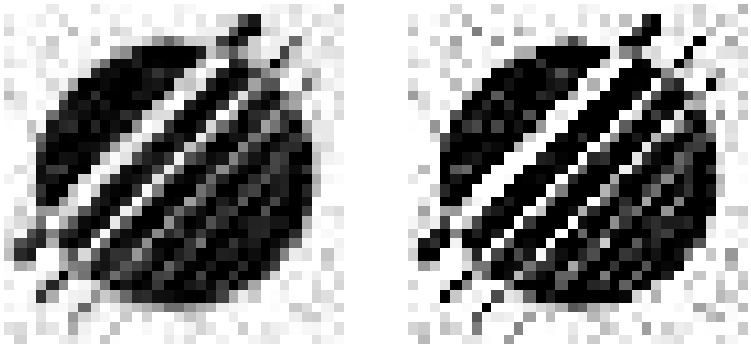


Figure 6. Oversharpening increases noise and aliases edges.

Edge Fitting

Many algorithms work to enhance jagged or blurry edges by finding the edges in an image (the areas where pixels differ from the average of their neighbors), fitting curves to them, and enhancing them, either by anti-aliasing or selective sharpening. Such algorithms, however, are usually used as an aesthetic enhancement and not in situations that call for scientific accuracy; although their output can be very pleasing to the eye, they are guilty in the extreme of introducing artificial detail. Thus, although they might produce visually pleasing results in our problem, they should be used very judiciously if at all.

Image Staining

Even with very clear interpolations and a graphical aid to help visualize the orientation of various cross sections, it can be difficult for a human to mentally compose cross sections into a solid and to identify the same object in different cuts. To assist in this visualization process, it would be nice to allow the user to "stain" portions of the data with a color. A physician could mark a reference point or an anomalous object and be sure of its location in different cross sections. Such a feature should work so as to make a colored area visible even in cross sections that do not exactly intersect the region that the user originally marked. Thus, a marked region should be "feathered" or blurred outward slightly from the plane in which the user places it.

Implementing Our Algorithm

Specifying the Plane of Cross Section

We allow the user to specify the plane for a cross section by either:

- selecting three noncollinear points in the plane of the cross section, a good method for selecting an initial arbitrary cut based on image features; or
- selecting an arbitrary point to be included and specifying two angles, known as *Euler angles*. The first angle is the angle between the xy-plane and the plane of cross section; the second is the angle between the x-axis and the intersection of the cross-sectional plane with the xy-plane. This is a good method for continuously traversing the image or for fine-tuning the orientation of a particular cut.

To calculate the cross section, we need to transform the input data into a triplet $(\vec{p}, \hat{x}, \hat{y})$, where $\vec{p}$ is an arbitrary point on the plane and $(\hat{x}, \hat{y})$ forms an orthonormal basis for the plane.

Three-Point Representation

To obtain $(\vec{p}, \hat{x}, \hat{y})$ from the three-point representation $(\vec{p_1}, \vec{p_2}, \vec{p_3})$, we take the cross product of the two vectors $\vec{p_2} - \vec{p_1}$ and $\vec{p_3} - \vec{p_1}$ to produce the normal vector $\vec{n}$, which is perpendicular to the plane. Then we solve the system of equations

$$\vec{n} \cdot \hat{x} = \vec{n} \cdot \hat{y} = 0, \qquad \hat{x} \cdot \hat{x} = \hat{y} \cdot \hat{y} = 1, \qquad \hat{x} \cdot \hat{y} = 0, \qquad \hat{x}_z = 0$$

for $\hat{x}$ and $\hat{y}$. The first two equations ensure that the basis vectors are in the plane; the next two ensure they are of unit magnitude; the fifth makes them perpendicular. These five equations in six unknowns do not specify a unique basis, so we need one more constraint. We choose $\hat{x}_z = 0$ as that last constraint because it simplifies the resulting formulas greatly. Finally, we let $\vec{p} = \vec{p_1}$.

Point Plus Euler Angles

For input of the form $(\vec{p}, \phi, \theta)$, we think of the plane as a rotation of the xy-plane, with the origin set to $\vec{p}$. We first rotate the plane by ϕ around the x-axis, and then rotate it by θ around the z-axis. The resulting transformations are given by

$$\hat{x} = R_\theta R_\phi \hat{i}, \qquad \hat{y} = R_\theta R_\phi \hat{j},$$

where $\hat{i}$, $\hat{j}$ are the standard basis for the xy plane and

$$R_\phi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{pmatrix}, \qquad R_\theta = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

The vectors that we obtain from these methods are not always the best ones for our purposes. We would like the display orientation to correspond to the user's concept of the volume; that is, up should remain up and left should remain left whenever possible. Therefore, we try to align the basis vectors as

closely as possible with the xy-plane's basis. To do this, we rotate the basis vectors in the plane so as to maximize $\hat{x} \cdot \vec{i}$, thus bringing the $\hat{x}$ vector as close as possible to the true x-axis. We then reverse the direction of $\hat{y}$ (effectively flipping the image over) if that reversal increases the value of $\hat{y} \cdot \vec{j}$.

Once we have our adjusted basis, we calculate where, if anywhere, the plane intersects each of the 12 edges of the data volume. These edge intersection points define the boundary of the cross section of the volume. We define the data volume to be the parallelepiped with corners at $(0, 0, 0)$ and $(x_{\max}, y_{\max}, z_{\max})$. Each edge has two fixed coordinates; thus, we can compute each edge intersection by solving two equations in two unknowns. For example, to compute the point of intersection with the edge running along the z-axis, we solve the equation system

$$p_x + c_1\hat{x}_x + c_2\hat{y}_x = 0, \qquad p_y + c_1\hat{x}_y + c_2\hat{y}_y = 0$$

(where p_x is the x-component of $\vec{p}$, $\hat{x}_x$ is the x-component of $\hat{x}$, and so on) for c_1 and c_2. This system will always have a unique solution unless the plane is parallel to the z-axis. If that happens, either the plane does not intersect the z-axis, or else the z-axis lies in the plane; in the latter case, we take the two endpoints of the edge at 0 and $z_{\max}$ to be the intersection points.

If we have unique values for c_1 and c_2, we then solve for the z-coordinate of the intersection: $z_{\text{intercept}} = p_z + c_1\hat{x}_z + c_2\hat{y}_z$. If $z_{\text{intercept}} \in [0, z_{\max}]$, then the plane does indeed intersect this edge of the data volume. Otherwise, it intersects the line defined by the edge at a point outside the volume.

When we have all the edge intersection points, we can define a rectangle bounding the cross section in terms of the basis vectors: Just take the maximum and minimum values of c_1 and c_2 over all the points. Finally, we calculate the upper left corner of the bounding rectangle and proceed to the interpolation.

Performing the Interpolation

At this phase in the computation, we scale the individual components of the $\vec{p}$, $\hat{x}$, and $\hat{y}$ to account for the possibility of voxels with different sizes in different dimensions, which could result from MRI slices taken far apart. In other words, if the actual size of a voxel is $(a\ b\ c)$, we scale from the basis

$$(1\ 0\ 0) \qquad (0\ 1\ 0) \qquad (0\ 0\ 1)$$

(which reflects geometric reality) to the basis

$$(a\ 0\ 0) \qquad (0\ b\ 0) \qquad (0\ 0\ c)$$

(which is appropriate for our data array). This method presumes that the size of a voxel in each dimension is constant and thus that MRI slices are spaced evenly.

The cross-section sample points are now described by $\vec{p} + a\hat{x} + b\hat{y}$, where a and b are integers. From this, we can easily construct a double loop to traverse the cross section.

At each of these cross-section sample points, we perform a tricubic interpolation. Since doing so involves two neighbor points in each direction, we define the value of a sample point outside the data array to be a uniform dark gray, so that we can interpolate for points near the edge.

The assumption that the data slices are equally spaced allows a number of simplifications in the Lagrange polynomials and thus in the code to perform the interpolation. Allowing for uneven voxel size within a dimension (as might result from an uneven series of slices) would require substantial extension of this portion of the program.

Performing Image Staining

[EDITOR'S NOTE: We omit the authors' description of implementation of the staining feature.]

Testing the Algorithm

Correctness Testing

As a simple test that the algorithm was working properly, we used it to take sections at various angles through two different geometric objects (see **Figure 7**):

- a cube, filled with smaller cubes alternating black and white in a checkerboard pattern; and
- a torus, filled according to a variable grayscale gradient, with three perpendicular cylinders of different diameters, filled with white, intersecting at the center of the torus.

We chose these objects because their correct cross sections at any given angle are readily identifiable. The algorithm did indeed take correct cross sections of these objects at a variety of angles.

Real-World Testing

To provide test data reflecting conditions actually encountered in diagnosis, we downloaded four series of axial (xy-plane) MRI slices from the Whole Brain Atlas [Johnson and Becker 1997], a database of information on brain anatomy and pathology. We converted these slices into four three-dimensional arrays of test data using Adobe Photoshop.

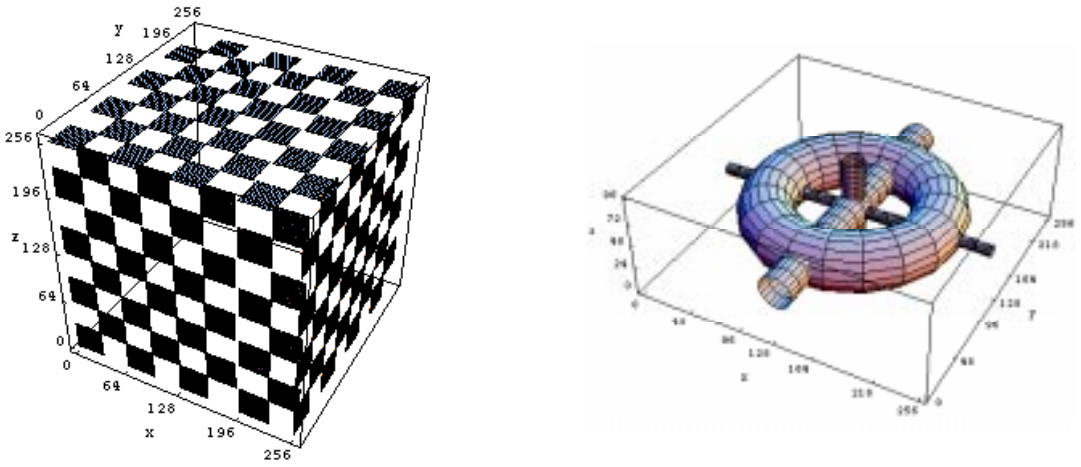


Figure 7. Geometric test objects.

- Data Set 1 was from a normal, healthy brain;
- Data Set 2 was from a brain containing a type of tumor known as a glioblastoma;
- Data Set 3 was from a brain affected by cerebral hemorrhaging; and
- Data Set 4 was from the brain of a woman with advanced Alzheimer's disease.

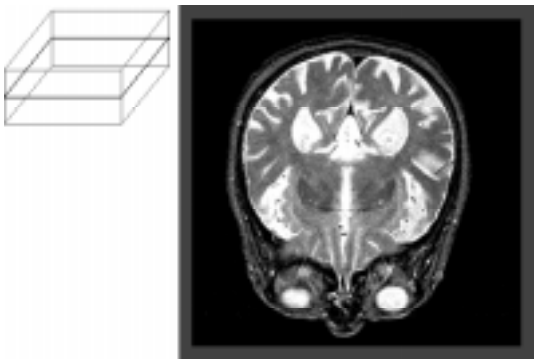
Examples of the resulting cross-sectional images are shown in **Figure 8**. We found that the algorithm worked better in perpendicular planes than in oblique planes, as expected. However, almost all the images were quite sharp and clear, preserving object boundaries and shapes excellently. Note in particular the series of oblique cross sections of Data Set 2, showing the shape and boundaries of the glioblastoma with great clarity.

When our image data contained artifacts, these too were preserved; they occurred most notably in the images from Data Set 3, which also produced by far the lowest-quality images. The primary reason for this is that it contained only 24 slices, as against 54 for Data Set 1, 56 for Data Set 2, and 45 for Data Set 4. Thus, the pixels in this data set were much more "stretched" in the z-direction, forcing the algorithm to do more vertical interpolation in taking cross sections.

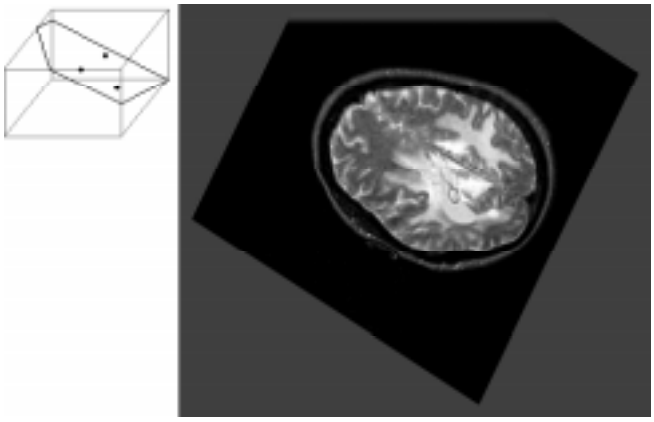
Finally, we found that staining worked well in highlighting important image features in different cross sections. [EDITOR'S NOTE: We must omit the authors' two-color figures, which strikingly highlight the hemorrhage in Data Set 3.]

Problems in Our Algorithm

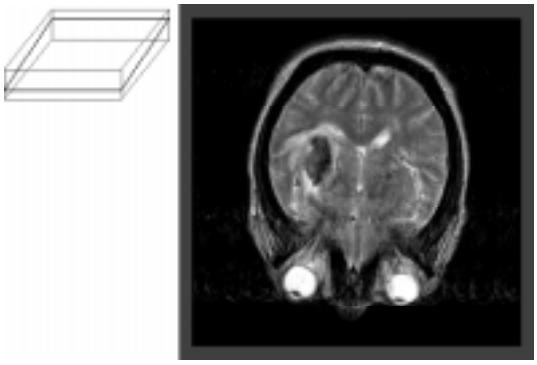
Our interpolation produces a slight increase in blurriness or fuzziness of edges characteristic of most image interpolation methods. Different cuts of the



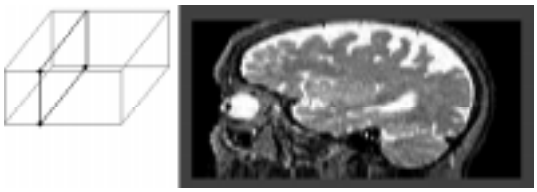
Slice from Data Set 1, the brain of a healthy elderly woman.



Slice from Data Set 2, the brain of a man with glioblastoma. The large bright mass at lower right is the tumor.



Slice from Data Set 3, the brain of a man with acute cerebral hemorrhage (the dark mass on the right side of the brain). Note the image artifacts; this is our lowest-quality data set.



Slice from Data Set 4, the brain of a woman in the advanced stages of Alzheimer's disease. Note the enlarged lateral ventricles (bright structures in the center of the brain) and unusually large, bright convolutions at the top of the image.

Figure 8. Sample slices from the four data sets.

data can produce widely varying results; compare (see **Figure 9**):

A. a cut along a horizontal plane, in which the points in the cut correspond to actual data points and we have maximal clarity (**Figure 9a**);

B. a cut along a horizontal plane halfway between two actual slices, in which points on the slice are blurred between the neighboring layers (**Figure 9b**);

C. a vertical cut, in which the image is blurry in the z dimension, where the voxels are very tall (**Figure 9c**); and

D. a maximally oblique cut, in we are cutting across the long diagonal of a voxel (**Figure 9d**; note the jaggedness along the edge of the skull).

We do have some control over this blurring. Examples A and B are essentially the same image, but the former is slightly clearer; a smarter algorithm could take this into account. However, these are special cases; in general, we cannot avoid moving through areas that require a high degree of interpolation.

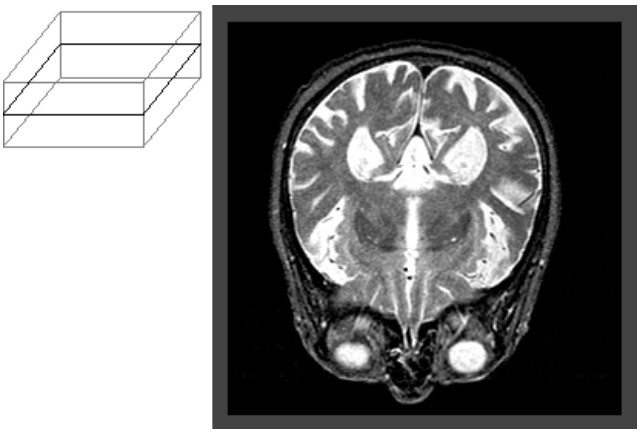
We can illustrate this problem with the extreme case of a three-dimensional checkerboard of $1 \times 1 \times 1$-voxel black and white squares. Tricubic interpolation gives an even 50% gray at points midway between data points; thus, a cross section of this object shows high-contrast checkering in areas with a low degree of interpolation, and gray in areas that are heavily interpolated. Note, for example, the effect of shifting a horizontal plane up half a pixel just as we did in examples A and B above (**Figure 10a**). Because neighboring pixels differ so much, the blurring effect is much more pronounced.

A nearly horizontal but slightly oblique plane passes closer and farther from data points, producing an interference pattern (**Figure 10b**). The gray areas in this image correspond to points where sharp edges would blur slightly in real sample data. We could move these interference patterns around by translating the plane of cross section and rotating our basis vectors. In certain circumstances, this can actually decrease the overall blurriness, as in examples A and B. However, we cannot eliminate the interference pattern without compromising the integrity of the interpolation.

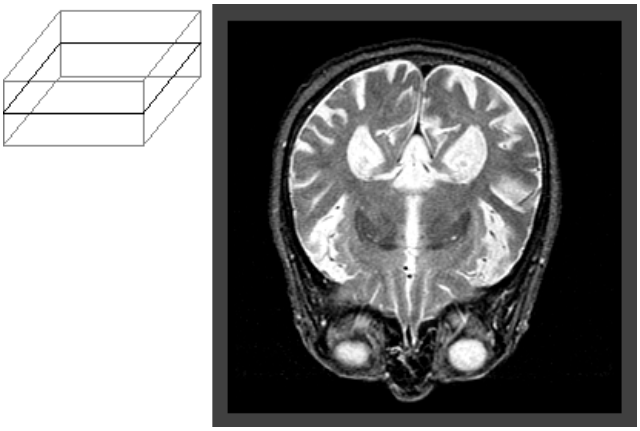
It might seem that we could perturb the sample plane slightly, bending it toward data points (i.e., avoiding the gray areas). However, as the perturbation increases, this algorithm degrades to nearest-neighbor, which distorts proportions locally and essentially defeats the purpose of interpolation. We experimented with restricting this perturbation to a direction normal to the plane; however, we found that unadorned bicubic interpolation worked best.

It is important to realize that the checkerboard is a very poor model of actual data—it is nothing but very high-contrast noise, which is not at all typical of MRI data. Its usefulness lies in illustrating the fundamental problem of discrete sampling: We simply cannot avoid approximating the values for a significant portion of an arbitrary cut.

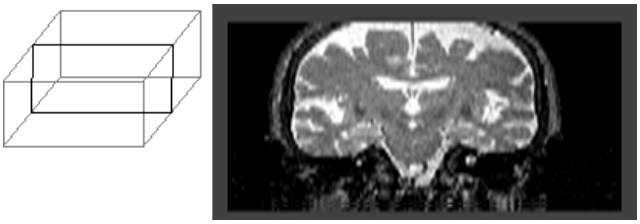
The blurring that tricubic interpolation produces, however, does not mangle detail beyond the one-voxel level. Even a $2 \times 2 \times 2$-voxel checkerboard shows its



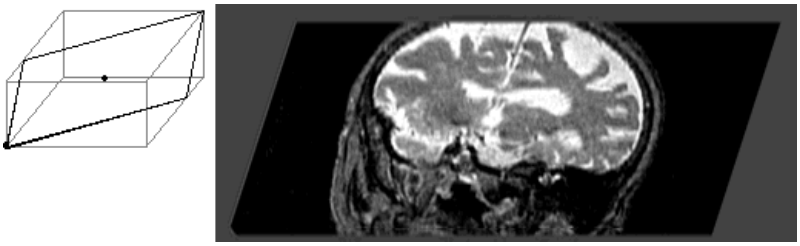
a. A cut along a horizontal plane; points in the cut correspond to actual data points and we have maximal clarity.



b. A cut along a horizontal plane halfway between two actual slices; points are blurred between the neighboring layers.

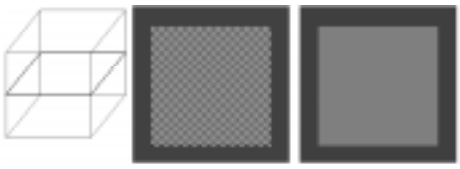


c. A vertical cut; the image is blurry in the z dimension, where the voxels are very tall.

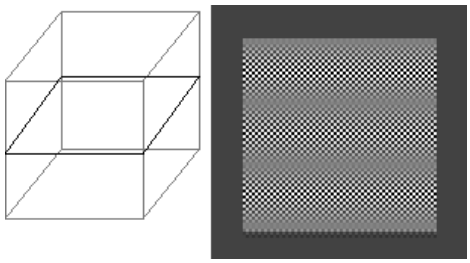


d. A maximally oblique cut, across the long diagonal of a voxel; note the jaggedness along the edge of the skull.

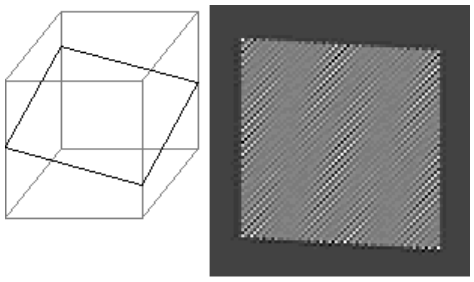
Figure 9. Slices along various planes through Data Set 1.



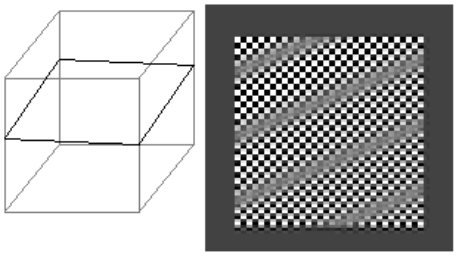
a. A cut along a data plane (center) and a cut halfway between data planes (right).



b. A cut along a nearly horizontal but slightly oblique plane produces interference bands.



c. A cut along a grossly oblique plane produces two families of interference bands.



d. A cut along an oblique plane through a checkerboard with $2 \times 2 \times 2$ voxels.

Figure 10. Various cross sections of 3-D checkerboards.

checkered pattern more clearly than the $1 \times 1 \times 1$ at oblique angles (**Figures 10cd**), and real data are even better behaved.

One other problematic situation is machine calibration. Suppose that the user has scanned a solid cube to align the machine and now wishes to know the exact angle at which that cube is oriented in the data array. The user could use our algorithm to align a cut with the top of the cube. However, the precision of the image degrades when the angle between the cross section and the cube is very small, especially if the cube is only slightly offset from the axes of the data array. The pixellated top of the cube produces a mild interference pattern, and the user would have to re-scan once or twice to align the scanner past sample resolution. In this case, an edge-fitting enhancement algorithm would be entirely appropriate.

Conclusion

Our algorithm's strengths working with real MRI data are:

- robust and intuitive specification of the cross-section plane,
- preservation of large-scale image features,
- preservation of fine detail present in the source image,

- preservation of image proportions given knowledge of the voxel dimensions,
- preservation of features of diagnostic interest,
- conceptually useful coloring of three-dimensional features in the image, and
- real-time performance sufficient to allow interactive exploration of the data.

There is, of course, a good deal of room for improvement. Our current implementation is not as fast as it should be, nor as easy to use. There are circumstances where it would be useful to offer several interpolation options (e.g., nearest-neighbor for cross sections of line drawings), or to perform edge-fitting and sharpening on the interpolated image (e.g., for the calibration situation described above). It would also be nice to extend the algorithm to deal with unequally spaced slices; this would require a more general (and significantly slower) implementation of cubic interpolation. However, our algorithm serves its principal purpose very well, giving good results on a variety of real data.

Acknowledgment

The authors wish to Alexa Pragman for her help in preparing the figures for publication.

References

Ballinger, Ray. 1997. Gainesville VAMC MRI Teaching File. `http://www.xray.ufl.edu/~rball/teach/mriteach.html` .

_____. 1998. The MRI Tutor. `http://128.227.164.224/mritutor/index.html` .

Hornak, Joseph. 1997. The Basics of MRI. `http://www.cis.rit.edu/htbooks/mri/` .

Johnson, Keith, and Alex Becker. 1997. The Whole Brain Atlas. `http://www.med.harvard.edu/AANLIB/home.html` .

Mahan, Steven L. 1996. Resolution enhancement. `http://aurora.phys.utk.edu/~mahan/enhancement.html` .

Makivic, Miloje. 1996. Bicubic interpolation. `http://www.npac.syr.edu/projects/nasa/MILOJE/final/node36.html` .

Mnuk, Michal. 1997. Lagrange interpolation (in German). `http://www.risc.uni-linz.ac.at/people/mmnuk/FHS/MTD/MAT2/Skriptum/K5node3.html`.

Rodriguez, Paul. 1995. MRI Indications for the Referring Physician. `http://www.gcnet.com/maven/aurora/mri/toc.html` .