Distance Field Manipulation of Surface Models

Bradley A. Payne and Arthur W. Toga
UCLA School of Medicine

This surface manipulation technique uses distance fields—scalar fields derived geometrically from surface models—to combine, modify and analyze surfaces.

Scientific visualization frequently gives rise to complex models of surfaces. In biomedical imaging, organs such as the brain provide elaborate structures to image and analyze. Surfaces might come from interactive techniques, such as outlining,[1] or automatic ones like isosurface creation.[2] Surface models of simpler objects (such as cubes and spheres) are also common, and you can use a number of techniques to image such models.[3,4] But manipulating surfaces using either direct or implicit methods presents a number of challenges. To manipulate surfaces more effectively, we developed a method that uses *distance fields*—the scalar fields derived from triangle-based surface models.

We face four specific challenges when trying to manipulate the surfaces in a visualization model:

1. Averaging/interpolation: What is an "average" of two or more surfaces? Given models of different objects, what is the typical or repre-

0272-17-16/92/0100-0065$03.00 ©1992 IEEE

sentative surface? Finding a representative surface allows interpolation, continuously deforming one surface into another.
2. Offsets: What is the surface a fixed distance above or below a given surface? What is the surface a fixed proportion between two given surfaces? The cortex of the brain, for example, is organized into layers (laminae) that might be accessible by offset surfaces.
3. Blending: How can we combine multiple surfaces in one view with continuous joins between adjacent objects? This combining is similar to what we do with blobby objects in implicit surface modeling or blends and fillets in CAD/CAM.[5,6]
4. Blurring: How can we represent a surface with less resolution or detail while preserving large-scale features? For example, can we remove high frequency errors while preserving lower frequency information?

Direct manipulation of surfaces does not address these four challenges. CAD techniques using boundary representations of objects emphasize Boolean operations.[7] Working directly with surfaces requires separate and complex techniques for each problem. This might be a necessary cost in CAD/CAM, where exact representations are crucial for manufacturing, but we often need alternative approaches when we try to manipulate surfaces that come from discretely sampled data.

Implicit methods offer highly flexible object manipulation. Blinn,[8] Wyvill,[9] and Perlin[10] built objects starting with *scalar fields,* functions defined at each point in space. They used these variously named blobby objects, soft objects, or hypertextures in place of surface modeling. They combined and distorted fields in different ways, producing a final field that they could volume render or use to produce an isosurface.

To apply these implicit surface methods to given surface models, we must first represent the models by scalar fields. A suitable field is the distance field, which represents the distance from a surface as a signed magnitude. The sign designates inside/outside with inside positive, and the magnitude shows the distance from the point to the nearest point on the surface.

Previously, we used surface models' distance fields to remove surfaces[11] and interpolate between upper and lower levels of an object.[12] Raya and Udupa used voxel models' distance fields to interpolate structures in two and three dimensions.[13] Blum originated distance-based formulation for objects and represented distance fields in terms of a symmetric axis (location of the local maximum of a distance field) and radius function (value of distance field defined over symmetric axis).[14]

You can use distance fields to do all the previously mentioned operations readily in three standard steps:

1. Compute a distance field over a 3D sampling lattice from the given surface or surfaces.
2. Perform operations such as addition or filtering on the resulting field or fields, producing a final field.
3. Extract a new surface as the isosurface of the final field.

This approach's strength is that it lets simple 3D algorithms substitute for potentially very complex 2D methods.

Distance fields

Step one is computing the distance field from input surface models. Since input surfaces are often complex, a high sampling rate is desirable, but this makes distance field computation very expensive. Appropriate data structures for searching the surface model can reduce this cost substantially. We first discuss computing distance from single triangles, then build an optimized algorithm for computing the distance field from an entire closed surface.

Single triangle distance

First, we need to compute the distance from a triangle in space to a specified location, the query point. We can reduce this problem to two dimensions by defining a rigid transformation that places the triangle into the x, y plane, then transforming both triangle and query point accordingly. We can then break the desired distance into the distance-to-plane component, represented by the z component of the transformed query point, and the within-plane distance, the distance between the x, y components of the transformed query point and the transformed triangle.

Within the plane, the nearest point on the triangle might be inside the triangle (distance 0), on any edge, or on any vertex for seven possible cases. We can determine which case applies by classifying the point with respect to each edge: inside/outside of edge direction and right/left of edge extent. We can use these classifications to determine which of the seven cases applies, determining the nearest feature. We then compute the in-plane distance from this feature.

Surface distance field: Naive algorithm

We can now combine the single-triangle distances to provide the minimum distance field for a triangulated surface. The simplest, and slowest, method of computing surface distance is the triangle distance of minimum absolute value over all triangles:

```
for each loc (x, y, z) to sample
  dist = ∞
  for = each triangle t
    newdist = distance_to_triangle(loc, t)
    if abs(newdist) < abs(dist)
      dist = newdist
    end if
  end for
end for
```

Optimizations

We can raise several objections to this algorithm. In addition to being very slow, it does not address the sign computation adequately. We commonly get tie distances of opposite sign when the nearest surface point lies on an edge. Surface models might also have discontinuities, creating more problems with the sign. Although this is not strictly correct for a surface that is supposed to have an inside and outside, we should be able to handle such cases. We can use several optimizations to mitigate these problems:

1. Compute the sign separately. For each z-level sampling plane, intersection with the surface model produces line segments in closed loops. Discontinuities in the model leave gaps in the loops, gaps that can be filled by selective matching of unpaired points. Then use the winding rule to scan convert these loops, providing a bitmask for inside/outside.
2. Compute selectively. Depending on the operations to be performed on the fields, we might need only some regions. We can derive isosurfaces of composite fields by root finding and evaluating fields at neighboring points to the roots. Since the distance field is derivative bounded, this should be straightforward. Raya and Udupa used selective computation for voxel model interpolation.[13] Since we use fields for many purposes, we favor techniques that provide the entire field.
3. Postpone square root. Euclidean distance calculation takes a square root at the last step, at substantial cost. Since we are concerned only with the closest point in the model and get the sign separately, we can derive the minimum squared distance and take the square root of only the closest candidate squared distance.
4. Precompute triangle tables. Triangles being mapped to the x, y plane are subject to the same transformations each time, so we can precompute all information for them. We need to transform only the query point at each step.
5. Organize hierarchically. We can organize triangles into a hierarchical tree of bounding boxes. Possible methods include octree and k-dimensional tree.[15] Our method closely resembles the k-dimensional one, except sibling nodes are allowed to overlap. This lets each triangle occupy exactly one box at each level, and this is necessary since splitting our already large models quickly exceeds any plausible memory constraints.

We can use breadth-first traversal of the bounding box tree to find the minimum distance. At each stage, the bounding box gives a quick best and worst minimum distance possible for objects within. We can maintain a pessimistic estimate from the minimum of the worst estimates. Boxes at a given level whose best estimates are worse than this pessimistic figure cannot provide the minimum distance, so we discard them. Boxes that survive this culling are split to the next level of search. We use triangle distance computation to evaluate terminal nodes—boxes that bottom out to lists of triangles—for actual minimum distance. The search method is somewhat analogous to pruning a game tree; the "opponent" is the assumed likelihood that children of a box are positioned as far away as possible.

This improved algorithm does not give logarithmic behavior, since candidate boxes are not always reduced to a single choice at each level. But the algorithm substantially accelerates distance evaluations—about two orders of magnitude for our models.

```
input:   tree: bounding box tree made from surface model
output:  distfld (x, y, z): distance field of the surface model
variables:
         candl: candidate list of bounding box nodes
         top: top node of bounding box search tree
         loc: query point
         front: a bounding box node
         signmask(x, y): value ±1 specifying in/out for model

for each level z
         compute signmask(x, y) for level z
         for each x, y
            sign = signmask(x, y)
            candl = ∅
            append top to candl
            front = first_element(candl)
            worst = upper bound of dist² to front
            while = (candl ≠ ∅)
               front = first_element(candl)
               delete_first_element(candl)
               newbest, newworst = lower, upper bounds of
                  dist² to front
               if (newbest ≤ worst)
                  if front is a terminal node
                     newworst = worst estimate of
                        dist² to triangle list of front
                  else
                     append front children to candl
                  end if
                  worst = min(worst, newworst)
                  end if
            end while
            distfld(x, y, z) = sign · √worst
         end for
end for
```

Figure 1. The improved algorithm for distance fields.

The improved algorithm in Figure 1 uses a bounding box tree and separate sign computation to produce a distance field faster and with more tolerance for discontinuities in the input model.

Alternate fields for blobby modelers

The distance field has a gradient magnitude of one everywhere (except the medial axis) and goes to negative infinity outside the object. For the kind of operations done by the blobby modelers, we might prefer an alternative. Here the field should go to zero at a sufficient distance from the object and have a maximum of one in the object's center. Thus, we make the field more of a fuzzy-set or probabilistic characterization of the object. We can control the magnitude of the gradient in the region of the object boundary to give greater or lesser "blobbiness" to the object. We can transform the distance field to a function designed to meet these criteria. We use Perlin's gain function,[10] such that the value blob(d) for distance field $d(\vec{r})$ with maximum radius R is

$$\text{blob}(d(\vec{r})) = \begin{cases} 1 & \text{if } d(\vec{r}) = R \\ 1/2 & \text{if } d(\vec{r}) = 0 \\ 0 & d(\vec{r}) \le -R \end{cases}$$

This leaves a remaining parameter to adjust sharpness of rise/fall in the vicinity of the surface.

Surface manipulation

Once we compute the distance fields for the input surfaces, we can combine or modify them to produce surface effects.

Offsets—surface removal

Distance fields were originally developed for surface removal, computing the surfaces a fixed distance into a model, as in Figure 2. We can compute these simply by taking the isosurfaces of the distance field. This method lets us compute surfaces of fixed thickness from arbitrarily convoluted surfaces (equivalent to Blum's brush fire formulation[14]). Removed surfaces inside a concave initial surface become increasingly jagged and might split into multiple components, each of which ultimately disappears. Outside the surface, they become cloudlike and ultimately approach spheres.

The motivation for surface removal was approximation of the brain's cortical laminae, the layers of cortical tissue. Layers of varying thickness present a complication not addressed by surface removal. Using varying depth surface removal with solid texturing of function provides an alternative to the typical sweeps of cutting planes. Here the cutaway is one or more curved surfaces determined by the shape of the object itself.

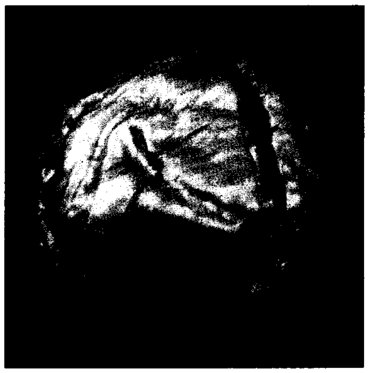


Figure 2. Offset: We produced this surface (shown solid textured) as an offset of the outer transparent surface. We constructed the distance field for a brain surface (transparent) first, then constructed the subsurface (colored) as an isosurface of the field.

Surface interpolation

We can use interpolation of distance fields to compute surfaces that are intermediate between two "keyframe" surfaces. To seek a surface with a specified proportion r between the distance d_1 from one key and the other d_2, distances oppositely directed, we require

$$d_1 = -rd_2$$

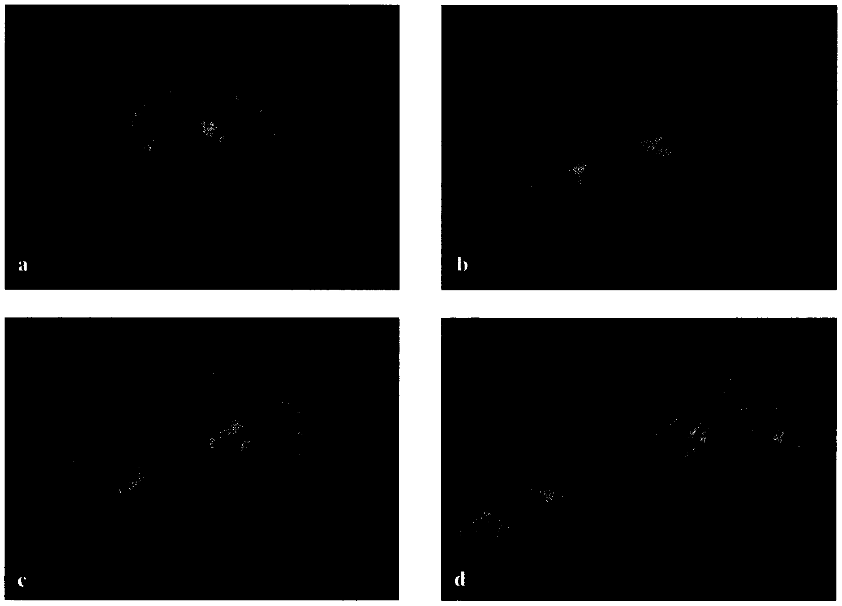


Figure 3. Interpolation: This figure illustrates the change in shape and form of the developing rat brain by interpolating the surface models of two different sampled time points. We computed distance fields from models derived from early embryonic (a) and neonatal stages (d). Weighted averages of the distance fields followed by isosurface construction results in the intermediate models of development (b, c).

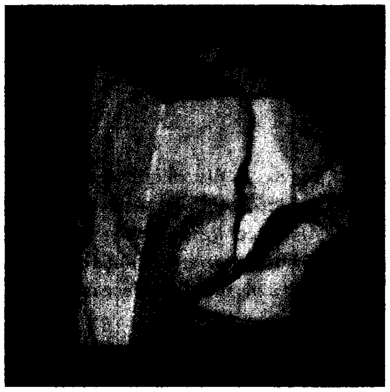


Figure 4. Spatially weighted interpolation: Interpolation between a brain and a cube (blockhead modeling). We computed distance fields for the two objects as in Figure 3. This was followed by nonconstant-weight interpolation, which favors the brain on one side and the cube on the other. We then rotated the interpolated object.

Letting $t = r/(r + 1)$ and rearranging, we get

$$(1 - t)d_1 + td_2 = 0$$

That is, the desired surface is the 0-level isosurface of the linearly interpolated keyframe fields.

This type of interpolation is natural for finding intermediate layers between an upper and lower surface, but we can get interesting results by applying it to more general cases, as in Figure 3. We can animate transformations from one surface to another by varying the interpolation value.

Interpolation between objects with substantially different radii (maximum distance field value) leads to the smaller object's disappearing quickly. Weighting fields so all have equal radius gives equal importance to both keys:

$$(1 - t)(d_1/R_1) + (t)(d_2/R_2) = 0$$

Multiple-object averaging

We can carry out multiple-object averaging similarly, combining fields with weights summing to one. We often need a "reference" or "standard" object. We can construct it from a multiple-object average of many sample structures.

Spatially-weighted interpolation

Spatially varying weighting functions can be used to interpolate between objects within one view. For Figure 4, we used weights proportional to x and $1 - x$.

Texturing

We can get interesting texture effects by combining object distance fields with synthetic textures, rather than with other object distance fields. Figure 5 shows a surface model modulated by a distance field derived from a cloud of random points. This type of texturing quickly raises the required sampling frequency, as textures generally contain higher spatial frequencies than the objects they are applied to.

Surface blending

Researchers studying blobby or soft objects emphasized blending of objects rather than interpolation. Individual components were simple fields, of interest only as building material. We can use the blobby version of the distance field similarly, producing multiple objects with joins between them. Subtractive blobby fields serve for bites or indentations. We used blobby fields for Figure 6.

Surface blurring and compression

To remove extraneous information or noise or to reduce storage requirements, we might want to produce a less detailed

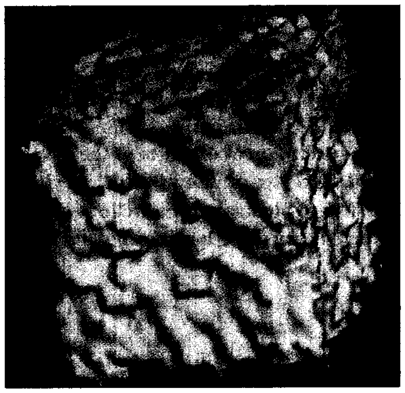


Figure 5. Texturing: We computed distance fields for a cube and for a random collection of points. By interpolating these initial fields, we constructed a new field, then isosurfaced it.

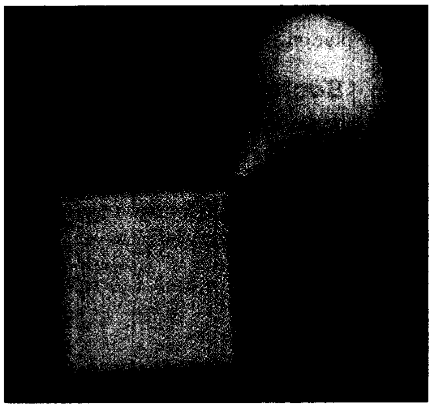


Figure 6. Blending: Surfaces blended by the blobby-field technique. We computed distance fields for initial cube and sphere models. We converted these to blobby distance fields, which were then combined and isosurfaced.

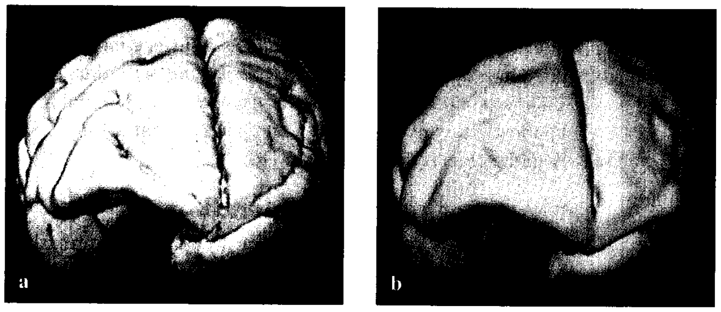


Figure 7. Blurring: We computed a distance field from a brain surface (a). We filtered the distance field with a Gaussian, then an isosurface construction (b). This is a way to blur and smooth surface models.

Figure 8. A visualization of one slice of the distance field of a brain model. In this slice we show the blobby distance value as height. The surface boundary is the yellow-green border. The field is continuous and slope-bounded but not smooth.

version of surface models. This is extremely difficult to do with surfaces directly, but it's straightforward with distance fields. We use a low-pass filter (a Gaussian in this example) to convolve the field. We can also downsample the field before isosurface creation. Figure 7 shows a surface smoothed by distance field convolution.

Boolean operations

We can compute the union and intersection of objects trivially, since we only need the sign of the distance to provide a sort of object bitmask. Since positive sign denotes the inside of the object, a reasonable union field for two objects a, b with distance fields $d_a(\vec{r})$, $d_b(\vec{r})$ can be derived from

$$d_{\cup}(\vec{r}) = \max(d_a(\vec{r}), d_b(\vec{r}))$$

intersection from

$$d_{\cap}(\vec{r}) = \min(d_a(\vec{r}), d_b(\vec{r}))$$

These derived fields are not equivalent to the distance fields of the object unions and intersections, since portions of the surfaces eliminated in the true union/intersection still influence the combined field. The combined fields do have the right sign and gradual variation to provide the correct 0-level union and intersection isosurfaces.

We can use the blobby field to perform probabilistic Boolean operations. Instead of a classifying function like min or max, we can use $\text{blob}(d_a) \cdot \text{blob}(d_b)$ for intersection and $\text{blob}(d_a) + \text{blob}(d_b)$ for union. We can also compare surfaces using the difference of the blob fields; high values denote regions of significant difference between surfaces, where one surface is significantly closer than the other. The 0-value tail of the blob function ensures that where both surfaces are "far away," we can detect no significant differences. Because of the unlimited base of support, subtracting the original distance fields produces large spurious difference regions.

Properties of the distance field

The distance field is continuous but not smooth. We can see this in Figure 8, which shows one slice of a blobby distance field as a height field. Since input triangle-based surface models have the same limitation, the distance field does not additionally worsen matters. For spline-based surfaces with good continuity between patches, a smoother field would be useful; distance fields have gradient discontinuities even when the source surface is smooth.

Interpolation quality

Object-averaging is a little explored and ill-defined area. Chen and Parent interpolated and averaged objects slice by slice, identifying corresponding points in the slice pairs.[16] This requires a knowledge of the "same" slices in both objects and demands limited complexity in the slice outlines.

Distance field interpolation offers a true 3D averaging scheme. Objects do not have to have a similar shape or identifiable corresponding features. And neither object is restricted to a set number of components. Components in different objects influence each other during averaging if they occupy overlapping regions of space.

Since our method does not rely on establishing corresponding points on the key surfaces, it is both general and automatic, but for the same reason, we cannot reliably interpolate similar surfaces. For example, the "natural" interpolation between two orientations of an object is a minimal rotation and translation. Distance fields will not seek out optimal combinations of shape-preserving motions as they do not address preserving shape or volume properties. Features on the interpolated objects that might obviously correspond will not blend into each other during interpolation unless there is an initial spatial overlap. Results are highly sensitive to the initial placement of the objects.

Isosurface creation

The uniform sampling of distance fields limits the detail of constructed surface models. Isosurface creation is also subject to aliasing artifacts. We reduced these considerably by

1. avoiding filtering and downsampling by computing the distance field at the same resolution used for isosurface computation, and
2. avoiding quantization of the distance field by using floating-point distances.

Further improvement requires an adaptive scheme to sample at higher densities in more active regions of the field, such as in the vicinity of small surface features. ❑

Acknowledgments

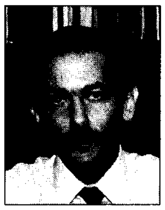
This work was supported by grants from the National Science Foundation (NSF DIR 89-08174) and the National Institutes of Health (ROI RR5956). Thanks to Emily Santori, Allen Lekhtgolts, and Colleen Kelly for their contributions and technical assistance.

References

1. C. Levinthal and R. Ware, "Three-Dimensional Reconstruction from Serial Sections," *Nature*, Vol. 236, No. 5344, 1972, pp. 207-210.
2. W.E. Lorensen and H.E. Cline, "Marching Cubes: A High Resolution 3D Surface Construct Algorithm," *Computer Graphics* (Proc. Siggraph), Vol. 21, No. 3, July 1987, pp. 163-169.
3. B.T. Phong, "Illumination for Computer Generated Pictures," *Comm. ACM*, Vol. 18, No. 6, June 1975, pp. 311-317.
4. L. Carpenter, "The A-Buffer: An Antialiased Hidden Surface Method," *Computer Graphics* (Proc. Siggraph), Vol. 18, No. 3, July 1984, pp. 103-108.
5. C.M. Hoffmann, *Geometric and Solid Modeling: An Introduction*, Morgan Kaufmann, New York, 1989.
6. A.P. Rockwood, "The Displacement Method for Implicit Blending Surfaces in Solid Models," *ACM Trans. Graphics*, Vol. 8, No. 4, Oct. 1989, pp. 279-297.
7. M. Mantyla, *An Introduction to Solid Modeling*, Computer Science Press, Rockville, Md., 1988.
8. J.F. Blinn, "A Generalization of Algebraic Surface Drawing," *ACM Trans. Graphics*, Vol. 1, No. 3, July 1982, pp. 235-256.
9. G. Wyvill, C. McPheeters, and B. Wyvill, "Data Structure for Soft Objects," *Visual Computer*, Vol. 2, No. 4, Aug. 1986, pp. 227-234.
10. K. Perlin and E.M. Hoffert, "Hypertexture," *Computer Graphics* (Proc. Siggraph), Vol. 23, No. 3, July 1989, pp. 253-262.
11. B.A. Payne and A.W. Toga, "Surface Mapping Brain Function on 3D Models," *IEEE CG&A*, Vol. 10, No. 5, Sept. 1990, pp. 33-41.
12. A.W. Toga, B.A. Payne, and E.M. Santori, "Laminar Analysis of a 3D Reconstruction of the Superior Colliculus," *Soc. Neurosci Abstracts*, Vol. 15, 1989, p. 1792.
13. S.P. Raya and J.K. Udupa, "Shape-based Interpolation of Multidimensional Objects," *IEEE Trans. Medical Imaging*, Vol. 9, No. 1, Mar. 1990, pp. 32-42.
14. H. Blum, "Biological Shapes Visual Science (Part 1)," *J. Theoretical Biology*, Vol. 38, 1973, pp. 205-287.
15. F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.
16. S.E. Chen and R.E. Parent, "Shape Averaging and Its Applications to Industrial Design," *IEEE CG&A*, Vol. 9, No. 1, Jan. 1989, pp. 47-54.

Bradley A. Payne is senior programmer/analyst at the UCLA Laboratory of Neuro Imaging. His research interests include biological visualization, software engineering, and computational geometry.

Payne received his BS in computer science and AB in biology from Washington University in St. Louis in 1987. He is a member of ACM and Tau Beta Pi.



Arthur W. Toga is an associate professor in the neurology department at the UCLA School of Medicine. He is also director of the Laboratory of Neuro Imaging and assistant chairman of the Department for Research Affairs. His research interests include metabolic brain mapping, cognitive neuroscience, and brain structure visualization.

Toga received his MS and PhD degrees from St. Louis University. He is a member of ACM and the IEEE Computer Society.

Address correspondence to Arthur W. Toga at the Laboratory of Neuro Imaging, Department of Neurology, UCLA School of Medicine, 710 Westwood Plaza, Los Angeles, CA 90024. Payne's e-mail address is payne@loni.ucla.edu, and Toga's e-mail address is toga@-loni.ucla.edu.