Pre-Computed Illumination for Isosurfaces

Kevin M. Beason[a], Josh Grant[b], David C. Banks[a], Brad Futch[a], and M. Yousuff Hussaini[a]

[a]Florida State University, Tallahassee, USA;
[b]Pixar Animation Studios, Emeryville, USA

ABSTRACT

Commercial software systems are available for displaying isosurfaces (also known as level sets, implicit surfaces, varieties, membranes, or contours) of 3D scalar-valued data at interactive rates, allowing a user to browse the data by adjusting the isovalue. We present a technique for applying global illumination to the resulting scene by pre-computing the illumination for level sets and storing it in a 3D illumination grid. The technique permits globally illuminated surfaces to be rendered at interactive rates on an ordinary desktop computer with a 3D graphics card. We demonstrate the technique on datasets from magnetic resonance imaging (MRI) of the human brain, confocal laser microscopy of neural tissue in the mouse hippocampus, computer simulation of a Lennard-Jones fluid, and computer simulation of a neutron star.

Keywords: global illumination, isosurface, level set, photon mapping, texture mapping, shape from shading

1. INTRODUCTION

An isosurface (also known as a level set, implicit surface, variety, or contour) of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is the locus of points $L_c = \{x : f(x) = c\}$ at which f attains the value of some constant c. The constant c is called the isovalue. The function f is explicit (*i.e.*, parametric) whereas the level set is defined implicitly. Level sets are employed in surface modeling to create a blobby object, and used in visualization to display a three-dimensional scalar field. In an interactive session, a user browses the scalar data by adjusting the isovalue while the visualization system generates the corresponding isosurface. Several techniques have been devised to accelerate the performance of the original Marching Cubes algorithm[1] for extracting level sets of a 3D scalar function; see the survey paper by Sutton *et al.*[2] for a comparison of fast techniques for generating isosurfaces.

Scalar-valued 3D datasets abound in science, engineering, and medicine; often they contain level sets with complicated and nearly-touching shapes. Global illumination provides important shape cues (including shadows and inter-reflections) to the human visual system for inferring shape from shading[3].[4] It is therefore often desirable to apply global illumination to isosurfaces in order to make their shape more comprehensible. Recently, several different approaches have been demonstrated that accelerate the computation of global illumination in a scene. They exploit multi-processing,[5] careful sampling strategies,[6] and precomputing radiance transfer,[7] among other techniques. In the future, commercial systems for visualization may integrate these sophisticated techniques into their core renderer, allowing the display of isosurfaces that are generated and globally-illuminated in real time.

In the meantime, we aim for the practical goal of bringing global illumination to the casual user of graphics tools, such as the physicist, engineer, biologist, or designer who employs a commercial software tool to visualize a 3D scalar dataset by viewing its isosurfaces. Without re-implemention of the renderer "under the hood" of the software package, and without making special demands of the underlying graphics hardware, how can interactive global illumination be inserted into an existing 3D visualization system that was not originally designed to incorporate it? Our answer is to create a 3D texture and apply it onto an isosurface.

Further author information: (Send correspondence to K.M.B.)
K.M.B.: E-mail: beason@cs.fsu.edu, Telephone: 1 850 644 5437
J.G.: E-mail: jag@pixar.com
D.C.B: E-mail: banks@cs.fsu.edu, Telephone: 1 850 644 5437
B.F.: futch@cs.fsu.edu
M.Y.H.: myh@csit.fsu.edu

Stewart's "Vicinity shading"[8] is similar to our technique in that it stores illumination values in a 3D texture. Vicinity shading assumes a uniform spherical light surrounds the volume and calculates isosurface occlusions in a small region near each voxel and stores the lighting in a 3D texure. During display, isosurfaces are texture-mapped and combined with a headlight for a combination of view-independent and view-dependent lighting. This produces soft self-shadowing, which is a partial solution to global illumination, but it is not a full solution since it does not account for color bleeding (indirect illumination), caustics, or discrete luminaires.

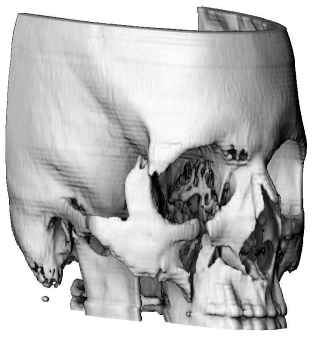
The result of illuminating a level set from medical data using a spherical light source together with self-shadowing is shown in Figure 1(a). But it is certainly possible that the end user desires a more sophisticated rendering. For example, the skull dataset might include a lesion. The user may wish to position the skull in a 3D scene (such as the Cornell box) for spatial reference, to make the surface translucent and the lesion emissive in order to see the lesion within the volume, and to use inter-reflection to help infer the position of the lesion even when it is obscured. Figure 1(b) shows the same isosurface rendered by incorporating these additional features. Global illumination supports such a combination of lighting and reflectance properties; unfortunately, the user's visualization system might not implement global illumination, or else may provide a renderer that requires minutes or hours to produce an image.

Our method (which includes vicinity shading as a special case) constructs a 3D texture whose colors capture the illumination solution for the isosurfaces passing through the volume. By pre-computing the illumination throughout the volume on an illumination grid, we prepare a 3D texture that gracefully incorporates global illumination into an ordinary isosurface generator. As a result the user can sweep through different level sets in real time, even when they are rendered with global illumination techniques that may themselves be quite slow to compute; that is, the cost of illuminating individual isosurfaces is amortized over the entire 3D volume.

Our method of applying photorealistic lighting to scalar valued 3D datasets is described in the following three sections. Section 2 describes sampling level sets of a scalar function, Section 3 explains how to generate the 3D texture, and Section 4 describes how to display the isosurfaces with the precomputed illumination grid.

2. SAMPLING OF LEVEL SETS

Our eventual goal is to illuminate level sets of a function $f : \mathbb{R}^3 \to \mathbb{R}$ but the idea can be presented more easily in a lower dimension, using a function $f : \mathbb{R}^2 \to \mathbb{R}$. Each level set $L_c = \{\mathbf{x} : f(\mathbf{x}) = c\}$ is the set of points where the function f attains the value c. There are infinitely many level sets of the height field f, so precomputing the illumination for each one of them would require infinite time. Instead, some strategy must be employed to sample a finite collection of the level sets. Two obvious strategies are to uniformly sample (1) the domain or (2) the range of f. We discuss each strategy in turn, plus a hybrid of the two.


(a) Vicinity Shading

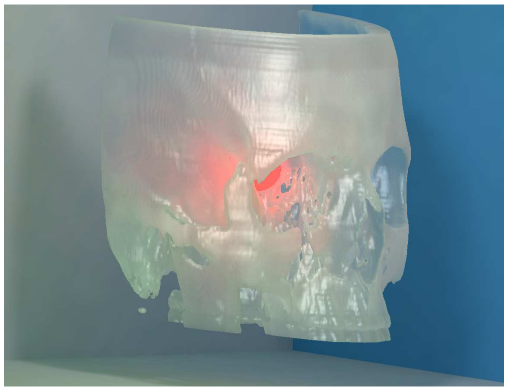

(b) Global illumination

Figure 1. Level set of 3D MRI data shows surface of human skull. Above: Spherical light source plus shadows convey 3D shape. Below: Additional shape cues are provided by self-shadowing, inter-reflection, and internal light source positioned at an internal lesion (simulated by the red blob behind the eye).

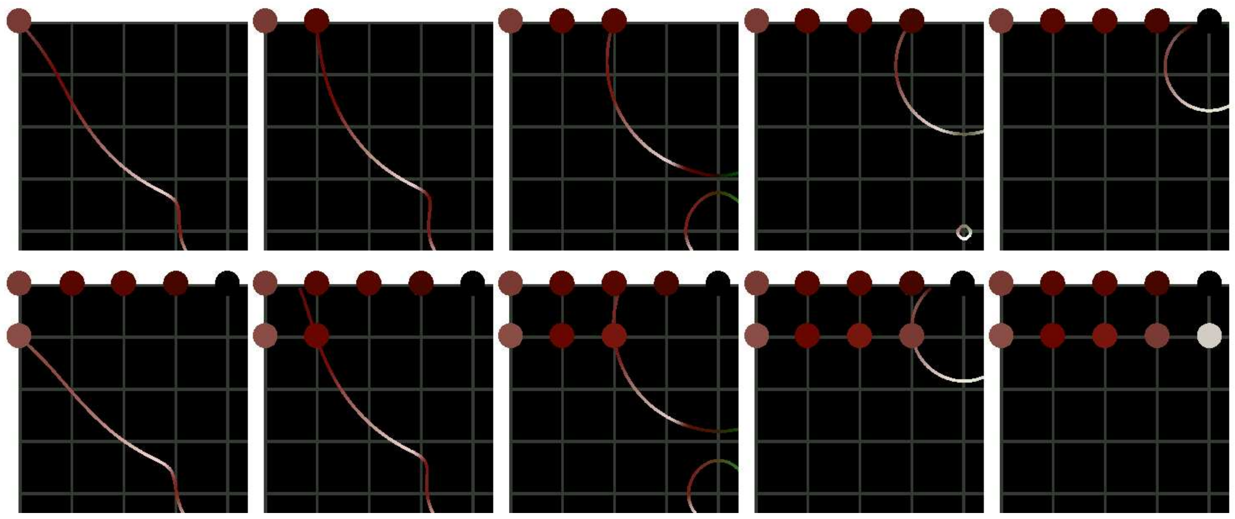


Figure 2. Level sets (isocurves) of a 2D scalar function. Top row: Through each grid point on the first row of the domain a level set is constructed; a light source (positioned at bottom of grid) illuminates the level set; the color at the grid point is retained in the 2D illumination grid. Bottom row: Same process applied to second row.

2.1. Uniform sampling in the domain

A conceptually simple, brute-force approach for precomputing the illumination of level sets is to generate the level sets that pass through regulary-spaced samples (gridpoints) in the domain of f. At each gridpoint x, the level set corresponding to the value $f(x)$ is created and then illuminated; the resulting radiance function for that single gridpoint is stored in the illumination grid.

Figure 2 illustrates this approach of uniform sampling in the domain of a 2D function $f : \mathbb{R}^2 \to \mathbb{R}$. The gridpoint determines a level set; the level set is illuminated in the 2D plane; the gridpoint is then shaded with the result from illuminating the level set through that point. Once this array of red-green-blue (rgb) values is computed, a point on any arbitrary level set (whether passing through a gridpoint or not) can be assigned a color by interpolating neighboring colors stored in the illumination grid.

2.2. Uniform sampling in the range

On a 2D grid, sampling each isocurve and applying 2D global illumination may be a manageable affair because of the modest size of the grid, but in $\mathbb{R}^3$ the strategy becomes impractical. Sampling the isosurfaces through 1000^3 gridpoints of a 3D dataset would yield one billion isosurfaces. Even if each isosurface could be generated and globally illuminated in only a millisecond, the computation would require nearly two weeks to complete. A more efficient approach follows from the observation that many scalar datasets (such as medical data acquired from magnetic resonance imaging) are represented by 8-bit unsigned integers, so only 256 distinct isovalues pass through the grid points. Looping over these 256 values captures every isosurface through the grid, even for large grids.

Figure 3 (left) illustrates the technique of sampling evenly-spaced isovalues in the range of f, computing the corresponding level sets, and illuminating each of them in turn. Although sampling k isovalues in the range of f is more efficient than looping over each of the m gridpoints in the domain of f (when $k \ll m$), we immediately observe some difficulties raised by uniform sampling of the isovalues in the range.

The first problem is that the level sets corresponding to values c_k in the range do not, in general, pass through any of the gridpoints when f is a floating-point scalar. Since the texture map is evaluated on a uniform grid of rgb colors, the values on the grid must be estimated from the isosurface's off-grid colors by employing a scattered data interpolation technique.

The second problem is that the level sets are very close together when the gradient magnitude ∇f is large and far apart when the the gradient is near zero. *Dense* level sets pose a problem if their exitant radiance changes

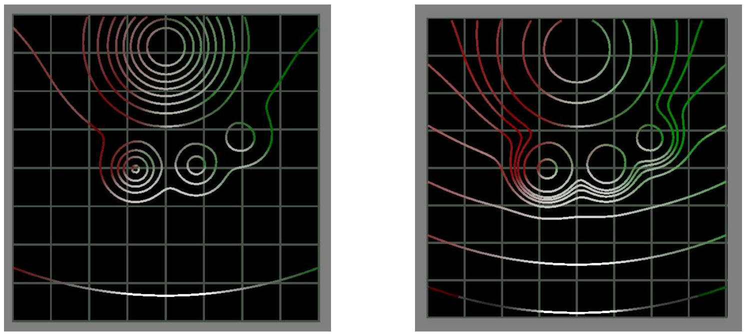


Figure 3. Left: level sets L_c resulting from uniform sampling in the range with constant stepsize. Right: level sets resulting from adaptive sampling with stepsize $\delta(c)$.

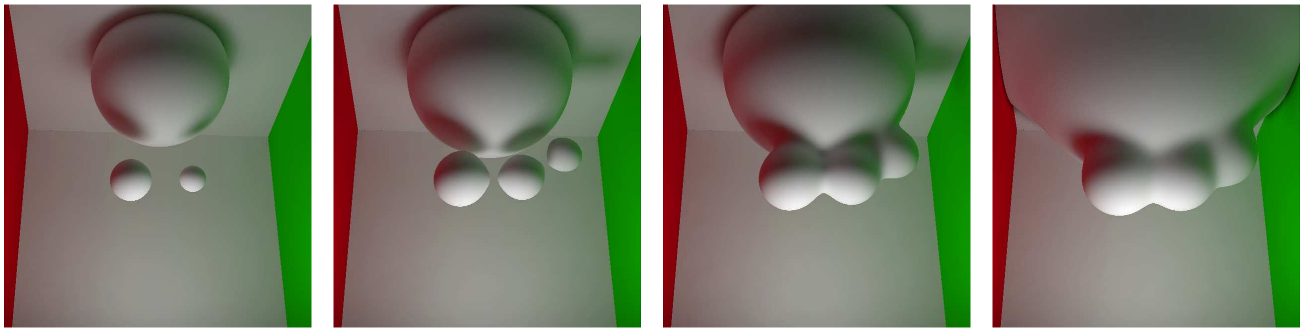


Figure 4. Level sets of 3D scalar function (analogous to level sets of 2D scalar function in Figure 3). Adaptive sampling is used to generate a sequence of isosurfaces L_c that fill the 3D volume.

quickly between gridpoints, which can occur if a sharp shadow boundary lies between gridpoints or if the normal vector changes quickly. If the interpolation filter kernel is too small, aliasing may result from undersampling the illuminated level sets; if the filter kernel is too large, light leaks from one level set to another. One solution to this problem is to use a multi-level grid to refine the regions where the change in the quantity of interest (*e.g.,* radiance or surface normal) exceeds a threshhold[9].[10] Another approach would be to replace the regular grid with an irregular mesh that captures discontinuities (or nonlinear changes) in the color.[11] *Sparse* level sets pose a problem because they undersample the illumination grid. This problem can be addressed by increasing the width of the filter function when the gradient magnitude is small. A simpler solution is to estimate the step size that should be used in order to preserve a minimum spatial distance between level sets. This approach is described in the following section.

2.3. Adaptive sampling

The level sets shown in figure 3 (left) are evenly spaced in the range, but they leave many of the grid cells completely empty. A hybrid approach to sampling permits the stepsize in the range of f to be chosen adaptively to meet a constraint on the sampling distance in its domain. Define the minimum gradient magnitude *gMin(c)* on the level set L_c by

$$gMin(c) = \min_{\mathbf{x}\in L_c} |\nabla f(\mathbf{x})|$$

From this minimum we estimate that the level set $L_{c+\delta(c)}$ is separated from L_c in the domain by as much as

$$\epsilon = \max_{\mathbf{x}\in L_c}\left(\frac{\delta(c)}{|\nabla f(\mathbf{x})|}\right) = \frac{\delta(c)}{\min\limits_{\mathbf{x}\in L_c} |\nabla f(\mathbf{x})|} = \frac{\delta(c)}{gMin(c)}$$

To keep the next level set within ϵ of L_c, the loop variable is incremented by $\delta(c)$ rather than being incremented by a fixed step size. To prevent incrementing by zero in the case of a zero gradient, one may add a "softening parameter" k^2 yielding

$$\delta(c) = \epsilon \ (gMin(c) + k^2)$$

as the step size. Thus

$$c \leftarrow c + \delta(c)$$

determines the next isovalue. Figure 3 (right) shows the result of this approach: with adaptive sampling, the same number of level sets become spaced such that each grid cell is visited at least once. Note that the resolution of the illumination grid can be completely decoupled from the resolution of the scalar dataset. As a consequence, when a scalar function has level sets with rapid color changes within a grid cell, one may construct the illumination grid at a finer resolution. The resolution needed to maintain a certain error tolerance depends on the interpolation scheme that will be employed when the illumination grid is texture-mapped onto reconstructed isosurfaces: piecewise constant texturing requires a finer illumination grid than cubic interpolation in order to produce similar tolerances.

3. CREATING THE ILLUMINATION GRID

Upon selecting appropriate isovalues for isosurface extraction, it is then necessary to apply realistic lighting to these surfaces, and having done that, to store their collective data in a 3D texture. We now present our approach to these two latter tasks.

3.1. Illuminating the level sets

For a given 3D scalar dataset, level sets are chosen using adaptive sampling described in Section 2.3. We typically chose between 15-50 samples, although more level sets might be needed depending on the complexity of the dataset. The isosurfaces are extracted using Marching Cubes and stored to disk. We place each level set in a 3D scene (such as the Cornell box, complete with walls and overhead light) in order to create a realistic environment with natural lighting. We illuminate the scene with a photon-mapping ray tracer[12] that stores radiance values per vertex. The radiance values are estimated by sampling the exitant radiance in the direction normal to the surface at the vertex. Figure 4 shows the result of illuminating four level sets of a simple 3D scalar valued function.

Rendering all these isosurfaces may very well be a time-consuming precomputation job. But the point of the algorithm is to treat global illumination as a batch-processing chore to be performed before the user ever explores the level sets in the 3D data. Thus when the user examines the 3D data by sweeping through its level sets, the proper illumination is texture-mapped onto them in real-time.

3.2. Interpolating to produce the texture

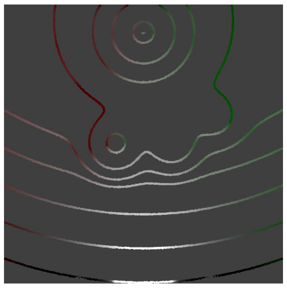
When the user sweeps through the level sets of a 3D scalar dataset, the resulting isosurfaces fill the 3D volume (namely, the domain of the scalar function f). The idea is to determine the illumination at points on the level sets and then to store them in a 3D texture map (the illumination grid). However, since the vertices very likely do not fall on grid points, a scattered data interpolation technique is used in order to estimate the illumination at grid points of the 3D texture map. We implemented Shepard's method for interpolating the illumination from non-regular vertex positions of the level sets,[13] but other scattered-data interpolation techniques could be tried as well. Figure 5 (top) illustrates how a collection of level sets in $\mathbb{R}^2$ is filtered to produce a 2D texture map; Figure 5 (bottom) illustrates how a collection of level sets in $\mathbb{R}^3$ is filtered to produce a 3D texture map.

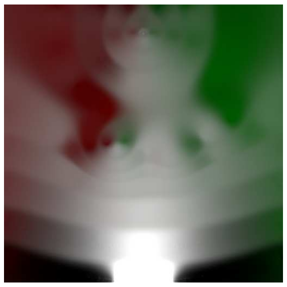
4. USING THE ILLUMINATION GRID

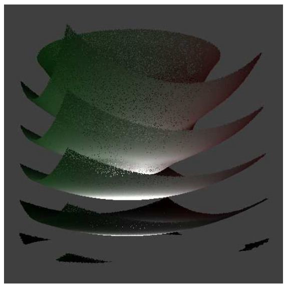
Although rendering the level sets and interpolating the results into an illumination grid may be a very time consuming process, using the resulting grid as a texture map is quite fast. This section describes two implementations of an interactive graphics system that can generate an arbitrary isosurface, interpolate colors from the illumination grid, and assign them to the surface interactively. First, we demonstrate the technique using a commercial visualization product, which proves that photo-realistic rendering can be enjoyed by the casual user of a commercial visualization system without the need for any special graphics support beyond 3D rendering and texturing, and without requiring modification to the commercial product's rendering engine. Next, we show how the illumination grid can be used within the framework of a custom visualization system for even better results.

4.1. Incorporating into commercial software

The "amira" visualization system from Template Graphics Systems (TGS) is a commercial visualization tool used for imaging medical, energy, government, and engineering data. It has a dataflow interface patterned after that of the Application Visualization System (AVS) ,[14] and provides isosurface generation coupled with texture mapping. We incorporated our illumination grid within amira without any need to modify the software itself. Figure 6 (top) shows the dataflow network we used to combine the illumination grid with amira's isosurface module. At the top is the node that reads the 4-channel rgba illumination grid, which feeds into the isosurface module (shown in yellow). The isosurface module is also fed by the scalar MRI data of the brain[15] (decimated to 109^3). We drive the isovalues by remote control from amira's "Animate" module. The "MovieMaker" module


(a) 2D level sets


(b) 2D illumination grid


(c) 3D level sets

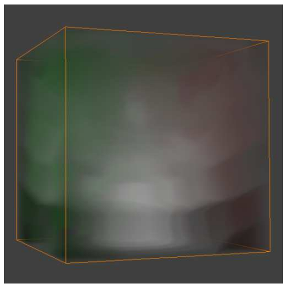
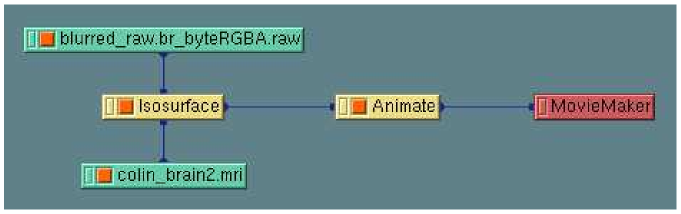
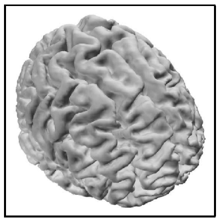

(d) 3D illumination grid

Figure 5. Top row: Level sets of $f : \mathbb{R}^2 \to \mathbb{R}$ are illuminated and their values interpolated onto a 2D texture. Bottom row: Level sets of $f : \mathbb{R}^3 \to \mathbb{R}$ are illuminated and their values interpolated onto a 3D texture.





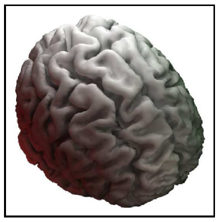


Figure 6. Incorporating pre-computed global illumination into the commercial visualization tool "amira." Top: Data flow network of amira modules. Bottom: (left) Level set displayed with amira's native lighting model; (right) displayed with 3D global illumination texture.

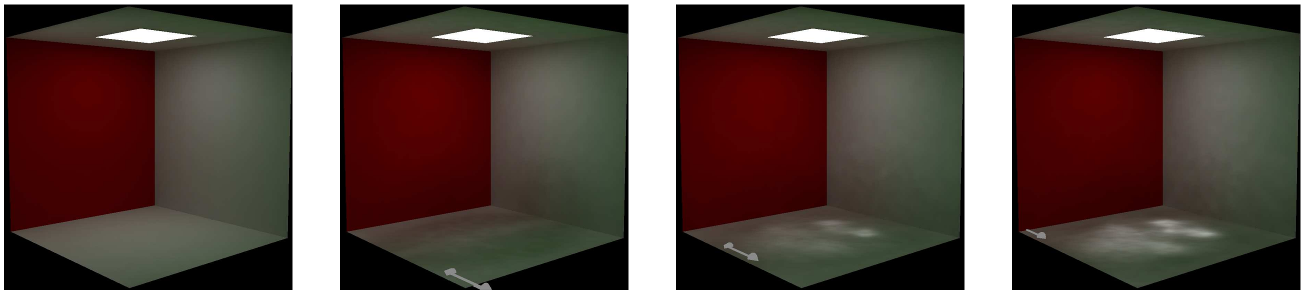

(a) $f = 150$ (b) $f = 100$ (c) $f = 50$ (d) $f = 2$

Figure 7. When isosurfaces of the brain dataset $f : \mathbb{R}^3 \to \mathbb{R}$ (see Figure 8) are semi-transparent, they produce reflections, shadows, and caustics on the walls and floor of the surrounding environment. Images a-d show just the walls and floor as if various level sets of the brain were in the scene, at $f = 150$, $f = 100$, $f = 50$, and $f = 2$. Each vertex of the walls and floor stores a vector of color values corresponding to its appearance for different values of f.

(shown in red) allows the movies to be saved to disk.

On an Intel Dual Xeon 1.7GHz machine running Linux with 1GB of memory and an nVidia GeForce FX 5900 Ultra, amira completed a sweep of constructing 126 level sets from the MRI dataset in 44 seconds using its default local illumination for rendering the level sets. One of the level sets is shown in Figure 6 (bottom left). By comparison, the sweep through the same level sets using the illumination grid as shown in Figure 6 (bottom right) required 48 seconds. Thus, amira was able to display level sets with global illumination, incurring a mere 10% additional overhead. The point of the demonstration is that this commercial visualization package, whose rendering engine was not even intended to produce global illumination, can nevertheless be coaxed into displaying globally illuminated level sets in real time. Creating the 3D texture map as a preprocess completely decouples this operation from the users' real-time sessions during which they explore level sets from their data. As a result, our technique permits the scientific user to enjoy high-quality renderings without demanding any modification whatsoever of the commercial product.

4.2. Incorporationg into custom software

The illumination grid allows commercial software such as amira to display globally illuminated isosurfaces. But additional effects, such as shadows cast into the 3D scene, are missing. These cues may be important to the user for apprehending 3D shape, so we would like to include them in the display.

We wrote a custom visualization tool to produce globally illuminated isosurfaces together with additional effects. This software tool, written using Open Inventor,[16] accepts meshes with multiple colors at each vertex. This vector of colors allows the tool to display dynamic shadows and caustics on the floor and walls as the level sets change shape: for level set L_c, we store the corresponding luminance at the vertices of ordinary surfaces (not including level sets) in the scene. Our viewer interpolates these colors in lock-step with the isovalue c the user selects, so that when a new isosurface is generated its caustics and shadows are displayed on the ordinary surfaces in the scene. This process is illustrated in Figure 7 which shows the effects level sets (not shown) on their surrounding environment.

The illumination grid contains only a single (view-independent) color at each point. Our custom viewer approximates the full view-dependent solution by adding the missing directional component using OpenGL. Since we create the illumination grid via photon mapping, the photons are shot from the luminaires and may sustain multiple bounces with non-diffuse surfaces. This allows us to compute the contribution from light paths represented by the regular expression L(S|D)*D, where L represents a light source, S represents a specular (directional) reflection, and D represents a diffuse reflection.[17] The expression ends with a diffuse reflection; to include the specular component of the final gather, we combine Open Inventor's two lighting models "base color" and "Phong." We texture-map using the illumination grid as the base color during the first (view-independent) pass, using no lights since the illumination grid already stores the values of the illuminated level sets. Then we

Dataset	Res.	# Iso.	Precompute	Software Texturing
			Time/Iso (s)	Time/Iso (s)
Brain	109^3	52	1626	0.3
Dirt	101^3	20	393	0.2
Neuron	150^3	45	772	0.8
Nucleon	101^3	17	480	0.6

Table 1. Rendering vs. texture-mapping times for datasets shown in Figure 9.

re-render the polygons in a second pass to produce the view-dependent specular highlights. During the second pass, the polygons are rendered with additive alpha blending. Figure 8 shows the two separate passes and the combined result. Note that this combination reduces the rendering rate by a factor of two.

5. RESULTS

We created illumination grids for datasets representing four distinct disciplines: medicine, nanochemistry, microbiology, and nuclear physics. Each dataset contains isosurfaces with complicated shapes, making them candidates for the improved shape-from-shading cues that global illumination offers.

Figure 9 (top row) shows our custom viewer's rendering of a translucent MRI dataset of the brain. The opalescent inter-reflections and caustics are very sensitive to the surface geometry, so although the last two isosurfaces have almost indistinguishable shapes, a user can infer that they are different based on the caustics they produce on the floor. On a desktop machine, our custom viewer swept through 120 level sets of the dataset at 1.06 seconds per level set using the illumination grid alone: 0.20 sec represents the overhead incurred by texture-mapping, with the remainder spent constructing the level sets. Using color-indexed walls, applying two-pass reflectance, and calculating gradients, the same sweep incurred an additional 0.06 sec for interpolating the color vectors on the walls and floor.

Figure 9 (second row) shows different iso-densities in a molecular dynamics simulation of explosively boiling fluid lifting a nanoparticle of dirt. Translucency allows the user to see the shape of the fluid and the sphere simultaneously.

Figure 9 (third row) shows different iso-densities of laser microscopy of a mouse neuron. The velvety skin-like texture of the neuron comes courtesy of subsurface scattering within the isosurfaces.

Figure 9 (bottom row) shows different iso-densities of protons (red) and neutrons (white) in a computational simulation of stellar material in a neutron star. Neutrino transport is implicated in the explosion of these stars; neutrinos couple more strongly with proton/neutron clusters that exhibit a periodic spatial structure, which is suggested by the shadows on the floor.

Table 1 contains the render times for the datasets used in our results shown in Figure 9. The first two columns indiciate the name of the dataset and its resolution. The third column indicates how many isosurfaces

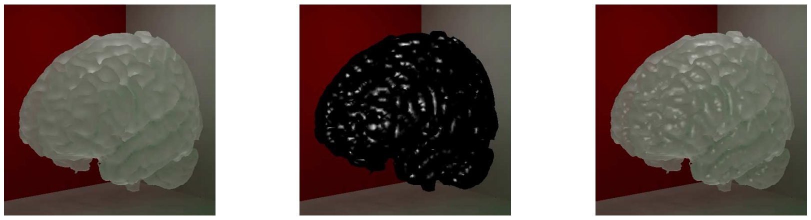

(a) Texture (b) Specular (c) Combined

Figure 8. Two rendering passes capture the (a) view-independent and (b) specular components of the final gather; these are combined (c) using additive alpha blending.

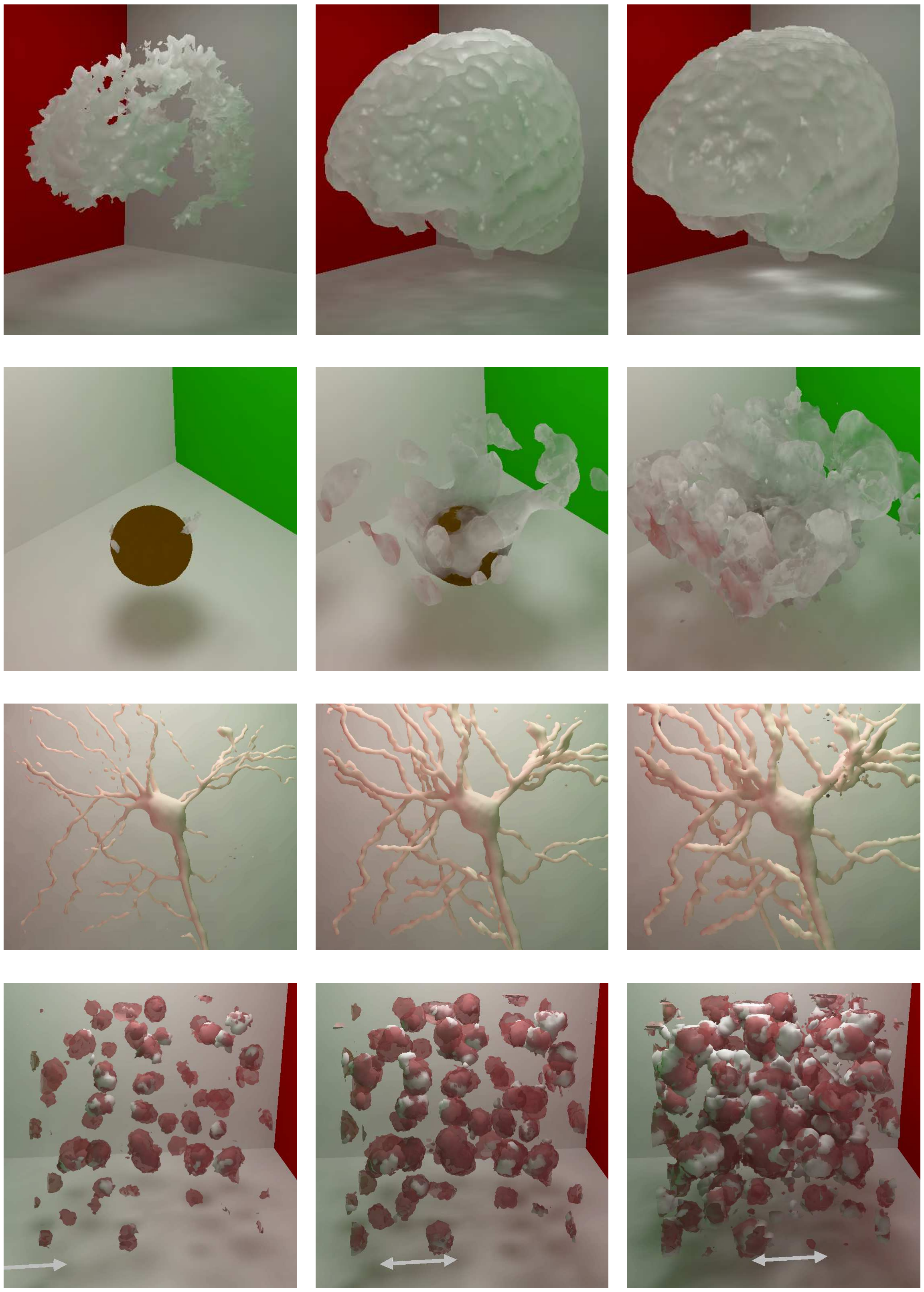


Figure 9. Enhancements of pre-computed global illumination integrated into a custom isosurface engine. The arrow at the bottom is a 3D widget that the user slides to adjust the isovalue interactively.

were illuminated to build the 3D illumination grid. The fourth column shows how many seconds, on average, were spent illuminating each isosurface. The last column shows how many seconds, on average, were spent indexing into the illumination grid to texture-map vertices of the level sets within our custom viewer. Open Inventor offered the convenience of its graphical user interface (GUI), including 3D widgets within the scene, but does not support hardware 3D textures; as a result the texture interpolation is performed per-vertex in software. Notice that rendering an isosurface with the illumination grid was about 1000 times faster than rendering it using software. These times were measured using a Dual Xeon 3.0GHz PC with 4GB of memory. Our results would be even faster by using hardware texturing or even hardware shaders, an area of future work that will further improve the result of using the illumination grid. Note that after the isosurface is constructed and the illumination grid is applied at its vertices, the user can fly through the globally illuminated scene at full speed (*e.g.* 60Hz, depending on the graphics card). The point of the table is not to suggest that our implementation of global illumination is particularly fast; rather, the table illustrates that even a renderer that requires minutes to produce an image can still generate a 3D illumination grid which can later be exploited to produce the same globally-illuminated image in real time.

6. SUMMARY

We have developed a technique for displaying globally-illuminated isosurfaces at interactive rates. The technique is based on illuminating individual level sets of a height field. The resulting illumination is stored in a grid and texture-mapped onto the level sets that a user explores with a visualization tool. One strength of this texture-mapping approach is that it can be used with a commercial visualization system to display photo-realistic renderings of isosurfaces without the need to re-implement the software internal to the system: we demonstrated the technique using the "amira" visualization system. We also implemented a custom isosurface generator that interpolates color vectors at vertices of the static environment surrounding the level sets. This option allows us to display shadows cast on walls by opaque isosurfaces, and caustics cast on walls by translucent or shiny isosurfaces. The result is that images with photo-realistic quality are available at interactive rates for scientists, engineers, designers, and clinicians who browse scalar-valued 3D volumes.

The authors gratefully acknowledge NSF grants #0083898 and #0430954.

REFERENCES

1. W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *Computer Graphics (Proceedings of ACM SIGGRAPH '87)* **21**(3), pp. 163–169, 1987.
2. P. M. Sutton, C. D. Hansen, H.-W. Shen, and D. Schikore, "A case study of isosurface extraction algorithm performance," in *VisSym '00: Joint Eurographics-IEEE TVCG Symposium on Visualization (Amsterdam)*, W. de Leeuw and R. van Liere, eds., pp. 259–268, Springer, 2000.
3. C. Madison, W. Thompson, D. Kersten, P. Shirley, and B. Smits, "Use of interreflection and shadow for surface contact," *Perception and Psychophysics* **63**, pp. 187–194, 2001.
4. H. H. Hu, A. A. Gooch, W. B. Thompson, B. E. Smits, J. J. Rieser, and P. Shirley, "Visual cues for imminent object contact in realistic virtual environments," in *Proceedings Visualization 2000*, T. Ertl, B. Hamann, and A. Varshney, eds., pp. 179–185, 2000.
5. S. Parker, W. Martin, P.-P. J. Sloan, P. Shirley, B. Smits, and C. Hansen, "Interactive ray tracing," in *Symposium on Interactive 3D Graphics*, pp. 119–126, 1999.
6. A. Keller, "Instant radiosity," *Computer Graphics* **31**(Annual Conference Series), pp. 49–56, 1997.
7. P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments," *ACM Trans. Graph.* **21**(3), pp. 527–536, 2002.
8. A. J. Stewart, "Vicinity shading for enhanced perception of volumetric data," in *Proceedings of the 2003 IEEE symposium on Visualization*, p. 47, Institute of Electrical and Electronics Engineers, Inc., 2003.
9. R. Westermann, L. Kobbelt, and T. Ertl, "Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces," *The Visual Computer* **15**(2), pp. 100–111, 1999.
10. G. H. Weber, O. Kreylos, T. J. Ligocki, J. M. Shalf, H. Hagen, B. Hamann, and K. I. Joy, "Extraction of crack-free isosurfaces from adaptive mesh refinement data," *Hierarchical and Geometrical Methods in Scientific Visualization* , pp. 19–40, 2003.

11. F. Durand, G. Drettakis, and C. Puech, “Fast and accurate hierarchical radiosity using global visibility,” *ACM Transactions on Graphics* **18**(2), pp. 128–170, 1999.
12. H. W. Jensen, *Realistic Image Synthesis Using Photon Mapping*, A. K. Peters, 2001.
13. D. Shepard, “A two-dimensional interpolation function for irregularly-spaced data,” in *Proceedings of the 1968 23rd ACM national conference*, pp. 517–524, ACM Press, (New York, NY, USA), 1968.
14. C. Upson, J. Thomas Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam, “The application visualization system: A computational environment for scientific visualization,” *IEEE Computer Graphics and Applications* **9**(4), pp. 30–42, 1989.
15. C. Holmes, R. Hoge, L. Collins, R. Woods, A. Toga, and A. Evans, “Enhancement of MR images using registration for signal averaging,” *Journal of Computer Assisted Tomography* **22**(2), pp. 324–333, 1998 Mar-Apr.
16. J. Wernecke and The Open Inventor Architecture Group, *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor*, Addison-Wesley, 1994.
17. P. S. Heckbert, “Adaptive radiosity textures for bidirectional ray tracing,” in *SIGGRAPH ’90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pp. 145–154, ACM Press, (New York, NY, USA), 1990.