Fast Global Illumination for Visualizing Isosurfaces with a 3D Illumination Grid

Users who examine isosurfaces of their 3D data sets generally view them with local illumination because global illumination is too computationally intensive. By storing the precomputed illumination in a texture map, visualization systems can let users sweep through globally illuminated isosurfaces of their data at interactive speeds.

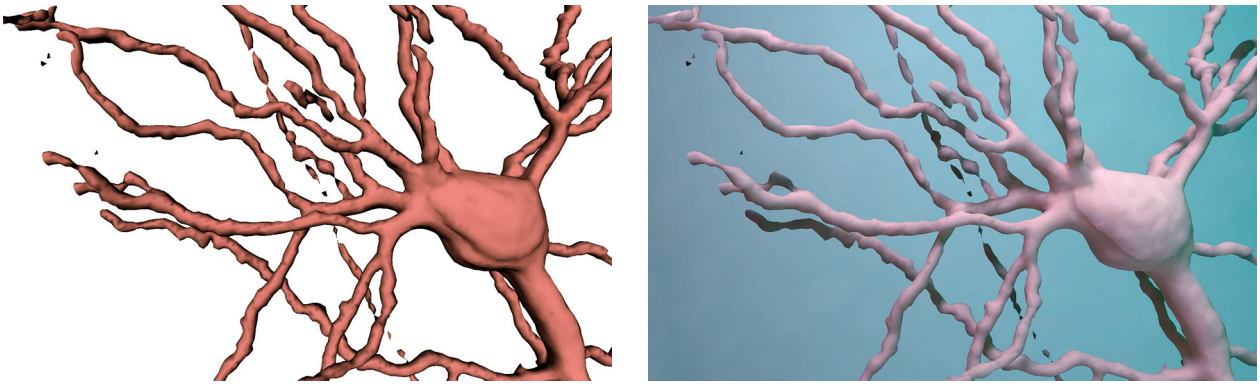
Visualizing a 3D scalar function is an important aspect of data analysis in science, medicine, and engineering. However, many software systems for 3D data visualization offer only the default rendering capability provided by the computer's graphics card, which implements a graphics library such as OpenGL[1] at the hardware level to render polygons using *local illumination*. Local illumination computes the amount of light that would be received at a point on a surface from a luminaire, neglecting the shadows cast by any intervening surfaces and indirect illumination from light reflected from other surfaces. This simplified version of light transport is nonphysical, but it produces images that look 3D and that graphics hardware can generate at rates of millions of triangles per second.

By comparison, more realistic *global illumination* results from solving (or at least approximating) the equation for light transport. Generally, renderers that solve the light transport equation are implemented in software and are consequently much slower than their hardware-based counterparts. So a user faces the choice between fast-but-incorrect and slow-but-accurate displays of illuminated 3D scenes. Global illumination might make complicated 3D scenes more comprehensible to the human visual system, but unless we can produce it at interactive rates (faster than one frame per second), scientific users will continue to analyze isosurfaces of their 3D scalar data rendered with less realistic, but faster, local illumination.

We propose a solution to this trade-off that involves precomputing global illumination and storing the values in a 3D illumination grid. We can perform this task as a batch process on multiple processors. Later, when the user applies a visualization tool to sweep through level sets of the scalar function, the precomputed illumination is simply texture mapped at interactive rates onto the isosurfaces. Consequently, users can perform arbitrarily complex illumination calculations (for example, to capture subsurface scattering or caustics) and display the results in real time while examining different level sets. This strategy lets users apply realistic illumination at interactive speeds in a data visualization software system.

1521-9615/07/$20.00 © 2007 IEEE
Copublished by the IEEE CS and the AIP

David C. Banks
University of Tennessee / Oak Ridge National Laboratory
Joint Institute for Computational Science
Kevin Beason
Rhythm and Hues Studios



(a) (b)

Figure 1. A mouse neuron, imaged with confocal laser microscopy, has numerous long thin dendrites projecting outward from the cell body. (a) With local illumination, it's difficult to discern the dendrites' relative depths. (b) With global illumination, shadows near a crossing (such as the X shape on the lower left) indicate that the two dendrites are nearly touching.

(a) (b)

Figure 2. Two hemispheres of a human brain separated by the cleft of the central commissure. (a) With local illumination, even the cleft's deepest section (the bright crescent at the bottom center) is as bright as the cortex's outermost surface. (b) With global illumination, the cleft is immediately evident as a dark valley.

Global Illumination

Determining the spatial arrangement of one feature with respect to another is a basic concern for a user analyzing 3D data. However, if a scene is complex, with many objects obstructing others, it can be tedious or even impossible for a user to select a viewpoint that clearly discloses the different objects' relative pose. As data sets grow in size, their geometric complexity generally grows as well. Figure 1 illustrates a complicated isosurface rendered with local and global illumination. This data set, imaged *ex vivo* using confocal microscopy, shows a neuron from a mouse hippocampus with several long cylindrical dendrites emanating from the cell body. Global illumination in Figure 1b disambiguates the dendrites' relative depths at crossing points in the image.

Global illumination is also useful for displaying depth in isosurfaces with complicated shapes. Figure 2 shows an isosurface of the brain from magnetic resonance imaging (MRI) data. Global illumination in Figure 2b makes the central commissure (separating the cerebral hemispheres) readily apparent. We presented both images in Figure 2 to one of our collaborators (a neurosurgeon), who remarked that the globally illuminated image in Figure 2b "is what a brain actually looks like." Although making 3D surfaces look real might not be data visualization's most important task, users deserve to have this choice available when they're analyzing data.

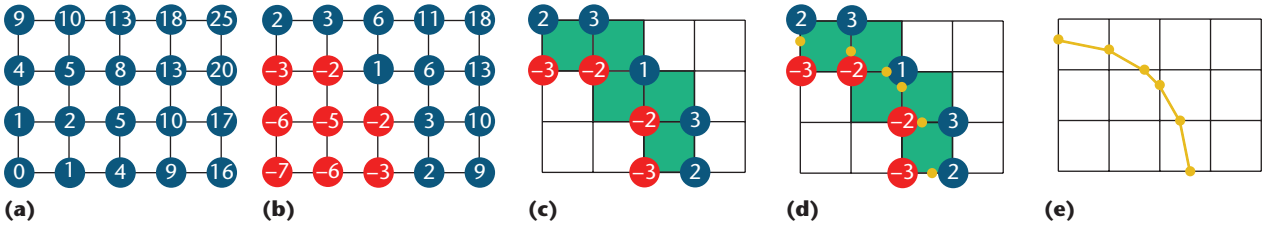


Figure 3. Level set L_c of the function $h(x, y) = x^2 + y^2$. The isovalue $c = 7$ yields a circle of radius $\sqrt{7}$ centered at the origin. (a) h evaluated at grid points. (b) $h - 7$ evaluated at grid points. (c) Squares and edges that straddle negative and positive values of $h - 7$. (d) Roots (yellow) of $h - 7$. (e) The level set L_7 approximated by a polyline (gold).

Isosurfaces

Examining scalar-valued data on a 3D grid is a common starting point for 3D visualization. Such data sets arise, for example, from a patient's MRI, in confocal microscopy of cells and tissue, in the solution of heat flow on a 3D domain, or from computing vorticity magnitude in a turbulent flow. The scalar function $h(\mathbf{x})$: $\mathbb{R}^n \rightarrow \mathbb{R}$ assigns a real value to each point $\mathbf{x}$ in $\mathbb{R}^n$. A *level set* L_c of h is the locus of points $\mathbf{x}$ satisfying $L_c = \{\mathbf{x}: h(\mathbf{x}) - c = 0\}$, which corresponds to points $\mathbf{x}$ lying on the intersection of the graph of h and the hyperplane at height c. Sweeping through level sets is a standard way for a user to examine such a scalar function. In the 3D case ($n = 3$), the level set forms an isosurface in $\mathbb{R}^3$.

A typical 3D data set contains the values of h only at discrete points $\mathbf{x}_a$ on the grid, so we must infer the values of h at nongrid points from the grid values by some interpolation scheme. Researchers have widely used the marching cubes (MC) algorithm[2] for interpolating the data and constructing a polygonal mesh approximating the isosurface. MC uses linear interpolation to locate roots of $h(\mathbf{x}) - c$ at points along edges of cubes tiling the domain and then connects the points into polygons. Figure 3 illustrates a version of the MC algorithm in two dimensions, called *marching squares*, for a scalar function defined on a grid in the plane whose level sets are curves. Figure 3a shows the values at grid points in the plane's positive quadrant of the function $h(x, y) = x^2 + y^2$, which has level sets $L_c = \{(x, y): x^2 + y^2 = c\}$ that are circles of radius $\sqrt{c}$. The grid points define squares that tile the domain. Figure 3b shows the evaluation of $h - c$ when the isovalue c happens to be seven.

A typical software system for generating isosurfaces—the *isosurface engine*—connects the isovalue c with a GUI component that the user sees. The user moves a slider bar to dynamically change the isovalue, and then the isosurface engine generates a polygonal mesh representing L_c. Finally, the graphics card displays the mesh (using local illumination). For scalar data sets of size $256 \times 256 \times 256$, an ordinary desktop computer can perform this pipeline of tasks at interactive rates. Running a global illumination code to make the isosurface look more realistic, however, can add tens of minutes to the display time when the isovalue c is updated.

To achieve an interactive display rate while rendering globally illuminated isosurfaces, the visualization system can absorb the calculation of light transport into a precomputing stage. In the most naive approach, the system would repeatedly extract isosurface after isosurface of the function h, compute global illumination for each isosurface, and then store the illuminated meshes—all before the user even begins an interactive session to sweep through the data. As the user moves a slider bar in the GUI, the visualization system would simply retrieve the mesh with precomputed illumination. The storage cost, however, for archiving an unlimited number of isosurfaces is unreasonably high. A more prudent tactic is to store only the illumination values and then "paint" them onto isosurfaces, a generic technique known as *texture mapping*, which we describe later.

Illumination as a Texture Map

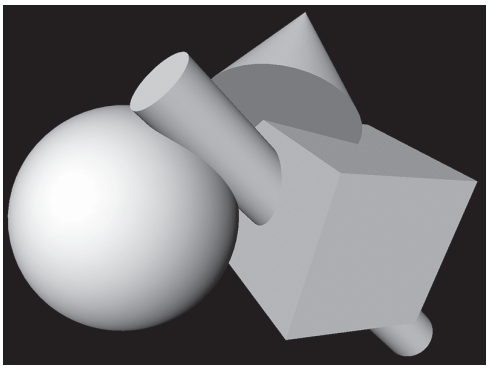
Texture mapping (see Figure 4) assigns a coordinate chart to a surface and then applies colors from a 2D image onto the surface according to the texture coordinates. With 3D texture mapping, we assign each vertex of the mesh a triple (i, j, k) of texture coordinates that index into a 3D volume of colors. We can exploit 3D textures to paint realistic lighting onto polygons much faster than the lighting can actually be computed (in milliseconds rather than minutes).

During the precomputation phase, a series of isosurfaces is constructed and globally illuminated

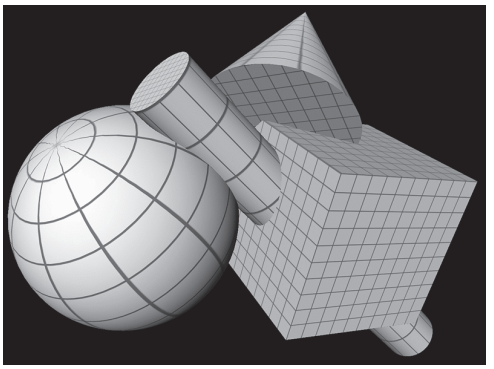
using a numerical solver for light transport. This batch-processing step might take minutes or hours depending on the data's size and the sophistication of the rendering algorithm used. Figures 5a through 5d demonstrate four steps in this process. At many sample points (x, y, z) on a given level set, the incoming light is computed and saved by the numerical solver. Figure 5e depicts thousands of samples of incident light on several isosurfaces. These scattered samples are interpolated by a subsequent process onto a 3D uniform grid, or *illumination grid* (see Figure 5f).

Where the isosurfaces are spaced far apart from each other, they produce visible undersampling artifacts in the illumination grid. A high-quality illumination grid requires a large number of closely spaced level sets during the precomputing phase to produce a dense collection of illumination samples. Ideally, the level sets would be uniformly spaced in the volume, and the precomputing phase would simply increment one isovalue c_m by a constant amount δ to produce the next isovalue $c_{m+1} = c_m + \delta$. But the spacing between level sets depends on the increment δ and the gradient of the scalar function h; when the gradient magnitude is small near a level set L_c, neighboring isovalues produce level sets that are far apart. Kevin Beason and his colleagues[3] (who developed the technical basis for the work we describe in this article) describe different strategies for extracting level sets for use in precomputed global illumination, including an adaptive sampling method that consults the gradient to produce roughly evenly spaced isosurfaces.

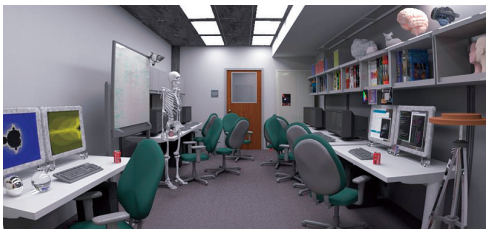
Figure 6 shows a volumetric rendering of the illumination grid (viewed directly from one side) that results from a dense set of isosurfaces. We can interpret the illumination grid in the figure as follows: the isosurface nearest the luminaire (bottom of the box) receives a great amount of incident light over a large region. As the isosurface recedes from the light, the direct illumination from the luminaire becomes less intense. Taken together, this sweep of directly illuminated isosurfaces produces a kind of "light bubble" at the bottom of the illumination grid. Where an isosurface passes through the bubble, it's necessarily in close proximity to the luminaire. On the left and right sides of the illumination grid, we can see light spillage from the red and green walls. The colors aren't due to red and green light scattering through a participating medium; rather, they mark spots on isosurfaces where light has bounced off a colored wall and onto the isosurface. Likewise, the illumination grid's dark portions mark spots where an isosurface is in a shadow.



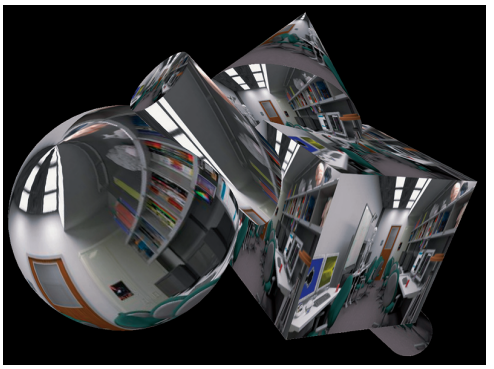
(a)



(b)



(c)



(d)

Figure 4. 2D texture mapping. (a) Each vertex of a polygonal mesh has 3D spatial coordinates (*x*, *y*, *z*). (b) Each vertex is assigned additional coordinates (*i*, *j*) that index into a 2D texture grid. (c) The image texture provides the 2D grid of colors. (d) The color at position (*i*, *j*) in the image texture is applied to points having texture coordinates (*i*, *j*) in the scene.

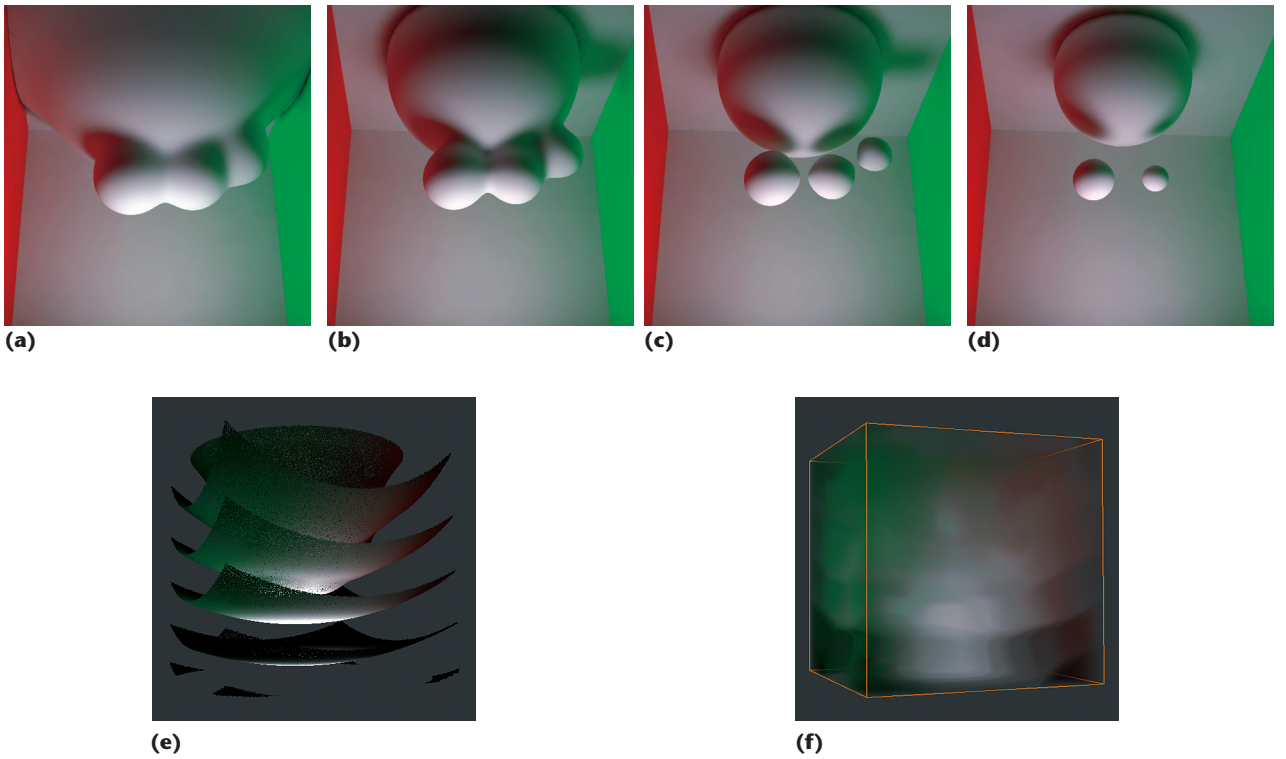


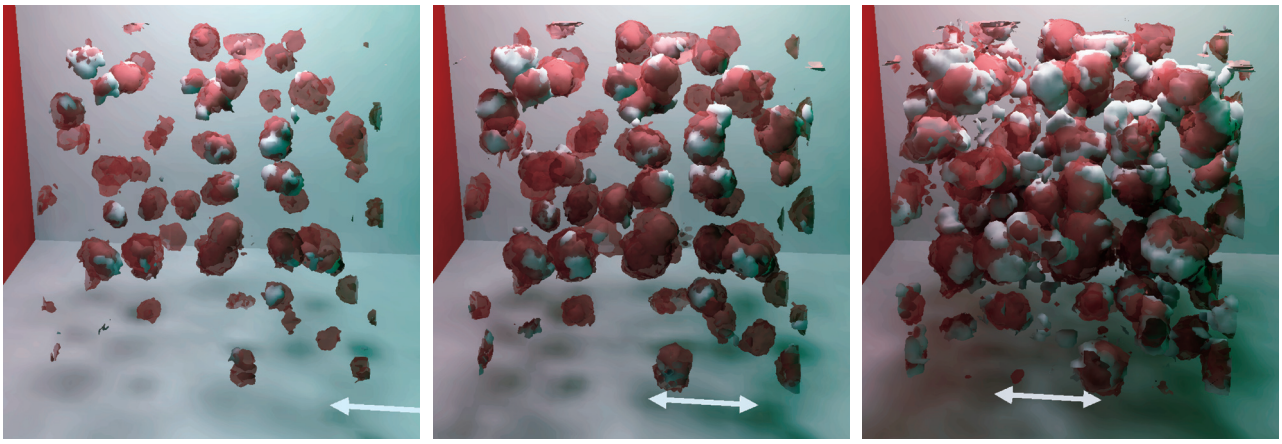
Figure 5. 3D texture constructed from globally illuminated isosurfaces. The surrounding scene includes a box with a luminaire at the bottom, a red wall on the left, and a green wall on the right. (a) through (d) Successive level sets are placed in the scene and globally illuminated. Shadows are evident on the top wall and on the largest blob of the isosurfaces; indirect illumination is evident in the red and green tint on the sides of the isosurfaces. (e) The incoming light is computed at sample points on each level set. (f) The scattered samples are interpolated onto a 3D illumination grid that serves as a 3D texture for later use.

Figure 6. Densely sampled illumination grid. Many isosurfaces, spaced closely within the volume, are each globally illuminated. We interpolated the resulting illumination values onto the 3D uniform grid.

Thus, the illumination grid stores, at each point **x**, the light that would be received if the only surface in the box were the level set L_c passing through **x**. Of course, that level set is the one with isovalue $c = h(\mathbf{x})$. If the illumination grid were a block of solid material, we could carve it away to expose any isosurface, revealing a layer that was precolored in exactly the right way to mimic the effect of being situated inside the illuminated box.

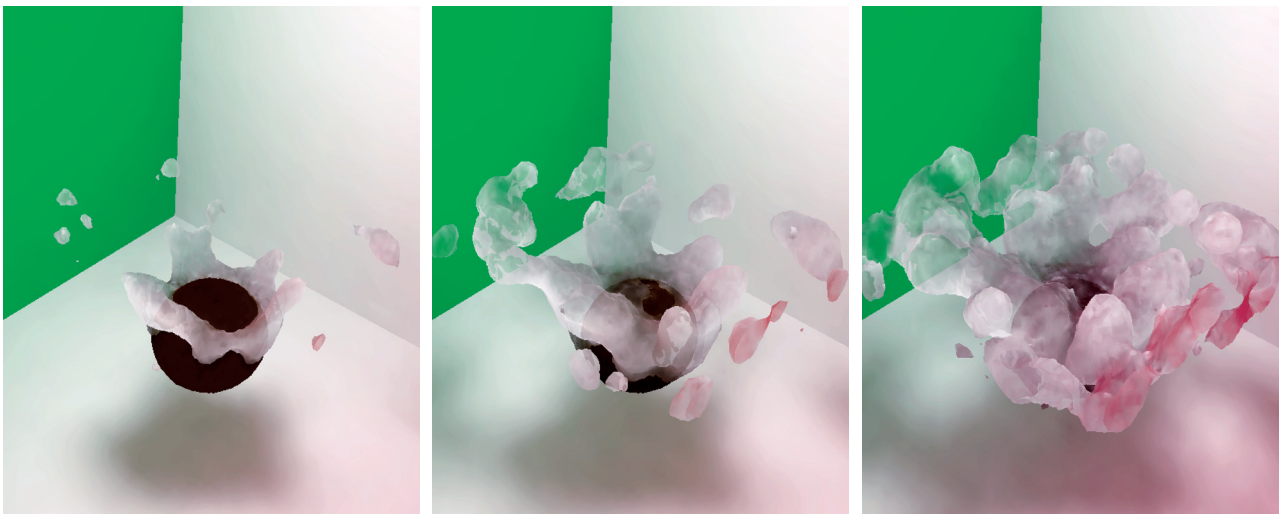
Once the illumination grid has been created by the interpolation step, we can use it as a 3D texture map to provide real-time global illumination of isosurfaces. As the user sweeps through isovalue c, the isosurface engine generates polygonal meshes as usual. In addition to the spatial coordinates (x, y, z) defining each vertex, the polygonal mesh is endowed with a 3D texture index (i, j, k) at each vertex. If the illumination grid is scaled to match the spatial dimensions of the domain of function h, then the spatial coordinates are the same as the texture coordinates—namely, $(i, j, k) = (x, y, z)$.

Some data sets contain two separate scalar quan-



(a) (b) (c)

Figure 7. Nucleon arrangement in a neutron star. At high densities, protons (red) and neutrons (white) form into clusters. We applied global illumination to the entire scene and stored the results in individual textures for the proton and neutron densities. The user sweeps, from (a) to (c), through isodensity values from high to low density.



(a) (b) (c)

Figure 8. Laser-assisted particle removal. A nanoparticle of dirt (brown sphere) adheres to a substrate (white floor in the scene) covered by a thin layer of fluid (translucent white material). Rapidly heating the substrate causes the fluid to vaporize upward, lifting the nanoparticle. The user sweeps, from (a) to (c), through different isodensity values of the fluid, from high to low density.

ties h_1 and h_2 that the user wants to inspect simultaneously. Each scalar function possesses its own level sets, but when the level set $L_{1,c}$ for h_1 is displayed together with the level set $L_{2,c}$ for h_2, light can inter-reflect between the two isosurfaces.

To manage this situation, we construct two separate illumination grids g_1 and g_2 during the preprocessing stage. Each grid contains the average reflected illumination at points on its corresponding level sets. Later when the user selects isovalue c, the isosurface engine constructs a polygonal mesh for $L_{1,c}$ and applies the precomputed illumination texture from g_1. Then, it constructs the mesh for $L_{2,c}$ and applies the texture from g_2.

Figure 7 illustrates this use of two illumination grids for isosurfaces of the density of protons (red) and neutrons (white) in a computational simulation of the arrangement of nucleons in a neutron star. The simulation predicts that protons and neutrons

form these popcorn-shaped clusters at subnuclear densities of 10^{14} g/cm^3.[4] The figure shows an interactive session in which the user moves the 3D widget (bottom of scene) to select different isovalues c.

The illumination grid stores the reflected light at each point on an isosurface, averaged over incident directions. For a diffusely reflecting surface, this single color is sufficient to later display a realistic image. If the isosurface is shiny or translucent, however, a single color value is insufficient to exhibit the directional (view-dependent) values of the reflected or transmitted light. (Beason and his colleagues[3] describe how local illumination provided by the computer's graphics card can approximate this view-dependent component for shiny and translucent surfaces, but a more accurate approach, using spherical harmonics, is available elsewhere.[5])

Figure 8 illustrates this use of texture-mapped illumination together with hardware-assisted lighting for translucent isosurfaces. We used molecular dynamics to simulate a nanoparticle of dirt, covered by a thin film of fluid, resting on a substrate. When we apply heat to the substrate, the fluid layer rapidly changes phases and explosively lifts the dirt from the substrate, a process called *laser-assisted particle removal*.[6] The fluid particles' point masses were averaged by convolution with a point-spread function over finite volumes to produce a scalar density function h. The figure shows different isosurfaces of h, from a single animation frame of time-series data, used to visualize the fluid's wake as it moves past the nanoparticle.

These examples illustrate interactive sessions of a user examining 3D scalar data sets. The precomputed 3D illumination grid lets hardware- or software-based texture mapping apply realistic global illumination to isosurfaces at interactive speeds. The bottleneck for the visualization system becomes the isosurface engine rather than the global-illumination solver.

We're actively pursuing better techniques for sampling the radiance throughout the 3D volume. To limit the maximum error when the illumination grid is texture-mapped to an arbitrary level set, we must run the preprocessing step on many closely spaced isosurfaces. We're investigating how to decouple the process of simulating light transport from the process of extracting isosurfaces prior to generating the illumination grid. The goal is to optimally distribute photon samples without constraining them to lie on a fixed set of isosurfaces, thereby accelerating the costly preprocessing step.

Acknowledgments

The neuron data set in Figure 1 was provided by Charles Ouimet and Karen Dietz, Department of Biomedical Sciences, Florida State University. The brain data set in Figure 2 comes from Colin Holmes at the Brain Imaging Center, Montreal Neurological Institute, McGill University. The laboratory scene in Figure 4c was modeled by Yoshihito Yagi, Department of Computer Science, Florida State University. The nucleon data set was provided by Jorge Piekarewicz, Department of Physics, Florida State University. The nanoparticle data set in Figure 8 was computed by Simon-Serge Sablin and M. Yousuff Hussaini, School of Computational Science, Florida State University. Josh Grant, Pixar Studios, developed the isosurface engine. The interactive 3D display tool used for Figures 7 and 8 was developed by Brad Futch, Department of Computer Science, Florida State University. US National Science Foundation grant numbers 0101429 and 0430954 supported this work.

References

1. D. Shreiner et al., *OpenGL Programming Guide: The Official Guide to Learning OpenGL Version 2*, 5th ed., OpenGL Architecture Rev. Board, Addison-Wesley Professional, 2005.
2. W.E. Lorensen and H.E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics* (*Proc. ACM Siggraph 1987*), vol. 21, July 1987, pp. 163–169.
3. K.M. Beason et al., "Pre-Computed Illumination for Isosurfaces," *Proc. IS&T/SPIE Int'l Symp. Electronic Imaging, Conf. Visualization and Data Analysis* (EI10), R.F. Erbacher et al., eds., vol. 6060, Int'l Soc. Optical Eng. (SPIE), pp. 6060B:1–11.
4. C.J. Horowitz et al., "Dynamical Response of the Nuclear 'Pasta' in Neutron Star Crusts," *Physics Rev. C*, vol. 72, 2005, 031001.
5. C. Wyman et al., "Interactive Display of Isosurfaces with Global Illumination," *IEEE Trans. Visualization and Computer Graphics*, vol. 12, no. 2, 2006, pp. 186–196.
6. K.M. Smith et al., "Modeling Laser-Assisted Particle Removal Using Molecular Dynamics," *Applied Physics A: Materials Science and Processing*, vol. 77, no. 7, 2003, pp. 877–882.

David C. Banks *is a member of the University of Tennessee/Oak Ridge National Laboratory (UT/ORNL) Joint Institute for Computational Science and is a visiting professor of radiology at Harvard Medical School. His research interests include visualizing scalar and vector-valued data sets that arise in science and medicine. Banks has a PhD in computer science from the University of North Carolina at Chapel Hill. Contact him at banks@bwh.harvard.edu.*

Kevin Beason *is a graphics software developer at Rhythm and Hues Studios, developing rendering software for producing animations and special effects for film and TV. Contact him at beason@rhythm.com.*