Vicinity Shading for Enhanced Perception of Volumetric Data

A. James Stewart

School of Computing
Queen's University

Abstract

This paper presents a shading model for volumetric data which enhances the perception of surfaces within the volume. The model incorporates uniform diffuse illumination, which arrives equally from all directions at each surface point in the volume. This illumination is attenuated by occlusions in the local vicinity of the surface point, resulting in shadows in depressions and crevices. Experiments by other authors have shown that perception of a surface is superior under uniform diffuse lighting, compared to illumination from point source lighting.

CR Categories: I.3.6 [Computing Methodologies]: Computer Graphics—Methodology and Techniques

Keywords: volume rendering, shading model, diffuse illumination, perceptual cues

1 Introduction

This paper describes a new illumination model for surfaces within a volumetric data set. The model, which is based upon the idea of "obscurances" [Zhukov et al. 1998], produces more accurate shading than the regular "diffuse–plus–specular" Phong model [Phong 1975] which is commonly used with volume rendering. The chief characteristic of the new model is that depressions, folds, and crevices are shadowed, which provides an additional perceptual cue to surface shape.

Volumetric data is typically illuminated by one or more point light sources, and the shading at each point in the volume is calculated as a sum of diffuse and specular components involving the directions of the light source, L, the viewer, V, and the surface normal (or gradient), N. This makes for a very quick shading calculation, which is simple enough to be implemented in graphics card hardware using the Blinn approximation [Blinn 1977] for the specular component.

This illumination method provides good perceptual cues to the *orientation* of the surface that is embedded in the volume, due to the diffuse $N \cdot L$ term: Surfaces are bright if illuminated from directly above, and dark if illuminated from the side.

But this model gives poor cues to the *relative depth* of a surface. It can be difficult to determine whether one part of the surface is higher or lower than an adjacent part. The ambiguity in relative depth can be reduced by introducing shadows, which provide perceptual cues to the surface shape.

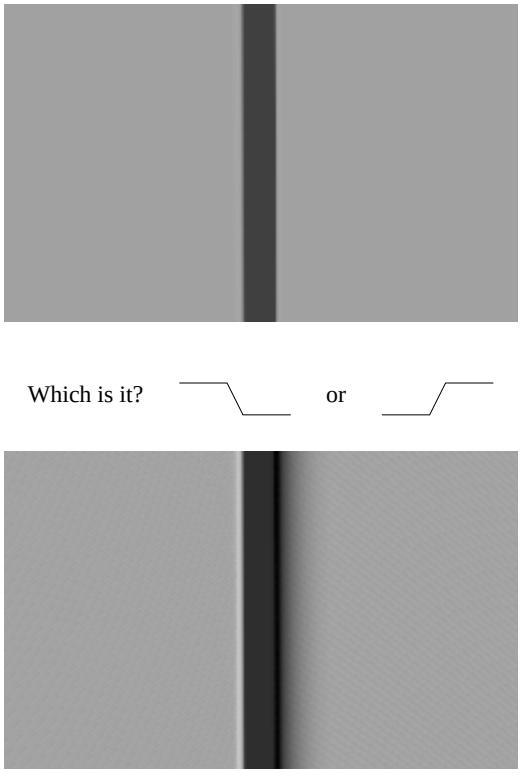


Figure 1: **Top:** Under point–source illumination, a surface provides no cues to the relative depths of its two halves. **Bottom:** Under uniform diffuse illumination, the relative depths are clearer.

The usual way to provide shadows is with ray tracing [Whitted 1980]: From each surface point, rays are shot to each light source and those that arrive without intersecting an object contribute to the point's illumination. But shadows from point sources and opaque objects are sharp, and introduce confusing illumination discontinuities on the surface: It can be difficult for the viewer to distinguish between a dark–to–light transition due to shadowing, and one due to a sudden change in surface orientation.

Better shadowing is provided under *uniform diffuse lighting*, which arrives equally from all directions, much like the illumination on a cloudy day (see Figure 1). With such lighting, the shading of a surface point is a function of the solid angle subtended by the visible part of the light source: Points in valleys don't get much light, whereas points on peaks get a lot of light. Langer and Bülkhoff [Langer and Bülthoff 1997] have shown that, for a common class of surfaces, perception of shape is more accurate under such "cloudy day" lighting than under point lighting.

Distribution ray tracing [Cook et al. 1984] could calculate shading under uniform diffuse lighting, but at a very high computational cost: Many rays must be sent outward from each surface point, in-

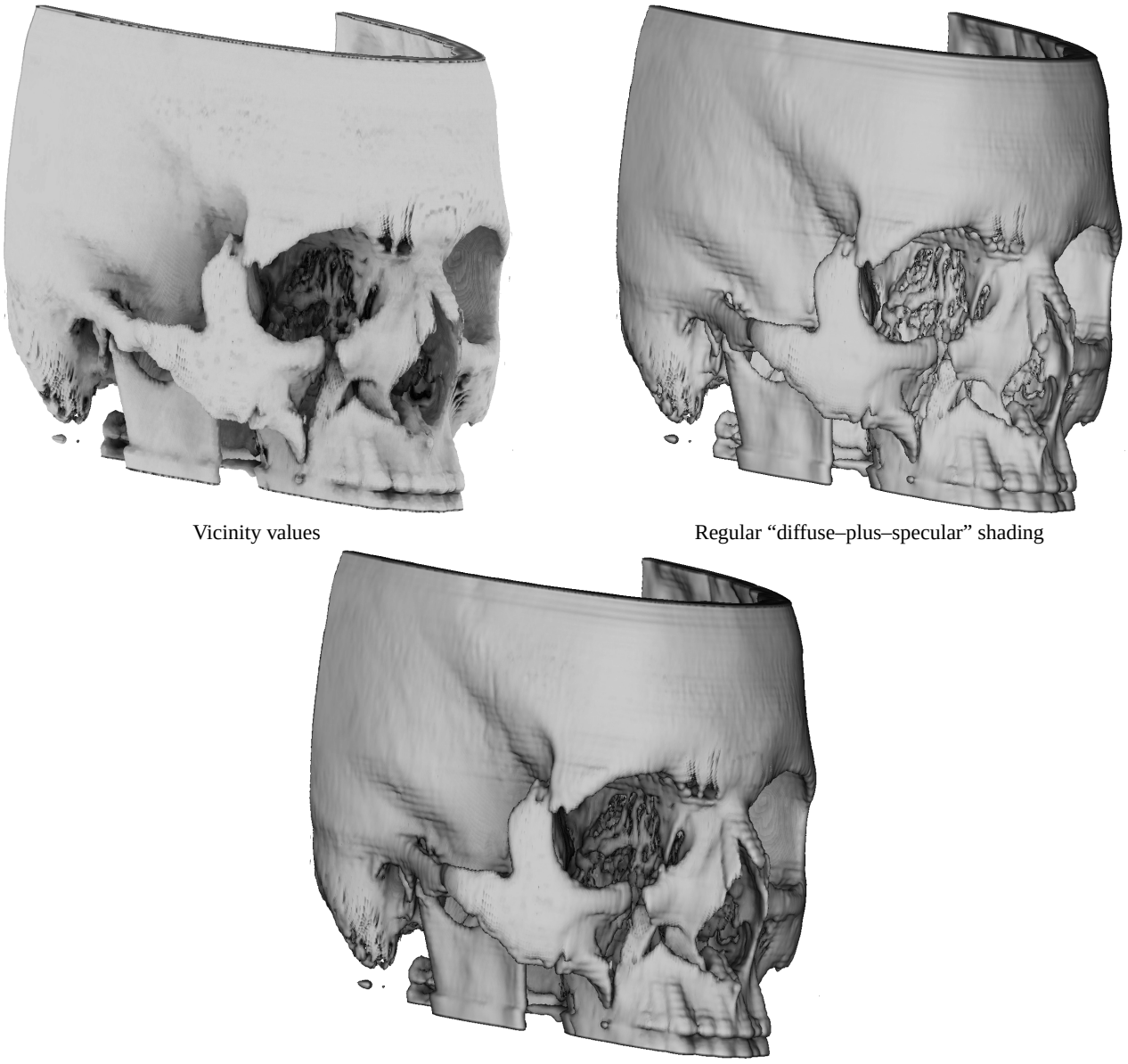


Figure 2: A human skull with fractures of the checkbone in three places. Dark areas in the upper–left image correspond to more occluded vicinities. The vicinity values are combined with regular “diffuse–plus–specular” shading by multiplying the two, resulting in the bottom, vicinity shaded image.

stead of the single ray used with a point light source.

This paper introduces a model of illumination for volumetric data, dubbed **vicinity shading**. In essence, we shade each surface point according to uniform diffuse lighting that is blocked only in the vicinity of the surface point. Vicinity shading provides better perceptual cues than the regular diffuse–plus–specular model. As shown in Figure 2, vicinity shading provides a good approximation of the shadowing that occurs in depressions and cavities.

This paper also describes a novel algorithm to compute vicinity shading for *all isosurfaces* in a volumetric data set. Thus, different isosurfaces can be rendered without requiring recomputation of the vicinity shading. Volumetric data poses some interesting problems and opportunities that are not present with other data representations, such as polygon meshes. These problems and opportunities are addressed by the algorithm.

2 Related Work

The original illumination model for volumetric data was described by Levoy [Levoy 1988], and includes components for scattering and absorption: Light from a point source is reflected off of each voxel, and is attenuated due to absorption along the path from the voxel to the viewer. No absorption occurs before the ray arrives at the voxel, so no shadows are present with this model. Levoy rendered the volume by tracing rays from the eye through the volume, accumulating light from sample points along each ray. Cabral [Cabral et al. 1994]

introduced the use of graphics texture mapping hardware to accelerate the process.

Shadows in volumetric data have been produced by "two–pass" methods [Behrens and Ratering 1998; Kajiya and Herzen 1984; Meinzer et al. 1991]. The first pass propagates point–source light through the volume, taking into account absorption, and stores in each voxel of a separate "light volume" the intensity of the arriving light. A second pass propagates light from the light volume toward the viewer, in the manner of Levoy or Cabral. Splatting has also been used to compute the light volume [Nulkar and Mueller 2001] with this two–pass approach. Other methods combine the two passes using volume slices that are aligned with neither the viewer nor the light source [Kniss et al. 2002a; Zhang and Crawfis 2002; Zhang and Crawfis 2003]. Another interesting approach [Kniss et al. 2002b; Kniss et al. 2003] adds a scattering component to the lighting model. A good survey of other illumination models for volume rendering was written by Max [Max 1995].

These methods for producing shadows in volumetric data all produce shadows from point light sources, and can't achieve the effect of uniform diffuse lighting that we're looking for. However, these methods are fast, typically requiring only one rendering of the data to produce the light volume, which can then be used without recomputation until the position of the light source changes. The vicinity shading described in this paper takes substantially longer (in the order of tens of minutes) to compute its own light volume but, once computed, the shadow information from that volume can be used without ever requiring recomputation.

Vicinity shading is the volumetric version of "obscurances," [Zhukov et al. 1998; Iones et al. 2003] which were used to achieve the same "cloudy day" illumination in polygonal scenes. The **obscurance** of a surface point is a measure of the empty space above the surface point, taking into account only the geometry (typically 10 to 100 polygons) within a certain distance of the surface point. More detail is provided in Section 3.

Vicinity shading applies the idea of obscurances to volumetric data, which poses challenges: For example, what characterizes those voxels that block light from arriving at a particular isosurface, and how is vicinity shading computed efficiently for all possible isosurfaces in the volume? But volumetric data also provides opportunities, such as the potential to exploit the regular and space–coherent structure of the data. These points will be discussed in Section 3.3.

Vicinity shading and obscurances are closely related to accessibility shading. The "tangent accessibility" of a surface point is equal to the radius of the largest sphere that can touch that point without intersecting any other part of the surface. This notion of accessibility was originally introduced [Lee and Richards 1971] in molecular modelling to determine what parts of one molecule are accessible to (the spherical atoms of) another molecule, in order to gain insight into chemical reactions between the molecules.

The accessibility measure (i.e. the sphere radius) can be used to modulate the intensity of a surface point, producing very appealing images of polished surfaces in which dark polish has accumulated in small, inaccessible cracks and pits [Miller 1994]. Efficient algorithms to compute accessibility have exploited graphics hardware [Spitz and Requicha 2000] and geometric techniques.

Accessibility gives a coarse approximation of the intensity of light arriving at a surface point, but fails in some cases. For example, the interior of a long tube will have uniform accessibility along its entire length, whereas the shading of the tube's interior should be darkest in the middle of the tube (see Figure 3). As another example, a surface point with a tiny occlusion directly above it will have low accessibility (and hence, dark shading), whereas the tiny occlusion should have almost no effect on the shading of the surface point.

Vicinity shading overcomes these problems by sampling in directions outward from the surface point to estimate the amount of unoccluded light arriving at the surface. However, since vicinity shading only estimates direct "primary" illumination, it will likely produce overly dark images for surfaces with high reflectivity, on which light can reflect multiple times to arrive inside folds and depressions. For such surfaces, a more accurate (and more computationally expensive) model would be used to account for multiple scattering [Rushmeier and Torrance 1987].

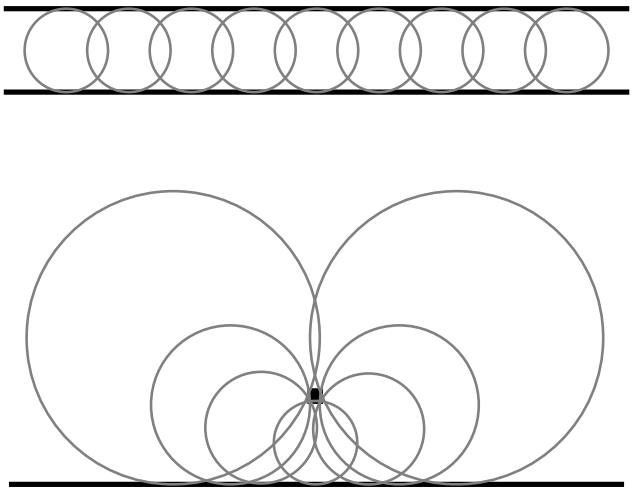


Figure 3: Accessibility shading provides incorrect depth cues. **Top:** A tube has uniform accessibility on its interior, as shown by the equally sized tangent spheres. However, we would expect darker shading toward the middle. **Bottom:** A surface with a tiny square occluder above it has dramatically reduced accessibility right below the occluder. However, we would expect only a very slight reduction in surface illumination under the occluder.

3 Vicinity Shading

We'll consider the vicinity shading at one surface point, p, leaving the question of what constitutes a surface to Section 3.3. As with the volume shading method of Levoy, we assume that light arrives at each point without attenuation due to the medium except (in our case) in the vicinity of the surface point. After being reflected from the surface, the light is attenuated in the standard way as it travels toward the viewer.

The goal is to compute for p a single coefficient between zero and one that represents the reduction of light at p due to occlusions in the vicinity. The light volume consisting of all of these coefficients will later be used while rendering, in the same manner as the two–pass shadow algorithms described above.

Let N be the surface normal at p, equal to the normalized gradient of the data field at p. A variant of the rendering equation [Kajiya 1986] gives the irradiance, E, arriving at p:

$$E(p) = \int_{\Omega} N \cdot L(\omega)\, d\omega$$

where $L(\omega)$ is the radiance arriving from direction ω (written as a vector with magnitude equal to the radiance), and Ω is the set of directions above the surface – those for which $N \cdot \omega > 0$.

This equation is evaluated by discretizing the domain Ω into k **sectors** of equal solid angle, $\Delta\omega$, and sampling $L(\omega)$ within each sector:

$$E(p) \approx \sum_{i=1}^{k} N \cdot L(\omega_i)\, \Delta\omega$$

Since our goal is to produce a coefficient in the range zero to one that represents the fraction of total possible irradiance arriving at p, we normalize to get $\widetilde{E}(p)$:

$$\begin{aligned}\widetilde{E}(p) &= \frac{\sum N \cdot L(\omega_i)\,\Delta\omega}{\sum N \cdot L^{max}(\omega_i)\,\Delta\omega} \\ &= \frac{\sum L_i \cos\theta_i}{\sum L_i^{max} \cos\theta_i} \qquad (1)\end{aligned}$$

where $\cos\theta_i = N \cdot \omega_i$, L_i is the computed scalar value of $L(\omega_i)$, and L_i^{max} is the maximum possible value of $L(\omega_i)$ (more on this last item below).

3.1 Discretizing the Domain Ω

The set of all directions around p can be represented by points on a Gaussian sphere centered at p. We can produce a fairly evenly distributed set of sample directions on the sphere with a commonly–used technique: Start with an icosahedron whose vertices lie on the unit sphere and subdivide each triangular face into four smaller triangles, whose new vertices are projected outward onto the unit sphere. Two or three iterations of this procedure produce a set of sample directions corresponding to vertices of the resulting polyhedron.

Each sample direction, ω_i, corresponds to one sector, $\Delta\omega_i$ of the discretized domain. At a particular point, p, on the object surface, we only consider the sectors, i, that are outward facing: $N \cdot \omega_i > 0$.

3.2 Definitions of L_i and L_i^{max}

For each sector, i, we must determine the radiance, L_i. Since the light source is uniform and diffuse, the radiance arriving directly at p — if unobstructed and unreflected — is equal from all directions, making the computation of L_i purely geometric.

With vicinity shading, occlusions are detected only within the vicinity of p, where the **vicinity** is an outward facing hemisphere of radius r centered at p. In the volumetric data set, the vicinity contains a number of voxels, each of which has been classified as interior or exterior to the surface to which p belongs (more on this below). Any interior voxel will occlude the light arriving at p.

To determine how much light arrives at p from direction ω_i, we simply sample the voxels lying in that direction, up to a distance r from p. There are (at least) two ways to estimate L_i:

> If none of the voxels sampled in direction ω_i is interior, $L_i = 1$. Otherwise, $L_i = 0$. This **all–or–nothing method** only collects light that is unoccluded as it travels from the boundary of the vicinity to p. In this case, $L_i^{max} = 1$.

In some situations, such as inside of a long tube, the all–or–nothing method will result in too little illumination for reasonable perception of the surface. For such cases, it's useful to estimate L_i according to the *unobstructed distance from p in direction* ω_i. This **partial occlusion method** provides (reduced) illumination even in tight spaces, which the "all or nothing" method does not.

> With the partial occlusion method, L_i is set equal to the distance from p to the nearest interior voxel in direction ω_i. L_i^{max} is the distance to the farthest voxel in the vicinity in direction ω_i. Note that $L_i^{max} \le r$, but is usually not equal to r.

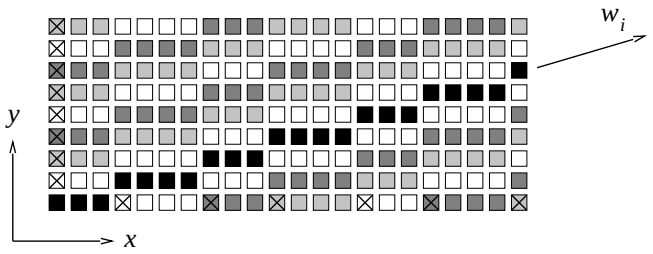


Figure 5: A 2D discrete line (shown as solid black pixels) in direction ω_i with dominant direction x. Integer translations of the discrete line (shown as shades of gray) in the non–dominant y direction are sufficient to completely cover the area such that each pixel lies on exactly one of the translated discrete lines. Each X marks the start of a line enumerated by the algorithm of Section 3.3.1.

3.3 Computation of L_i and L_i^{max}

With vicinity shading, there's no notion of transparency: light that is directed toward a voxel, v, is either blocked, or is not. If v lies on a "surface" in the volume and has density $\rho(v)$, it is reasonable to assume that the rest of the same surface has density greater than or equal to $\rho(v)$. Thus, we will assume the following:

> A voxel v of density $\rho(v)$ has its incoming light blocked by any other voxel of density greater than or equal to $\rho(v)$.

To compute v's vicinity shading, $\widetilde{E}(v)$, we evaluate Equation 1 by sampling in directions ω_i above v's surface. A sample in direction ω_i is blocked when it arrives at a voxel of density greater than or equal to $\rho(v)$, and L_i and L_i^{max} are determined by the partial occlusion method.

One approach to computing $\widetilde{E}$ for all voxels would be to iterate over all voxels and, for each voxel, to iterate over all sample directions. This would be prohibitively expensive: For n voxels, s sample directions, and a vicinity radius of r, the running time would be in $\mathcal{O}(n\,s\,r)$.

Instead, this section will describe an algorithm to compute $\widetilde{E}$ for all voxels in time $\mathcal{O}(n\,s)$, which removes any dependence on the vicinity radius, r. In fact, the vicinity can be arbitrarily large without affecting the running time.

The new algorithm exploits the following observation: *For a fixed line through the volume in direction* ω_i*, a voxel on that line can be blocked in direction* ω_i **only** *by another voxel on that line.*

Suppose that we can cover the volume with a set of parallel lines in direction ω_i, such that each voxel appears on exactly one line. Then each such line can be treated independently to compute L_i for all voxels on that line. This process can be repeated for each direction, ω_i, and the results accumulated according to Equation 1.

3.3.1 Covering the Volume with Disjoint Discrete Lines

The **dominant direction** of a line is the axis (x, y, or z) to which it is most parallel. In the context of a particular volume of voxels, the **discrete line** in direction ω_i is the (finite) set of voxels enumerated by a 3D Bresenham algorithm that starts from one corner of the volume and moves in direction ω_i until it exits the volume. The corner is chosen to maximize the number of enumerated voxels.

For a particular direction, ω_i, assume without loss of generality that x is the dominant direction. Consider a translation of the discrete line an integer number of voxels in the non–dominant y direction, followed by an integer number of voxels in the non–dominant z direction. (See Figure 5, in which a 2D example is given: The

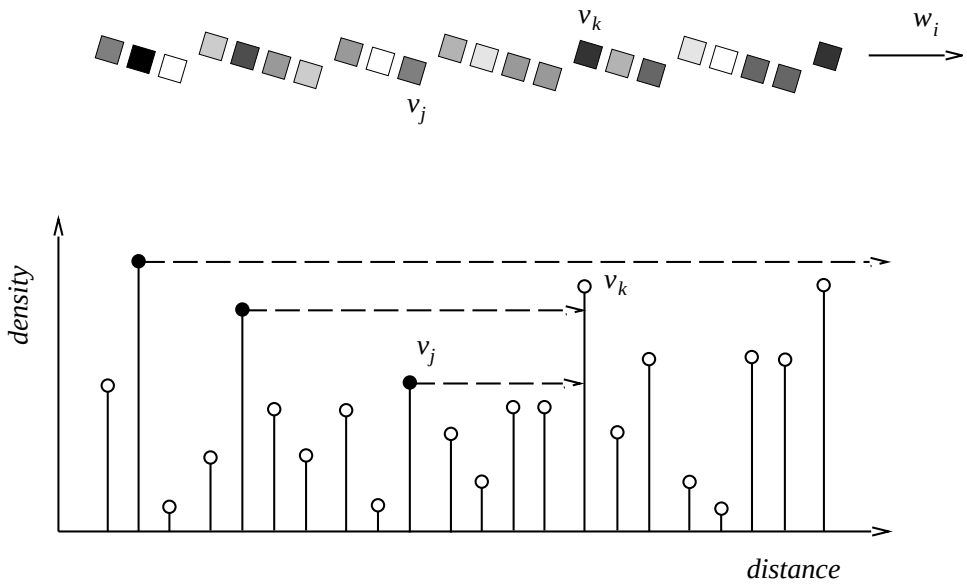


Figure 4: A discrete line and the corresponding plot of density versus distance. The discrete line has been rotated so that the plot point corresponding to a voxel appears directly below the voxel. The voxel v_j is blocked in the ω_i direction by voxel v_k since (in the plot) v_k is the closest point to the right of v_j that is above v_j.

dominant direction is x, and the discrete line is translated integer amounts in the non–dominant y direction.) Two observations can be made:

1. Each voxel lies on exactly one translated discrete line.
2. No two translated discrete lines intersect (unless the translations are identical).

This means that we can "cover" the volume with a set of translated discrete lines in direction ω_i, such that each voxel appears on *exactly one* translated discrete line, as was required in the previous section.

The following algorithm enumerates the translated discrete lines.[1] For simplicity, we've assumed that the dominant direction is x and that the discrete line starts at the $(0,0,0)$ voxel.

1. Each voxel $(0,y,z)$ on the $x = 0$ face of the volume is the beginning of a translated discrete line.
2. Use the 3D Bresenham algorithm to enumerate voxels on the discrete line that starts at $(0,0,0)$. Let (x_j,y_j,z_j) be the j^{th} voxel on the discrete line. Note that $x_j = x_{j-1} + 1$ since x is the dominant direction.
   (a) If $y_j \neq y_{j-1}$, then each voxel $(x_j,0,z)$ on the $y = 0$ face of the volume is the beginning of a translated discrete line.
   (b) If $z_j \neq z_{j-1}$, then each voxel $(x_j,y,0)$ on the $z = 0$ face of the volume is the beginning of a translated discrete line.

For new lines that are enumerated in Steps 2a and 2b above, the state of the 3D Bresenham algorithm at the first voxel of the new line must be identical to the state of the 3D Bresenham algorithm of Step 2 at the time that the new line is enumerated. This ensures that the new line is not translated in the dominant x direction. (If it were translated in that direction, it would intersect with another line, violating our requirement that each voxel lie on exactly one line.)

[1] This program code (and the other code of the algorithm) is available by email request to the author.

3.3.2 Computing L_i for All Voxels on One Discrete Line

The preceding section described how to allocate voxels to discrete lines such that any voxel on a line can be blocked in direction ω_i only by another voxel on that line. This section will describe how to determine the blocking voxels and, hence, how to compute L_i for each voxel on a line.

Let the voxels on a line be $v_1, v_2, v_3, \ldots$ with the index increasing in the ω_i direction. Voxel v_j has density $\rho(v_j)$ and Euclidean distance $d(v_j)$ from the first voxel on the line. Figure 4 shows an example of a line and the corresponding plot of density versus distance.

A voxel v_j is blocked by a voxel v_k in direction ω_i if $j < k$ and $\rho(v_j) \leq \rho(v_k)$. In the plot of Figure 4, the blocking v_k is found by drawing a horizontal line rightward from $(d(v_j), \rho(v_j))$ until it arrives at a higher (i.e. denser) point.

For each voxel on the line, our algorithm must compute its corresponding blocking voxel. The algorithm maintains a stack of as–yet "unresolved" voxels, for which no blocking voxel has yet been found. The top of the stack stores the unresolved voxel of minimum density, and density increases down the stack. Initially the stack is empty.

The algorithm processes the voxels by order of increasing distance, d. Upon processing the next voxel, v, two steps are taken:

1. If the voxel, u, at the top of the stack has $\rho(u) \leq \rho(v)$, then v is the closest blocker to u. In this case, pop u from the stack and (only if $N(u) \cdot \omega_i > 0$) compute $L_i(u)$ based upon the distance $d(v) - d(u)$ between u and v. Repeat Step 1 until the stack is empty or the topmost voxel has a greater density than that of v.

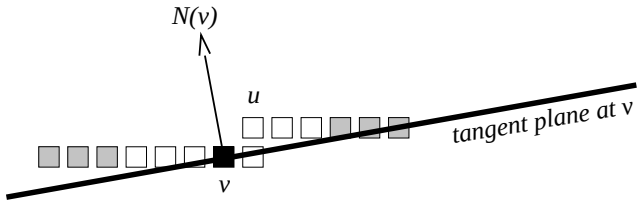


Figure 6: A gently sloping surface with a step will result in overly dark vicinity shading of the voxel, *v*, on the lower side of the step due to occlusion by the voxel, *u*, on the higher side of the step. The solution is prevent blocking by any voxel that is (a) close to *v* and (b) close to the tangent plane of *v*. The white voxels meet these criteria.

2. Push *v* onto the top of the stack, since *v* is now the minimum–density unresolved voxel.

After all voxels on the line have been processed, any unresolved voxels remaining on the stack are unoccluded in direction ω_i. For each unoccluded voxel, set $L_i = L_i^{max}$.

For a better intuition, refer to Figure 4: Suppose we are just about to process v_k, and the stack contains the voxels shown as solid circles to the left of v_k. When v_k is processed, the first two such voxels are removed from the stack, since v_k blocks them (as shown by the dashed horizontal lines). Then v_k is added to the stack.

The algorithm can be modified to compute blocking pixels in *both* directions (ω_i and $-\omega_i$) in the same pass along the line. Note that *v*'s closest occluder in the $-\omega_i$ direction is the topmost voxel on the stack after Step 1 is completed. A Step 1.5 could be added to take advantage of this.

A problem arises due to the voxelization of volumetric data. As shown in the 2D example of Figure 6, a flat surface that is oriented close to one of the volume axes will have a number of steps. The voxel on the lower side of a step will be very much occluded by the voxel on the upper side of the step, resulting in overly dark vicinity shading of the lower voxel (which will result in dark bands in the rendered volume). To avoid this, we apply a heuristic: A voxel *u* does *not* block a voxel *v* if (a) *u* is within three voxel widths of *v* and (b) *u* is within 1.5 voxel widths of the tangent plane at *v*.

To implement the heuristic, Step 1 of the algorithm must be modified: If the voxel removed from the stack in Step 1 satisfies conditions (a) and (b) above, that voxel is instead put onto a separate list of "postponed" voxels. A Step 0 is added in which the new voxel *v* is checked against every postponed voxel, to see whether it blocks a postponed voxel. If it does, the postponed voxel is removed from the list and its L_i is calculated with *v* as the blocker.

The first enhancement (that of processing the direction $-\omega_i$) and the second (that of postponing some voxels) are incompatible; only one can be implemented in the algorithm. Furthermore, if we postpone some voxels, the running time of $\mathcal{O}(n\,s)$ is no longer valid, since we might, in theory, inspect a large number of postponed voxels with each new voxel on the line. In practice, however, postponing voxels was found to have very little effect on the running time.

3.4 Using $\widetilde{E}(p)$ in Texture–Based Volume Rendering

The previous sections described how to compute L_i for each voxel, given a particular direction ω_i. We apply that procedure to all directions and compute the vicinity shading, $\widetilde{E}$, of each voxel according to Equation 1.

The result of computing $\widetilde{E}$ is a "light volume" in which each voxel stores a value between zero and one, which is an estimate of the fraction of total possible illumination arriving at the voxel.

In texture–based volume rendering, each voxel in a 3D texture volume stores an *RGBA* 4–tuple, where the *RGB* are the three components of the normalized voxel gradient, and *A* is the voxel density. The value for *A* is used to index into a transfer function, which provides a colour and opacity for the voxel.

The simplest way to incorporate the light volume in the illumination calculation is to use it to modulate the *RGB* gradient lengths: Each normalized gradient $N(p)$ is replaced by $\widetilde{E}(p) \times N(p)$. The intensity of light leaving *p* is then computed as

$$\begin{aligned} I &= k_d\,(\widetilde{E}N)\cdot L + k_s\,((\widetilde{E}N)\cdot H)^n \\ &= \widetilde{E}\,k_d\,N\cdot L + \widetilde{E}^n\,k_s\,(N\cdot H)^n \end{aligned}$$

with L the direction to the point light source, H the half–angle vector between L and the direction to the viewer, and k_d and k_s the diffuse and specular lighting coefficients.

This is not a physically correct model, because the uniform diffuse lighting should be independent of the point source lighting. This approach also tends to remove specular highlights, due to the specular exponent on $\widetilde{E}$. But it is simple and does produce better perceptual cues than without vicinity shading. Its principal advantage is that it is embarrassingly easy to implement. The images in this paper were rendered with this method.

It would be better to separate the contribution of the uniform diffuse lighting from that of the point source lighting. To do so, we can encode the normalized voxel gradient in *two* components, *RG*, and encode the light volume in the third component, *B*. The graphics hardware can then use *RG* as indices into a 2D texture containing 3D gradients, and can include $\widetilde{E}$ (from *B*) as a separate term in the calculation:

$$I = k_e\,\widetilde{E} + k_d\,N\cdot L + k_s\,(N\cdot H)^n$$

4 Experimental Results

Vicinity shading was applied to a number of volumetric data sets generated from CT scans, and to one data set generated by voxelizing a polygonized model. Two of these sets are shown in this paper. With each set, the vicinity radius was set to 100 voxel widths, but could have been set to any value without affecting the execution times. The images of this paper used vicinity shading that was computed from 1272 sample directions.

Figure 2 shows a skull with a fractured cheekbone. Vicinity shading provides good cues to the positions of the three fractures, and helps to distinguish the interior of the skull as seen through the eye orbit. It also brings out several small features which are not obvious with the regular shading method.

Figure 7 shows a cerebral cortex, which was converted to voxel data from a polygonized model. The regular diffuse–plus–specular shading produces an artificially bright image, in which deep folds are as brightly illuminated as the outer surface (except where the light direction is almost tangent to the surface of the fold). This is corrected with vicinity shading, yielding a more realistic image with better depth cues.

Table 1 summarizes the results on a 1.8 GHz Pentium PC with 1.0 GB of memory. The running time of the vicinity computation is approximately proportional to the total number of voxels and to the number of sample directions. The running time is completely independent of the vicinity radius, *r*.

Table 1: Data Sets and Execution Times

Data	Dimensions	Number of Directions	Execution Time (minutes)
Skull	$256 \times 256 \times 203$	312	14.3
Skull	$256 \times 256 \times 203$	1272	59.0
Cortex	$128 \times 512 \times 256$	1272	57.5

In a $256 \times 256 \times 256$ volume, each sampling direction requires about 2.7 seconds of computation. Despite the "efficient" algorithm, this is still a huge amount of time, and is an obvious area for future work. (Of course, we can always trade off time for accuracy by reducing the number of sampling directions.)

5 Discussion

Vicinity shading provides perceptual cues to relative surface depth. It does so by estimating the illumination that the surface would receive under uniform diffuse lighting. Experiments by other authors [Langer and Bülthoff 1997] have shown that — for at least some surfaces — perception of shape is more accurate under such uniform, diffuse lighting than it is under point lighting.

Vicinity shading samples for occluders in a vicinity around each surface point. An algorithm was described which efficiently determines the vicinity shading of *all* voxels by exploiting the coherence of voxels along a number of discretely sampled directions through the volume.

An important feature of the algorithm is that it computes vicinity shading for *all possible isosurfaces* simultaneously, and not just for the one surface shown in the images of this paper. Thus, the transfer function can (to some degree) be manipulated without requiring recomputation of the vicinity shading.

At least two avenues of future work appear interesting. Firstly, it should be possible to accelerate the algorithm by using the functionality of graphics hardware. One potential problem, however, is that the current algorithm uses a variable–size stack, which does not have an obvious graphics card hardware implementation.

Secondly, a user study should be performed to verify that perception of shape is, indeed, superior with vicinity shading. The results of Langer and Bülkhoff only *suggest* that this is the case for vicinity shading, since vicinity shading only provides an estimate of the overall illumination, and is applied to a different type of surface than in the Langer and Bülkhoff study. Such an experiment might, for example, test users on their ability to distinguish peaks and valleys in images with and without vicinity shading.

6 Acknowledgements

The author wishes to thank Paul Bourke [Bourke 1997] for the cerebral cortex data, and to thank the Department of Radiology at the University of Iowa for their publically available data sets of the foot and skull. The thoughtful comments of the anonymous reviewers were also appreciated. This work is supported in part by a grant from Communications and Information Technology Ontario.

References

BEHRENS, U., AND RATERING, R. 1998. Adding shadows to a texture-based volume renderer. In *IEEE Symposium on Volume Visualization*, 39–46.

BLINN, J. F. 1977. Models of light reflection for computer synthesized pictures. *Computer Graphics (SIGGRAPH) 11*, 2, 192–198.

BOURKE, P. 1997. Modelling the surface of the human cortex. In *online notes, http://astronomy.swin.edu.au/ pbourke/modelling/cortex*.

CABRAL, B., CAM, N., AND FORAN, J. 1994. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *IEEE Symposium on Volume Visualization*, 91–98.

COOK, R., PORTER, T., AND CARPENTER, L. 1984. Distributed ray tracing. *Computer Graphics (SIGGRAPH) 18*, 3, 137–145.

IONES, A., KRUPKIN, A., SBERT, M., AND ZHUKOV, S. 2003. Fast, realistic lighting for video games. *IEEE Computer Graphics and Applications 23*, 3, 54–64.

KAJIYA, J., AND HERZEN, B. V. 1984. Ray tracing volume densities. *Computer Graphics (SIGGRAPH) 18*, 3, 165–174.

KAJIYA, J. 1986. The rendering equation. *Computer Graphics (SIGGRAPH) 20*, 4, 143–150.

KNISS, J., KINDLMANN, G., AND HANSEN, C. D. 2002. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics 8*, 4, 270–285.

KNISS, J., PREMOZE, S., HANSEN, C. D., AND EBERT, D. S. 2002. Interactive translucent volume rendering and procedural modeling. In *IEEE Visualization Conference*, 255–262.

KNISS, J., PREMOZE, S., HANSEN, C. D., SHIRLEY, P., AND MCPHERSON, A. 2003. A model for volume lighting and modeling. *IEEE Transactions on Visualization and Computer Graphics 9*, 2, 150–162.

LANGER, M. S., AND BÜLTHOFF, H. H. 1997. Do humans perceive shape from shading better on sunny days or on cloudy days? Tech. Rep. 97–130, NEC Research Institute.

LEE, B. K., AND RICHARDS, F. M. 1971. The interpretation of protein structures: Estimation of static accessibility. *Journal of Molecular Biology 55*, 379–400.

LEVOY, M. 1988. Display of surfaces from volume data. *IEEE Computer Graphics and Applications 8*, 3, 29–37.

MAX, N. 1995. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics 1*, 2.

MEINZER, H.-P., MEETZ, K., SCHEPPELMANN, D., ENGELMANN, U., AND BAUR, H. J. 1991. The heidelberg ray tracing model. *IEEE Computer Graphics and Applications 11*, 6, 34–43.

MILLER, G. 1994. Efficient algorithms for local and global accessibility. *Computer Graphics (SIGGRAPH)*, 319–326.

NULKAR, M., AND MUELLER, K. 2001. Splatting with shadows,. In *International Workshop on Volume Graphics 2001*, 35–50.

PHONG, B.-T. 1975. Illumination for computer generated images. *Communications of the ACM 18*, 6, 311–317.

RUSHMEIER, H., AND TORRANCE, K. 1987. The zonal method for calculating light intensities in the presence of a participating medium. *Computer Graphics (SIGGRAPH) 21*, 293–302.

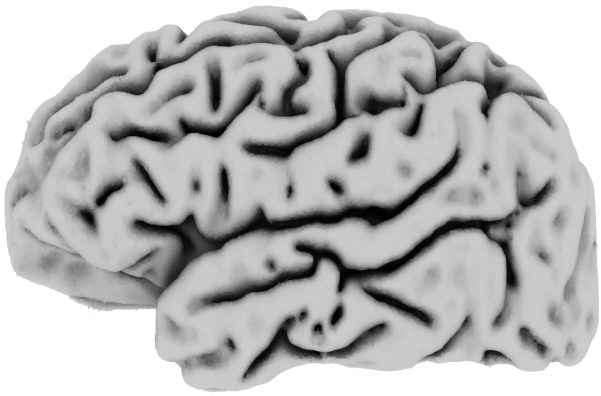
SPITZ, S. N., AND REQUICHA, A. A. G. 2000. Accessibility analysis using computer graphics hardware. *IEEE Transactions on Visualization and Computer Graphics 6*, 3, 208–219.

WHITTED, T. 1980. An improved illumination model for shaded display. *Communications of the ACM 23*, 6, 343–349.

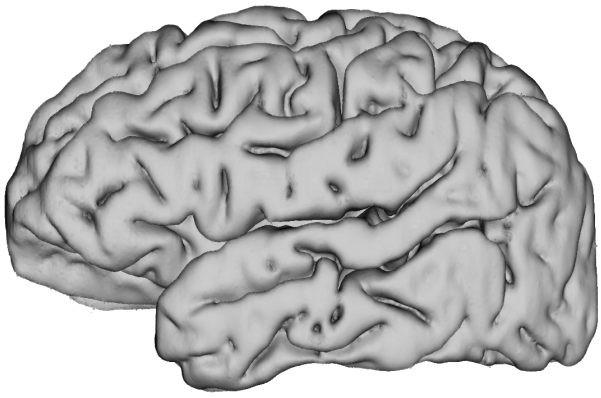
ZHANG, C., AND CRAWFIS, R. 2002. Volumetric shadows using splatting. In *IEEE Visualization Conference*, 85–92.

ZHANG, C., AND CRAWFIS, R. 2003. Shadows and soft shadows with participating media using splatting. *IEEE Transactions on Visualization and Computer Graphics 9*, 2, 139–149.

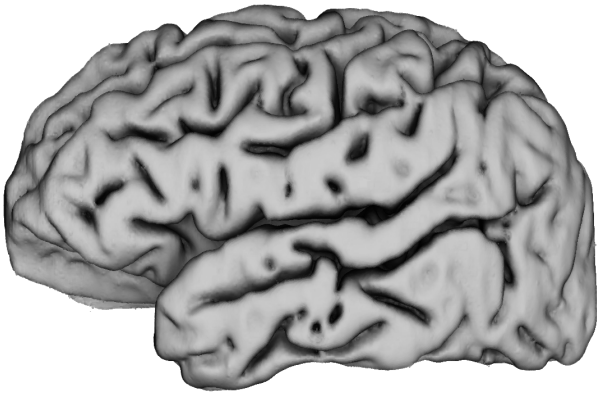
ZHUKOV, S., IONES, A., AND KRONIN, G. 1998. An ambient light illumination model. In *Eurographics Workshop on Rendering*, 45–56.



Vicinity values



Regular “diffuse–plus–specular” shading



Vicinity shading

Figure 7: Vicinity shading of a cerebral cortex using 1272 sample directions.