To appear in the proceedings of the *Visualization'99* conference

Splatting Without The Blur

Klaus Mueller, Torsten Möller, and Roger Crawfis

Department of Computer and Information Science, The Ohio State University, Columbus, OH

ABSTRACT

Splatting is a volume rendering algorithm that combines efficient volume projection with a sparse data representation: Only voxels that have values inside the iso-range need to be considered, and these voxels can be projected via efficient rasterization schemes. In splatting, each projected voxel is represented as a radially symmetric interpolation kernel, equivalent to a fuzzy ball. Projecting such a basis function leaves a fuzzy impression, called a footprint or splat, on the screen. Splatting traditionally classifies and shades the voxels prior to projection, and thus each voxel footprint is weighted by the assigned voxel color and opacity. Projecting these fuzzy color balls provides a uniform screen image for homogeneous object regions, but leads to a blurry appearance of object edges. The latter is clearly undesirable, especially when the view is zoomed on the object. In this work, we manipulate the rendering pipeline of splatting by performing the classification and shading process after the voxels have been projected onto the screen. In this way, volume contributions outside the iso-range never affect the image. Since shading requires gradients, we not only splat the density volume, using regular splats, but we also project the gradient volume, using gradient splats. However, alternative to gradient splats, we can also compute the gradients on the projection plane, using central differencing. This latter scheme cuts the number of footprint rasterization by a factor of four, since only the voxel densities have to be projected. Our new method renders objects with crisp edges and well-preserved surface detail. Added overhead is the calculation of the screen gradients and the per-pixel shading. Both of these operations, however, may be performed using fast techniques employing lookup tables.

1 INTRODUCTION

Volume visualization deals with the display of volumetric data, represented as sample points on a regular or irregular 3D raster. Volumetric data may be produced by medical scanners, such as MRI, CT, PET, or SPECT, by confocal or electron microscopy, by numerical methods, such as scientific simulations and finite element analysis, or by voxelization of analytic functions. In recent years, many tools and techniques have been proposed to aid us in the visualization of volumetric datasets. On one side is the group of direct volume renderers, which seek to capture a visual impression of the complete 3D dataset by accounting for the emission and absorption effects of all data elements [7][13]-[15][30]-[32][34]. On the other side is the group of indirect volume renderers that reduce the data into a set of isosurfaces [17], which are conveniently rendered as polygonal meshes using z-buffer algorithms. The latter representation is appropriate when such isosurfaces exist in the data, but may be less effective when the volume is a space-filling gas, such as in fluid-flow simulations, or is composed of many micro-surfaces, such as tissue in a medical dataset.

Address: 2015 Neil Ave, 395 Dreese Lab, Columbus, OH 43210,
{mueller, moeller, crawfis}@cis.ohio-state.edu,

A good argument for indirect volume renderers is that they can take advantage of widely available sophisticated graphics hardware to quickly render the polygonal meshes. However, a large volumetric dataset may give rise to a huge number of polygons, so huge that it may overwhelm the graphics engine. This observation recently motivated the parallel raycaster developed by Parker [25]. Although it is true that the magnitude of the polygonal mesh may be reduced to a more manageable size using the error-minimizing methods proposed by Hoppe [10] and others, these methods are rather expensive, and therefore cannot be applied when the polygonal mesh is not static. This is a scenario that occurs when viewing time-varying data or when the isosurface is interactively varied, e.g., during data exploration, which requires the extraction (and simplification) of a new polygonal mesh for each new iso-interval.

Popular direct volume rendering algorithms are raycasting [14][15], Shear-warp [13], splatting [30]-[32], cell-projection methods [34], and approaches using 3D texture-mapping hardware [4][28][29] or custom volume rendering boards [24]. All of these methods perform some sort of explicit or implicit volume interpolation at points along the viewing direction. The interpolation results are then composited in front-to-back or back-to-front order. A distinction has to be made with respect to the nature of the interpolated value, and this distinction depends on the order of the volume rendering pipeline constituents: classification, shading, interpolation, and compositing (see Fig. 1). Classification determines the (fuzzy) object or material membership of a voxel or interpolated sample point and is usually given by a range of volume densities, specified in the transfer function. Based on this classification transfer function, the voxel or interpolated sample point is assigned a color and an opacity. The color is then scaled by the result of the shading operation, which determines the amount of light, coming from one or more lightsources, that is reflected towards the eye. If the volume voxels are classified and shaded as a pre-processing step before the projection occurs, then the interpolation operations yield colors and opacities, which can be directly composited along the viewing direction (Fig. 1a). We will refer to this mode of volume rendering as *pre-shaded volume rendering*. On the other hand, when the raw density volume is interpolated, then each interpolated value must first be classified and shaded before it is composited (Fig. 1b). Since classification and shading occurs after sample interpolation, we will call this type of volume rendering *post-shaded volume rendering*.

Splatting traditionally uses the pre-shaded scheme: The voxels are first classified and shaded, and then each shaded voxel is projected to the screen as a fuzzy ball, i.e., the 3D interpolation kernel. The 2D screen projection of such a kernel is called a *splat* or *footprint* and can be performed very efficiently. The outcome of this rendering procedure is an object projection that has that typical pre-shaded volume rendering-look: blurred boundaries and smooth surfaces (see Fig. 1a where we have splatted a cube composed of 8^3 voxels). These blurred boundaries are due to the smooth decay of the iso-voxels' splatting kernels, located at the object edges. A positive aspect of the blurring effect is that the rendering results with splatting are almost always smooth and alias-free, even in the

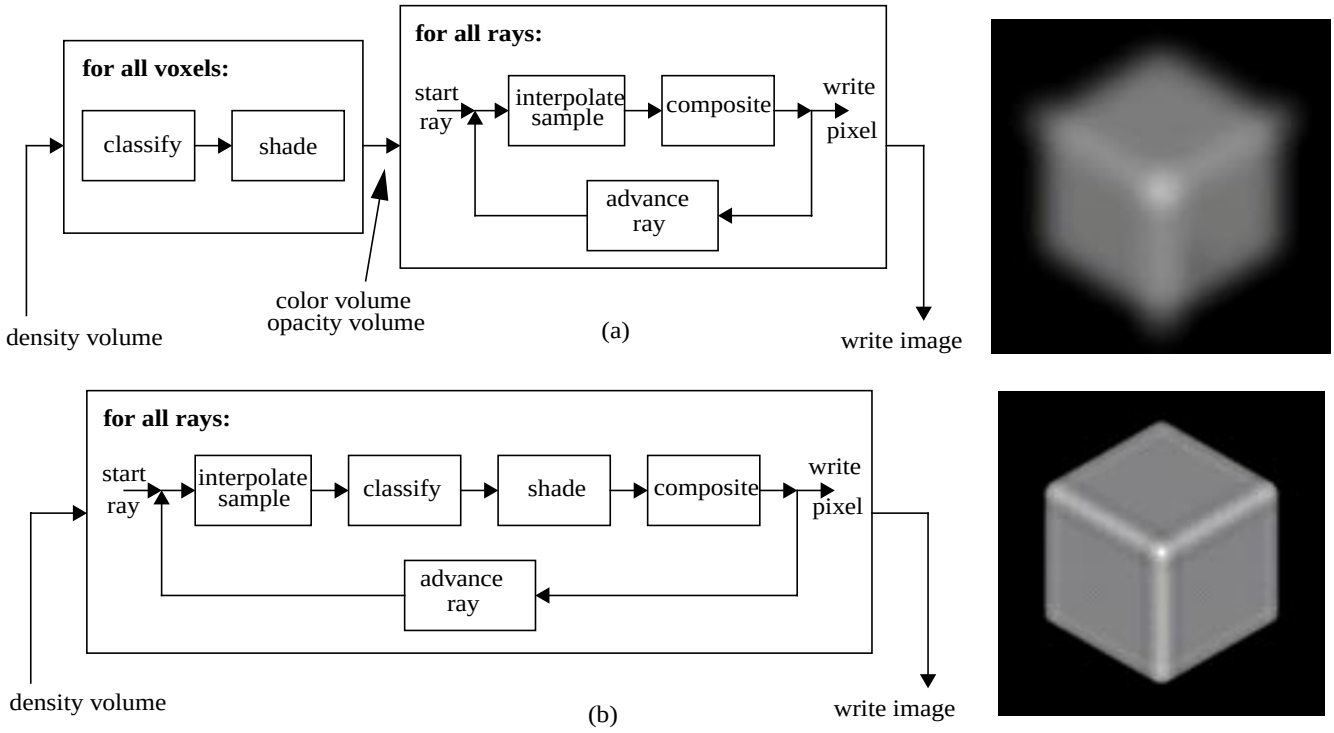


Fig. 1: Two standard volume rendering pipelines. (a) Pre-shaded volume rendering: The raw density volume is first classified and shaded, and the volume renderer then interpolates the resulting color and opacity volumes and composites the sample values for image generation. On the right: A cube (8^3 voxels) rendered with splatting from a pre-shaded color and opacity volume (a Gaussian filter of radial extent 2.0 was used for interpolation). (b) Post-shaded volume rendering: The volume renderer interpolates the raw density volume, and these sample values are then classified and shaded and composited for image generation. On the right: The cube of (a) is rendered from the density volume with a raycaster and trilinear interpolation in post-shaded mode.

presence of poor gradients around the iso-surfaces. However, the blurring becomes a great nuisance when rendering intricate and detailed structures, such as nasal passageways [33] or colons [9] in medical fly-throughs. The blurring (i.e., lowpassing) washes out much of the crucial object detail, which greatly hampers the realism and utility of the visualization. A direct volume rendering is, however, preferable, should we desire to deform or manipulate the object in a surgical simulation scenario.

Raycasting algorithms, on the other hand, come in both flavors, i.e., pre-shaded [7][14][15][35] and post-shaded [1][8][24][27]. (Note that we, for the sake of our discussion, refer to the raw, original grid values in a volume as densities.) See Fig. 1b, where we show the cube of Fig. 1a, now rendered with post-shaded raycasting. We observe that the edges appear very crisp. Although it is generally agreed upon in the literature [35] that the post-shaded pipeline produces sharper edges than the pre-shaded pipeline, one may argue, that even when raycasting is applied in conjunction with pre-shading, one rarely notices blurring as pronounced as with splatting. This can be explained by the fact that raycasting most commonly uses trilinear interpolation filters, which cause less blurring (but possibly more aliasing). Splatting, on the other hand, usually employs relatively large, radially symmetric interpolation kernels (such as a Gaussian of radial extent 2.0). These kernel are chosen since they allow the same kernel footprint to be used for all viewing directions and ensure good footprint overlap on the screen [31]. Unfortunately, these larger interpolation kernels cause a higher amount of blurring (but also reduce aliasing).

In previous work, Huang [11] has proposed edge splats, i.e., specialized kernels with rapid interpolation function decay, to be used on object boundaries. These kernels, in essence, model the look of an edge. Splatting with edge splats placed at iso-surfaces and regular splats used in homogeneous volume regions provides the desired effect: crisp edges and smooth surfaces. It, however, requires the detection of directional edges in volume space as a pre-processing step. The gradient at an iso-surface provides the edge direction, while the voxel-edge distance is estimated by the gradient magnitude/direction and the difference between the voxel's value and the iso-value. In the pre-processing step, all voxels that are close to an iso-surface are augmented with values for edge direction, edge distance, and gradient-strength. The splatting process then uses these parameters at rendering time to select the edge splat with the most appropriate edge decay, location, and 3D orientation for each voxel. Although the approach is very effective for the iso-surface edges, micro-edges within the iso-range are not handled easily. Other drawbacks are: (i) The method has problems with discontinuities in the edge profile, e.g., sharp corners such as cube edges, where the direction of an edge is ambiguous in volume space and its perception is view-dependent. (ii) It requires a pre-processing step (which, however, may be performed on the fly) whenever the iso-range or transfer function is changed. This slows interactive transfer function changes. (iii) The edge splats are equivalent to applying a local high frequency filter to the volume, hence noise in the iso-contour range may be amplified.

The splatting approach presented in this paper does not add any artificial detail to the volume. Rather, it eliminates blurring by applying the volume rendering pipeline of Fig. 1b to the original density data using the splatting paradigm. In this way, only that part of a projected voxel footprint that falls within the iso-range is shaded and displayed, while the footprint portion with values outside the iso-range (or the interval volume) is culled from the shad-

ing process and never contributes to the final image. It is this latter portion that is responsible for the blurring in the pipeline of Fig. 1a.

The outline of the paper is as follows. First, in Section 2, we describe splatting in general and then summarize a splatting method that we have proposed recently and in which we will embed our new splatting pipeline. Then, in Section 3, we provide the underlying theory of the new enhancement. Section 4 presents the new algorithm, and Section 5 presents visual and temporal results. Finally, Section 6 provides some concluding remarks, and Section 7 finishes with a list of research topics for the future.

2 PRELIMINARIES

The basic element in most volume rendering applications is the volume rendering integral in its low-albedo form [3], as formulated by Kajiya and Von Herzen [12] and also formally derived by Max [19]. For each pixel ray, we compute $I_\lambda(\boldsymbol{x},\boldsymbol{r})$, the amount of light of wavelength λ coming from ray direction $\boldsymbol{r}$ that is received at point $\boldsymbol{x}$ on the image plane:

$$I_\lambda(\boldsymbol{x},\boldsymbol{r}) = \int_0^L C_\lambda(s)\mu(s)\exp\left(-\int_0^s \mu(t)dt\right)ds \tag{1}$$

Here L is the length of ray $\boldsymbol{r}$. We can think of the volume as being composed of particles with certain densities μ (Max calls them light extinction coefficients [19]). These particles receive light from all surrounding light sources and reflect this light towards the observer according to the specular and diffuse material properties of the particles. Thus, in (1), C_λ is the light of wavelength λ reflected at location s in the direction of $\boldsymbol{r}$. To account for the higher reflectivity of particles with larger densities, we must weight the reflected color by the particle density. The light scattered at s is then attenuated by the densities of the particles between s and the eye according to the exponential attenuation function.

The analytic volume rendering integral can, in most cases, not be computed efficiently, if at all, and therefore a variety of approximations are in use. An approximation of (1) can be obtained by raycasting, using a discrete Riemann sum. Rays are cast into the volume, and samples, spaced apart by a distance Δs, are interpolated along the ray by means of a 3D interpolation kernel h. Equation (1) can be written as a Riemann sum equation as follows:

$$I_\lambda(\boldsymbol{x},\boldsymbol{r}) = \sum_{i=0}^{L/\Delta s-1} C_\lambda(i\Delta s)\mu(i\Delta s)\Delta s \prod_{j=0}^{i-1} \exp(-\mu(j\Delta s)\Delta s) \tag{2}$$

A few approximations make the computation of this equation more efficient. First, the transparency $t(i\Delta s)$ is defined as $\exp(-\mu(i\Delta s)\Delta s) = t(i\Delta s)$. Transparency assumes values in the range [0.0, 1.0]. Then opacity $\alpha(i\Delta s) = (1 - t(i\Delta s))$. The exponential term in (2) can be approximated by the first two terms of its Taylor expansion: $t(i\Delta s) = \exp(-\mu(i\Delta s)\Delta s) \approx 1 - \mu(i\Delta s)\Delta s$. Then one can write: $\mu(i\Delta s)\Delta s \approx 1 - t(i\Delta s) = \alpha(i\Delta s)$.

This transforms (2) into the familiar compositing equation:

$$I_\lambda(\boldsymbol{x},\boldsymbol{r}) = \sum_{i=0}^{L/\Delta s-1} C_\lambda(i\Delta s)\alpha(i\Delta s)\cdot \prod_{j=0}^{i-1} (1-\alpha(j\Delta s)) \tag{3}$$

This equation, popularized by Levoy [14], is valid if we interpolate a discrete, pre-classified and pre-shaded color and opacity volume $C_d(x,y,z)$ and $\alpha_d(x,y,z)$, respectively, to get $C(i\Delta s)$ and $\alpha(i\Delta s)$ (pipeline of Fig. 1a). However, if we classify interpolated samples obtained from the discrete raw density volume $f_d(x,y,z)$ (pipeline of Fig. 1b), then (3) is more appropriately written as follows:

$$I_\lambda(\boldsymbol{x},\boldsymbol{r}) = \sum_{i=0}^{L/\Delta s-1} C_\lambda(f(i\Delta s), g(i\Delta s))\alpha(f(i\Delta s)) \prod_{j=0}^{i-1} (1-\alpha(f(j\Delta s))) \tag{4}$$

The function value $f(i\Delta s)$ and the gradient vector $g(i\Delta s)$ is interpolated from $f_d(x,y,z)$ using some 3D interpolation kernel, and C_λ and α are now functions that translate the interpolated volume function values into color and opacity.

The splatting algorithm [30]-[32] was proposed by Lee Westover to provide the level of efficiency that previous raycasting approaches were lacking. It gains its speed by reordering the volume rendering integral so that each voxel's contribution to the integral can be viewed isolated from the other voxels. In splatting, an interpolation kernel h is placed at each voxel location. This enables one to view the volume as a field of overlapping interpolation kernels which, as an ensemble, make up the continuous object representation. The contribution of a voxel j with value f_j is then given by $f_j \cdot \int h(s)ds$, where s follows the line of kernel integration in the direction of the ray. If the interpolation kernel is radially symmetric, we may pre-integrate $\int h(s)ds$ into a lookup-table, i.e., the kernel footprint, and use this table for all voxels. We can then map the voxel footprints, scaled by the voxel values, to the screen where they accumulate into the projection image [31]. By mapping the footprint (image) onto a polygon, we can employ standard 2D texture mapping hardware for the projection process [6]. However, the footprint interpolation is also easily done in software with fast DDA procedures [18][21]. Thus, splatting performs all interpolations in 2D (i.e., the footprint rasterizations), while raycasting performs all interpolations more expensively in 3D (i.e., when interpolating a volume sample). In addition, as an object-order approach, splatting only needs to store and render the relevant voxels, which in many cases constitutes a mere 10-20% of the volume voxels [34]. However, until recently, one of the downsides of splatting was that the pre-integrated kernels restricted the volume rendering integral, since the 3D reconstruction kernel was composited as a whole, and not piecewise, as part of an interpolated sample along a viewing ray. This caused popping artifacts in animated viewing.

The image-aligned splatting approach was recently proposed [23] to eliminate these restrictions. It unifies the qualitative advantages of raycasting with the efficiency of splatting. Unlike the traditional splatting approach, image-aligned splatting does not splat the interpolation kernels as a whole. Rather, it slices the interpolation kernels by a series of cutting planes that are aligned parallel to the image plane. The kernel sections that fall within a pair of cutting planes or thin slab are summed into a sheet-buffer, and consecutive sheet-buffers are composited from back-to-front or front-to-back. Again, pre-integrated kernel sections are used for fast rasterization. This approach gives rise to a volume integral calculation that is similar to the one computed by raycasting in the color-interpolation model. It mimics a set of simultaneous rays that re-sample the volume into a set of parallel sheet images, spaced apart by Δs, which are then composited in front-to-back order. The distinction is that the composited colors and opacities are now obtained by true function integration, and not by the Riemann square integration rule. Equation (3) is then written as follows:

$$I_\lambda(\boldsymbol{x},\boldsymbol{r}) = \sum_{i=0}^{L/\Delta s-1} \frac{1}{\Delta s} \int_{i\Delta s}^{(i+1)\Delta s} C_\lambda(s)ds \int_{i\Delta s}^{(i+1)\Delta s} \alpha(s)ds \prod_{j=0}^{i-1} \left(1 - \int_{j\Delta s}^{(j+1)\Delta s} \alpha(s)ds\right) \tag{5}$$

We will use the image-aligned splatting algorithm along with the early splat elimination scheme proposed in [22]. This scheme

employs a progressively refined screen occlusion map to cull from the splatting pipeline those voxels that have no effect on the final image. The screen occlusion map is computed from the opacity image layer, each time a sheet image has been composited with the current image buffer. The volume is traversed in front-to-back order. To determine occlusion, the center of a voxel (splat) is projected, and the splat is only rasterized if the occlusion map value at the projected point is above a pre-set opacity threshold. The splat culling can lead to tremendous savings, especially when the transfer function specifies large opaque regions. In many cases, only 10% of the relevant voxels survive the visibility test. But our tests also indicate that even semi-transparent volumes benefit considerably. The scheme is similar to early ray termination acceleration for raycasting, however, it does require the calculation of the occlusion map for every sheet-buffer slice, and it also requires the projection of each voxel's center.

3 THEORETICAL ASPECTS

In this section, we discuss the theoretical difference between pre-shaded and post-shaded volume rendering, and we derive the gradient filter necessary to implement the latter for splatting.

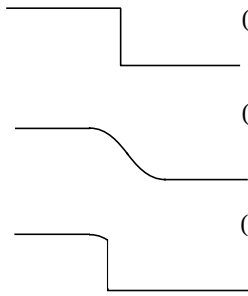
3.1 Pre-shaded vs. post-shaded volume rendering

We have seen in Fig. 1 that the volume rendering pipeline has four components: classification, shading, interpolation, and compositing. We have also seen that two permutations of these pipeline components exist, depending on if we interpolate colors and opacities or if we interpolate raw densities. Compositing comes last in both permutations, so we will not concern ourselves with this component for the matter of this discussion. Shading, on the other hand, requires the estimation of a gradient, so we need to add this operation to our pipeline. We can write the pipeline in a terse form using four operators: *T* for classification with a transfer function, *S* for shading, *D* for gradient (derivative) estimation, and *H* for interpolation [20]. Furthermore, we denote the discrete volume density function given at the volume grid points as *F*. For the remainder of this discussion we will use *H* and *h* interchangeably: *H* can both be the interpolation operator and the interpolation filter. Likewise, *D* can both be the derivative operator and the derivative filter. The combination of operators *F*, *D*, and *H* constitutes a convolution operation in the spatial domain, or alternatively, a multiplication in the frequency domain.

Splatting presently uses the pipeline in Fig. 1a, which interpolates a pre-classified and pre-shaded volume. Using the five operators from left to right, this pipeline can be written as $(FT, FD)SH$ (read from left to right). In this equation, the density volume is first classified using the iso-value transfer function, then the gradients are computed at the grid points using e.g., a central difference filter, and finally the grid samples are shaded, using the gradients and the material colors assigned in the classification step. This pipeline allows fast processing for two reasons: (i) the gradients need only to be computed once per viewing session as a pre-processing step (unless we change the gradient transfer functions), and (ii) shading occurs only as a pre-processing step once per viewpoint, and does not have to be performed each time a voxel kernel is intersected by a slab. The colors and opacities stored at the grid points are then interpolated using the interpolation filter *h*.

The interpolation filter, *h*, usually deviates from the ideal *sinc* filter, and it will have a lowpassing effect when used to interpolate the color and opacity volume. A filter kernel that is often used in splatting is a radially symmetric Gaussian function of radial extent 2.0:

$$h(r) = b \cdot 2^{-2 \cdot r^2} \tag{6}$$



(a) original step function: the edge

(b) blurred edge, after convolving the edge with Gaussian filter *h*

(c) crisp edge, after thresholding the blurred edge with iso-value

Fig. 2: Origin of blurring, and its prevention.

Here, $r^2 = x^2 + y^2 + z^2$, and $b = 0.214$ is a scaling factor that normalizes the kernel. This kernel is pre-integrated into a 2D footprint, which gives rise to another Gaussian. Fig. 2 shows what happens if this kernel is used to interpolate a step function, constituted by an iso-edge (Fig. 2a). If the voxels store colors and opacities, the edge will appear blurred, as is evident in Fig. 2b.

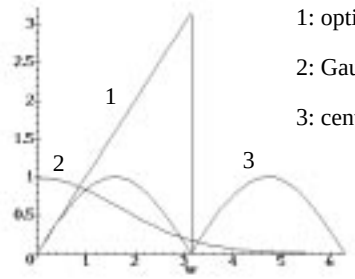
The post-shading pipeline of Fig. 1b can be written as: $(FHT, FHD)S$. Here, the grid voxel densities are first interpolated, and the interpolation result is then classified and shaded. Fig. 2b illustrates that *h* will again have a lowpassing effect and blur the interpolated volume. However, this time the classification step is still ahead, and we can undo some of the blurring by clipping off the blurred image regions, using the iso-thresholds. The surviving pixels are then shaded, and we observe that the rendered silhouette edge is considerably crisper and is located close to the true edge (Fig. 2c).

3.2 Gradient estimation

The remaining question is: How does one compute the gradients? The post-shading pipeline $(FHT, FHD)S$ indicates that the gradient is calculated by applying the two operators *H* and *D* to the discrete volume function. It was shown by Möller [20] that the operators *F*, *D*, and *H* are both associative and commutative, i.e., the sequences (*FH*)*D*, *F*(*DH*), and (*FD*)*H* are all equivalent. Thus, we could first interpolate *F* and then compute the gradients (the (*FH*)*D* scheme) or we could compute the gradients at the discrete grid locations and then interpolate the gradients (the (*FD*)*H* scheme), or we could construct a filter *HD* and use it to interpolate *F* (the *F*(*DH*) scheme). In [20] it was also shown that (*FH*)*D* is the most efficient of the three, as long as we provide some sort of caching of the interpolated values *FH*, to be utilized for the calculation of gradients nearby. An alternative, but not equivalent, form of *FHD* is *FH'*, where *H'* is the derivative of *H* (see Bentum [2] for further details).

In splatting, we usually employ radially symmetric filters, such as Gaussians, such that we can use the footprints for all viewing directions. However, neither Möller nor Bentum consider radially symmetric filters. Thus we find it worthwhile here to offer some further investigations on this subject. Fig. 3a shows the frequency spectrum of the Gaussian kernel of (6). The frequency response of the central difference filter is also shown (derived in [2]). Since (*FH*)*D* is equivalent to *F*(*DH*), we can capture the effect of *DH* on *F* by multiplying the frequency responses of the two filters. The result is spectrum 2 in Fig. 3b. Notice that the frequency response of the ideal gradient filter is given by a linear highpass of unit slope, that stops at a frequency of π, i.e., half the grid's Nyquist frequency (spectrum 1 in Fig. 3). Finally, spectrum 3 in Fig. 3b is the frequency response of *H'*, the derivative of the Gaussian given in (6). The spatial filter function of *H'* is given by:

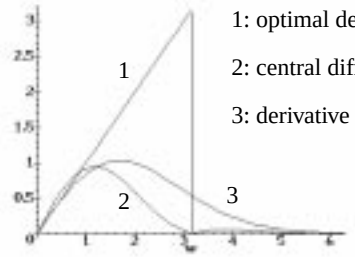
$$h'(r) = -b' \cdot r \cdot 2^{-2 \cdot r^2} \tag{7}$$



1: optimal derivative filter

2: Gaussian (H)

3: central difference (D)



1: optimal derivative filter

2: central difference * Gaussian (DH)

3: derivative of Gaussian (H')

Fig. 3: (a) Frequency spectra of Gaussian (*H*) and central difference filter (*D*). The optimal derivative filter is an ideal highpass: a line of slope=1 in frequency space, that stops at frequency=π. (b) Frequency spectra of the *DH* and the *H*' derivative filter, respectively.

where b' = 0.855 is a normalization factor. Looking at the frequency responses in Fig. 3b, we notice that *DH* has a faster fall-off in the stop band, but also attenuates frequencies in the upper passband. On the other hand, *H*' preserves the passband frequencies better, but also passes more frequencies in the stopband. It is therefore more susceptible to noise. Clearly, a trade-off must be made. If our dataset contains fine detail, like bone fractures in medical images, then we would want a gradient filter with a high acuity in the upper passband to ensure the proper accentuation of these structures in the rendered image. However, if our dataset contains much high-frequency noise, as is often the case in medical datasets (especially in MRI), then we desire a gradient filter that is less sensitive in these regions of the spectrum. Since a gradient filter is a high-pass filter and emphasizes high frequencies, noise that usually resides in these frequency bands can be greatly amplified by these filters, generating undesirable high-frequency artifacts.

4 NEW BLUR-FREE SPLATTING

We have just established that in order to rid splatting from much of the blurring, we need to formulate splatting as a post-shading pipeline. We will now describe both conceptual and implementational issues.

4.1 Concepts

Recall Equation (4), which gives the compositing equation for raycasting when the post-shading pipeline is used. Notice that the density (and gradients) are assumed to be constant within the sampling interval. Splatting, on the other hand, can provide the density (and gradient) *integral* within the sheet-buffer slab, due to its pre-integration scheme. Thus the compositing equation for splatting using the post-shading pipeline can be written as follows:

$$I_\lambda(\boldsymbol{x}, \boldsymbol{r}) = \sum_{i=0}^{L/\Delta s - 1} C_\lambda(q(i\Delta s), g(i\Delta s)) \cdot \alpha(q(i\Delta s))\Delta s \prod_{j=0}^{i-1} (1 - \alpha(q(i\Delta s))\Delta s) \quad (8)$$

where

$$q(i\Delta s) = \int_{i\Delta s}^{(i+1)\Delta s} \frac{f(s)}{\Delta s} ds \qquad g(i\Delta s) = \int_{i\Delta s}^{(i+1)\Delta s} \frac{g(s)}{\Delta s} ds \quad (9)$$

Again, C_λ and α are now functions that translate the interpolated volume function and gradient values into color and opacity. Notice that splatting provides the average of the integrated density and gradients within a volume slab. On the other hand, ray-tracing (as well as the Shear-warp and the 3D texture mapping approaches) interpolates a random position and then uses a square rule for integration, as per equations (3) and (4). The latter approach can lead to aliasing artifacts, thus many researchers employ an expensive root-finding operation to determine the actual surface position [25].

4.2 Implementation issues

The operator (*FH*)*D* can be implemented by splatting the volume densities into a sheet buffer, and computing the in-sheet gradients by convolution with two orthogonal central-difference filters. The third gradient vector is perpendicular to the sheet-buffer plane and can be efficiently computed by caching the sheet-buffers immediately following and preceding the current sheet-buffer.

H' can be implemented by constructing three extra footprint section arrays, one for $\partial H/\partial x$, $\partial H/\partial y$, and $\partial H/\partial z$, respectively. The footprint integration is performed similar to that of the regular density footprint. We just use a different underlying 3D kernel, i.e., the $\partial H/\partial x$, $\partial H/\partial y$, and $\partial H/\partial z$ kernel, respectively, and perform each integration along z (assume that we always look down the z-axis after the viewing transformation). In (*FH*)*D*, the volume gradients are computed with respect to an orthogonal coordinate system in which two axes are aligned with the sheet-buffer plane and the other is perpendicular to the sheet-buffer plane. Thus the gradient coordinate system is always aligned with the viewing coordinate system, and its orientation changes with the view orientation. The gradient splats reconstruct the gradients along the same coordinate system: The $\partial H/\partial x$ and $\partial H/\partial y$ splats compute the in-sheet gradients, and the $\partial H/\partial z$ splats compute the gradient perpendicular to the sheet-buffer. Rotating the gradient splats in this way yields correct results, since the kernel *H* is radially symmetric and its gradients are identical for all viewing directions. Hence, the gradient splats can be used for all viewing directions. Due to the imperfect kernels, however, the resulting gradient vector may be somewhat view-dependent.

Once the gradient vector components g_x, g_y, g_z have been obtained, we must normalize them by division with the gradient vector magnitude before we can use them for shading. This involves an expensive square root computation. Clyne and Dennis [5] have proposed an efficient work-around: If only the gradient direction is of interest, then we can perform a mapping from cartesian space to spherical space to get the latitude/longitude angles of the gradient vector. These can then be used to obtain the normalized gradient components by mapping back into cartesian space. Briefly, the spherical mapping is computed by:

$$\lambda = \operatorname{atan}\frac{g_y}{g_x} \qquad \phi = \operatorname{atan}\frac{g_z}{\sqrt{g_x^2 + g_y^2}} \quad (10)$$

These angles are then mapped back into normalized cartesian space:

$$g_x^n = \cos\lambda \cdot \cos\phi \qquad g_y^n = \sin\lambda \cdot \cos\phi \qquad g_z^n = \sin\phi \quad (11)$$

All major function calculations can be efficiently performed

using lookup tables, if we are willing to sacrifice some resolution, say by converting g_x, g_y, g_z into signed bytes. Furthermore, a variety of schemes exist to efficiently compute approximate gradient magnitudes as well (see the *Graphics Gems* series, e.g., [26]). Finally, the angles λ and ϕ can also be used directly, to index a lookup table that stores the result of the shading equation for a set of normals uniformly distributed on a sphere [28].

We can limit the amount of computation in the shading procedure by dividing the sheet-buffer into a set of tiles. Each tile is associated with a counter that is initialized to zero for each new sheet-buffer. This tile counter is incremented whenever a voxel center projects into the tile. Then, once all voxels have been projected onto the sheet-buffer, we only need to perform gradient and shading computations in those tiles that have been touched by a footprint. To account for the entire footprint-tile coverage, we must add a seam of half the footprint size to each tile (since only the footprint centers were used to determine what tile was hit). We have also experimented with bounding boxes that encompass the rectangular sheet area that received footprint contributions in the current sheet.

Finally, compositing the shaded sheet-buffers is performed as usual, and the occlusion map acceleration can also be employed unchanged.

4.3 Cost analysis

Although extra overhead is incurred by the need for gradient estimation and shading on the image plane, post-shaded splatting offers substantial savings in another step of the volume rendering pipeline: the footprint rasterization step. These savings help balance the costs of the two methods. When a color image is generated in pre-shaded splatting, four footprints must be rasterized per kernel section: red, green, blue, and alpha. On the other hand, if we use the *F*(*DH*) scheme in post-shaded splatting, then we only have to splat one footprint per voxel, i.e., the density footprint. Savings ensue, since lookup-table assisted shading and gradient estimation requires considerably fewer operations than the over 128 multiplications and additions that are required to rasterize four footprints at one-to-one viewing.

When we use the *H*' gradient estimation mode, then we do not save any splat rasterization operations. Just as in pre-shaded splatting, we have to splat four footprints per voxel: the three gradients and the density. So we do not anticipate any savings for *H*'. Rather, we expect somewhat higher cost due to the extra work for gradient estimation and shading on the image plane.

We should add that with the new algorithm, the number of voxels to be splatted is generally larger than with pre-shaded splatting. This is simply because in order to ensure proper gradients, we also need to splat voxels with values that are slightly outside the specified iso-ranges. Otherwise we would perform a reconstruction of a binary object, which would lead to possibly large aliasing artifacts. Another consequence of not loading a seam of voxels below the isovalue can be a shrinking of the object.

4.4 A hybrid method: the pre-shaded post-alpha splatting pipeline

Pre-shaded splatting is potentially faster than post-shaded splatting since it is likely that the number of visible voxels that require pre-shading is less than the number of pixels that require post-shading. Furthermore, the number of voxels to be projected in pre-shaded splatting is lower than the number of voxels in post-shaded splatting, because we need only project the voxels that are classified within the iso-interval. Of course, we gain crisper images in post-shaded splatting, due to the iso-contouring on the image plane before shading, and we would like to maintain that advantage. Thus the question is, can we use the pre-shaded approach that saves us the extra voxel projections and the per-pixel shading, and still use iso-contouring on the sheet-buffer plane? In such an approach, we would pre-shade the voxels as usual, but then project this color volume along with the raw density volume. We would not pre-classify the alpha channel. Instead, we would classify the pixels on each sheet-buffer plane, similar to the post-shaded approach. All pixels that have a projected density within the iso-interval are set to some alpha value, while all other pixels are reset to fully transparent. This hybrid approach seems to combine the good aspects from both the pre-shaded and the post-shaded volume rendering pipeline, and we call this approach the *pre-shaded post-alpha splatting pipelin*e, due to the delayed alpha classification.

5 RESULTS

We have applied the presented splatting pipelines to the UNC MRI dataset (256×256×145 voxels) and a ganglion nerve cell dataset (512×512×76 voxels). Our rendering results are shown in Fig. 6 (color plates), the images have 512×512 pixels. The difference in image quality is quite striking. Consider first the UNC head dataset. The new post-shaded splatting algorithm (Fig. 6, column 2) produces images that look significantly sharper than those produced with the traditional algorithm (Fig. 6, column 1), especially when the object is viewed under magnification. Flickering between a pair of corresponding images has the feel of a visit at the optometrist: the left image is seen with glasses off, the right is seen with glasses on.

The new algorithm accentuates fine surface detail very well and renders the object with a much sharper look than the traditional splatter. In our experiments, we have set alpha to 1.0 whenever a pixel was within the iso-interval. This provides for very crisp surfaces. However, semi-transparent transfer functions are also possible. As a matter of fact, the algorithm works anytime a transfer function is available for post-shading, both for color and opacity.

The images in the center column of Fig. 6 were produced using central differencing for gradient estimation. We can even improve the image quality, to some extent, by using gradient splats instead. For example, the zoomed brain view in the right column of row 4 in Fig. 6 was generated with gradient splats. We notice that the detail is somewhat crisper and the specular highlights are more pronounced. This was to be expected since the Gaussian derivative splat preserves higher frequency better than the central differenced Gaussian, as was illustrated earlier.

Row 2 and 3 in column 3 of Fig. 6 show the ganglion nerve cell rendered with pre-shaded splatting and with post-shaded splatting, respectively. We again notice a crisper, almost plastic-like, appearance of the nerve cell when rendered with post-shaded splatting.

For this paper, we have stressed qualitative issues over code optimization. We have used occlusion and frustrum culling as well as bounding boxes to limit the amount of shading calculations on the image plane. But we have neither implemented the fast gradient calculation schemes nor do we use the fast shading methods that were mentioned in Section 4.2. Our timings, given in Table 1, reflect this circumstance: Our traditional splatter is still significantly faster than the newly developed algorithm. This is expected to remain true as far as the *H*' gradient estimator is concerned, but we anticipate significant speedups for the *F*(*DH*) scheme when fast lookup-table assisted shading is implemented. We have conducted some preliminary experiments in this area, and we found that the time required for shading and gradient normalization amounts to about 75% of the value given in row 2, 'shading', of Table 1. On the other hand, due to the circumstance that we use texture mapping hardware to project the splats, we do not get the anticipated savings in the footprint rasterization stage, since the texture mapping hardware projects 4-channel RGBA splats at similar speeds as it does 1-channel Luminance splats. We expect the savings in a

image in Fig. 6	full head		eye		brain		brain zoomed			ganglion nerve cell	
splatting method	pre-shade	post-shade	pre-shade	post-shade	pre-shade	post-shade	pre-shade	post-shade centr. diff.	post-shade grad. splat	pre-shade	post-shade
footprint rasterization	15.6	11.4	17.4	14.5	14.5	13.0	16.7	16.2	16.8	9.4	6.5
shading	1.8	14.0	0.7	17.1	1.8	13.6	0.3	17.8	17.0	3.5	15.6
compositing	3.5	3.5	2.4	2.7	3.9	4.3	1.4	1.4	1.4	2.7	3.0
total	21.0	29.9	20.5	35.2	20.3	31.7	18.5	36.2	36.0	15.7	25.6

Table 1: Timings in secs obtained on a SGI Onyx using a R10000 194MHz CPU. The columns correspond to the images in Fig. 6. 2D texture mapping hardware was employed to perform the footprint rasterization, and graphics hardware was utilized for sheet-buffer compositing. However, we have recently also implemented a software splatter that runs at the same speed and better.

pure software implementation to be significantly more pronounced.

5.1 Results with pre-shaded post-alpha splatting

Column 3 in row 1 of Fig. 6 shows an image that was obtained with the pre-shaded post-alpha splatting pipeline. We notice significant staircasing artifacts in this image. To explain this artifact, consider a simple 2D example with two slices of three voxels each (see Fig. 4). Here, the white voxels have a density f_d=0, the others have f_d=2·f_{iso}, with f_{iso} being the iso-value. Let us further assume that we have a pre-shading function that sets the white voxels to color C_d=0.0 and the black voxels to C_d=1.0. Consider the interpolated image pixel due to pixel ray 1. It passes halfway between the rightmost two voxels of the first slice in Fig. 4. Using simple linear interpolation, we would interpolate C=0.5 and f=f_{iso}. Thus the iso-contour traverses right between these two voxels and the assigned pixel color would be C=0.5. When using the post-shaded pipeline instead, we again get f=f_{iso}, but now lighting is yet to be applied, yielding C=1.0 (assuming an appropriate gradient and light source position). We set α=1.0 in both cases, so no more color can be composited at that pixel. Thus in the pre-shaded post-alpha image, C=0.5 is the final color at that pixel, and this explains the dark rings right at the boundary of the stairstep. Pixel ray 2 does not receive any contribution from slice 1 (the interpolated voxels are both below the iso-threshold), but in slice 2 both interpolated voxels have f_d=2·f_{iso} and C_d=1.0. Thus the interpolated color between them is 1.0. This explains the bright regions inside the dark rings.

We can avoid the slice effect, if, for example, we set the transfer function $\alpha(f(s))$ to a smooth ramp. Then slice 2 would have a chance to contribute to pixel 1. We have observed that this eases the staircasing artifacts, but at the cost of an X-ray effect, since we now make our object artificially transparent (see Fig. 5a). So we again get a blurring, just this time we get it in the z-direction. For the image in Fig. 6, we actually used a smooth ramp for α. Otherwise, had we set α=1.0 for all f>f_{iso}, the staircasing would have been even more pronounced (see Fig. 5b).

6 CONCLUSIONS

Splatting has long been plagued by the blurry look of the rendered objects. Sharp edges appear fuzzy and fine object detail is greatly diminished. The blurriness occurs since splatting projects pre-shaded voxels with a relatively large splatting kernel, which leaves a fuzzy footprint on the screen. The overall effect is a low-passing of the projected color and opacity volume, which cannot be easily reversed. A simple iso-contouring of each projected color slice, via classification of a simultaneously projected density slice (i.e. the pre-shaded post-alpha pipeline), does not yield the desired results. It is hence apparent that one cannot resolve the blurring with pre-shaded volumes. Thus, our solution to this problem is to use a post-shaded pipeline for splatting instead, performing both classification and shading operations after the projection of the voxels that fall into a sheet-buffer slice. In this way, the non-linear iso-surfacing can be performed after the lowpassing effect of the splatting interpolation kernel. The result is a much sharper look to the volume rendered object.

Since this new method must perform the shading on the projection plane, the estimation of gradients is required. We have proposed two methods for gradient reconstruction: (i) Central differencing on the projection image, and (ii) the projection of a gradient volume using gradient splats. Here it turns out that the regrouping of the volume rendering pipeline is also advantageous in terms of efficiency. When the gradients are estimated via central differencing, then only one footprint rasterization is needed per voxel, i.e., the rasterization of the density footprint. Traditional splatting requires four such rasterizations per voxel (two, if a grey-level image is rendered), which is also required if a gradient volume is splatted to reconstruct the gradients on the screen. Although splatting the gradient volume preserves high frequencies better, it may also may cause aliasing for noisy volume data. Central differencing, on the other hand, may not preserve the gradients of fine

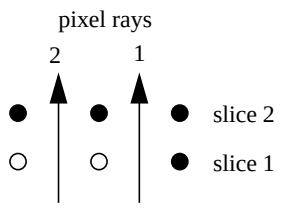


Fig. 4: Interpolating two pixel rays traversing two voxel slices of three voxels each.

(a) (b)

Fig. 5: Pre-shaded post-alpha splatting: (a) using a ramp starting at $\alpha(f_{iso})$=0.1, (b) setting all $\alpha(f)$=1.0 for f>f_{iso}.

object detail as well, but may tend to less aliasing.

The new method preserves the advantages of splatting: a sparse volume representation where only the relevant voxels have to be projected (now along with a layer of boundary voxels), and a volume integration using segments of density integrals instead of point samples. The new method also allows the use of the efficient and qualitatively superior framework of image-aligned splatting, in conjunction with screen occlusion maps for early culling of occluded voxels.

7 FUTURE WORK

We would like to investigate better gradient kernels than the Gaussian. However, we are limited by the fact that we have to use radially symmetric kernels. Thus we cannot efficiently use the kernels presented in [2] and [20], but work is underway to expand the theory presented in [20] to rotationally symmetric kernels. Another promising avenue of research in this respect are the Bessel-Kaiser kernels derived by Lewitt [16]. These kernels have very desirable frequency characteristics, and have been shown to yield interpolation kernels superior to the Gaussian. We plan to investigate the derivative of these kernels to improve the frequency characteristics of our *H*' kernel. Another kernel worthwhile investigating for gradient estimation is the Crawfis-Max kernel [6], which has been designed for optimal traditional splatting.

We are currently also investigating better ways to deal with the staircasing artifacts in the pre-shaded post-alpha splatting pipeline. This pipeline may be well suited to remove blurring in presegmented, tagged, and distance volumes, where no transfer function is available for post-shading.

Much work needs to be done on speeding up the gradient estimation and shading stage. Implementing the fast schemes mentioned in Section 4.2 should make the new algorithm a lot more competitive with the traditional one in terms of speed.

Unlike the hardware employed in the 3D texture mapping approaches of [4][28][29], the 2D texture mapping hardware currently used by our algorithm is not confined to expensive graphics workstations. Rather, it can be obtained for a few $100 in form of plug-in graphics boards for use on PCs. But we are also presently porting the footprint rasterization and compositing operations into a pure software environment. Initial results indicate that, in many cases, this software implementation is actually faster than the hardware-assisted implementation.

We have only just begun to explore the power of our new approach for the generation of higher quality images with splatting. So far we have only rendered fully opaque objects, such as the UNC head, but we have obtained initial results for semitransparent datasets as well. In future work, we seek to develop transfer functions that use the gradient and the gradient strength to enhance apparent surfaces in the volumetric dataset. The works by Levoy [14] and Drebin [7] shall be an initial starting point here.

References

[1] R. Avila, T. He, L. Hong, A. Kaufman, H. Pfister, C. Silva, L. Sobierajski, and S. Wang., "VolVis: a diversified volume visualization system," *Proc. Visualization' 94*, pp. 31-38, 1994.

[2] Bentum M.J., Lichtenbelt B.B.A., Malzbender T., "Frequency analysis of gradient estimators in volume rendering", *IEEE Trans. on Visualization and Computer Graphics*, vol. 2, no. 3, pp. 242-254, 1996.

[3] J. F. Blinn, "Light reflection functions for simulation of clouds and dusty surfaces," *Proc. SIGGRAPH '82*, pp. 21-29, 1982.

[4] B. Cabral, N. Cam, and J. Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware," *1994 Symposium on Volume Visualization*, pp. 91-98, 1994.

[5] J. Clyne, J.M. Dennis, " Interactive direct volume rendering of time-varying data," *VisSym'99*, Vienna, Austria, May 1999.

[6] R. Crawfis and N. Max, "Texture splats for 3D scalar and vector field visualization," *Visualization'93*, pp. 261-266, 1993.

[7] R. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," *Proc. SIGGRAPH'88*, pp. 65-74, 1988.

[8] K.H. Hoehne B. Pfiesser, A. Pommert, M. Riemer, T. Schiemann, R. Schubert, U. Tiede., "A virtual body model for surgical education and rehearsal," *IEEE Computer*, vol. 29, no. 1, 1996.

[9] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He, "Virtual voyage: intercative navigation in the human colon," *Proc. SIGGRAPH'97*, pp. 27-34, 1997.

[10] H. Hoppe, "Progressive Meshes," *Proc. SIGGRAPH'96*, pp. 99-108, 1996.

[11] J. Huang, R. Crawfis, and D. Stredney, "Edge preservation in volume rendering using splatting," *1998 Symp. Volume. Vis.*, pp. 63-69, 1998.

[12] J. T. Kajiya and B.P. Von Herzen, "Ray tracing volume densities," *Proc. SIGGRAPH '84*, pp. 165-174, 1994.

[13] P. Lacroute and M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation," *Proc. SIGGRAPH '94*, pp. 451- 458, 1994.

[14] M. Levoy, "Display of surfaces from volume data," *IEEE Comp. Graph. & Appl.*, vol. 8, no. 5, pp. 29-37, 1988.

[15] M. Levoy, "Efficient ray tracing of volume data," *ACM Trans. Comp. Graph.*, vol. 9, no. 3, pp. 245-261, 1990.

[16] R.M. Lewitt, "Multi-dimensional digital image representations using generalized Kaiser-Bessel window functions," *J. Opt. Sec. Am. A*, vol. 7, no.10, pp. 1834-1846, 1990.

[17] W. E. Lorensen and H. E. Cline, "Marching cubes: a high resolution 3D surface construction algorithm," *Proc. SIGGRAPH'87*, pp. 163-169, 1987.

[18] R. Machiraju and R. Yagel, "Efficient Feed-Forward Volume Rendering Techniques for Vector and Parallel Processors," *SUPERCOMPUTING'93*, pp. 699-708, 1993.

[19] N. Max, "Optical models for direct volume rendering," *IEEE Trans. Vis. and Comp. Graph.*, vol. 1, no. 2, pp. 99-108, 1995.

[20] T. Moeller, R. Machiraju, K. Mueller, and R. Yagel, " A comparison of normal estimation schemes," *Visualization'97*, pp. 19-26, 1997.

[21] K. Mueller and R. Yagel, "Fast perspective volume rendering with splatting by using a ray-driven approach," *Proc. Visualization'96*, pp. 65-72, 1996.

[22] K. Mueller, N. Shareef, J. Huang, and R. Crawfis, "High-quality splatting on rectilinear grids with efficient culling of occluded voxels," *IEEE Trans. Vis. Comp. Graph.*, vol. 5, no. 2, pp. 116-134, 1999.

[23] K. Mueller and R. Crawfis, "Eliminating popping artifacts in sheet buffer-based splatting," *Proc. Visualization'98*, pp. 239-245, 1998.

[24] R. Osborne, H. Pfister, H. Lauer, T. Ohkami, N. McKenzie, S. Gibson, and W. Hiatt, "EM-Cube: an architecture for low-cost real-time volume rendering," *Proc. Eurographics Hardware Rendering Workshop*, pp. 131-138, 1997.

[25] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P. Sloan, "Interactive ray tracing for isosurface rendering," *Proc. Vis'98*, pp. 233-238, 1998.

[26] J. Ritter, "A fast approximation to 3D euclidian distance," *Graphics Gems*, A. Glassner, editor, pg. 432-433, Academic Press, 1990.

[27] U. Tiede, T. Schiemann, and K.H. Hoehne, "High quality rendering of attributed volume data," *Proc. Visualization'98*, pp. 255-262, 1998.

[28] A. Van Gelder and K. Kim, "Direct volume rendering via 3D texture mapping hardware," *Proc. 1996 Vol. Rend. Symp.*, pp. 23-30, 1996.

[29] R. Westermann, T. Ertl, "Efficiently using graphics hardware in volume rendering applications," *Proc. SIGGRAPH'99*, pp.169-177, 1999

[30] L. Westover, "Interactive volume rendering," *1989 Chapel Hill Volume Visualization Workshop*, pp. 9-16, 1989.

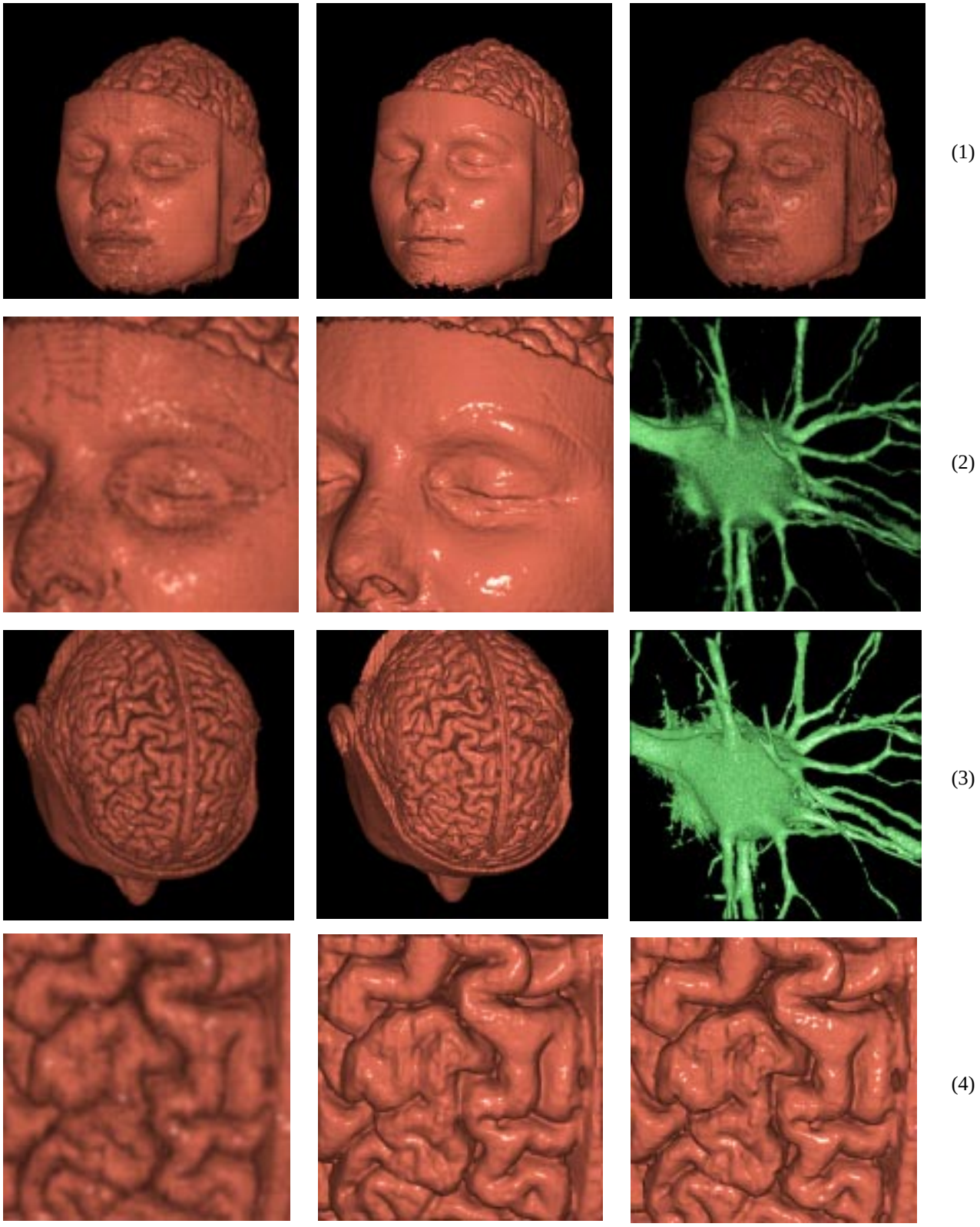
[31] L. Westover, "Footprint evaluation for volume rendering," *SIGGRAPH'90*, pp. 367-376, 1990.

[32] L. Westover, "SPLATTING: A parallel, feed-forward volume rendering algorithm," *PhD Dissert.*, UNC-Chapel Hill, 1991.

[33] G.J. Wiet, R. Yagel, D. Stredney, P. Schmalbrock, D.J. Sessanna, Y. Kurzion, L. Rosenberg, M. Levin, K. Martin, "A volumetric approach to virtual simulation of functional endoscopic sinus surgery", *Medicine Meets Virtual Reality 5*, 1997.

[34] J. Wilhelms and A. Van Gelder, "A coherent projection approach for direct volume rendering," *Computer Graphics (SIGGRAPH '91 Proceedings)*, pp. 275-284, 1991.

[35] C. Wittenbrink, T. Malzbender, and M. Goss, "Opacity-weighted color interpolation for volume sampling," *1998 Symposium on Volume Visualization*, pp. 135-142, 1998.



(1)

(2)

(3)

(4)

(1) full head, (2) eye, (3) brain, (4) brain zoomed rendered with pre-shaded splatting

(1) head, pre-shaded w/ post-alpha, $\alpha(f_{iso})=0.7$
(2) ganglion nerve cell, pre-shaded
(3) ganglion nerve cell, post-shaded
(4) zoomed brain, post-shaded with gradients estimated with gradient splats

rendered with post-shaded splatting, gradients estimated with central differencing

Fig. 6: Colorplate