Appeared in the Proceedings of the *Visualization'98* conference, pp. 239-245

Eliminating Popping Artifacts in Sheet Buffer-Based Splatting

Klaus Mueller and Roger Crawfis

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

ABSTRACT

Splatting is a fast volume rendering algorithm which achieves its speed by projecting voxels in the form of pre-integrated interpolation kernels, or splats. Presently, two main variants of the splatting algorithm exist: (i) the original method, in which all splats are composited back-to-front, and (ii) the sheet-buffer method, in which the splats are added in cache-sheets, aligned with the volume face most parallel to the image plane, which are subsequently composited back-to-front. The former method is prone to cause bleeding artifacts from hidden objects, while the latter method reduces bleeding, but causes very visible color popping artifacts when the orientation of the compositing sheets changes suddenly as the image screen becomes more parallel to another volume face. We present a new variant of the splatting algorithm in which the compositing sheets are always parallel to the image plane, eliminating the condition for popping, while maintaining the insensitivity to color bleeding. This enables pleasing animated viewing of volumetric objects without temporal color and lighting discontinuities. The method uses a hierarchy of partial splats and employs an efficient list-based volume traversal scheme for fast splat access. It also offers more accuracy for perspective splatting as the decomposition of the individual splats facilitates a better approximation to the diverging nature of the rays that traverse the splatting kernels.

1 INTRODUCTION

Volume rendering has gained great popularity in recent years as it allows the user to comprehend and visualize a volumetric dataset in its true continuous three-dimensional representation, and not just as a shell of isosurfaces, as is the case in polygonal models. In volume rendering, the three-dimensional structures do not need to be segmented into binary objects, but can retain their natural fuzzy character, which is more appropriate considering that a voxel possibly constitutes a mix of various materials. A binarization with the goal of producing an isosurface, tiled with polygons, destroys this interplay of microsurfaces that will produce subtle but perceptible variations of color in a volumetric display. With volume visualization, objects of interest can be rendered embedded in their surrounding structures, represented by semitransparent clouds. This helps the user to keep track of existing spatial relationships, and at the same time makes for a more natural and realistic display. Maintaining the volumetric representation also enables users to manipulate volumes and interact with its structures. Volume morphing, sculpting and surgical simulations are just a few examples of the immense potential that such a representation has to offer.

Volume rendering is appropriate for any discrete dataset that was acquired from a formerly continuous object via sampling. Most medical imaging technologies, such as MRI, CT, Ultrasound, PET, and SPECT fall into this category. Volume rendering is also often used in scientific simulations, such as CFD or FEM, which approximate a continuum by a discrete representation, and generate their output on some discrete grid, usually irregular or curvilinear. The medical applications, on the other hand, mostly acquire their data as an axial stack of 2D slices, where the slices are uniformly sampled on a square grid, but the axial distance between slices is larger than the L_1 sample distance within the slices. Hence, a voxel is not a cube, but a box with a square base. Interpolation of intermittent slices is required if one desires a cubic grid. Shape-based interpolation [5] has become the method of choice for this task, but note that the interpolation of slice data increases the magnitude of the already large volume datasets even more. It is preferable that the volume renderer can deal with this unequal grid scaling without requiring extra interpolated slices.

Polygonal surface renderers compress the 3D data into a very sparse, vertex-edge-polygon representation, which enables fast display, even if none of the widely available polygon graphics hardware is used. Volume renderers, on the other hand, consider the full, uncompressed dataset, which increases both the required computational effort and the demands on data management.

In recent years, various volume rendering methods have been proposed that all aim to be faster than the traditional raycasting method. One way to classify these algorithms is by how much graphics hardware they utilize. On one end is the pure software-based shear-warp algorithm, which achieves impressive speeds by using a smart data structure and intensive data pre-processing [8]. On the other end is the class of dedicated volume rendering boards [7][14]. The performance of the shear-warp algorithm is very sensitive to changes in the transfer functions, while the hardware solutions, although extremely fast, suffer from the symptoms of any hardwired approach: they provide less flexibility than an algorithm configured on a general purpose machine. For this ongoing research our philosphy is to exploit the readily available, highly optimized polygon graphics hardware with 2D texturing as much as possible for the purpose of volume rendering, without adding dedicated custom hardware. On the other hand, we also seek an algorithm that can be efficiently used should no graphics hardware be at hand.

The method that fits our demands best is the splatting technique, proposed by Lee Westover [19]-[21]. This algorithm reduces the interpolation complexity of raycasting from the number of samples along the rays to the number of voxels within one or several iso-ranges. In splatting, each voxel is represented by a 3D kernel which is pre-integrated into a 2D footprint, weighted by the voxel value and mapped onto the image plane. The collection of all projected footprints then forms the final image. By mapping this footprint as an image onto a polygon, we can employ standard 2D texture mapping hardware for the projection process [2]. However, the footprint interpolation is also easily done in software with fast DDA and bit-blt procedures [10][12]. Splatting can be performed either in object-order [19]-[21] or, as a ray-based approach, in image-order [12]. Each approach offers its own set of acceleration techniques: iso-voxel lists [3], splat hierarchies [9] for the object-order technique, and space leaping [23], bounding boxes [16], and early ray-termination [4] for the ray-based approach. The optimal choice depends on the nature of the data.

The use of pre-integrated kernels introduces inaccuracies into the compositing process since the 3D reconstruction kernel is com-

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

posited as a whole, and not piecewise, as part of an interpolated sample along a viewing ray. This may lead to the bleeding of colors of hidden background objects into the final image [19][21]. The sheet buffer method, proposed subsequently in [20][21], reduces this problem, but causes disturbing popping artifacts in animated viewing. Section 3 discusses both methods in greater detail.

While in orthographic splatting all footprints have the same size, in perspective splatting the size of the footprints must be varied depending on their distance from the observer. This distortion, achieved by a distant-dependent stretch of the footprints, is necessary to enable the interpolation kernel to act as a lowpass filter in volume regions in which the sampling rate of the diverging ray grid falls below the grid sampling rate. More detail on this problem is provided in [13] and [17].

This paper is structured as follows. First, in Section 2, we provide a description of previous work that is relevant to the research presented here. Then, in Section 3, we deal with the disturbing color popping artifacts that occur in sheet buffer-based splatting. We first analyze the origins of these artifacts and then proceed to describe our new splatting approach that eliminates the condition for popping while maintaining the insensitivity to color bleeding. In Section 4 we outline a convenient list-based volume traversal method that allows an efficient implementation of all three splatting methods, *i.e.*, the two original ones and the new one, to run efficiently. Finally, in Section 5, we show some results that were obtained with the new enhancements, and follow with conclusions and an outlook into future work in this line of research.

2 RELEVANT PREVIOUS WORK

The basic element in most volume rendering applications is the volume rendering integral in its low-albedo form, first described by Kajiya and Von Herzen [6]. 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:

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

Here L is the length of ray $\boldsymbol{r}$. We can think of the volume as being composed of particles that receive light from all surrounding lightsources and reflect this light towards the observer according to the specular and diffuse material properties of the particles [11]. Thus, in (1), ϕ_λ is the light of wavelength λ reflected at location s in the direction of $\boldsymbol{r}$. Since volume particles have certain densities μ (*i.e.*, opacities), the light scattered at s is attenuated by the volume particles between s and the eye according to the exponential attenuation function. The process of merging colors and opacities along the ray is called *compositing*.

Usually (1) cannot be solved analytically, and thus a discretized form is used. A good approximation is obtained with raycasting, where a ray samples the volume at equidistant points, compositing the sample values from front to back. A ray sample value is computed by placing an interpolation kernel at the ray sample location and weighting the surrounding volume grid samples by the kernel function. The interpolated values can then be used to index transfer functions that steer the effect that this interpolated value has on the integral. Transfer functions may control color, opacity, and also gradients.

As described by Porter and Duff [15], the **over** operator composites a back sample with a front sample. The back sample is attenuated by the front sample's opacity, and the color contributions of the individual samples are attenuated by their individual transparencies. The compositing process is a weighted sum of the two samples, based on their attenuation factors. For the **over** operator (the **under** operator is just the complement), a composited color C_0 is computed from a back sample with color/opacity (C_B, α_B) and a front sample (C_F, α_F) with the following expression:

$$c_0 = C_B\alpha_B(1-\alpha_F) + C_F\alpha_F = c_B(1-\alpha_F) + c_F \qquad (2)$$

Here, the colors written in lower case denote sample colors that were pre-multiplied by their respective sample opacities. By using the **over** operator in sequence, all sample values along a ray can be composited in this way to yield the final color at the image pixel.

Porter and Duff used their framework to composite multilayer cel images. Volume rendering can be represented in this framework as well. By decomposing a volume into a stack of sampled (image) sheets, aligned parallel to the projection plane, we can render a projection image by compositing these sheets back-to-front (or front-to-back). The 3D texture-mapping hardware approach by Van Gelder and Kim [18], and also the one by Cabral, Cam, and Foran [1], does just this. In both approaches, each image sheet is individually interpolated from the volume samples. It is easy to see that interpolated sheet based-rendering is equivalent to raycasting, with all rays being traced simultaneously (see Figure 1). It is bound to be the best approach to approximate the volume rendering integral of (1). However, just like in raycasting, we must choose the sheet distance sufficiently small to avoid aliasing effects due to the imperfect interpolation filter. For sample distances other than the unit voxel distance we must also normalize the sample opacities.

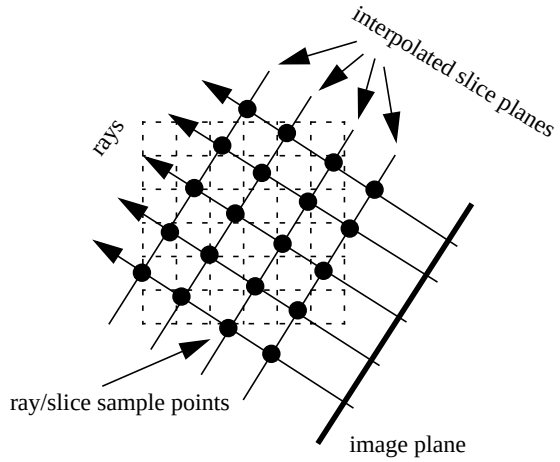


FIGURE 1. Equivalence of raycasting and slice-based volume rendering. In slice-based volume rendering, the interpolated slice planes are composited as images back-to-front (or front-to-back). Slice-based volume rendering is equivalent to raycasting, with all rays being cast simultaneously.

The interpolated slice-based approach was also mentioned by Westover in his dissertation [21] as the ideal approximation to volume rendering. However, it is an approach that does not lend itself well to acceleration methods (other than 3D texture mapping), since the sheets slice the volume data indiscriminately.

As was mentioned in the introduction, splatting 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. An interpolation kernel is placed at each voxel location. This enables one to view the volume as a field of overlapping interpolation kernels h which, as an ensemble, make up the continuous object representation. A voxel v_j's contribution is then given by $v_j \cdot \int h(s)ds$, where s follows the integration of the interpolation kernel in the direction of the ray. If the viewing direction is constant for all voxels or 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 [19]. Thus, in contrast to raycasting, splatting considers each voxel only once (for a 2D interpolation on the screen), and not several times (for a 3D interpolation in world space). In addition, as an object-order approach we only need to consider the relevant voxels, which in many cases only constitutes 10% of the volume voxels [22]. Also in contrast to raycasting, line integrals across a voxel are now continuous or approximated with good quadrature, and don't require normalization of α to compensate for sample distance. Finally, the efficient pre-integrated kernel representation allows splatting to use qualitatively better kernels (with larger extents) than the trilinear filter typically employed by raycasting.

3 A NEW, SUPERIOR SHEET-BUFFER SPLATTING ALGORITHM

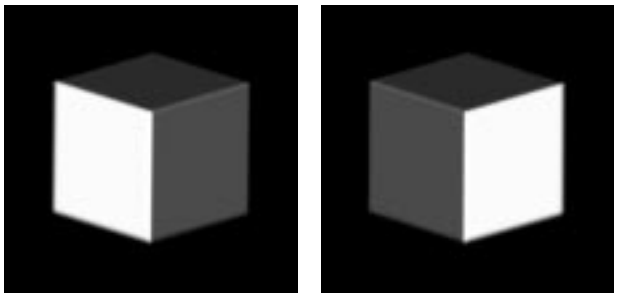
Both of the splatting methods that are in use today, *i.e.*, the *composite-every-sample* method and the sheet-buffer method, may exhibit very visible artifacts. These artifacts stem from the significant deviation of these methods from the discrete volume rendering model shown in Figure 1. The composite-every-sample splatting method violates the discrete volume rendering model because a volume sample point is not first reconstructed based on the values of the surrounding voxels before its visibility is determined. Instead, each voxel is independently composited on the image plane, without spreading its contribution along the main viewing axis (see [21] for a more thorough treatment). The sheet-buffer method was prescribed to eliminate these artifacts, however, it did so at the cost of disturbing popping artifacts for some viewpoint transitions. This is described next.

3.1 Origin of the popping artifacts in sheet buffer-based splatting

In the sheet buffer method, splats are added within sheets that are aligned parallel to the volume face most parallel to the image plane. (As a matter of fact, each sheet is constituted by a volume slice.) After a sheet buffer has been accumulated, it is composited into a cache image that traverses the volume back to front. The sheet-buffer splatting method comes closer to the discrete volume rendering model. In contrast to the composite-every-sample method, the voxel contributions are now added in slices, just like in the discrete model. However, two problems remain: (i) the slices are not parallel to the image plane, and (ii) a voxel spreads its entire energy into one slice, and not several, as dictated by the extent of the interpolation kernel. Thus the visibility calculation is still not accurate.

It is primarily the former issue that leads to the disturbing color popping artifacts when the main orientation of the sheet-buffers abruptly switches from one volume axis to another. An example of this artifact is shown in Figure 2. Here, Figure 2a shows a binary cube viewed at a 45˚ angle. We notice that the left face is much brighter than the right face of the cube. Figure 2b shows the cube viewed at an angle of 45.2˚. Now the right cube face is much brighter than the left face. Besides the fact that neither of the two renderings is correct (both faces should have the same shade and none of them should be as bright), the switch of the bright areas from the left to the right is disturbing in animated viewing.

Consider now Figure 3, where we illustrate this situation in 2D. In this figure, the image plane makes a 45˚ angle with the volume axis. This is the angle at which the sheet-buffer orientation is just about to switch and color popping is just about to occur. Let us consider an extreme case, where we render a binary cube with the lightsource and the eye at infinity. Hence, the two head-on cube faces both receive the same amount of shading. Notice that only the face voxels receive a non-zero shading value, since all other voxels have a gradient of zero.



(a) (b)

FIGURE 2. A cube is rendered with sheet buffer based-splatting at an orientation of (a) 45˚, (b) 45.2˚. In (a), the sheet buffers are parallel with the left cube face, in (b) they are parallel to the right cube face. Notice that the brighter cube face is always the one that is parallel to the sheet buffers.

Let us now go through a numerical example to illustrate our point. A Gaussian function with a radial extent of 2.0 is chosen as the splatting kernel. The pre-integrated kernel function is given by:

$$f(r) = 0.446 \cdot e^{-2.0 \cdot r^2} \tag{3}$$

where the overlapping kernels sum to 1.0 in a unity volume, within some tolerance. Table 1 shows the results of the color compositing process for the two representative face rays, ray_1 and ray_2. Here, we use front-to-back compositing (see equation (2)).

We see in Table 1, that ray_1's color is obtained in an additive process within a single sheet, while ray_2's color is composited across five sheets. We also see that the bright color of ray_1 comes from adding the entire kernel contribution at once, instead of compositing the kernel contributions along the ray. It is an inherent disadvantage of splatting's pre-integration scheme that it replaces the compositing of voxel contributions along a viewing ray by addi-

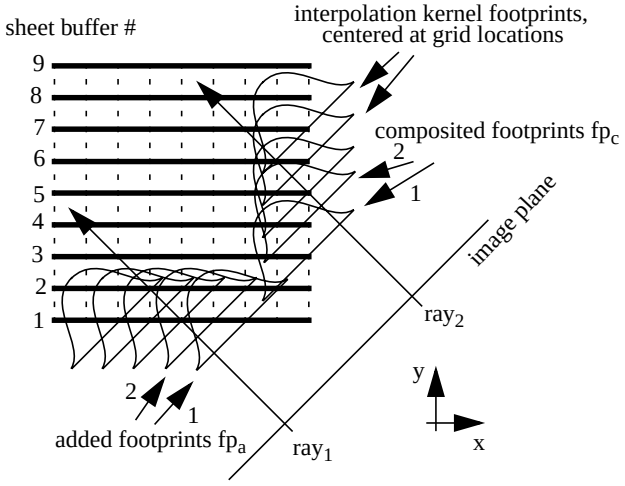


FIGURE 3. Sheet buffer-based splatting: Ray_1 and ray_2, penetrating one of each visible volume faces, accumulate considerably different colors opacities. (Only the face voxels have non-zero shaded colors, since everywhere else the gradients are zero.) While ray_1 adds all splat colors within a single sheet buffer, ray_2 composites them across several sheet buffers. Since compositing yields smaller results than adding, the pixel of ray_1 is much brighter that than the pixel of ray_2. Had the sheet-buffer been oriented along the y-axis, this situation would be reversed.

tions. This leads to a magnification of the contributions of the front voxels since the color c_{add} (and the opacity α_{add}) produced by addition is greater than the color c_{comp} (and α_{comp}) produced by composition. Consider the two pairs of voxel footprints fp_a and fp_c shown in Figure 3 that accumulate via addition or compositing, respectively. Here, α_1, α_2, and c_1, c_2 are the opacities and colors, respectively, that are interpolated by the footprints. We see that the color c_{add} obtained by adding the footprint pair fp_a:

$$c_{add} = (\alpha_1 + \alpha_2) \cdot (c_1 + c_2) = \alpha_1 c_1 + \alpha_1 c_2 + \alpha_2 c_1 + \alpha_2 c_2 \quad (4)$$

is significantly greater than the color c_{comp} produced by composition of footprint pair fp_c:

$$c_{comp} = \alpha_2 c_2 (1 - \alpha_1) + \alpha_1 c_1 = \alpha_2 c_2 - \alpha_2 c_2 \alpha_1 + \alpha_1 c_1 \quad (5)$$

We shall now offer a solution that prevents these problems.

3.2 A new sheet buffer-based splatting method that eliminates popping

We just illustrated that the popping artifacts in the sheet-buffer splatting method are due to the circumstance that the sheet buffers do not maintain a constant spatial orientation with respect to the image plane. Furthermore, the bright colors on the left face of Figure 2a resulted from adding the entire splat contributions at once, instead of compositing them along a ray. Both of these problems can be taken care of by recalling the discretized volume rendering model of Figure 1. Here, parallel slices cut across the volume, interpolating it into a sequence of 2D images, which are then composited back-to-front. Splatting has a somewhat different volume representation: Here, the volume is not an array of discrete data points that are used to support 3D interpolation, rather, it is a field of overlapping 3D spherical interpolation kernels, each chopped into (sphere) sections by the parallel slicing planes. The thickness of these kernel sections is determined by the distance between the slicing planes. Since we add to the sheet buffer all kernel material that is bounded by two slicing planes, and not just the kernel cross-section that is cut by a slicing plane, we view a pair of slicing planes as a slicing slab of certain width (or thickness).

Consider Figure 4, where our new splatting algorithm is illustrated. The sheet buffer is now parallel to the image plane, and only the contributions of the kernel sections that fall within the extent of the current slicing slab are added to the sheet buffer. Then, similar

Ray_1

sheet buffer #	kernel weight	opacity α_0	pre-mult. color c_0	true color C_0
1	0.446·(1+2·0.5+2·0.0625) = 0.947	0.947	0.898	0.948

Ray_2

sheet buffer #	kernel weight	opacity α_0	pre-mult. color c_0	true color C_0
3	0.0278	0.0278	0.0007	0.0025
4	0.223	0.2446	0.0483	0.197
5	0.446	0.592	0.198	0.334
6	0.223	0.683	0.218	0.319
7	0.0278	0.6912	0.218	0.315

Table 1: Rays ray_1 and ray_2 composite different colors. The accumulated color depends on the orientation of the sheet buffer with respect to the orientation of the volume faces the rays are penetrating. The true color $C_0 = c_0/\alpha_0$.

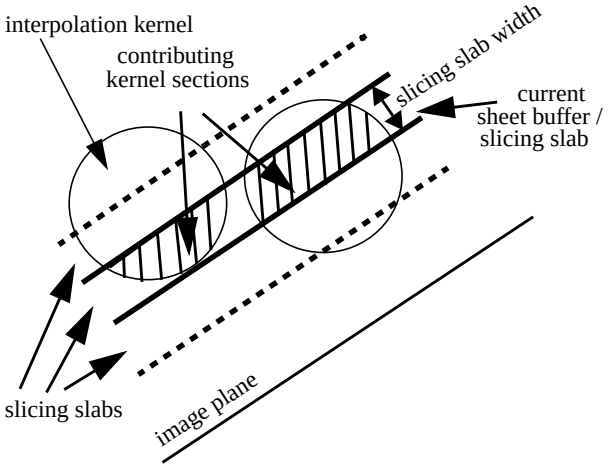


FIGURE 4. Adding the kernel sections, obtained by slicing the interpolation kernel by the current slicing plane, to the current sheet buffer.

to the traditional sheet-buffer method, once a sheet buffer has received all contributions, it is composited with the current image, and the slicing slab is advanced forward.

This new method requires an array of overlapping, pre-integrated kernel sections (see Figure 5). A slab which intersects a voxel kernel then simply picks the appropriate pre-integrated kernel section. We use 128 such sections, spaced apart by $\Delta s = (kernelExtent + slabWidth) / 128$ (see Figure 5, note that symmetry allows the reuse of sections for part of the kernel width). The integration width is the pre-set slab width. By using a splatting kernel of small extent, we can keep the number of intersected kernel slabs per voxel and the associated amount of additional interpolation operations at moderate levels. For instance, the kernel proposed by Crawfis and Max [2], that was specifically designed for accurate splatting, has a radial extent of only 1.6, which would require about four footprint mappings per voxel (given a slab width of 1.0). The number of compositing operations is increased by the same amount than the number of footprint mappings, due to the increased number of sheet buffers. Notice that shading is only performed once per voxel, thus no extra computations are required here. In addition, since we use 2D texture mapping hardware to map the footprints plus the graphics engine to perform the compositing, at least some of the extra work inflicted by the increased number of sheet buffers may be hidden by the high performance of today's graphics workstations.

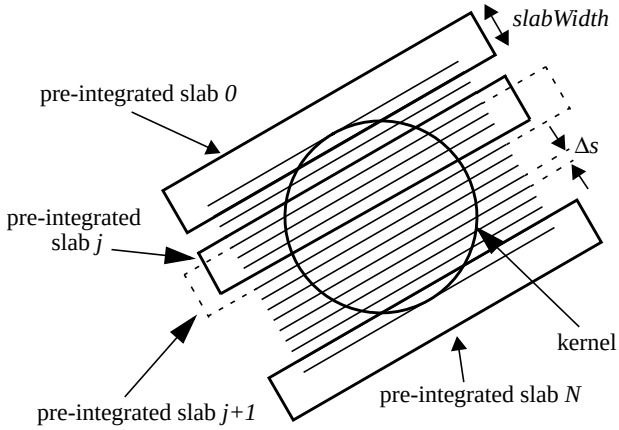
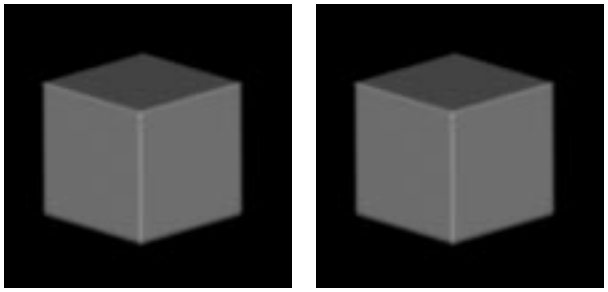


FIGURE 5. Array of pre-integrated overlapping kernel sections. The integration width of the pre-integrated slabs is determined by the slab width. The offset between adjacent slabs is Δs.



(a) (b)

FIGURE 6. Binary cube rendered with the new image-parallel sheet buffer splatting method at: (a) 45˚, (b) 45.2˚. In both cases the slab width was 1.0. We observe that both front faces have now equal brightness in both renderings. Not only is this correct, but it also implies that popping at 45˚ no longer occurs.

Figure 7a and b show the cube of Figure 2, now rendered with the new image-parallel sheet-buffer method. The Gaussian kernel of equation (3) and a slab width of 1.0 was used. We observe that the previous imbalance of brightness between the two cube faces no longer exists. The cube has the same shades for both image plane orientations, 45˚ and 45.2˚, which implies that popping no longer occurs. Notice also that the overall brightness of the cube is reduced, as now the kernel contribution has been divided into four parts which are composited back-to-front and are no longer added all at once. This represents an improvement of the splatting method towards the ideal discrete volume rendering model, however, it retains the advantages of splatting in that the ray integral is still continuous. Hence, the opacities do not have to be normalized for ray length, no matter what slab thickness we use.

In Figure 7a, b and c we have varied the slab width. Figure 7a shows the cube rendered with a slab width of 0.5, while Figure 7b and c show the cube rendered with a slab width of 1.0 and 2.0, respectively. The improvements for a slab width of 0.5 are visible but not significant, while a slab width of 2.0 produces strong periodic artifacts. These occur since the slice planes section the kernels differently depending on their location on the cube face. In regions where “thick” kernel sections are composited close to the image plane, bright regions ensue. This effect is equivalent to the one expressed on the left cube face in Figure 2a.

Notice that the thinner we make the slabs, the closer we approximate splatting to the ideal volume rendering integral of equation (1), however, the cost is quite high.

Thinner slabs produce darker images (as can be observed in Figure 7a, b, and c), due to the inequivalence of the adding and compositing operation. If we want to make the images obtained with different slab widths similar in color intensity we need to manipulate the footprint tables. Since within a sheet, the contributions of all intersected voxel kernels are additive, we can determine this normalization factor by considering one voxel kernel in isolation from all others. Figure 8a shows a kernel intersected by two thin slicing slabs of width w_{slab}. The contributions of the two kernel sections are composited back-to-front. In Figure 8b we have doubled the distance between the slicing planes, and the kernel is intersected by thicker slicing slabs of width $2 \cdot w_{slab}$. The pre-integrated kernel sections are now twice as thick than the ones in Figure 8a, and the contributions of the two kernel sections, formerly separated in Figure 8b, have been added into a thicker slab.

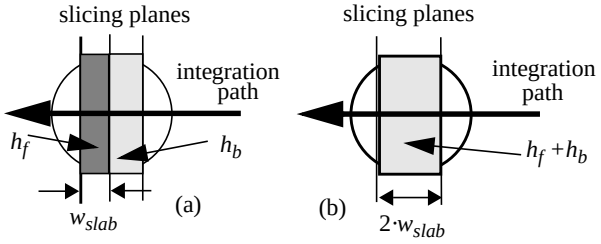


FIGURE 8. The composited integrals of two different widths of slicing slabs for a single voxel kernel: (a) h_f is the kernel section integral of a slab of width w_{slab} and h_b is the section integral of an adjacent slab of the same width, (b) $h_f + h_b$ is the integral of a thick slab of width $2 \cdot w_{slab}$.

The color c_{comp} resulting from compositing the two thin slabs is (note that we obtain interpolated color and opacity by scaling the voxel opacity α_v and color c_v, respectively):

$$\begin{aligned} c_{comp} &= (h_b\alpha_v)(h_b c_v)(1-\alpha_v h_f) + (h_f\alpha_v)(h_f c_v) \\ &= \alpha_v c_v(h_b^2 - h_b^2\alpha_v h_f + h_f^2) \end{aligned} \tag{6}$$

The color resulting from adding the two thin slabs into the thick one is as follows (this is equivalent of using the thick slabs for rendering instead of the thin ones):

$$c_{add} = \alpha_v c_v(h_b + h_f)^2 = \alpha_v c_v(h_b^2 + 2h_b h_f + h_f^2) \tag{7}$$

With good approximation:

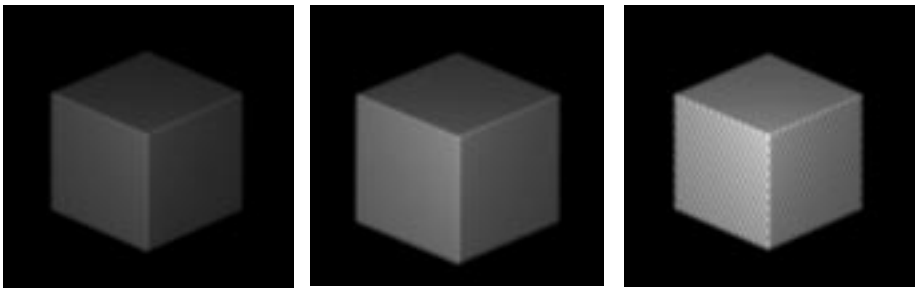
$$h_{add} = h_{comp} + 2h_b h_f + 0.5 \cdot h_b^2 h_f \tag{8}$$

(taking an average α_v=0.5, but other values are possible).

Thus, when computing the kernel footprints, we subtract the factor

$$2 \cdot h_b h_f + 0.5 \cdot h_b^2 h_f$$

for each section of the thick slab. In this way, it will render voxels at approximately the same color intensity than the two composited thin slabs. Using this approximation, we can build slab hierarchies with thicknesses of powers of 2. Other thicknesses may be normalized via linear interpolation. This differs from Laur and Hanrahan’s [9] scheme of increasing the Gaussian exponent, but it can more



(a) (b) (c)

FIGURE 7. Binary cube rendered with the new image-parallel sheet buffer splatting method at 45˚ with a slab width of: (a) 0.5, (b) 1.0, (c) 2.0. We observe that a slab width of 2.0 produces strong periodic artifacts, while a slab width of 0.5 does not improve image quality much. We conclude that a slab width of 1.0 is sufficient for our method.

generally be applied to non-Gaussian kernels.

The algorithm is easily expanded to perspective, the traversal and selection algorithm remain the same. Due to the partitioning of the splats the quality is likely to increase, since in perspective the splats must undergo a linear distortion along the viewing axis (as mentioned in the introduction and described in [17]), which is better represented by pre-integrated kernel sections than by pre-integrated full kernels. The necessary tilt of the kernels towards the direction of the traversing rays is also more accurately implemented this way. (See [13] for more details on accurate perspective splatting).

4 IMPLEMENTATION

We have implemented all three splatting methods discussed, i.e, the traditional sheet-buffer method, the new image-parallel sheet-buffer method, and the composite-every-sample method. We used an algorithm that builds upon the fast list-splat traversal method presented by Crawfis [3]. However, while the original algorithm capitalized on the fact that all voxels within the iso-range had similar colors and therefore did not require an ordered back-to-front (or front-to-back) traversal, the new version does not make this assumption. It is therefore suitable for a more general class of volume data sets. Let us now explain these new enhancements in further detail.

After the volume is read from memory, the voxels are sorted with respect to their value. This pre-sorted list of voxel values then enables the quick retrieval of relevant voxels within one or more iso-ranges via binary search. By maintaining a list of these relevant voxels, L_r, which can be sorted by various keys, we can quickly change shading parameters and transfer functions. Spatial and temporal coherency can be used to quickly update L_r, should the iso-range(s) change.

In both the composite-every-sample method and the new image-aligned sheet-buffer method we sort the voxels in L_r with respect to their distance from the viewing plane, while in the sheet-buffer method we sort the voxels in L_r according to their position on the volume axis most perpendicular to the image plane. The sorted voxels in L_r are then mapped and composited according to the respective splatting method. We used 2D texture mapping hardware to both project the splats and to perform the compositing. Diffuse and specular shading was done in software. It was found that the algorithm's run time is dominated by the polygon transform and rasterization operations. The added complexity of the image-aligned sheet-buffer method with respect to the traditional sheet-buffer method can be determined by the factor *splatExtent*/*slabWidth* and affects largely only the projection operations.

5 RESULTS

Figure 9a shows two consecutive frames of a flight around the UNC MRI-brain dataset (256×256×145 voxels), rendered with the traditional sheet-buffer splatting method. The viewing plane is angled about 45˚ with respect to either of the two front volume faces. The sheet buffer direction changes between the two frames, and we observe that in the first frame the patient's front face is overly bright, while in the next frame the persons cheek appears highly illuminated. In an animated view we would perceive this sudden change of brightness distribution as a very noticeable popping. Notice that the alignment of the compositing sheets with the volume slices reveals the severe staircasing of the object, for instance at the forehead and at the cheeks, where the object is insufficiently lowpass-filtered.

Now consider Figure 9b, where two frames at similar orientations to the ones in Figure 9a are shown, but this time rendered with the new slicing-slab splatting method. We observe that the brightness distribution is now more coherent with the underlying shading model and changes in a continuous fashion as the head rotates. Notice also that the object's surface appears much smoother, the discontinuities at the cheek and the forehead are completely gone. It is especially in these regions where the sectioned interpolation kernel provides for better reconstruction and compositing. In this figure, the width of the slicing slab was set to 1.0, the unit voxel spacing distance. Figure 9c illustrates what happens when a wider slab is used, *i.e.*, a slab of width 2.0. One can now see rather disturbing artifacts due to the insufficient and irregular compositing along the object surface.

Finally, Figure 9d shows two views of a nerve cell dataset, acquired by a confocal microscope and rendered with our new variant of sheet-buffer splatting. Again, no discrepancies in illumination are visible between the two frames.

The number of relevant voxels in the MRI-brain dataset is about 1.2M (16% of the total number of voxels). Shading took 3.8s, while rendering the splats took 4.2s for the traditional sheet-buffer method and 16.6s for the new method. Sorting took an extra 7s. The nerve dataset had 0.9M (5%) relevant voxels. Shading took 2.9s and rendering took 2.8s vs. 9.0s. (All timings are for a SGI Onyx with InfiniteReality engine.) It is observed that the increased number of (partial) splats is roughly proportional to the increased computation time.

Since our algorithm is best appreciated in animated viewing, the reader may refer to the provided conference CD for animation sequences featuring the nerve cell, the UNC MRI dataset, and a segmented brain dataset. Animations obtained with both sheet-buffer splatting techniques are shown there. These sequences are also available on the World Wide Web under http://www.cis.ohio-state.edu/~mueller/popSplat/pop.html.

6 CONCLUSIONS

In this work, we have tackled a prominent flaw of current splatting methods, i.e., the disturbing color popping artifacts that occur with the traditional, sheet buffer-based splatting technique. In this effort, we first analyzed the origins of these popping artifacts. It was found that the color popping occurs when the orientation of the compositing sheets changes suddenly as the image screen becomes more parallel to another volume face. To cope with these artifacts, we presented a new splatting variant that still uses sheet-buffers, but in a more favorable way, eliminating the popping and reducing color bleeding. In our method, the sheet buffers are always parallel to the image plane, which prevents the main source of the popping artifacts. However, since the sheet buffers are no longer aligned with rows of splats, we need to add slabs of partial kernels within the sheets. This has several advantageous side effects: (i) A voxel kernel is no longer added as a whole, but composited in several parts along the viewing axis. This approximates the volume rendering integral better, yielding more accurate colors and reducing color bleeding even more; (ii) The accuracy of the image and the rendering speed can be controlled by varying the width of the sheets. We could think of an adaptive algorithm in which the width of the sheets is data-driven. This requires that the composition of a few thin kernel sections yields the same color as one thick section, pre-integrated over the same length. To address this issue we have proposed a normalizing scheme for hierarchies of kernel widths of powers of 2. Finally, (iii), the new sheet-based algorithm also has several advantages for splatting in perspective. By dividing the kernels into several sections we can better approximate the depth-dependent linear scaling of the kernel required for anti-aliasing. It also approximates the "diverging ray problem" better: Since kernels are traversed by diverging rays, but the footprint can only hold ray integrals due to parallel rays, errors are committed. The division of the kernels into several parallel footprints allows the rays to approximate a step-wise diverging kernel traversal.

The new sheet-buffer method is more closely related to the discretized volume rendering model than previous splatting meth-

ods. The advantages over raycasting and 3D texture mapping approaches [1][18], which are often used to apply this model, are the added accuracy of continuous ray integrals and the ability to use larger and better interpolation kernels. While Westover originally hinted on this basic idea in his thesis [21], it has not been published or implemented to this date. In this paper, we have shown several unaddressed problems and our solutions for these. We have also shown how the chopped integrals can be pre-computed into footprints and indexed efficiently during rendering.

Although the interpolation complexity of our algorithm is higher than that of the traditional splatting approaches, it is still much smaller than the effort required for raycasting. Our algorithm still performs only 2D interpolations (raycasting does it in 3D), and the number of required slab intervals is far less than the number of sampling points along a ray a raycasting algorithm would need to ensure proper anti-aliasing and integration. Given the much improved image quality we consider the higher complexity of our algorithm, with respect to current splatting approaches, a good investment.

7 FUTURE WORK

We successfully utilized the graphics hardware for efficient footprint mapping and sheet-buffer composition with the aim to improve splatting's rendering quality. In future work we plan to concentrate on more improvement of the rendering speed. For example, the sorting of our list of relevant voxels in the image-aligned sheet-buffer method can be simplified to a bucket-toss, where a bucket contains all splatting kernel slices that are added within a particular slab. Furthermore, one could make the number of kernel slices for an individual voxel dependent on the voxel's opacity. This is motivated by the observation that a voxel of low opacity does not contribute much to the popping, and hence could be splatted as a whole, while voxels of high opacity are more noticeable and should be composited gradually as partial kernels. Another criterion for choosing the number of voxel slices could be the proximity of the voxel center to the slicing plane.

We currently use a Gaussian kernel with a radial extent of 2.0. We plan to switch to the smaller kernel proposed by Crawfis and Max [2] soon. This smaller kernel would reduce the number of partial footprints that must be mapped per voxel.

Finally, we are also studying a mechanism that allows splatting of volume grids with unequal grid scaling without having to interpolate extra volume slices. It relies on a grid warping scheme that transforms the non-cubic grid into a sheared cubic grid, which can then be rendered with spherical splatting kernels as usual.

ACKNOWLEDGMENTS

This work was funded by a grant from the Office of Research at The Ohio State University. We also thank the Ohio Supercomputer Center for the generous use of their equipment, and UNC at Chapel Hill for making available the MRI dataset. Finally, we would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

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

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

[3] R. Crawfis, "Real-time slicing of data-space," *Visualization'96*, pp. 271-276, 1996.

[4] J. Danskin, P. Hanrahan, "Fast algorithms for volume raytracing," *1992 Volume Visualization Workshop*, pp. 91-98, 1992

[5] G.T. Herman, J. Zheng, and C. A. Bucholtz., "Shape-based interpolation," *IEEE Computer Graphics and Applications*, vol. 12, no. 3, pp. 69-79, 1992.

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

[7] G. Knittel and W. Strasser, "A compact volume rendering accelerator", *1994 Workshop on Vol. Vis.*, pp. 67-74, 1994.

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

[9] D. Laur and P. Hanrahan, "Hierarchical splatting: a progressive refinement algorithm for volume rendering," *Computer Graphics*, vol. 25, no. 4, pp. 285-288, 1991.

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

[11] N. Max, "Optical models for direct volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, no. 2, pp. 99-108, 1995.

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

[13] K. Mueller, T. Moeller, J.E. Swan, R. Crawfis, N. Shareef, and R. Yagel, "Splatting errors and anti-aliasing," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 2, pp. 178-191, 1998.

[14] H. Pfister and A. Kaufman, "Cube-4 - a scalable architecture for real-time volume rendering," *1996 Symposium on Volume Visualization*, pp. 47-54, 1996.

[15] T. Porter and T. Duff, "Compositing digital images," *Computer Graphics (SIGGRAPH '84)*, pp. 253-259, 1994.

[16] L.M. Sobierajski and R.S. Avila, "Hardware Acceleration for Volumetric Ray Tracing," *Visualization `95*, pp. 27-34, 1995.

[17] J.E. Swan, K. Mueller, T. Moeller, N. Shareef, R. Crawfis, and R. Yagel, "An anti-aliasing technique for splatting," *Visualization'97*, pp. 197-204, 1997.

[18] A. Van Gelder and K. Kim, "Direct volume rendering via 3D texture mapping hardware," *1996 Volume Rendering Symposium*, pp. 23-30, 1996.

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

[20] L. Westover, "Footprint evaluation for volume rendering," *Computer Graphics (SIGGRAPH'90)*, vol. 24, no. 4, pp. 367-376, 1990.

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

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

[23] R. Yagel, and Z. Shi, "Accelerating volume animation by space-leaping," *Visualization'93*, pp. 63-69, 1993.

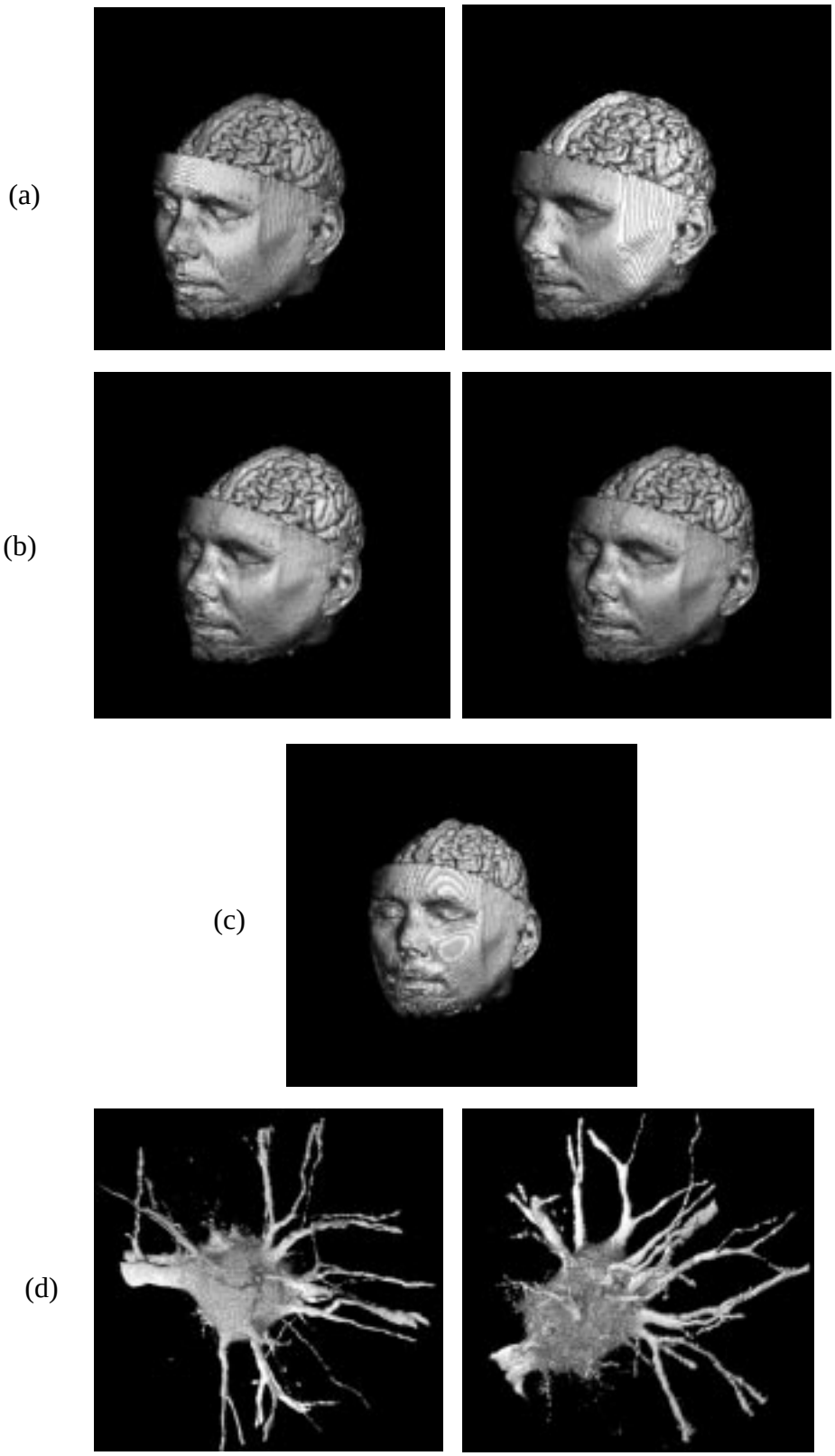


FIGURE 9. UNC MRI-brain dataset rendered (a) with traditional sheet-buffer splatting, (b) with image-aligned sheet-buffer splatting and a slab distance of 1.0, (c) with image-aligned sheet-buffer splatting and a slab distance of 2.0. In (d) a nerve cell is shown, rendered with the new image-aligned sheet-buffer splatting method. (This figure is replicated in the color plates.)