Layered Volume Splatting

Philipp Schlegel and Renato Pajarola

University of Zurich, Binzmuhlestrasse 14, 8050 Zurich, Switzerland
`schlegel@ifi.uzh.ch,pajarola@acm.org`

Abstract. We present a new layered, hardware-accelerated splatting algorithm for volume rendering. Layered volume splatting features the speed benefits of fast axis-aligned pre-classified sheet-buffer splatting while at the same time exhibiting display quality comparable to high-quality post-classified view-aligned sheet-buffer splatting. Additionally, we enhance the quality by using a more accurate approximation of the volume rendering integral. Commonly, the extinction coefficient of the volume rendering integral is approximated by the first two elements of its Taylor series expansion to allow for simple α-blending. In our approach we use the original, exponential extinction coefficient to achieve a better approximation. In this paper we describe the layered splatting algorithm and how it can be implemented on the GPU. We compare the results in terms of performance and quality to prior state-of-the-art volume splatting methods.

Key words: Volume rendering, volume splatting.

1 Introduction

Direct volume rendering [1] is a method for visualizing discrete datasets without extracting explicit geometry. These datasets are often generated by regularly sampling a continuous scalar field. In order to visualize a dataset, the continuous scalar field (3D function) has to be reconstructed from the discrete dataset. Once the reconstruction step is finished, the volume rendering integral needs to be evaluated along the viewing rays. This can be done either in screen or in object space. A popular method is ray casting in conjunction with trilinear interpolation. While ray casting delivers good results, it is more costly to compute and only recent developments have achieved interactive frame rates.

Splatting as an object space method was introduced in [2]. Instead of evaluating every ray from the screen space as with ray casting, each voxel is being illuminated, classified and supplied with a footprint of an interpolation kernel and then projected onto the screen. Due to the inappropriate evaluation of the volume rendering integral, the results suffer from blurring and color bleeding. These issues were addressed by introducing axis-aligned sheet splatting [3]. Post-classified image-aligned sheet splatting has further overcome some drawbacks [4, 17]. Image-aligned approaches typically split the interpolation kernel into several slabs to better approximate the volume rendering integral. Thus for every single voxel, multiple slabs have to be rasterized. In terms of performance the multiplied rasterization costs are a major bottleneck. Our new algorithm limits the number of required splatting operations to exactly one per voxel without losing the quality advantages of splatting multiple kernel slabs per voxel. We achieve this by applying a correction term based on the previous and consecutive sheet. Hence the sheets are not independent from each other anymore and that's why we call a sheet a *layer* and the method *layered volume splatting*.

Furthermore, common approaches to volume rendering make simplifications regarding the evaluation of the volume rendering integral [5, 6]. The integral in its original form cannot be solved analytically without making some confining assumptions and thus needs to be approximated. It is usually developed into a Riemann sum. Moreover, only the first two elements of the Taylor series expansion of the exponential extinction coefficient are taken. This leads directly to Porter-Duff compositing as described in [7] and is well supported in graphics hardware. We think it is now feasible to use the original exponential extinction coefficient, by virtue of fast and programmable GPUs, in order to achieve a closer approximation of the volume rendering integral and thus a better quality.

The contributions of this paper are manifold. First, we introduce a novel, fast, GPU-accelerated volume splatting algorithm based on an axis-aligned layer concept. Second, we provide an effective interpolation correction solution that accounts for the overlap of blending kernels into adjacent layers. Also, we avoid the simplification of the attenuation integral in favor of a more accurate solution. Finally, we demonstrate the superior performance of layered splatting, achieving excellent quality equal to prior state-of-the-art splatting algorithms.

2 Previous Work

Volume splatting was originally introduced by Westover [2]. The algorithm works as follows: Every voxel is mapped from grid into screen space and the density and gradient values are converted into color values (pre-classification). Finally a reconstruction step and the compositing into the framebuffer are performed. Projecting the footprint of an interpolation kernel determines which pixels are affected by a voxel (forward mapping). The algorithm works quite fast but suffers from blurring and color bleeding as a result of pre-classification and improper visibility determination. In [3] a revised algorithm divides the volume into sheets along the axis most parallel to the view direction. The voxel contributions are summed up into a sheet buffer before being composited. However, the algorithm still sticks to pre-classification and pre-shading. Another drawback is the popping artifacts that may occur when the orientation of the sheet direction changes.

Crawfis and Max [8] exploit texture mapping hardware to accelerate the splatting operations and introduce a new reconstruction kernel based on Max' previous 2D optimization [9]. Seminal work on reconstruction and interpolation kernels is provided by Marschner and Lobb [10]. They study several reconstruction filters and classify them according to a new metric that includes smoothing, postaliasing and overshoot. Carlbom [11] provides another discussion about filters. This includes research on weighted Chebyshev approximation and comparisons to piecewise cubic filters. Other quality enhancements have been proposed by Zwicker et al. [12, 13] with their EWA splatting. To avoid aliasing artifacts they introduce a new splatting primitive consisting of an elliptical Gaussian reconstruction kernel with a Gaussian low-pass filter. An anti-aliasing extension including an error analysis of the splatting process has been published by Mueller et al. [14]. Hadwiger et al. [15] investigate quality issues that arise from limited precision and range on graphics hardware when using high-quality filtering with arbitrary filter kernels. In [16] they present a framework for performing fast convolution with arbitrary filter kernels to substitute linear filtering.

Mueller and Crawfis introduce view-aligned sheet splatting in [4]. To overcome the popping artifacts of axis-aligned splatting when switching to a different axis, sheets that are perpendicular to the view direction are used. This requires the voxels to be resorted in every frame where the view direction changes. Because the support radius of the Gaussian interpolation kernel is larger than the distance between two sheets, the kernel is sliced into slabs and for each kernel slice its footprints are generated. This means that for a single voxel several of these footprints have to be splatted, multiplying the display costs. To reduce the blur from splatting, Mueller et al. [17, 18] suggest displacing classification and shading to after projection onto the screen. In addition, not only the density volume is splatted but also the gradient volume. The gradients are required for shading calculations, which now take place after splatting.

Modern, programmable graphics hardware makes it possible to greatly enhance volume rendering and splatting performance. Apart from splatting, there are approaches using 2D and 3D textures for volume rendering [19–21]. 3D texture techniques are generally very fast because of the hardware support but not well suited for high-order interpolation. In turn, splatting can also benefit from fast graphics hardware. Neophytou et al. [22, 23] present a combined CPU/GPU method where bucket distribution of the voxels to the sheets is done on the CPU and compositing, classification and shading on the GPU. The special properties of the Gaussian kernel and footprint enable splatting of four footprint slices at a time using all color channels. Opaque pixels are marked by the shader and henceforth omitted from splatting. Furthermore, using point sprites instead of polygons for splatting reduces the amount of geometry to be sent to the graphics board [22, 24]. Grau et al. [25] extended Neophytou's method to an all GPU algorithm by doing the necessary bucket sorting on the GPU.

Our new layered volume splatting approach uniquely combines the performance advantages of direct (axis-aligned) splatting and hardware acceleration, with the quality improvements of post-classified (sheet-buffered) splatting. The focus of our comparison lies on state-of-the-art splatting methods. Anyhow, compared to 3D texture based volume rendering, we achieve a better image quality due to higher-order interpolation as well as a similar performance in some cases.

3 Layered Volume Splatting

3.1 Performance Considerations

A performance analysis of volume splatting shows three areas where expensive operations may become a bottleneck:

Sorting is necessary to guarantee back-to-front or front-to-back traversal of the splats or sheets. Sheetless approaches such as the original splatting algorithm [2] have different traversal orders for every frame in which the view direction changes. Sheet splatting methods have the advantage that the individual voxels only need to be distributed to the different sheets whereas the order within a sheet is not important. For that reason a cheap bucket sorting algorithm can be employed. When using axis-aligned sheet splatting, the distribution of the voxels to the sheets remains valid as long as a view direction change does not exceed a 45° angle. In this case another axis will become most parallel to the view direction and the sheet orientation changes. The voxels have to be redistributed to the newly oriented sheets. For view-aligned sheet splatting, since the sheets are perpendicular to the view direction, the voxels have to be resorted for every change in view direction. This is a clear disadvantage over axis-aligned sheet splatting. Resorting on the CPU causes a lot of traffic on the bus because each time the whole geometry for the splats has to be transferred to the graphics card. Point sprites can diminish the amount of data sent to the graphics card since only one vertex is required per splat instead of several when using polygons. A recent method by Grau et al. [25] does the resorting completely on the GPU eliminating the need to resend the geometry to the graphics card.

In our layered splatting approach we apply fast axis-aligned ordering such that voxel redistribution only has to be performed when crossing a 45° angle. With normal axis-aligned sheet splatting, popping artifacts may occur when this happens. However, our layered volume splatting strongly abates the popping artifacts, which can be attributed to the use of more compact interpolation kernels and the interpolation correction term, see Section 3.2, as well as to the improved attenuation integration via the exponential extinction coefficient, see Section 3.3.

Rasterization The splatting operation itself is a kind of 2D texture mapping including point sprites. Mostly it is the real bottleneck of volume splatting because texturing of millions of effective splats drives the current graphics cards to the rasterization limits. This applies especially when using sheet-buffered splatting in conjunction with an interpolation kernel that has a large support radius. The kernel then contributes to many sheets, and hence many slabs of a kernel have to be rasterized for a single voxel as shown in Figure 1(a). Neophytou et al. [22] address this issue by using all color channels to splat four kernel slabs at a time. Although this solution is very fast, it has some weaknesses. For one, it only works if the additional color channels are not required for transmitting the normal. Furthermore, it is only well suitable using a Gaussian kernel because individual, pre-integrated kernel slabs can be conveyed from a base kernel by a single factor.

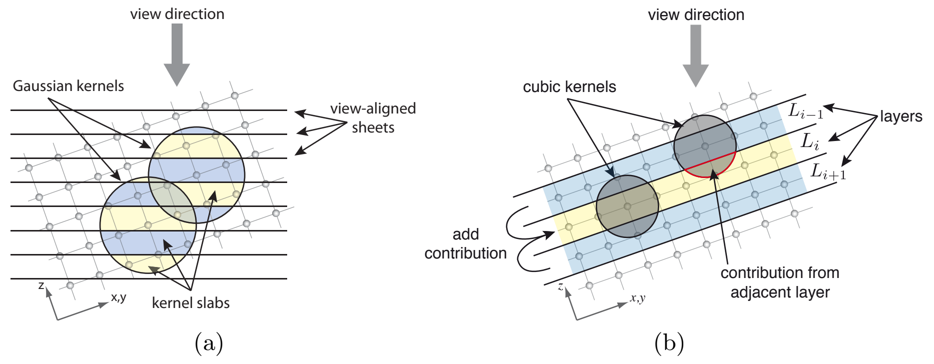


Fig. 1. (a) Sheet buffers are perpendicular to the view direction. Each Gaussian interpolation kernel with radius 2.0 spreads across five sheets resulting in five slabs per kernel at arbitrary position. All slabs are explicitly splat. (b) The voxel grid with a layer overlay and two footprints of our cubic kernel. L_i is the currently processed layer. Contributions from footprints in the adjacent layers are approximated. They are not explicitly splatted as kernel slab footprints when using layered splatting.

Our goal was to strictly have one single texturing operation per voxel including the possibility to provide a normal, either from the gradient volume or a gradient interpolation kernel. To achieve this, we switch from a splat centric view to a sheet centric view and, therefore, call a sheet a *layer*. A layer L_i contains the contributions of all interpolation kernels for which the corresponding voxel centers fall directly into L_i, plus correction terms from interpolation kernels from voxels in adjacent layers as illustrated in Figure 1(b). We define the invariant that only contributions from voxels centered in the current layer L_i are explicitly splatted. Contributions to L_i from voxels in adjacent layers are not explicitly splatted but approximated using a correction term.

To minimize errors introduced by the correction terms, we no longer use a Gaussian interpolation kernel with radius 2.0 which may contribute to five layers. Instead we use a cubic interpolation filter with radius 1.0 that contributes to at most three layers. From a layer centric view this means that only voxels in the current layer plus voxels in the two adjacent layers must be considered. For a given layer L_i and its adjacent layers L_{i-1} and L_{i+1}, only the parts of voxels in L_i are rasterized as splats into the current layer's frame buffer. The missing contributions from adjacent layers L_{i-1} and L_{i+1} are accounted for on a per-pixel basis. This is achieved by accumulating contributions from L_{i-1} and L_{i+1} to the current layer L_i according to the ratio κ of the pre-integrated kernel intersecting L_i, see also Figure 1(b). Consequently the correction addend consists of the contributions from the adjacent layers weighted by the correction factors κ.

The ratio κ may change for every voxel if they do not have the same positions perpendicular to the layers. This is typically the case with view-aligned layers, since in general the volume axes do not align with the view direction. To avoid this, we exploit axis-aligned layers to keep the relative positions of the voxels

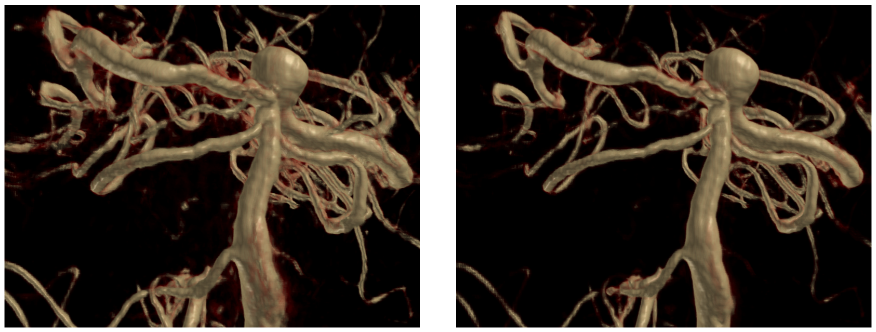


Fig. 2. Right image without using the interpolation correction term shows significant artifacts from missing contributions from parts of interpolation kernels overlapping adjacent layers.

constant within a layer. Thus for a given blending kernel $h(x, y, z)$ we can precompute the correction factors $\kappa(x, y)$ once along the projection dimension since the kernels' intersections with adjacent layers L_{i-1} and L_{i+1} are constant for all voxels, as described in the following section.

Compositing Using per-pixel post-classification and post-shading for high quality rendering, compositing and blending becomes crucial from a performance point of view, especially when the number of sheets or layers rises. It gets even worse if any kind of z-supersampling as in typical sheet based splatting is used to better approximate the volume rendering integral. Let us define the grid resolution of the volume being 1.0 and the distance between two sheets as 0.5. This effectively doubles the required amount of compositing operations but produces a higher quality image, particularly for low-resolution volumes. Huang et al. demonstrate this in their OpenSplat framework [26]. The compositing performance is basically independent from the effective number of voxels or splats as long as no special optimizations are made. Assuming classification and shading is done in a fragment shader, Neophytou et al. [22] show how special OpenGL extensions can be used to optimize performance. Early z-culling and depth-bounds test extensions allow dropping of fragments that are not affected during splatting or which are already opaque in a front-to-back traversal. As we use a different extinction model, we cannot use the default OpenGL blending. Thus we calculate blending within the fragment shader where classification and lighting takes place, and subsequently can take advantage from the same optimizations.

As z-supersampling is not required by our layered splatting approach, it benefits from a reduced number of compositing operations. This is feasible because of compact blending kernels, the interpolation correction terms accounting for adjacent layer contributions, and the improved attenuation factor from the exponential extinction coefficient. Excellent rendering quality is furthermore achieved due to high-resolution interpolation within layers and post-classification.

3.2 Cubic Interpolation Kernel

Because of the discrete resolution of a sampled scalar (or vector) field, the gaps between the sample points must be interpolated for direct rendering and zooming. In other words, a continuous 3D function has to be reconstructed from the available spatial samples. This reconstruction is not only crucial for quality but also for performance. The most common interpolation scheme is the (tri-)linear interpolation that is heavily used in ray casting based volume rendering. In the volume splatting context the Gaussian interpolation kernel is very popular. Apart from the superior quality of the Gaussian kernel over trilinear interpolation, there are some other properties that make it very attractive. The derivative of the Gaussian is a Gaussian again. Further it can be considered spherically symmetric, making it independent from the view direction. Frequently a Gaussian with a support radius of 2.0 is used: $h(r) = [|r| < 2.0]\, c \cdot e^{-2.0r^2}$. However, the Gaussian kernel does not satisfy very well the needs of layered splatting. As only the footprint in the layer where the voxel center lies is explicitly rendered, an error is introduced for every contribution of the kernel that lies outside of that central layer. A Gaussian with radius 2.0 contributes to four additional layers apart from the main layer where the voxel lies. Accordingly, it is better to use a kernel with a smaller support radius. In terms of performance this has an additional benefit. The individual footprint splats are smaller and thus fewer pixels have to be rasterized per footprint, further deferring the rasterization limit.

Interpolation kernel filters can roughly be arranged in three categories: separable filters, spherically symmetric and pass-band optimal discrete filters. The latter are proposed by Hsu et al. [27] and adapted to volume rendering by Carlbom [11]. Because they are quite expensive they are not feasible for fast rendering. Given a 1D kernel $f(r)$, a separable 3D filter can be written as

$$h(x, y, z) = f(x)f(y)f(z), \tag{1}$$

and we use the following 1D function from the family of cubic filters for our purposes

$$f(x) = [|x| < 1.0]\, 1 - 3|x^2| + 2|x^3|. \tag{2}$$

A discussion of cubic interpolation filters can be found in [28] and [10]. We chose this particular filter because it is zero outside a box with edge length 2.0 and subsequently spans exactly the three layers L_{i-1}, L_i and L_{i+1} in a regular voxel grid as indicated in Figure 1(b). Thus the error introduced by layered interpolation along the projection dimension z can significantly be reduced by a correction term. In fact, the integral of the 3D interpolation filter $h(x, y, z)$ of Equation 1 equals zero inside $[-1, 1]$. The correction factors $\kappa(x, y)$ for the correction term can be calculated as follows:

$$\begin{aligned}
\kappa_{L_{i-1}}(x,y) &= \int_{-1}^{-0.5} h(x,y,z)dz \left(\int_{-1}^{1} h(x,y,z)dz\right)^{-1} = 0.09375 \\
\kappa_{L_i}(x,y) &= \int_{-0.5}^{0.5} h(x,y,z)dz \left(\int_{-1}^{1} h(x,y,z)dz\right)^{-1} = 0.8125 \\
\kappa_{L_{i+1}}(x,y) &= \int_{0.5}^{1} h(x,y,z)dz \left(\int_{-1}^{1} h(x,y,z)dz\right)^{-1} = 0.09375
\end{aligned}$$

It shows that κ is independent from the position (x, y) due to the separable characteristics of the kernel. The final interpolated result for layer L_i can be obtained by first splatting the voxels centered in L_i, followed by κ-corrected accumulation of values from layers L_{i-1} and L_{i+1}. Figure 2 demonstrates the effect of the correction term. Note that in contrast to Gaussian interpolation the amount of pixels to be rasterized is reduced by a factor of 4.0 without loss of rendering quality (see also Section 4).

However, there is one disadvantage when using a filter that is not spherically symmetric as it is not independent from the view direction anymore. When splatting the footprints of such kernels, the view direction has to be taken into account and an appropriate footprint has to be selected. From a performance point of view it is not feasible to generate the footprints on the fly during splatting. However, precalculating a set of oriented footprint images, storing them in some small texture cache, and choosing the most suitable in every situation solves the problem. Selecting the right footprint image can be done in a vertex shader program. Currently we use a set of 856 pre-calculated footprint images. Each of these has a size of 64^2 pixels with 1 byte per pixel requiring a total of only 3.3 MByte texture memory. According to our experiments, the discretization of the view direction does not lead to visual artifacts.

3.3 Extinction Coefficient

The goal of every volume rendering algorithm is to approximate the volume rendering integral as closely as possible. Unfortunately, the volume rendering integral cannot be solved analytically without making some confining assumptions [6]. The volume rendering integral is based on the absorption and emission model by Max [5]. I_0 denotes the intensity of the light when it enters the volume. $\tau(t)$ is an extinction coefficient and $E(s)$ is the light emitted by the volume (samples) itself. The integral goes along a viewing ray through the volume calculating the resulting light intensity.

$$I(D) = I_0 e^{-\int_0^D \tau(t)dt} + \int_0^D E(s)\tau(s) e^{-\int_s^D \tau(t)dt} ds$$

The integral is now approximated using a Riemann sum as

$$I(D) \approx I_0 \prod_{i=0}^{D/\Delta t} e^{-\tau(t_i)\Delta t} + \sum_{i=0}^{D/\Delta t} E_i \tau(t_i)\Delta t \prod_{j=i+1}^{D/\Delta t} e^{-\tau(t_j)\Delta t}, \tag{3}$$

and the extinction term is generally developed into a Taylor series as

$$\alpha_j = 1 - e^{-\tau(t_j)\Delta t} \approx 1 - (1 - \tau(t_j)\Delta t) \approx \tau(t_j)\Delta t. \tag{4}$$

Putting Equation 4 into 3 and substituting the emitted light E by the voxel color C results in:

$$I(D) \approx I_0 \prod_{i=0}^{D/\Delta t} (1 - \alpha_i) + \sum_{i=0}^{D/\Delta t} C_i \alpha_i \prod_{j=i+1}^{D/\Delta t} (1 - \alpha_j) \tag{5}$$

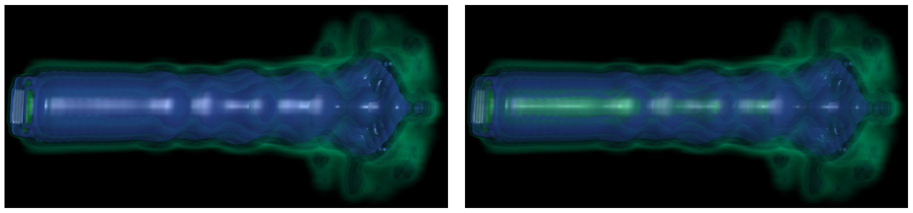


Fig. 3. Comparison between α (left) and the exponential extinction (right).

When inspecting Equation 5 it turns out that this attenuation and blending is equal to the over or under operator (depending on whether back-to-front or front-to-back compositing is used) from Porter and Duff [7]. This makes it attractive because it is very well supported by the hardware. Figure 4(a), however, shows the error introduced by the simplification of taking the first two elements of the Taylor series expansion ($1 - \alpha_j$ in Equation 5) of the extinction coefficient instead of the original exponential function ($e^{-\tau(t_j)\Delta t}$ in Equation 3).

With nowadays powerful and programmable graphics hardware, there is no need to use the α-attenuation but we can return to the original $e^{-\tau}$ extinction coefficient to more closely approximate the volume rendering integral (see also Figure 3). The extinction and blending has therefore to be calculated explicitly in the compositing fragment shader.

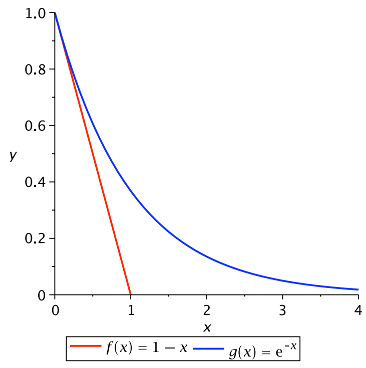
On the other hand, exchanging the classical α with τ implicates a new transfer function. Typically the transfer function maps from the scalar volume field ρ to color and opacity values: $(\rho) \longmapsto (r, g, b, \alpha)$. α hereby denotes the linear opacity where $\alpha = 0.0$ is completely transparent and $\alpha = 1.0$ is fully opaque. Since the α parameter is exchanged by τ the new extinction domain ranges from 0.0 to ∞. Thus existing transfer functions have to be transformed according to this new semantics as follows:

$$f(x) = 1 - e^{-x}$$
$$\alpha \longmapsto \tau = f^{-1}(\alpha) = (1 - e^{-\alpha})^{-1} = \ln\left(\tfrac{1}{1-\alpha}\right)$$

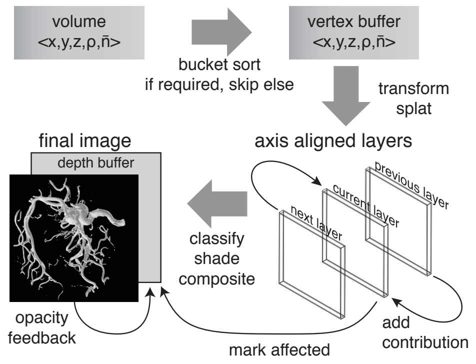
4 Experimental Results

Figure 4(b) shows a brief overview of the rendering pipeline. For each dataset a full rotation over 125 frames with a constant angular step is measured. The average framerate of this rotation is listed in Table 1, all sorting steps are included. The rendering is done in a viewport of 512^2 pixels size. All measurements were made on an Intel Xeon 2.66GHz machine and a Geforce 8800 GT graphics card.

We use the method from [22] as a benchmark. On equal hardware our implementation of their renderer, labeled *view-aligned splatter*, achieves the same or slightly higher framerates. Nevertheless, our layered splatting is able to outperform this well recognized approach by a factor 10 for the large foot dataset and still a factor 2 for the tiny fuel dataset.



(a)



(b)

Fig. 4. (a) Difference between the approximated attenuation term $1-\alpha$ and the original extinction term $e^{-\tau}$. (b) The rendering pipeline of layered volume splatting. The bucket sorting step can be omitted if no 45° angle is crossed.

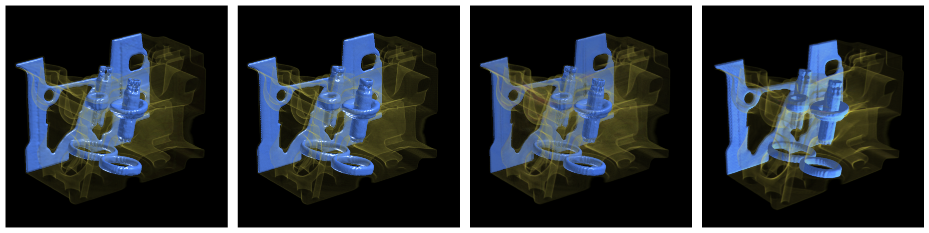


Fig. 5. Left-to-right: axis-aligned splatter, view-aligned splatter, layered splatter, 3D texture slicing.

Figure 5 shows the engine dataset rendered with a transparent transfer function using the renderers from Table 1. The image of the layered splatter shows a quality comparable to the images from the two other splatters but with reduced specular highlights due to the exponential extinction. The image from the 3D texture slicing renderer shows artifacts from the slices on the opaque rear panel.

5 Conclusion

We have described a new volume splatting approach called layered splatting. The goal of this new algorithm is to enhance the performance of existing volume splatting methods while maintaining an excellent display quality. This is particularly useful for interactively visualizing large datasets. Our approach shows tremendous speedups when rendering multimillion-splat datasets while maintaining an excellent quality, i.e. much closer to view-aligned sheet splatting than to sheetless rendering.

Table 1. Performance results

Dataset	Dimension	Effective splats	Axis-Aligned Sheet Splatting	View-Aligned Sheet Splatting	Layered Splatting	Preintegrated 3D Texture Slicing
Fuel Injection	64^3	14K	39.8fps	47.0fps	100.3fps	197.1fps
Lobster	$301 \times 324 \times 56$	233K	15.0fps	13.2fps	43.0fps	
Aneurism	256^3	79K	26.7fps	23.0fps	48.7fps	47.3fps
Neghip	64^3	122K	7.9fps	11.1fps	30.7fps	192.6fps
Engine	$256^2 \times 128$	1.3M	3.8fps	3.0fps	23.2fps	43.0fps
Skull	256^3	1.4M	3.6fps	2.7fps	16.5fps	43.6fps
Foot	256^3	4.6M	0.9fps	0.8fps	8.3fps	40.9fps
Vertebra	512^3	1.6M	3.0fps	2.4fps	14.9fps	

Recently GPU ray casting has become popular and fast, approaching the performance known from 3D texture slicing and splatting. Even though ray casting is known for its good image quality, most implementations of GPU ray casting rely on the built-in trilinear interpolation scheme while sampling along the rays. Having a fast interpolation scheme is crucial for the performance of GPU ray casting. Unlike splatting, implementing an efficient high-order interpolation scheme using GPU ray casting may be more difficult.

References

1. Engel, K., Hadwiger, M., Kniss, J.M., Rezk-Salama, C., Weiskopf, D.: Real-Time Volume Graphics. AK Peters (2006)
2. Westover, L.: Interactive volume rendering. In: VVS '89: Proceedings of the 1989 Chapel Hill workshop on Volume visualization, New York, NY, USA, ACM (1989) 9–16
3. Westover, L.: Footprint evaluation for volume rendering. In Baskett, F., ed.: Computer Graphics (SIGGRAPH '90 Proceedings). Volume 24. (1990) 367–376
4. Mueller, K., Crawfis, R.: Eliminating popping artifacts in sheet buffer-based splatting. In: IEEE Visualization '98 (VIS '98), Washington - Brussels - Tokyo, IEEE (1998) 239–246
5. Max, N.: Optical models for direct volume rendering. IEEE Transactions on Visualization and Computer Graphics **1** (1995) 99–108
6. Moreland, K.D.: Fast High Accuracy Volume Rendering. PhD thesis, The University of New Mexico (2004)
7. Porter, T., Duff, T.: Compositing digital images. In Christiansen, H., ed.: SIGGRAPH '84 Conference Proceedings (Minneapolis, MN, July 23-27, 1984), ACM (1984) 253–259
8. Crawfis, R., Max, N.: Texture splats for 3D scalar and vector field visualization. In: IEEE Visualization '93 Proceedings, IEEE Computer Society (1993) 261–266
9. Max, N.: An optimal filter for image reconstruction. In Arvo, J., ed.: Graphics Gem II, New York, Academic Press (1991) 101 – 104
10. Marschner, S.R., Lobb, R.J.: An evaluation of reconstruction filters for volume rendering. In Bergeron, R.D., Kaufman, A.E., eds.: Proceedings of the Conference on Visualization, Los Alamitos, CA, USA, IEEE Computer Society Press (1994) 100–107

11. Carlbom, I.: Optimal filter design for volume reconstruction and visualization. In Nielson, G.M., Bergeron, D., eds.: Proceedings of the Visualization '93 Conference, San Jose, CA, IEEE Computer Society Press (1993) 54–61
12. Zwicker, M., Pfister, H., van Baar, J., Gross, M.: EWA volume splatting. In Ertl, T., Joy, K., Varshney, A., eds.: Proceedings of the Conference on Visualization 2001 (VIS-01), Piscataway, NJ, IEEE Computer Society (2001) 29–36
13. Chen, W., Ren, L., Zwicker, M., Pfister, H.: Hardware-accelerated adaptive EWA volume splatting. In: Proceedings of IEEE Visualization 2004. (2004)
14. Mueller, K., Möller, T., II, J.E.S., Crawfis, R., Shareef, N., Yagel, R.: Splatting errors and antialiasing. IEEE Transactions on Visualization and Computer Graphics **4** (1998) 178–191
15. Hadwiger, M., Hauser, H., Möller, T.: Quality issues of hardware-accelerated high-quality filtering on pc graphics hardware. In: In Proceedings of WSCG 2003. (2003) 213–220
16. Hadwiger, M., Viola, I., Theußl, T., Hauser, H.: Fast and flexible high-quality texture filtering with tiled high-resolution filters. In: Vision, Modeling and Visualization 2002, Akademische Verlagsgesellschaft Aka GmbH, Berlin (2002) 155–162
17. Mueller, K., Möller, T., Crawfis, R.: Splatting without the blur. In: IEEE Visualization. (1999) 363–370
18. Mueller, K., Shareef, N., Huang, J., Crawfis, R.: High-quality splatting on rectilinear grids with efficient culling of occluded voxels. IEEE Trans. Vis. Comput. Graph **5** (1999) 116–134
19. Rezk-Salama, C., Engel, K., Bauer, M., Greiner, G., Ertl, T.: Interactive volume on standard pc graphics hardware using multi-textures and multi-stage rasterization. In: HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, New York, NY, USA, ACM (2000) 109–118
20. Cabral, B., Cam, N., Foran, J.: Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In: Proceedings ACM/IEEE Symposium on Volume Visualization. (1994) 91–98
21. Xue, D., Zhang, C., Crawfis, R.: isbvr: Isosurface-aided hardware acceleration techniques for slice-based volume rendering. In: Volume Graphics. (2005) 207–215
22. Neophytou, N., Mueller, K.: GPU accelerated image aligned splatting. In Gröller, E., Fujishiro, I., eds.: Eurographics/IEEE VGTC Workshop on Volume Graphics, Stony Brook, NY, Eurographics Association (2005) 197–205
23. Neophytou, N., Mueller, K., McDonnell, K.T., Hong, W., Guan, X., Qin, H., Kaufman, A.E.: GPU-accelerated volume splatting with elliptical RBFs. In Santos, B.S., Ertl, T., Joy, K.I., eds.: EuroVis06: Joint Eurographics - IEEE VGTC Symposium on Visualization, Lisbon, Portugal, 8-10 May 2006, Eurographics Association (2006) 13–20
24. Higuera, F.V., Hastreiter, P., Fahlbusch, R., Greiner, G.: High performance volume splatting for visualization of neurovascular data. In: IEEE Visualization, IEEE Computer Society (2005) 35
25. Grau, S., Tost, D.: Image-space sheet-buffered splatting on the gpu. In: IADIS 07, International Conference on Computer Graphics and Visualization 2007. (2007) 75–82
26. Huang, J., Crawfis, R., Shareef, N., Mueller, K.: Fastsplats: optimized splatting on rectilinear grids. In: IEEE Visualization. (2000) 219–226
27. Hsu, K., Marzetta, T.L.: Velocity filtering of acoustic well logging waveforms. IEEE Trans. Acoustics, Speech and Signal Processing **ASSP-37** (1989) 265
28. Mitchell, D.P., Netravali, A.N.: Reconstruction filters in computer graphics. In: Proceedings ACM SIGGRAPH. (1988) 221–228