Eurographics Symposium on Parallel Graphics and Visualization (2020)
S. Frey, J. Huang, F. Sadlo (Editors)

Moment-Based Opacity Optimization

M. Zeidan[1], T. Rapp[1], C. Peters[1] and C. Dachsbacher[1]

[1]Computer Graphics Group
Karlsruhe Institute of Technology, Germany

Abstract

Geometric structures such as points, lines, and surfaces play a vital role in scientific visualization. However, these visualizations frequently suffer from visual clutter that hinders the inspection of important features behind dense but less important features. In the past few years, geometric cluttering and occlusion avoidance has been addressed in scientific visualization with various approaches such as opacity optimization techniques. In this paper, we present a novel approach for opacity optimization based on recent state-of-the-art moment-based techniques for signal reconstruction. In contrast to truncated Fourier series, moment-based reconstructions of feature importance and optical depth along view rays are highly accurate for sparse regions but also plausible for densely covered regions. At the same time, moment-based methods do not suffer from ringing artifacts. Moreover, this representation enables fast evaluation and compact storage, which is crucial for per-pixel optimization especially for large geometric structures. We also present a fast screen space filtering approach for optimized opacities that works directly on moment buffers. This filtering approach is suitable for real-time visualization applications, while providing comparable quality to object space smoothing. Its implementation is independent of the type of geometry such that it is general and easy to integrate. We compare our technique to recent state of the art techniques for opacity optimization and apply it to real and synthetic data sets in various applications.

Categories and Subject Descriptors (according to ACM CCS): I.6.9 [Computer Graphics]: Picture/Image Generation—Viewing algorithms

1. Introduction

Flow visualization is a branch of scientific visualization where scientists and engineers are interested in exploring various phenomena on 2D manifolds or inside 3D vector fields. Lines and surfaces [MLP*10a, ELC*12] are used by domain experts to detect and classify important features inside the domain [BCP*12]. Such geometric structures are generated by integrating steady and unsteady flows at specific or predefined locations. However, most of the time a direct visualization results in a large number of lines and surfaces. Without an appropriate visualization technique, cluttering might be a problem, where important structures are surrounded and hidden by less important structures.

In order to better study dense and cluttered flow structures, an efficient visualization technique seeks to find a smaller, but representative set of flow structures to preserve important features in a fast and efficient way. One possible way to find a representative set of lines is to use a suitable seeding algorithm, where carefully chosen seed points are used to start the line integration. Good seeding points for line integration can be found via density-based estimates of local importance [MTHG03, SHH*07], as locations of interesting features [YKP05, YWSC11], or based on a similarity measure of flow structures [MLP*10b, MJL*12].

An alternate approach to visualize important flow structures is to optimize the opacity of each geometric structure based on its importance. This class of algorithms models the opacity of each rendered segment using an optimization problem and tries to maximize the opacity of important structures inside cluttered regions while minimizing the opacity of less important surrounding structures. Günther et al. model the opacity optimization as a global optimization process for 3D lines [GRT13, GRT14] and surfaces [GSE*14]. Lastly, a fast per-pixel opacity optimization for points, lines, and surfaces [GTG17] finds an analytic solution for opacity in screen space, independently for each pixel. This is followed by an object space smoothing step. However, such techniques come at the cost of a large memory footprint due to the generation of a per-pixel linked list on the GPU, and are relatively slow due to the need for sorting a large number of linked lists in screen space.

In this paper, we present a novel technique for opacity optimization using a moment-based representation. Moment-based reconstructions [PK15, MKKP18] provide the means to recover monotonic functions from a small number of coefficients. In our case, they help recover the sum of squared importance values of surfaces that contribute to the occlusion of any given fragment. They are highly accurate for sparse regions in the visualization, where only few structures are visible, and smooth and plausible for dense re-

DOI: 10.2312/pgv.20201072

gions. Our approach enables screen-space filtering of this occlusion information. Thus, no filtering in object space is needed. These desirable properties lead to an efficient, reliable and general opacity optimization technique. Our technique is not limited by the geometric representation of input data and can be directly applied to any kind of geometry that fits into the rasterization pipeline.

For compositing, we use moment-based order-independent transparency [MKKP18], which scales well to scenarios with strong overdraw. A recent work of Baeza Rojo et al. [BGG19] introduces a similar approach with a truncated Fourier series for reconstruction. The basic steps of their method for opacity optimization are similar, but prior work on order-independent transparency found that moment-based reconstructions are more robust than truncated Fourier series [JB10, SML11, MKKP18]. Besides, their technique still relies on object space filtering.

Our main contributions are:

- An application of moment-based approaches from prior work [MKKP18] to the problem of opacity optimization for points, lines, and surfaces,
- An efficient screen space filtering approach that works directly on the moment buffers used for opacity optimization,
- An extensive comparison and analysis of our approach with current state of the art solutions [GTG17, BGG19].

This paper is organized as follows: Section 2 introduces related work. Section 3 presents our approach for opacity optimization using a moment-based formulation and introduces our screen space filtering. Section 4 describes the integration into our visualization framework. Section 5 compares our approach to state of the art solutions for opacity optimization. We conclude with Section 6.

2. Related Work

In this section, we discuss recent work on occlusion avoidance, opacity optimization and rendering of semi-transparent objects.

2.1. Transfer Functions

Transfer functions are an essential tool in visualization to highlight important parts through high opacity. Since manual design of transfer functions is a time-consuming task, there are many automatic and semiautomatic techniques. For example, these employ geometric derivatives [PRW11], surface curvature [HKG00, KSW06], visibility histograms [CM10] and information theoretic strategies [RBB*11, SP13]. However, the core concept of mapping a scalar attribute to opacity, is quite limiting. Either important structures remain occluded or less important structures are removed everywhere, even when they do not occlude anything important. Opacity optimization [Gün16] overcomes these limitations.

2.2. Decoupled Opacity Optimization

Opacity optimization is a selection algorithm that highlights important geometric structures. It adjusts opacities to ensure that dense but unimportant structures become transparent whenever they occlude more important structures. Thus, it reduces visual clutter. For streamlines, an importance value is assigned to each segment and the optimization process finds the optimal opacity value for each segment. Importance values of geometric structures are derived from physical properties of the data such as density, geometric properties such as curvature, or based on user controlled parameters. Decoupled opacity optimization as originally proposed by Günther et al. [GRT13, GRT14] is a global least squares minimization process for line geometry within a 3D vector field. Later the technique was extended to surface geometry [GSE*14]. Ament et al. [AZD17] extend and solve such an optimization process in ray space for volumetric data. Günther et al. [GTG17] extend this approach to points, lines, and surfaces and perform the optimization process in parallel for each pixel on the GPU.

Since the analytic solution proposed by Günther et al. [GTG17] forms the basis for our formulations, we briefly recapitulate it here. Consider a single pixel that is covered by $n \in \mathbb{N}$ fragments. Fragment $l \in \{0, \ldots, n-1\}$ is associated with a depth $z_l \in [z_{\min}, z_{\max}]$ and an importance $g_l \in [0,1]$. An A-buffer [Car84] makes all of this information available using per-pixel linked lists, albeit at a high cost. To compute an optimized opacity for fragment l, we need the sum of squared importance values up to depth z_l and the full sum:

$$G(z_l) := \sum_{\substack{k=0 \\ z_k < z_l}}^{n-1} g_k^2, \qquad G_{\text{all}} := \sum_{k=0}^{n-1} g_k^2. \tag{1}$$

For each pixel in screen space, the opacity optimization algorithms finds the optimized opacity for fragment $l \in \{0, \ldots, n-1\}$ along the view ray as

$$\alpha_l := \frac{p}{p + (1-g_l)^{2\lambda} \left(rG(z_l) + q(G_{\text{all}} - G(z_l) - g_l^2) \right)}, \tag{2}$$

where $p, q, r, \lambda \geq 0$ are user controllable input parameters with the following purpose:

- p is a regularization term that prevents empty renderings and controls how close opacities get to one,
- q penalizes foreground clutter,
- r penalizes background clutter,
- λ steers the fall-off of importance from 1, which allows the user to put further emphasis on the important structures.

Decoupled opacity optimization [GTG17] solves Equation 6 using per-pixel linked lists that are sorted by depth for opacity optimization. This per-pixel optimization is followed by a geometric filtering pass to smooth different opacity values out of the optimization process along adjacent vertices. This parallel solution comes at the cost of a relatively large memory footprint due to the creation of per-pixel linked lists. Moreover, sorting of the lists gets prohibitively slow for dense geometry.

Fourier Opacity Optimization [BGG19] solves the opacity optimization process in the frequency domain using a Fourier series approximation of the importance function along the view ray. By doing so, per-pixel linked lists are replaced by a relatively small number of frame buffers for storing the Fourier coefficients. The cumulative importance per fragment along the view ray $G(z_l)$ is reconstructed using a truncated Fourier series. Besides the large memory reduction, a large processing gain is achieved by replacing sorting with the order-independent Fourier series reconstruction. The output is shown to be a plausible approximation of the reference solution using linked lists.

2.3. Order Independent Transparency

In this paper, we transfer methods from moment-based order-independent transparency [MKKP18] to opacity optimization. These two problems are orthogonal. Opacity optimization computes appropriate opacity values, thus providing input to the order-independent transparency, which takes care of correct compositing of transparent surfaces. Nonetheless, the problems have commonalities since both deal with an arbitrary number of fragments per pixel, each associated with a scalar attribute (either importance or opacity). Both of these fields have seen approaches using A-buffers [Car84, GTG17] and truncated Fourier series [JB10, BGG19] but thus far moment-based reconstructions have only been used for order-independent transparency.

With carefully optimized GPU implementations [YHGT10], A-buffers [Car84] are the most viable exact solution for order-independent transparency. The linked list of all fragments covering a pixel is constructed in a single rasterization pass using atomic reads and a large preallocated node buffer. Subsequently, each list is sorted front to back in a shader. Once the fragments are sorted, standard alpha blending suffices to composite them together correctly. In spite of the optimization efforts [YHGT10], A-buffers are still expensive and scale poorly to scenarios with strong overdraw.

Fourier opacity mapping [JB10] uses a truncated Fourier series to encode the transmittance from a light source to any point in a scene. It stores Fourier coefficients per pixel of a shadow map. Although this work focuses on shadows, it has been evaluated for order-independent transparency as well with unsatisfactory results on moderately large scenes [SML11]. Fourier opacity optimization [BGG19] also uses this approach for order-independent transparency.

Moment-based order-independent transparency [MKKP18] goes back to a different technique for shadow mapping [PK15]. Like Fourier opacity mapping, it stores a small number of coefficients per pixel (typically five or seven). They are referred to as moments and a non-linear reconstruction method uses them to recover the monotonic function that maps depth values to transmittance. Once the transmittance is known, correct compositing only requires additive blending. We provide further details throughout Section 3. The used reconstruction method offers nearly perfect results when there are few fragments covering a pixel. It also handles large sparsely populated depth ranges well. In contrast, a truncated Fourier series suffers from severe ringing in such cases. The authors also propose an alternative to Fourier opacity mapping that compares favorably for rendering shadows.

3. Moment-Based Opacity Optimization

In this section, we introduce our novel technique for opacity optimization. Figure 1 shows the main processing steps of our method. It computes optimized opacities per fragment for arbitrary rasterized geometry. To this end, it requires a representation of the importance of each fragment covering a pixel. Decoupled opacity optimization [GTG17] uses an A-buffer [Car84], which incurs a high overhead on graphics hardware [YHGT10]. We implement a more efficient approach by adopting approximations from moment-based OIT [MKKP18].

Our technique optimizes opacities in two passes. The first renders all transparent geometry to moment buffers to gather the importance of all fragments per pixel (Section 3.1). The second renders all transparent geometry again and uses the information from the first pass to perform the opacity optimization per fragment (Section 3.2). In our case, compositing uses moment-based order-independent transparency, which introduces a third pass (Section 4.2). We also implement an A-buffer as reference solution.

3.1. Accumulation of Importance

As described in Section 2.2, we are interested in a pixel with $n \in \mathbb{N}$ fragments having importance values $g_0, \dots, g_{n-1} \in [0,1]$ and depth values $z_0, \dots, z_{n-1} \in [z_{\min}, z_{\max}]$. A-buffers are costly because all of this information is made available explicitly. To optimize the opacity of fragment $l \in \{0, \dots, n-1\}$, we need to know

$$G(z_l) = \sum_{\substack{k=0 \\ z_k < z_l}}^{n-1} g_k^2. \tag{3}$$

Thus, the function $G(z)$ with $z \in [z_{\min}, z_{\max}]$ provides all the information that we need. It is constant between any two adjacent fragment depths and increases by g_l^2 at depth z_l.

To arrive at a faster opacity optimization technique, we seek a compact approximate representation of this monotonic function. Construction of this representation has to be efficient because it happens per frame and per pixel. It has to be robust, even when fragments are strongly localized in small parts of the depth range but should also work for more uniform distributions. Moment-based OIT [MKKP18] faces similar challenges and offers a compelling solution. Thus, we opt to represent the function $G(z)$ by a small number of moments.

The moments describe the accumulated importance $G(z_l)$ as a function of depth z_l. Therefore, the accuracy depends on the definition of depth values. For the sake of constant relative accuracy across the depth range, moment-based OIT warps depth values in a logarithmic fashion. We employ the same approach, i.e. we define

$$z_l' := \frac{\log z_l - \log z_{\min}}{\log z_{\max} - \log z_{\min}} 2 - 1 \in [-1, 1],$$

where $l \in \{0, \dots, n-1\}$. For this formula to be meaningful, z_l should be the linear view-space depth and $z_{\min}, z_{\max}$ should be reasonably tight bounds. We compute $z_{\min}, z_{\max}$ using a bounding box of the geometry and the near clipping plane.

Next, we define the moment of order $j \in \{0, \dots, m\}$ as

$$b_j := \sum_{k=0}^{n-1} g_k^2 \mathbf{b}_j(z_k'), \tag{4}$$

where the moment-generating function $\mathbf{b}_j$ is either

$$\mathbf{b}_j(z_k') := (z_k')^j \in \mathbb{R}$$

for power moments or

$$\mathbf{b}_j(z_k') := \exp\left(\left(2\pi - \frac{\pi}{10}\right) i j \frac{z_k' + 1}{2}\right) \in \mathbb{C}$$

for trigonometric moments. Power moments offer a faster but less

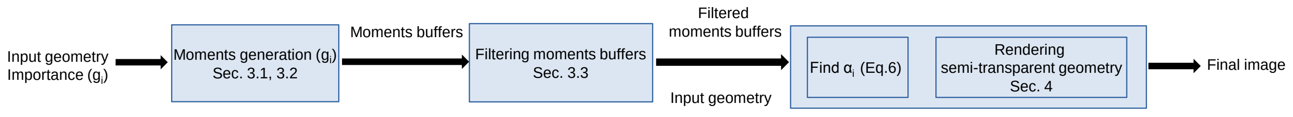


Figure 1: *Overview of our technique.*

accurate approximation. Since Equation (4) defines moments as summation over all fragments, a single rendering pass with additive blending lets us compute all moments. We simply compute the warped depth z'_k in a fragment shader and output $\mathbf{b}_j(z'_k)$ for all $j \in \{0,\dots,m\}$ to the channels of our render targets. Note that

$$b_0 = \sum_{k=0}^{n-1} g_k^2 \mathbf{b}_0(z'_k) = \sum_{k=0}^{n-1} g_k^2 = G_{\text{all}} \tag{5}$$

holds the total importance.

Viable values for the maximal order m are four or six for power moments and three or four for the complex-valued trigonometric moments. Typical visualization datasets have a high depth complexity. Therefore, we select a high-quality variant of the moment-based reconstructions. We use three trigonometric moments stored in seven single precision floats for both opacity optimization and OIT.

3.2. Opacity Optimization

Decoupled opacity optimization [GTG17] defines the optimized opacity for fragment $l \in \{0,\dots,n-1\}$ as

$$\alpha_l := \frac{p}{p + (1-g_l)^{2\lambda}\left(rG(z_l) + q(G_{\text{all}} - G(z_l) - g_l^2)\right)} \tag{6}$$

where $p,q,r,\lambda \geq 0$ are user-defined parameters. We obtain the exact G_{all} from the buffer storing the zeroth moment. The only remaining challenge is to get an approximation of $G(z_l)$ as defined in Equation (3). For this purpose, we rely on the moment-based reconstructions used by moment-based OIT [MKKP18].

These reconstructions use the moments $b_0,\dots,b_m$ to compute two approximations to $G(z_l)$. In the case of power moments, one is guaranteed to be less than $G(z_l)$ while the other is always greater. These approximations are proven to be the best possible lower and upper bounds to $G(z_l)$, considering that we only know the moments $b_0,\dots,b_m$ [PK15]. Since we want an approximation to $G(z_l)$ that excludes g_l^2 but do not want to be overly conservative, we weight the lower bound with 75% and the upper bound with 25%. This strategy is proven to work well in moment-based OIT [MKKP18]. For trigonometric moments the two approximations have a different meaning but are used in the same way. Note that the code published with moment-based OIT includes conversions to transmittance by taking the exponential of the reconstruction. Since we want to accumulate importance additively rather than multiplicatively, these are not needed here. Other than that, the reconstruction code applies to opacity optimization without modification.

Thus, we have the means to implement opacity optimization in a shader. The shader reads the moments from the render targets of the first pass, reconstructs $G(z_l)$ as described above and then evaluates Equation (6) to compute the optimized opacity α_l. The remaining problems are to filter these opacities (Section 3.3) and to perform order-independent compositing (Section 4.2).

3.3. Screen Space Filtering for Smoothed Opacity

Since decoupled opacity optimization [GTG17] optimizes opacities per pixel, the results are discontinuous at first. To alleviate this artifact, minimal opacity values are stored per segment. Then an object space filtering, namely Laplacian smoothing, is applied to smooth the opacity across adjacent segments [GTG17, BGG19]. The smoothing requires knowledge of the topology of the geometry and the implementation in a compute shader is non-trivial. Points, lines and surfaces need to be handled differently and additional data structures need to be implemented.

We propose a more general approach with a less intricate implementation. Our technique allows us to smooth the moment buffers directly. Recall that each moment b_j is stored in a channel of a texture for each pixel on screen. We simply apply a two-pass Gaussian blur to each moment individually. Such blurs are a standard operation in graphics and highly optimized implementations exist. Since moments depend on the function $G(z)$ linearly, this linear filtering effectively smoothes the function values $G(z)$ across pixels. The result differs from linear filtering of opacities for three reasons. Firstly, the moment-based reconstruction is non-linear and thus the errors of the approximation change in a non-linear fashion as we apply the filtering. Secondly, Equation (6) is a rational function with a non-linear dependence on $G(z_l)$. Finally, we use a fixed screen space filter size rather than performing the filtering in object space. Thus, the radius around important structures where clutter is removed is constant in screen space. In spite of these differences, our approach achieves the core goal of filtering in opacity optimization: The removal of clutter is no longer limited to fragments that occlude an important structure directly. Instead, nearby structures are also made transparent with a smooth transition.

4. Implementation

In the following, we describe our visualization framework briefly and detail the integration of moment-based opacity optimization into the renderer. In particular, we discuss the combination with moment-based OIT and A-buffers. Our visualization framework uses OpenGL and we evaluate it on a machine with an NVIDIA Geforce 1080 Ti. All aspects of the rendering are GPU-accelerated using either the rasterization pipeline or compute shaders.

4.1. Generating Visualization Primitives

To demonstrate the generality of our approach, we work with data sets consisting of points, lines and arbitrary triangle meshes. Our line data sets consist of streamlines. We use explicit Euler integration in a preprocessing step to obtain them from velocity fields. Seed points are placed uniformly at random within a user-defined volume. To define the importance for each line, we either compute the average line curvature [Bob08] or we use the velocity at this point in the field. Adjacency information is available for lines only. Thus, the Laplacian smoothing used by decoupled opacity optimization is not supported for points or triangle meshes in our implementation. However, our screen space filtering is applicable to all types of geometry.

4.2. Moment-Based OIT

Our method for opacity optimization is compatible with any method for OIT. We have implemented a reference solution using an A-buffer as well as moment-based OIT [MKKP18]. In the following, we briefly recapitulate how moment-based OIT works and explain how to combine it with our opacity optimization.

Moment-based OIT renders all transparent geometry twice. The first pass serves to generate moment buffers much like the ones that we use for opacity optimization. This time, moment $j \in \{0, \dots, m\}$ is defined as

$$b'_j := \sum_{k=0}^{n-1} -\log(1-\alpha_k)\mathbf{b}_j(z'_k). \tag{7}$$

Using the reconstruction methods described above, the moment buffers allow us to approximate the transmittance at depth z_l, which is given by

$$T(z_l) := \prod_{\substack{k=0 \\ z_k<z_l}}^{n-1} (1-\alpha_k) = \exp \sum_{\substack{k=0 \\ z_k<z_l}}^{n-1} \log(1-\alpha_k).$$

The sum inside the exponential is estimated from the moments. This way, the second pass estimates the transmittance for each fragment and multiplies the fragment color by transmittance and opacity before adding it to the pixel color using additive blending.

In the context of our opacity optimization, this means that we render transparent geometry that requires opacity optimization three times (Figure 1). All three passes use additive blending. The first pass implements Equation (4) to compute the moments needed for opacity optimization. It renders to multiple render targets holding single-precision floats. For three trigonometric moments, they have one, two and four channels for a total of seven. Screen space filtering applies to the moments produced by this first pass. It is implemented as separable Gaussian blur in two fragment shader passes exploiting hardware acceleration for bilinear filtering [Wol18]. The second pass computes optimized opacities α_l for each fragment and implements Equation (7) to compute the moments needed for OIT. The third pass reads from both moment buffers to compute the optimized opacity once more as well as the appropriate transmittance. It also implements shading and multiplies the fragment color by transmittance and opacity. A final pass executes one thread per pixel to composite the foreground with the background color using b'_0.

4.3. A-Buffers

Our implementation also supports A-buffers. They allow us to implement decoupled opacity optimization and give us a ground truth for OIT. Their GPU implementation follows along the lines of [YHGT10]. Thus, construction of the whole A-buffer only requires a single draw call that fills the per-pixel linked lists. To combine our moment-based opacity optimization with OIT using an A-buffer, we still generate the moments in an additive render pass and filter them. One thread per pixel loads each linked list, sorts it by depth and performs blending using opacities that are computed on the fly based on the moment-buffers. In the same manner, we can use the A-buffer to compute optimized opacities according to Equation (6) on the fly. This approach is not the same as decoupled opacity optimization because it does not filter opacities in any way. Screen space filtering is not applicable to A-buffers. However, it is useful for a validation of our moment-based approximation that we perform in Section 5.2.

Decoupled opacity optimization [GTG17] takes a different route. After construction of the A-buffer, one thread per pixel loads each linked list, sorts it by depth and performs opacity optimization for each fragment. The A-buffer stores indices of line segments per fragment, which allow us to store the minimal opacity encountered for each segment using atomic instructions. Once these minimal opacities are known, a compute shader filters them along the line. Since the filter kernel of repeated Laplacian smoothing of lines eventually approximates a Gaussian kernel, we perform this smoothing in a single pass with a Gaussian filter. With these opacities available, we load and sort all linked lists once more to perform blending.

4.4. Fourier Opacity Optimization

Fourier opacity optimization [BGG19] proposes to use the same object space filtering as decoupled opacity optimization. Since it does not use an A-buffer, this implies rendering all geometry four times, twice for opacity optimization and twice for OIT. Our approach with screen space filtering only takes three geometry passes because opacity optimization is merged into OIT. For the sake of a more fair comparison, our implementation of Fourier opacity optimization also uses screen space filtering. In our experiments, we use nine real Fourier coefficients for Fourier opacity optimization, because the authors reported artifacts when using fewer coefficients [BGG19].

5. Results

In this section, we compare our moment-based opacity optimization (MBOO) to decoupled opacity optimization (DOO) and Fourier opacity optimization (FOO) on a number of datasets featuring points, lines and surfaces. We begin by assessing how well the techniques highlight important structures (Section 5.1). Next we perform a direct comparison of all techniques without filtering to assess the approximation error of the moments (Section 5.2). An in-depth analysis of the effect of our screen space filtering follows (Section 5.3). Finally, we discuss run times of all tested techniques (Section 5.4).

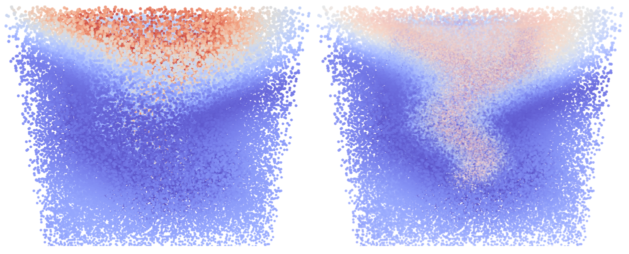


Figure 2: *The tornado vector field visualized using 100 k points. Colors and importance are derived from the velocity magnitude. Rendering uses moment-based OIT. Note how opacity optimization reveals the vortex in the center but keeps unimportant structures elsewhere.*

5.1. Quality of Opacity Optimization

We begin with the simple point dataset in Figure 2 to observe the basic functionality of our opacity optimization. The points are distributed uniformly at random in a synthetic tornado vector field of resolution 128^3 [MCHM10]. There are interesting structures in the central vortex, which should not be obscured by points for the slowly moving surrounding air. Our opacity optimization successfully reveals these structures without removing unimportant structures elsewhere.

Figure 3 provides a more comprehensive evaluation with challenging streamline datasets. All datasets use the average curvature to derive the importance of lines for opacity optimization. The transfer function visualizes the magnitude of the velocity. In the first row, we show streamlines for the tornado dataset used above. Lines of high curvature mostly occur in the central vortex and all opacity optimization techniques succeed in revealing those.

The second row shows magnetic field lines in the decay of magnetic knots [CB11]. We show a close-up of one of the two symmetric rings in this dataset. Without opacity optimization, it is hidden by foreground clutter. Opacity optimization reveals this ring and also emphasizes other lines of high curvature. The Rayleigh-Bénard convection shown in the third row arises when a thin layer of fluid is heated from below. Opacity optimization achieves good visibility for the resulting convection cells but keeps slower flows where they do not occlude important structures. The trefoil knot dataset [CB11] in the fourth row is another magnetic field. It contains three interlocked magnetic rings that decay over time and opacity optimization emphasizes the structures near these rings.

In all visualizations, opacity optimization greatly improves the clarity. Results of Fourier opacity optimization and our moment-based opacity optimization are similar. The largest differences are in the second row where Fourier opacity optimization fails to remove some foreground clutter at the left top. Results of decoupled opacity optimization differ substantially from our technique because of the differences in filtering. The filter size of the object space filtering on screen depends on the tesselation of the lines and the perspective. In most cases, it winds up being larger than our screen space filter. At the size at which the results are shown in the figure, this may provide greater clarity. However, our screen space filter with a footprint of 21×21 pixels provides details at a finer scale, which convey more information when the image is observed in fullscreen. The size of the filter on screen is intuitive to control and efficient to apply.

5.2. Validation Against Ground Truth

To compare decoupled opacity optimization to the other techniques more directly, we disable filtering. Decoupled opacity optimization is done on the fly during OIT compositing as described in Section 4.3. Screen space filtering is disabled as well. As importance measure, we use the velocity magnitude such that the importance is visible through the transfer function. In this setting, decoupled opacity optimization becomes our ground truth and all differences are errors introduced by the truncated Fourier series or the moment-based reconstruction. Both techniques come close to ground truth. The greatest errors occur in regions with important lines near unimportant lines and strong overdraw. In these regions, the advantage of moment-based reconstructions diminishes. The error of our technique is slightly greater than that of Fourier opacity optimization. Note however, that Fourier opacity optimization uses nine coefficients, whereas our technique uses seven.

5.3. Screen Space Filtering

Since screen space filtering for opacity optimization is novel in our work, we provide a more extensive comparison to alternatives in Figure 5. Without filtering, foreground clutter is only removed if it directly occludes important structures. Thus, there is no visible separation between clutter and important lines, which leads to an unclear visualization. The object space filtering of decoupled opacity optimization addresses this problem. However, the apparent size of the filter on screen is difficult to control and the implementation of the method is complicated. Our screen space filtering removes foreground clutter in a region whose size is fixed on screen (21×21 pixels in this case). In situations with a greater density of geometry (Figure 5 bottom), it adds detail to the visualization that aids the perception of the streamlines.

Figure 6 demonstrates another benefit of our screen space filtering. We use two triangle meshes with complicated topology. The bunny and the bush have constant importance of 0.9 and 0.2, respectively. Computing the adjacency information that is needed for Laplacian smoothing would be difficult since the meshes are not designed with such requirements in mind. Nonetheless, our screen space filtering is trivially applicable without adding anything special to the implementation. Any geometry that can be rendered by the rasterization pipeline is compatible with our method. Thus, we get a good opacity optimization result in a situation that would be difficult to address with decoupled opacity optimization.

5.4. Performance Evaluation

Table 1 provides timings for most figures measured on an NVIDIA Geforce 1080 Ti. All images are rendered at a resolution of 1920×1080. The timings where an A-buffer is used for OIT are less informative because of its high overhead. Thus, we focus our analysis

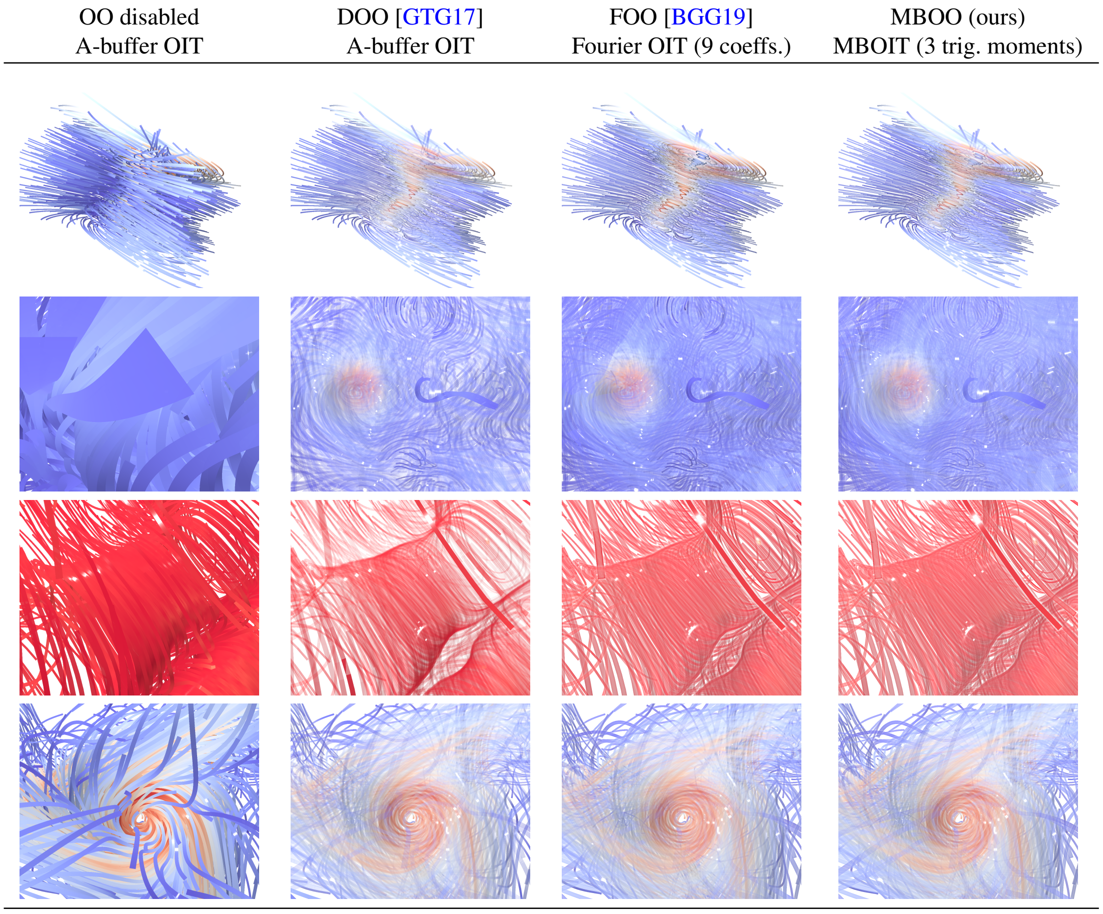


Figure 3: *Comparison of three opacity optimization techniques using four challenging streamline datasets: Tornado, Borromean rings, Rayleigh-Bénard convection and trefoil knot. Note the overall improvement in clarity through opacity optimization and the differences in filtering between DOO and FOO or MBOO. Our technique achieves comparable quality to DOO at a substantially lower cost.*

Table 1: *Total frame times in milliseconds and primitive counts for the results in the figures above. Our techniques are marked with an asterisk. The fastest technique in each comparison is marked bold. Note that our technique clearly outperforms decoupled opacity optimization and performs similar to Fourier opacity optimization.*

Dataset	Figure	Geometry			Run-time			
		Num. lines	Max. num. segments	Num. vertices				
Tornado	Fig. 3 first row	1728	110592	691200	6.8	12.0	5.8	**5.6***
	Fig. 5 top	512	204800	32768	3.1	3.7		
					2.2*	**3.0***		
	Fig. 5 bottom	4096	1638400	262144	32.5	35.7		
					23.1*	**24.0***		
Borromean	Fig. 3 second row	2845	182080	2845000	60.0	100.0	30.5	**26.9***
Rayleigh-Bénard	Fig. 3 third row	863	55232	161721	25.5	40.9	**7.9**	8.2*
TrefoilKnot	Fig. 3 fourth row	1293	82752	258600	18.7	31.3	**8.7**	9.4*
	Fig. 4				32.2	21.2	**20.9***	

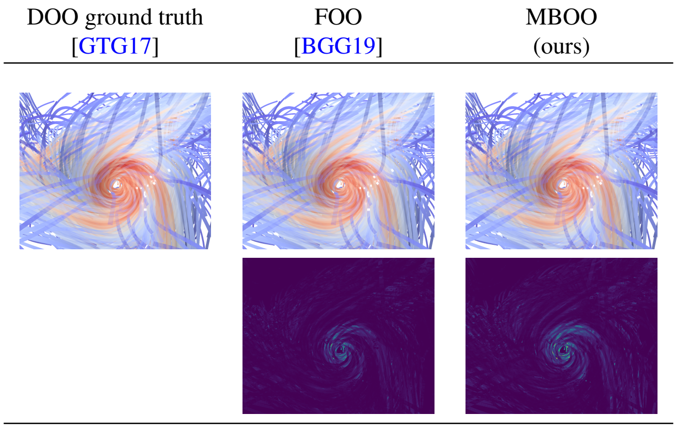


Figure 4: *A comparison of opacity optimization techniques on the trefoil knot with both object space filtering and screen space filtering disabled. OIT uses A-buffers. In this setting, all differences are due to the approximation of the truncated Fourier series or the moment-based reconstruction. The bottom row shows L1-difference images with Viridis colormap.*

on Figure 3. We observe that our moment-based opacity optimization outperforms decoupled opacity optimization clearly. The benefit grows with the scene complexity. Compared to Fourier opacity optimization, we achieve a similar speed. While our technique uses seven channels rather than nine, Fourier opacity optimization performs fewer arithmetic operations. Apparently, these effects roughly cancel out. In our implementation, both techniques use single-precision floating point values for each channel.

6. Conclusions and Future Work

Our moment-based opacity optimization is a useful new tool for visualization pipelines. It is faster and easier to implement than decoupled opacity optimization but just as capable of removing clutter. While the quality and speed is similar compared to Fourier opacity optimization, our technique achieves this with a lower memory footprint and inherits positive traits from moment-based OIT. Our screen space filtering decouples the implementation of opacity optimization from particular geometric primitives. Without the need for topological information, our method is easier to integrate into visualization pipelines.

Our method interacts with the geometry through three sequential additive rendering passes. Addition is of course embarrassingly parallel. This parallelism does not need to end within a GPU. Rendering with opacity optimization of a single dataset across multiple nodes is just as feasible, as long as the bandwidth suffices for an exchange of framebuffers. Thus, our work could enable the visualization of still greater datasets on multiple GPUs.

Acknowledgments

We would like to thank the anonymous reviewers for their constructive comments. The tornado dataset is courtesy of Marchesin

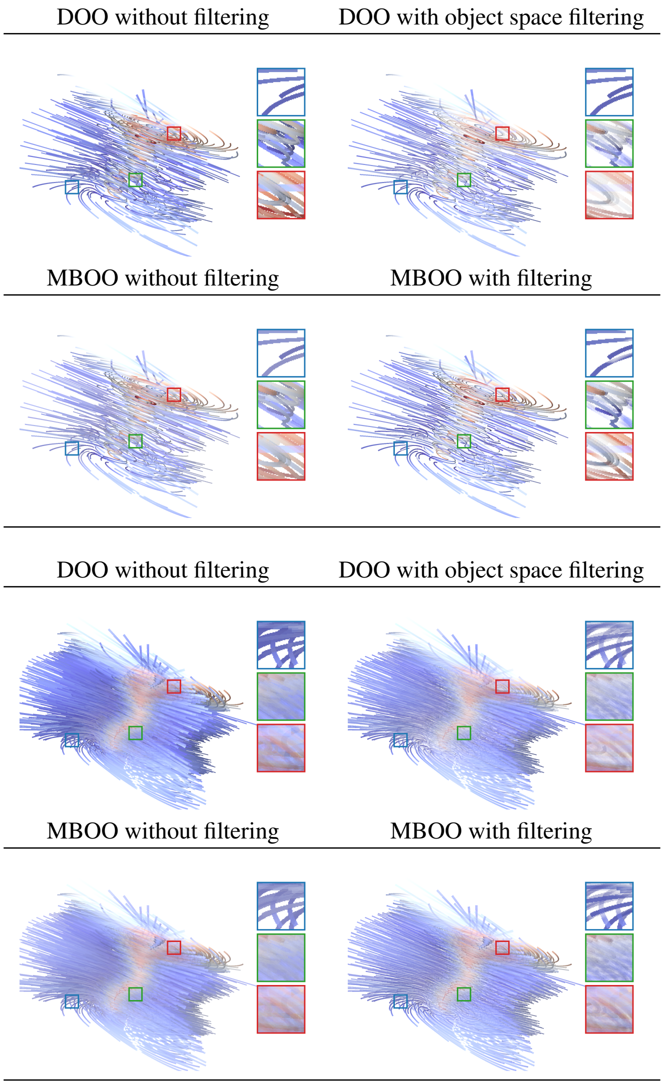
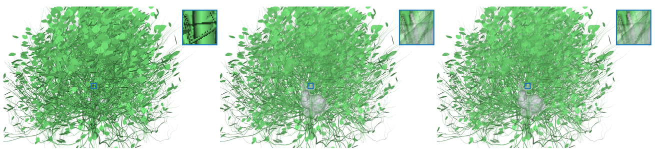


Figure 5: *The tornado dataset with two different line counts and different methods of filtering for opacity optimization. OIT uses A-buffers. Note how screen space filtering removes clutter in a fixed radius around important structures.*



(a) *OO disabled* **(b)** *FBOO* **(c)** *MBOO*

Figure 6: *Two complicated triangle meshes with different importance. Opacity optimization removes foreground clutter that would otherwise hide the bunny in the bush. OIT uses an A-buffer.*

et al. [MCHM10]. The Borromean rings and TrefoilKnot datasets are courtesy of Candelaresi and Brandenburg [CB11]. The Bunny model in Figure 6 is courtesy of the Stanford Computer Graphics Laboratory and the bush is courtesy of Blendswap user jlnh.

References

[AZD17] AMENT M., ZIRR T., DACHSBACHER C.: Extinction-optimized volume illumination. *IEEE Transactions on Visualization and Computer Graphics 23*, 7 (2017), 1767–1781. doi:10.1109/TVCG.2016.2569080. 2

[BCP*12] BRAMBILLA A., CARNECKY R., PEIKERT R., VIOLA I., HAUSER H.: Illustrative flow visualization: State of the art, trends and challenges. *Visibility-oriented Visualization Design for Flow Illustration* (2012). 1

[BGG19] BAEZA ROJO I., GROSS M., GUENTHER T.: Fourier opacity optimization for scalable exploration. *IEEE Transactions on Visualization and Computer Graphics* (2019), 1–1. doi:10.1109/TVCG.2019.2915222. 2, 3, 4, 5, 7, 8

[Bob08] BOBENKO A.: *Discrete differential geometry*. Springer, 2008. 5

[Car84] CARPENTER L.: The a-buffer, an antialiased hidden surface method. *SIGGRAPH Comput. Graph. 18*, 3 (1984), 103–108. doi:10.1145/964965.808585. 2, 3

[CB11] CANDELARESI S., BRANDENBURG A.: Decay of helical and nonhelical magnetic knots. *Physical Review E 84*, 1 (2011), 016406. 6, 9

[CM10] CORREA C. D., MA K.-L.: Visibility histograms and visibility-driven transfer functions. *IEEE Transactions on Visualization and Computer Graphics 17*, 2 (2010), 192–204. 2

[ELC*12] EDMUNDS M., LARAMEE R. S., CHEN G., MAX N., ZHANG E., WARE C.: Surface-based flow visualization. *Computers & Graphics 36*, 8 (2012), 974–990. 1

[GRT13] GÜNTHER T., RÖSSL C., THEISEL H.: Opacity optimization for 3D line fields. *ACM Transactions on Graphics 32*, 4 (2013), 120:1–120:8. doi:10.1145/2461912.2461930. 1, 2

[GRT14] GÜNTHER T., RÖSSL C., THEISEL H.: Hierarchical opacity optimization for sets of 3D line fields. *Computer Graphics Forum 33*, 2 (2014), 507–516. doi:10.1111/cgf.12336. 1, 2

[GSE*14] GÜNTHER T., SCHULZE M., ESTURO J. M., RÖSSL C., THEISEL H.: Opacity optimization for surfaces. *Computer Graphics Forum 33*, 3 (2014), 11–20. doi:10.1111/cgf.12357. 1, 2

[GTG17] GÜNTHER T., THEISEL H., GROSS M.: Decoupled opacity optimization for points, lines and surfaces. *Computer Graphics Forum 36*, 2 (2017), 153–162. doi:10.1111/cgf.13115. 1, 2, 3, 4, 5, 7, 8

[Gün16] GÜNTHER T.: *Opacity optimization and inertial particles in flow visualization*. PhD thesis, 2016. 2

[HKG00] HLADUVKA J., KÖNIG A., GRÖLLER E.: Curvature-based transfer functions for direct volume rendering. In *Spring Conference on Computer Graphics* (2000), vol. 16, Citeseer, pp. 58–65. 2

[JB10] JANSEN J., BAVOIL L.: Fourier opacity mapping. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2010), I3D '10, ACM, pp. 165–172. doi:10.1145/1730804.1730831. 2, 3

[KSW06] KRUGER J., SCHNEIDER J., WESTERMANN R.: Clearview: An interactive context preserving hotspot visualization technique. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 941–948. 2

[MCHM10] MARCHESIN S., CHEN C., HO C., MA K.: View-dependent streamlines for 3D vector fields. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1578–1586. doi:10.1109/TVCG.2010.212. 6, 9

[MJL*12] MCLOUGHLIN T., JONES M. W., LARAMEE R. S., MALKI R., MASTERS I., HANSEN C. D.: Similarity measures for enhancing interactive streamline seeding. *IEEE Transactions on Visualization and Computer Graphics 19*, 8 (2012), 1342–1353. 1

[MKKP18] MÜNSTERMANN C., KRUMPEN S., KLEIN R., PETERS C.: Moment-based order-independent transparency. *Proc. ACM Comput. Graph. Interact. Tech. (Proc. i3D) 1*, 1 (2018), 7:1–7:20. doi:10.1145/3203206. 1, 2, 3, 4, 5

[MLP*10a] MCLOUGHLIN T., LARAMEE R. S., PEIKERT R., POST F. H., CHEN M.: Over two decades of integration-based, geometric flow visualization. *Computer Graphics Forum 29*, 6 (2010), 1807–1829. doi:10.1111/j.1467-8659.2010.01650.x. 1

[MLP*10b] MCLOUGHLIN T., LARAMEE R. S., PEIKERT R., POST F. H., CHEN M.: Over two decades of integration-based, geometric flow visualization. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 1807–1829. 1

[MTHG03] MATTAUSCH O., THEUSSL T., HAUSER H., GRÖLLER E.: Strategies for interactive exploration of 3D flow using evenly-spaced illuminated streamlines. In *Proceedings of the 19th spring conference on Computer graphics* (2003), ACM, pp. 213–222. 1

[PK15] PETERS C., KLEIN R.: Moment shadow mapping. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games* (2015), i3D '15, ACM, pp. 7–14. doi:10.1145/2699276.2699277. 1, 3, 4

[PRW11] PFAFFELMOSER T., REITINGER M., WESTERMANN R.: Visualizing the positional and geometrical variability of isosurfaces in uncertain scalar fields. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 951–960. 2

[RBB*11] RUIZ M., BARDERA A., BOADA I., VIOLA I., FEIXAS M., SBERT M.: Automatic transfer functions based on informational divergence. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 1932–1941. 2

[SHH*07] SCHLEMMER M., HOTZ I., HAMANN B., MORR F., HAGEN H.: Priority streamlines: A context-based visualization of flow fields. In *EuroVis* (2007), pp. 227–234. 1

[SML11] SALVI M., MONTGOMERY J., LEFOHN A.: Adaptive transparency. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics* (2011), HPG '11, ACM, pp. 119–126. doi:10.1145/2018323.2018342. 2, 3

[SP13] SCHLEGEL P., PAJAROLA R.: Visibility-difference entropy for automatic transfer function generation. In *Visualization and Data Analysis 2013* (2013), vol. 8654, International Society for Optics and Photonics, p. 865406. 2

[Wol18] WOLFF D.: *OpenGL 4 Shading Language Cookbook: Build high-quality, real-time 3D graphics with OpenGL 4.6, GLSL 4.6 and C++ 17*. Packt Publishing Ltd, 2018. 5

[YHGT10] YANG J. C., HENSLEY J., GRÜN H., THIBIEROZ N.: Real-time concurrent linked list construction on the gpu. *Computer Graphics Forum 29*, 4 (2010), 1297–1304. doi:10.1111/j.1467-8659.2010.01725.x. 3, 5

[YKP05] YE X., KAO D., PANG A.: Strategy for seeding 3D streamlines. In *VIS 05. IEEE Visualization, 2005.* (2005), IEEE, pp. 471–478. 1

[YWSC11] YU H., WANG C., SHENE C.-K., CHEN J. H.: Hierarchical streamline bundles. *IEEE Transactions on Visualization and Computer Graphics 18*, 8 (2011), 1353–1367. 1