EUROGRAPHICS 2017 / L. Barthe and B. Benes
(Guest Editors)

Volume 36 (2017), Number 2

Decoupled Opacity Optimization for Points, Lines and Surfaces

Tobias Günther[1], Holger Theisel[2] and Markus Gross[1]

[1]Computer Graphics Laboratory, ETH Zurich
[2]Visual Computing Group, University of Magdeburg

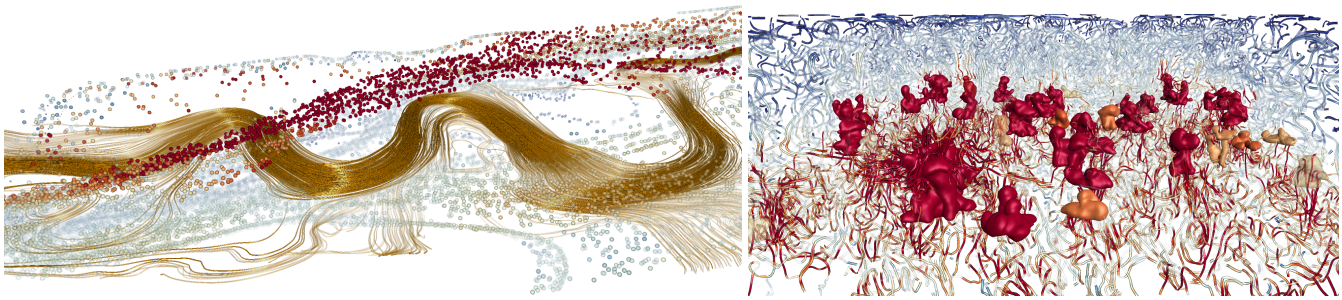


Figure 1: *Our method splits the previous opacity optimization technique [GRT13, GSM*14] into two smaller problems, which accelerates the optimization and allows us to combine different geometry types (points, lines and surfaces) in a single unified framework. Compared to previous work our method is completely GPU-based, runs the optimization per pixel, and has view-independent parameters. Left: atmospheric trace gas pathways in an air flow provided by the European Centre for Medium-Range Weather Forecasts (ECMWF) and right: streamlines and rain clouds (isosurfaces) at the boundary between troposphere and stratosphere in the cloud-topped boundary layer (CTBL) flow.*

Abstract

Displaying geometry in flow visualization is often accompanied by occlusion problems, making it difficult to perceive information that is relevant in the respective application. In a recent technique, named opacity optimization, the balance of occlusion avoidance and the selection of meaningful geometry was recognized to be a view-dependent, global optimization problem. The method solves a bounded-variable least-squares problem, which minimizes energy terms for the reduction of occlusion, background clutter, adding smoothness and regularization. The original technique operates on an object-space discretization and was shown for line and surface geometry. Recently, it has been extended to volumes, where it was solved locally per ray by dropping the smoothness energy term and replacing it by pre-filtering the importance measure. In this paper, we pick up the idea of splitting the opacity optimization problem into two smaller problems. The first problem is a minimization with analytic solution, and the second problem is a smoothing of the obtained minimizer in object-space. Thereby, the minimization problem can be solved locally per pixel, making it possible to combine all geometry types (points, lines and surfaces) consistently in a single optimization framework. We call this decoupled opacity optimization and apply it to a number of steady 3D vector fields.

1. Introduction

Geometry-based techniques, such as points, lines and surfaces have found great acceptance in the scientific visualization community [MLP*10, ELC*12a] across all major application areas. An inherent problem of geometry-based techniques in 3D however is that they can suffer severely from occlusion problems. That is, relevant data might be hidden behind rather unimportant geometry, cluttering up the viewport. The notion of relevance is thereby highly application-dependent, and is therefore modeled as a user parameter. A central goal of effective visualizations is to strive for a balance between the avoidance of occlusion and the representation of relevant information. To this end, Günther et al. [GRT13] proposed opacity optimization for line data, which was later extended to animated line geometry [GRT14] and surface geometry [GSM*14]. Their approach discretizes the geometry into segments, and computes

asynchronously on the CPU for each segment an opacity by minimizing a quadratic energy that characterizes desirable properties (avoidance of occlusion, removal of background clutter, smoothness and regularization). Thereby, opacities are bounded in the range $[0, 1]$ (0 is invisible, 1 is opaque). Such a bounded-variable least-squares problem can be solved using quadratic programming, which is for interactive tasks feasible for problem sizes in the order of only few thousand unknown opacities. A limitation was that the energy weights were view-dependent and in different orders of magnitude, when comparing lines and surfaces. Thus, different geometry types were not easy to combine in practice. Recently, Ament et al. [AZD16] extended opacity optimization to volume data by minimizing the energy in ray space (per pixel) and reformulating the objective function. They had the insight that when removing the smoothness term, the optimal solution can be calculated analytically. Instead of smoothing the opacities, they proposed to smooth the importance field. The obtained visualizations were comparable to the original opacity optimization, though at better frame rate.

In this paper, we pick up their insight and apply the idea to the rendering of points, lines and surfaces. That is, we drop the smoothness term in opacity optimization, and solve for the fragment opacities analytically per pixel. As shown later in Fig. 4, we found that smoothing the input importance field is not sufficient when dealing with sparse geometry data and instead propose to smooth the opacity solution iteratively in a post-process across the geometry. Thereby, all calculations can stay entirely on the GPU. The goal is a unified framework that allows the user to render point, line and surface geometry jointly in an optimal way. The performance of the resulting system is bound by the rendering of the transparent geometry, which involves sorting and compositing of fragment linked lists. In summary, our contributions are:

- Decoupling of opacity optimization into two smaller problems: a minimization problem with analytic solution and an iterative smoothing of its solution
- Application of opacity optimization to point data
- Combination of different geometry types (i.e., points, lines and surfaces) in a completely GPU-based pipeline

The benefits are a significantly increased solving rate, view-independent energy weights and a joint handling of point, line and surface data, which enhances applicability in practice. The solving speed-up comes at the expense of a slightly reduced framerate, since all operations moved to the GPU. We demonstrate our method in a number of steady 3D vector fields, see Fig. 1.

2. Related Work

Among the techniques that try to solve the occlusion problem, we can distinguish between seeding algorithms and selection algorithms. *Seeding algorithms* are either density-, feature- or similarity-based, and are typically difficult to adapt into interactive systems, because of their performance and lack of frame coherence. *Selection algorithms* on the other hand, pick geometry from a precomputed set of candidates, which is in interactive scenarios often the easier alternative, since the extraction (e.g., numerical integration) is moved into a preprocess. Our method is a selection-based technique. So far, the occlusion problem was addressed for each geometry type separately. Consequentially, we devote a subsection to each type.

2.1. Line data

Early works in 3D streamline placement were direct extensions from 2D, including density-based [MTHG03], feature-based [YKP05, YWSC12], and similarity-based measures [CCK07, MJL*13]. Li and Shen [LS07] were the first to recognize that line selection is a view-dependent problem, which led to their extension of Jobard and Lefer's [JL97] iterative 2D seeding strategy. Annen et al. [ATR*08] computed streamlines that resemble surface contours by using NPR-inspired local measures to terminate streamline integration. Xu et al. [XLS10] used entropy fields to guide selection and rendering so that streamlines represent the information of the original data set. Marchesin et al. [MCHM10] proposed a greedy approach that iteratively inserts lines by a score heuristic that accounts for the occupancy (per pixel fill rate) and local line entropy measures. Due to the greedy nature of this approach, the result is sensitive to small changes in the processing order, making it not frame coherent during camera navigation. Frame coherence is easier achieved with local measures, such as the view-dependent mapping of screen contribution (visible pixels of a line) to transparency by Günther et al. [GBWT11], which, however, tends to favor lines closer to the camera. Ma et al. [MWS13] combined lines from a view-independent and view-dependent candidate set. While they consider coherence between local views and the last frame, they do not account for the order of occlusions. Aggregating pixel statistics of the rasterized fragments is a common approach. Lee et al. [LMSC11] for instance, used maximum intensity projections of the entropy field to guide line and camera selection (both from candidates). Lawonn et al. [LGP14] used a curvature-based segmentation of foreground and background flow in blood vessels and Kanzler et al. [KFW16] selected lines from a precomputed line hierarchy, based on a maximum intensity projection of importance, depth and directional variation. Tao et al. [TMWS13] selected lines and the viewpoint simultaneously by treating them as interrelated information channels in an information-theoretic approach. In this paper, we build upon the global optimization-based line section approach of Günther et al. [GRT13], which was later extended to animated line geometry [GRT14]. We examine the method in Section 3.

2.2. Surface data

In surface-based flow visualization [ELC*12a], a number of surface placement method exist that for instance automatically fill a domain densely and evenly with surfaces [ELC*12b, ELM*12], select a single best surface [MSRT13a], multiple best surfaces [SMG*14] or find surfaces that meet certain desired properties best, e.g., be stretch-minimizing [BKC15], or be aligned with the flow or orthogonal to it [MSRT13b]. Günther et al. [GSM*14] extended opacity optimization to surfaces, which balances occlusion and visibility.

A general disadvantage of transparency-based approaches is that transparency inhibits depth perception. Illustrative flow visualization [BCP*12] has contributed a number of methods that improve transparent surface perception. Hummel et al. [HGH*10] described two view-dependent local transparency mappings that highlight silhouettes: angle-based transparency and normal variation. Carnecky et al. [CFM*13] made the layer order of transparent surfaces distinguishable by a diffusion of silhouettes and halos.

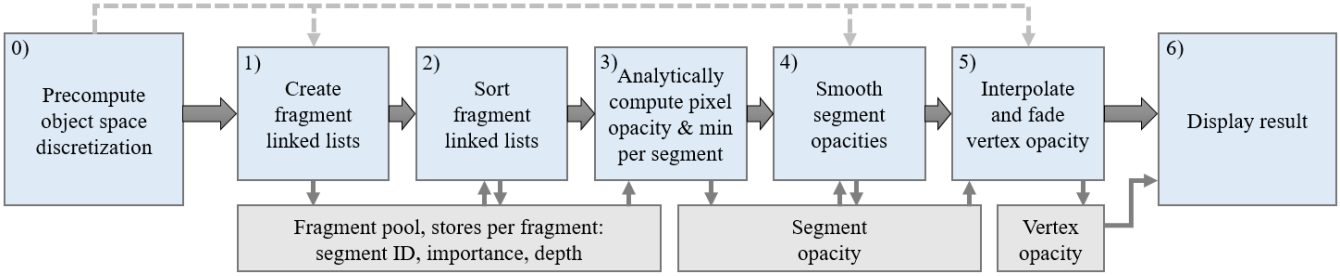


Figure 2: *Overview of the rendering pipeline, see Section 4.1 for details. Blue boxes denote the subsequent rendering stages and gray boxes represent memory. The dashed gray lines indicate stages that use the precomputed object-space discretization.*

2.3. Volume data

In volume data, occlusion is typically treated with transparency. Viola and Gröller [VG05] regarded smart visibility techniques, including ghosting and cutaways. Especially in direct volume rendering, a number of techniques exist, including importance-based opacity mappings [VKG04], opacity adjustments based on psychological principles [CWM*09], or incorporation of a relevance function into the volume rendering equation [MDM10]. Other conceptually different ways to avoid occlusion are exploded views [APH*03] and interactive spatial separation by deformation [CSC07].

Ament et al. [AZD16] extended opacity optimization to volume data, naming it extinction optimization. They not only optimized the extinction along view rays, but also along shadow rays. They showed that the opacity optimization energy has an analytic minimum, when the smoothness term is neglected. In this paper, we make use of this observation, but apply the concept to point, line and surface data.

3. Opacity Optimization

In this section, we briefly describe the original opacity optimization approach for line data [GRT13]. Given is a set of lines that cover the domain densely. First, all lines are split into equally-sized segments. An importance g_i (with $g_i \in [0,1]$) is assigned to each segment to specify the relevance to the user. Importance might be derived from geometric properties (size, curvature), information theory (linear and angular entropy), statistics (directional variance) or specific properties (vorticity, distance to domain boundary). In screen-space, the segments occlude each other. The occlusion degree h_{ij} states the number of pixels of segment i that occlude another segment j, and a_{ij} denotes object-space adjacency, which is 1 if segments are adjacent and 0 otherwise. The optimal opacity α_i for each of the n segments is computed by minimizing the quadratic energy:

$$E = \frac{p}{2}\sum_{i=1}^{n}(\alpha_i - 1)^2 \tag{1}$$

$$+\frac{q}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\alpha_i\,(1-g_i)^{\lambda}\,h_{ij}\,g_j\right)^2 \tag{2}$$

$$+\frac{r}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\alpha_i\,(1-g_i)^{\lambda}\,h_{ji}\,g_j\right)^2 \tag{3}$$

$$+\frac{s}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{a_{ij}}{2}(\alpha_i - \alpha_j)^2 \tag{4}$$

with bounded variables $0 \leq \alpha_i \leq 1$.

Term (1) is a regularization that prevents empty renderings by penalizing α_i unequal to 1 (opaque). Term (2) introduces a penalty if an unimportant segment i (i.e., $1-g_i$ is large) occludes to some degree an important segment j (i.e., g_j is large). Then, $h_{ij} > 0$ and thus the product $(1-g_i)^{\lambda} h_{ij}\, g_j$ has a high value. To minimize the term, the optimization will fade out the segment in front, i.e., α_i is reduced. Thereby, λ steers the fall-off of g_i from 1, which allows the user to put further emphasis on the important structures. Term (3) removes background clutter by fading out unimportant segments behind important ones. Term (4) enforces a slow change of the opacity along a line by penalizing the difference in opacity between adjacent segments. The energy weights $q \geq 0$, $r \geq 0$ and $s \geq 0$, and the emphasis exponent $\lambda > 0$ are set by the user, whereas $p = 1$ by default. Minimizing this energy leads to a (convex constrained) linear system, which is low-banded (e.g., 3-diagonal for lines) [Gün16], yet still a global system needs to be solved.

The method was later extended to animated lines [GRT14], for which the discretization was selected view-dependently from a precomputed split tree, and a temporal smoothness term was added to the energy to ensure frame coherence. Shortly later, the original approach was extended to surfaces [GSM*14], which were subdivided into patches. The objective function was the same as in Eqs. (1)-(4), though a_{ij} was related to geodesic distance.

The object-space opacity optimization requires the solution of a global system and only a few thousand elements can be optimized interactively, which is not enough for large point data sets. Other disadvantages are that parameters q, r, s and λ are view-dependent and that different geometry types could not easily be combined.

4. Decoupled Opacity Optimization

In the following, we introduce a unified framework for the treatment of point, line and surface data. First, we provide a general overview of the system, them elaborate on the objective function and afterwards proceed with implementation details.

4.1. System Overview

Fig. 2 gives an overview of the pipeline. While a main difference to the original opacity optimization is that the energy is now minimized per pixel, we still require an object-space discretization for the subsequent smoothing. Thus in a preprocess, line and surface geometry is discretized into segments as previously done in [GRT14, GSM*14]. Points are not subdivided, see Fig. 3.

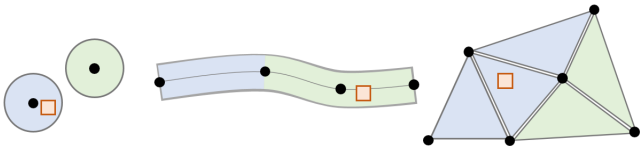


Figure 3: *Terminology used for points, lines and surfaces: vertices (black points), segments (green and blue), fragments (orange).*

At runtime, we render all point, line and surface geometry, compiling fragment linked lists that store a segment ID, user-defined importance and depth of each fragment. After sorting, these lists allow us to infer the depth order, which, together with the fragment importance, is used to calculate an optimal opacity value per fragment. Using the object-space discretization, we gather the minimum opacity of each segment and afterwards apply an object-space smoothing. We interpolate the opacity values per vertex and fade into the newly created opacity solution for temporal smoothness. For final display, all geometry is rendered again with the per-vertex opacities as input, including fragment linked list construction, sorting and final compositing (optionally at high resolution and multi-sampling).

4.2. Objective Function

Previous opacity optimization approaches used the object-space discretization to keep the problem size small. In this work, we solve the problem locally per pixel.

Energy Function. When minimizing the energy for each pixel independently, the discrete elements are the n fragments that were rasterized into the same pixel, i.e., the elements of a fragment linked list, sorted from front to back:

$$E = \frac{p}{2}\sum_{i=1}^{n}(\alpha_i - 1)^2 \tag{5}$$

$$+\frac{q}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\alpha_i\,(1-g_i)^{\lambda}\,h_{ij}\,g_j)^2 \tag{6}$$

$$+\frac{r}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\alpha_i\,(1-g_i)^{\lambda}\,h_{ji}\,g_j)^2 \tag{7}$$

which is the energy by Günther et al. [GRT13, Gün16] from Eqs. (1)-(3) without the smoothness term Eq. (4). Note that Eqs. (5)-(7) sum over fragments, not segments. (For brevity, we abstain from a formal redefinition.) Thus, α_i is the unknown opacity of a fragment, h_{ij} is the occlusion term that denotes whether fragment i occludes fragment j. If the fragments are sorted from front to back, then:

$$h_{ij} = \begin{cases} 1: & i < j \\ 0: & else \end{cases} \qquad h_{ji} = \begin{cases} 1: & i > j \\ 0: & else \end{cases} \tag{8}$$

The importance g_i (with $g_i \in [0,1]$) specifies the relevance of a fragment, e.g., using geometric properties, information theory, statistics, samples from scalar fields or distance measures.

Term (5) is a regularization that makes the geometry try to be as visible as possible. Term (6) reduces the opacity α_i of an unimportant fragment $(1-g_i)$ if it occludes to some degree h_{ij} an important fragment (g_j). Similarly, Term (7) removes background clutter by fading out unimportant fragments behind important ones. For reference, we set $p = 1$, whereas the energy weights $q \geq 0$ and $r \geq 0$, and the emphasis exponent $\lambda > 0$ are set by the user.

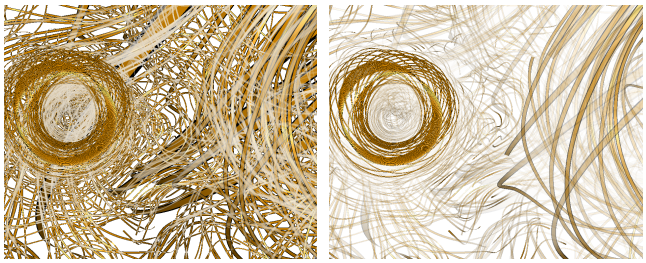


Figure 4: *Comparison: computing opacities per pixel and smoothing of importance as in [AZD16] (left), opacities computed per pixel with subsequent object-space smoothing (our method, right).*

Analytic Solution. Without a smoothness term, the opacity values of the individual fragments are independent of each other [AZD16]. Hence, the energy can be rearranged to $E = \sum_{i=1}^{n} E_i(\alpha_i)$ with

$$E_i(\alpha_i) = \frac{p}{2}(\alpha_i - 1)^2 + \frac{q}{2}\sum_{j=1}^{n}(\alpha_i\,(1-g_i)^{\lambda}\,h_{ij}\,g_j)^2 + \frac{r}{2}\sum_{j=1}^{n}(\alpha_i\,(1-g_i)^{\lambda}\,h_{ji}\,g_j)^2$$

so that for each fragment the opacity α_i can be optimized independently by minimizing the quadratic energy $E_i(\alpha_i)$. Using Eq. (8), the optimal opacity is found by setting $\frac{\mathrm{d}E_i(\alpha_i)}{\mathrm{d}\alpha_i} = 0$ and rearranging for α_i [AZD16]:

$$\alpha_i = \frac{p}{p + (1-g_i)^{2\lambda}\left(r\sum_{j=1}^{i-i} g_j^2 + q\sum_{j=i+1}^{n} g_j^2\right)} \tag{9}$$

The resulting opacities are in the range $[0,1]$.

Smoothing. Minimizing the energies above per fragment leads to an efficient, analytic closed-form solution. With sparse geometry, however, discontinuities occur, as visible in Fig. 4 (left) for line geometry. If only one line is visible in a pixel, the line fragment will be fully opaque. At line crossings, however, the occlusion terms will reduce the opacities of the fragments in the linked list, which creates unwanted discontinuities. Smoothing the importance [AZD16] across the geometry does not remove this artifact, since the opacity in pixels with a single fragment is not affected. Instead, we smooth the opacities across the geometry, for which we use the precomputed object-space discretization. In a first step, we determine among the fragments of each discrete segment the one with smallest opacity value, yielding a per-segment opacity. (We discuss in Section 6.5 why we prefer the minimum over the average operator.) Afterward, we iteratively smooth (s iterations) the per-segment opacities, and interpolate per-vertex opacities for final display, see Fig. 4 (right).

5. Implementation

In the following, we reiterate the steps of the computation pipeline (see Fig. 2) and provide implementation details on each step.

In a preprocess, the object-space discretization is computed. The discretization of lines follows [GRT14], i.e., we proportionally subdivide the candidate line set based on the length of the individual lines, such that segment sizes are as uniform as possible. For surfaces, we use a geodesic farthest point sampling as in [GSM*14] to distribute points uniformly on the surfaces. The corresponding Voronoi cells form the discrete surface elements. The distances are computed with the method of Crane et al. [CWW13]. At runtime, we iterate the following steps each frame:

1. Render all point, line and surface geometry, and construct one linked list per pixel [YHGT10] that stores the fragments of all geometry types. Each list entry stores the importance g_i, the depth value (for sorting), an ID that allows us to retrieve the segment that the rasterized fragment belongs to, and the index of the next fragment in the list. All lists are constructed in fragment shaders by rendering with single-sampling.
2. Generate one thread per pixel that sorts the fragment linked list of the pixel by depth from front to back.
3. Iterate the fragment linked lists to compute the optimal opacity value α_i of each fragment using importance g_i, see Alg. 1. Using the aforementioned ID, each fragment can be associated with a segment. In the same step, per-segment opacities can be determined by atomically computing the minimum opacity of the fragments that belong to one segment.
4. The per-segment opacities of lines and surfaces are iteratively smoothed using Laplacian smoothing. For line geometry, we use the two adjacent segments and for surface geometry the eight geodesically closest segments.
5. For lines and surfaces, a per-vertex opacity is interpolated from the per-segment opacities. (For points, each vertex is a segment.) The interpolation weights are precomputed. See the blending weight parameterization in [GRT13] (lines) and the blending weights in [GSM*14] (surfaces) for details. Care must be taken with segments that are not on the viewport, since their per-segment opacity is undefined and cannot be used in the per-vertex interpolation. Once the per-vertex opacities are computed, we fade toward the solution for temporal smoothness. The result is written to a vertex buffer that contains the current opacities.
6. The final rendering pass uses the current opacity values as vertex input and simply displays the result, including fragment linked list construction, sorting and compositing. This pass can be at higher resolution and with multi-sampling for better quality.

Self-occlusion. With line and surface data, large segments can self-occlude. While self-occlusions were easy to exclude in the original optimization [GRT13], we need a special treatment in Alg. 1. Each fragment keeps an ID that allows us to retrieve its corresponding segment. When self-occlusion handling is enabled, we skip a fragment, if its predecessor had the same ID. The difference is seen in Fig. 5. Here, each geometry is represented as one segment. The top image contains objects that do not occlude another object, yet they appear transparent. This is due to self-occlusion, which is not desired and taken care of in the bottom image.

6. Results

In the following section, we demonstrate decoupled opacity optimization in a number of data sets, in which different geometry types are combined in one scene. We further discuss parameters, compare with the original opacity optimization, measure the performance and finally discuss algorithmic choices and limitations.

Input: List of fragments with importance g_i, weights p, q, r, λ.
Output: Optimal opacity α_i of each fragment in the list.

$g_{all} \leftarrow 0,\ g_f \leftarrow 0$;
for $i = 1,\ldots,n$ **do**
  $g_{all} \leftarrow g_{all} + g_i^2$;
end
for $i = 1,\ldots,n$ **do**
  $g_b \leftarrow g_{all} - g_i^2 - g_f$;
  $\alpha_i \leftarrow p / \left(p + (1-g_i)^{2\lambda}\left(r \cdot g_f + q \cdot g_b\right)\right)$; // Eq. (9)
  $g_f \leftarrow g_f + g_i^2$;
end

Algorithm 1: Per-fragment opacities α_i are computed by iterating the fragment linked lists twice. First, the squared sum of all importance values g_{all} is computed, which is used in the second pass to compute the squared sum of importance values g_b behind the current fragment. (Pseudo code adapted from [AZD16].)

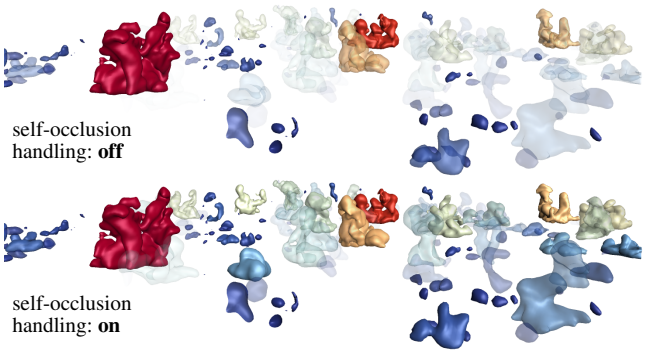


Figure 5: *In the* CTBL *data set, each geometry is represented as one segment. Some segments self-occlude, thus the objects fade out, since they apparently occlude something of interest (top). We can avoid this effect by skipping self-occlusions (bottom).*

6.1. Points, Lines and Surfaces

The CTBL data set contains a cloud resolving boundary layer simulation between troposphere and stratosphere (UCLA-LES in [Ste13]). It was used to derive cloud statistics that can be fed as sub-scale parameters into large scale climate simulations. The air flow is visualized in Figs. 1 (right) and 6 by streamlines. Thereby, line color encodes importance, in this case velocity magnitude. Isosurfaces depict the scalar cloud indicator field, showing a correlation between (fast) vertical transport and the presence of clouds. The cloud color encodes the surface area of a cloud. Larger clouds are assumed to be more important. By using opacity optimization, line geometry is faded out that hinders the view on large clouds. Since streamlines with high magnitude are important, as well, they stay visible in the clouds, illustrating their vertical uplift.

The ECMWF flow contains a large scale reanalysis of the weather in the Northern hemisphere from April $10-19$, 2010 and was provided by the European Centre for Medium-Range Weather Forecasts.

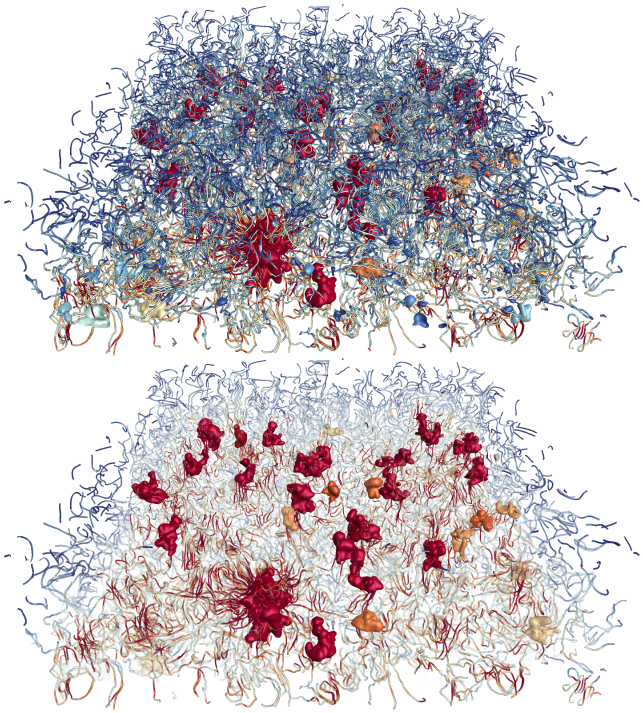


Figure 6: *Clouds (isosurfaces) and airflow (streamlines) at boundary between troposphere and stratosphere in* CTBL. *Top: all geometry is opaque, causing occlusion. Bottom: opacity optimization clears the view on large clouds and streamlines with high magnitude, which allows us to analyze the correlation between fast vertical transport and the presence of clouds ($q = 50$, $r = 100$, $s = 5$, $\lambda = 3$).*

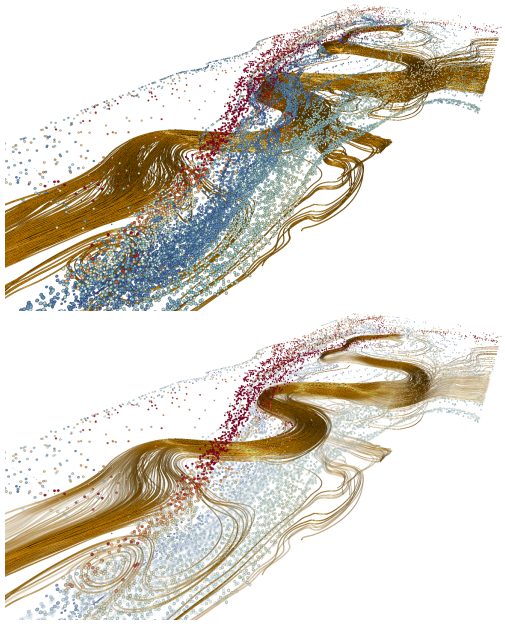


Figure 7: *Pathways of trace gases. Top: opaque display of particles and lines. Bottom: opacity optimization reveals pathways with high velocity magnitude ($q = 400$, $r = 50$, $s = 5$, $\lambda = 4$).*

For the analysis, convective flow features in the wind velocity field are of interest, since they are related to the exchange of energy and transport of trace gases in the atmosphere. Figs. 1 (left) and 7 combine particles and streamline geometry, showing different transport paths. Point and line geometry was seeded in different altitude layers. We used the wind velocity magnitude as importance measure, which emphasizes the fastest transport pathways.

6.2. Parameters

The energy weights q, r and λ behave intuitively as in the original opacity optimization. First, the overall opacity is adjusted by varying q, then emphasis λ is set to adjust the important structures. Afterwards, the background can be cleared using r and finally the result is iteratively smoothed. We refer to [GRT13] and [GSM*14] for a parameter study and a description of the incremental setting of the parameters.

View Independence. A strong advantage of per-pixel optimization (this paper) over per-segment optimization (previous work) is that the energy parameters are now entirely view-independent, i.e., they no longer depend on the camera location. Fig. 8 views the TREFOIL data set from different distances (far, near and close-up) using *the same* energy parameters. With per-segment optimization, the parameters would have to be readjusted, since the occlusion degrees are accumulated from all visible fragments of a segment. For a distant viewpoint, the total number of fragments and therefore the occlusion is lower, which leads the optimization to assign more opaque values. The TREFOIL data set was provided by Candelaresi and Brandenburg [CB11] and contains a magnetic field. Here, velocity magnitude was used as importance measure, which reveals the trefoil knot.

Smoothing. The key difference to the original method is that in the decoupled approach, the opacity solution is smoothed separately in a post-process by Laplacian smoothing. Fig. 9 shows results for varying numbers of smoothing iterations in the DELTA WING flow, which was provided by Markus Rütten. If the number of iterations is too small, the object-space discretization becomes apparent.

Importance. In opacity optimization, the relevance of a geometric object is prescribed by the user. The importance measure is the main tool to adjust which structures should be visible and which may fade out if necessary. When combining different types of geometries, it is also the method of choice to prioritize. In Fig. 10 for instance, we adjusted the importance to steer the visibility of points and lines. In Fig. 10a, important points have higher relevance than important lines. Therefore, the core of the tornado is illustrated by points. In Fig. 10b, the situation is vice versa. Here, lines reveal the vortex core. In both images, stream surfaces that wind around the core are faded out to clear the view. For reference, Fig. 10c was rendered with all points, lines and surfaces opaque: it is clear that occlusion prevents understanding of the vortex core.

Problem Size. The VISCOUS FINGERS data set was subject of the 2016 Scientific visualization contest [Sci16]. It is result of a finite point-based ensemble simulation of a salt layer dissolving in a cylindrical water container [Kuh14]. All particles carry a salt concentration, which serves as importance measure (color-coded).

Figure 8: *With per-pixel optimization (this paper), the energy parameters are independent of the camera location. Here, the* TREFOIL *is shown from three camera locations (far, near and close-up), using* the same *energy weights ($q = 50$, $r = 400$, $s = 15$, $\lambda = 2$).*

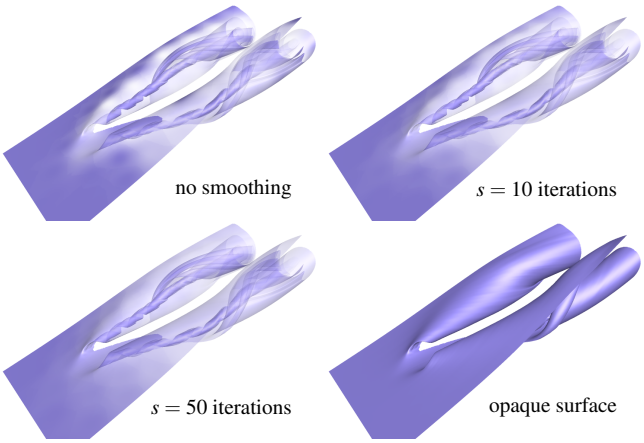


Figure 9: *Result after varying numbers of Laplacian smoothing iterations. If the number is too small, the underlying object-space discretization becomes visible ($q = 50$, $r = 10$, $\lambda = 2$).*

To reduce the fill rate, we decreased the size of unimportant particles. The ensemble simulations contain a number of runs for three resolutions. Fig. 11 shows the original point set and our result for all three resolutions. For all, we used the 80th time step of the first run. With our method, the viscous fingers become visible.

6.3. Comparison

In the following, we compare the results of our decoupled opacity optimization with the original opacity optimization, which solves a bounded-variable least-squares problem. For line data, we compare with hierarchical opacity optimization [GRT14] and for surface data with [GSM*14]. Fig. 12 gives qualitative and quantitative results. The top row shows the results of the previous methods and the bottom row shows our approach. In all cases, the results are visually similar, which means our simpler approach can approximate the full opacity optimization quite well.

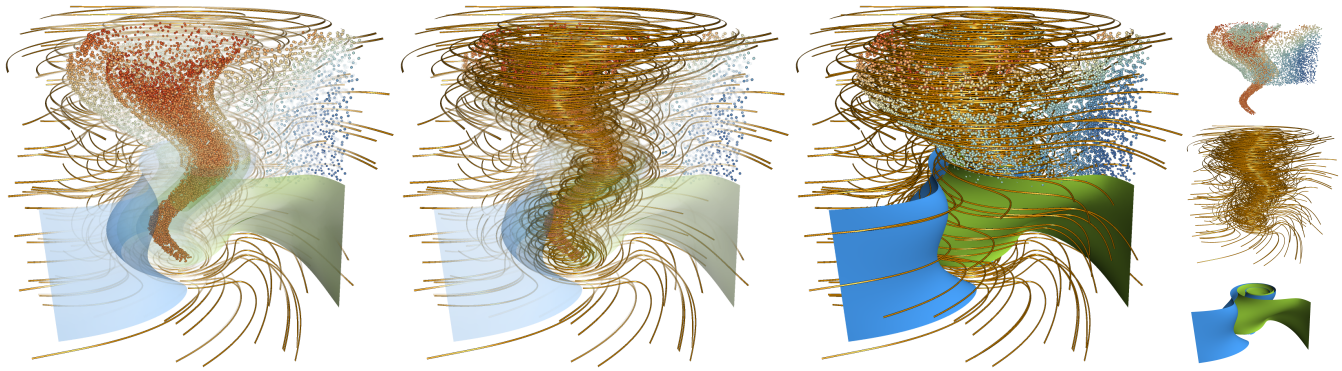
Solves per Second. The previous approaches computed the minimization asynchronously on the CPU, which can become slow for large problem sizes or when the depth complexity is high. With line geometry, we typically have much larger problem sizes ($n \approx 10k$) than we observe for surface data ($n \approx 400$). Also, the depth complexity is higher for dense line data, which results in longer fragment linked lists that need to be copied to the CPU for further processing. Consequentially, the setup of the system matrix also gets slower. In summary, we expect a higher gain for line data than for surfaces when applying our method. Indeed, for lines we observe a $23-83\times$ speedup, for surfaces only $2.61-2.78\times$.

Frames per Second. While we expect a higher solving rate, we must expect a slower rendering performance, since all calculations moved to the GPU. This problem is less severe than the bottlenecks created by the CPU, since GPU performance has increased rapidly over the years. For line data, the frame rate dropped by $0.24-0.46\times$ and for surfaces by $0.3-0.5\times$. Still, the frame rates have been well beyond 60*fps*. (In extreme cases it would be imaginable to distribute the computation over multiple frames.)

Data Sets. The first column demonstrated results on the BORROMEAN data set, which contains magnetic decaying rings that were numerically simulated [CB11]. The second column contains a HELICOPTER flow, which shows experimental wind tunnel data of a helicopter in descent [YTvdW*02]. Here, the two wake vortices are of interest. The third column shows the flow in a STATIC MIXER, provided by Markus Rütten. The last data set was also courtesy of him and contains the DELTA WING, which contains two vortices.

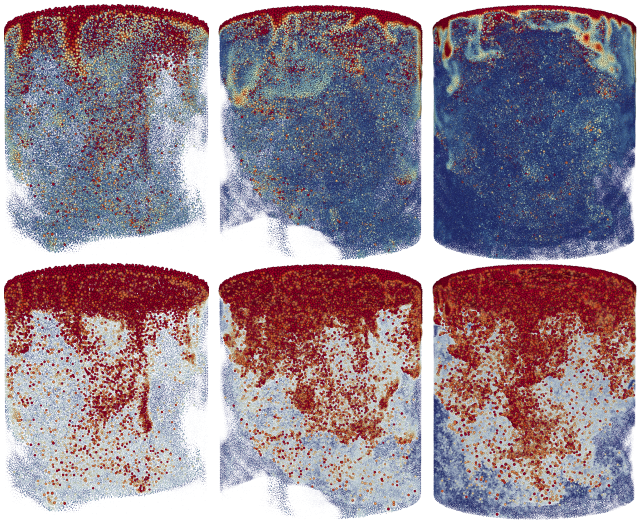
6.4. Performance

Table 1 contains detailed performance measurements of the pipeline steps described in Section 5, including linked list construction (1), sorting (2), per-segment opacity computation (3), iterative smoothing (4), interpolation of per-vertex opacity (5) and final display (6). For all measurements, we used an Nvidia GTX 1080 GPU with 8 GB VRAM and an Intel i7-6700K CPU with 4.0 GHz. All images were rendered with a viewport resolution of 1000×1000 pixels. Among the first 5 steps (computation of per-vertex opacities), we printed the bottleneck bold, which was in almost all cases the sorting of the fragment linked lists. This means, the performance is bound by the number of fragments per pixel. This can be seen by the lower performance of dense point sets (Viscous high) and dense line sets (Helicopter), compared to surface data (Static Mixer or Delta Wing).



(a) *Points are more important than lines.* **(b)** *Lines are more important than points.* **(c)** *Opaque rendering as a reference (cluttered).*

Figure 10: *Importance can be used to prioritize individual geometry types. Here, in the* TORNADO *flow ($q = 1000$, $r = 50$, $s = 50$, $\lambda = 3.5$).*



194,248 particles 571,817 particles 1,790,040 particles

Figure 11: *Performance comparison in* VISCOUS FINGERS *data set, demonstrating our method (bottom) for different point set sizes. The top row contains the original opaque data (top), which suffers from occlusion. The limiting factor is the fill rate-related sorting and processing of the transparent fragments ($q = 200$, $r = 50$, $\lambda = 3$).*

Data set	1	2	3	4	5	6	Total
CTBL	0.79	**1.93**	0.52	0.03	0.10	3.03	6.4
ECMWF	0.67	**12.0**	1.38	0.02	0.01	9.62	23.7
Tornado	0.47	**1.96**	0.59	0.04	0.01	1.85	4.92
Borromean	0.88	**2.95**	0.94	0.04	0.02	4.32	9.15
Helicopter	1.04	**5.96**	0.71	0.03	0.02	7.59	15.4
Static Mixer	0.32	**0.89**	0.73	0.09	0.02	0.94	2.99
Delta Wing	0.63	0.70	**0.73**	0.27	0.25	0.92	3.50
Viscous (low)	0.53	**1.15**	0.41	–	0.02	1.95	4.06
Viscous (med)	1.09	**4.72**	1.96	–	0.02	6.79	14.6
Viscous (high)	2.40	**14.1**	7.34	–	0.08	19.8	43.72

Table 1: *Runtime in ms for the individual computation steps ($1-6$), described in Section 5, as well as the total time per frame.*

6.5. Discussion

In the following, we discuss algorithmic choices and limitations.

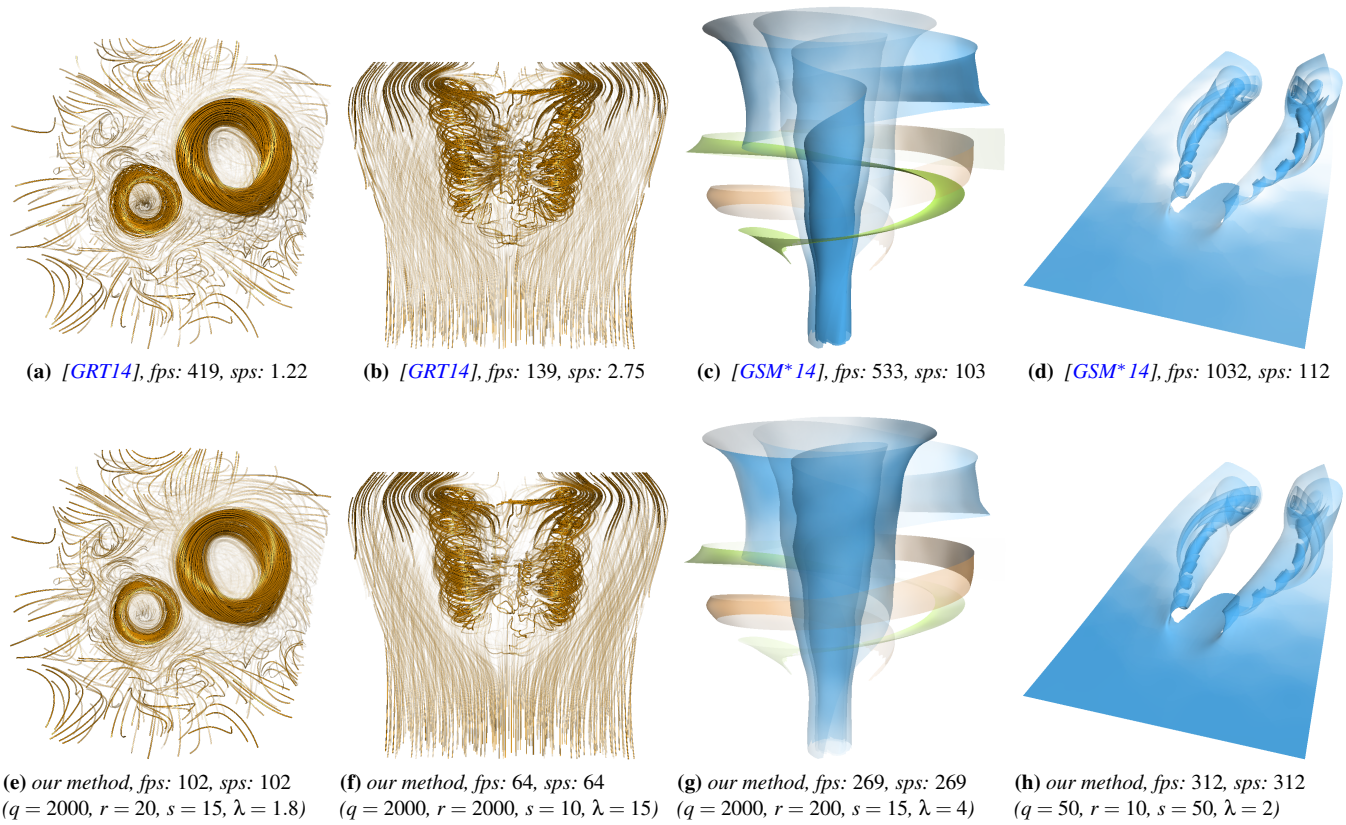
Per-Segment Minimum vs. Average. To calculate an opacity value per segment, we take the minimum opacity of its visible fragments. This is a conservative choice, which is favorable over calculating the average of the fragments. If there is just one fragment that needs to fade out as it occludes an important object, than the segment should be removed, otherwise the important object might not be visible. This decision should be made regardless of how many other fragments of the segment are not occluding anything, since this would be dependent on the distance to the camera.

Spatial Smoothing in Screen Space. The necessity to compute an object-space discretization is somewhat limiting for surface geometry. (For line geometry it is trivial, and for points it is not required.) It would also be possible to perform the smoothing in screen-space. Carnecky et al. [CFM*13] for instance, applied screen-space smoothing on fragment linked lists in order to diffuse color that was injected at silhouettes. Their approach, however, requires to establish neighborhoods between fragments, for which they used heuristics that can fail in extreme cases, such as very close surfaces.

Temporal Smoothing. The conservative choice of using the minimum per-pixel opacity might cause frame incoherence when the respective minimum pixel vanishes during camera navigation. Thus, we temporally smooth the per-vertex opacities, similar to [GRT13]. See the video for examples with and without temporal smoothing.

Volume Data. Our pipeline can be conceptually combined with volume data. After construction of sorted fragment linked lists for all non-volume data, a volume ray cast could be performed in the spirit of [AZD16]. As the ray traverses front to back through the volume, the fragment linked list could be iterated alongside, advancing to the next fragment, when the volume ray passed the last fragment.

Time-Dependent Data. In this paper, we focused on steady vector fields. When dealing with time-dependent geometry, a correspondence heuristic is required to ensure temporal coherence. In [GRT14], this was demonstrated for line geometry, where previous opacity solutions were advected to the next frame. Suitable



(a) *[GRT14], fps:* 419, *sps:* 1.22 **(b)** *[GRT14], fps:* 139, *sps:* 2.75 **(c)** *[GSM*14], fps:* 533, *sps:* 103 **(d)** *[GSM*14], fps:* 1032, *sps:* 112

(e) *our method, fps:* 102, *sps:* 102 *($q = 2000$, $r = 20$, $s = 15$, $\lambda = 1.8$)* **(f)** *our method, fps:* 64, *sps:* 64 *($q = 2000$, $r = 2000$, $s = 10$, $\lambda = 15$)* **(g)** *our method, fps:* 269, *sps:* 269 *($q = 2000$, $r = 200$, $s = 15$, $\lambda = 4$)* **(h)** *our method, fps:* 312, *sps:* 312 *($q = 50$, $r = 10$, $s = 50$, $\lambda = 2$)*

Figure 12: *Comparison of [GRT14] and [GSM*14] (top row) with our decoupled opacity optimization (lower row). The results are visually similar. The solves per second (sps) increased significantly, whereas the frames per second (fps) reduced, since our opacity computation runs on the GPU, not the CPU. While previous method solved for the opacities asynchronously, our method can calculate it every frame.*

correspondence heuristics depend on the underlying geometry and on the employed refinement strategies, e.g., if additional vertices appear because of insertion of points, line vertices or surface triangles.

Perception Issues with Transparency. An inherent problem of transparency is that fading might be confused with depth cueing, which inhibits spatial perception. A number of approaches exist that add visual cues to compensate this effect, such as halos [CFM*13], or that avoid transparency by modulating the line width [KFW16].

7. Conclusions

Following Ament et al.'s [AZD16] idea of extinction optimization for volumes, we split opacity optimization for geometric data [GRT13, GSM*14] into two smaller problems: an analytic computation of per-pixel opacities, and an iterative smoothing and blending across the geometry. In the pixel-based formulation, different geometry types such as points, lines and surfaces are easily combined in a completely GPU-based framework. Instead of smoothing the input importance field [AZD16], we perform the smoothing on a precomputed object-space discretization. Compared to the original opacity optimization we observed a solving speedup of $23-83\times$ for lines, and $2.6-2.8\times$ for surfaces. The rendering performance decreased slightly, since the opacity calculations have migrated to the GPU. The decoupled approach has more desirable parameter behavior, i.e., parameter choices are no longer view-dependent.

In the future, we would like to include a view-dependent, possibly hierarchical selection of the candidate geometry from a precomputed candidate set, or even with adaptive insertion of geometry if the data becomes too sparse in close-ups. Additionally, we would like to perform not only a view optimization, but also a shadow optimization as in [AZD16]. Since geometry is rather sparse, shadows could add too many color discontinuities, which would have to be addressed.

References

[APH*03] AGRAWALA M., PHAN D., HEISER J., HAYMAKER J., KLINGNER J., HANRAHAN P., TVERSKY B.: Designing effective step-by-step assembly instructions. *ACM Trans. Graph. (Proc. SIGGRAPH) 22*, 3 (2003), 828–837. 3

[ATR*08] ANNEN T., THEISEL H., RÖSSL C., ZIEGLER G., SEIDEL H.-P.: Vector field contours. In *Proc. Graphics Interface* (2008), pp. 97–105. 2

[AZD16] AMENT M., ZIRR T., DACHSBACHER C.: Extinction-optimized volume illumination. *IEEE Transactions on Visualization and Computer Graphics* (2016), to appear. 2, 3, 4, 5, 8, 9

[BCP*12] BRAMBILLA A., CARNECKY R., PEIKERT R., VIOLA I., HAUSER H.: Illustrative flow visualization: State of the art, trends and challenges. In *EuroGraphics 2012 State of the Art Reports (STARs)* (2012), pp. 75–94. 2

[BKC15] BARTOŇ M., KOSINKA J., CALO V. M.: Stretch-minimising stream surfaces. *Graphical Models 79* (2015), 12 – 22. 2

[CB11] CANDELARESI S., BRANDENBURG A.: Decay of helical and nonhelical magnetic knots. *Phys. Rev. E 84* (2011), 016406. 6, 7

[CCK07] CHEN Y., COHEN J., KROLIK J.: Similarity-guided streamline placement with error evaluation. *IEEE Transactions on Visualization and Computer Graphics 13* (2007), 1448–1455. 2

[CFM*13] CARNECKY R., FUCHS R., MEHL S., JANG Y., PEIKERT R.: Smart transparency for illustrative visualization of complex flow surfaces. *IEEE TVCG 19*, 5 (2013), 838–851. 2, 8, 9

[CSC07] CORREA C., SILVER D., CHEN M.: Illustrative deformation for data exploration. *IEEE TVCG 13*, 6 (2007), 1320–1327. 3

[CWM*09] CHAN M.-Y., WU Y., MAK W.-H., CHEN W., QU H.: Perception-based transparency optimization for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1283–1290. 3

[CWW13] CRANE K., WEISCHEDEL C., WARDETZKY M.: Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Trans. Graph. (Proc. SIGGRAPH) 32*, 5 (2013), 152:1–152:11. 5

[ELC*12a] 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, 2

[ELC*12b] EDMUNDS M., LARAMEE R. S., CHEN G., ZHANG E., MAX N.: Advanced, automatic stream surface seeding and filtering. In *Theory and Practice of Computer Graphics* (2012), pp. 53–60. 2

[ELM*12] EDMUNDS M., LARAMEE R., MALKI R., MASTERS I., CROFT T., CHEN G., ZHANG E.: Automatic stream surface seeding: A feature centered approach. *Computer Graphics Forum (Proc. EuroVis) 31*, 3 (2012), 1095–1104. 2

[GBWT11] GÜNTHER T., BÜRGER K., WESTERMANN R., THEISEL H.: A view-dependent and inter-frame coherent visualization of integral lines using screen contribution. *Proc. Vision, Modeling, and Visualization (VMV)* (2011), 215–222. 2

[GRT13] GÜNTHER T., RÖSSL C., THEISEL H.: Opacity optimization for 3D line fields. *ACM Transaction on Graphics (Proc. SIGGRAPH) 32*, 4 (2013), 120:1–120:8. 1, 2, 3, 4, 5, 6, 8, 9

[GRT14] GÜNTHER T., RÖSSL C., THEISEL H.: Hierarchical opacity optimization for sets of 3D line fields. *Computer Graphics Forum (Proc. Eurographics) 33*, 2 (2014), 507–516. 1, 2, 3, 5, 7, 8, 9

[GSM*14] GÜNTHER T., SCHULZE M., MARTINEZ ESTURO J., RÖSSL C., THEISEL H.: Opacity optimization for surfaces. *Computer Graphics Forum (Proc. EuroVis) 33*, 3 (2014), 11–20. 1, 2, 3, 5, 6, 7, 9

[Gün16] GÜNTHER T.: *Opacity Optimization and Inertial Particles in Flow Visualization*. PhD thesis, University of Magdeburg, June 2016. 3, 4

[HGH*10] HUMMEL M., GARTH C., HAMANN B., HAGEN H., JOY K. I.: IRIS: Illustrative rendering for integral surfaces. *IEEE TVCG 16*, 6 (2010), 1319–1328. 2

[JL97] JOBARD B., LEFER W.: Creating evenly-spaced streamlines of arbitrary density. *Proc. Eurographics Workshop on Visualization in Scientific Computing 7* (1997), 45–55. 2

[KFW16] KANZLER M., FERSTL F., WESTERMANN R.: Line density control in screen-space via balanced line hierarchies. *Computers & Graphics 61* (2016), 29–39. 2, 9

[Kuh14] KUHNERT J.: Meshfree numerical schemes for time dependent problems in fluid and continuum mechanics. In *Advances in PDE modeling and computation*, Sudarshan S., (Ed.). Ane Books, New Delhi, 2014, pp. 119–136. 6

[LGP14] LAWONN K., GÜNTHER T., PREIM B.: Coherent view-dependent streamlines for understanding blood flow. In *EuroVis - Short Papers* (2014), pp. 19–23. 2

[LMSC11] LEE T.-Y., MISHCHENKO O., SHEN H.-W., CRAWFIS R.: View point evaluation and streamline filtering for flow visualization. In *Proc. IEEE Pacific Visualization* (2011), pp. 83–90. 2

[LS07] LI L., SHEN H.-W.: Image-based streamline generation and rendering. *IEEE Transactions on Visualization and Computer Graphics 13* (2007), 630–640. 2

[MCHM10] MARCHESIN S., CHEN C.-K., HO C., MA K.-L.: View-dependent streamlines for 3D vector fields. *IEEE Transactions on Visualization and Computer Graphics 16* (2010), 1578–1586. 2

[MDM10] MARCHESIN S., DISCHLER J. M., MONGENET C.: Per-pixel opacity modulation for feature enhancement in volume rendering. *IEEE Trans. on Visualization and Computer Graphics 16*, 4 (2010), 560–570. 3

[MJL*13] 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 (2013), 1342–1353. 2

[MLP*10] 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. 1

[MSRT13a] MARTINEZ ESTURO J., SCHULZE M., RÖSSL C., THEISEL H.: Global selection of stream surfaces. *Computer Graphics Forum (Proc. Eurographics) 32*, 2 (2013), 113–122. 2

[MSRT13b] MARTINEZ ESTURO J., SCHULZE M., RÖSSL C., THEISEL H.: Poisson-based tools for flow visualization. In *IEEE PacificVis* (2013), pp. 241–248. 2

[MTHG03] MATTAUSCH O., THEUSSL T., HAUSER H., GRÖLLER E.: Strategies for interactive exploration of 3D flow using evenly-spaced illuminated streamlines. In *Proc. Spring Conference on Computer Graphics (SSCG)* (2003), ACM, pp. 213–222. 2

[MWS13] MA J., WANG C., SHENE C.-K.: Coherent view-dependent streamline selection for importance-driven flow visualization. *Proc. SPIE 8654, Visualization and Data Analysis* (2013). 2

[Sci16] SCIENTIFIC VISUALIZATION CONTEST: Particular Ensembles, 2016. http://www.uni-kl.de/sciviscontest/; accessed on 2016-09-09, contest chairs: Berk Geveci and Christoph Garth. 6

[SMG*14] SCHULZE M., MARTINEZ ESTURO J., GÜNTHER T., RÖSSL C., SEIDEL H.-P., WEINKAUF T., THEISEL H.: Sets of globally optimal stream surfaces for flow visualization. *Computer Graphics Forum (Proc. EuroVis) 33*, 3 (2014), 1–10. 2

[Ste13] STEVENS B.: Introduction to UCLA-LES, 2013. 5

[TMWS13] TAO J., MA J., WANG C., SHENE C.: A unified approach to streamline selection and viewpoint selection for 3D flow visualization. *IEEE Trans. Visualization and Computer Graphics 19* (2013), 393–406. 2

[VG05] VIOLA I., GRÖLLER E.: Smart visibility in visualization. In *Proc. Computational Aesthetics* (2005), pp. 209–216. 3

[VKG04] VIOLA I., KANITSAR A., GRÖLLER E.: Importance-driven volume rendering. In *Proc. Visualization* (2004), pp. 139–146. 3

[XLS10] XU L., LEE T.-Y., SHEN H.-W.: An information-theoretic framework for flow visualization. In *IEEE Transactions on Visualization and Computer Graphics* (2010), pp. 1216–1224. 2

[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. 5

[YKP05] YE X., KAO D., PANG A.: Strategy for seeding 3D streamlines. *IEEE Visualization Conference* (2005), 471–478. 2

[YTvdW*02] YU Y., TUNG C., VAN DER WALL B., PAUSDER H.-J., BURLEY C., BROOKS T., BEAUMIER P., MERCKER Y. D. E., PENGEL K.: The HART-II test: Rotor wakes and aeroacoustics with higher-harmonic pitch control (HHC) inputs – the joint German/French/Dutch/US project. *American Helicopter Society 58th Annual Forum* (2002). 7

[YWSC12] YU H., WANG C., SHENE C.-K., CHEN J.: Hierarchical streamline bundles. *IEEE Transactions on Visualization and Computer Graphics 18*, 8 (2012), 1353–1367. 2