Vision, Modeling, and Visualization (2011)
Peter Eisert, Konrad Polthier, and Joachim Hornegger (Eds.)

A View-Dependent and Inter-Frame Coherent Visualization of Integral Lines using Screen Contribution

Tobias Günther[1], Kai Bürger[2], Rüdiger Westermann[2] and Holger Theisel[1]

[1]Visual Computing group, University of Magdeburg
[2]Computer Graphics & Visualization group, Technische Universität München

Abstract

In vector field visualization, integral lines like stream, path, or streak lines are often used to examine the behavior of steady and unsteady flows. In 3D, however, visualizing integral lines is problematic since the resulting geometric structures cause occlusions, often hiding relevant features in the data. For this reason one important goal is to find a minimum number of lines which can represent all relevant features in the vector field. In this paper we propose a novel approach that reduces the number of displayed lines, and occlusions thereof, by smoothly fading out lines based on their contribution to the viewport. In order to reduce visual clutter that is introduced by rendering multiple line contribution into one pixel, the blending equation is slightly modified. In addition, an interactive brushing is applied to further support exploration. Our approach attains a view-dependent visualization of integral lines that is inter-frame coherent and achieves real-time frame rates. To demonstrate the effectiveness and efficiency of our approach, we pursued a number of tests using real-world steady and unsteady vector fields.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

Integral lines are a popular tool for the investigation of flow data. Usually a large number of lines is required to represent a vector field and since they are simple geometric structures, many of them can be rendered in a single scene. For 2D vector fields, a variety of algorithms exist to select an appropriate number of stream lines to provide a complete covering of the flow. The problem of selecting appropriate stream lines for 3D fields is much more challenging because two competing criteria have to be fulfilled: dense covering of the domain and avoidance of visual clutter due to occlusion. Thus, the selection of appropriate integral lines is a view-dependent problem. Recently, a number of approaches for this have been introduced. None of them has all interactive frame rates, inter-frame coherence during interactive navigation, and uncut stream lines (i.e., stream lines of maximal length).

In this paper we describe the - to the best of our knowledge - first algorithm of 3D stream line selection which fulfills all of the properties mentioned above. Firstly, we select a set of candidate stream lines which guarantee a dense sampling of the domain. Then the main idea is to render *all* of them with a varying view-dependent transparency. For computing the transparency of a stream line, we propose a new line property which we call *screen contribution*. It depends on the number of fragments an integral line has on the viewport. This inherently attains inter-frame coherence, which means lines do not suddenly appear or disappear. Because of the recent advances in general purpose computing, our GPU implementation achieves real-time frame rates for reasonably complex flows. Since flow visualizations have a high depth complexity, we slightly modify the utilized order-independent transparency technique to decrease the otherwise high perceptual load.

The remainder of the paper is structured as follows. Related work is referred in section 2. Afterwards, we introduce our approach (3). In the sequent section (4), we describe the interactive brushing technique we additionally applied. In section 5 the implementation is discussed and section 6 addresses the performance and compares to results from related work. Finally, section 7 concludes our achievements and encourages further research. To test the algorithms the

3D vector field sequences 'Stuttgart LES' [FWT08] and 'Square Cylinder' [Int] were used as well as the steady field 'Benzene' provided by Hans-Christian Hege, which shows the electrostatic field around a benzene molecule. The uniformly resampled version of the 'Square Cylinder' sequence is provided by Tino Weinkauf (cf. [vFWTS08]) and is based on the Navier-Stokes simulation of Camarri et al. [CSBI05].

2. Related Work

In the last years many approaches were proposed for 2D vector fields. Turk and Banks [TB96] and Jobard and Lefer [JL97] early contributed influencing work. By now, the seeding problem in 2D is essentially solved [VKP00, JL01, MAD05, LMG06, LHS08]. The first 3D seeding approaches were density-based (Mattausch et al. [MTHG03]), feature-based (Ye et al. [YKP05]) or similarity-based (Chen et al. [CCK07]). None of these approaches addressed occlusion, the problem on which we focus.

Li et al. [LS07] introduced the first view-dependent 3D seeding strategy, by executing the 2D seeding strategy of Jobard and Lefer [JL97] to produce evenly-spaced stream lines in image space. Since the seeding of lines requires a depth value at every pixel, several alternatives for generating depth maps were suggested. For a better representation of the vector field the seeding algorithm was executed from different points of view and the seedpoint sets were combined for the final image. To regard inter-frame coherence the stream lines from the previous frame were validated first. However, newly added lines introduce popping artifacts.

Annen et al. [ATR*08] applied ideas from non-photorealistic rendering to the seeding and integration of streamlines and thereby introduced vector field contours. These are selected stream lines that have the most similar behavior to contours on surfaces. Their seeding points depend on the view direction, additional parameters and local conditions. The forward and backward integration is stopped, if the similarity to surface contours exceeds a certain threshold. The extraction and rendering is inter-frame coherent and achieves real-time frame rates.

Marchesin et al. [MCHM10] proposed a view-dependent stream line algorithm, in which view-dependent and view-independent properties were used for the selection of stream lines. For the view-dependence an occupancy buffer was computed. In this the number of overlapping lines for each pixel is stored, divided by the line thickness. The occupancy contributed to the pruning of a precomputed stream line set and controlled the filling of empty regions with newly seeded stream lines. The view-independent properties were the linear entropy by Furuya et al. [FI08], which encodes the uniformity of the stream line's velocity and the angular entropy, which constitutes the uniformity of the stream line's curvature. In the current state, this approach is neither interactive nor inter-frame coherent.

Recently Lee et al. [LMSC11] suggested to use a conventional maximum intensity projection (MIP) of the scalar entropy field, called maximum entropy projection (MEP). Based on the thereby yielded MEP-framebuffer they pruned a stream line set, which was precomputed by the seeding algorithm of Xu et al. [XLS10]. This interactive visualization was further improved by an optimization of the viewpoint by maximizing the accumulated entropy of all pixels of the viewport. This seeding algorithm does not yet maintain inter-frame coherence.

3. Screen Contribution

If the user can navigate interactively in the scene, he might look at objects he is interested in or move closer to them. Lines covering more area on the screen are assumed to be of interest, while lines having only a few pixels on the screen do not contain much information and can thus be rejected. Our approach emphasizes integral lines that appear to be of interest for the user by letting unimportant lines vanish. Therefore, we propose a parameter that is defined per integral line and call it *screen contribution* $S(i)$.

In a scene with many lines the depth complexity is very high. Each pixel could contain hundreds of fragments that have to be sorted. In order to deal with high depth complexities, an order-independent transparency algorithm is needed that maintains the required memory dynamically. Fragment linked lists [YHGT10] are currently the best choice for this application, because memory is taken from a large memory pool only if it is needed and the technique requires just one rendering pass. But there are two disadvantages: For the memory allocation from the memory pool all threads have to be synchronized at one single counter, which is a bottleneck. And the execution passes of the threads during the bitonic sort differ very often, which makes dynamic branching slow. Nevertheless, the performance is good enough and no better options are available so far.

3.1. Mapping to Transparency

Let the value $N(i)$ constitute the number of pixels the line with id i has in the first depth layer. Let i be the line id, w the screen width, h the screen height and f a function that returns one, if the first fragment of the fragment linked list at position (x,y) has the integral line id i.

$$N(i) = \sum_{x=0}^{w-1}\sum_{y=0}^{h-1} f(x,y,i)$$

$$f(x,y,i) = \begin{cases} 1 & front(x,y) = i \\ 0 & else \end{cases}$$

If the user looks at an integral line, $N(i)$ is high and increases if the user moves closer to the line. Lines that are mostly covered are less interesting at this view. If the user is more interested in a line, he has to adjust the camera position to

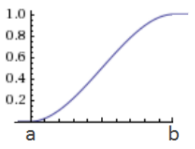


Figure 1: Cubic Hermite Interpolation

get a better look. A cubic hermite interpolation (Figure 1) between the borders a and b is used in order to map $N(i)$ to an opacity value in $[0..1]$. The High Level Shading Language (HLSL) offers an intrinsic function called *smoothstep* that implements this interpolation.

$$\begin{aligned} smoothstep(a,b,x) &= 3t(a,b,x)^2 - 2t(a,b,x)^3 \\ t(a,b,x) &= saturate\left(\frac{x-a}{b-a}\right) \end{aligned}$$

The screen contribution S(i) is then computed as

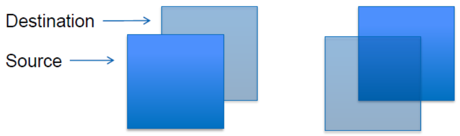
$$S(i) = smoothstep(a,b,N(i)) \tag{1}$$

Note that a and b are parameters set by the user. Different choices of a and b are shown in figures 8, 11 and 12 later on.

3.2. Screen Contribution Blending

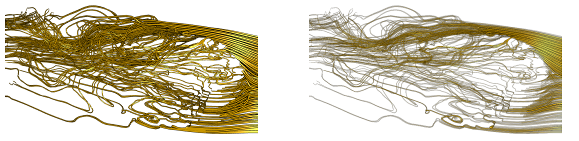
The human visual perception is capable to distinguish only a rather limited number of transparent objects (compare figures 2a, 2b and 2c). To reduce the information encoded by a certain pixel, we modify the blending equation. When blending two fragments, the more interesting fragment is known, since important and less important integral lines can be already distinguished. If the source fragment is more interesting than the destination fragment, the destination fragment is rejected entirely by setting the opacity of the source pixel to one. In the opposite case the two fragments get interpolated by the alpha value of the source fragment (see figure 3). This means, important integral lines are always visible even if they are partially covered by a less important line, because the covering pixel is transparent. The minimum transparency of all fragments in the list is used to blend the final fragment color with the backbuffer (see figures 2c and 4). The modified back-to-front blending operation looks as follows, if all fragments are sorted by depth from far to near: $\forall n : z_n > z_{n+1}$

$$f_{sc}(n) = \begin{cases} c_0 & n = 0 \\ f_{sc}(n-1)\alpha_n + c_n(1-\alpha_n) & S(n) < S(n-1) \\ c_n & else \end{cases}$$



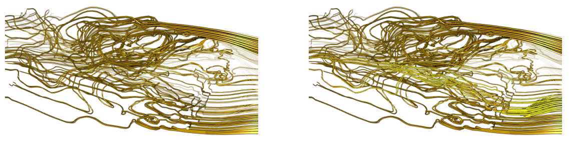
Left: Front fragment has higher screen contribution than the fragment behind and thus the color behind is rejected entirely. Right: Front fragment has lower screen contribution. It is rendered transparently.

Figure 3: Depiction of the screen contribution blending.



(a) 500 opaque path lines with 500 segments each, depth-dependent halos and 4x MSAA at 55fps.

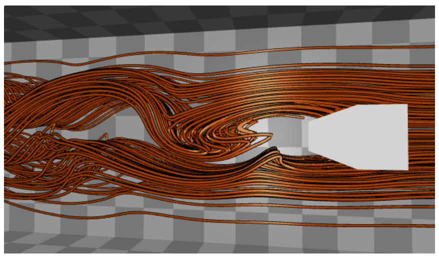
(b) Fragment linked lists with uniform transparency and standard blending. Achieves 26fps at 4x MSAA.



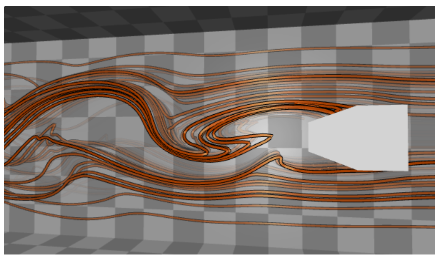
(c) Fragment linked lists with screen contribution blending. Achieves 16fps at 4x MSAA.

(d) Fragment linked lists with screen constribution blending and brushing. Achieves 13fps at 4x MSAA.

Figure 2: Screen contribution blending and brushing compared to standard blending.



(a) Opaque streak lines.



(b) Using screen contribution.

Figure 4: 400 streak lines with 4x MSAA at 34 fps.

4. Interactive Brushing

Since the fading to transparency is based on a global criterion, the decision might not be ideal locally. Thus, we implemented an exploration tool that allows to fade selected lines back in. For the selection of lines or line bundles many techniques are imaginable. In our implementation the viewer selects one integral line and in its screen-space neighborhood all similar lines in a certain window are selected as well. We preferred to select similar lines to support the user in exploring regions of coherent flow. Assuming a window size of w times h pixels, the rectangular window is spanned around the selected pixel P_{sel} by the points:

$$\begin{aligned} P_{topleft} &= \left(P_{sel}.x - \frac{w}{2}, P_{sel}.y - \frac{h}{2}\right) \\ P_{bottomright} &= \left(P_{sel}.x + \frac{w}{2}, P_{sel}.y + \frac{h}{2}\right) \end{aligned}$$

For each pixel inside the window the program iterates over the fragment linked lists and compares the recorded integral lines to the selected line by using the following similarity metric.

$$similarity(i,j) = \sqrt{(E_A(i) - E_A(j))^2 + (E_L(i) - E_L(j))^2} \quad (2)$$

Furuya et al. [FI08] showed an equation for the computation of the linear entropy E_L and Marchesin et al. [MCHM10] formed an equation for the angular entropy E_A (first line of equation 3), but in the proposed form, two iterations over all line segments are necessary. For this reason we rearranged the formulas into a form requiring only a single pass. See equation 3 for the angular entropy. The rearrangement of the linear entropy formula is analog. This allows to compute the measures on-the-fly during the integration, without any need for an additional iteration over all line segments.

m: number of line segments
A_j: absolute value of the angle at the j-th line joint
$L_A = \sum_{j=0}^{m-1} A_j$

$$\begin{aligned} E_A &= -\frac{1}{log_2(m)} \sum_{j=0}^{m-1} \frac{A_j}{L_A} log_2 \frac{A_j}{L_A} \quad (3) \\ &= -\frac{1}{log_2(m)} \left(\frac{\sum_{j=0}^{m-1} A_j log_2 A_j}{L_A} - log_2 L_A \right) \end{aligned}$$

All selected lines set their respective highlight flag $h_{id} \in \{0,1\}$ in a buffer, which is cleared to begin at each frame. In an additional interest buffer, the interest $i_{id} \in [0,1]$ is computed smoothly by progressive fading.

$$i_{id} := i_{id} + (h_{id} - i_{id}) \cdot f \qquad f \in \mathbb{R}, 0 < f \leq 1 \quad (4)$$

Thereby i_{id} moves each frame f percent closer to the aimed value, i.e. h_{id}. The context is visualized by using screen contribution blending, while the focus region is rendered fully opaque with slight over exposure (see figure 2d).

5. Implementation

The following section describes the implementation of the screen contribution blending in more detail. The integral lines were seeded by the user and use fourth order Runge-Kutta integration with fixed step size. During the integration, the linear and angular entropy are computed on-the-fly (see equation 3) and stored in a byte address buffer, which we call metrics buffer. Yang et al. [YHGT10] described how to create fragment linked lists concurrently during rendering. Their approach is available in the DirectX SDK [DxS09] and was further improved by Yakiimo [Yak10]. The steps of our approach are depicted in figure 5.

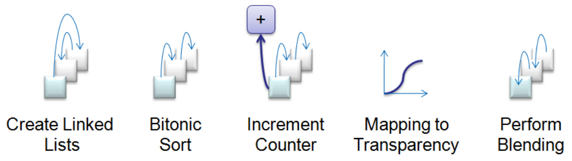


Illustration of the five steps performed by our approach.

Figure 5: Concept of Screen Contribution OIT.

In a pre-processing pass we create fragment linked lists, which requires two buffers. The first buffer is a structured buffer that serves as fragment pool for all fragments on the screen, because dynamic memory allocation is not possible on the GPU so far. Each fragment consists of a color (8 bit per component quantized), a depth value (32 bit), the ID of the line it belongs to and an index in the linked list buffer. The index references the predecessor in the linked list. The second buffer is a byte address buffer, thus it allows atomic operations. This buffer stores for each sample the entry in the linked list buffer. This entry point is the last fragment of the samples linked list. As already mentioned, each fragment stores a reference to its predecessor. The value -1 indicates the end of a linked list. In the pre-processing step all objects append their fragments to the list. Thereby the pixel shaders allocate an address in the fragment pool by atomically incrementing the hidden counter of the structured buffer. This unique value is the array index in the pool. An interlocked exchange operation is executed on the start offset buffer to store the index of the new fragment and to get access to the predecessor's index. Figure 6 depicts a sample for a 4x3 viewport. In this sample a green rectangle and an orange circle are rendered. On the top right the start offset buffer is visible and on the bottom right the fragment buffer is depicted. In a sequent fullscreen pass all fragments get sorted by a bitonic sort. When the sorting is finished, the threads atomically increment the fragment counter of the respective integral line that is in front of the pixel, which is identified by the ID stored with the fragment. Afterwards the number of fragments on the screen is known per integral line and is mapped to opacity (equation 1). When brushing is enabled, two compute shaders are executed. The first

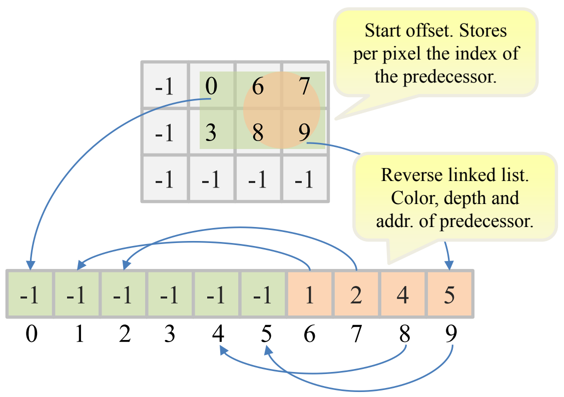

Start offset. Stores per pixel the index of the predecessor.
-1 0 6 7
-1 3 8 9
-1 -1 -1 -1
Reverse linked list. Color, depth and addr. of predecessor.
-1 -1 -1 -1 -1 -1 1 2 4 5
0 1 2 3 4 5 6 7 8 9

Figure 6: Sample illustration of fragment linked lists for a 4x3 viewport. Top: start offset buffer. Bottom: fragment pool.

one invokes a thread for each pixel around the cursor position in a user-defined window. These threads use the similarity metric (equation 2) to set the highlight flags h_{id} in a byte address buffer. The sequent compute shader executes a thread for each streamline and smoothly fades the transparency, which is stored in another byte address buffer, by using equation 4. Finally, the screen contribution blending operation is executed in a fullscreen pass by iterating over all fragment linked lists in parallel. If brushing is enabled, the pixel shaders additionally fetch the brushing transparency from the aforementioned byte address buffer. To enhance the appearance of the lines, illuminated stream line shading by Zöckler et al. [ZSH96] and depth-dependent halos by Everts et al. [EBRI09] are used.

6. Evaluation

In the following we analyze the performance, compare our results to related work and discuss limitations.

6.1. Performance

In this section the performance of our approach is analyzed. Since the algorithm is pixel shader bound, we list the number of pixel shader invocations. All performance measures were taken on a Nvidia GeForce GTX 460.

fps	10	20	30	40	50	60
triangles	306k	336k	380k	391k	388k	388k
PS exec.	6.36M	2.77M	1.53M	934k	767k	581k

Table 1: Performance without brushing

Table 1 shows the performance of our approach in the Stuttgart LES data set. 400 path lines are integrated each frame using an RK4 integration and 4x MSAA is utilized. First row is the achieved frame rate. Beneath the number of triangles drawn and the number of pixel shader invocations are listed. The statistics were conducted by looking at the scene from different distances and directions.

fps	10	20	30	40
triangles	310k	330k	334k	334k
PS exec.	5.54M	1.58M	502k	119k

Table 2: Performance with brushing

Table 2 lists the achieved frame rates with brushing enabled.

6.2. Comparison

An informal survey has shown that the important structures are more visible than if lines are randomly generated. This can be seen in figure 7. Here we compare a visualization of a benzene molecule rendered with vector field contours by Annen et al. [ATR*08] to our screen contribution blending. It is denotable that our approach shows all important structures, reduces occlusion and gives more context information than the vector field contours. Further comparisons to related work were not possible, since the data sets were not available to us. However, in similar scenarios our approach shows the important structures and attains all three goals: interactive frame rates, inter-frame coherence during interactive navigation, and uncut stream lines. The figures 9, 11 and 12 show some examples.

6.3. Limitations

Our approach is limited by the amount of memory required for the fragment linked lists. Furthermore this approach is pixel shader bound, as we have shown in the performance sections (section 6.1). The visual results of our approach depend on the initial seeding. In our implementation the user can place probes by creating seed boxes. Usually, the seed boxes contain the entire vector field and the seeds are placed randomly to cover the entire domain. During screen contribution blending a binary decision is made, whether fragments should be rejected or not. If this decision changes between two frames, the final color alters. This happens rarely and is in practice barely noticeable, since it only differs by one transparent layer. Our approach is well suited to reduce integral lines from dense and turbulent regions (see figure 8 and 9). However, our approach does not reduce strong occlusion, i.e. if two interesting regions lie behind each other, only the first is represented well. The same statement is true for all previous approaches referred in the related work section. Thus, this problem is still unsolved. But, since our approach is interactive and the seeding is user-driven, the user can take corrective action at any time by adapting the seeding to reduce this problem and thereby improve the exploration of the data set. In figure 10 the covered stream is visible, since it contributes to the viewport on the top of the image. Since we evaluate our visibility metric per integral line, covered streams can be revealed under this condition.

7. Conclusion and Future Work

An interactive, view-dependent visualization of integral lines requires the isolation of 'important' lines, i.e. lines that represent the vector field. Therefore the *screen contribution* was introduced, which is the absolute number of fragments on the screen, a value that was mapped to transparency. To reduce the information density a modified blending operation was proposed, called *screen contribution blending*. The approach uses state-of-the-art order-independent transparency techniques, i.e. fragment linked lists generated using Direct Compute. In addition this approach was combined with an interactive brushing technique to enable user-driven exploration. The approach is improvable since properties of the lines were not regarded so far, e.g. the importance of an occlusion or the revelation of vortices. Additionally further Non-photorealistic Rendering approaches could be utilized to enhance the visualization and automatic seeding strategies could be used in a hybrid combination. The screen contribution blending could also be used for other visualization tasks that are limited by high transparency complexity, e.g. volume rendering.

8. Acknowledgment

We thank John McLaughlin for English proof reading.

References

[ATR*08] ANNEN T., THEISEL H., RÖSSL C., ZIEGLER G., SEIDEL H.-P.: Vector field contours. In *Proc. Graphics Interface* (2008). 2, 5, 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

[CSBI05] CAMARRI S., SALVETTI M.-V., BUFFONI M., IOLLO A.: Simulation of the three-dimensional flow around a square cylinder between parallel walls at moderate Reynolds numbers. In *XVII Congresso di Meccanica Teorica ed Applicata* (2005). 2

[DxS09] Oit11 sample. Microsoft DirectX SDK, August 2009. 4

[EBRI09] EVERTS M. H., BEKKER H., ROERDINK J. B. T. M., ISENBERG T.: Depth-dependent halos: Illustrative rendering of dense line data. *IEEE Transactions on Visualization and Computer Graphics 15* (November 2009), 1299–1306. 5

[FI08] FURUYA S., ITOH T.: A streamline selection technique for integrated scalar and vector visualization. In *Vis Š08: IEEE Visualization Poster Session* (2008). 2, 4

[FWT08] FREDERICH O., WASSEN E., THIELE F.: Prediction of the flow around a short wall-mounted cylinder using les and des. *Journal of Numerical Analysis, Industrial and Applied Mathematics (JNAIAM) 3*, 3-4 (2008), 231–247. 2

[Int] International CFD Database, http://cfd.cineca.it/. 2

[JL97] JOBARD B., LEFER W.: Creating evenly-spaced streamlines of arbitrary density. *Proceedings of the Eurographics Workshop on Visualization in Scientific Computing '97 7* (1997), 45–55. 2

[JL01] JOBARD B., LEFER W.: Multiresolution flow visualization. *WSCG 2001 Conference Proceedings* (February 2001), 33–37. 2

[LHS08] LI L., HSIEN H. H., SHEN H. W.: Illustrative streamline placement and visualization. *IEEE Pacific Visualization Symposium 2008* (2008), 79–86. 2

[LMG06] LIU Z., MOORHEAD R., GRONER J.: An advanced evenly-spaced streamline placement algorithm. *IEEE Transactions on Visualization and Computer Graphics 12* (September 2006), 965–972. 2

[LMSC11] LEE T.-Y., MISHCHENKO O., SHEN H.-W., CRAWFIS R.: View point evaluation and streamline filtering for flow visualization. In *Proceedings of the IEEE Pacific Visualization Symposium 2011* (March 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* (May 2007), 630–640. 2

[MAD05] MEBARKI A., ALLIEZ P., DEVILLERS O.: Farthest point seeding for efficient placement of streamlines. In *IEEE Visualization* (2005), p. 61. 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* (November 2010), 1578–1586. 2, 4

[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* (New York, NY, USA, 2003), SCCG '03, ACM, pp. 213–222. 2

[TB96] TURK G., BANKS D.: Image-guided streamline placement. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 453–460. 2

[vFWTS08] VON FUNCK W., WEINKAUF T., THEISEL H., SEIDEL H.-P.: Smoke surfaces: An interactive flow visualization technique inspired by real-world flow experiments. *IEEE Transactions on Visualization and Computer Graphics 14* (November 2008), 1396–1403. 2

[VKP00] VERMA V., KAO D., PANG A.: A flow-guided streamline seeding strategy. In *Proceedings of the conference on Visualization '00* (Los Alamitos, CA, USA, 2000), VIS '00, IEEE Computer Society Press, pp. 163–170. 2

[XLS10] XU L., LEE T.-Y., SHEN H.-W.: An information-theoretic framework for flow visualization. In *IEEE Transactions on Visualization and Computer Graphics* (Nov.-Dec. 2010), vol. 16(6), pp. 1216–1224. 2

[Yak10] YAKIIMO02: Dx11 order independent transparency with msaa. Codeplex, July 2010. 4

[YHGT10] YANG J. C., HENSLEY J., GRÜN H., THIBIEROZ N.: *Real-Time Concurrent Linked List Construction on the GPU*, vol. 29. Wiley-Blackwell, June 2010, pp. 1297–1304(8). 2, 4

[YKP05] YE X., KAO D., PANG A.: Strategy for seeding 3d streamlines. *Visualization Conference, IEEE 0* (2005), 60. 2

[ZSH96] ZÖCKLER M., STALLING D., HEGE H.-C.: Interactive visualization of 3d-vector fields using illuminated stream lines. In *Proceedings of the 7th conference on Visualization '96* (Los Alamitos, CA, USA, 1996), VIS '96, IEEE Computer Society Press, pp. 107–ff. 5

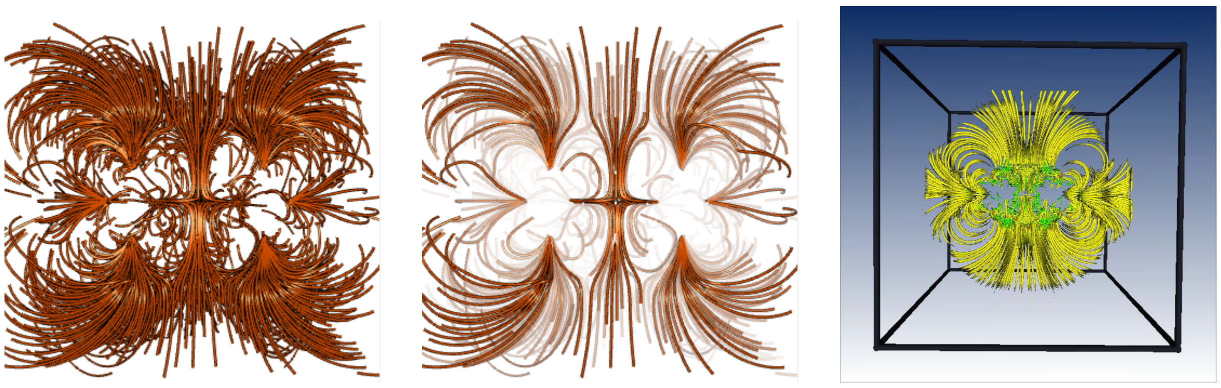


Figure 7: Left: 1200 randomly placed stream lines in the benzene data set. Center: Activation of screen contribution blending reduces occlusion and gives more context information than vector field contours. Right: Vector field contours from Annen et al. [ATR*08]

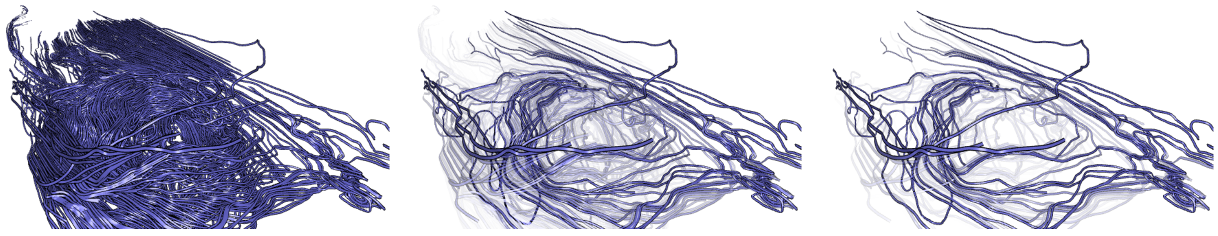


Figure 8: Left: path lines are opaque. Center and right: screen contribution blending with different mappings. Please notice the reduction of occlusion.

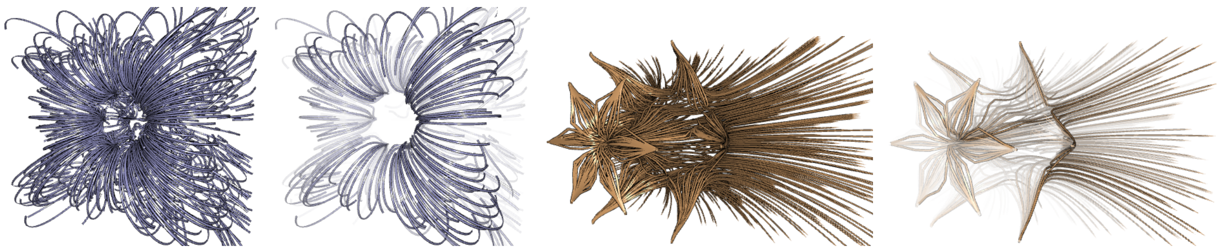


Figure 9: These figures depict the benzene data set. The two images on the left oppose a random seeding to the screen contribution blending and show the reduction of occlusion introduced by our approach. The images on the right depict the same scene with different seed points, again with screen contribution blending disabled (left) and enabled (right).

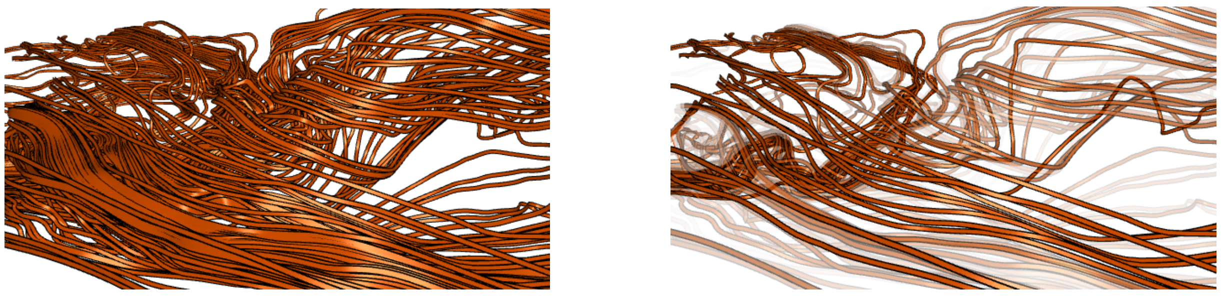


Figure 10: The random seeding on the left produces occlusion which makes it impossible to see the stream behind. Our approach (on the right) does not only improve the perceptibility. It can make structures visible that would be occluded.

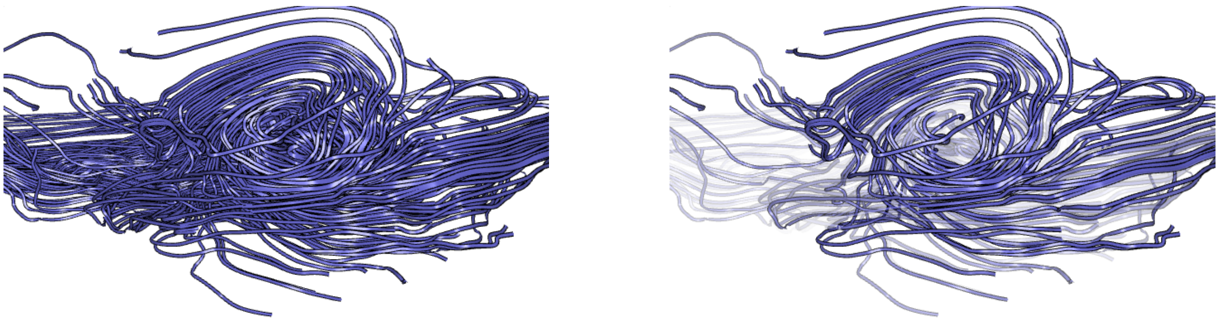


Figure 11: 400 path lines. Left: opaque. Right: screen contribution blending at 20 fps, 4x MSAA, a=200, b=5000

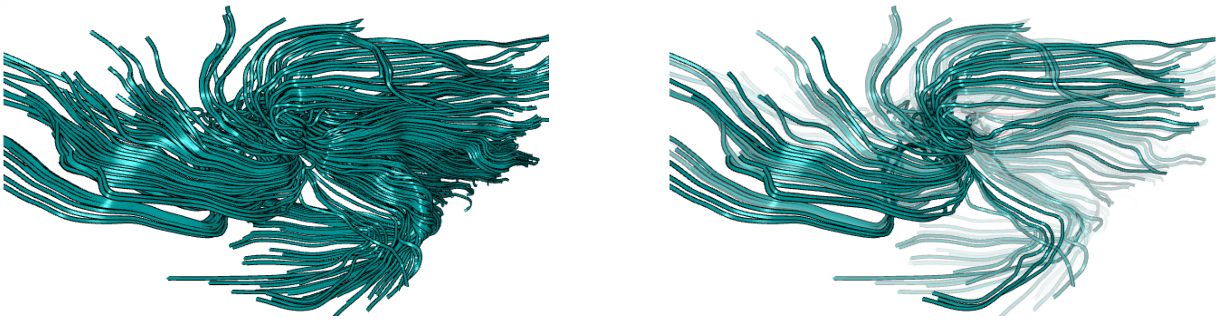


Figure 12: 400 path lines. Left: opaque. Right: screen contribution blending at 19 fps, 4x MSAA, a=500, b=4500

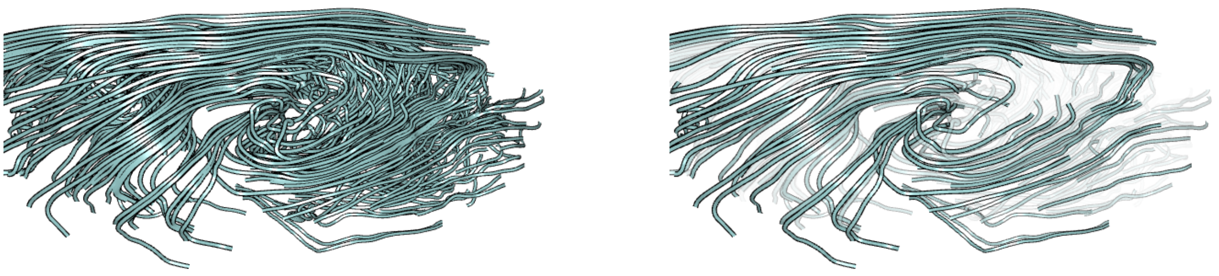


Figure 13: Another example in the stuttgart LES data set. The left set of randomly placed stream lines serves as input for our algorithm yielding the image on the right.

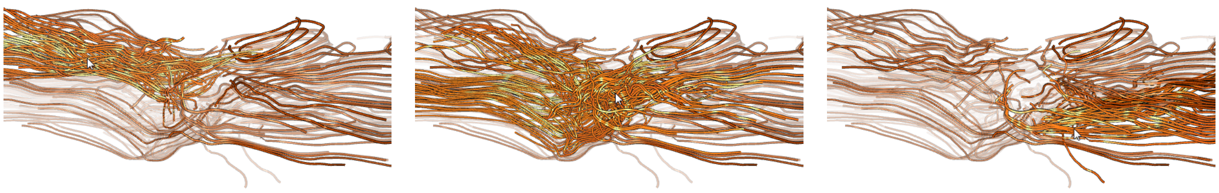


Figure 14: Figures that show the brushing technique. The user has the possibility to render all stream lines in his focus region fully opaque with slight overexposure in order to explore where streams came from.