Feature-based Visualization of Dense Integral Line Data

Simon Schröder[1,2], Harald Obermaier[4], Christoph Garth[3], and Kenneth I. Joy[4]

1 **Computer Graphics and HCI Group**
University of Kaiserslautern, Germany
2 **Fraunhofer ITWM, Kaiserslautern, Germany**
`simon.schroeder@itwm.fraunhofer.de`
3 **Computational Topology Group**
University of Kaiserslautern, Germany
`garth@cs.uni-kl.de`
4 **Institute for Data Analysis and Visualization,**
University of California Davis, USA
`hobermaier@ucdavis.edu, joy@cs.ucdavis.edu`

Abstract

Feature-based visualization of flow fields has proven as an effective tool for flow analysis. While most flow visualization techniques operate on vector field data, our visualization techniques make use of a different simulation output: Particle Tracers. Our approach solely relies on integral lines that can be easily obtained from most simulation software. The task is the visualization of dense integral line data. We combine existing methods for streamline visualization, i. e. illumination, transparency, and halos, and add ambient occlusion for lines. But, this only solves one part of the problem: because of the high density of lines, visualization has to fight with occlusion, high frequency noise, and overlaps. As a solution we propose non-automated choices of transfer functions on curve properties that help highlighting important flow features like vortices or turbulent areas. These curve properties resemble some of the original flow properties. With the new combination of existing line drawing methods and the addition of ambient occlusion we improve the visualization of lines by adding better shape and depth cues. The intelligent use of transfer functions on curve properties reduces visual clutter and helps focusing on important features while still retaining context, as demonstrated in the examples given in this work.

1998 ACM Subject Classification I.3.8 Computer Graphics Application

Keywords and phrases Flow simulation, feature-based visualization, dense lines, ambient occlusion

Digital Object Identifier 10.4230/OASIcs.VLUDS.2011.71

1 Introduction

Flow simulation has a long history in scientific computing. For several decades simulation was limited to 2D because of symmetry or for computational considerations. Hence, a lot of excellent methods for integral flow visualization have been developed, e. g. LIC (Line Integral Convolution). Now, with the availability of more computational power, simulations have expanded to three dimensions. This leaves the scientific visualization community with new challenges for the visualization of flows in 3D. Existing techniques from two dimensions cannot be easily adapted for 3D as most display devices are still 2D.

Proceedings of IRTG 1131 – Visualization of Large and Unstructured Data Sets Workshop 2011.
Editors: Christoph Garth, Ariane Middel, Hans Hagen; pp. 71–87
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Integral lines, i. e. streamlines and pathlines, are a common way to show the structure of a flow. Several techniques like illuminated lines, halos, and tube-like rendering have been developed to enhance rendering of these lines. In many cases there are too few or too many lines showing a lot of unnecessary information instead of focusing on important flow features.

For our visualization we solely use integral lines as output from simulations without the need of the underlying flow field. This makes it easy to use our visualization approach as a post-processing step with any common flow simulation software that can produce integral lines as output. There also are some tools available for visualizing integral lines. In contrast to these existing methods we require a dense sampling of integral lines. The idea behind this is that we are then able to provide high resolution visualization results in important regions and adaptively dim out unwanted information.

Visualization of dense lines has two major problems: overlapping of lines and occlusion of lines. These problems are even more severe with our requirement of dense sampling of integral lines. There are already solutions to these two problems: (1) halos around lines reduce overlap and provide visual separation, and (2) transparency reduces occlusion and reveals hidden lines.

In this paper we combine illuminated lines, transparency, and halos for an improved line visualization. To stress the spatial relationship of lines we introduce an ambient occlusion technique that is suitable for dense line data sets. Transparency is mainly used to highlight interesting flow features. The second contribution of our work is the definition of interactive transfer functions on curve properties that allow to extract otherwise hidden structures, e. g. vortices and turbulent regions. In addition, transfer functions on colors or standard color maps can be applied for illustrative purposes or deeper insight into the flow.

In the next section we start by discussing existing related work. Section 3 motivates the two problems that we are solving: improvement of visual quality and feature extraction. Section 4 lays the mathematical foundation for the calculation of curve properties, e. g. curvature and torsion. Section 5 picks up on this and explains how to use curve properties to extract flow features. Ambient occlusion is described in Section 6. Here, we discuss ambient occlusion in more detail, including the theoretical background, and explain the structure of our voxel based algorithm. Section 7 details all necessary parts of the visualization: we shortly describe lighting and transfer functions. In the results section we show some examples of feature-based visualization of integral lines. In a separate subsection we investigate different parameter settings of our algorithm for ambient occlusion computation. We conclude the paper in the last section and suggest areas for future research.

2 Related Work

2.1 Lines

The basis for illuminated lines has been laid out by Zöckler et al. [30]. They introduced new techniques for the calculation of Phong illumination for line primitives. Later, Schussman and Ma used this approach in volume rendering [23]. Also, the techniques for basic illuminated streamlines have been updated to current graphics hardware using shaders [11].

In many cases, line data is not rendered as line primitives on graphics cards at all. Instead, visualization is achieved by either drawing tubes or, more often, self-orienting surfaces [22, 10, 3]. The latter technique has been enhanced introducing bump mapping [10], where the surfaces are rendered like tubes.

A common problem, especially when introducing transparency, is a lack of depth perception for the entire scene. This problem has been addressed using halos [13]. Special operations on the graphic card depth buffer can improve depth perception through halos by making their width depth-dependent [3]. In this paper, we discuss an implementation which has the same properties, but uses OpenGL's line primitives instead of self-orienting surfaces.

There are further approaches connected to our visualization that we discuss in the following. The method of Mattausch et al. [13] provides means for additional seeding of streamlines in regions of interest. As already mentioned, we do not make use of the vector field and hence are able to develop methods that work on arbitrary line input. Hence, we will not discuss the differences of other methods [6, 9, 28].

Another approach developed for pathlines is provided by Shi et al. [24]. They implemented methods for querying of pathline attributes. Interactive brushing and focus+context visualization methods provide new means for investigation of pathlines.

In a two-dimensional setting, a number of methods for enhancing streamline visualization were discussed previously. Most common are methods for streamline clustering [1], or intelligent seeding of streamlines (cf. [27] and [8]). Some newer approaches extend such techniques to 3D (cf. [12]). We will not use any of these approaches in our visualization.

A further application for line visualization is rendering of white matter fibers of the human brain (e. g. [2]). Visual grouping and distinction of homogeneous paths can again be achieved by clustering. Otten et al. [15] used a combination of hint lines, colored halos, silhouettes, and outlines to enhance cluster visualization for illustrative output.

2.2 Ambient Occlusion

Ambient occlusion is a well established area of research. Ray tracing techniques do not need ambient occlusion as calculation of physically accurate lighting already includes shadows. Ambient occlusion mimics some of this behavior for arbitrary illumination models. Most non-real-time approaches use object space techniques, but for real time rendering Screen Space Ambient Occlusion (SSAO) can be applied. A short overview of the results of different approaches can be found in [14].

For rendering of hair or fur there exist no real ambient occlusion methods yet. Usually, the problem can be solved by depth-based approaches [29, 7]: Hair under the surface, for instance, is occluded and hence darker.

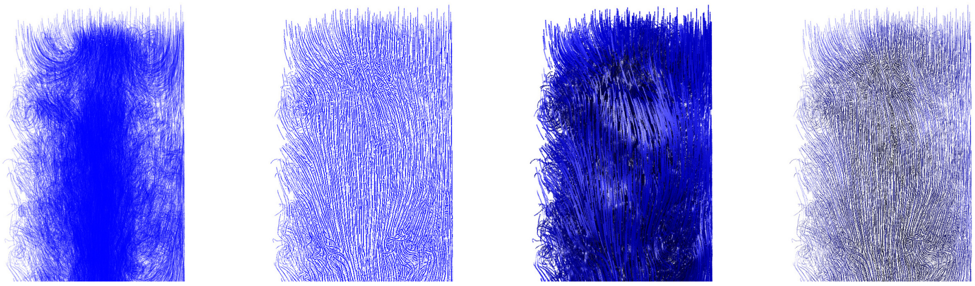
A commonly used approach to accelerate the calculation of ambient occlusion is to voxelize the scene [18]. This approach easily boosts ray tracing for the occlusion calculation by approximation of the scene.

2.3 Flow Features

Flow features have been extensively investigated. A good overview over existing methods is provided by Post et al. [17]. Related to our selection of flow features is the approach by Salzbrunn and Scheuermann [21]. The problem of vortex detection/vortex core extraction has not been solved entirely. Good approaches to this can be found in [20] and [5].

3 Motivation

The visualization of dense integral line data has many challenges. They can be split into the visualization problems and problems concerning the data directly, i. e. extraction of important flow features.



(a) Using transparency of 40% to reduce occlusion of lines: more context, no shape or depth cues.

(b) Halos have been traditionally used to provide depth cues. In this paper they are used for resolving overlaps.

(c) Enhancing illuminated lines with our ambient occlusion technique: good shape and depth cues.

(d) Combining illuminated lines, ambient occlusion intensity mapping, transparency of 60%, and transparent halos into one image.

Figure 1 Comparison of the effects of different line drawing techniques.

3.1 Improvement of Visual Quality

Obvious problems in the visualization are overlapping of lines and occlusion of lines in the back. Common approaches to solve these two problems conflict each other.

Transparency, as discussed in the related work section, is a common method to reduce occlusion (see Figure 1a). But, at the same time it reduces separation of lines at overlaps. On the other hand overlaps are resolved by using halos around the lines (see Figure 1b). This provides a better separation of lines. But again, halos add to the width of the line and occlude more lines in the back.

More important for the understanding of the visualization is the perception of shape and depth of the lines.

Illumination reveals the shape and curvature of the lines. This has been done right from the beginning (see [30]). But, this approach only works well for few lines as there is no solid visual clue for spatial ordering of lines.

Spatial ordering or depth perception has been generally solved by using halos. We already discussed one problem of using halos. Another problem is that in order to keep the spatial cues halos cannot be reasonably combined with transparency.

Our solution to this is that we use halos only for visual separation of lines. In this case halos are still useful when combined with transparency. Instead, we use ambient occlusion to improve visual spatial ordering of lines (see Figure 1c). For this, we had to adapt existing methods for the computation of ambient occlusion to work with lines and allow for partial occlusion.

The combination of these four methods, i. e. illumination, transparency, halos, and ambient occlusion, still does not reduce the information overload when used with dense line data (see Figure 1d). Therefore, we extract curve features to highlight important regions and dim out unnecessary information.

3.2 Feature Extraction

In order to provide better visualizations which focus on interesting and important regions we use transfer functions on curve properties such as length, curvature, and torsion. As we require that we do not need access to the whole simulation data but just the generated lines we use numerical methods to approximate these properties. We show with our results that this is sufficient to highlight regions of interest.

4 Curve Properties

In the following we discuss the computation of curve properties. For the scope of this paper we concentrate on integral lines computed from flow simulation. But, in general we could use arbitrary line data that represents a polyline by a sequence of points.

All the computed curve properties are scalar properties that can be easily mapped to color or transparency by using transfer functions. Beside computing the properties locally for each point of the polyline we additionally compute some global properties by computing minima, maxima, and the average.

Throughout the remainder of this section we use the following conventions for variables of a single curve or polyline:

- $v_i, i = 1, \ldots, n$ are n vertices describing the curve as a polyline
- $s_i, i = 1, \ldots, n-1$ are the segments of the line enclosed by v_i and v_{i+1}.

Local curve characteristics at a given vertex i are given the same index. Contrarily, variables without an index are always global curve parameters.

The segment length d_i is computed by the distance of neighboring vertices

$$d_i = \|v_{i+1} - v_i\|. \tag{1}$$

Assuming a reasonable sampling of the curve we get its length l by summation

$$l = \sum_i d_i. \tag{2}$$

In addition to this we also compute the minimum segment length d_{min}, the maximum segment length d_{max}, and the average segment length of a curve d_{avg}.

4.1 Derivative-based Curve Features

4.1.1 Curvature

For the input we only required that is has to fulfill C^0 continuity. Hence, we approximate derivatives through finite differences. The first derivative is the tangent t_i computed by a central difference scheme (cf. [25]). Accordingly, the second derivative c_i is computed by a first order scheme using the tangents t_i. This yields the curvature κ_i:

$$\kappa_i = \frac{\|t_i \times c_i\|}{\|t_i\|^3}. \tag{3}$$

4.1.2 Torsion

For approximation of the torsion τ_i we need the third derivative w_i along the curve. This derivative again is obtained by a first order finite difference of the second derivatives c_i. Finally, the formula for the torsion τ_i reads:

$$\tau_i = \frac{\|(t_i \times c_i) \cdot w_i\|}{\|t_i \times c_i\|^2}. \tag{4}$$

4.2 Depth

The depth of a point is computed as the distance to the camera plane. In contrast to the previous parameters this one is dynamic and has to be updated each time the camera moves. The depth parameter can be used in visualization to clip unnecessary lines that occlude flow features on the inside of the flow volume.

4.3 Ambient Occlusion

Ambient occlusion is also computed for each vertex of the polyline. Details concerning the methodology and implementation of this technique can be found in Section 6. Ambient occlusion is mostly used to enhance the illumination model to include soft shadows where the intensity of ambient lighting is reduced. Furthermore, with the right choice of parameters ambient occlusion can also encode the density of a flow field. In visualization this can be used for peeling.

5 Curve-based Flow Features

The previous section described how to compute curve properties. In this section we elaborate on the use of these properties to find and extract flow features.

Most visualizations of integral lines have access to the underlying flow field and can exactly determine properties like curvature and torsion. This is especially helpful for extraction of e. g. vortex core lines. Instead, our requirements stated that we do not rely on the underlying flow structures, but we only rely on the line data as input. Hence, we have to find an adequate mapping of flow field features to curve features. Appropriate filtering of the curve data will then provide us with a feature-based visualization.

5.1 Vortex Core Lines

There have been attempts to formalize the description of vortex core lines [16, 19]. Still, there is no agreement in the community on this. Instead, we use the intuition of Jiang et al. [4]: A vortex is characterized by a central core and swirling streamlines surrounding it.

We do not have the information about neighboring lines. But, what we can conclude from this is that the vortex core line has to have a high torsion because of continuity in the flow field and the swirling streamlines surrounding it. Still, it is not possible to extract every vortex core line since for a straight line the torsion computed according to equation 4 is always zero.

Nevertheless, we were able to reliably identify lines in the neighborhood of a vortex core. Our assumption for this is that these lines have high torsion and low curvature at the same time. This assumption works in a lot of cases, but does not guarantee to find all vortices.

5.2 Turbulence

As for vortex core lines it is hard to describe turbulences by a formal definition based on curve properties. Instead, we will again use intuition.

In a turbulent region of a flow streamlines have high rotational components. This is mapped to a high curvature in these regions. In addition, turbulences are small scale feature and particles have a low velocity magnitude. Hence, our adaptive streamline integration used for generation of the test data results in shorter streamlines than in other areas. Combining these two properties, high curvature and short total length, helps finding turbulences in the flow.

6 Ambient Occlusion

The main idea behind using ambient occlusion for line visualization is to add further spatial information through additional shading. Especially for dense line data sets these additional spatial cues will significantly enhance depth perception of the scene.

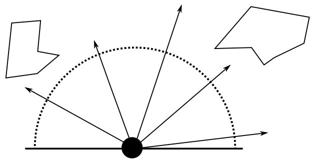


Figure 2 Tracing rays on the hemisphere of a surface for visibility calculations.

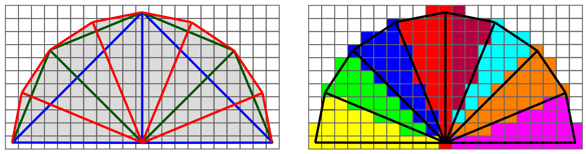


Figure 3 Voxelization for ambient occlusion. (a) Two subdivision steps in 2D for one half-circle resulting in eight bins. Blue: before subdivision, green: first subdivision, red: second subdivision. (b) Putting voxels into bins. Voxels exactly on the boundary are sorted into both adjoint bins.

6.1 Challenges

Compared to existing applications of ambient occlusion the computation of ambient occlusion for lines introduces new challenges: one-dimensional geometry, no geometry normals, SSAO is not suitable, high geometric complexity, and unclear contribution to occlusion. In the following we discuss these challenges and why they pose a problem in further detail.

The common approach to compute ambient occlusion for a specific point in the scene is to use ray casting. On any hardware we can only trace a finite number of rays. Since lines have a one-dimensional geometry for most lines the computer cannot detect a collision of the ray and a line. We will see that this problem is solved by seamlessly partitioning a sphere around a point into so-called *ray bins*. Because of this seamless partition each line that has an influence on the ambient occlusion will be put into a least one ray bin.

As we describe in the background section the original idea behind ambient occlusion assumes a surface normal for the computation. Only objects on a hemisphere in the direction of the normal can have an influence on occlusion. However, one-dimensional objects do not have a surface normal. From this we derive that the ambient occlusion of the line is dependent on the viewing direction and this has to be solved.

Screen Space Ambient Occlusion (SSAO) is commonly used to accelerate the computation of ambient occlusion. As the name already tells this approach works in screen space. For many usage scenarios SSAO is a good approximation for ambient occlusion. But, our dense line data results in high depth-frequencies in the projected 2D image. This is why SSAO as an approximation will yield a completely different result than a more accurate calculation of ambient occlusion.

High geometric complexity is another problem. Since this paper is specifically about the visualization of dense line data there are a lot of objects to be visualized. Hence, tracing rays in such a scene has high computational costs. We use a common approach of voxelization of the scene to speed up computation. However, this raises the question how much a line contributes to occlusion and how to handle transparency.

The contribution of a line to occlusion is unclear due to its one-dimensionality. Intuitively a thin line inside a voxel should not set the voxel's occlusion value to one. Our approach attacks this problem and suggests a solution that we show works well with our visualization.

6.2 General Background

The general idea of ambient occlusion is to map the visibility of an object to soft shadows. Together with common shading and lighting formulas such as the Phong illumination model this provides a more realistic and intuitive visualization.

For the computation of ambient occlusion for a certain point on a surface we trace rays in every direction of the hemisphere pointing away from the surface (see Figure 2). Then the occlusion is obtained by integrating the visibility for each ray. The general equation for this reads:

$$AO = \frac{1}{\pi} \int_{\Omega} V_{\omega}(n \cdot \omega) d\omega \tag{5}$$

where Ω is the hemisphere, ω is the direction of the ray, and $V_{\omega}(\cdot)$ the binary visibility function for this ray.

Actual implementations usually perform an approximation of the integral described in equation (5). A very common approach is using Monte Carlo Integration as a numerical method (e. g. see [18]). Another simplification is sampling the geometry on a voxel grid. This still yields good results for small voxels, but usually is faster.

6.3 Ambient Occlusion for Lines

The previous paragraph described how ambient occlusion is implemented for general geometry. As we mentioned before this can only be a basis for occlusion computations with lines. For our adaption of ambient occlusion we use a voxel based method. Instead of tracing rays, we sample hemispheres in voxel space. In the following we explain the details of our approach.

First, we give an overview over all steps needed for the computation of ambient occlusion implemented by our method:

1. Rasterize lines into voxels and count the number of lines per voxel
2. Create a subdivision of a hemisphere into ray bins for each axis
3. Create a voxel stencil for each hemisphere
4. Sort voxels of each ray bin according to the distance to the center of the hemisphere
5. Compute occlusion for each voxel
6. Store the occlusion values for each vertex of a curve

6.3.1 Rasterization

As we mentioned before we use a voxel based approach for the computation of ambient occlusion. In general we use a $128 \times 128 \times 128$ voxel grid on a unit cube – a justification for this can be found in the evaluation section. The line data is then scaled to fit the volume. As most simulations are run on boxes that are not cubes in most case we have a lot of empty space.

In this first step we take the segments of our lines and rasterize them using the 3D Bresenham line algorithm. For each traversed voxel a counter is incremented. After that the line count per voxel is scaled down to the range $[0, 1]$. This results in a discretized three-dimensional line density map. It is important that the values are normalized because in a later step we use them for our own adaption of occlusion blending.

6.3.2 Subdivision

As we already discussed it is problematic to find collisions of rays and lines. To compute the occlusion of a voxel, we have to traverse a spherical neighborhood. We therefore partition a sphere around the voxel into so-called *ray bins*. As we will see later it does not make sense to compute the overall occlusion of a point. We rather compute the occlusion for hemispheres pointing in six different directions – one along each axis and its opposite direction. Later, for visualization three of these occlusion values are blend together depending on the viewing direction.

In order to generate the ray bins we use a subdivision scheme to approximate the spherical hull by a set of tetrahedra. We start off with a pyramid divided into four tetrahedra as a rough approximation of a hemisphere. The general concept is shown in Figure 3(a) for half a circle in 2D. Each tetrahedron (or triangle in 2D) is recursively split into four new tetrahedra (two triangles). For most visualizations two of these subdivision steps already proved to be sufficient.

6.3.3 Stencil

Each of the previously generated tetrahedra corresponds to a ray bin. In this next step we create a stencil that can be applied to any voxel to find voxels belonging to its ray bins. So, we take the relative offset of voxels to the current voxel and put them into the ray bins according to their relative position (see Figure 3(b)).

It is only feasible to use such a stencil if its size is significantly smaller than that of the entire voxel grid. The physical explanation for this is that the influence of occlusion is attenuated over distance. In our implementation we use the formula

$$w(d,r) = \begin{cases} \left(1 - \frac{d}{r}\right)^4 \left(\frac{4d}{r} + 1\right) & 0 \leq d \leq r \\ 0 & d > r \end{cases} \tag{6}$$

which has been successfully used in ambient occlusion [18]. The attenuation function $w(d,r)$ reaches zero at a previously defined maximum distance r. An evaluation of different radii r can be found in Section 8.1. Generally, voxels outside of this radius have a weight of zero and hence do not have to be included into the stencil.

6.3.4 Sorting

In this step all voxels of a ray bin are sorted into a single list with increasing distance from the center of the hemisphere. This sorting is needed for efficient blending of occlusion values. This step still operates on the stencil, thus the sorting has to be done only once.

6.3.5 Occlusion Computation

The previous steps, i. e. sphere subdivision, stencil generation, and voxel sorting for a hemisphere, is a one-time process and independent from concrete properties of a data set. Thus, this information can be stored to disc and loaded on demand for ambient occlusion computation. Nevertheless, setup time in the tables 3 and 2 show that this is not necessary.

Now, the setup for the actual occlusion computation is finished. In the actual occlusion computation pre-computed stencil information is applied to a given data set. At first, the occlusion calculation is done for each ray bin separately. Voxels are successively taken from the sorted list of the stencil. Their occlusion values are blended together from front to back just like alpha blending is used to emulate transparency. Additionally, the occlusion values are weighted according to equation 6.

For the final occlusion value of a voxel in one of the six directions we use the average of the occlusion values of all the ray bins of the corresponding hemisphere. To speed up the computation ray bins are first combined into octants of the sphere. Then, four octants are combined into the actual hemisphere to contain the occlusion value.

Finally, for faster access we store the six occlusion values for each vertex of a curve. For this, we just have to find the corresponding voxel for a vertex and transfer its occlusion values.

Our actual implementation of this method computes the occlusion for a voxel at most once and only if there is a least one vertex inside the voxel. Furthermore, each voxel has a list of all vertices that are located inside the voxel. Then, we get an additional speed up because we can just transfer the occlusion values from the voxel to all corresponding vertices.

7 Visualization

The main purpose of this paper is to find a method which provides a feasible visualization option for dealing with dense line data. The obvious solution to use transparency which allows us to show previously hidden lines. With the help of transfer functions based on curve features we can even extract flow features. However, as we discussed in the motivation, Section 3, transparency introduces new problems. Hence, we combine existing methods like illuminated lines and halos and add ambient occlusion for lines to address these problems.

In the following we discuss the basics that are needed for expedient rendering of integral lines. Since the common method to emulate transparency on the GPU is alpha blending we need depth sorting of the line segment. This computation is quite slow for the high number of lines we are using. And a pre-computation for every possible viewing direction is not feasible.

Instead, line segments are pre-sorted along the coordinate axes. During visualization the sorting that corresponds most to the current viewing direction is used. According to [30] this induces an error for at most 1% of the rendered pixels. Looking at our pictures rendering artifacts are very rare.

Many existing methods use a tube-like geometry for rending streamlines (for example see [10]). In our implementation we use OpenGL's line primitive to draw the line segments. The line width is usually given in screen space and does not scale with depth by itself. Thus, for some renderings we use the depth information to adjust the line width per segment. This approach allows for a more natural depth perception while reducing the amount of vertex information compared to rendering a more complex geometry like tubes.

Halos are most commonly used to help the viewer with depth perception. However, we figured out that combining halos with transparency destroys this perception. Nevertheless, halos are useful to visually separate lines from each other. Our implementation draws the original lines first and in a second run draws halos as a thicker line using the background's color. Using the depth buffer of the graphics card the halo line is only drawn along the sides of the actual line segment. The halo width is set as ratio to the original line's thickness. This combines well with the depth dependent scaling of the line width if needed.

7.1 Lighting

Since the first rendering of 3D streamlines [30] the Phong illumination model has been used to support perception of the line's shape. In order for the Phong illumination to work we need a surface normal that is not available for a one-dimensional geometry like a line. Hence, the normal is calculated such that it is in one plane with the tangent t_i at the position of the vertex v_i and the light vector L_i. The corresponding formula reads:

$$n_i = \frac{(L_i \times t_i) \times t_i}{\|(L_i \times t_i) \times t_i\|}. \tag{7}$$

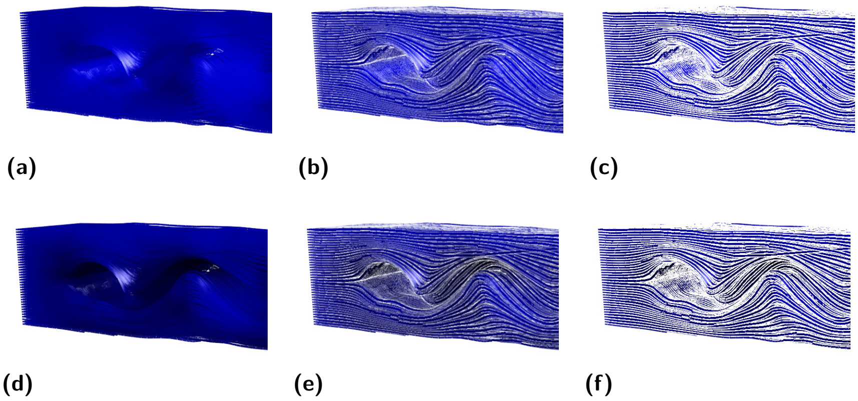


Figure 4 Flow around a cylinder. Left: no halos. Middle: transparent halos. Right: opaque halos. Top row: no ambient occlusion. Bottom row: ambient occlusion mapped to intensity.

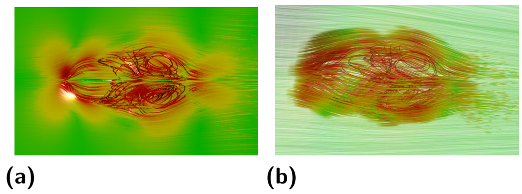


Figure 5 Flow around an ellipsoid. (a) Focusing on a region of interest by cutting off lines in front. (b) Selecting lines with higher curvature through transparency. Coloring shows the magnitude of curvature per segment.

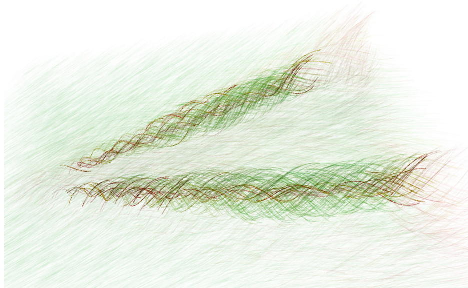


Figure 6 Extraction of vortex cores for the delta wing data set [26]. Transparency selects regions with high torsion and low curvature. Red streamlines on the center of the vortex have a high torsion, green streamlines have a low torsion and are only provided for context.

With this information we compute the illumination of each line segment.

Ambient occlusion is mostly implemented to assist with illumination. It feels natural to the human eye that objects that are partially occluded have soft shadows which means that they are darker. So, the occlusion value is used to reduce the intensity of a line segment, i. e. the influence of ambient lighting is reduced for hidden lines. The result feels more natural and enhances depth perception.

7.2 Transfer Functions and Color Maps

Transfer functions, both on transparency and color, are heavily used in our visualizations. Simply applying the same transparency to all lines reveals more lines, but does not reduce visual clutter. Hence, we are using transfer functions on line properties like e. g. curvature and torsion to focus on important flow features. There is no automated process for this. But instead, there are some general rules as described in Section 5 which combined with basic expert knowledge yield good results.

For illustrative purposes or deeper insight into the data transfer functions can also be applied for colors. Coloring is often used for visualization of streamlines. It shows how flow

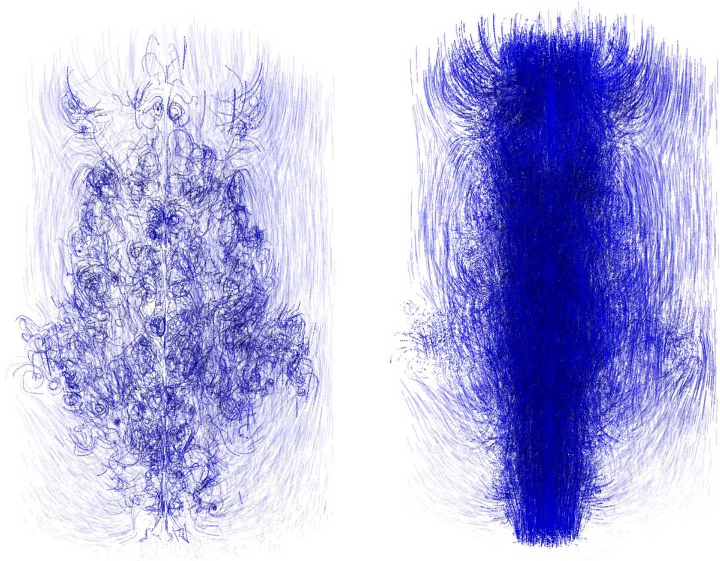


Figure 7 Extraction of turbulences in the jet stream data set. Streamlines are selected by high curvature and short total length. The right image shows the inverse selection.

properties change inside the simulation domain. With the dense line data that we have it is only useful in combination with transparency highlighting important regions.

8 Results

In a first part, we shortly discuss results using ambient occlusion for line rendering. Then, we focus in depth on the use of transfer functions on transparency to highlight important flow features. In a separate subsection we evaluate different parameter settings for the calculation of ambient occlusion.

Figure 4 shows differences for halos and ambient occlusion used for highlighting spatial relationships and separation of lines. Within the dense data set it is not possible for the viewer to clearly separate lines by just using illuminated lines (see Figure 4a). Ambient occlusion clearly improves depth perception in all cases. But without halos it only weakly separates lines in the foreground from those in the background (see Figure 4d). Opaque halos clearly give a nice illustrative visualization (Figure 4f) that can be further improved with ambient occlusion (Figure 4c). Still the insight gained from this visualization is questionable. In general we conclude that we can use intensity mapping based on ambient occlusion for improved depth perception, and transparent halos for separation of lines.

In the following, we show examples of how transfer functions on curve properties can be used to extract or highlight important flow features. We start with a generally known example.

Figure 5a shows the flow around an ellipsoid by cutting off lines in front of the region of interest. The problem here is that in order to get close to the interesting flow features we also cut off streamlines with important flow features. This focusing technique can be improved by using different curve properties. In Figure 5b we use transparency transfer functions on curvature that only select streamline segments that have a high curvature. This removes all straight lines that occluded the view before.

Another well known example for streamline visualization is the delta wing data set. Here, we extract streamlines near the vortex cores according to our description in Section 5.1. In Figure 6 we see that the extraction works well showing streamlines near the center of vortices in red. Streamlines with lower torsion are colored in green. Rendering them with higher transparency gives some context for the overall data set.

In a last example, we show the extraction of turbulent regions in a jet stream data set. As for the vortex core lines we apply the description from Section 5.2. This means that we can find turbulent streamlines by looking for high curvature and a short total length of the streamline.

8.1 Evaluation of Ambient Occlusion Computations

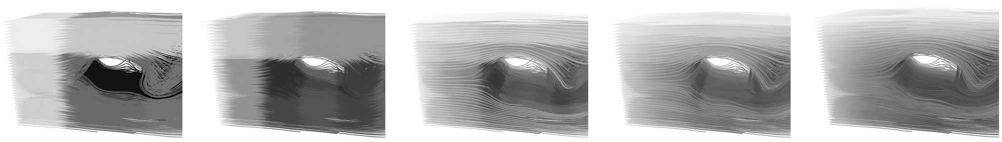
In this section we evaluate the choice of different parameters for ambient occlusion. There are three parameters that can be adjusted: the number of voxels, the maximum distance of the sphere around a voxel, and the number of subdivision steps used as approximation of the sphere. The last parameter directly influences the number of ray bins – two subdivision steps already yield 64 ray bins per hemisphere.

Although the computation of the ambient occlusion is done as a pre-processing step prior to the visualization the right choice of parameters is a trade-off between computation time – which is $\mathcal{O}(n^3)$ – and visual quality. The latter one in many cases is specific to ones visual preferences.

In a first experiment we adjusted the number of voxels used for rasterization. At the same time we linearly scaled the maximum radius of the sphere, given as number of voxels, such that the same lines have an impact on occlusion. This is the only way to make a visual comparison of the results. Our benchmark machine is a notebook with an Intel Pentium T2330 Dual Core CPU running at 1.6 GHz. Our algorithm is completely parallelized for the actual calculation of the ambient occlusion and entirely uses up the two cores. Table 1 shows the runtimes for different parameters averaged over five runs and Figure 8 shows the corresponding visualization for comparison.

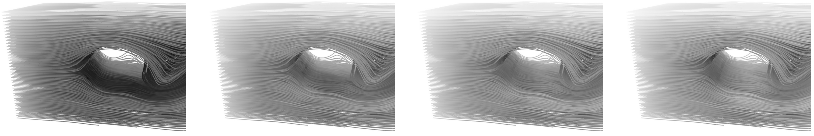
For a second test we varied just the maximum distance. Low values are especially interesting when using ambient occlusion as some measurement for density. Then, the influence on lines to occlusion is restricted to the immediate neighborhood. In visualization the interpretation as density can be used for some sort of peeling. From the results in Table 3 we can derive that the maximum distance already has a high impact on runtimes.

Finally, we tested the influence of subdivision steps of the hemisphere. From the runtime results in Table 2 we see that number of subdivisions has only a minor impact on the overall runtime. The reason for this is that we have the same amount of voxels that have an influence on occlusion regardless of the subdivision. There is only a slight overhead for descending to the next subdivision and additional recursive function calls.



(a) $8 \times 2 \times 2$ **(b)** $16 \times 4 \times 2$ **(c)** $32 \times 8 \times 2$ **(d)** $64 \times 16 \times 2$ **(e)** $128 \times 32 \times 2$

Figure 8 Grayscale maps of occlusion values. The number of voxels used for AO computation increases from left to right (compare Table 1). Low resolutions show voxelization artifacts, but 128 voxels already provides smooth results.



(a) $128 \times 20 \times 0$ **(b)** $128 \times 20 \times 1$ **(c)** $128 \times 20 \times 2$ **(d)** $128 \times 20 \times 3$

Figure 9 Grayscale maps of occlusion values for increasing subdivision steps. There are only minor differences for one and two subdivision steps, but no visible changes for two and three subdivision steps.

The images in Figure 9 show that no or only one subdivision step do not give satisfactory visual results. For a lot of data sets two subdivisions are sufficient and only for very dense data sets there is a visible difference compared to three subdivision steps. From these three tests we derived that the parameter setting of $128 \times 20 \times 2$ is a good trade-off between computation times and visual quality for most cases. Our images including a mapping of ambient occlusion to intensity enhancing the lighting model were generated with these settings.

9 Conclusions and Future Work

In the future there is clearly a need for good visualizations of flow simulations. Integral lines build a good basis for this. In contrast to previous methods we do not rely on the underlying flow field. Instead, we solely use integral lines that can be generated as output by most simulation software. Another difference is that we require a dense sampling of integral lines: without the flow field it is impossible to add information in the visualization step if you have too few lines. On the other hand, if there are too many lines, we provide a solution based on transfer functions to reduce visual clutter and highlight important flow features. To sum it up, this paper has two major contributions. Our results show that our approaches yield insightful pictures.

First, our approach improves rendering of lines, especially for dense line data. For the rendering part we have two major contributions. For one, transparency can be used while maintaining separation of lines and their spatial relationships. Transparency is really helpful for focusing on important flow features. Then, we contribute to ambient occlusion computation by extending existing methods to include AO for lines. With the combination of this and existing techniques – sometimes combining only some of them – we can provide better visualizations than before.

Second, we provide a new method to find important flow features. For this, we do not

Table 1 Runtimes for different voxel grid resolution. The increasing maximum distance has a high impact on increasing computation times.

$n_{voxels} \times max_{dist} \times n_{subdiv}$	runtime
$8 \times 2 \times 2$	0.14 s
$16 \times 4 \times 2$	0.16 s
$32 \times 8 \times 2$	0.88 s
$64 \times 16 \times 2$	27.93 s
$128 \times 32 \times 2$	1313.24 s

Table 2 Runtimes for different numbers of subdivisions of the hemisphere.

$n_{voxels} \times max_{dist} \times n_{subdiv}$	runtime	setup time
$128 \times 20 \times 0$	374.27 s	0.88 s
$128 \times 20 \times 1$	420.63 s	0.89 s
$128 \times 20 \times 2$	490.62 s	0.90 s
$128 \times 20 \times 3$	530.42 s	0.88 s

Table 3 Runtimes for increasing maximum distance. Additionally, we include the setup time for the stencil and rasterization, i. e. steps 1-4 of our method. The setup is only single threaded and the setup times are already included in the overall runtimes.

$n_{voxels} \times max_{dist} \times n_{subdiv}$	runtime	setup time
$128 \times 1 \times 2$	1.47 s	0.66 s
$128 \times 2 \times 2$	2.55 s	0.66 s
$128 \times 4 \times 2$	8.76 s	0.66 s
$128 \times 8 \times 2$	46.22 s	0.68 s
$128 \times 16 \times 2$	282.06 s	0.78 s
$128 \times 32 \times 2$	1314.10 s	1.72 s

rely on the underlying flow field information. This makes it easier to integrate with existing simulation software. Instead, we use only integral lines for visualization. In order to have all the information that is needed we require a dense sampling of integral lines. For the flow expert we provide the steps that are necessary to find vortices and turbulent regions.

Still, there is a lot of improvement for future research. For one, ambient occlusion can be used with a small surrounding sphere to compute the local density. These regions might incorporate interesting flow features as well. For example, regions with a high density can point to sources, sinks, or separation lines.

Furthermore, in this paper we just used properties on single curves. Combining curve properties with neighboring integral lines can give even more insight into the data. Especially, the definition of vortices is easier if the neighborhood is included. Then, it might also be possible to detect vortex cores that are straight lines and hence do not have a torsion according to our calculation from Section 4.

References

1. Bernardo Silva Carmo, Y H Pauline Ng, Adam Prügel-Bennett, and Guang-Zhong Yang. A data clustering and streamline reduction method for 3d mr flow vector field simplification. *Lecture Notes in Computer Science*, 3216:451–458, 2004.
2. Frank Enders, Natascha Sauber, Dorit Merhof, Peter Hastreiter, Christopher Nimsky, and Marc Stamminger Stamminger. Visualization of white matter tracts with wrapped streamlines. *Visualization Conference, IEEE*, 0:51–58, 2005.
3. Maarten H. Everts, Henk Bekker, Jos B.T.M. Roerdink, and Tobias Isenberg. Depth-dependent halos: Illustrative rendering of dense line data. *IEEE Transactions on Visualization and Computer Graphics*, 15:1299–1306, 2009.
4. Ming Jiang, Raghu Machiraju, and David Thompson. A novel approach to vortex core region detection. In *Proceedings of the Symposium on Data Visualisation 2002*, VISSYM '02, pages 217–ff, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.
5. Ming Jiang, Raghu Machiraju, and David Thompson. Detection and visualization of vortices. In *The Visualization Handbook*, pages 295–309. Academic Press, 2005.
6. Bruno Jobard and Wilfrid Lefer. Creating evenly-spaced streamlines of arbitrary density. In *Proceedings of the 8^{th} Eurographics Workshop on Visualization in Scientific Computing*, pages 45–55, 1997.
7. Tae-Yong Kim and Ulrich Neumann. Opacity shadow maps. In *In Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 177–182. Springer-Verlag, 2001.

8 Liya Li, Hsien-His Hsieh, and Han-Wei Shen. Illustrative streamline placement and visualization. In *PacificVis '08: Proceedings of the IEEE Pacific Visualization Symposium 2008*, pages 79–86, 2008.

9 Zhanping Liu, Robert Moorhead, and Joe Groner. An advanced evenly-spaced streamline placement algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 12:965–972, September 2006.

10 Kwan-Liu Ma, Greg Schussman, Brett Wilson, Kwok Ko, Ji Qiang, and Robert Ryne. Advanced visualization technology for terascale particle accelerator simulations. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, Supercomputing '02, pages 1–11, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

11 Ovidio Mallo, Ronald Peikert, Christian Sigg, and Filip Sadlo. Illuminated Lines Revisited. In *IEEE Visualization*, pages 19–26, 2005.

12 Stephane Marchesin, Cheng-Kai Chen, Chris Ho, and Kwan-Liu Ma. View-dependent streamlines for 3d vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 16:1578–1586, November 2010.

13 Oliver Mattausch, Thomas Theußl, Helwig Hauser, and Eduard Gröller. Strategies for interactive exploration of 3d flow using evenly-spaced illuminated streamlines. In *Proceedings of the 19th spring conference on Computer graphics*, SCCG '03, pages 213–222, New York, NY, USA, 2003. ACM.

14 Morgan McGuire. Ambient occlusion volumes. In *Proceedings of High Performance Graphics 2010*, pages 47–56, June 2010.

15 Ron Otten, Anna Vilanova, and Huub van de Wetering. Illustrative white matter fiber bundles. *Comput. Graph. Forum*, 29(3):1013–1022, 2010.

16 Luis M. Portela. *Identification and Characterization of Vortices in the Turbulent Boundary Layer*. PhD thesis, Stanford University, December 1998.

17 Frits H. Post, Benjamin Vrolijk, Helwig Hauser, Robert S. Laramee, and Helmut Doleisch. Feature extraction and visualisation of flow fields. In *In Eurographics 2002 State-of-the-Art Reports*, pages 69–100, 2002.

18 Christoph Reinbothe, Tamy Boubekeur, and Marc Alexa. Hybrid ambient occlusion. *EUROGRAPHICS 2009 Areas Papers*, 2009.

19 Stephen K. Robinson. Coherent motions in the turbulent boundary layer. *Annual Review of Fluid Mechanics*, 23(1):601–639, 1991.

20 I. Ari Sadarjoen and Frits H. Post. Detection, quantification, and tracking of vortices using streamline geometry. *Computers & Graphics*, 24(3):333–341, 2000.

21 Tobias Salzbrunn and Gerik Scheuermann. Streamline predicates. *IEEE Transactions on Visualization and Computer Graphics*, 12:1601–1612, November 2006.

22 Greg Schussman and Kwan-Liu Ma. Scalable self-orienting surfaces: A compact, texture-enhanced representation for interactive visualization of 3d vector fields. In *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, PG '02, pages 356–365, Washington, DC, USA, 2002. IEEE Computer Society.

23 Greg Schussman and Kwan-Liu Ma. Anisotropic volume rendering for extremely dense, thin line data. In *Proceedings of the conference on Visualization '04*, VIS '04, pages 107–114, Washington, DC, USA, 2004. IEEE Computer Society.

24 Kuangyu Shi, Holger Theisel, Helwig Hauser, Tino Weinkauf, Kresimir Matkovic, Hans-Christian Hege, and Hans-Peter Seidel. Path line attributes – an information visualization approach to analyzing the dynamic behavior of 3d time-dependent flow fields. In *Topology-Based Methods in Visualization II*, pages 75–88, 2009.

25 John C. Strikwerda. *Finite difference schemes and partial differential equations*. Wadsworth Publ. Co., Belmont, CA, USA, 1989.

26 X. Tricoche, C. Garth, T. Bobach, G. Scheuermann, and M. Rütten. Accurate and efficient visualization of flow structures in a delta wing simulation. In *Proceedings of 34th AIAA Fluid Dynamics Conference and Exhibit*, number AIAA Paper 2004-2153, June 2004.

27 Vivek Verma, David Kao, and Alex Pang. A flow-guided streamline seeding strategy. In *In Proceedings IEEE Visualization 2000*, pages 163–170.

28 Xiaohong Ye, David Kao, and Alex Pang. Strategy for seeding 3d streamlines. *Visualization Conference, IEEE*, 0:471–476, 2005.

29 Cem Yuksel and John Keyser. Deep opacity maps. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2008)*, 27(2):675–680, 2008.

30 Malte Zöckler, Detlev Stalling, and Hans-Christian Hege. Interactive visualization of 3d-vector fields using illuminated stream lines. In *IN PROCEEDINGS OF IEEE VISUALIZATION '96*, pages 107–113, 1996.