ROBERT AUTENRIETH

ILLUSTRATIVE VISIBILITY ENHANCEMENT OF FLOW FEATURES

ETH Diss No. 23244

ILLUSTRATIVE VISIBILITY ENHANCEMENT OF FLOW FEATURES

A thesis submitted to attain the degree of
DOCTOR OF SCIENCES OF ETH ZURICH
(Dr. sc. ETH Zurich)

presented by
ROBERT AUTENRIETH
MSc Computational Science and Engineering,
ETH Zurich

born on July 1st, 1982
citizen of Dietikon, Switzerland

accepted on the recommendation of

Prof. Dr. Ronald Peikert, examiner
Prof. Dr. Petros Koumoutsakos, co-examiner
Prof. Dr. Helwig Hauser, co-examiner

2015

ABSTRACT

The use of computers and digital technology has allowed researchers to study our environment through simulations or measurements of ever-increasing complexity. However, the resulting amount of raw data is difficult to understand. The focus of the field of visualization is therefore to take this abstract data and present it in a form that conveys important information and offers insight about the underlying processes to human observers. In a recent trend, visualization methods have been using techniques inspired by hand-drawn illustrations, emphasizing the use of abstraction and non-photorealistic depiction.

One area of application for visualization is fluid dynamics. When dealing with the time-dependent motion of a fluid through a volume, it is impossible to display all of the data at once. Flow visualization methods therefore often extract interesting surfaces (e.g., isosurfaces or integral surfaces) and display only those. However, such surfaces can still form folds and twists or suffer from occlusion and cluttering, which makes them difficult to understand.

In this thesis, we present several techniques that address this issue by using illustrative visualization. In a first part, we focus on higher-level abstractions. We study the use of the proper orthogonal decomposition (POD) for the visualization of noisy 4D magnetic resonance imaging (MRI) blood flow data. By discarding high-frequency scales, we seek to improve the visualization of large-scale vortices. Furthermore, we propose a method for automatically placing cutaway primitives for the visualization of arbitrary three-dimensional data suffering form occlusion. The method uses a Monte Carlo algorithm, as the optimal placement of such primitives is suspected to be infeasible.

In a second part, we focus on lower-level improvements of the perception of flow surfaces. We present a 2.5D screen space data stucture and a framework that can be used for illustrative enhancements of surfaces forming multiple layers due to occlusion. Using this framework, we propose a method for improving the shape perception of multi-layered transparent surfaces, inspired by findings from perception theory. Finally, we study the use of dense flow visualization to additionally highlight the flow direction along multi-layered surfaces.

ZUSAMMENFASSUNG

Die fortschreitende Entwicklung von Rechnern und digitaler Technik erlaubt es vielen Forschern unsere natürliche Umgebung durch immer komplexer werdende Simulationen und Messungen zu studieren. Diese liefern allerdings eine unüberschaubare Menge an Daten und sind schwer zu interpretieren. Das Forschungsfeld der Visualisierung hat sich deshalb zur Aufgabe gemacht, solche abstrakten Daten zu nehmen und diese in einer Weise zu präsentieren, welche wichtige Informationen und Erkentnisse über die zugrundeliegenden Prozesse an den menschlichen Betrachter liefert. Inspiriert von handgezeichneten Illustrationen haben viele Methoden in letzter Zeit die starke Abstraktion und eine nicht-fotorealistische Darstellung der Daten hervorgehoben und haben damit das Gebiet der illustrativen Visualisierung gegründet.

Visualisierung findet unter anderem in der Fluiddynamik eine Anwendung. Da man alle Daten einer zeitabhängigen, dreidimensionalen Strömung nicht direkt darstellen kann, werden daraus oft nur interessante Stromflächen extrahiert und angezeigt. Besonders bei turbulenten Strömungen sind diese aber so kompliziert und verdreht, so dass sie immer noch schwer zu verstehen sind.

In dieser Arbeit stellen wir deshalb einige neue illustrative Visualisierungsmethoden vor, um dieses Problem anzugehen. Der erste Teil der Arbeit konzentriert sich auf Abstraktionen auf hoher Ebene. Dabei wird erst die POD für die Visualisierung von verrauschten 4D MRI Blutströmungsdaten angewandt, wobei durch Unterdrückung der hochfrequentigen Skalen die Darstellung von Wirbeln verbessert wird. Danach stellen wir eine auf Monte Carlo basierte Methode vor, welche automatisch Cutaway-Primitiven plaziert um verdeckte Daten besser darzustellen.

Im zweiten Teil beschreiben wir eine neuartige 2.5D Datenstruktur und ein dazugehörendes System, mit welchem sich Flächen mit mehreren überlappenden Schichten illustrativ darstellen lassen. Basierend auf diesem System stellen wir eine Methode vor, mit welcher sich die Formwahrnehmung von Flächen mit mehreren Schichten verbessern lässt. Schliesslich untersuchen wir noch die Verwendung von globaler Visualisierung, um zusätzlich die Strömungsrichtung auf mehrschichtigen Flächen darzustellen.

PUBLICATIONS

This thesis is based on the following publications:

[1] Stephan Sigg, Raphael Fuchs, **Robert Carnecky**, and Ronald Peikert. "Intelligent Cutaway Illustrations". In: *Proceedings of IEEE Pacific Visualization Symposium*. 2012, pp. 185–192.

[2] **Robert Carnecky**, Benjamin Schindler, Raphael Fuchs, and Ronald Peikert. "Multi-layer Illustrative Dense Flow Visualization". In: *Computer Graphics Forum* 31.3 (2012), pp. 895–904.

[3] **Robert Carnecky**, Raphael Fuchs, Stephanie Mehl, Yun Jang, and Ronald Peikert. "Smart transparency for illustrative visualization of complex flow surfaces". In: *IEEE Trans. Vis. Comput. Graph.* 19.5 (2013), pp. 838–851.

[4] **Robert Carnecky**, Thomas Brunner, Silvia Born, Jürgen Waser, Christian Heine, and Ronald Peikert. "Vortex Detection in 4D MRI Data: Using the Proper Orthogonal Decomposition for Improved Noise-Robustness". In: *EuroVis – Short Papers*. 2014, pp. 127–131.

During my PhD program at ETH Zurich, I have also contributed to several other publications:

[5] Andrea Brambilla, **Robert Carnecky**, Ronald Peikert, Ivan Viola, and Helwig Hauser. "Illustrative Flow Visualization: State of the Art, Trends and Challenges". In: *EG 2012 - State of the Art Reports*. Cagliari, Sardinia, Italy: Eurographics Association, 2012, pp. 75–94 (cit. on p. 9).

[6] Raphael Fuchs, Benjamin Schindler, **Robert Carnecky**, Jürgen Waser, Yun Jang, and Ronald Peikert. "Adaptive Treelet Meshes for Efficient Streak-Surface Visualization on the GPU". In: *VMV 2012: Vision, Modeling & Visualization*. 2012, pp. 119–126.

[7] Ronald Peikert, Benjamin Schindler, and **Robert Carnecky**. "Ridge Surface Methods for the Visualization of Lagrangian Coherent Structures". In: *Proc. Ninth International Conference on Flow Dynamics, Sendai, Japan*. 2012, pp. 206–207.

[8] Bernhard Sadransky, Hrvoje Ribičić, **Robert Carnecky**, and Jürgen Waser. "Visdom Mobile: Decision Support On-site Using Visual Simulation Control". In: *Spring Conference on Computer Graphics*. SCCG '13. 2013, pp. 99–106.

[9] Benjamin Schindler, Raphael Fuchs, Stephan Barp, Jürgen Waser, Armin Pobitzer, **Robert Carnecky**, Krešimir Matković, and Ronald Peikert. "Lagrangian Coherent Structures for Design Analysis of Revolving Doors". In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2159–2168 (cit. on p. 5).

[10] Jürgen Waser, Artem Konev, Bernhard Sadransky, Zsolt Horváth, Hrvoje Ribičić, **Robert Carnecky**, Patrick Kluding, and Benjamin Schindler. "Many Plans: Multidimensional Ensembles for Visual Decision Support in Flood Management". In: *Computer Graphics Forum* 33.3 (2014), pp. 281–290.

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor Ronny Peikert, for his constant support, openness and patience. I am also grateful to Petros Koumoutsakos and Helwig Hauser for agreeing to be my co-examinators, and Helwig Hauser inviting me to Bergen.

My colleagues at the SciVis group have contributed immensely to my personal and professional time at the ETH. I would like to thank Raphael Fuchs, Benjamin Schindler, Christian Heine, Yun Jang, Thomas Brunner, and Stephan Sigg for the many fruitful discussions and making my time at the ETH a very positive experience.

Furthermore, I would like to thank Jürgen Waser and Hrvoje Ribičić from the Visdom [112] team for our close collaboration on the framework which accompanied me throughout my PhD studies, and Silvia Born, Armin Pobitzer, and Ivan Viola for their input and support.

Last but not least, I would like to thank my family, my parents Robert and Lida, and my brother Tom, for always being there for me, and my wife Chrissy for her unconditional love and encouragement.

This work was supported by ETH Independent Investigators' Research Awards (ETHIIRA) grant 12 09-3 Swiss National Science Foundation (SNF) grant 200021_127022, as well as Future and Emerging Technologies (FET) programme grant 226042

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

CFD	Computational fluid dynamics
CT	Computed tomography
DOI	Degree of interest
LIC	Line integral convolution
MC	Monte carlo
MRI	Magnetic resonance imaging
POD	Proper orthogonal decomposition
SA	Simulated annealing
SNR	Signal-to-noise ratio

INTRODUCTION

This thesis deals with the illustrative visualization of data derived from fluid dynamics simulations or measurements. The following sections give a short introduction into the relevant topics, in order to provide the necessary context.

1.1 VISUALIZATION

The term *visualization* is used in many different contexts. Although the exact meaning differs, it most often describes processes creating (mental or physical) images of some abstract input. In computer science, visualization is a process which takes data or knowledge and presents them in a form that gives insight about the data, or conveys a message contained therein.

A central aspect in visualization is the human observer – visualization uses our brain to find interesting patterns, draw conclusions, or learn information. It is therefore useful in situations where an automatic search for answers (e.g., machine learning) is not feasible. Another important aspect of visualization is the abstraction of data. A realistic depiction of the data (e.g., through a photo-realistic image) is often overwhelming and important information is lost in the detail. By omitting non-important parts and replacing complex data with simple, abstract objects, visualizations can increase the amount of information conveyed to the human observer.

The above definition of visualization is still rather broad, and there are many sub-fields and applications. Examples include visualizing the network of metro stations for easy navigation, illustrations of the human body in medical text books that teach anatomy, or 3D renderings of molecules that help predict their properties by highlighting chemically relevant regions.

1.2 ILLUSTRATIVE VISUALIZATION

Illustrative visualization is a category of approaches that use techniques from hand drawn illustrations [94]. These approaches follow the same goal as visualization in general, and the term *illustrative* is mainly used to emphasize the focus on non-photorealistic depiction and abstraction of the data, extensively used by human illustrators. In a recent work, Brambilla et al. [22] survey illustrative visualization methods in flow visualization and classify them into two major categories: *High level* visual abstractions deal with *what* to draw, and use methods such as selective visualization, cutaways, or exploded views. *Low level* visual abstractions on the other hand deal with *how* to draw, and use non-photorealistic rendering techniques that enhance shape or depth perception. We adapt this classification for the structure of this thesis, as detailed in Section 1.5.

1.3 FLUID DYNAMICS

Fluid dynamics is the scientific discipline that deals with the motion of fluids (liquids and gases). Such motion is relevant to many science and engineering applications, such as building more efficient hydroelectric plants by understanding the flow of water through its turbines, predicting weather by computing the flow of air through our atmosphere, or diagnosing cardiovascular diseases by analyzing the flow of blood through vessels. Traditionally, the motion of fluids was studied through real life experiments. While these can still be relevant in cases where properly modelling the physical process is difficult, many applications are simple enough to be accurately simulated with the use of computers. computational fluid dynamics (CFD) simulations have the advantage that they produce large amounts of data. They allow us to precisely inspect all physical values at every point in space and time – something generally not possible with measured experiments.

The amount of data produced by CFD simulations is however also a challenge. Simulations are often three-dimensional, time dependent, multi-variate (e.g., measuring velocity and pressure at the same time), and very finely resolved. They produce a huge number of plain numerical values, which cannot easily be interpreted by humans. Simulations are therefore usually followed up by a postprocessing phase, where the raw data is transformed into a form that provides insight about the underlying process.

1.4 FLOW VISUALIZATION

Flow visualization deals with the visualization of processes that define a concept of a flow. Although it can be used to visualize abstract flows, such as the "flow" of humans during an evacuation, the most common data source for flow visualization are CFD simulations. A typical example would be visualizing the simulated flow of water through a turbine in order to find undesired vortices and thereby gain ideas for improving the design of the machine. There are many different approaches to visualizing flow data. Most of the methods fall into one of the following categories [92]:

DIRECT These techniques visualize the data directly, without much preprocessing. Examples are simple color-coding or visualizing a dataset through a grid of arrows (see Figure 1a).

DENSE OR TEXTURE BASED These techniques usually start with a noise image which is then smeared along the flow. The resulting image gives a dense visualization of the flow, i.e., the user can see the flow direction at every point of the domain (see Figure 1b). A survey of related approaches is given by Laramee et al. [67].

GEOMETRIC These techniques usually start with a set of seeding points, from which particles are inserted into the flow. The particle trajectories form lines or surfaces, which are then visualized. (see Figure 1c). A survey of related approaches is given by McLoughlin et al. [82].

FEATURE BASED These techniques extract *features* from the data and visualize those. Features are interesting or physically meaningful objects, such as shock waves, vortices, vector field topology [50], or lagrangian coherent structures [46] (see Figure 1d). A survey of related approaches is given by Post et al. [92].

Most flow visualization applications deal with complex three-dimensional data, which is difficult to visualize in an image. Common problems of all approaches are therefore cluttering, occlusion, and limited depth perception. One of the most powerful tools to alleviate these problems is inter-

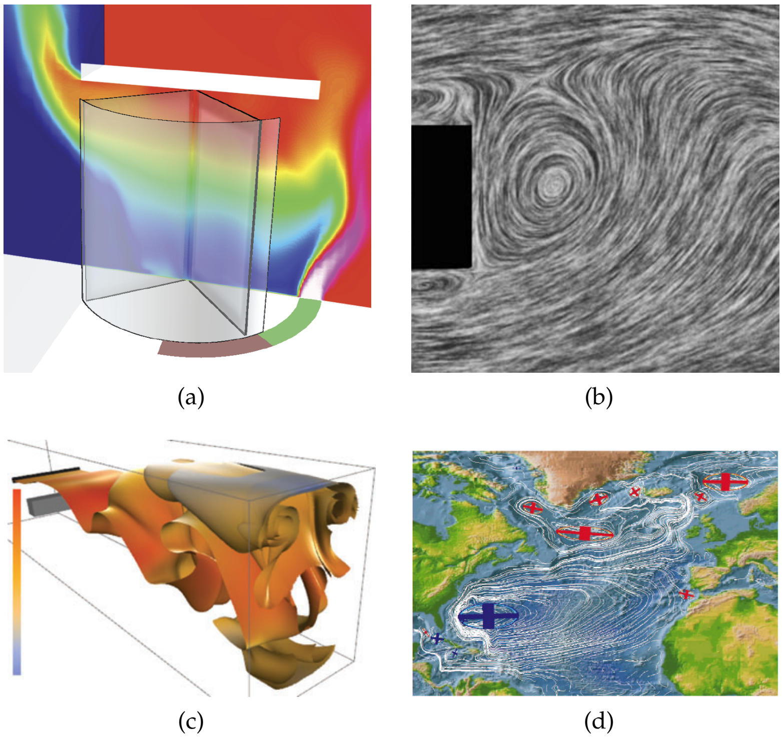


Figure 1: Flow visualization examples. (a) direct visualization [9] (b) texture based visualization [67] (c) geometry based visualization [81] (d) feature based visualization [97].

action. By manipulating the view and visualization parameters, users can explore datasets and gain information that would be impossible to get from a static image.

1.5 CONTRIBUTIONS AND LAYOUT OF THIS THESIS

In this thesis, we propose four methods that improve different aspects of illustrative visualization:

- A method for automatically placing cutaways (Chapter 3).
- A method for improving the visualizing of noisy MRI data using the proper orthogonal decomposition (Chapter 4).
- A method for improving the perception of complex, multi-layered transparent surfaces (Chapter 8).
- A method for computing the line integral convolution (LIC) on multi-layered surfaces (Chapter 9).

The thesis is organized in two parts: *high level abstraction* and *low level abstraction*.

Part I

HIGH LEVEL ABSTRACTION

In this part, we present two methods based on higher-level abstractions. These methods hide, deform, or otherwise modify features in order to improve the amount of information conveyed by the visualization.

2 OVERVIEW

High level abstractions hide or deform displayed features in order to increase the communicative intent of an illustration [5]. Figure 2 shows examples of such abstractions used in traditional hand-crafted illustrations.

One important paradigm in illustrative visualization is *focus emphasis* (or *focus and context*), which is based on the assumption that some parts of the data are more important than others. The important data is therefore highlighted, while the less important data is displayed in a simplified form to provide context [48]. The importance can be expressed using a degree of interest (DOI) function, which maps data items to a dimensionless *importance* value. There are many approaches for the speficifation of DOI functions, including explicit selection in a 3D view, analytical expressions using physical quantities, or interactive brushing of scatterplots [34].

On the other hand, the goal of *visibility management* (or *smart visibility*) techniques is to improve the visibility of the data by hiding or deforming features such that they optimally use the available space [110].

A very popular abstraction in illustrative visualization is the hiding of features in order to show occluded parts of the data. This can be achieved using cutaways and ghosting [38, 33], where surfaces are rendered transparently or virtual objects are used to clip parts of the data. These techniques can be applied interactively [41, 28] or using heuristics [32, 111]. In chapter 3 we present an approach for computing

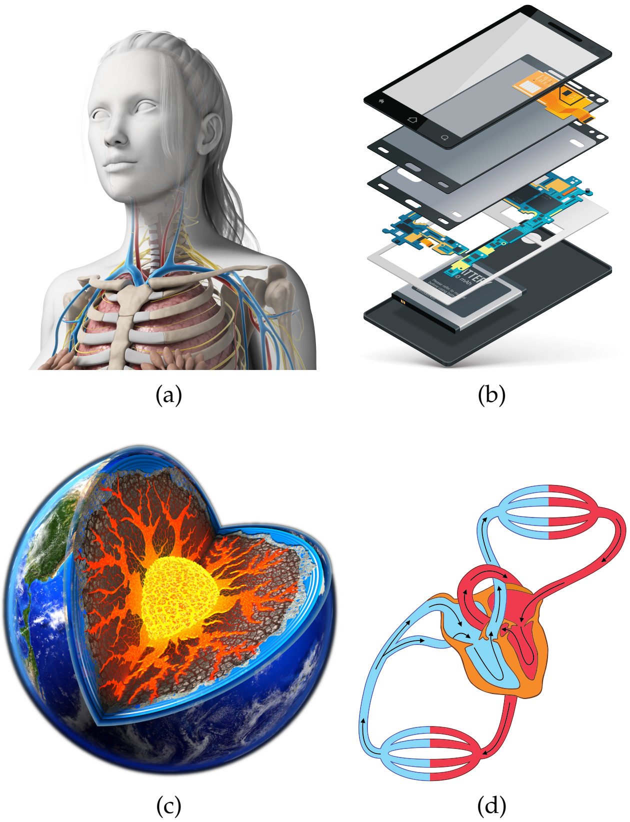


Figure 2: Examples of high level abstractions in traditional illustrations. (a) A ghosted view hiding occluding skin layers. (b) An exploded view of a smart phone. (c) Cutout view revealing the inner structure of Earth. (d) Semantic simplification of blood vessels.

the optimal positioning of clipping objects directly from a DOI function.

Apart from being hidden or removed, displayed features may also be simplified or generalized. Removing less important high frequency details may help the user concentrate on the important high-level concepts. This can be easily achieved through simple noise removal or smoothing. More sophisticated approaches might use information about the underlying process in order to produce simple structures that capture the important semantics of the actual data (see Figure 2d). A typical example for generalization is digital cartography, where geographical features are progressively simplified as the user zooms in or out [83]. In Chapter 4 we study the use of the proper orthogonal decomposition (POD) in order to extract large scale flow behavior and detect vortices in noisy blood flow data.

3 INTELLIGENT CUTAWAY ILLUSTRATIONS

With the increasing amount of complexity in 3D datasets, it gets more and more difficult to locate and visualize important features. Occlusion becomes a problem as soon as the features of interest are not located right at the front. Rendering the data with high transparency could show everything at once, but then, the image is hard to understand. Especially in flow visualization this problem arises frequently, because salient structures often exhibit complex folding and twisting. In this paper, we tackle the problem of occlusion with cutaways, a method originating from scientific illustration.

Scientific illustrators have been dealing with the problem of occlusion for a long time. Occluding geometry is retained where it clarifies the spatial structure but it is removed where it covers important features. A good cutaway illustration has to meet certain requirements. First of all, it should maximize the visibility of important features. Furthermore, the amount of omitted data – even if it is considered not to be important – has to be minimized in order to provide context for the important regions. The cutaway has to respect the user's specification of importance. Finally, the shape of the cutaway has to be comprehensible and the user should be able to understand which parts of the dataset are omitted.

To achieve comprehensibility of the cutaway, we suggest to restrict the approach to fixed, user-defined, parameterized geometric objects such as cuboids, spheres, or cylinders. The goal of this paper is to develop a method which places parameterized cutaway objects automatically and optimally with respect to the aforementioned requirements, because in many cases, this cannot be done manually. For example in Fig. 3a, there is no hint for an important region on the left side, but it is revealed nevertheless by our algorithm in Fig. 3c. There is one problem though: placing cutaways optimally is NP-hard in the number of cutting objects and therefore we cannot hope to find an algorithm which solves this problem directly. Instead we suggest a Monte Carlo (MC) method which can find very good solutions by directed, randomized search.

3.1 METHOD OVERVIEW

The proposed method consists of an interactive step where the user selects interesting parts of the data based on insight or background knowledge, and a non-interactive step where the computer automatically finds an optimal placement of cutaway primitives based on the user selection, as illustrated in Fig. 4. First, the user defines a DOI function that assigns an importance value to each vertex (for polygonal meshes) or voxel (volumetric data). In addition, the user can select the shape and maximum number of geometric primitives that will be used to cut out parts of the visualized object. To achieve comprehensibility of the cutaway, we suggest to restrict the approach to fixed user-defined primitives – in this paper, we use cuboids, spheres, and cylinders. An example of a cutaway for each of those primitives is given in Fig. 5.

After the importance is specified, the computer finds the approximation for the best size and position of the primi-

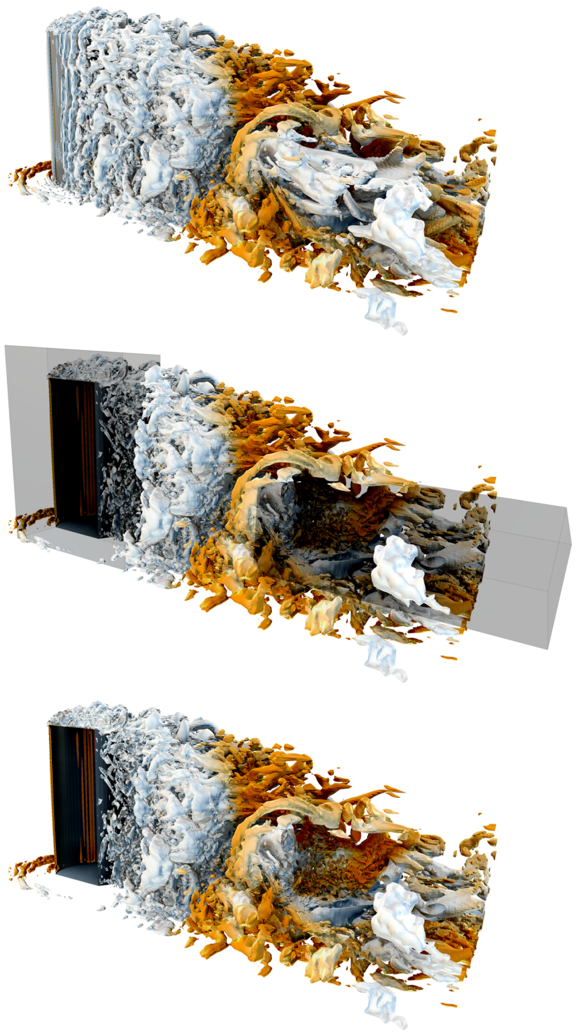


Figure 3: Cutaway image from a flow visualization. (a) Features of interest are colored orange. (b) The algorithm determines the positions of the cutting objects in order to reveal as much of the features as possible. (c) The resulting cutaway.

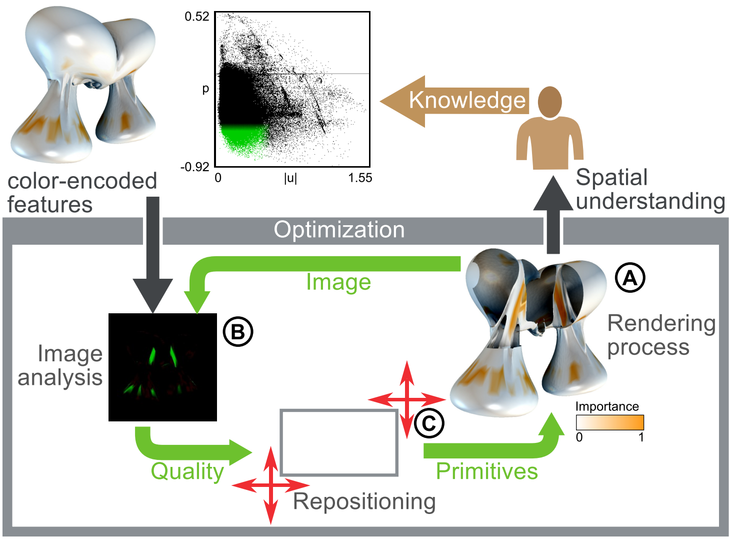


Figure 4: Computer assisted creation of cutaway illustrations. The user knows which features are relevant and can understand a rendering of the data. To place the cutaway geometry optimally in the illustration, an optimization algorithm based on a MC method interacts with the rendering system.

tives without the user's help by cycling through the three stages displayed in Fig. 4: (A) An image of the cutaway is rendered, (B) a quality measure is calculated from the generated image, and (C) the primitives are repositioned and resized. This iterative optimization process is repeated in order to maximize the quality measure.

One key idea is that, based on the feature selection of the user, the rendering system can immediately produce an image that encodes which pixels show important features. This information is used to evaluate the current cutaway positioning and to perform the optimization algorithm. This

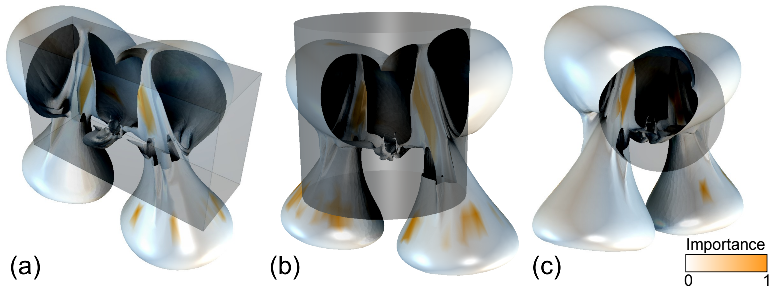


Figure 5: Three primitives used to produce cutaways: (a) cuboid, (b) cylinder, and (c) sphere. Unlike in the real results, primitives are rendered here in order to get an impression of how the method works.

process layout enables a seamless integration in an existing rendering framework because only the resulting image from the renderer is used. By using the appropriate renderers, the method is able to produce cutaways for example for volumetric data or polygonal meshes.

The presented approach has several benefits: First, the cutaway shapes are less complex than the regions of interest and are therefore easier to comprehend. Second, all rendering settings such as transparency and shading are preserved. One can think of the renderer as a black box. This way, cutaway illustrations can be generated from both volumetric data and polygonal meshes. The only change required is to add the ability to take the cutaway objects into account.

3.1.1 *Example: Collision of Two Vortex Tubes*

In order to illustrate the structure and the implementation of the algorithm, a vorticity isosurface of the collision of two vortex tubes simulated by van Rees et al. [95] is used. We extract the isosurface with $\omega = 0.1$ using the algorithm of Schindler et al. [99] and select a range of curvatures to define the region of interest. The examples in Fig. 5 are created using this dataset.

3.1.2 *Multivariate Degree of Interest*

During an initial analysis step, attributes of the data are presented to the user in information visualization views. We use interactive visual analysis based on derived attributes [24] to present feature selection in an abstract way. This method allows us to combine flow features such as vorticity or λ_2 with data attributes such as velocity or pressure or with geometry attributes such as curvature. Based on these attributes, the user can specify what he or she is currently interested in.

3.1.3 *Discretized Configuration Space*

One source of complexity in the cutaway problem is the size of the discretized configuration space. It consists of all possible placements and sizes of the selected primitives in the bounding box of the dataset. For the definition of the discretized configuration space, each dimension of the bounding box is split into n pieces. Doing so, a grid is created on which the primitives will be placed. The configuration space size can be influenced directly by the coarseness of the grid and it does not depend on the complexity of the dataset. More precisely, because an interval can be placed in $\sum_{i=1}^{n} i$ different ways on a grid with n places, the size of the configuration space when placing one primitive is $|D| = (\sum_{i=1}^{n} i)^3$, which grows with n^6.

The algorithm searches for the optimum in the *discretized* configuration space D, which is not the same as the optimum in the continuous space. D contains many local optima which may be located wide apart. During optimization, it is not clear if a local optimum or the global optimum is currently hit. Additionally, because of its size, it is not possible to traverse the whole discretized configuration space. The placement of the primitives is defined by the location and the size of their axis-aligned bounding box. Because two corners are sufficient to determine an axis-aligned bounding box, we get no more than 6 degrees of freedom per primitive.

3.1.4 *Objective Function*

When generating cutaways, perspective plays an important role. A cutaway from one direction may look good and suit the needs of the viewer perfectly. However, from a different perspective, the same cutaway can become meaningless. The objective function has therefore to take into account the camera position and measure the quality of the current solution for further optimization. In order to find the objective function value, the data is colored according to the importance. Then, the data is projected on the image plane and the resulting image is transformed to a single number by averaging the values of a color channel over all pixels. Therefore we allow positive and negative selection using two color channels pos and neg. As illustrated in Fig. 6, the image is produced using an off-screen renderer that draws an image with the objective function values at each pixel as

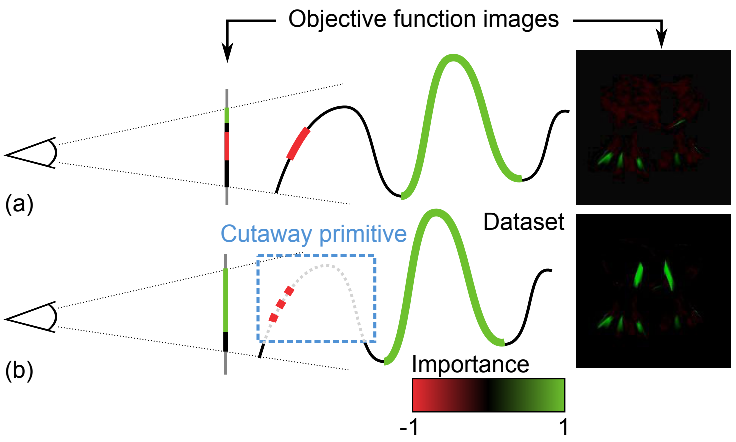


Figure 6: Composition of the objective function images. From the viewpoint of the user, the objective function images are produced by coloring the data according to the importance. Objective function images resulting from the dataset defined in Section 3.1.1 are displayed on the right side. (a) Without cutaway. Most of the important parts (green) are occluded while unwanted parts are visible (red). (b) After subtraction of a cutaway primitive, the important regions become visible and the unwanted regions vanish.

color. The background color is set to black so that it does not influence the quality measurement. The alpha channel is not influenced by this method and hence, transparency in the rendered data can be taken into account as well.

In addition to the requirement given by the user's selection discussed above, the amount of omitted data has to be minimized in order to provide the context for the important regions. A user-defined parameter controls the weighting of the omitted data against the image-based quality defined above. Based on these requirements, we can write down the definition of the objective function to be maximized:

$$f(\mathbf{A}, P) := \frac{1}{d} \sum_{i=1}^{n} \left(\alpha \cdot \mathbf{A}_{i,pos} - \mathbf{A}_{i,neg}\right) - \beta \cdot V(P) \qquad (1)$$

A is the image as a pixel array of length $d = \text{width} \cdot \text{height}$. Each pixel has the properties pos and neg, encoded in the green and in the red color channel, respectively. P is the set of primitives and $V(P)$ measures the amount of omitted data. In our applications, we use either $V(P) = |P|$ or we limit the number of primitives and use their total volume for $V(P)$. The parameters α and β are used for weighting: α balances positive and negative selections while β indicates how much the size of the cutaway geometry is penalized. This objective function has many local optima, for example if revealing important regions requires to clip other important regions.

3.1.5 *Environment of a Configuration*

When implementing an iterative optimization process, it is important to define how the configuration space is traversed. The environment $E(x)$ of a configuration x defines all configurations which are reachable in one step. In our case, the configurations are of spatial nature, and the environment can therefore be defined quite intuitively using the Euclidean distance. Each primitive is defined by its axis-aligned bounding box which can be deformed in 12 ways: For each dimension, either the minimum or the maximum corner is moved one step in positive or negative direction along one axis. However, two rules have to be fulfilled by these deformations: First, the bounding boxes should intersect the bounding box of the data, and second, the signed volume of the bounding box must be positive. Therefore, the algorithm can select from *at most* 12 moves per primitive. In many cases, there will be less than 12 possible moves because one of the restrictions would not be fulfilled by one or several deformations. The deformation step size is defined by the grid of the possible placements of the corners.

3.2 SIMULATED ANNEALING

Simulated annealing (SA) is a MC method to solve complex optimization problems. The inspiration to this algorithm comes from statistical mechanics where the process of heating up and controlled cooling down in metallurgy is modeled. Defects in crystals are reduced by slow cooling and this can be seen as an optimization of the internal energy. Kirkpatrick et al. [62] showed that SA is applicable to general optimization problems and also pointed out that the method performs well for hard to solve optimization problems [62].

At each step, the configuration is changed with a probability depending on the quality difference between the current and the new configuration. To avoid getting stuck at local optima, even configurations that decrease the quality of the result can get accepted. Kirkpatrick et al. proposed to use the acceptance probability introduced by Metropolis et al. for their simulation of atoms in equilibrium at a given temperature T [84]:

$$p_{x \to x_{new}} = \min \left\{ 1, \quad \exp\left[\frac{1}{T} \left(f(x_{new}) - f(x) \right) \right] \right\} \qquad (2)$$

where x is the current configuration and x_{new} is a randomly selected configuration from the environment $E(x)$, defined in Section 3.1.5. SA can therefore be seen as an extension to Metropolis' algorithm originating from the beginnings of computer simulation.

3.2.1 *A Single Optimization Step*

We describe now the algorithm in more detail and provide an overview of the definitions in Table 1. First of all, we explain what happens during a single optimization step. A configuration x in the configuration space D is considered as the start configuration. Then, a new configuration x_{new} is taken randomly from the environment $E(x)$. The transition probability from x to x_{new} is defined in Eq. 2.

As we can see in Eq. 2, if the solution gets better ($f(x_{new}) > f(x)$), it will be accepted immediately because $p_{x \to x_{new}} = 1$; if it gets worse, the acceptance probability decays exponentially with the decrease of the quality, but never reaches zero. This is needed to fulfill ergodicity: Any configuration has to be reachable in a finite number of steps.

3.2.2 *Temperature Regime*

The temperature T used in the acceptance probability $p_{x \to x_{new}}$ has no physical meaning in our case and is used as control parameter. In the beginning, it asserts that even steps that lower the objective function substantially can be accepted. This is needed to leave local optima in order to find the global optimum.

The temperature is controlled by an adaptive schedule. During initialization, the system is heated up as proposed

Name	Description
D	Discretized configuration space.
x	Configuration in D.
$p_{x \to x_{new}}$	Transition probability from x to x_{new}.
$f(x)$	Objective function.
$E(x)$	Environment of x.
T	Temperature.
m_T	Temperature factor.
R	Acceptance rate.
S_H	Number of steps per temperature during heating phase.
S_T	Number of steps before the cooling process starts.
S_S	Number of successful steps that has to be reached at a temperature.
S_{max}	Maximum number of steps at a temperature.

Table 1: Definitions used in the description of the SA algorithm.

by Kirkpatrick [62]. For each temperature, S_H steps are executed and the acceptance rate R is measured:

$$R = \frac{\text{\# accepted steps}}{S_H} \tag{3}$$

As long as R does not exceed a melting threshold (we use 0.8), the temperature is doubled and the procedure is repeated. When the acceptance rate exceeds this value, the system is considered to be melted because the objects are able to float around freely. After having thermalized the system for S_T steps, the cooling process starts. At each temperature, the number of successful steps is counted. When a fixed value S_S is reached, the temperature is multiplied with a constant factor m_T $(0 < m_T < 1)$, leading to an exponential decreasing temperature regime. If the number of acceptances does not reach S_S, the temperature is lowered after S_{max} steps nevertheless $(S_{max} > S_S)$. If S_S is not reached three times in a row, the system is considered to be frozen and the algorithm stops.

3.3 INTERACTIVITY

Interactive exploration of the data while using a cutaway is achieved by a repositioning of the primitives. As a starting point, the optimal position from the initial optimization process described in Section 3.2 can be used. Considering that changes of the cutaway position have to be smooth in order not to confuse the user, the distances between the primitive positions before and after the update should not be large. In order to fulfill this requirement, the grid defined in Section 3.1.3 is refined by increasing n. This leads to smaller steps and hence, to a smoother cutaway repositioning.

Camera movements and time-dependent dataset exploration are handled by the same algorithm even if they are different problems. What they have in common is the smoothness of the changes they induce. This allows us to modify the original optimization result instead of searching for a new global optimum from scratch. For each frame, one step of a hillclimbing optimization is used. In combination with the refined grid, this provides interactivity because it uses a minimal number of objective function evaluations which are the most time intensive part of the optimization process. Hillclimbing traverses the whole environment defined in Section 3.1.5 and compares the qualities of the configurations against each other. Then, either the best among them is picked and the cutaway is repositioned, or the cutaway remains at the same place when no neighboring configuration is better than the current one. Although the configuration space grows with the grid refinement, the performance is not affected because the environment is not dependent on the configuration space size.

At first glance, the selection of a less powerful optimization method might look confusing as the problem to be solved has been described to be hard to solve. However, in the case of interactivity, a different problem is arising: A solution suitable for a similar situation has to be transferred so that its change is not large but nevertheless goes into the direction the optimum takes.

Because the hillclimbing optimization is stopped after one step, we have at most 12 possibilities per primitive (see Section 3.1.5 for justification), meaning 12 evaluations of the objective function and one execution of the rendering step when using one primitive. This leads to an upper bound of 13 rendering steps per frame and enables us to perform updates at interactive speed for many datasets, since the objective function image can be rendered with inexpensive rendering settings such as flat shading.

3.4 IMPLEMENTATION DETAILS

While implementing a new approach for the problem of cutaway generation, we pay attention to seamless integration into any rendering framework. In this section, we explain how the most important step of the algorithm, the objective function evaluation, is implemented and we explain how we integrate it into our existing rendering framework.

3.4.1 *Objective Function Evaluation*

In a first implementation, rendering was performed on the GPU and the evaluation of the objective function was implemented on the CPU. Although a profiling revealed that evaluating Eq. 1 for a rendering on the CPU is no performance problem, it turned out that communication between CPU and GPU took the most time during the objective function evaluation. Because this evaluation is the most often executed part of the algorithm, this issue prevented us from having a fast optimization process. Interactivity was not possible even if we used a single step of a hillclimbing optimization.

The solution is to avoid sending back the image from GPU to CPU by calculating the mean values directly on the GPU. The image is not copied into main memory after rendering but is processed directly on the device. Nvidia CUDA [88] provides an OpenGL interoperability feature which enables CUDA to directly process the data produced by OpenGL. As a starting point, the image post-processing example from the CUDA SDK can be used. After registering the image for access by CUDA, the sum of all pixels has to be calculated. The CUDA data parallel primitives library (CUDPP) is a small toolkit that provides basic operations like a parallel prefix sum for vectors of arbitrary length [47]. Using this feature, the dataset has to be uploaded to the GPU only once in the beginning. In the consecutive steps, only the primitives will be uploaded to, and only the current objective function value (one float) will be downloaded from the GPU. In addition to the speedup by avoiding communication, the parallel prefix sum on the GPU performs also faster than evaluating the objective function on the CPU.

3.4.2 *Integration in an Existing Rendering Framework*

Because the objective function is calculated using only an image of the data, the algorithm can easily be integrated in any rendering framework. The renderer has to take the data, its color, and the primitives to be cut out of the data as input values and it should be able to produce the resulting image without taking into account lights and shadows. Fig. 7 illustrates the dataflow through the algorithm. This means that the method is not dependent on which sort of renderer is used. It can therefore be applied to meshes as well as to volumetric data and it does make sense in both cases to apply it.

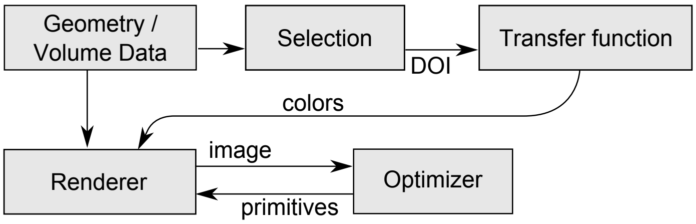


Figure 7: Integration in the rendering framework. The renderer processes the usual inputs (geometry or volume data and colors) plus a list of primitives to be clipped. The optimization process takes an image as input and measures its quality. Then, a new placement of the cut objects is proposed and transmitted to the renderer which produces a new image.

3.5 RESULTS

Because of its straightforward integration into existing rendering pipelines, the algorithm can be applied on different data types. First, we demonstrate the algorithm using a simple example of a piston gas engine in order to get an intuition of its impact. Then, a more complex example originating from a flow simulation output is presented as a demonstration of its performance. As a third example, a volumetric dataset is treated, so that the integration in a volume renderer can be shown.

3.5.1 *Piston Engine Dataset*

The Piston Engine dataset [107] has parts (polygon meshes) which can be selected by the user. A cuboid and a sphere should be used to generate the cutaway displayed in Fig. 8c. This example shows that a cutaway is superior to the transparency based solution when it comes to structure and context of the selected parts.

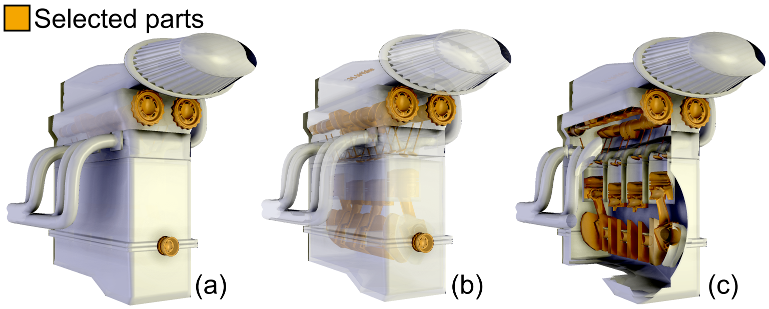


Figure 8: Piston gas engine dataset. (a) Orange parts are selected by the user. (b) Making the unwanted parts transparent makes the important parts visible whereas (c) a cutaway provides more information about spatial structure. In this example, a cuboid and a sphere are used to produce the cutaway shape.

3.5.2 *Colliding Vortex Tubes*

As stated earlier, positive and negative importance can be selected concurrently. The example of the colliding vortex tubes introduced in Section 3.1.1 is used to demonstrate a case where the enclosing geometry has a negative importance and should be removed. At the same time, the geometry inside has a positive importance and should not be removed. The algorithm successfully finds a cutaway by subtracting two cuboids (see Figure 9).

In a second example with the same dataset, we demonstrate the difference induced by transparency. As Figure 10 shows, taking transparency into account leads to a smaller amount of data removed by the cutaway.

3.5.3 *LES Cylinder Flow*

As a realistic case to apply the developed algorithm in flow visualization, the result of a Large-Eddy Simulation (LES) of flow behind a cylinder created by Frederich et al. [39] is used. We are interested in understanding and illustrating the vortex structures emanating from the cylinder. The geometry of the vortex structures is extracted from the data using the method proposed by Jeong and Hussain [57]. In the case study, we use $\lambda_2 = -3$ to create the isosurfaces.

Once the isosurfaces are extracted, we would like to understand the properties of the vortices behind the obstacle. Therefore we want to analyse the pressure p, velocity magnitude $|\mathbf{u}|$ and vorticity magnitude $|\omega|$ distributions throughout the volume. As a first step, we select the respective regions by brushing scatterplots. Fig. 11 illustrates two different selection cases where the locations of the impor-

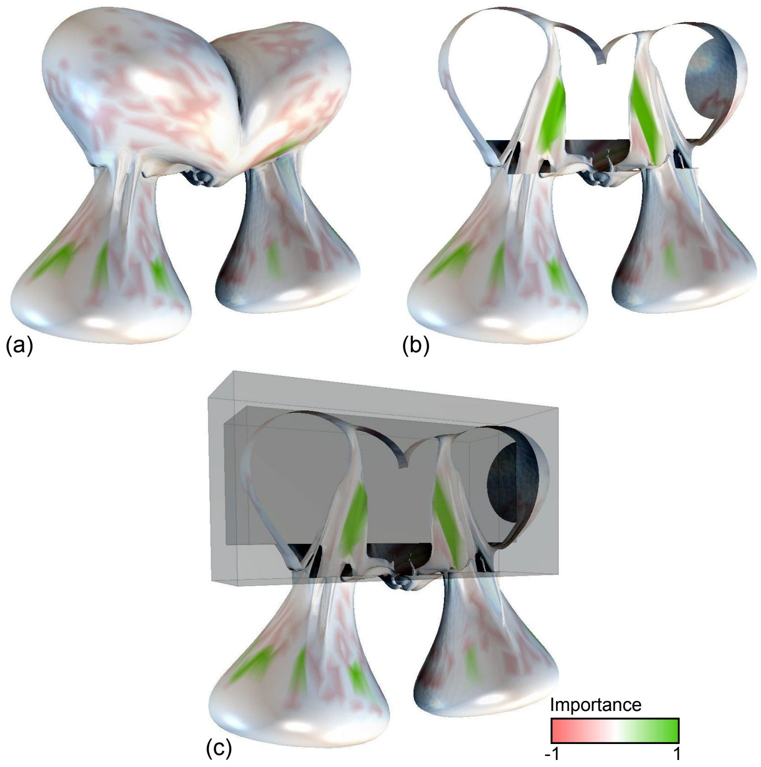


Figure 9: Colliding vortex tubes dataset. (a) Positive (green) and negative (red) importance are used concurrently. (b) The enclosing geometry is removed completely while the parts with positive importance are retained. (c) The cutaway is constructed using two cuboids.

tant regions do not appear to be located at the surface of the data and hence, the algorithm has to find cutaways to reveal them to the user.

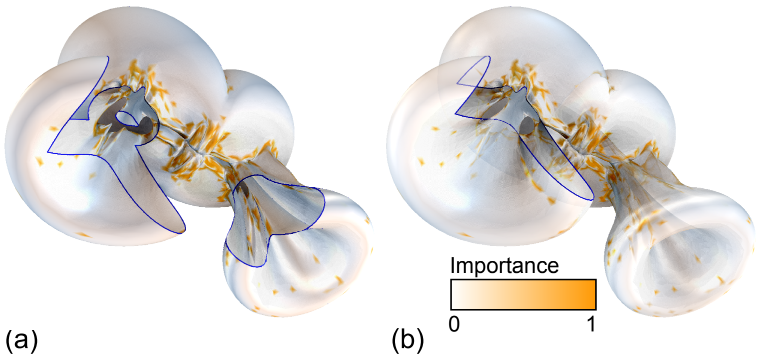


Figure 10: Transparency is taken into account when calculating the objective function. (a) Colliding vortex tubes dataset without transparency. (b) For the same dataset with enabled transparency, a smaller amount of data is clipped.

3.5.4 *Volumetric Data*

In order to show that the algorithm also supports volumetric datasets, an example of a cutaway using the computed tomography (CT) study of a cadaver head from the Stanford volume data archive [105] is provided in Fig. 12. Even though the volume renderer is just a prototype implementation, we can see how the amount of visible data is optimized independent of the rendering algorithm.

3.6 EVALUATION

In MC algorithms such as SA, evaluation plays an important role because they are based on randomness and a stable and reliable solution often needs many steps to establish. Convergence has to be checked and in the case of

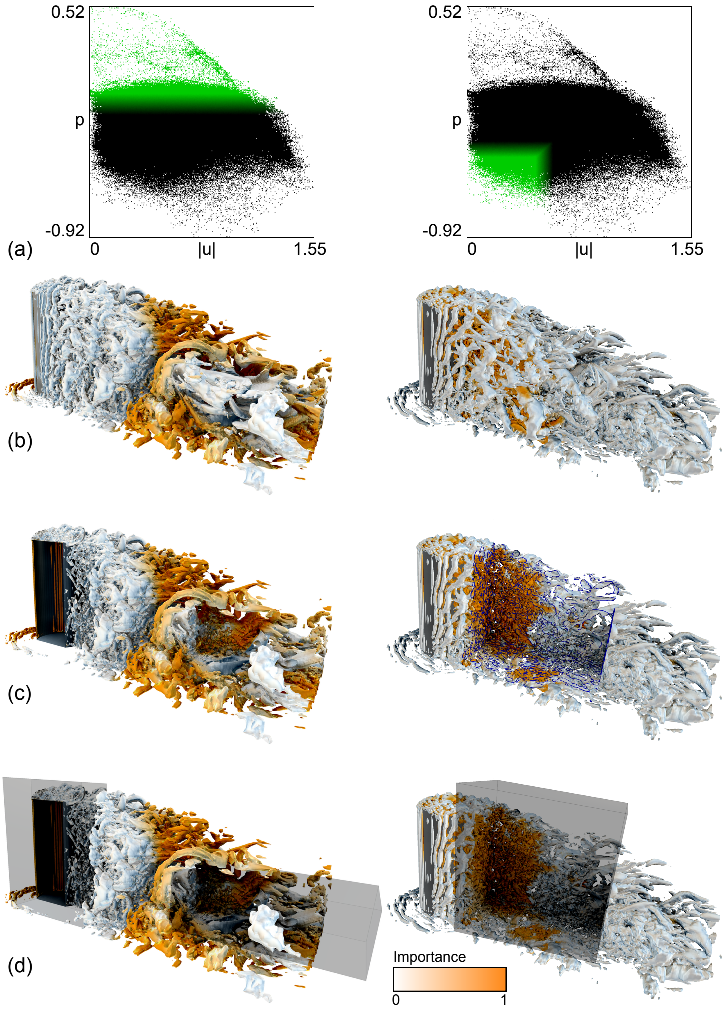


Figure 11: LES cylinder flow. (a) The user selects regions with different physical properties. (b) The data contains the selected features (orange) but they are occluded by other parts of the isosurface. (c) Parts of the data are removed by the cutaway in order to reveal the selected regions. In the right image, the cutting lines are highlighted blue. (d) The resulting cutaway.

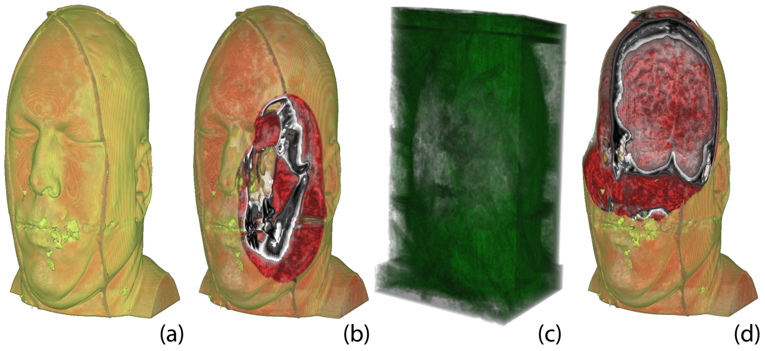


Figure 12: Volumetric dataset. (a) Direct volume rendering. (b) Example of a spherical cut-away. (c-d) Even a rough DOI with the highest values inside the brain leads to the clean cutaway in (d), which maximizes the brain cross-section.

the developed algorithm, it is especially interesting to see whether it can find the global optimum or gets stuck at a local one. We performed several measurements, each of them consisting of a number of runs, and evaluated them statistically. The results of these evaluations are discussed in this section.

3.6.1 *Reference Run*

In order to get reference values, a naïve optimization process has been implemented by traversing the whole configuration space and remembering the best configuration for a very small example. The values calculated using SA are then compared to the outcomes of the naïve process.

Producing the reference values is only possible for small configuration spaces. On the hardware used (2.4GHz Quad CPU, 4GB RAM, NVIDIA GeForce GTX 470), the objective function evaluation rate was approximately 150 evaluations per second for the colliding vortex tubes example introduced in Section 3.1.1 and around 15 evaluations per second for the LES cylinder example discussed in Section 3.5.3. As the configuration space defined in Section 3.1.3 grows with n^6, we use an example with one cuboid and $n = 10$ to perform the evaluation.

3.6.2 *Convergence*

The quality of the result can be seen as a combination of the reached objective function value and the convergence of the algorithm to a fixed configuration. To measure the convergence of the algorithm, the spatial overlap of the primitive sets of different runs is calculated. In order to get a reliable measurement, the same setup is optimized 50 times. The resulting primitive sets are compared pairwise and the average of the overlap values is taken as the measure for the convergence of the algorithm.

Because this measurement is implemented only to evaluate the algorithm, it is not executed every time the program is run. The viewing direction and the zooming factor of the camera are not changed among the different evaluations. This is particularly important because the objective function is view-dependent.

3.6.3 *Evaluation Results*

In Fig. 13, the result of a measurement run is shown. We can see that the algorithm converges with perfect overlap as soon as the search becomes wide enough. All results are normalized with the results from the reference task in order to show the convergence to the global optimum. The most important result is that for $S_{max} > 500$, SA converges to the optimal solution. The values of the constant parameters can be found in Table 2.

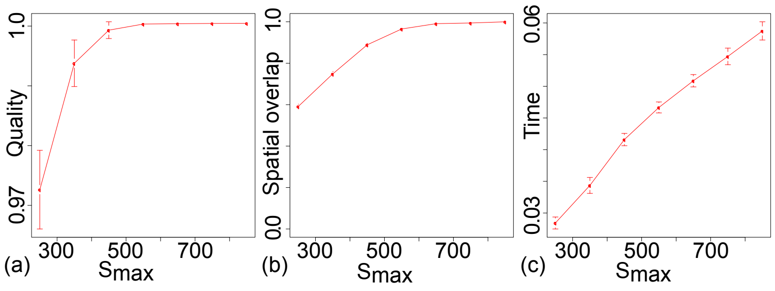


Figure 13: Evaluation results, normalized with respect to the reference run. (a) The objective function approaches the value obtained by the reference run. (b) The convergence measurement shows that the global optimum is reached consistently. (c) SA performs magnitudes better than searching the best position in the whole configuration space. Each data point is produced by taking the mean of 50 samples. The mean time for the samples with $S_{max} = 550$ is 158s, compared to 3388s for the reference run. For all plots, the error bars denote the borders of the 95% confidence interval.

3.6.4 *Parametrization*

Evaluation results from different runs were used to find optimal parameters for SA. It turned out that the maximum number of steps S_{max} and the number of successful steps S_S at a temperature influence the behavior most. In addition, they are related: we have achieved best results with a rate of $\frac{S_{max}}{S_S} \approx 10$. The increasing consistency with a grow-

Name	Description	Value
S_H	Heating steps	100
S_T	Thermalizing steps	100
S_S	Successful steps per temperature	75
n	Grid cells per dimension	10
m_T	Temperature factor	0.9
α	Positive/negative balance factor	1.0
β	Volume penalty	0.001

Table 2: SA settings for the evaluation run. The results of the measurements can be found in Fig. 13.

ing S_{max} can be explained by the design of the algorithm: the lower and upper limits for the number of steps s at a constant temperature are defined by S_S and S_{max}, respectively: $S_S \leqslant s \leqslant S_{max}$. Hence, if both of them increase, more samples are taken to find the equilibrium at each temperature and it is reached more likely.

3.7 DIFFICULTY OF OPTIMAL CUTAWAYS

In this section we show that finding an optimal cutaway geometry is a difficult problem. We will show the proof for cutaway boxes with an objective function that minimizes the number of boxes. The construction for other convex shapes and objective functions is essentially the same. The proof is divided into two parts. First, we start with a simple, two-dimensional problem without occlusion.

CUTPOINTS: Given a $n \times n$ array $\mathbf{A}$ of numbers with $a_{ij} \in \mathbb{N}$, find a set of boxes B that maximizes the objective function

$$f(\mathbf{A}, B) = \sum_{(i,j) \notin B} a_{ij} - \sum_{(i,j) \in B} a_{ij} - |B| \tag{4}$$

where the notation $(i,j) \in B$ stands for the set of all array cells (i,j) for which there is a box $b \in B$ such that the cell (i,j) lies in b. The problem of finding the optimal set of boxes B is NP-hard.

Proof. The proof reduces PLANAR-3SAT to CUTPOINTS. Reduction of geometric problems to PLANAR-3SAT is a common strategy. For a more detailed discussion we refer the reader to proofs for similar problems, such as the rectangle tiling by Khanna et al. [60] or special segmentation and scene analysis by Cooper [29].

Lichtenstein [68] has shown that PLANAR-3SAT is NP-complete. We use the terminology from his paper: In the PLANAR-3SAT problem we are given a 3CNF (conjunctive normal form with at most 3 variables per conjunct) formula F with the additional property that the following graph G_F is planar. The bipartite graph G_F has the variables as one vertex set and the clauses as the other. An edge corresponds to each occurrence of a variable or its negation in a clause. Fig. 14a shows an example of such a graph.

The reduction is now performed as follows: For any formula in 3CNF for which G_F is planar we construct an array $\mathbf{A}_F$ and a number m_F with the following property: F is satisfiable if and only if the optimal set of boxes $\mathbf{B}_F$ maximizes the objective function with $f(\mathbf{A}_F, \mathbf{B}_F) = m_F$. We use the input matrix $\mathbf{A}_F$ as a "drawing table" onto which we embed *variable loops* and *clause gadgets*. More precisely, for every variable in F we create a closed loop of matrix cells with certain values (described below) and for every clause we create a clause gadget, consisting of a block of matrix cells with certain values. Each variable loop intersects a clause gadget precisely if the variable is part of the given clause. All remaining cells are labeled as background cells. Since the graph G_F is planar, we can draw the variable loops such that they do not intersect. The layout of this construction is illustrated in Fig. 14b.

The value of the background cells is set to a large positive number $b = n^2$ and the variable loops are covered by pairs of cells with value -1 interleaved by cells with value 0 as shown in Fig. 14c. This leaves exactly two ways to maximize the objective function value for a single variable loop, with boxes covering either odd or even pairs of negative cells. Each of these layouts corresponds to a truth assignment of the variable.

The values of clause gadget cells are shown in Fig. 14d. The gadget contains parts of the three affected variable loops and a single negative cell in the middle. Only loops with the correct layout (and therefore the correct truth assignment) can extend one of their boxes to cover the central cell without using an additional box, thereby increasing the objective function value. Therefore, the formula F is satisfiable if and only if the optimal solution for Eq. 4 does not use any additional boxes in any clause gadget. $\square$

Corollary 1. *The problem defined in Section 3.1.4 with* $V(P) = |P|$ *is NP-hard.*

Proof. Given an instance of CUTPOINTS, we create a regular quad mesh in the xy-plane with mesh points corresponding to array cells. The view point is now chosen along the

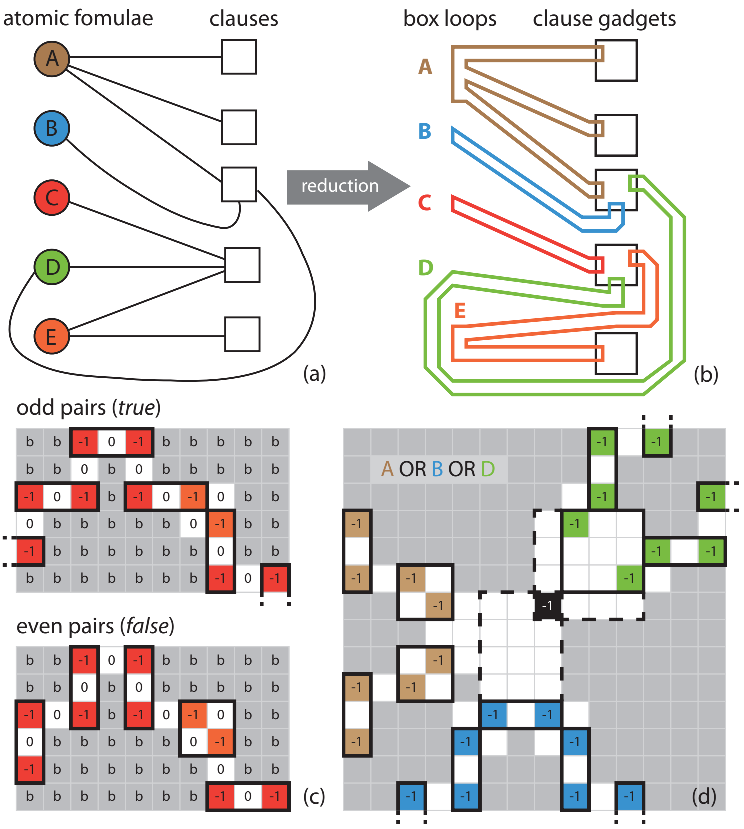


Figure 14: (a) An instance of PLANAR-3SAT. (b) An instance of CUTPOINTS. The array **A** is used as a drawing board to draw one closed loop per variable and one *clause gadget* per clause. (c) Part of a variable loop. There are only two ways to cover all negative cells in the loop with a minimal number of boxes, such that no box includes a background cell. The square element containing the two orange cells is used to swap the layout of the boxes along the loop to account for negations. (d) A clause gadget for the expression A OR B OR D. In this example, A is false and B and D are true. Only loops with the correct box layout can extend one of their boxes to collect the additional black cell in the middle.

z-axis such that every mesh point is uniquely projected onto a screen pixel. If the scene consists only of the quad mesh, there is no occlusion and every pixel is directly related to a point in the scene. Optimizing now the objective function Eq. 1 leads to a set of axis-aligned cutting boxes, and their intersection with the xy-plane solves the CUTPOINTS problem. $\square$

4 VORTEX DETECTION IN 4D MRI DATA: USING THE PROPER ORTHOGONAL DECOMPOSITION FOR IMPROVED NOISE-ROBUSTNESS

In a healthy cardiovascular system, morphology and hemodynamics are attuned to one another. The specific geometry of the heart and vessels allows an efficient blood circulation through the body. The blood flow exerts mechanical forces on the vessel walls and triggers a continuous renewal of the tissue. However, if either morphology or hemodynamics develop anomalies, their synchronized interplay gets out of balance and cardiovascular diseases may develop [96].

In this process of disease development, vortices play an important role. Vortices are not per se an indicator for a cardiovascular dysfunction since healthy blood flow also comprises recirculation and vortices — especially in curved regions or at bifurcations [61, 30]. Still, vortices often occur abnormally because of morphological alterations, e.g., in vessel widenings or after narrowings (stenosis, calcified heart valves). In these cases, the vortical flow alters the pressure and shear forces on the vessel walls (in direction, frequency, or magnitude) and triggers cellular processes leading to aneurysms or atherosclerosis [30]. Also, in slowly rotating vortices the thrombogenesis risk is increased, since the blood may come to a halt and start to clot. These thrombi can cause stroke or heart attacks [40].

Altogether, pathologies are indicated when vortices appear in unusual regions, persist exceptionally long, or show very low blood velocities. Therefore, a strong research interest exists in the more detailed relationship between vortical flow and specific diseases and their progression [40]. For this, vortices are examined in 4D MRI blood flow data. 4D MRI is a flow-sensitive imaging technique, which allows to measure time-resolved blood flow velocities in three dimensions. In these data, automatic vortex detection has so far been done with algorithms commonly used with simulation data [104, 20, 63]. However, the results are not satisfactory yet because 4D MRI has lower resolution and signal-to-noise ratio as well as different noise characteristics than simulation data. Since the vortex detection algorithms rely on first- and second-order derivatives, they are especially sensitive to lower-quality flow data like 4D MRI.

Our approach is therefore to preprocess the 4D MRI data to improve vortex detection. In particular, we propose the use of the proper orthogonal decomposition (POD), which decomposes the velocity field into different scales of motion. Its use is motivated by the fact that vortices in blood flow are medium- to large-scale phenomena, while noise is a small-scale phenomenon. By discarding the high-frequency scales, we seek to improve the quality of the detected vortices.

4.1 4D PC-MRI

"4D PC-MRI" is a technical name for time-resolved three dimensional phase-contrast magnetic resonance imaging, which is a method capable of acquiring a time-dependent vector field of a patient's blood flow [78]. In general, a 4D PC-MRI dataset consists of several time steps spanning a single heart beat, where each time step contains one magnitude and three phase difference images The magnitude image shows anatomical features, while the phase difference images contain the velocity components A slice of a 4D PC-MRI dataset is shown in Fig. 15.

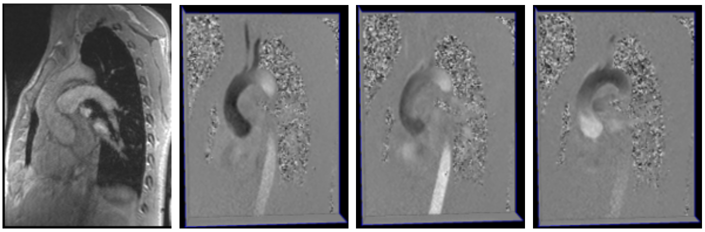


Figure 15: A slice of a 4D MRI dataset with (from left to right) magnitude channel and u,*v* and *w* velocities.

4.1.1 *MRI-related error and noise*

Due to the nature of MRI measurements, the resulting blood flow data contains a high amount of noise and inaccuracies [113]. Partly, these are systematic errors, such as:

MOTION ARTIFACTS Due to long acquisition times (up to 20 minutes), respiration and heart contraction are a source of motion artifacts. These can be countered by respiratory gating approaches [79].

EDDY CURRENTS Fast switching of the magnetic field gradient may induce currents in the patient or the equipment, resulting in a background phase distortion. This distortion field can be estimated and corrected for [66].

PHASE WRAPPING PC-MRI requires the user to set a *velocity encoding parameter*, which defines the maximum measurable velocity magnitude along one axis. If the actual velocity exceeds this threshold, the measured value will simply wrap around, resulting in artifacts. A large number of methods exist for phase unwrapping [69].

Even after correction for systematic errors, the resulting data still contains a high amount of noise. Noise in the raw measurements is often modeled as independent Gaussian noise. Since MRI acquires complex-valued frequency space raw data, noise in the resulting magnitude and phase offset images follow complicated distributions, which, however, can be approximated by Gaussian distributions if the signal-to-noise ratio (SNR) is high enough [44]. Apart from generic image processing algorithms, specialized methods have been proposed for the denoising of magnitude images, focusing on the reconstruction of anatomical features and often considering the Rician distribution of the noise [76]. In flow fields, noise can be reduced by imposing a divergence-free condition inside the blood vessel [25] or comparing measured data with a database of numerically simulated datasets [80]. Unlike these methods, our approach does not require a precise segmentation of the vessel or any a-priori knowledge about the data.

4.1.2 *Vessel segmentation*

A segmentation of blood vessels is mainly required to visualize the vessels themselves, in order to provide anatomical context information. However, several flow visualization methods use this information internally, e.g., to clip stream- and pathlines when they leave the vessel due to noise and numerical errors.

While there are sophisticated methods for a precise segmentation [65, 89], approximate segmentations can be easily obtained by thresholding a scalar field derived from the measured data. Popular choices for the scalar field include the temporal maximum intensity projection of the velocity magnitude or the PCMRA field, defined as

$$\mathrm{PCMRA} = \sqrt{\frac{1}{N}\sum_{t=1}^{N} m^2(t)\left(u^2(t)+v^2(t)+w^2(t)\right)}, \quad (5)$$

where $\mathbf{u} = (u, v, w)$ is the velocity vector and m is the magnitude image [19].

Note that some blood vessels, like the aorta, show considerable movement during each heart cycle. A static segmentation should therefore be used with caution when masking the initial dataset or clipping features like streamlines.

4.2 VORTEX DETECTION

A large number of methods for detecting vortices in a flow have been proposed in the field of flow visualization. Recently, Köhler at al. [63] have compared several of these methods in the context of blood flow visualization. In accordance to their findings, we have chosen the λ_2 method by Jeong and Hussain [57] to visualize vortices in the denoised flow fields. In addition we consider the residual vorticity method recently introduced by Kolář [64, 101].

4.2.1 *The λ_2 method*

The λ_2 method defines a vortex in an incompressible flow in terms of the eigenvalues of the symmetric tensor $\mathbf{S}^2 +$

$\boldsymbol{\Omega}^2$, where $\mathbf{S}$ is the symmetric and $\boldsymbol{\Omega}$ is the antisymmetric part of the velocity gradient $\nabla\mathbf{u}$. $\mathbf{S}$ and $\boldsymbol{\Omega}$ are called the strain rate tensor and the vorticity tensor. A vortex is then defined as a maximally connected fluid region with two negative eigenvalues of the symmetric tensor $\mathbf{S}^2 + \boldsymbol{\Omega}^2$. Due to the symmetry of $\mathbf{S}^2 + \boldsymbol{\Omega}^2$, the eigenvalue decomposition is guaranteed to result in real eigenvalues. In other words, the vortex identification criteria is defined as $\lambda_2 < c$, where λ_2 is the second largest eigenvalue and $c \leqslant 0$ a user-defined constant.

4.2.2 *Residual vorticity*

Like λ_2, *residual vorticity* is defined in terms of $\mathbf{S}$ and $\boldsymbol{\Omega}$. Since $\mathbf{S}$ has real, orthogonal eigenvectors, the coordinate frame can be rotated to become aligned with the eigenvectors of $\mathbf{S}$, called the *strain basis*. The *vorticity matrix* Ω does not change when transformed to the strain basis. In the 2D case, the strain rate matrix can be further decomposed into *mean* and *deviatoric* strain rate, s_m and s_d, and we have:

$$\nabla\mathbf{u} = \begin{bmatrix} s_m & 0 \\ 0 & s_m \end{bmatrix} + \begin{bmatrix} s_d & 0 \\ 0 & -s_d \end{bmatrix} + \begin{bmatrix} 0 & -\omega \\ \omega & 0 \end{bmatrix} \tag{6}$$

The divergence-free terms in Eq. 6 can be rearranged to give a decomposition into *pure shear* and a second part, which is either *residual vorticity* or *residual straining*.

In the "corotation case" $|\omega| \geqslant s_d$, we can decompose

$$\begin{bmatrix} s_d & -\omega \\ \omega & -s_d \end{bmatrix} = \begin{bmatrix} s_d & \mp s_d \\ \pm s_d & -s_d \end{bmatrix} + \begin{bmatrix} 0 & -\omega \pm s_d \\ \omega \mp s_d & 0 \end{bmatrix} \tag{7}$$

where the upper sign applies if $\omega > 0$ and the lower sign if $\omega < 0$. The first matrix on the right hand side describes a velocity field with all vectors parallel to the positive (negative) diagonal and with zero velocity on this diagonal. The second matrix is the residual vorticity matrix. In the "contrarotation" case $|\omega| < s_d$, there is nonzero residual straining, while residual vorticity is zero.

In terms of the velocity gradient (in both original and strain basis!), magnitude of residual vorticity can be computed as

$$\max(|v_x - u_y| - \sqrt{(u_x - v_y)^2 + (v_x + u_y)^2}, 0) \tag{8}$$

where the subscripts denote differentiation.

In 3D, Kolář et al. [64] define residual vorticity (magnitude) at a given point **x** as the maximum of residual vorticity magnitude over the 2D slices through **x** taken in all possible directions. They approximate this maximum by taking slices at one-degree steps in longitude and latitude. This means that $1 + 90 * 360 = 32401$ calculations of 2D residual vorticity have to be made per point.

Rather than using this brute-force approach, we propose to find the maximum using local optimization. There are two easy-to-handle degenerate cases, namely if the (3D) vorticity vector is zero and if the two smaller principal strains (eigenvalues of **S**) are equal. In all other cases, both $|\omega|$ and $-s_d$ have a maximum on a single direction per hemisphere. Unfortunately, this does not imply that the sum of the two terms is a convex function, which would guarantee that local optimization finds the global optimum. As a heuristic, we use the mean of the two directions as the initial value for the optimization process. For a practical verification, we compared our results with those of the brute-force method. For this comparison, we sampled the space of velocity gradients, which has four dimensions after transformation to strain basis, subtraction of the mean strain, and normalization of scaling. We tested on 336000 samples and found the same maximum in all cases. Our method needed on average 189.9 evaluations on 2D slices, as compared to the 32401 of the brute-force method, and does not quantize the result.

4.3 PROPER ORTHOGONAL DECOMPOSITION

The POD [70] approximates each of N outcomes $S^{(n)}(\mathbf{x})$ of a random scalar field variable by a "best fit" linear combination of $M \leqslant N$ uncorrelated *modes* $\hat{S}_i$. The n^{th} field is approximated by

$$\hat{S}^{(n)}(\mathbf{x}) = \hat{S}_0(\mathbf{x}) + \sum_{i=1}^{M} a_i^{(n)} \hat{S}_i(\mathbf{x}), \tag{9}$$

where $\hat{S}_0(\mathbf{x})$ is the mean of all outcomes and $\hat{S}_i$ are eigenfunctions to the M largest eigenvalues of the two-point spatial correlation operator [70]. Because random noise has a low spatial correlation, it tends to reside in modes of low eigenvalues. Hence, if $M < N$, POD removes noise.

POD was originally defined for scalar fields, Lumley [74] extended it to vector fields and showed that the modes decompose a vector field into large-scale and small-scale phenomena. Sirovich [103] proposed what is now known as snapshot POD. The N outcomes are simply taken to be snapshots of a time-dependent vector field at different times, therefore the approximations give a new time-dependent field. Furthermore, he showed that because fields resulting from numerical simulation can be described as linear combination of interpolation polynomials, the coefficients $a_i^{(n)}$ can be directly computed from the eigensystem of a matrix built directly from the values at grid positions. In the case of a time-dependent velocity field, the data consists of N 3D vector fields, each given on a grid with L nodes Thus, each sample consists of a 3D vector field, whose L velocity vectors are lined up to one feature vector. The sample matrix then becomes a $3L \times N$ when we fill up $\mathbf{X}$ column-wise. Eq. 10 illustrates the shape of the data matrix $\mathbf{X}$.

$$\mathbf{X} = \begin{bmatrix} x & x & \cdots & x & y & y & \cdots & y & z & z & \cdots & z \\ x & x & \cdots & x & y & y & \cdots & y & z & z & \cdots & z \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ x & x & \cdots & x & y & y & \cdots & y & z & z & \cdots & z \end{bmatrix}^T \tag{10}$$

Once the POD is computed, a question remains how to choose an appropriate number of modes M. A commonly used option is to require that the contribution of the selected eigenfunctions to the total variance exceeds a fixed percentage, e.g., 90%. Another option is to analyze the *scree graph*, which is a plot of the eigenvalues in decreasing order. One may then choose M as the point where the graph starts to level off. This is very subjective, but can be useful if some a-priori information about the distribution of relevant eigenvalues is known.

Since there is no noise-free ground truth for 4D MRI, we tested the noise suppression capability of POD on synthetic data of a circular flow in a torus. The flow varies over time only by a factor and thus contains a single POD mode. A constant amount of white noise has been added to all (17) time steps, see Fig. 16. While in the full data, the noise accounts for 65.9% of the total energy, this is reduced to 10.2% if data are reconstructed from a single POD mode.

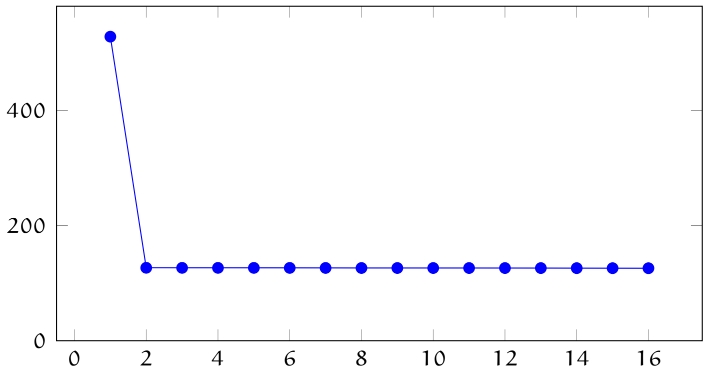


Figure 16: Singular values (proportional to the square root of the energy) in POD of synthetic data, plotted per mode.

4.4 RESULTS

In this section, we compare the POD method with simple Gaussian smoothing for the visualization of vortices in aortic blood flow. In accordance with the findings of Köhler et al. [63], we have chosen the λ_2 method [57] for extracting vortices from 4D MRI data. Note that Pobitzer et al. argue that the POD is incompatible with the λ_2 method since POD extracts large-scale features, whereas λ_2 is related to small-scale features [91]. In our application however, we are interested in large-scale vortices. Disjoint vortex core regions are obtained by using negative isosurface levels [58, 35]. Both methods were implemented in the Visdom visualization framework [112].

4.4.1 *Datasets*

All tests are performed on one of the following three datasets, summarized in Table 3: a healthy aorta from a volunteer

(Fig. 17a), a dataset with an aneurysm in the descending aorta (Fig. 17b), and an aorta with a pathologically distorted shape (Fig. 17c). The datasets are preprocessed as described in [19, 77].

Name	Resolution
Healthy	$192 \times 144 \times 26$ $(1.7 \times 1.7 \times 2.2$ mm$)$ 17 time steps
Aneurysm	$192 \times 144 \times 28$ $(1.77 \times 1.77 \times 2.6$ mm$)$ 20 time steps
Distorted	$192 \times 144 \times 26$ $(1.66 \times 1.66 \times 2.2$ mm$)$ 22 time steps

Table 3: Overview of used datasets. The resolution is given in voxels, values in parentheses indicate voxel size.

4.4.2 *POD of blood flow data*

Fig. 18 shows the spectrum of singular values when applying POD to our datasets. The energy of the modes decays fast, but does not reach zero. Note that a white noise random vector field has a perfectly flat spectrum. It is therefore likely that the first few modes have a high SNR, while the last few modes contain mostly noise.

Because the noise is distributed among all modes, the POD-filtered data will still be slightly noisy, even if using a

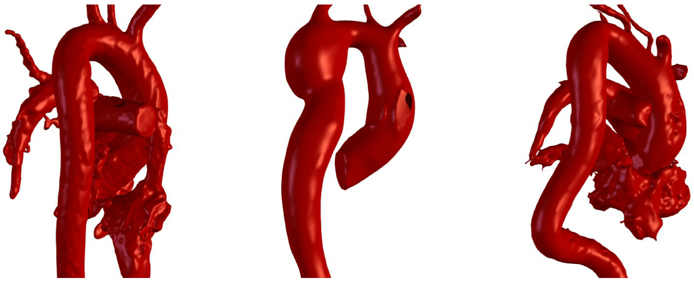

(a) Healthy (b) Aneurysm (c) Distorted

Figure 17: Approximate segmentations of datasets, computed as isosurfaces of the PCMRA field [19]. Models were manually cleaned to improve visual quality.

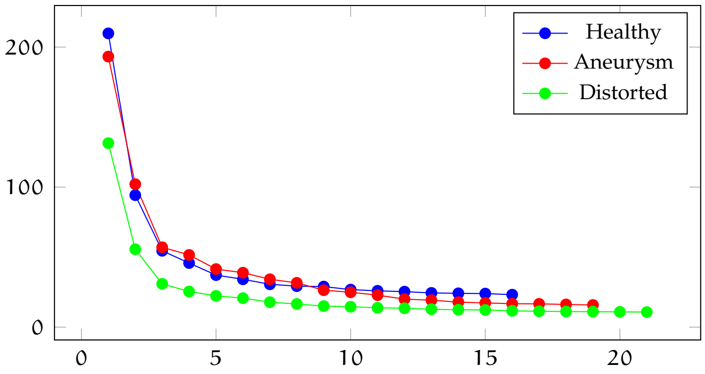


Figure 18: Spectrum of singular values in the POD. The horizontal axis shows the index of the mode, the vertical axis shows the eigenvalue of the mode.

very low number of modes. A way to further reduce noise is to do gradient estimation using a Gaussian derivative kernel instead of finite differences.

4.4.3 *Comparing POD and Gaussian smoothing*

Gaussian smoothing and the POD are fundamentally different methods. However, they both have one parameter (σ and the number of modes, respectively) which controls the strength of the filtering effect. In order to select comparable values of these parameters, we compute the amount of change introduced to the original data by

$$\text{change}(t) = \sqrt{\sum_i \|\hat{\mathbf{v}}(\mathbf{x}_i, t) - \mathbf{v}(\mathbf{x}_i, t)\|_2^2}, \tag{11}$$

where $\hat{\mathbf{v}}(\mathbf{x}, t)$ is the filtered and $\mathbf{v}(\mathbf{x}, t)$ the original vector field. This corresponds to the L^2 norm of the difference between the two vector fields.

Given two filtered datasets with comparable change(t), we can now perform a qualitative comparison of detected vortices.

4.4.4 *Results*

Fig. 20 shows the detected vortices in the healthy aorta dataset. Filtering methods were performed on the original dataset, but the resulting λ_2 isosurfaces were clipped to only include the relevant part of the aorta. Depicted in yellow is the isosurface of the PCMRA field [19], showing an approximate static segmentation of the aorta for context purposes. The number of modes was chosen subjectively after inspecting the spectrum of singular values (Fig. 18). The corresponding value of σ for the Gaussian filtering was chosen such that both methods modify the original data approximately by the same amount (Fig. 19). Fig. 20 shows several combinations of preprocessing meth-

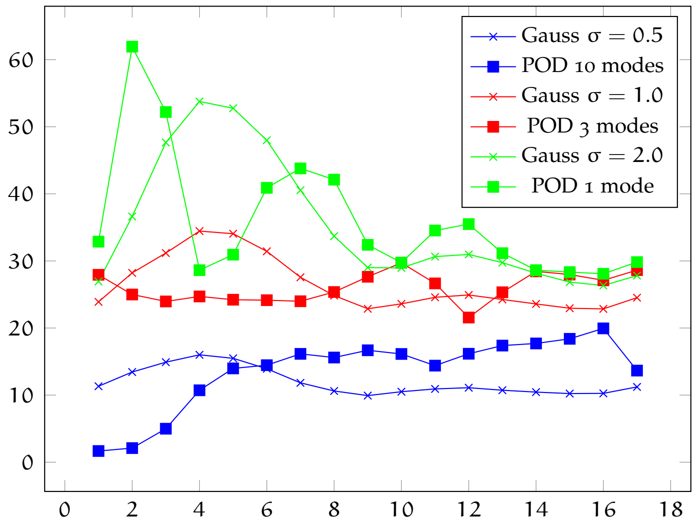


Figure 19: Change in the healthy aorta dataset introduced by filtering methods. $\mathrm{change}(t)$ is plotted against the time step.

ods (POD and Gaussian filtering) and methods for computing derivatives (finite differences and convolution with Gaussian derivatives). Further results are shown in the accompanying video.

The time step chosen for Fig. 20 is at the end of the diastole, where the blood flows slowly and the SNR is small. Medical literature tells us that the blood forms rotational flow patterns at this time (see Fig. 21), which is consistent with the tube-like structures of the λ_2 isosurfaces. Note that naïvely using finite differences without any preprocessing does not produce useful results. A POD reconstruction with 3 modes shows considerably less noise and vortical structures start to become visible. In comparison, Gaussian filtering yields smoother results at the cost of finding fewer features. After using Gaussian derivatives, results are comparable in visual quality, while the POD method shows more features. In order to verify that features visible in the POD-processed dataset correspond to a vortical flow, we seed streamlines of the original dataset around the features (Fig. 24).

Similar results can be observed in the distorted aorta dataset in Fig. 22, again at the end of the diastole. As before, the POD reconstruction with 3 modes and Gaussian filtering are comparable in strength, with a difference in $err(t)$ of less than 5%. In order to check that the features found in the POD-preprocessed dataset are not false positives, we have seeded streamlines around the isosurfaces of λ_2, as shown in Fig. 24.

A slightly different result can be seen in Fig. 23, showing the aneurysm aorta dataset near the end of the systole, The time step chosen here is near the end of the systole, where the blood flows faster and the vortices are stronger. The isovalue of λ_2 is therefore chosen significantly smaller than in the previous figures. Note that the signal-to-noise ratio is higher than previously and even a weak smoothing filter already gives nice results. Nevertheless, the POD method produces consistent results and does not miss features when compared to the Gaussian filtered data. Difference in $change(t) \approx 15\%$. The blood flows fast at this time step and the signal to noise ratio is high.

4.4.5 *Comparison of vortex detection methods*

Vortex detection in 4D MRI data is typically done using the λ_2 method [63, 104]. We also implemented residual vorticity as described in Sec. 4.2.2, using the Polak-Ribière optimization method from the GNU Scientific Library [43]. Fig. 25 compares results for the healthy aorta and aneurysm datasets. The time steps with highest average velocity were chosen, and no POD filtering was used. The two methods turned out to be highly consistent in general. A difference can be seen in the ascending part of the healthy aorta, where there is a pair of roughly parallel vortices. Here, the vortex appearing more prominently in the residual vorticity isosurface is the stronger of the two, as can be concluded from the streamline pattern and the Sujudi-Haimes vortex core lines. These two means of verification are valid since we have longitudinal vortices here. We found that residual vorticity is a valid, though computationally slightly more expensive, alternative. The average number of 2D slices to be evaluated per 3D residual vorticity computation was 43.4 and 40.6 for the two datasets.

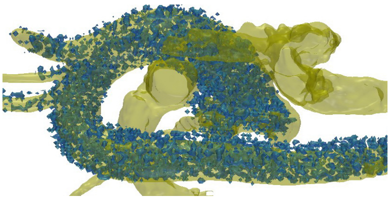
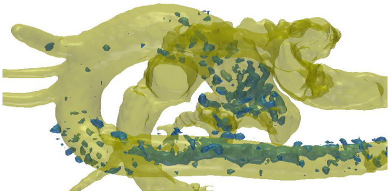
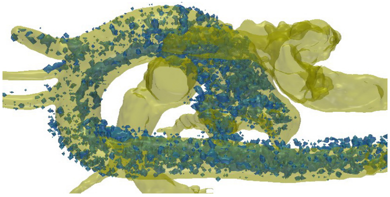
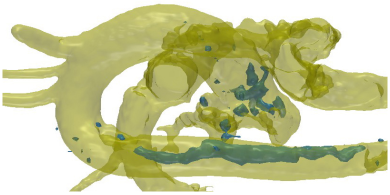
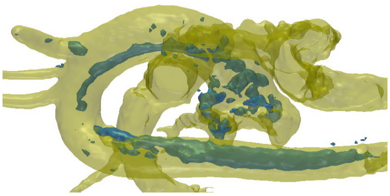
Isosurface of $\lambda_2 = -100$	**Filter**	**Derivatives**
	None	Finite differences
	Gauss ($\sigma = 1$)	Finite differences
	POD (3 modes)	Finite differences
	Gauss ($\sigma = 1$)	Gauss ($\sigma = 1$)
	POD (3 modes)	Gauss ($\sigma = 1$)

Figure 20: Isosurfaces of $\lambda_2 = -100$. Comparison of filtering methods for the healthy aorta dataset at the end of the diastole.

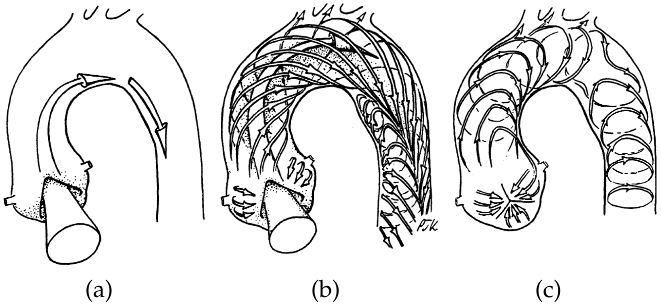


Figure 21: Illustrative visualization of the blood flow in the aorta. (a) Early systole, accelerating axial flow. (b) Mid systole, secondary helical flows develop. (c) Late systole, rotational and recirculating secondary flows. Image reprinted with permission from [61].

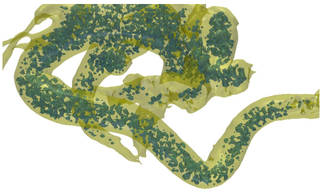
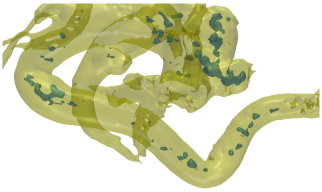
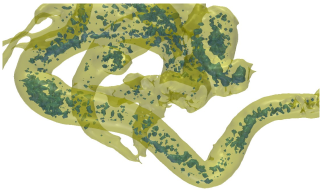
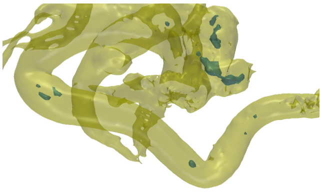
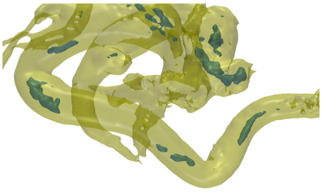
Isosurface of $\lambda_2 = -80$	**Filter**	**Derivatives**
	None	Finite differences
	Gauss ($\sigma = 1$)	Finite differences
	POD (3 modes)	Finite differences
	Gauss ($\sigma = 1$)	Gauss ($\sigma = 1$)
	POD (3 modes)	Gauss ($\sigma = 1$)

Figure 22: Isosurface of $\lambda_2 = -80$. Comparison of filtering methods for the distorted aorta dataset at the end of the diastole.

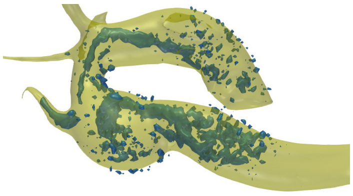
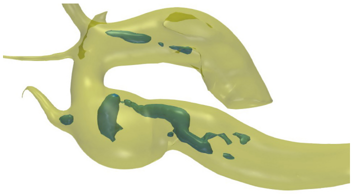
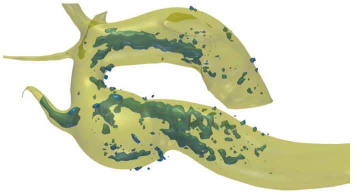
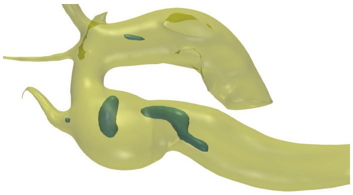
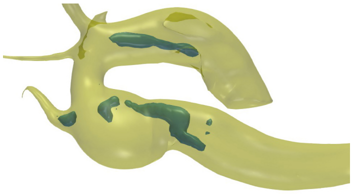
Isosurface of $\lambda_2 = -3000$	**Filter**	**Derivatives**
	None	Finite differences
	Gauss ($\sigma = 1$)	Finite differences
	POD (4 modes)	Finite differences
	Gauss ($\sigma = 1$)	Gauss ($\sigma = 1$)
	POD (4 modes)	Gauss ($\sigma = 1$)

Figure 23: Isosurface of $\lambda_2 = -3000$. Comparison of filtering methods for the aneurysm aorta dataset at the end of the systole.

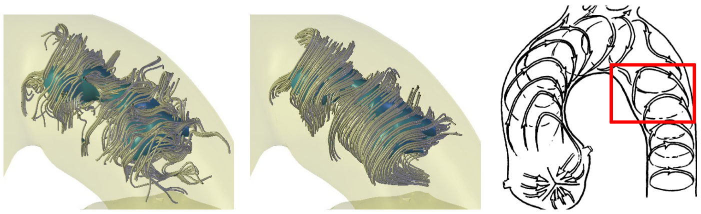
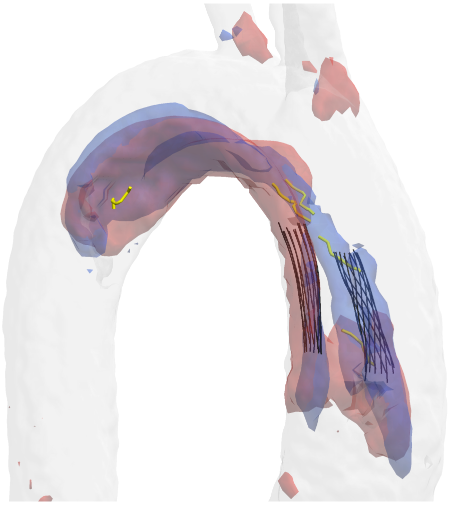
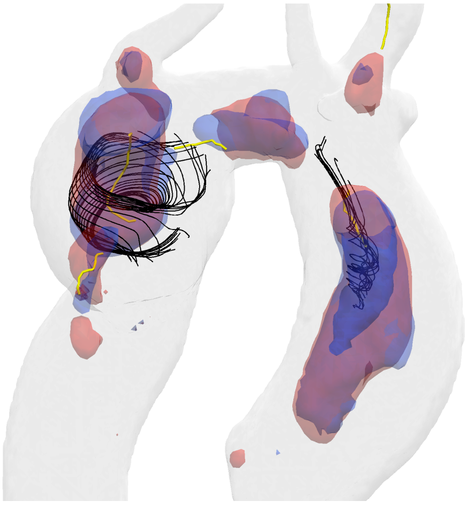

(a) (b) (c)

Figure 24: Streamlines around features found in the POD-filtered distorted aorta dataset, showing that those are not false positives. (a) Streamlines in the original dataset. (b) Streamlines after Gaussian smoothing with $\sigma = 1$. (c) Corresponding region with expected helical flow.



(a) Healthy



(b) Aneurysm

Figure 25: Vortices in a late systole time step of the healthy aorta (top) and aneurysm (bottom) datasets. Images show isosurfaces of λ_2 (red) and residual vorticity (blue), Sujudi-Haimes vortex core lines (yellow), and some manually seeded streamlines (black).

5 CONCLUSION

In this part we have presented two flow visalization methods based on higher-level abstractions and targeting clutter and occlusion problems.

In the first method, we have presented a fully automatic approach to position cutaways based on a degree of interest specified by the user. Even though we only apply the method to automatic cutaway placement, the approach is general enough to solve a wide range of view-dependent optimization problems.

One benefit of our approach is the fact that rendering parameters such as the position of cutaway boxes does not need to be specified by the user, avoiding thus tedious user interaction. In contrast to this, we do not suggest to take the specification of semantics out of the hands of the user: Specification of importance is what the user is interested in and we suggest to use interactive visual analysis for this task.

A limitation of the method is that the type of primitives has to be chosen by the user, however we argue that using known geometries eases perception.

We also show that the problem of placing cutaways optimally is NP-hard. Some approaches solve this by using heuristics and hoping that a good solution will arise for many relevant inputs. In this paper we suggest a solution based on a randomization strategy and show that it quickly converges to the optimal solutions for a problem which is small enough. In general, the presented approach created good solutions for all datasets presented in this paper and we have experienced it to work robustly during evaluation.

This can be a sign that simulated annealing can be a good starting point for other optimization problems in visualization. However, it is important to realize that not all instances of a problem that is NP-hard are difficult to solve, and another route for future research might be to see if the problems that appear in practice can be solved quickly by a potentially exponential algorithm.

In the second method, we have analyzed the use of the POD method for the preprocessing of PC-MRI blood flow dataset and its effect on the detection of vortices using the λ_2 method. Our results show that data processed with the POD method yields better results than the unprocessed data.

A simple Gaussian filtering with the same filtering strength as POD produced smoother results, but resulted in fewer vortices being detected. As some amount of noise is contained in all POD modes, the standard gradient estimation based on finite differences does not produce sufficiently smooth results. By using instead a Gaussian derivative kernel, we achieved good results.

Note that Pobitzer et al. argue that the POD is incompatible with the λ_2 method since POD extracts large-scale features, whereas λ_2 is related to small-scale features of their turbulent flow [91]. In our application however, we are interested in large-scale vortices and found the POD method to be applicable.

As there is no ground truth for measured MRI data, we did not perform a rigorous quantitative analysis of the POD method. Such an analysis could be performed on numerically simulated data with a physically correct model for PC-MRI noise, which is an interesting topic for future work.

Part II

LOW LEVEL ABSTRACTION

In this part, we present a 2.5D illustrative rendering framework and two methods based on lower-level abstractions. These methods use non-photorealistic rendering in order to improve the perception of shape or direction on multi-layered transparent surfaces.

6 OVERVIEW

The use of non-photorealistic techniques to enhance the perceptual effectiveness of hand drawings has been used for a long time by illustrators [52]. Their list of tricks and techniques for adding shape, depth, material, or directional cues is long and many techniques have been adopted into visualization algorithms. Figure 26 shows examples of such low level abstractions.

An important technique for enhancing shape perception is adding depth cues. While a realistic illumination model (including shadows and global illumination effects like ambient occlusion) greatly improves the depth perception, sometimes it is too expensive or incompatible with other techniques. In this case, additional depth cues like unsharp masking [72] or volumetric halos [23] may be useful.

When rendering objects with many occluding surface layers, transparency may be used to show multiple layers in one image. However, using a constant surface transparency tends to produce images where understanding individual layers is difficult. Some techniques therefore use an adaptive transparency [32, 53, 45], add additional cues to improve the shape perception [54, 55, 73], or try to minimize the difference between actual and perceived transparency [27]. Instead of using traditional transparency based on alpha blending, multiple layers may also be visualized using color weawing [71] or overlaying different visualization techniques [108].

Several techniques seek to convey the shape and structure of rendered objects by using lines, such as silhouettes

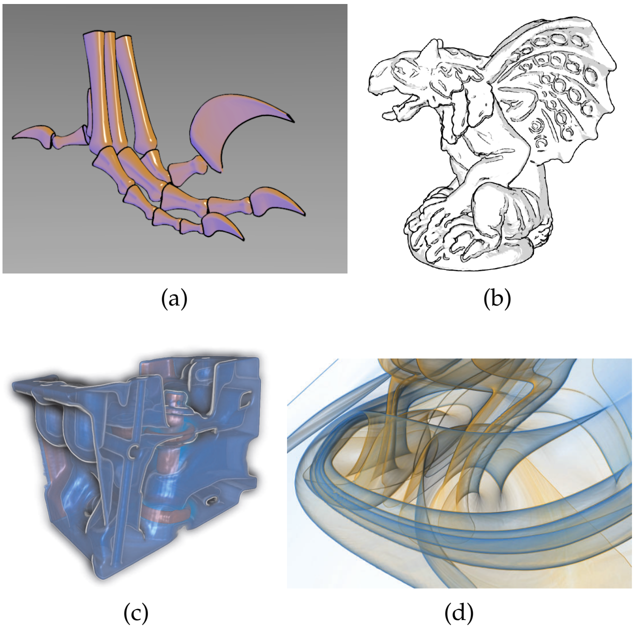


Figure 26: Examples of low level abstractions. (a) A non-photorealistic lighting model [42]. (b) Conveying shape using Laplacian lines [116]. (c) Depth cues using volumetric haloes [23]. (d) Shape enhancing surface transparency [53].

or contours [31, 59, 116]. Line-based techniques are important because they use pixels very economically without occluding inner structures. The drawback is though that it is difficult to apply proper shading. Complex objects also often produce a dense set of feature lines which is difficult to understand. If the lines occlude each other, a haloed line effect may be used to improve the perception of the relative depth ordering [14, 36].

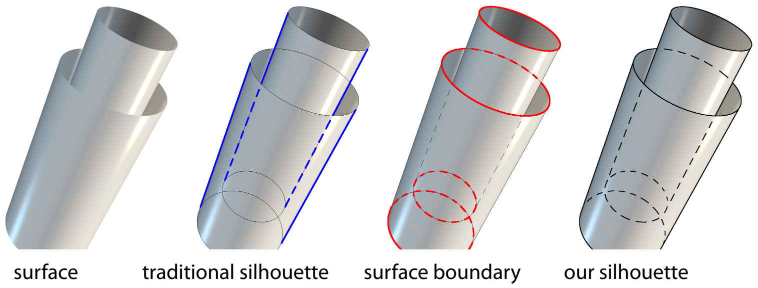


Figure 27: Silhouette lines for two open concentric cylinders.

Another approach for improving the shape perception is using a non-photorealistic shading model. Gooch shading uses model that uses both luminance and hue to indicate surface orientation [42]. Light warping locally compresses patterns of reflected lighting to enhance shape depiction [109].

6.1 DEFINITIONS AND SYMBOLS

6.1.1 *Silhouettes and contours*

As discussed above, artists often enhance silhouette lines of the illustrated surfaces. Since there is a significant variability in terminology, we give here our definition of a *silhouette*, as illustrated in Figure 27. Traditionally in computer graphics, the silhouette is defined as the set of points where the surface normal is perpendicular to the view direction (blue lines) [56]. However, this definition does not capture the surface boundary (red lines), which intuitively belongs to the silhouette as well. Therefore, we define a *silhouette* as the union of the two aforementioned sets (black lines).

6.1.2 *Pixels and fragments*

For the purpose of this work, we define a pixel as the smallest addressable image element and a fragment as an intersection of the rendered surface with a ray going through the pixel center. Typically, surfaces are rendered as triangle meshes. Large triangles that cover multiple pixels in the image will therefore consists of multiple fragments. Conversely, small triangles that cover an area far smaller than a pixel might not generate any fragment at all. Some applications use multiple fragments for each pixel for the purpose of anti-aliasing (e.g., the supersampling technique). For the sake of simplicity, we do not discuss this case here, however, it would be straightforward to extend our methods to handle multiple fragments per pixel.

6.1.3 *Symbols*

Table 4 shows a list of important variables used in the following chapters. As the methods presented in those chapters work in screen space, coordinates are generally given in screen space, unless otherwise noted.

Name	Description
$\mathbf{x}$	Screen (pixel) coordinates of a surface point.
$\mathbf{x}_e$	Eye coordinates of a surface point.
$\mathbf{x}_w$	World coordinates of a surface point.
$\mathbf{u}$	The vector field at a surface point (in screen coordinates).
$\mathbf{n}$	The surface normal at a surface point (in screen coordinates).
ρ	The grayscale color of a surface point
α	The transparency of a surface point
Ω	The screen space computational domain, i.e., all pixels that contain some part of the surface.
k	The current iteration for iterative methods.
t	Time.

Table 4: Definitions of variables used throughout this part.

7 ILLUSTRATION BUFFER

The illustration buffer is a novel 2.5D image space representation of a rendered scene. For each pixel of the image, it provides access to all surface fragments that intersect a viewing ray through that pixel. Additionally, for each fragment, it provides access to neighboring fragments along the surface. This flexible representation can not only be used to implement a wide range of existing transparency assignment techniques, but serves also as a base for novel 2.5D image filters.

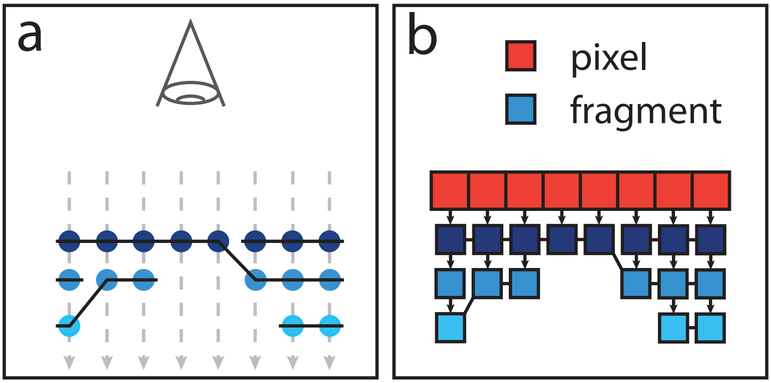


Figure 28: The illustration buffer. (a) A cross section of a surface with three layers. Dashed lines indicating view rays (one for each pixel). (b) The corresponding buffer layout. Horizontal lines between fragments indicate explicit links to geodesically neighboring fragments.

Before describing the illustration buffer, we review the conventional way of generating images. Traditionally, graphics cards use two buffers to store the rendered image: the color buffer (or frame buffer) stores the color of each pixel and the z-buffer stores its distance to the viewer. Whenever an object is rendered, its polygons are projected onto the screen and rasterized into fragments. Each fragment then checks in an atomic way if it is closer to the viewer than the currently stored distance at the given pixel. If the fragment passes the z-buffer test, it updates both the color and depth information.

If the scene contains transparent surfaces rendered with alpha blending, the surfaces need to be rendered from back to front. Simply sorting objects by their distance to the camera does not work since triangles can intersect each other. Guaranteeing correct results in this case is the goal of so-called order-independent transparency rendering methods. Early appoaches are based on the depth peeling technique by Mammen [75]. In this technique, the front-most fragments for each pixel are extracted and processed as a flat 2D image. This depth peeling layer is illustrated by the dark blue fragments in Figure 28. After that, the next layer (light blue fragments) is extracted and blended with the previous layer. This is repeated until no more fragments are available. A disadvantage of this technique is that it requires multiple rendering passes and that some of the depth peeling layers contain very few non-empty pixels (in Figure 28, the first layer contains 8 pixels, while the third layer contains only 3).

A different approach is followed by the A-buffer, introduced by Carpenter [26]. In this technique, the color buffer is replaced by a linked list of fragments for each pixel. After all surfaces have been rendered, each linked list is sorted individually, guaranteeing a correct order for each pixel. Early GPU implementations of the A-buffer are presented by Bavoil et al. [17], and Myers and Bavoil [85]. However, both implementations can only store a limited number of fragments in each list. Yang et al. [115] present an unbounded GPU implementation of the A-buffer by exploiting recent advances in graphics hardware. This work is a major inspiration for the implementation of our technique.

Similar to the A-buffer, our *illustration buffer* stores an unbounded linked list of all surface layers for each pixel. Unlike existing implementations, however, each fragment stores an arbitrary number of surface properties for deferred computations, as well as explicit links to geodesically neighboring fragments. The illustration buffer is implemented on the GPU and can be used at interactive frame rates.

Having simultaneous access to all fragments along a viewing ray as well as neighboring fragments along the surface makes our representation very flexible. It can not only be used to implement a wide range of existing transparency assignment techniques, but serves also as a base for novel 2.5D image filters. Note that for filters where surface fragments need to exchange information along the surface (such as solving a differential equation on the surface using finite differences), a naïve depth peeling approach is not appropriate. The depth peeling layers contain adjacent fragments from different surfaces, and surfaces may pass through different depth peeling layers. The filter would therefore need to exchange information with other depth peeling layers, effectively creating a 2.5D data structure similar to the illustration buffer.

7.1 BUFFER LAYOUT

The illustration buffer data structure consists of several separate buffers, which are all stored in graphics memory. Each buffer represents an array, whose elements store a fixed number of values. There are three types of buffers: those that contain a constant number of elements, those that contain one element per screen pixel, and those that contain one element per surface fragment. The following list describes the individual buffers. Their basic layouts are illustrated in Figure 29.

- `fragData` contains the data for all fragments. Each element in this buffer stores a number of application dependent properties of one fragment, such as the fragment position, normal direction, and color. This buffer is implemented as a 1D four channel texture object, with one buffer element spanning N consecutive texels, leaving room for 4N properties for each fragment.

- `fragData2` has the same layout as `fragData` and is used in ping-pong computation schemes.

- `fragNext` contains for each fragment the index of the next fragment belonging to the same pixel. This is similar to a "next" pointer in a linked list.

- `pixelHead` contains for each pixel the index of first fragment belonging to that pixel. This is similar to a "head" pointer in a linked list.

- `pixelCount` contains for each pixel the number of fragments belonging to that pixel.

- `fragCount` contains one single element, storing the total number of fragments written so far. This is also the index of the next free cell in the `fragData` array.
- `fragCountMax` is a constant number storing the size of the `fragData` buffer.

7.2 BUFFER FILLING

As in conventional rendering, polygons are first rasterized into fragments. This is done automatically by the graphics hardware. Each fragment is then simply inserted into the fragment list, using a fragment shader described in Algorithm 1. The shader consists of three important steps: First, `fragCount` is increased by one to get the index of a free cell in `fragData` (line 1). Next, the previous value of `pixelHead` is stored and replaced by the new index (line 3). After this step, `prevHeadId` and `newFragId` contain the old and the new fragment list heads, respectively. Finally, these two list elements are linked (line 4). The return value of this function is then the index of a free element that can be used to store all fragment properties.

The above algorithm needs several non-standard fragment shader operations, all of which are available in current graphics hardware: atomic operations for protection against concurrent access, and read and write from arbitrary memory locations in a buffer. In our implementation, we access this functionality through OpenGL with the extension *GL_EXT_shader_image_load_store*.

Note that `fragData` can only store a limited number of fragments and any additional fragments are simply ignored. After each rendering, we check if the number of generated

Algorithm 1 Inserting a fragment into the illustration buffer

```
1: newFragId ← add(fragCount, 1)
2: if newFragId < fragMaxCount then
3:     prevHeadId ←
       exchange(pixelHead[x,y], newFragId)
4:     fragNext[newFragId] ← prevHeadId
5:     return newFragId
6: else
7:     return -1
8: end if
```

fragments was larger than the buffer size. If this was the case, we resize the buffer to the required size and repeat the whole rendering process. This approach might be seen as wasteful of resources. However, we regard this limitation as temporary since current graphics hardware already contains specific constructs for dynamically sized buffers, available, e.g., in DirectX 11 as *append/consume buffers*.

7.2.1 *Fragment sorting*

In the second step, the fragment lists are sorted by depth. This is implemented with a *full-screen render pass*, where a single quad covering the whole image is rendered. The fragment shader for this pass uses selection sort to construct the sorted sequence. This sort needs to repeatedly traverse the input fragment list. However, unlike previous methods based on insertion sort [115], it only uses one write operation per output list element and does not need a fixed-size local array. In our experiments, we have found no performance difference to the insertion sort method, presumably because repeated read operations benefit from caching.

At this point, the sorted elements could be used to compute the final pixel color. Unlike existing A-buffer implementations, however, we use the sorted lists in multiple subsequent passes and therefore we copy them back to global memory. For efficient access, the list elements are stored in consecutive buffer elements. Since this reordering cannot be done in-place, the second buffer `fragData2` is used to store the sorted lists. Similar to Algorithm 1, the corresponding shader program first reserves a block of free buffer elements. The index of the first element as well as the number of elements in the block are stored in `pixelHead` and `pixelCount`, respectively.

7.3 NEIGHBOR SEARCH

In the third step, we connect each fragment to its four geodesic neighbors. More precisely, for a fragment with pixel coordinates (x, y), we search and store the index of the four nearby fragments with pixel coordinates $(x + dx, y + dy)$ with $(dx, dy) \in \{(-1, 0), (1, 0), (0, -1), (0, 1)\}$ in the `fragData` buffer. This information is used later to access the surface neighborhood of a fragment. Since fragments at the boundary or silhouette of a surface have fewer than four neighbors, a special value is used to indicate missing neighbors.

The problem of finding the correct neighbors with screen space information only is inherently difficult: in the situation depicted in Figure 30(a), one cannot decide how the fragments are connected. One difficulty is that during the rasterization of the triangle mesh into fragments, all connectivity information is lost. Additionally, using object space information is not simple: for finely triangulated surfaces, or for surfaces which are almost parallel to the viewing direction, neighboring fragments may lie on triangles which are very far apart (see Figure 30(b)). Since we cannot afford a global object space search for each fragment, we use a heuristic screen space method to decide which fragments are connected.

In practice, the visualized surfaces are often locally smooth and continuous. We therefore use a heuristic measure ϵ for the continuity of a fragment pair, as shown in Figure 31(a). This measure will be small for neighboring fragments along a continuous surface, and large for two fragments belonging to different surface layers. Given such a measure, a necessary condition for two fragments i and j being geodesic neighbors is that $\epsilon(i, j)$ is smaller than the measure between i or j and any of the other neighbor candidates, i.e., that both fragments think of each other as the best candidates among the fragment list of the adjacent pixel (see Figure 31(b)).

Our neighbor search is based upon the above observations and is again implemented in a full-screen render pass. The fragment shader traverses the fragment list of the current pixel and for each fragment, uses Algorithm 2 to find the corresponding geodesic neighbor. This search is repeated for each adjacent pixel.

The continuity measure $\epsilon(i, j)$ used in our neighbor search is given by

$$\epsilon_z(i,j) = \frac{1}{r}\left[(z_e)_i + (\mathbf{x}_j - \mathbf{x}_i)\cdot\left(\frac{dz_e}{d\mathbf{x}}\right)_i - (z_e)_j\right] \quad (12)$$

$$\epsilon_n(i,j) = 1 - \mathbf{n}_i \cdot \mathbf{n}_j \quad (13)$$

$$\epsilon_{uv}(i,j) = \left\|\mathbf{uv}_i - \mathbf{uv}_j\right\| \quad (14)$$

$$\epsilon(i,j) = w_z \cdot \epsilon_z(i,j) + w_n \cdot \epsilon_n(i,j) + w_{uv} \cdot \epsilon_{uv} \quad (15)$$

Algorithm 2 Finding geodesic neighbors.

```
1: fragNeighbor ← -1 {current neighbor candidate}
2: eNeighbor ← ∞ {measure $\epsilon$ for fragNeighbor}
3: {find the best candidate for A among all B's}
4: for all fragB ∈ fragListB do
5:    eB ← $\epsilon$(fragA, fragB)
6:    if eB < eNeighbor then
7:       eNeighbor ← eB
8:       fragNeighbor ← fragB
9:    end if
10: end for
11: {check if there is a better candidate among all A's}
12: for all fragA2 ∈ fragListA do
13:    eA ← $\epsilon$(fragA2, fragNeighbor)
14:    if eA < eNeighbor then
15:       return -1
16:    end if
17: end for
18: return fragNeighbor
```

Here r is the radius of the bounding sphere of the rendered object and $\mathbf{x}_i$, $\mathbf{n}_i$, $\mathbf{uv}_i$, $(z_e)_i$, and $(\frac{dz_e}{dx})_i$ are the pixel coordinates, normal vector, texture coordintes, eye space z coordinate, and z coordinate gradient of fragment i, respectively. Note that the gradient corresponds to the slope of the triangle that produces the fragment and can be computed efficiently with the GLSL shader instruction *dFdx*. In our measure, ϵ_z is a second-order difference in the normalized fragment z coordinate, and ϵ_n a function of the angle between the fragment normals. Both components are small for densely sampled neighboring fragments on a smooth surface. However, due to noise and discretization errors, none of them is reliable on its own. Additionally, $\epsilon_u v$ is the difference in texture coordinates, which may be useful if a low-distortion surface parametrization is available. In our applications, we have achieved best results with a weighted average error with $w_z = w_n$ and $w_{uv} = 0$. With this choice of parameters, normals pointing in opposite directions have the same weight as fragments lying on opposite sides of the rendered object. For objects with multiple connected components, we additionally store the component ID for each fragment and set ϵ to infinity if the two fragments belong to different components.

Note that our algorithm for the neighbor search can potentially introduce errors. The first type of error results from the loss of geometric information during the rasterization: if there is a sub-pixel size surface gap between two neighboring fragments, Algorithm 2 will still connect the fragments as if the surface was continuous. The second type of error is introduced by our heuristic measure. For very noisy surfaces, the measure could classify the wrong fragment pair as being closest along the surface. In this case, other measures can be used, such as the fragment distance in parameter space. This measure is particularly interesting for integral surfaces, which often have a natural global parametrization. In practice, we have found our neighbor search to be robust enough for the practical use, demonstrated by the absence of artifacts or flickering in animated scenes.

7.4 OPERATORS

After the neighbor search, the illustration buffer contains a complete and general screen-space representation of the rendered surface, where each fragment has access to its geodesic neighbors along the surface as well as its neighboring fragments along the viewing ray. This representation can be processed by any number of pointwise, local, or global operators, as discussed below. The choice of operators depends on the application and is discussed in Chapters 8 and 9. The following paragraphs therefore only give a definition of the three operator types.

7.4.1 *Pointwise operator*

A pointwise operator is a function that takes the fragment list of a single pixel and computes new values for some properties for each fragment in that list. In other words, this operator may only access the properties of all fragments associated with a given pixel. For example, a pointwise operator could implement angle-based transparency [53] by setting the fragment transparency to the angle between surface normal and viewing direction. Such an operator can again be implemented in a full-screen render pass with an appropriate fragment shader.

7.4.2 *Local operator*

A local operator is similar to a pointwise operator, except that it uses information from neighboring fragments, as described in Section 7.3. In other words, this operator may only access the properties of all fragments associated with a given pixel, plus the geodesic neighbors of those fragments. Since updating the fragment properties could affect the input of other concurrently processed fragments in an unpredictable way, the two fragment data buffer `fragData` and `fragData2` are used in a ping-pong computation scheme. Examples of local operators include image filters such as the Sobel operator. Local operators can also be used for iterative implementations of global operations, such as image diffusion, and can therefore be repeated for a user-specified number of iterations.

7.4.3 *Global operator*

A global operator is a function that takes as input all fragment lists in the illustration buffer. Similar to a local operator, two fragment data buffers have to be used in a ping-pong computation scheme. Since memory bandwidth and latency are the two major bottlenecks in our method, care must be taken to limit the actual amount of input data consumed by such an operator. We do not use global operators in our visualization, however, they could be useful in other methods such as a multilayered screen space global illumination.

7.4.4 *Fragment compositing*

The last step in our method takes the fragment list of each pixel and computes the final pixel color. The details of this operation are application dependent. For example, opaque rendering sets the pixel color to the color of the first fragment in the list. Another very common example is alpha blending, where the final color is a linear combination of all fragment colors. Similar to a local operator, this step is implemented in a full-screen render pass with an appropriate fragment shader. Note that having simultaneous access to all surface layers in a single shader pass enables compositing operations that can not be easily implemented using conventional rendering and depth peeling. Additionally, since depth peeling renders the whole geometry each time a surface layer is extracted, compositing operations that traverse the fragment list multiple times might result in a large rendering overhead.

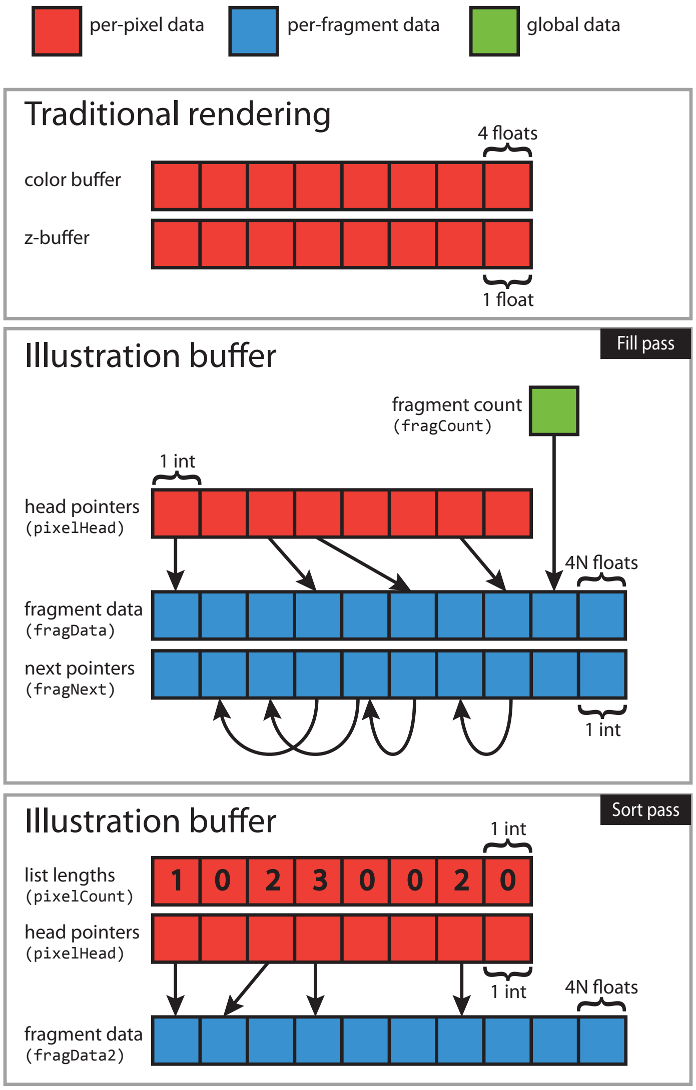


Figure 29: The illustration buffer layout. Each red cell stores data of one pixel in the image, each blue cell stores data of one fragment. Green cells are global data.

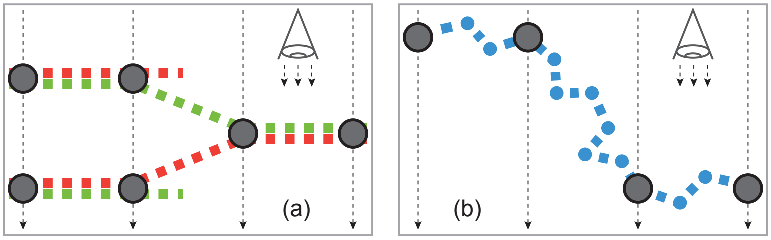


Figure 30: Four neighboring viewing rays intersect the rendered surface at fragments (black circles). (a) Given these fragments alone, one cannot decide which ones are neighbors along the surface. Two possible cross sections of the original surface are indicated in red and green. (b) For finely triangulated objects and surfaces almost parallel to the viewing direction, two neighboring fragments might be separated by an arbitrary number of polygons.

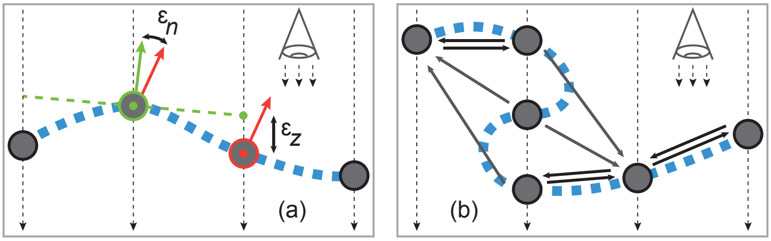


Figure 31: (a) The difference between the fragment normal and eye distance gives a measure for how likely the fragments are neighbors along a continuous surface. (b) Arrows indicate the most likely neighbor according to the continuity measure. Fragments are marked as neighbors only if this relationship is mutual.

8 SMART TRANSPARENCY FOR ILLUSTRATIVE VISUALIZATION OF COMPLEX FLOW SURFACES

The shape of a complex surface can be very hard to understand from a single image. The difficulties arise from a wide range of sources: a rendering on the screen does not have binocular depth cues, the lighting is very different from what we are used to, and the often unfamiliar shape does not allow the human visual system (HVS) to apply context knowledge as it often does for real-world objects. Examples of such complex surfaces include integral surfaces, which are an important tool to visualize the behavior of time-dependent flows, and which can contain complex twists and self-intersections. Other examples are Lagrangian coherent structures as used in flow visualization, which are often non-orientable and non-manifold, or nested technical designs for industrial prototypes created in computer aided design (CAD) applications.

For all these examples it is critical to provide users with the capability to see through all surface layers to reveal interior parts of the object. If users need to understand the whole object at once, opaque rendering is not an option. Cutting away parts of the object can be a good approach, however, for many geometries the important structures are layered in a way that a cutaway would remove important information.

Even though transparency can show more information about the object, transparent surfaces are generally more difficult to understand. The aim of this work is to develop a method for assigning transparency in illustrations that improves the understanding of transparent surfaces. In fact, this approach has been adopted by illustrators for a long time [52]. In Figure 32 we can see an example of illustrative transparency as applied in a hand-crafted illustration. In this drawing, the opacity of the outer object is high near the silhouette and decreases smoothly toward the inner part of the object in order to create a strong depth ordering cue.

The goal of illustrative visualization is not to render images in a physically correct way, but to convey information. On the other hand, the HVS is accustomed to perceive physically correct images most of the time. In this part, we do not want to create purely non-photorealistic (NPR) renderings since this rendering style tends to remove much of the visible features. For example, we are not interested in approaches that reduce the shape to a set of important lines. We believe it is important to retain the information available from shading. Based on these conditions, we discuss relevant findings from perception research, where the importance of so-called X- and T-junctions for the perception of transparency is known for a long time. Recent results were even able to find neurons with specific response patterns to a display of squares that form a X-junctions in the image [93]. Therefore, the motivation for our approach is two-fold: it is inspired by hand-drawn illustrations and also motivated by positive results from perception research.

8.0.1 *Perception*

Apart from being motivated by hand drawn illustrations, our method is also supported by the findings of percep-

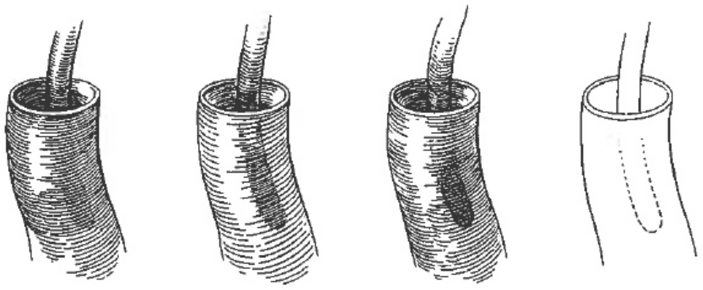


Figure 32: Example of artistic drawings. The occluded object is hidden near the edge and its visibility increases with the distance to the edge. This effect is combined with four different shading styles. ©Gerald P. Hodge

tion research. In particular, Wilson and Keil [114] explain that the perception of transparency relies, as does visual perception in general, upon context to determine the most likely interpretation. Figure 33 illustrates the importance of X- and T-junctions, which are the single most important monocular cue for transparency [114]. Albert [11] shows that also the perception of lightness is based on the relation of contours forming X- or T-junctions and that the HVS does not compute layered decompositions of luminance. Nakayame et al. [86] explain that the HVS functions effectively by employing "crude tricks" or heuristics, rather than performing a detailed and/or exhaustive interpretation of the scene with all of the information available. They show that transparency is not coupled strongly to real-world chromatic constraints since combinations of luminance and color that would be unlikely to arise in real-world scenes still give rise to the perception of transparency. From this we are motivated to apply non-local changes to the rendering of transparent surfaces even though such a configuration is unlikely to occur in reality. The approach suggested in this paper is also supported by work of Beck [18], who explains that the visual system appears to incorporate an assumption of balanced transparency. In cases of unbalanced transparency the visual system is able to choose an interpretation that in some sense is the simplest. Anderson [13] discusses the importance of perceived contours to understand depth relations in transparent surfaces. He states evidence that the HVS employs rules that combine contrast magnitude and contour continuity to decide the depth ordering of surfaces. This insight serves as an additional motivation for the contour enhancement as illustrated in Figure 33. In other words, he conjectures that the HVS treats the highest contrast portion of a contour as a region in plain view. This relationship is exaggerated by our non-local transparency approach.

8.1 METHOD OVERVIEW

From the discussed results of perception research we can draw some conclusions: First, the heuristics applied by the HVS are robust even under non-realistic conditions as long as contrast relations are kept intact. Second, X-junctions (Figure 33a) are important cues for transparency. Third, T-junctions (Figure 33b) are good for understanding which surface is above which. The strength of the depth-ordering cue provided by transparent occlusion is directly proportional to the degree of contrast reduction. In the limit case the depth-ordering cue is maximal for fully opaque surfaces, as in the case of a T-junction. Therefore, we can infer from perception research a motivation to keep the T-junction property of layered surfaces. X-junctions, on the other hand, give rise to the perception of transparency, therefore we suggest to depict crossings as a fusion of both cues (Figure 33c).

Illustrators use similar rules as we can see in Figure 32. We can see that the illustrators modify the transparency to improve the perception of layers behind. Transparency is decreased where shading is important, for example for boundary enhancement. To further improve the perception of the silhouettes, we enhance these further by not only decreasing the transparency on the boundary, but also by increasing the transparency behind a boundary such that the background becomes visible. This results in a halo-like effect (Figure 33d). The final result is illustrated in Figure 33e.

Based on the concept of XT-junctions, we design a novel *transparency enhancement method* for the rendering of layered surfaces. First, we define three transparency fields on the surface: one for the base transparency, one for the silhouette enhancement and one for the halos. The values for the two latter fields are fixed at the surface silhouette and a diffusion process is used to spread the information along the surface to the local neighborhood of the silhouettes. Finally, all three fields are combined and a modified alpha blending procedure is used to compute the final image color. The details of this method are described in Section 8.2.

The implementation of the transparency enhancement requires view-dependent and non-local information at each image point. In order to achieve interactive performance while maintaining full flexibility, we use our illustration buffer data structure (Chapter 7). The work flow of our approach is illustrated in Figure 34.

In order to objectively assess the effectiveness of our method, we designed and conducted a rigorous *user study* measuring the understanding of complex surfaces from transpar-

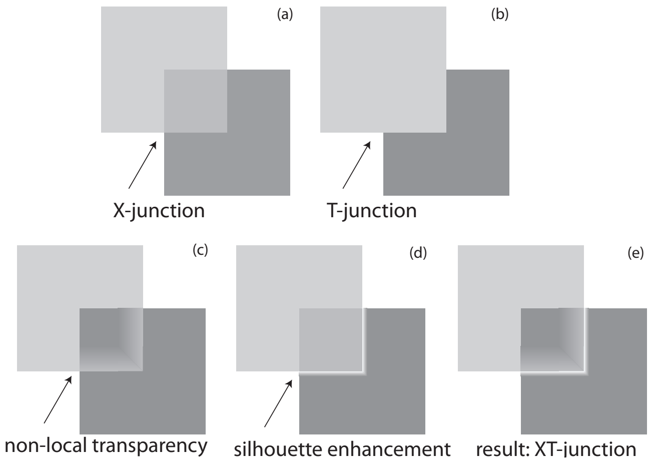


Figure 33: The perception of transparency in the HVS is crucially dependent on the distinction between X- and T-junctions. (a) X-junctions evoke the perception of transparency. (b) T-junctions are best cues for depth-ordering. (c) Non-local enhancements to improve the perception of depth-layers. (d) Silhouette enhancement is important in the case of crossings. (e) We suggest a visual cue which combines properties of X- and T-junctions.

ent renderings. In this study, three different tasks are used to quantitatively measure the task performance related to understanding of complex surfaces. The task scores are then used to compare our method to two other transparency assignment techniques. The study is described in Section 8.5.

8.2 NON-LOCAL TRANSPARENCY ENHANCEMENT

This section describes how we use the illustration buffer data structure to implement an illustrative transparency assignment method that improves the understanding of trans-

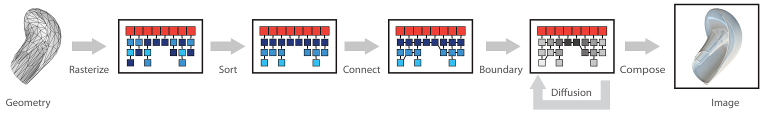


Figure 34: Overview of our method. The input geometry is first rasterized into fragments which are then stored in our illustration buffer. This 2.5D image space representation is implemented on the GPU. All fragments are then sorted by depth and connected with geodesic neighbors. In the next step, we set boundary conditions and apply a transparency diffusion process based on results from perception research. Finally, all fragments in the illustration buffer are composed into the output image.

parent surfaces. Based on the result of previous perception research, we have extracted a set of simple rules for the surface transparency:

1. The transparency of the surface is set to zero at the silhouette and increases slowly with the distance to the contour. This enhances all surface silhouettes.

2. The opacity of the surface is set to zero wherever it is occluded by another surface silhouette and increases rapidly with the distance. This will create narrow halos around occluding surfaces. The increased contrast further enhances the perception of T-junctions.

3. The transparency should be a smooth function of the surface, except at the points described above. A discontinuity could easily be mistaken for a surface contour.

The first two rules are a straightforward application of our concept of the XT-junction (see Figure 33). Note that the first rule also corresponds to the rules postulated by Hodges (see Figure 32). The third rule was chosen to prevent high frequency changes in the transparency to be mistaken for shading details or surface shape.

These rules can be implemented with our illustration buffer approach. We use a pointwise operator to set initial and boundary conditions for the transparency of individual fragments and then use a local operator to apply transparency diffusion. Similar to the solution of the heat equation after a finite time, we use the transparency diffusion with a fixed number of iterations to spread the boundary enhancement to the neighborhood of each silhouette.

8.2.1 *Transparency fields*

Because of the non-linear behavior of the transparency near junctions, we define three scalar fields on the surface:

- The initial transparency of the surface α
- A silhouette highlight field β, used to implement rule 1. This field will have high values at surface silhouettes and fall off with the distance to the silhouette.
- A halo highlight field γ, used to implement rule 2. This field will have high values near occluding edges and fall off with the distance to those edges.

In order to identify boundary fragments for each of the above fields, we also store binary fields b_α, b_β and b_γ with values of 0 for boundary fragments and 1 for all other fragments.

The values of the three fields α, β, and γ are stored as custom fragment properties (see Section 7.1). Since the scalar fields only take values from zero to one and b is a binary value, we can store both values in one scalar with minimal precision loss, e.g., as $2 \cdot b_\alpha + \alpha$.

8.2.2 *Initial conditions*

All fragments are first initialized with $\beta = 0$, $\gamma = 0$, $b_\alpha = 1$, $b_\beta = 1$, and $b_\gamma = 1$. For the initial value of α, we let the user choose among two different styles, defined by one of the following equations, where i is the index of the fragment in the sorted fragment list, n the number of fragments at the given pixel, and $s \in [0, 1]$ a user-specified parameter:

$$\alpha_i = s \tag{16}$$

$$\alpha_i = 1 - (1 - s)^{\frac{1}{n}} \tag{17}$$

Equation 16 assigns a constant initial transparency to each fragment and therefore produces results most similar to traditional alpha blending. A disadvantage of this approach is that regions with many layers will appear too opaque, while regions with only a few layers will appear washed out. Therefore, we additionally provide an adaptive transparency defined by Equation 17. This approach initializes the transparency so that the accumulated color intensity in the final image remains constant regardless of the number of layers, i.e., it solves the following problem:

$$\sum_{i=1}^{n} (1 - \alpha)^{i-1} \alpha = s \tag{18}$$

8.2.3 *Boundary conditions*

After the initial transparency values are set, a local operator classifies the fragments to find fragments at the boundary of the surface. Since this operator does not depend on the output of the previous step, it can be implemented in the same rendering pass as the operator that sets the initial conditions. The boundary conditions are set according to the following rules, illustrated in Figure 35:

- If the fragment has fewer than four neighbors, it is a boundary fragment (red cell). Set $\beta = 1$ and $b_\beta = 0$.
- If the fragment layer index is smaller than the index of one of its neighbors, it is adjacent to a silhouette fragment (blue cell). Set $\gamma = 1$ and $b_\gamma = 0$.
- If the fragment layer index is greater than the index of one of its neighbors, it lies directly underneath some silhouette fragment (green cell). Set $\beta = 0$ and $b_\beta = 0$.
- Otherwise, it is an ordinary fragment (gray cells). No changes to the initial values are made in this case.

These boundary conditions are required for the subsequent diffusion step.

8.2.4 *Transparency diffusion*

In order to smooth the initial transparency α according to rule 3, we apply a homogeneous diffusion defined by

$$\frac{\partial}{\partial t}\alpha = \lambda_\alpha \Delta \alpha \tag{19}$$

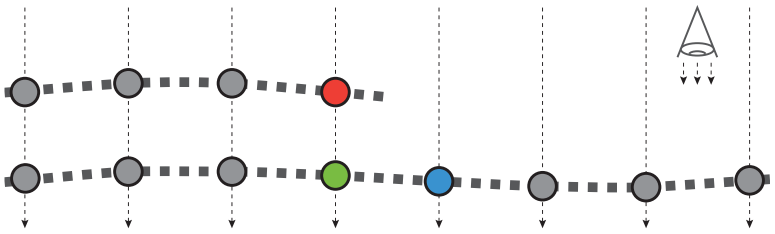


Figure 35: Cross-section of the rendered surface. Boundary conditions for the transparency are set at the red, green and blue fragments.

with a diffusion coefficient λ_α. We do not reach the steady state of the diffusion process, but instead stop after a given time. The diffusion process is implemented as a local operator using a forward discretization of Equation 19:

$$\alpha^{k+1} = \alpha^k + b_\alpha \lambda_\alpha \Delta \alpha^k \tag{20}$$

with a 2nd order central finite difference approximation of $\Delta\alpha$. In our default configuration, we use 10 to 50 iterations with a diffusion coefficient of $\lambda_\alpha = 1$.

The same diffusion process could be used to spread the initial values of β and γ to the neighborhood of silhouettes and occluding edges. However, the homogeneous diffusion transports values very inefficiently. Instead, we use a non-physical process

$$\beta^{k+1} = \max\{\beta^k, \max\{\beta^k_{\mathtt{neighbor}}\} - \lambda_u\} \tag{21}$$

where $\beta_{\mathtt{neighbor}}$ are the values of β for the neighboring fragments. The same process is used for γ. Using this procedure, the values of $1-\beta$ and $1-\gamma$ will be approximately proportional to the distance to the nearest boundary.

After the diffusion process, transparency fields β and γ will have a characteristic profile with high values near a boundary and values falling off with the distance to the boundary. For artistic reasons, we apply one of the following functions to the transparency fields in order to modify this profile:

$$u \leftarrow \frac{1}{1-e^{-p}}\left[e^{-p(1-u)^2} - e^{-p}\right] \tag{22}$$

$$u \leftarrow u^p \tag{23}$$

where u is a transparency field and p a user-defined parameter.

For $p < 1$, the apparent width of the silhouette highlight and halos increases, while for $p > 1$, the apparent width decreases. Note that both functions are bijective on the range $[0, 1]$. Figure 36 shows results of different configurations of parameters described in this section.

8.2.5 *Fragment compositing*

The three transparency fields α, β, and γ are combined in the final stage using a modified front-to-back alpha blending procedure. In this procedure, the transparency of a fragment is first initialized with α. The edge highlight field β is then multiplied with the remaining transparency $1-\alpha$ and added to the base value. Finally, the transparency is multiplied with $1-\gamma$ in order to account for halos. Note that since all input fields are smooth and we do not produce values outside of the acceptable range $[0, 1]$, the resulting fragment transparency will be smooth along the surface.

Algorithm 3 illustrates our blending procedure, where n is the number of fragments for this pixel, $\mathbf{c}$ is the final pixel color, $\mathbf{c}_i$ is the color of i-th fragment and α_i, β_i, and γ_i are the values of the three fields as defined above. A close-up

Algorithm 3 Calculate the final pixel color

```
1: $\alpha \leftarrow 1$
2: $\mathbf{c} \leftarrow 0$
3: for $i \in [1..n]$ do
4:     $\hat{\alpha}_i \leftarrow (1 - \gamma_i)(\alpha_i + (1 - \alpha_i)\beta_i)$
5:     $\mathbf{c} \leftarrow \mathbf{c} + \alpha\hat{\alpha}_i\mathbf{c}_i$
6:     $\alpha \leftarrow \alpha(1 - \alpha_i)$
7: end for
8: $\mathbf{c} \leftarrow \mathbf{c} + \alpha\mathbf{c}_{background}$
9: return $\mathbf{c}$
```

comparison with constant transparency alpha blending is given in Figure 37.

8.3 RESULTS

In this section we present images generated with our technique and compare our method with previous work.

Figure 38 shows a simple scene rendered with several common transparency assignment techniques [53]. While existing techniques only use local surface properties and are therefore very fast, they show various disadvantages. Opaque rendering (Figure 38a) has strong depth ordering cues, however does not show occluded objects. Constant transparency (Figure 38b) shows all occluded parts, but lacks depth ordering cues. Additionally, the transparency has to be made very high if one is to see all surface layers, which leads to a low contrast in the resulting image. Angle based transparency (Figure 38c) nicely highlights the silhouettes of curved objects, but not those of the flat cube. Similarly, normal variation transparency (Figure 38d) completely fails for the flat cube. Moreover, it is sensitive to noise and irregular tessellation due to the use of derivatives of the surface normal. Finally, our method (Figure 38e) properly highlights all silhouettes and maintains a high contrast while giving view of all surface layers.

Figures 39 and 44 show several examples from flow visualization. Such visualizations often contain complex surfaces for which one has to rely on the expressiveness of the rendering as it is difficult to apply context knowledge. Figure 39 shows a stream surface as well as an isosurface of vorticity in the simulation of two colliding vortex rings. Figures 44a and 44c show two different stream surfaces of a turbulent flow rendered with our method. A rendering of an entangled parametric surface is shown in Figure 44b.

Figure 40 shows the visualization of a jet engine. Similar to blueprint rendering [87], Figure 40a was rendered with low constant transparency and enhanced with silhouette and crease lines. Figure 40b was rendered with normal variation transparency enhanced with haloed lines for increased depth ordering cues. Finally, Figure 40c was rendered with our method. Since the density of silhouettes is very high for this model, halos have been omitted to reduce visual clutter. Instead the same feature lines as in Figure 40a have been used. Even though all three methods show the overall structure of the engine and haloed lines provide strong depth ordering queues, our method additionally features unique stylistic elements such as a more uniform image brightness and a more visible surface shading around silhouettes due to increased opacity. All feature lines in these figures have been computed in object space and rendered as alpha blended triangle strips.

8.4 EVALUATION

We have implemented our method as a custom rendering subsystem in the Visdom visualization toolkit [112]. Our implementation uses OpenGL 4.0 and its extension *GL_EXT_shader_image_load_store* for the concurrent global memory access as required for the illustration buffer. This extension is available on all consumer level hardware starting with the NVIDIA GeForce 400 and AMD Radeon 5000 series. Alternatively, OpenGL 4.2 core extensions *GL_ARB_shader_image_load_store* and *GL_ARB_shader_atomic_counters* could be used.

Figures 41 and 42 show the performance of our method in various settings. All tests were performed on a desktop computer with a NVIDIA GeForce 470 graphics card. The rendering time is broken down to the individual rendering passes *fill* (Section 7.2), *sort* (7.2.1), *connect* (7.3, 8.2.2, and 8.2.3), *solve* (8.2.4) and *compose* (8.2.5). The *clear* rendering pass is used to clear the contents of the illustration buffer. Each line represents the average time it takes to finish the respective rendering pass, including all previous passes. The measurements were performed by taking averages among 1000 images with a random camera position, so that the bounding sphere of the rendered object fills the entire image. Note that the *solve* pass corresponds to one iteration of our transparency diffusion. In practice, this pass will be repeated for a user specified number of iterations (typically 10 to 50), depending on the artistic preference and the resolution of the image. A comparison of different settings is shown in Figure 36.

Figure 41 presents the rendering time of the toroid surface as a function of the total number of fragments in the illustration buffer. The corresponding image resolutions range from 128×128 to 2048×2048. As expected from a screen-space method, our implementation scales linearly with the number of fragments in the scene. Note that for a square image, the number of fragments scales quadratically with the image width. This makes our method highly output sensitive: a 1024×1024 image with 50 iterations will take 20 times longer to compute than a 512×512 image with 10 iterations. In practice, applications should adapt the image resolution and/or the number of iterations to the desired level of interactivity. As an example, the scene from Figure 39b with a resolution of 512×512 pixels and 20 iterations runs at around 15 frames per second.

Figure 42 presents the rendering time of the toroid surface as a function of the total number of triangles in the surface mesh. The image resolution is set constant to 1024×1024 in this case. As seen from the timings of the *fill* pass, its run time has a constant component (indicated by the non-zero intercept) and increases approximately linearly with the number of triangles in the mesh. All other passes are independent of the input geometry.

For most images, the lengths of the fragment lists will vary across the image. The distribution of the list lengths depends on the shape of the input geometry and obviously influences the performance. However, we do not try an extensive evaluation of this effect since the distribution is hard to control without changing other parameters at the same time. In our examples, we have found the maximal list length (the number of passes required for depth peeling) to be usually several times higher than the average list length. In Figure 40c, the average list length across the image is 5.64 while the maximal length is 95 (34 layers of geometry and 61 layers of feature lines). A similar variation was present in the toroid surface with a mesh resolution of 500K triangles. Even though most fragment lists contain only 1 or 2 elements, the average maximal list length (averaged over 1000 random camera settings) is surprisingly as high as 8.94. This can be explained by the fact that some isolated pixels depict parts of the surface which are almost parallel to the viewing direction and contain many fragments of the very fine and slightly uneven surface mesh.

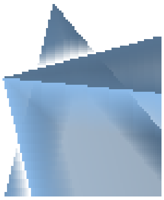
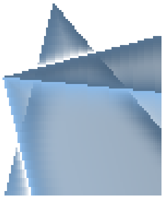
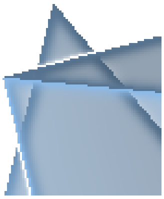
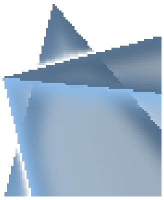
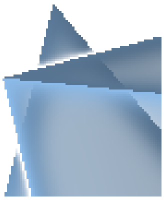
Result	Parameters
	Diff.: Equation 21 $n = 20, \lambda_\beta = 0.05, \lambda_\gamma = 0.1$
	Diff.: Equation 21 $n = 20, \lambda_\beta = 0.05, \lambda_\gamma = 0.1$ Mod.: Equation 22 $p = 3$
	Diff.: Equation 20 $n = 20, \lambda_\beta = 1, \lambda_\gamma = 0.1$
	Diff.: Equation 20 $n = 20, \lambda_\beta = 1.0, \lambda_\gamma = 0.1$ Mod.: Equation 23 $p = 0.3$
	Diff.: Equation 20 $n = 100, \lambda_\beta = 1.0, \lambda_\gamma = 0.1$

Figure 36: Different parameter configurations for the transparency diffusion. Diff is the equation used for the diffusion process, Mod is the equation used for modifying the field (if any), n is the number of iterations, and λ_β, λ_γ and p are the parameters for the diffusion, as described in Section 8.2.4

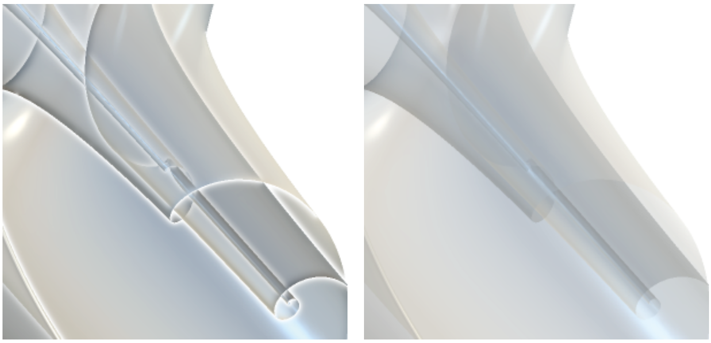


Figure 37: A close up comparison of our method (left) with constant transparency (right).

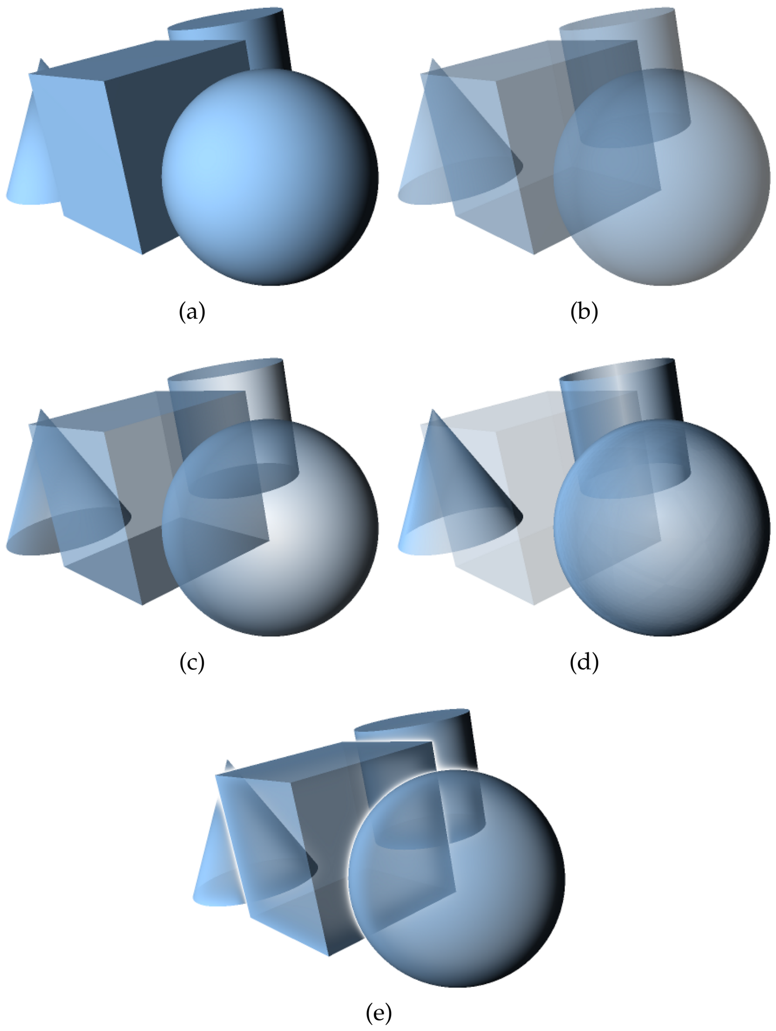
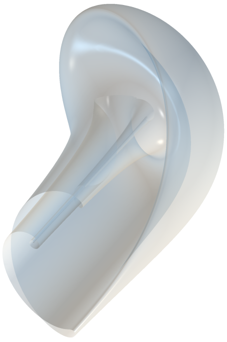
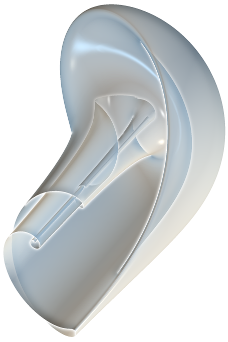
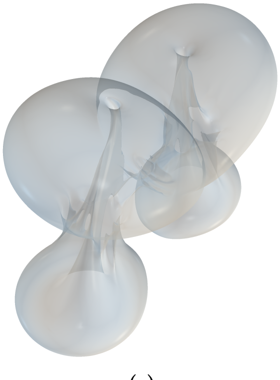


Figure 38: A comparison of different transparency assignment methods: (a) opaque rendering, (b) constant transparency, (c) angle based transparency, (d) normal variation transparency, and (e) our method.


(a)


(b)


(c)

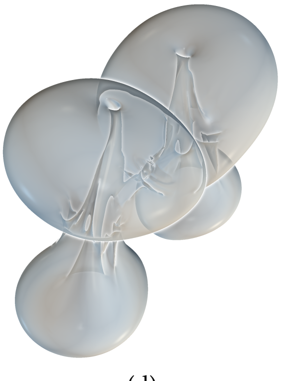
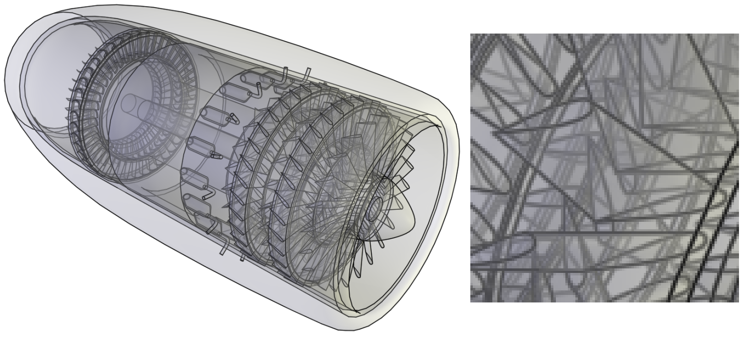
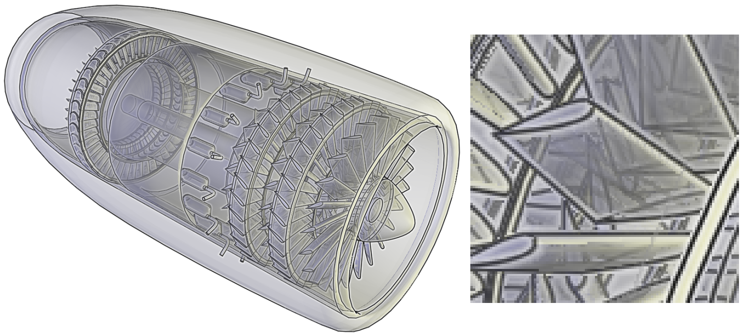

(d)

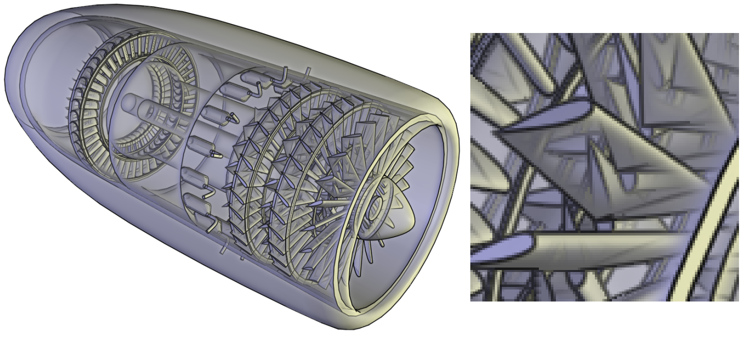
Figure 39: A stream surface and an isosurface of two different turbulent flows. (a,c) Constant transparency. (b,d) Our method. Note the improved depth ordering cues at object silhouettes as well as an adaptively increasing transparency at points with many depth layers.



(a)



(b)



(c)

Figure 40: Visualization of a jet engine model: (a) constant transparency, (b) normal variation transparency, and (c) our method. All methods have been enhanced with silhouette and crease lines.

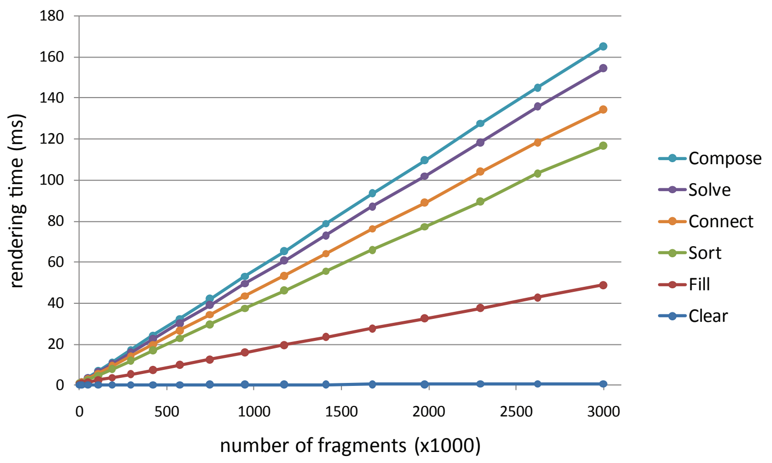


Figure 41: Rendering time vs. number of fragments. Lines depict cumulative rendering times.

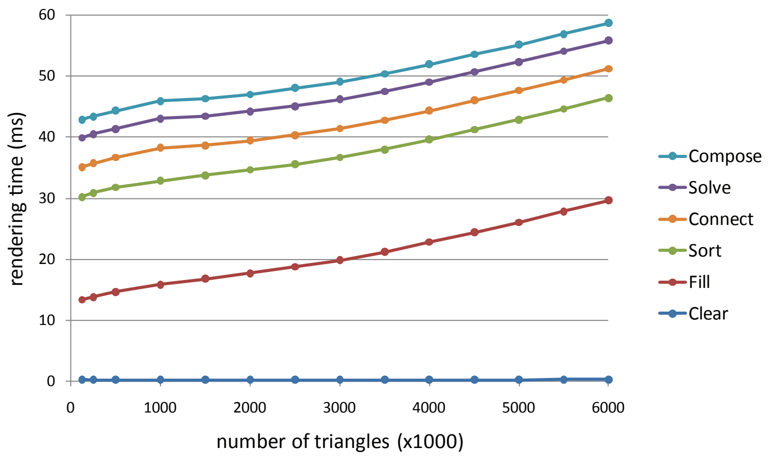


Figure 42: Rendering time vs. number of triangles. Lines depict cumulative rendering times.

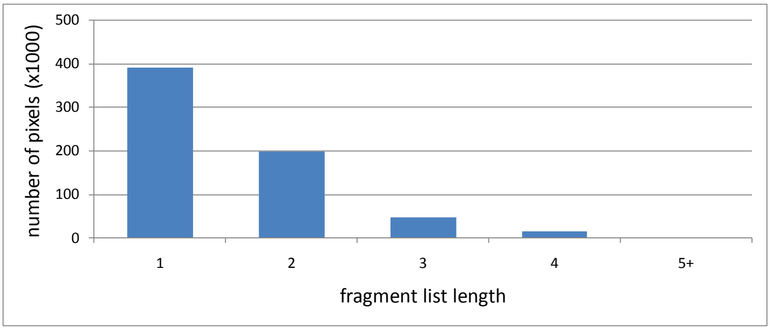


Figure 43: Average number of pixels having a given fragment list length.

8.5 USER STUDY

In this section we discuss the design and results of a user study performed to analyze the effectiveness of our illustrative transparency. The goal of this case study is to evaluate whether using the illustrative transparency presented in this work instead of alpha blending or normal variation helps with the understanding of a three-dimensional flow surface.

8.5.1 *Study Design*

One important question in our study design is how to operationalize the concept of "understanding the visualization of a surface" so as to make it measurable [15].

TASKS One problem is that other factors such as general intelligence or visual memory of the participants may constitute the main cause for variations in the test scores. Therefore, it is important to have multiple different tasks to balance such effects. In the present example we have selected the following tasks (see also Figure 44):

- Task 1 – layer index: The participant is asked to specify on which layer the surface silhouette at the given point is located. For example in Figure 44a, the correct solution for A is two, since A is on the second layer.
- Task 2 – follow the boundary: The participant is asked to follow the shortest path on the boundary of the surface from a given start point to a given end point. The participant is asked to mark all other points encountered on the way. For example in Figure 44b, the correct solution for the path from A to B is Y.

- Task 3 – nearest point: The participant is asked to specify which point is geodesically closest to another given point. For example in Figure 44c, the nearest point to A is X.

These tasks were chosen among a set of candidate tasks that test the participants ability to decide the depth ordering of surface layers or the ability to follow the surface. Criteria for selection were a precise yet simple definition in a paper-based study, as well as a balance of tasks that require a local or global understanding of the surface shape. Due to the limited number of participants, we have restricted ourselves to three tasks. Of course, each of these tasks could be better solved by presenting a specialized visualization; however, this affects their value as an operationalization for the presented technique. For each task we present multiple sets of given points (chosen manually such that the task is sufficiently difficult) and calculate the score of a participant in this task as the number of correct answers.

CONSIDERATION OF CONFOUNDING FACTORS Another issue of a user study is the presence of confounding factors [21]. When multiple variables of the rendering such as transparency, texturing, coloring, and shading vary at the same time, it is impossible to understand the importance of each variable in separation. Therefore we modify only the transparency of the surfaces and keep other factors such as viewing parameters, texturing, coloring, or shading fixed.

STUDY EXECUTION To avoid surface shape training effects over the tasks, we use three different flow surfaces "Pretzel", "Bubble", and "Toroid", which are shown in Figure 44. This way each task can be performed on a different scene. The surfaces were selected among a set of candidate surfaces such their shape is sufficiently complex and not similar to each other. During the user study, tasks are in a fixed order and each task is performed for one scene and one transparency assignment method. For example, one participant receives the first task on scene "Pretzel" rendered with alpha blending, the second task on the scene "Bubble" with normal variation, and the third task on the scene "Toroid" with illustrative transparency. The order of the factors "rendering method" and "scene" is counterbalanced [21] between the subjects. Figure 45 illustrates the three transparency assignment methods.

Since time consumed to understand a complex surface is an important factor, the time for solving each task was strictly limited to two minutes. Additionally, the number of points in each task was chosen so that it is realistically impossible to answer all of them within the given time limit. Missing answers were treated the same as wrong answers.

Partipants are recruited from local university students. To estimate the required sample size, we use a power calculation using G-power [37]. For our study, the required minimum sample size is 34 to find a correlation of large effect size.

MEASURES All participants were informed on the study and signed informed consent. Participants answered a questionnaire on sociodemographic data (age, gender, school education, years of education). We assess motivation using an item that asks the participants to estimate in percent whether they are motivated to solve the task, and self-efficacy [16] using an item that expects participants to rate in percent whether they feel able to solve the tasks with regard to their academic competency. We assess figural intelligence using the subtest "Würfelaufgabe" (cube rotation) of the German Intelligenzstrukturtest IST-2000-R, a task assessing mental rotation [12]. To measure numerical intelligence we use the subtest "Zahlenreihen" (number patterns) of the IST-2000-R, a task that requires numerical reasoning. Table 5 shows mean and standard deviations of the sociodemographic data, self-efficacy, motivation, and numerical and figural intelligence of the participants.

	Mean (Standard deviation)	Range
Age (yrs)	26.6 (4.87)	19–42
Gender	32 male 4 fem.	
High school education	100 %	
Years of education	17.54 (2.85)	11–24
Numerical intelligence	8.86 (4.16)	2–20
Figural intelligence	7.86 (3.96)	
Self-efficacy	62.03% (23.46%)	20–90%
Motivation	74.78% (16.33%)	30–100%

Table 5: Means, standard deviations, and ranges of sociodemographic data in the student sample (n=36).

8.5.2 *Study Results*

The total sample includes 36 students (4 female, 32 male) from the ETH Zürich, aged between 19 and 42. The most important question for the user study is the difference in the task performance that is attributable to the different transparency assignment methods. Our starting hypothesis is that illustrative transparency allows participants to perform better than normal variation and alpha transparency. In a first step, we check the data for outliers, but all scores are within the acceptable range.

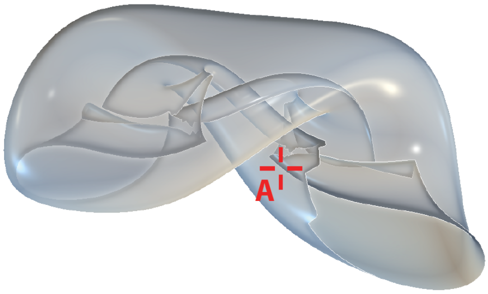
The data was analyzed using paired samples t-tests comparing the percentage of correct answers. Results of paired samples t-tests [21] show that performance of subjects that were presented with alpha blending is comparable to the performance of subjects that were presented with normal variation ($p = 0.54$). In other words, the user study does not find a significant difference between alpha blending and normal variation. On the other hand, the difference between subjects that were presented with illustrative transparency compared to normal variation is highly significant ($p = 0.005$). The performance difference between illustrative transparency and alpha blending ($p = 0.026$) is significant as well. Tables 6, 7, and 8 give an overview of these results, together with the average percentage of correct answers and its variation. We repeated the analysis of group differences using non-parametrical tests and obtained comparable results.

	Normal Variation	Alpha Blending
average	0.23379	0.26388
variance	0.03484	0.05099
degrees of freedom	68	
t-Statistics	0.616259	
$P(T < t)$ one-sided	**0.26989**	
Critical t-value	1.99547	

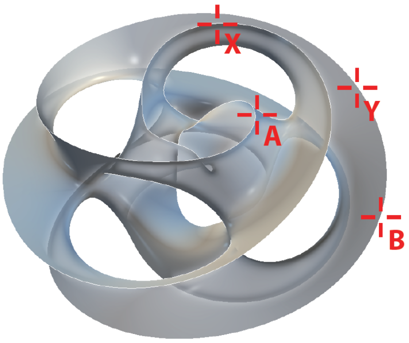
Table 6: The difference between normal variation and alpha blending is not significant.

	Alpha Blending	Illustrative Transparency
average	0.26388	0.38194
variance	0.05099	0.07674
degrees of freedom	67	
t-Statistics	-1.98196	
$P(T < t)$ one-sided	**0.025794**	
Critical t-value	1.667916	

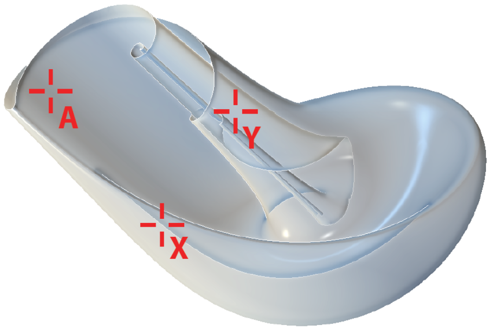
Table 7: Scores for illustrative transparency are significantly better than for alpha blending.



(a)

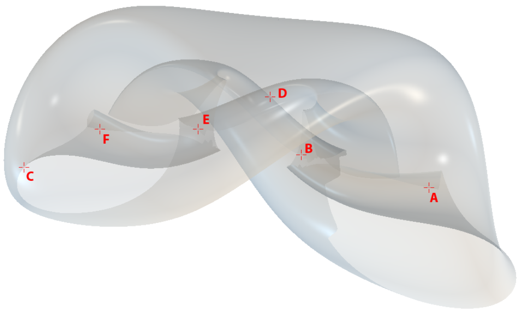


(b)

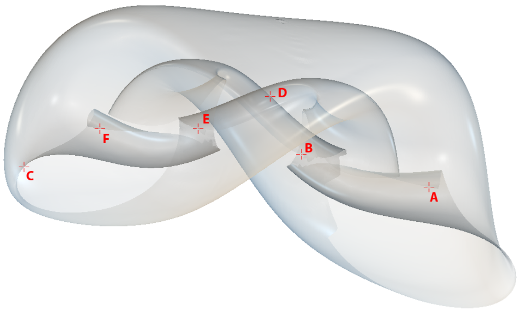


(c)

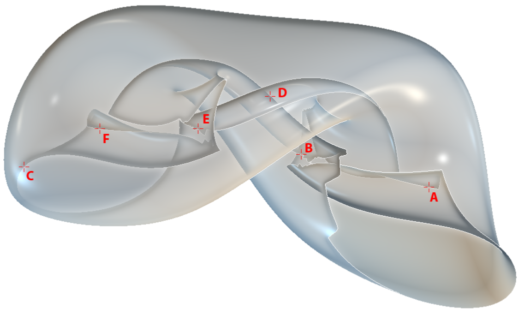
Figure 44: Examples of the three tasks of the user study. (a) Task "layer index" with the "Pretzel" surface. (b) Task "follow the boundary" with the "Toroid" surface. (c) Task "nearest point" with the "Bubble" surface.



(a)



(b)



(c)

Figure 45: Rendering methods used in the user study. (a) Constant transparency. (b) Normal variation transparency. (c) Our method

	Normal Variation	Illustrative Transparency
average	0.23379	0.38194
variance	0.03484	0.07674
degrees of freedom	61	
t-Statistics	-2.66099	
$P(T < t)$ one-sided	**0.00497**	
Critical t-value	1.67021	

Table 8: Scores for illustrative transparency are significantly better than for normal variation.

9 MULTI-LAYER ILLUSTRATIVE DENSE FLOW VISUALIZATION

The previous chapter presented a method for improving the perception of complex transparent surfaces. This is useful for flow visualizations based on integral surfaces, such as the analysis of a streak surface evolving from a seed curve. There are other applications where the user is not interested in the surface itself, but rather in the flow along a given surface of interest. This can be a physical object boundary, but also a virtual surface, such a stream surface, pressure isosurface, or Lagrangian coherent structure (LCS) [46].

Restricting the visualization to a surface allows us to use 2D flow visualization methods, which do not suffer from the same clutter and occlusion problems as 3D flow visualizations. However, occlusion still has to be handled carefully, especially if the surface of interest is deformed with the flow. Twists and folds induced by turbulent flow will almost always lead to multiple self-occluding layers in the image.

One approach to solve this problem is to use cutaways and focus the visualization of the flow to a specific layer or part of the surface. This, however, requires tedious user interaction for moving the cutaway primitives or manipulating degree-of-interest functions. More importantly, cutting away large parts reduces the usefulness of the method for an explorative analysis of the data set without a priori knowledge of what the user is interested in. Another approach is to sparsely place arrow glyphs on the surface, indicating the local flow direction. If the glyphs are sparse enough and the surfaces transparent, all layers will be visible, but the glyphs will cause difficulties with depth perception. Additionally, finding an optimal placement for the glyphs is not trivial and even then, important features in between the glyphs may be missed.

For 2D flows and single layer surfaces, this problem has been solved by dense vector field visualization. Several of these methods could be used for multi-layered surfaces, but require a surface parametrization or the knowledge of the entire vector field. In particular, the first requirement is not always given: LCS surfaces are often topologically complex and non-orientable, and a global parametrization might not always exist.

We therefore propose a novel dense visualization method that addresses the aforementioned shortcomings. The method is designed for visualizing flows along multi-layered surfaces, where both the shape of the surface and the flow direction at all layers is important. In contrast to previous work, it only requires as input a triangle mesh with the vector field values specified at the mesh vertices. The mesh does not need to have a parametrization and both the mesh and the vector field can undergo arbitrary changes over time. This format is suitable for portable data sets and readily available in many visualization frameworks. Moreover, the method is output sensitive, i.e., its run time depends mainly on the size and quality of the output. Additionally, it is not limited to a fixed view point and produces patterns of a constant screen space frequency, regardless of the current view or zooming factor.

9.1 METHOD OVERVIEW

Our method targets the visualization of flow along multi-layered surfaces. Our general approach consists of the following steps, illustrated in Figure 46.

1. A 2.5D screen space representation of the surface is built. We use the *illustration buffer* data structure for this purpose, as described in Chapter 7 All subsequent operations are performed on this representation.

2. A procedural noise function is used to assign a grayscale color value to each surface point. The function depends on both view-dependent and view-independent properties, such that the noise pattern is coherent under view changes and has a configurable, constant screen space frequency. It is described in Section 9.1.1.

3. Anisotropic diffusion is performed simultaneously on all surface layers. The diffusion process is guided by the vector field, resulting in the surface being covered by grayscale streaks that are aligned to local streamlines. A suitable finite difference discretization of this process is presented in Section 9.1.2.

4. The surface color is converted into transparency, and surface layers are composited using alpha blending. This final step is described in Section 9.1.4.

The illustration buffer is filled in a first step by rendering the surface with a fragment shader that inserts data into the illustration buffer data structure. Both the noise generation and diffusion process are then implemented by rendering a full screen quad with a fragment shader that

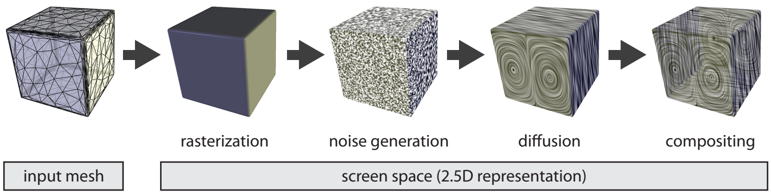


Figure 46: Overview of our method. All steps are performed on a 2.5D screen space representation of the surface.

iterates through all fragments for that pixel. Note that we do not need a special treatment for pixels with multiple surface layers. Whenever a fragment at pixel coordinates (x, y) needs to access the data from its neighboring fragment at $(x+1, y)$, it simply follows its right neighbor pointer. For the sake of simplicity, we do not introduce a special notation for following fragment neighbor pointers and instead use their pixel coordinates (x, y) and $(x \pm 1, y \pm 1)$ in all formulas in this chapter.

The illustration buffer is a very flexible data structure and can be used to extend our method with various illustrative effects, as described in Section 9.1.5.

9.1.1 *Noise generation*

In order to get an initial noise pattern that is invariant to camera rotation, we want a noise that “sticks to the surface”. In our case we use a procedural noise function that maps the world coordinates of each surface point to a noise value $\rho^0 \in [-1, 1]$, using the basic approach

$$\rho^0(\mathbf{x}_w) = \texttt{snoise3}(s \cdot \mathbf{x}_w) \tag{24}$$

where `snoise3` is the three-dimensional simplex noise function [90], $\mathbf{x}_w$ the world coordinates, and s a parameter that scales the input coordinates. This function has a number of useful properties: it is reasonably fast, C^2 continuous, visually isotropic, and produces spot noise like patterns, where the size of the spots depends on s. An additional advantage is that it can be extended to arbitrary dimensions. For example, a fourth parameter can be used to include time and create continuously animated noise patterns.

The above approach has the disadvantage that for a constant s, the noise is uniquely determined by the world coordinates of the surface and thus the screen space frequency of the noise pattern changes when zooming in and out. This is undesirable and can lead to aliasing. To fix this, we would like to compute for each pixel the optimal value of s, such that the screen space frequency of the noise is constant. We first compute the ratio of the area of one pixel in screen space to its projected area in world space using partial derivatives of the surface world coordinates, which are provided by the GPU. Then, since the frequency of the local maxima for simplex noise is equal to one per spatial unit, we set s to one times the square root of the inverse area ratio. This value will result in a constant screen space frequency of one white spot per pixel, which can then further be scaled to the desired frequency. Note that this method of computing s reduces the noise frequency for surface parts that are almost parallel to the view direction and thus prevents aliasing of the noise pattern. Depending on the application requirements, other methods for computing s might be more appropriate, such as methods based on the surface distance to the camera.

Unfortunately, using the optimal value for s makes the initial noise highly view dependent and would lead to a noise pattern that moves along the surface as the view is zoomed in or out. In order to retain coherency under changing views, we instead define a fixed set of discrete scaling values s_l with $s_{l+1} = 2s_l$. Then, for each pixel we first compute the optimal value for s, choose the two closest scaling values s_l and s_{l+1}, and linearly blend the two corresponding noise patterns ρ_l^0 and ρ_{l+1}^0. This approach is similar to traditional mipmapping and trilinear filtering [49].

When blending two noise images, we should make sure that the result has a similar appearance as a single noise image. Figures 47a and 47b show the noise patterns for different values of s. By analyzing the histogram of the images, we can see that the noise values for different s follow approximately the same distribution with some unknown variance σ^2. Assuming noise values for different s are unrelated, a linear combination of noise images $\rho^0 = (1-\alpha)\rho_l^0 + \alpha\rho_{l+1}^0$ will have the same distribution with a reduced variance $((1-\alpha)^2 + \alpha^2)\sigma^2$. In order to maintain a constant contrast, we scale the blended noise by $1/\sqrt{(1-\alpha)^2+\alpha^2}$. The effect of this scaling is demonstrated in Figures 47c and 47d.

The whole algorithm for the initial noise is summarized in Algorithm 4, where `log2` and `exp2` are the logarithm and exponential to the base of 2, `dFdx` and `dFdy` derivatives with respect to screen coordinates, and `clamp` a function that clamps its first argument to the range given by the last two arguments. The parameter *spotsize* is the desired approximate diameter of the noise spots, given in screen pixels. Note that the initial noise depends on the world coordinates of each fragment. These coordinates are stored as per-fragment data when filling the illustration buffer. All subsequent operations operate in screen space and therefore only use the screen space coordinates of the fragments.

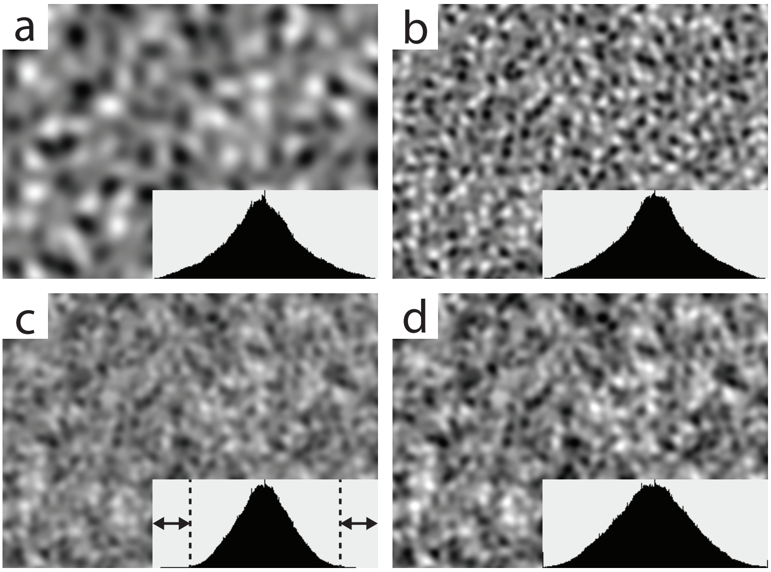


Figure 47: Simplex noise for $\mathbf{x} \in [-1, 1]^2$. (a) Noise for $s = 5$. (b) Noise for $s = 10$. (c) The average of the two above images. (d) The average with applied contrast correction.

9.1.2 *Anisotropic diffusion*

Similar to many approaches in dense vector field visualization, we "smear" the initial noise along the vector field, so that the color of neighboring pixels along a streamline is correlated, while the color of neighboring streamlines is not. Since our 2.5D image representation contains multiple layers and only allows access to neighboring fragments, we formulate our smearing process as an anisotropic diffusion where each fragment only exchanges data with neighboring fragments. Since we are interested in the flow along the surface, we first project the original vector field onto the surface and then transform it to screen space. The result-

Algorithm 4 Computing the initial noise.

```
1: $s \leftarrow 1/(\text{spotsize} \cdot \sqrt{\|\texttt{dFdx}(\mathbf{x}) \times \texttt{dFdy}(\mathbf{x})\|})$
2: $ls \leftarrow \lfloor \texttt{log2}(s) \rfloor$
3: $s_1 \leftarrow \texttt{exp2}(ls)$
4: $s_2 \leftarrow \texttt{exp2}(ls + 1)$
5: $\rho_1 \leftarrow \texttt{snoise3}(s_1 \mathbf{x}_w)$
6: $\rho_2 \leftarrow \texttt{snoise3}(s_2 \mathbf{x}_w)$
7: $\alpha \leftarrow (s - s_1)/(s_2 - s_1)$
8: $c \leftarrow 1/\texttt{sqrt}(\alpha^2 + (1-\alpha)^2)$
9: **return** $\texttt{clamp}(c((1-\alpha)\rho_1 + \alpha\rho_2), 0, 1)$
```

ing vector field **u** is used to guide the anisotropic diffusion, given by the partial differential equation

$$\frac{\partial \rho}{\partial t} = \nabla \cdot (\mathbf{A} \nabla \rho) \tag{25}$$

with the initial and boundary conditions

$$\rho(\mathbf{x}, 0) = \rho^0(\mathbf{x}), \quad \mathbf{A}\nabla\rho \cdot \mathbf{n} = 0 \text{ on } \partial\Omega \tag{26}$$

where **A** is a diffusion tensor and **n** the screen space normal to the surface silhouette $\partial\Omega$. Expressing the diffusion tensor in terms of diffusivity along and perpendicular to the local vector field leads to

$$\mathbf{A}(\mathbf{u}) = \begin{pmatrix} u & \hat{u} \\ v & \hat{v} \end{pmatrix} \begin{pmatrix} a & 0 \\ 0 & \hat{a} \end{pmatrix} \begin{pmatrix} u & v \\ \hat{u} & \hat{v} \end{pmatrix} \tag{27}$$

where $\mathbf{u} = (u, v)^T$ is the local flow direction, $\hat{\mathbf{u}} = (\hat{u}, \hat{v})^T$ a vector normal to **u**, a the diffusivity along the flow direction, and $\hat{a}$ the diffusivity perpendicular to the flow direction. Using the above decomposition, Equation 25 can also be written as

$$\frac{\partial \rho}{\partial t} = \nabla \cdot [a\mathbf{u}(\mathbf{u} \cdot \nabla\rho) + \hat{a}\hat{\mathbf{u}}(\hat{\mathbf{u}} \cdot \nabla\rho)] \tag{28}$$

where $\mathbf{u}\,(\mathbf{u}\cdot\nabla\rho)$ is the projection of the gradient onto the local vector field direction. In our applications, we set $\hat{a} = 0$ in order to maintain a maximal contrast between streamlines.

We discretize Equation 28 using finite differences, more precisely, using second order approximations for the gradient and the divergence, and an explicit forward time stepping scheme. In order to limit the stencil of the spatial discretization to the four neighboring fragments, we first compute for each fragment the discrete gradient of ρ, multiply it with the diffusion tensor, and store it inside the fragment. In the next iteration, we compute the discrete divergence of these gradient vectors and use it to update ρ using the following explicit forward time scheme

$$\begin{pmatrix} g \\ h \end{pmatrix}_{x,y}^{k} = a\mathbf{u}\left[\mathbf{u}\cdot\begin{pmatrix} D_x(\rho)_{x,y}^{k} \\ D_y(\rho)_{x,y}^{k} \end{pmatrix}\right] \tag{29}$$

$$\rho_{i,j}^{k+1} = \rho_{x,y}^{k} + \Delta t\left[D_x(g)_{x,y}^{k} + D_y(h)_{x,y}^{k}\right] \tag{30}$$

where $\rho_{x,y}^{k}$ is the value of ρ at position (x, y) and time step k, $(g, h)^T$ the gradient field, and D_x and D_y the discrete derivatives given by the central difference operators

$$D_x(f)_{x,y}^{k} = \frac{1}{2}\left(f_{x+1,y}^{k} - f_{x-1,y}^{k}\right) \tag{31}$$

$$D_y(f)_{x,y}^{k} = \frac{1}{2}\left(f_{x,y+1}^{k} - f_{x,y-1}^{k}\right) \tag{32}$$

A necessary condition for the stability of this scheme is given by the Courant-Friedrichs-Lewy (CFL) condition $\Delta t \leqslant \|\mathbf{u}\|^2 / a$ [106]. In order to guarantee a fast convergence, we normalize the vector field and use $a = 1$ and $\Delta t = 0.95$. Using the non-normalized field would require limiting the time step to satisfy the CFL condition, leading to slow convergence, or using a more sophisticated discretization method that does not have such a condition. Note that Equation 30 depends on gradients computed by Equation 29. This is implemented by executing both equations in separate render passes. One may be tempted to instead execute both equations in one pass and use the gradients from the previous time step. This will however halve the maximal stable time step, thus bringing no real improvement.

Instead of central differences, other discretizations could be used. Figure 48a shows the result of a full 3x3 stencil, as used in [98]. This operator is more precise at the cost of accessing diagonal elements, which is expensive in the illustration buffer, as it requires two indirect loading operations (e.g., follow the left neighbor of the upper neighbor to get the upper left diagonal element). Figure 48b shows the result of first order forward differences for the gradient and backward differences for the divergence. Using this scheme, Equations 29 and 30 can be implemented in a single pass, however this scheme suffers from severe numerical diffusion. The central differences, shown in Figure 48c, are a good compromise between speed and quality. An analysis of similar five point schemes [102] revealed that central differences have the lowest numerical diffusivity at the cost of checkerboard patterns appearing near large gradients. A checkerboard has a constant zero gradient for the central differences operator and thus will not be removed by the diffusion process. Fortunately, our input noise is very smooth. For smooth vector fields and spot sizes of at least four pixels, these patterns are usually not noticeable. If necessary, more sophisticated schemes as presented in [102] could be used to avoid this problem.

A very important aspect of the discretization is the correct handling of boundary conditions. Since the computational domain Ω is the set of all connected fragments, we

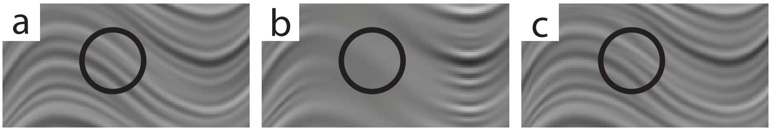


Figure 48: Discretizations of the gradient. (a) Full 3x3 stencil. (b) Forward and backward differences. (c) Central differences.

first have to find boundary fragments and estimate the domain normal $\mathbf{n}$. We use the following numerical approximation for the outer normal

$$\mathbf{n} = -\sum_{\Delta\mathbf{u}\in\mathcal{N}} \Delta\mathbf{u} \tag{33}$$

where $\mathcal{N} \subseteq \{(-1,0),(1,0),(0,-1),(0,1)\}$ is the set of offsets for which pixel (i,j) contains valid neighbors. This is illustrated in Figure 49a, where the red cells are fragments outside of the surface and the blue cells are fragments inside the surface. Note that this approach does not work for degenerate fragments where the normal is not uniquely defined, which we ignore since such fragments will already have zero gradient across the domain boundary.

When accessing fragments outside of the domain in Equations 31 and 32, we use reflecting boundary conditions. These conditions state that the function is virtually reflected at the boundary, i.e., $f_R^k = f_L^k$ and $D_x(f)_R^k = -D_x(f)_L^k$ in Figure 49a. This guarantees $\nabla\rho \cdot \mathbf{n} = 0$, but not $\mathbf{D}\nabla\rho \cdot \mathbf{n} = 0$. From Equation 28 we see that at a boundary, the projected gradient must not have a component parallel to $\mathbf{n}$. We enforce this condition by modifying $\mathbf{u}$ at the boundary so that it is perpendicular to $\mathbf{n}$. We have found this approach to be more stable than using first order differences at the boundary and removing the normal component from the flux $\mathbf{D}\nabla\rho$.

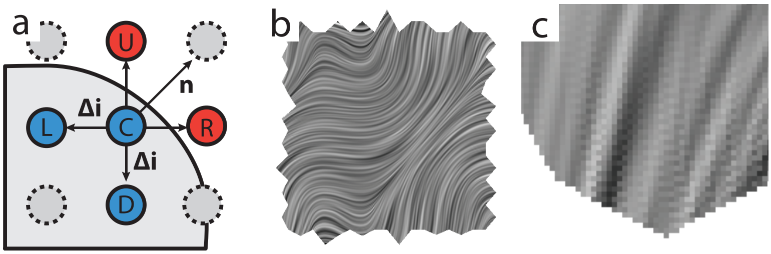


Figure 49: Boundary conditions. (a) Estimating the boundary normal. (b) 2000 diffusion iterations with a complex surface boundary. (c) Checkerboard pattern near boundaries.

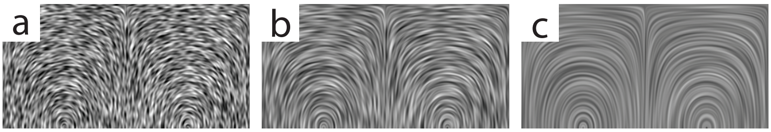


Figure 50: Number of iterations on a 512×256 pixel image. (a) 20 iterations. (b) 100 iterations. (c) 1000 iterations.

Figure 49b demonstrates that the diffusion produces correct results for arbitrary boundaries. The only numerical difficulties we have encountered are checkerboard patterns near boundaries, as shown in Figure 49c. We have not found them to be disturbing enough to justify more sophisticated discretizations as explained above.

The diffusion process is repeated for a number of iterations chosen by the user. Higher numbers will result in longer streaks and generally a higher image quality, however increase the run time proportionally. A comparison of different diffusion times is shown in Figure 50. In practice, the number of iterations should be adapted to the desired level of interactivity, i.e., decreased for interactive data exploration and increased for still images.

9.1.3 *Relation with line integral convolution*

In this section we analyze how the anisotropic diffusion is related to line integral convolution. Ignoring numerical diffusion, Equation 25 with $\hat{a} = 0$ can only transport information along streamlines and thus corresponds to a one-dimensional diffusion along a streamline. The analytic solution of the one-dimensional diffusion equation is

$$\rho(x, T) = \frac{1}{\sqrt{4\pi a T}} \int_{-\infty}^{\infty} e^{\frac{-(x-y)^2}{4aT}} \rho(y, 0) dy \tag{34}$$

which closely resembles the one-dimensional line integral convolution given by

$$\rho(x) = \int_{-L}^{L} k(x-y)\rho_0(y) dy \tag{35}$$

where $k(x)$ is the convolution kernel and L the convolution length. This comparison shows that the anisotropic diffusion after a time T corresponds to a line integral convolution with an infinite convolution length and a Gaussian kernel $k(x)$. Note that $k(x)$ is equal to the probability density function of the normal distribution with variance $\sigma^2 = 2aT$. Since 99.7% of the area of $k(x)$ lies in the range $[-3\sigma, 3\sigma]$, the convolution length can be safely limited to $L \approx 3\sqrt{2aT}$. This relation can be used to express parameters for our method in terms of parameters for existing LIC based visualizations in order to obtain similar results.

9.1.4 *Compositing*

The last stage of our method is the compositing of the illustration buffer into the final image. Our basic approach is to convert the fragment intensity $\rho \in [-1, 1]$ into the fragment transparency $\alpha \in [0, 1]$ and composite the fragments using alpha blending. We do not change the color of the fragments, leaving the choice of surface and background color to the user. This will result in the surface being covered by opaque line-like streaks, interleaved by transparent streaks. In order to facilitate the perception of the depth ordering for multiple layers, we want the streaks to be fully transparent and fully opaque, respectively, i.e., we want the image to have a high contrast. Since the diffusion process reduces the contrast of the image, we perform a simple contrast stretching

$$\alpha = \mathtt{min}(\mathtt{max}(r \cdot 0.5 \cdot \rho + 0.5, 0), 1) \tag{36}$$

with a stretching parameter r. In our applications, we use values between 2 and 4.

When illustrating multiple layered surfaces, care has to be taken when choosing the colors for the surfaces and the background. Dense vector field visualization methods only work well if there is sufficient contrast in the final image. Since the image will contain streaks from all surface layers interleaved by the background, we want the set of all used colors to contain elements with pairwise high contrast. Example choices are white surfaces on a black background, or cyan, magenta, and yellow surfaces on black background. The maximal number of layers that can be visualized greatly depends on the data. In general, a free choice of surface color and animated images increase the number of layers.

9.1.5 *Illustrative techniques*

The core of our method is the solution of a partial differential equation on a screen space representation of the surface with multiple layers. This framework is very flexible and can be used to implement a number of extensions. In order to enhance the perception of the surface shape and depth ordering, we use an illustrative method for the enhancement of surface boundaries (see Chapter 8). This effect uses a separate diffusion process to compute for each fragment its screen space distance to the nearest surface silhouette. The distance is then used to update the intensity ρ just before compositing with

$$\rho \leftarrow \rho + k_1 e^{-k_2 \beta^2} \tag{37}$$

where β is the fragment distance to the nearest silhouette and $k_1 \in [0, 2]$ and $k_2 > 0$ user-defined parameters modifying the strength and width of the illustrative enhancement. An additional advantage of the illustration buffer is that the number of surface layers is known for each pixel. We use this information to adaptively lower the density of surface streaks for image parts that contain many layers. This is implemented by lowering and rescaling the initial noise value ρ^0 with

$$\rho^0 \leftarrow \frac{\rho^0 - (k_3 + k_4 n)}{1 - (k_3 + k_4 n)} \tag{38}$$

where n is the number of layers and k_3 and k_4 user-defined parameters. The effect of these enhancements is shown in Figure 54.

9.2 RESULTS

In this section, we will present results of our method applied to three different problems: simulated atmospheric flow on an extrasolar planet, Lagrangian coherent structures in a revolving door simulation, and a stream surface of a synthetic velocity field. For higher print quality, images were rendered using a large number of iterations.

9.2.1 *Exoplanet*

In the first application, we visualize the results of a weather simulation on an extrasolar planet [51]. The planet is roughly the size of Jupiter and is tidally-locked, i.e., one side of the planet is always facing the star. Due to the physical nature of the problem, the atmospheric flow is mostly horizontal and limited to near-spherical surfaces, which were also used in the discretization of the computational domain. Therefore, a visualization of the flow along these surfaces is of great interest to the researchers.

Figure 51 illustrates the flow along two layers of the atmosphere. The lower layer, located about 2000 km above the planetary surface, is where infrared light is escaping. The upper layer, located about 7000 km above the surface, is not visible to the eye. Since the temperature is an important property in this problem, the surfaces were colored according to the temperature, with values ranging from 600K (blue) over 1300K (green) to 2000K (red). Note how the atmosphere on the left (day) side of the planet is warmer and more bulgy than the atmosphere on the right (night) side.

In this application, we have chosen a high frequency initial noise and long diffusion times, resulting in long, thin

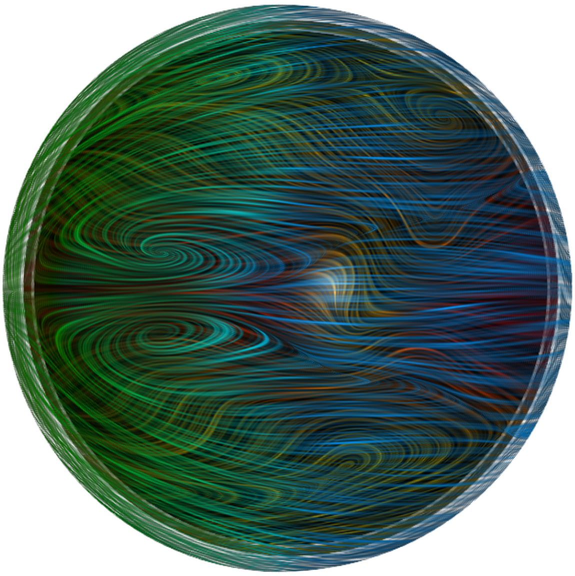


Figure 51: Two-layered dense visualization of the atmosphere of a tidally-locked exoplanet. The streaks are colored using the temperature. 800×600 pixels, 500 iterations.

streaks. In order to achieve the highest quality for these thin features, we have used the full 3×3 stencil for the gradient discretization instead of the central difference discretization that was used in all other results. The planet surface was rendered as an opaque black object in order to provide contrast for the atmosphere layers. When presented to domain experts, our results have been assessed as extremely useful because one can then get an idea of how the planetary flow is reacting to the starlight at different layers.

9.2.2 *Revolving door*

In the second application, we visualize Lagrangian coherent structures (LCS) [46] in a simulation of a revolving door [100]. In order to minimize energy losses from cold air entering through the revolving door, an air curtain was added, blowing warm air upwards from the floor behind the door. LCS have proved to be very useful in finding energy leaks in this scenario and are therefore important to visualize. Since by definition, the flux through the LCS is negligible, a visualization of the flow along these surfaces will provide physically relevant information.

Figures 52 and 53 show the LCS, where blue color denotes repelling structures and red attracting ones. The main problem with LCS-based visualizations is dealing with clutter as LCS tend to have complicated structures. In [100], cutaways were used to reveal the structure close to the air curtain, as seen in Figure 52. Even though cutting away half of the mesh does a very good job at revealing the structure of the LCS, cutting meshes is cumbersome and finding proper cut locations takes time. In Figure 53, we can see how our dense flow visualization helps understanding the global structure of the LCS even without using any cutaways. Since understanding the shape of the LCS is very important, we have chosen a sparse, low frequency initial noise and have applied an illustrative, non-local silhouette enhancement as described in Chapter 8. Note that our visualization shows only the instantaneous flow direction and care has to be taken when interpreting this information on the time-dependent LCS surfaces.

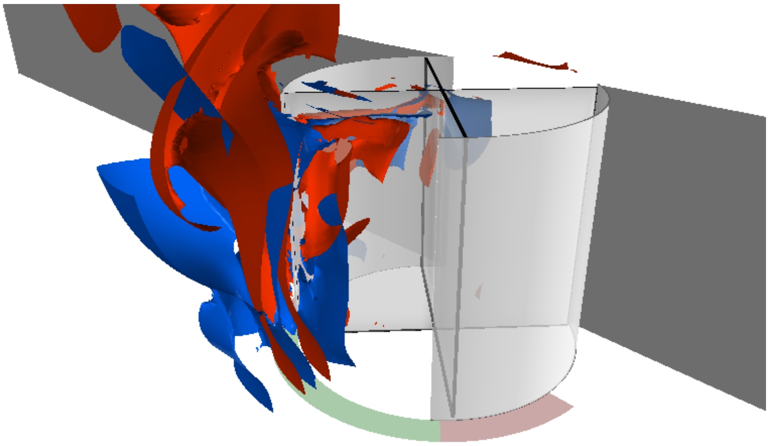


Figure 52: LCS based visualization of the simulation of a revolving door with air curtain using cutaways. The air curtain is modeled by an air outlet, as shown in green. On both sides of the air curtain, air intakes are placed to compensate for air blown in by the air outlet.

9.2.3 *Stream surface*

In the third application, we visualize a swirling flow. The vector field is an analytic modification of Hill's spherical vortex, and the surface was produced using high quality numerical integration. Since there is by definition no flow through a stream surface, and recirculating flows produce twisted and folded structures, this setting is suitable to explore the expressiveness of our visualization.

Figure 54 demonstrates four different variations of our method. In Figure 54a, the method was applied without any extensions. In Figure 54b, an illustrative cool-warm shading [42] and a non-local silhouette enhancement according to Equation 37 with $k_1 = 0.5$ and $k_2 = 30$ was added. In Figure 54c, the initial noise was adapted to the number of

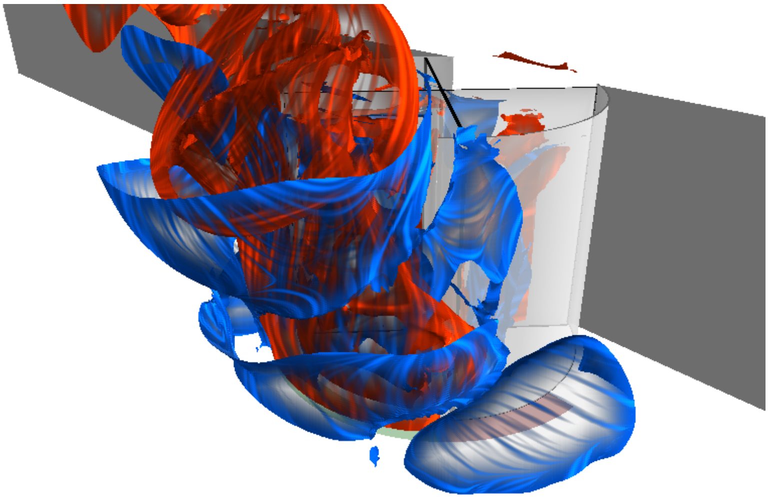


Figure 53: LCS based visualization of the simulation of a revolving door with an air curtain using our method. Even without cutaways, the general structure of the LCS is much more easily understood. 800×600 pixels, 250 iterations.

surface layers, according to Equation 38 with $k_3 = -0.4$ and $k_4 = 0.4$. In Figure 54d, the color of the surface was computed from the geodesic distance to three uniformly distributed points on the surface, resulting in a better color difference between streaks that are geodesically far apart.

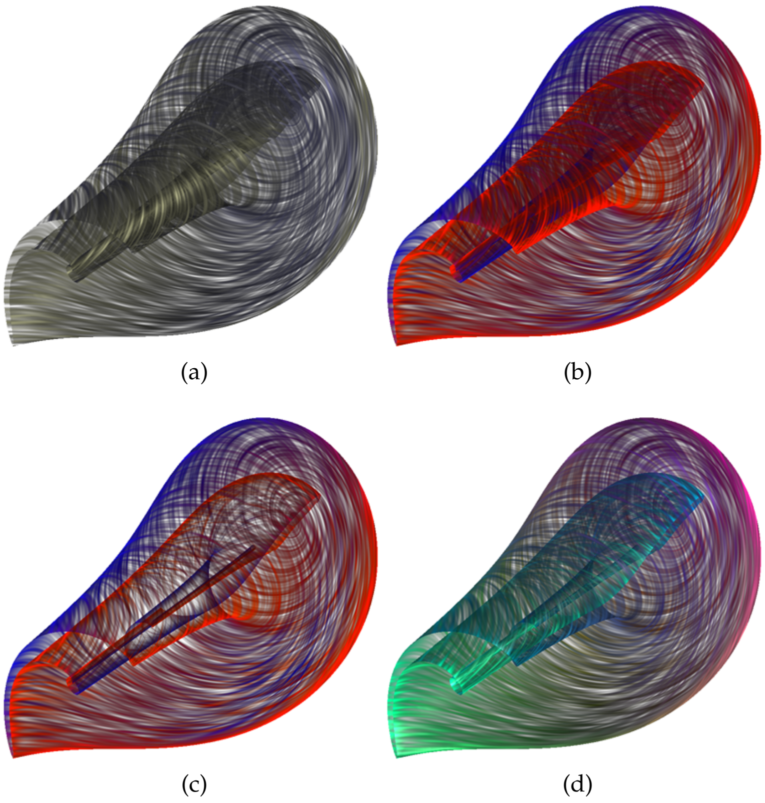


Figure 54: Visualization of the flow along a stream surface. (a) Base result. (b) Cool-warm shading and silhouette enhancement. (c) Adaptively reduced noise density. (d) Automatic surface coloring. 1024×1024 pixels, 1200 iterations.

10 CONCLUSION

In this part we have introduced the *illustration buffer*, a novel 2.5D screen space data structure which provides explicit links to neighboring fragments along the surface as well as along the viewing ray, and thus allows us to implement a wide range of visualization methods. The data structure works with arbitrary triangle meshes and does not need a surface parametrization, and can therefore be used to visualize difficult surfaces such as non-manifolds with a fast changing topology. The illustration buffer is used as a framework for presenting two novel surface visualization methods.

First, we present an illustrative transparency assignment method which is motivated from two directions. Perception research suggests that X- and T-junctions are the most important clue for the understanding of transparent surfaces. Therefore we suggest a hybrid cue which we call XT-junction. This approach is also motivated by traditional drawing methods of scientific illustrators, who use similar halos and non-local transparency modulation in their illustrations. In a user study we have shown that our approach may improve the understanding of complex surfaces.

Next, we present a dense flow visualization method that is applicable to surfaces with multiple self-occluding layers, where both the surface shape and the flow along the surface is of importance. By using both view-dependent and object space properties, the method generates patterns of constant screen space frequency configurable by an intuitive user parameter – the screen space diameter of the noise spots.

Since our methods use a non-local enhancement of surface silhouettes, they are not suitable for highly fragmented objects or surfaces with a dense set of silhouettes. This issue could be adressed by applying an anisotropic diffusion model that takes the local feature size into account. Another limitation is the use of a heuristic error measure to find geodesically neighboring fragments. Although we found it to be robust enough for practical purposes, it is possible to think of cases where it fails. A surface parametrization could be used in this case to compute the geodesic distance between fragments in parameter space. Alternatively, if a suitable surface parametrization is available, the transparency distribution could be computed in object space, eliminating the heuristic error measure. Günther et al. use this approach to combine our method with a global optimization of the initial surface transparency [45].

We employ an iterative screen space method to compute the diffusion of transparency. While this method is quite simple to implement, faster convergence could be achieved with a more sophisticated multi-scale method which takes the geometry topology and texture coordinates into account. Since the core of our method is the solution of a partial differential equation on multi-layered surfaces, other equations can easily be investigated as well. For example, our method could be extended to advection-diffusion schemes in order to visualize path lines or streak lines.

In our method we focus on transparency while keeping other important cues such as texture, lighting, or color fixed in order to allow a meaningful evaluation of transparency. We believe a decoupling of these parameters is necessary to develop a novel framework for transparency. On the other hand, this also forces us to limit the conclusions to the importance of transparency alone; we cannot make any speculations on the relative importance in relation to other properties, such as surface color (see Figure 55). An interesting future research project would be to extend our approach to a framework which takes multiple cues into account and to perform a user study which can provide insight into the relative importance of the different cues.

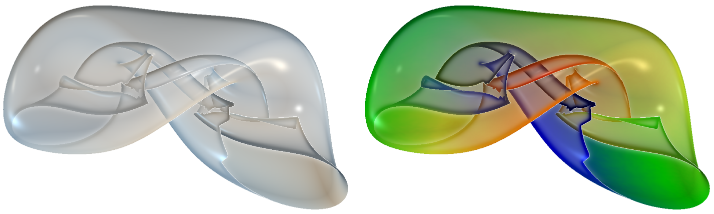


Figure 55: The perception of multi-layered surfaces could be enhanced by other visual cues, such as color or lighting.

Even though the user study used a small number of tasks, we have obtained statistically significant results. This may be due to the fact that we have used complex and difficult to understand surfaces in our tasks, since we suggest our method specifically for the visualization of complex surfaces. Additionally, our paper-based study and the choice of tasks was very general, and we do not claim that it is an optimal operationalization of the problem of understanding complex surfaces. For practical evaluations, an interactive study with application-specific tasks might be more appropriate.

Part III

APPENDIX

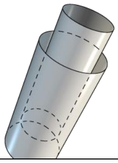
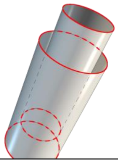
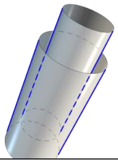
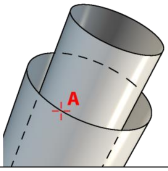
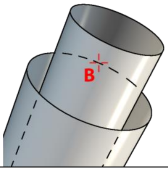
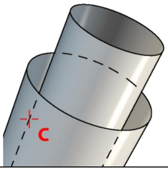
A USER STUDY

Example of the user study as described in Section 8.5, excluding the leading intelligence test tasks (see Section 8.5.1). The participants had 1:30min time to read each task description, after which they were allowed to continue to the actual task.

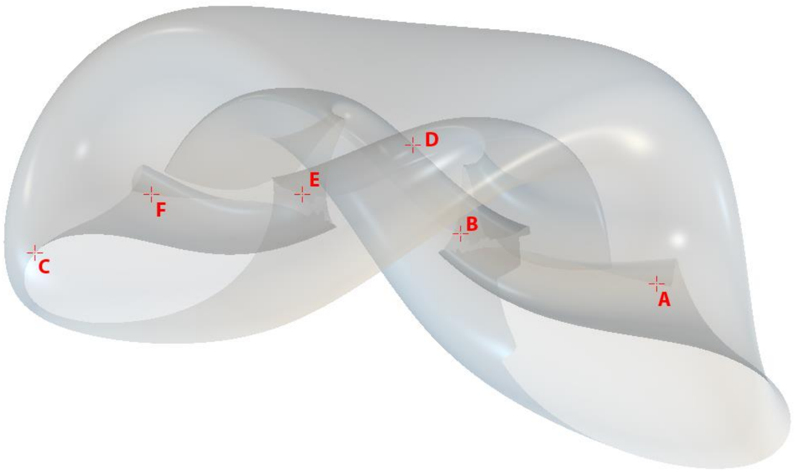
Task: on which layer is the point located

For this task you are asked to **specify on which layer a boundary or silhouette point is located**.

DEFINITIONS AND EXAMPLES

	For the examples, we will use a picture of two concentric open tubes. The actual task will follow later and will use a different style of rendering.
	Red highlighted are all surface **boundaries**. A surface boundary is where the surface ends; if the surface was a piece of sharp paper, you could cut yourself on the boundary.
	Blue highlighted are all surface **silhouettes**. Silhouettes appear as edges in the picture, but the surface does not end there.
	In this picture, point **A lies on the first surface layer**, since there are no other surfaces in front of the marked boundary.
	In this picture, point **B lies on the third surface layer**, since there are two other layers in front of the marked boundary.
	In this picture, point **C lies on the second surface layer**, since there is one surface layer in front of the marked silhouette.

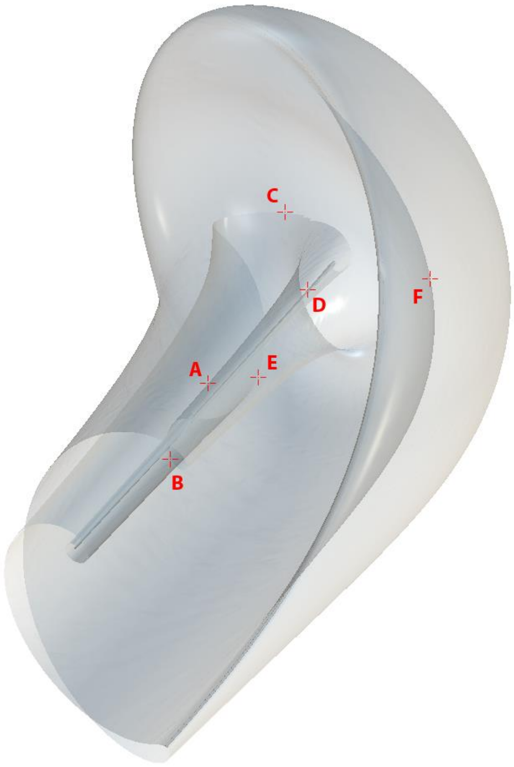
Hint: there is always exactly one boundary or silhouette at each marked position.



Task: on which layer is the point located

Please specify the layer for each point. L1 is the front-most surface layer, L2 the second layer, and so on. Cross one item per line.

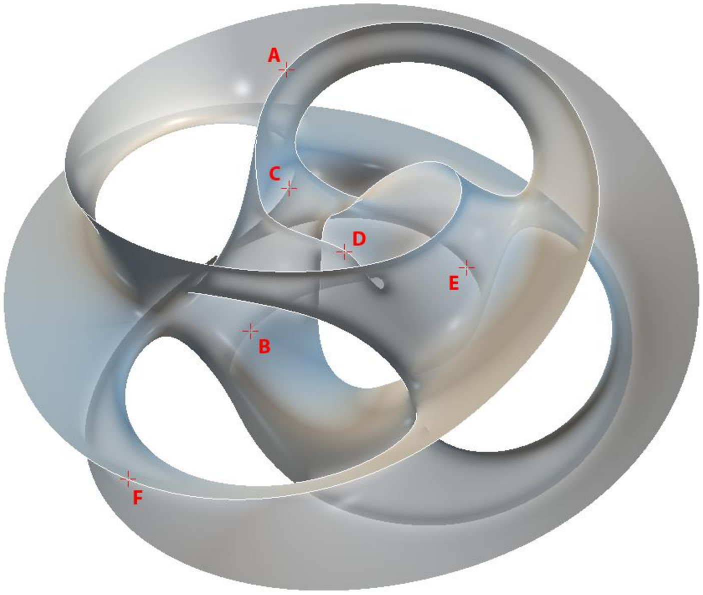
Point	L1	L2	L3	L4	L5	L6	L7
A	☐	☐	☐	☐	☐	☐	☐
B	☐	☐	☐	☐	☐	☐	☐
C	☐	☐	☐	☐	☐	☐	☐
D	☐	☐	☐	☐	☐	☐	☐
E	☐	☐	☐	☐	☐	☐	☐
F	☐	☐	☐	☐	☐	☐	☐



Task: on which layer is the point located

Please specify the layer for each point. L1 is the front-most surface layer, L2 the second layer, and so on. Cross one item per line.

Point	L1	L2	L3	L4	L5	L6	L7
A	☐	☐	☐	☐	☐	☐	☐
B	☐	☐	☐	☐	☐	☐	☐
C	☐	☐	☐	☐	☐	☐	☐
D	☐	☐	☐	☐	☐	☐	☐
E	☐	☐	☐	☐	☐	☐	☐
F	☐	☐	☐	☐	☐	☐	☐



Task: on which layer is the point located

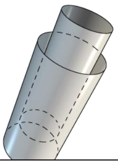
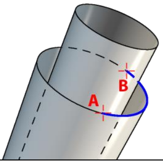
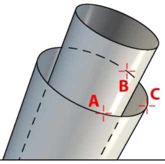
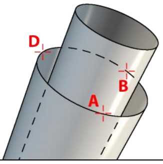
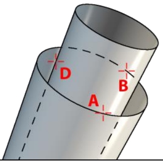
Please specify the layer for each point. L1 is the front-most surface layer, L2 the second layer, and so on. Cross one item per line.

Point	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**	**L7**
A	☐	☐	☐	☐	☐	☐	☐
B	☐	☐	☐	☐	☐	☐	☐
C	☐	☐	☐	☐	☐	☐	☐
D	☐	☐	☐	☐	☐	☐	☐
E	☐	☐	☐	☐	☐	☐	☐
F	☐	☐	☐	☐	☐	☐	☐

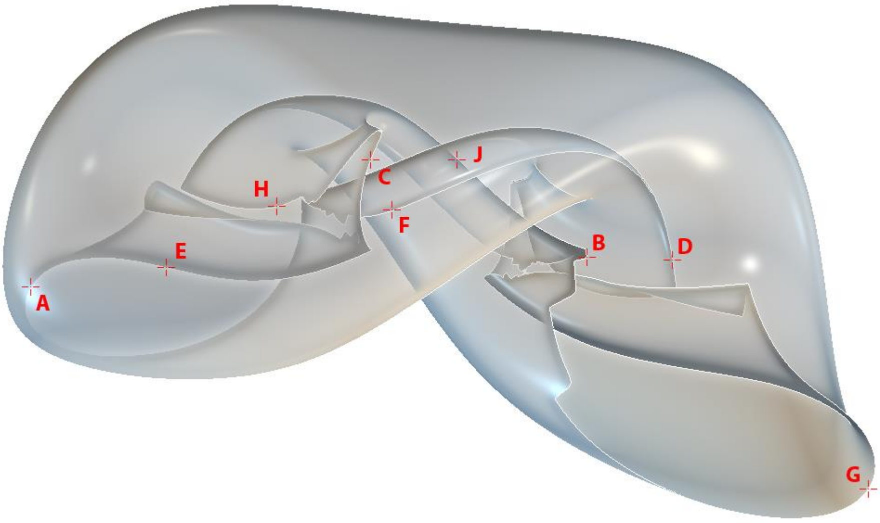
Task: follow the surface boundary

For this task you are asked to **follow the boundary of the surface**. You will be given two points on the boundary and have to decide which of the other points lie on the shortest path between the two points.

DEFINITIONS AND EXAMPLES

	For the examples, we will again use a picture of two concentric open tubes.
	In this picture, you are given two end points A and B. The **shortest path along the surface boundary** between these points is highlighted blue.
	In this picture, point **C lies on the shortest path from A to B** along the surface boundary.
	In this picture, point **D does NOT lie on the shortest path from A to B** along the surface boundary, since the path from A to B over D is much longer than the path in the previous example.
	In this picture, point D does NOT lie on the shortest path from A to B along the surface boundary, since **D does not lie on a boundary at all**.

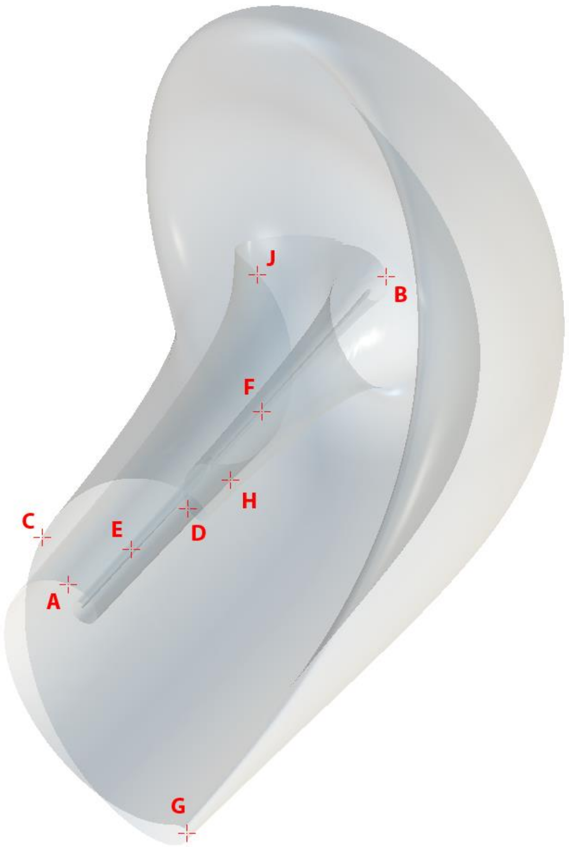
Hint: there is always at most one boundary at each marked position.



Task: follow the boundary

Which of the following points lie on shortest path between the given end points? For each row, cross all items where the crosshair marks a point on the shortest path along the surface boundary between the given start and end.

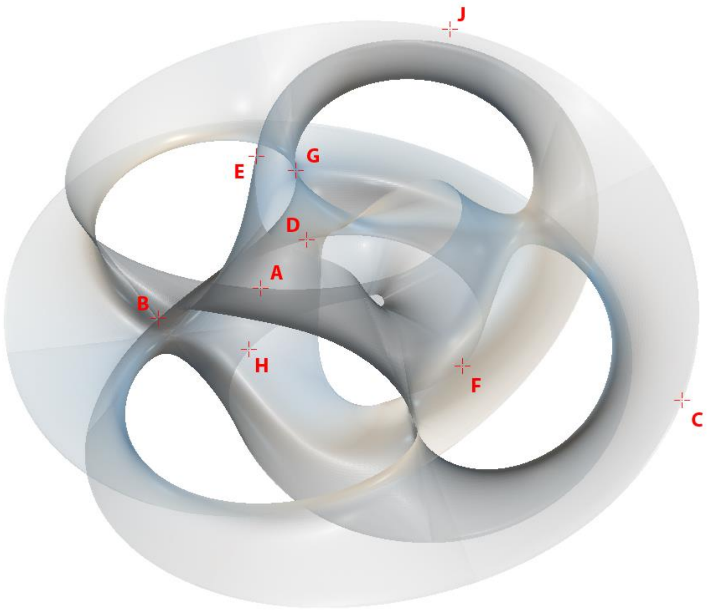
Start	End	A	B	C	D	E	F	G	H	J
A	B			☐	☐	☐	☐	☐	☐	☐
B	C	☐			☐	☐	☐	☐	☐	☐
D	B	☐		☐		☐	☐	☐	☐	☐
A	D		☐	☐		☐	☐	☐	☐	☐



Task: follow the boundary

Which of the following points lie on shortest path between the given end points? For each row, cross all items where the crosshair marks a point on the shortest path along the surface boundary between the given start and end.

Start	End	A	B	C	D	E	F	G	H	J
A	**B**			☐	☐	☐	☐	☐	☐	☐
B	**C**	☐			☐	☐	☐	☐	☐	☐
D	**B**	☐		☐		☐	☐	☐	☐	☐
A	**D**		☐	☐		☐	☐	☐	☐	☐



Task: follow the boundary

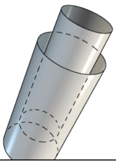
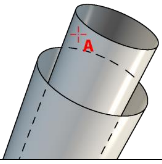
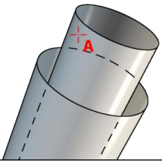
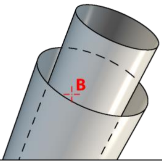
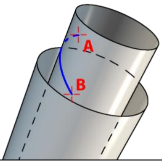
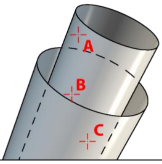
Which of the following points lie on shortest path between the given end points? For each row, cross all items where the crosshair marks a point on the shortest path along the surface boundary between the given start and end.

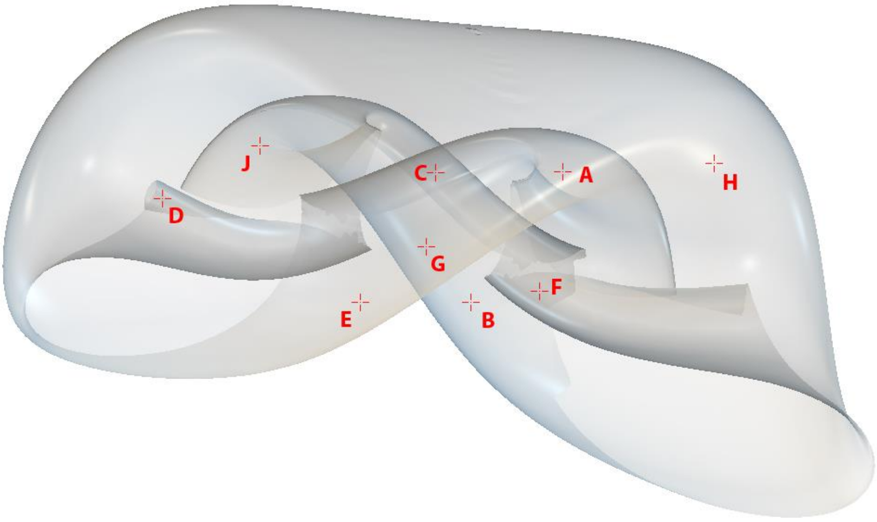
Start	End	A	B	C	D	E	F	G	H	J
A	B			☐	☐	☐	☐	☐	☐	☐
A	C		☐		☐	☐	☐	☐	☐	☐
C	D	☐	☐			☐	☐	☐	☐	☐
A	D		☐	☐		☐	☐	☐	☐	☐

Task: find the nearest element when moving on the surface

For this task you are asked to **follow the surface as if walking on it**. You will be given one point on the surface and have to decide which of the other points is closest to it.

DEFINITIONS AND EXAMPLE

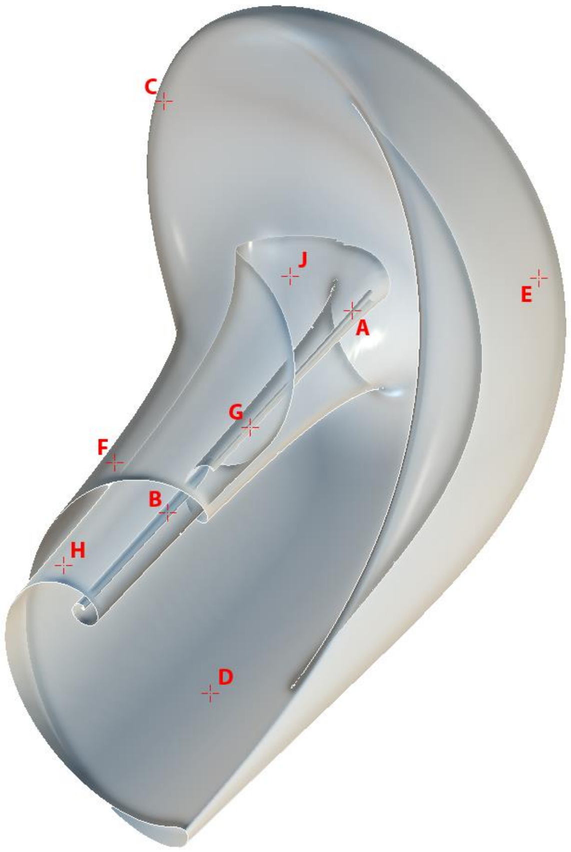
	For the examples, we will again use a picture of two concentric open tubes.
	In this task, you will also be given the layer at which a point is located. Point **"A(L2)" means that the point is at the second layer under the position labeled A**. In this picture, this would be the back side of the smaller tube.
	In this task, the **points lie on both sides of the specified surface layer**. In this picture, A(L2) lies both on the inner and the outer side of the back of the smaller tube. Always choose the side which results in the shorter path.
	Similarly, point **B(L1) is a point on the first surface layer at position B**. In this picture, this would be the front side of the smaller tube.
	Consider the points A(L2) and B(L1) in this picture. The shortest path along the surface between these points is highlighted blue.
	In this picture, you are given the start point A(L2) and two candidates B(L1) and C(L2). **Point B(L1) is the closest point to A(L2)**; both B(L1) and C(L2) lie on the front side of the smaller tube, but the path from A(L2) to B(L1) is much shorter than the path from A(L2) to C(L2).



Task: find the nearest element when moving on the surface

Which of the five given points is closest to the start point? Cross one item per line.

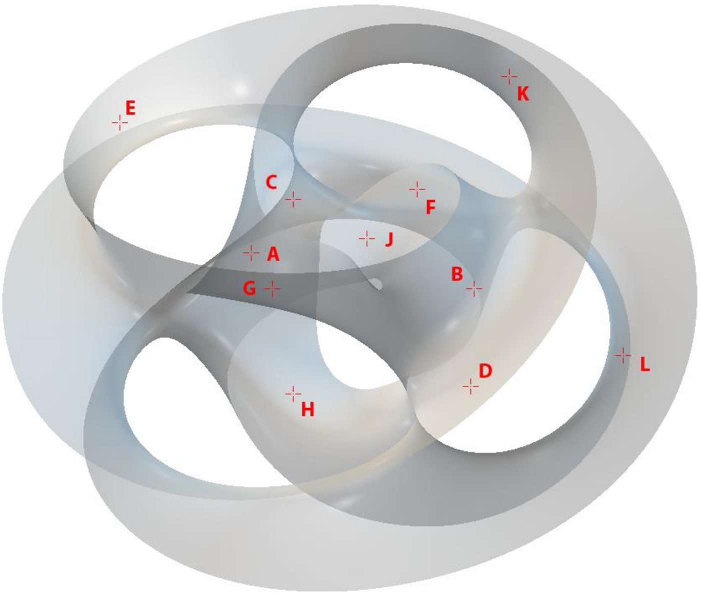
Start point	Point 1	Point 2	Point 3	Point 4	Point 5
A (L2)	☐ B(L1)	☐ C(L4)	☐ C(L5)	☐ H(L1)	☐ J(L2)
B (L1)	☐ A(L2)	☐ C(L1)	☐ C(L3)	☐ E(L2)	☐ J(L2)
C (L2)	☐ C(L1)	☐ C(L4)	☐ D(L2)	☐ F(L1)	☐ G(L3)
D (L1)	☐ A(L2)	☐ B(L1)	☐ F(L2)	☐ H(L2)	☐ J(L2)



Task: find the nearest element when moving on the surface

Which of the five given points is closest to the start point? Cross one item per line.

Start point	Point 1	Point 2	Point 3	Point 4	Point 5
A (L2)	☐ C(L1)	☐ D(L1)	☐ E(L1)	☐ F(L1)	☐ H(L1)
B (L2)	☐ C(L1)	☐ E(L1)	☐ F(L1)	☐ G(L3)	☐ J(L2)
C (L1)	☐ B(L1)	☐ D(L1)	☐ G(L3)	☐ H(L1)	☐ H(L2)
D (L1)	☐ F(L1)	☐ G(L1)	☐ G(L2)	☐ G(L3)	☐ J(L2)



Task: find the nearest element when moving on the surface

Which of the five given points is closest to the start point? Cross one item per line.

Start point	Point 1	Point 2	Point 3	Point 4	Point 5
A (L2)	☐ B(L1)	☐ D(L1)	☐ E(L1)	☐ F(L1)	☐ K(L1)
B (L1)	☐ A(L3)	☐ C(L3)	☐ G(L3)	☐ H(L1)	☐ L(L2)
A (L3)	☐ C(L1)	☐ E(L1)	☐ F(L1)	☐ G(L1)	☐ K(L1)
C (L1)	☐ D(L1)	☐ E(L1)	☐ H(L1)	☐ G(L1)	☐ K(L1)

B
BIBLIOGRAPHY

[11] Marc K. Albert. "Occlusion, transparency, and lightness". In: *Vision Research* 47.24 (2007), pp. 3061–3069 (cit. on p. 93).

[12] R. Amthauer et al. "Intelligenzstruktur-Test-2000-R". In: *Zeitschrift für Entwicklungspsychologie und Pädagogische Psychologie* 32(3) (2001), pp. 166–169 (cit. on p. 118).

[13] B. L. Anderson. "The Role of Occlusion in the Perception of Depth, Lightness, and Opacity". In: *Psychological Review* 110.4 (2003), pp. 785–801 (cit. on p. 94).

[14] Arthur Appel, F. James Rohlf, and Arthur J. Stein. "The haloed line effect for hidden line elimination". In: *SIGGRAPH Computer Graphics* 13.2 (1979), pp. 151–157 (cit. on p. 70).

[15] R. A. Bailey. *Design of Comparative Experiments*. Cambridge University Press, 2008 (cit. on p. 115).

[16] A. Bandura. "Self-efficacy: Towards a unifying theory of behavioral change". In: *Psychological Review* 84 (2) (1977), pp. 191–215 (cit. on p. 117).

[17] L. Bavoil et al. "Multi-fragment effects on the GPU using the k-buffer". In: *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*. 2007, pp. 97–104 (cit. on p. 77).

[18] J. Beck. "Perception of Transparency in Man and Machine". In: *Computer Vision, Graphics, and Image Processing* 31.2 (1985), pp. 127–138 (cit. on p. 94).

[19] Jelena Bock et al. "4D phase contrast MRI at 3T: Effect of standard and blood-pool contrast agents on SNR, PC-MRA, and blood flow visualization". In: *Magnetic Resonance in Medicine* 63.2 (2010), pp. 330–338 (cit. on pp. 47, 53–55).

[20] Silvia Born et al. "Visual Analysis of Cardiac 4D MRI Blood Flow Using Line Predicates". In: *IEEE Transactions on Visualization and Computer Graphics* 19.6 (2013), pp. 1–14 (cit. on p. 44).

[21] G. E. P. Box, J. S. Hunter, and W. G. Hunter. *Statistics for Experimenters: Design, Innovation, and Discovery*. Wiley-Interscience, 2005 (cit. on pp. 116, 117, 119).

[5] Andrea Brambilla, **Robert Carnecky**, Ronald Peikert, Ivan Viola, and Helwig Hauser. "Illustrative Flow Visualization: State of the Art, Trends and Challenges". In: *EG 2012 - State of the Art Reports*. Cagliari, Sardinia, Italy: Eurographics Association, 2012, pp. 75–94 (cit. on p. 9).

[22] Andrea Brambilla et al. "Illustrative Flow Visualization: State of the Art, Trends and Challenges". In: *EG 2012 - State of the Art Reports*. Eurographics Association, 2012, pp. 75–94 (cit. on p. 2).

[23] Stefan Bruckner and Eduard Gröller. "Enhancing Depth-Perception with Flexible Volumetric Halos". In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (2007), pp. 1344–1351 (cit. on pp. 69, 70).

[24] R. Bürger et al. “Integrating Local Feature Detectors in the Interactive Visual Analysis of Flow Simulation Data”. In: *Proc. Eurographics/IEEE-VGTC Symposium on Visualization 2007*. 2007, pp. 171–178 (cit. on p. 18).

[25] Julia Busch et al. “Reconstruction of divergence-free velocity fields from cine 3D phase-contrast flow measurements”. In: *Magnetic Resonance in Medicine* 69.1 (2013), pp. 200–210 (cit. on p. 46).

[26] Loren Carpenter. “The A-buffer, an antialiased hidden surface method”. In: *Proceedings of the 11th annual conference on Computer Graphics and Interactive Techniques*. 1984, pp. 103–108 (cit. on p. 76).

[27] Ming-Yuen Chan et al. “Perception-Based Transparency Optimization for Direct Volume Rendering”. In: *IEEE Transactions on Visualization and Computer Graphics* 15 (2009), pp. 1283–1290 (cit. on p. 69).

[28] Cheng-Kai Chen, R. Thomason, and Kwan-Liu Ma. “Intelligent Focus+Context Volume Visualization”. In: *Intelligent Systems Design and Applications, 2008. ISDA '08. Eighth International Conference on*. Vol. 1. 2008, pp. 368–374 (cit. on p. 9).

[29] M. Cooper. “The Tractability of Segmentation and Scene Analysis”. In: *International Journal of Computer Vision* 30(1) (1998), pp. 27–42 (cit. on p. 38).

[30] Peter F Davies. “Flow-Mediated Endothelial Mechanotransduction”. In: *Physiol. Rev.* 75.3 (1995), pp. 519–560 (cit. on p. 43).

[31] Doug DeCarlo et al. “Suggestive contours for conveying shape”. In: *ACM Transactions on Graphics* 22.3 (3 2003), pp. 848–855 (cit. on p. 70).

[32] J. Diepstraten, D. Weiskopf, and T. Ertl. "Transparency in Interactive Technical Illustrations". In: *Computer Graphics Forum* 21.3 (2002), pp. 317–325 (cit. on pp. 9, 69).

[33] J. Diepstraten, D. Weiskopf, and T. Ertl. "Interactive Cutaway Illustrations". In: *Computer Graphics Forum* 22.3 (2003), pp. 523–532 (cit. on p. 9).

[34] Helmut Doleisch, Martin Gasser, and Helwig Hauser. "Interactive Feature Specification for Focus+Context Visualization of Complex Simulation Data". In: *Proceedings of the Symposium on Data Visualisation 2003*. VISSYM '03. 2003, pp. 239–248 (cit. on p. 9).

[35] Y. Dubief and F. Delcayre. "On coherent-vortex identification in turbulence". In: *Journal of Turbulence* 1 (2000), N11 (cit. on p. 52).

[36] M. H. Everts et al. "Depth-Dependent Halos: Illustrative Rendering of Dense Line Data". In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), pp. 1299–1306 (cit. on p. 70).

[37] F. Faul et al. "Statistical power analyses using GPower 3.1: Tests for correlation and regression analyses". In: *Behavior Research Methods* 41 (2009), pp. 1149–1160 (cit. on p. 117).

[38] S. K. Feiner and D. D. Seligmann. "Cutaways and ghosting: satisfying visibility constraints in dynamic 3D illustrations". In: *The Visual Computer* 8(5) (5 1992), pp. 292–302 (cit. on p. 9).

[39] O. Frederich et al. "Large-scale dynamics in the flow around a finite cylinder with a ground plate". In: *Fluid Dynamics Research* 43(1) (2011), 015504:1–015504:22 (cit. on p. 30).

[40] Alex Frydrychowicz et al. “Multidirectional Flow Analysis by Cardiovascular Magnetic Resonance in Aneurysm Development Following Repair of Aortic Coarctation”. In: *Journal of Cardiovascular Magnetic Resonance* 10.30 (2008) (cit. on pp. 43, 44).

[41] Raphael Fuchs, Volkmar Welker, and Joachim Hornegger. “Non-convex polyhedral volume of interest selection”. In: *Computerized Medical Imaging and Graphics* 34 (2 2009), pp. 105–113 (cit. on p. 9).

[42] Amy Gooch et al. “A non-photorealistic lighting model for automatic technical illustration”. In: *Proc. SIGGRAPH*. ACM, 1998, pp. 447–452 (cit. on pp. 70, 71, 143).

[43] Brian Gough. *GNU Scientific Library Reference Manual - Third Edition*. 3rd. 2009 (cit. on p. 58).

[44] Hákon Gudbjartsson and Samuel Patz. “The rician distribution of noisy MRI data”. In: *Magnetic Resonance in Medicine* 34.6 (1995), pp. 910–914 (cit. on p. 46).

[45] Tobias Günther et al. “Opacity Optimization for Surfaces”. In: *Computer Graphics Forum* 33.3 (2014), pp. 11–20 (cit. on pp. 69, 148).

[46] George Haller. “Distinguished material surfaces and coherent structures in three-dimensional fluid flows”. In: *Phys. D* 149.4 (2001), pp. 248–277 (cit. on pp. 4, 125, 142).

[47] M. Harris, S. Sengupta, and J. D. Owens. “Parallel Prefix Sum (Scan) with CUDA.” In: *GPU Gems 3*. Addison Wesley, 2007. Chap. 39, pp. 851–876 (cit. on p. 27).

[48] Helwig Hauser. “Scientific Visualization: The Visual Extraction of Knowledge from Data.” In: 2003. Chap. Generalizing focus+context visualization, pp. 305–327 (cit. on p. 9).

[49] Paul Heckbert. “Texture mapping polygons in perspective”. In: *NYIT Computer Graphics Lab Tech. Memo* 13 (1983) (cit. on p. 130).

[50] J.L. Helman and Lambertus Hesselink. “Visualizing vector field topology in fluid flows”. In: *Computer Graphics and Applications, IEEE* 11.3 (1991), pp. 36–46 (cit. on p. 4).

[51] Kevin Heng, Kristen Menou, and Peter J. Phillipps. “Atmospheric circulation of tidally locked exoplanets: a suite of benchmark tests for dynamical solvers”. In: *Monthly Notices of the Royal Astronomical Society* 413.4 (2011), pp. 2380–2402 (cit. on p. 140).

[52] Elaine R. S. Hodges. *The Guild Handbook of Scientific Illustration*. Wiley, 2003, p. 656 (cit. on pp. 69, 92).

[53] Mathias Hummel et al. “IRIS: illustrative rendering for integral surfaces.” In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (2010), pp. 1319–28 (cit. on pp. 69, 70, 85, 103).

[54] Victoria Interrante, Henry Fuchs, and Stephen Pizer. “Enhancing Transparent Skin Surfaces with Ridge and Valley Lines”. In: *Proceedings of IEEE Visualization 1995*. 1995, pp. 52–60 (cit. on p. 69).

[55] Victoria Interrante, Henry Fuchs, and Stephen Pizer. “Illustrating transparent surfaces with curvature-directed strokes”. In: *Proceedings of IEEE Visualization 1996*. 1996, pp. 211–219 (cit. on p. 69).

[56] Tobias Isenberg et al. "A Developer's Guide to Silhouette Algorithms for Polygonal Models". In: *IEEE Computer Graphics and Applications* 23.4 (2003), pp. 28–37 (cit. on p. 71).

[57] J. Jeong and F. Hussain. "On the identification of a vortex". In: *Journal of Fluid Mechanics* 285 (1994), pp. 69–94 (cit. on pp. 30, 47, 52).

[58] J. Jeong et al. "Coherent structures near the wall in a turbulent channel flow". In: *Journal of Fluid Mechanics* 332 (1997), pp. 185–214 (cit. on p. 52).

[59] Tilke Judd, Frédo Durand, and Edward Adelson. "Apparent ridges for line drawing". In: *ACM Transactions on Graphics* 26.3 (2007), pp. 19–26 (cit. on p. 70).

[60] S. Khanna, S. Muthukrishnan, and M. Paterson. "On approximating rectangle tiling and packing". In: *Proc. 9th annual ACM-SIAM Symposium on Discrete Algorithms*. 1998, pp. 384–393 (cit. on p. 38).

[61] Philip J Kilner et al. "Helical and Retrograde Secondary Flow Patterns in the Aortic Arch Studied by Three-Directional Magnetic Resonance Velocity Mapping". In: *Circulation* 88 (1993), pp. 2235–2247 (cit. on pp. 43, 60).

[62] Scott Kirkpatrick. "Optimization by simulated annealing: Quantitative studies". In: *Journal of Statistical Physics* 34.5 (1984), pp. 975–986 (cit. on pp. 22, 24).

[63] Benjamin Köhler et al. "Semi-Automatic Vortex Extraction in 4D PC-MRI Cardiac Blood Flow Data using Line Predicates." In: *IEEE TVCG* 19.12 (Dec. 2013), pp. 2773–82 (cit. on pp. 44, 47, 52, 57).

[64] Václav Kolář, Pavel Moses, and Jakub Šístek. "Local corotation of line segments and vortex identification". In: *Proc. 17th Australasian Fluid Mech. Conf.* 2010, pp. 638–652 (cit. on pp. 47, 49).

[65] Harinarayan Krishnan et al. "Analysis of Time-Dependent Flow-Sensitive PC-MRI Data". In: *IEEE Transactions on Visualization and Computer Graphics* 18.6 (June 2012), pp. 966–977 (cit. on p. 47).

[66] Jan-Willem Lankhaar et al. "Correction of phase offset errors in main pulmonary artery flow quantification". In: *Journal of Magnetic Resonance Imaging* 22.1 (2005), pp. 73–79 (cit. on p. 45).

[67] Robert S. Laramee et al. "The State of the Art in Flow Visualization: Dense and Texture-Based Techniques". In: *Computer Graphics Forum* 23.2 (2004), pp. 203–221 (cit. on pp. 4, 5).

[68] D. Lichtenstein. "Planar formulae and their uses". In: *SIAM Journal on Computing* 11(2) (1982), pp. 329–343 (cit. on p. 38).

[69] Wentao Liu et al. "3D phase unwrapping using global expected phase as a reference: Application to MRI global shimming". In: *Magnetic Resonance in Medicine* 70.1 (2013), pp. 160–168 (cit. on p. 46).

[70] Michel Loeve. *Probability Theory*. 3rd. D. Van Nostrand, 1963 (cit. on p. 50).

[71] Martin Luboschik, Axel Radloff, and Heidrun Schumann. "A new weaving technique for handling overlapping regions". In: *Proceedings of the International Conference on Advanced Visual Interfaces*. 2010, pp. 25–32 (cit. on p. 69).

[72] Thomas Luft, Carsten Colditz, and Oliver Deussen. "Image enhancement by unsharp masking the depth buffer". In: *ACM Transactions on Graphics* 25.3 (2006), pp. 1206–1213 (cit. on p. 69).

[73] Eric B. Lum, Aleksander Stompel, and Kwan-Liu Ma. "Using Motion to Illustrate Static 3D Shape–Kinetic Visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 9.2 (2003), pp. 115–126 (cit. on p. 69).

[74] John L. Lumley. "The structure of inhomogeneous turbulent flows". In: *Atmospheric turbulence and radio wave propagation* (1967), pp. 166–178 (cit. on p. 50).

[75] A. Mammen. "Transparency and antialiasing algorithms implemented with the virtual pixel maps technique". In: *IEEE Computer Graphics and Applications* 9.4 (1989), pp. 43–55 (cit. on p. 76).

[76] José V. Manjón et al. *Diffusion Weighted Image Denoising Using Overcomplete Local PCA*. 2013 (cit. on p. 46).

[77] Michael Markl et al. "Time-resolved 3D MR velocity mapping at 3T: Improved navigator-gated assessment of vascular anatomy and blood flow". In: *Journal of Magnetic Resonance* 25.4 (2007), pp. 824–831 (cit. on p. 53).

[78] Michael Markl et al. "4D flow MRI". In: *Journal of Magnetic Resonance Imaging* 36.5 (2012), pp. 1015–1036 (cit. on p. 44).

[79] M.V. McConnell et al. "Comparison of respiratory suppression methods and navigator locations for MR coronary angiography". In: *American Journal of Roentgenology* 168.5 (1997), pp. 1369–75 (cit. on p. 45).

[80] Robert McGregor et al. “Exploring the Use of Proper Orthogonal Decomposition for Enhancing Blood Flow Images Via Computational Fluid Dynamics.” In: *Medical Image Computing and Computer-Assisted Intervention 2008*. Vol. 5242. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 782–789 (cit. on p. 46).

[81] Tony McLoughlin, Robert S. Laramee, and Eugene Zhang. “Constructing Streak Surfaces for 3D Unsteady Vector Fields”. In: *Proceedings of the 26th Spring Conference on Computer Graphics*. SCCG '10. 2010, pp. 17–26 (cit. on p. 5).

[82] Tony McLoughlin et al. “Over Two Decades of Integration-Based, Geometric Flow Visualization”. In: *Computer Graphics Forum* 29 (6 2010), pp. 1807–1829 (cit. on p. 4).

[83] R.B. McMaster and K.S. Shea. *Generalization in Digital Cartography*. Resource Publications for College Geography. Association of American Geographers, 1992 (cit. on p. 11).

[84] N. Metropolis et al. “Equation of State Calculations by Fast Computing Machines”. In: *Journal of Computational Physics* 21 (1953), p. 1087 (cit. on p. 22).

[85] Kevin Myers and Louis Bavoil. “Stencil routed A-Buffer”. In: *ACM SIGGRAPH 2007 sketches*. 2007 (cit. on p. 77).

[86] Ken Nakayame, Shinsuke Shimojo, and Vilayanur S Ramachandran. “Transparency: relation to depth, subjective contours, luminance, and neon color spreading”. In: *Perception* 19.4 (1990), pp. 497–513 (cit. on p. 93).

[87] Marc Nienhaus and Jürgen Döllner. "Blueprints: illustrating architecture and technical parts using hardware-accelerated non-photorealistic rendering". In: *Proceedings of Graphics Interface 2004*. 2004, pp. 49–56 (cit. on p. 104).

[88] NVIDIA. *CUDA: Compute Unified Device Architecture*. 2011. URL: https://developer.nvidia.com/about-cuda (cit. on p. 27).

[89] Roy van Pelt et al. "Automated segmentation of blood-flow regions in large thoracic arteries using 3D-cine PC-MRI measurements". In: *International Journal of Computer Assisted Radiology and Surgery* 2 (2012), pp. 217–224 (cit. on p. 47).

[90] Ken Perlin. "Noise Hardware." In: *SIGGRAPH 2001 Course Notes*. Ed. by M. Olano. ACM, 2001. Chap. 9 (cit. on p. 128).

[91] A. Pobitzer et al. "Energy-scale aware feature extraction for unsteady flow visualization". In: *Computer Graphics Forum* 30.3 (2011) (cit. on pp. 52, 66).

[92] Frits H. Post et al. "The State of the Art in Flow Visualisation: Feature Extraction and Tracking". In: *Computer Graphics Forum* 22.4 (2003), pp. 775–792 (cit. on p. 4).

[93] Fangtu T Qiu and Rüdiger von der Heydt. "Neural representation of transparent overlay". In: *Nature Neuroscience* 10 (2007), pp. 283–284 (cit. on p. 92).

[94] Peter Rautek et al. "Illustrative visualization: new technology or useless tautology?" In: *ACM SIGGRAPH Computer Graphics* 42.3 (Aug. 2008) (cit. on p. 2).

[95] W. M. van Rees et al. "A comparison of vortex and pseudo-spectral methods for the simulation of periodic vortical flows at high Reynolds numbers". In: *Journal of Computational Physics* 230.8 (2011), pp. 2794–2805 (cit. on p. 18).

[96] Yoram Richter and Elazer R Edelman. "Cardiology is flow". In: *Circulation* 113.23 (2006), pp. 2679–2682 (cit. on p. 43).

[97] Ari Sadarjoen and Frits H. Post. "Detection, quantification, and tracking of vortices using streamline geometry". In: *Computers & Graphics* 24.3 (2000), pp. 333 –341 (cit. on p. 5).

[98] Hanno Scharr and Joachim Weickert. "An anisotropic diffusion algorithm with optimized rotation invariance". In: *Proc. DAGM-symposium*. 2000, pp. 460–467 (cit. on p. 134).

[99] B. Schindler et al. "Marching Correctors – Fast and Precise Polygonal Isosurfaces of SPH Data". In: *Proc. of the 6th International SPHERIC workshop*. 2011, pp. 125–132 (cit. on p. 18).

[100] B. Schindler et al. "Lagrangian Coherent Structures for Design Analysis of Revolving Doors". In: *Visualization and Computer Graphics, IEEE Transactions on* 18.12 (2012), pp. 2159–2168 (cit. on p. 142).

[9] Benjamin Schindler, Raphael Fuchs, Stephan Barp, Jürgen Waser, Armin Pobitzer, **Robert Carnecky**, Krešimir Matković, and Ronald Peikert. "Lagrangian Coherent Structures for Design Analysis of Revolving Doors". In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2159–2168 (cit. on p. 5).

[101] Sohail Shafii et al. “Illustrative Rendering of Vortex Cores”. In: *EuroVis 2013 Short Papers*. 2013, pp. 61–65 (cit. on p. 47).

[102] Prateek Sharma and Gregory W. Hammett. “Preserving monotonicity in anisotropic diffusion”. In: *Journal of Computational Physics* 227.1 (2007), pp. 123–142 (cit. on p. 134).

[103] Lawrence Sirovich. “Turbulence and the dynamics of coherent structures”. In: *Quarterly of applied mathematics* 45 (1987), pp. 561–571 (cit. on p. 50).

[104] A F Stalder et al. “Vortex Core Detection and Visualization using 4D Flow-sensitive MRI”. In: *Proc. Intl. Soc. Mag. Reson. Med.* Vol. 18. 2010, p. 3708 (cit. on pp. 44, 57).

[105] Stanford. *The Stanford volume data archive*. URL: `http://graphics.stanford.edu/data/voldata/` (cit. on p. 32).

[106] John C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. SIAM, 2004, pp. 34–36 (cit. on p. 133).

[107] Turbosquid. *Model ID 369057*. 2007. URL: `http://www.turbosquid.com/3d-models/free-intake-3d-model/369057` (visited on 2015) (cit. on p. 29).

[108] T. Urness et al. “Strategies for the visualization of multiple 2D vector fields”. In: *IEEE Computer Graphics and Applications* 26.4 (2006), pp. 74–82 (cit. on p. 69).

[109] Romain Vergne et al. “Light warping for enhanced surface depiction”. In: *ACM Transactions on Graphics* 28.3 (2009), 25:1–25:8 (cit. on p. 71).

[110] Ivan Viola and Meister E. Gröller. "Smart Visibility in Visualization". In: *Proceedings of the First Eurographics Conference on Computational Aesthetics in Graphics, Visualization and Imaging*. Computational Aesthetics'05. 2005, pp. 209–216 (cit. on p. 9).

[111] Ivan Viola, Armin Kanitsar, and M. Eduard Groller. "Importance-Driven Feature Enhancement in Volume Visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 11.4 (July 2005), pp. 408–418 (cit. on p. 9).

[112] Jürgen Waser et al. "World Lines". In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (2010), pp. 1458–1467 (cit. on pp. xiii, 52, 105).

[113] Dominik Weishaupt, Victor D. Köchli, and Borut Marincek. *How Does MRI Work?* Springer, 2008 (cit. on p. 45).

[114] R. A. Wilson and F. C. Keil. *MIT Encyclopedia of the Cognitive Sciences*. MIT Press, 2001 (cit. on p. 93).

[115] Jason C. Yang et al. "Real-Time Concurrent Linked List Construction on the GPU". In: *Computer Graphics Forum* 29.4 (2010), pp. 1297–1304 (cit. on pp. 77, 80).

[116] Long Zhang et al. "Laplacian lines for real-time shape illustration". In: *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*. 2009, pp. 129–136 (cit. on p. 70).