Intelligent Cutaway Illustrations

Stephan Sigg* Raphael Fuchs† Robert Carnecky‡ Ronald Peikert§

ETH Zurich

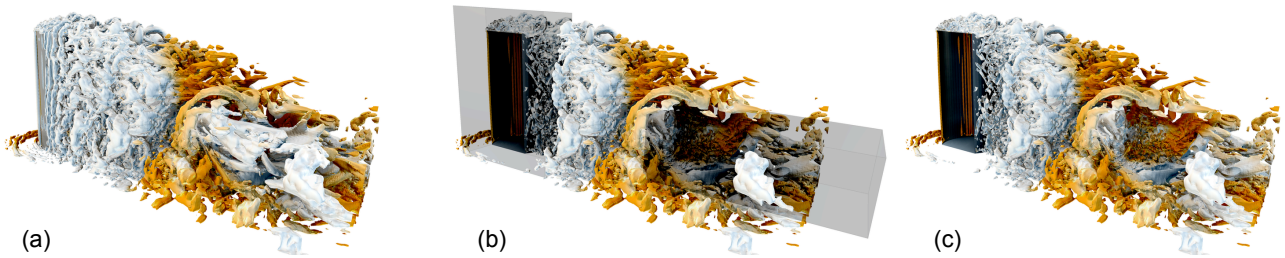

Figure 1: Cutaway image from a flow visualization. (a) Features of interest are colored orange. (b) The algorithm determines the positions of the cutting objects in order to reveal as much of the features as possible. (c) The resulting cutaway.

ABSTRACT

Artistic illustrations of important structures in fluid flow have a long-standing tradition and are appreciated as clearly perceivable, instructive, but still conveying all relevant information to the viewer. One important illustrative technique for such visualizations are cutaways. Currently cutaways are placed manually or using view-vector based approaches. We propose to optimize the visibility of important target features based on a degree-of-interest (DOI) function. The DOI is specified during interactive visual analysis, e.g., by brushing scatterplots. We show that the problem of placing cutaway boxes optimally is NP-hard in the number of boxes. To overcome this obstacle, we present an intelligent method to compute cutaways. Geometric cutaway objects are positioned using a view-dependent objective function which optimizes the visibility of all features. In order to approximate the optimal solution, we use a Monte Carlo method and exploit temporal coherence in dynamic scenes. Performance-critical parts are implemented on the GPU. The proposed method can be integrated easily into existing rendering frameworks and is general enough to be able to optimize other parameters besides cutaways as well. We evaluate the performance of the algorithm and provide a case study of vorticity visualization in a turbulent flow.

Keywords: Cutaway, Scientific Illustration, View-Dependent Optimization, Monte Carlo, Flow Visualization.

1 INTRODUCTION

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. 1a, there is no hint for an important region on the left side, but it is revealed nevertheless by our algorithm in Fig. 1c. 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.

The contributions of this paper are

- A method to place comprehensible cutaway objects automatically based on a flexible degree-of-interest (DOI) function.
- Cutaway placement which is independent of the rendering technique.
- A proof that the problem of placing multiple cutaway boxes optimally is NP-hard.
- An approach for optimizing rendering parameters based on a MC method.

This paper is structured as follows: Section 2 discusses related work. In Section 3, we provide an overview of the algorithm. Section 4 discusses the MC algorithm and the adaptations required for our problem. Section 5 discusses frame-to-frame coherency. In Section 6, we discuss implementation details. Section 7 presents results and in Section 8, we evaluate the consistency of the algorithm. Section 9 contains a proof that the problem of finding opti-

*e-mail: stephan.sigg@gmail.com
†e-mail: raphael@inf.ethz.ch
‡e-mail: crobi@inf.ethz.ch
§e-mail: peikert@inf.ethz.ch

IEEE Pacific Visualization Symposium 2012
28 February - 2 March, Songdo, Korea
978-1-4673-0866-3/12/$31.00 ©2012 IEEE

mal cutaway boxes is NP-hard. Finally, we draw conclusions and propose future work.

2 Related Work

In this section, we review different techniques to produce images from polygonal meshes or volumetric data which reveal interesting features while preserving the context. In this paper we focus on cutaway illustrations that use clipping operations to cut out parts of the displayed object [33]. For volume rendering, similar approaches that optimize the transfer function based on visibility and quality terms have been proposed by Correa and Ma [10] and Wu and Qu [34].

Interactive specification of cutaways Pelizzari et al. discuss the application of manually positioned cutaways for clinical treatment planning [29]. McGuffin et al. [23] present a set of interactive volume cutting tools and combine these tools with deformations to keep the context information. Coffin [8] suggests a method for defining perspective cutouts to improve virtual X-ray vision. The approach suggested by Chen et al. [7] is based on interaction metaphors such as drilling, lasering and peeling operations. Bonanni et al. [4] suggest scraping as an interaction metaphor to remove occluding layers on a touch-screen. McInerney and Crawford [24] introduce a paint roller as an interaction metaphor to define the cutaway position and shape manually. To provide context, they suggest to retain ribbons of the occluding structure. Another interactive method is proposed by Fuchs et al. [15], who suggest non-convex polyhedral cutaway objects.

All of these methods introduce novel interaction metaphors to let the user search and define the cutaway directly and cannot reveal important features automatically. Of course interactive approaches are important, but they have two major drawbacks: First, in many cases there is not enough time available for user interaction and a solution which transfers this problem to the computer is preferred. Second, in many cases the human user is not able to find an optimal solution interactively, due to the complexity of the surface or the distribution of degree of interest on the surface. In contrast to these interactive approaches we do not suggest to position cutaway objects themselves but allow the user to specify a DOI from which the position of the cutaways is derived.

DOI based analysis In this paper we follow the paradigm of interactive visual analysis based on a notion of interest which is specified by brushing information visualization views [2, 11, 5]. In contrast to these approaches we do not directly incorporate the DOI into the rendering equation (i.e., by reducing opacity and saturation of fragments with low DOI), but perform an additional step which places a cutaway based on the DOI.

Heuristic and semi-interactive cutaways. Several related methods reveal interior or occluded structures solely based on view-dependent heuristics. Feiner et al. propose a solution for cutaways and ghosting when a fixed object in the mesh has to be visible during interaction and remove all geometry occluding it [13]. Such an approach obviously cannot deal with ambiguous situations where important parts occlude themselves and an optimal cutaway has to be found. In a seminal paper, Viola et al. [32] introduce an additional importance dimension which is based on a segmentation of the volume. The cutaway is defined directly by the viewing position similar to the approach of Feiner et al. [13]. Burns et al. [6] extend this approach to multimodal volume data. Li et al. [21] present an authoring tool based on segmentation and user classifications to create cut-away illustrations for surface meshes. Here the user has to specify the location and parameters of the cutaways but the system assists in aligning the cutaways with the geometry. Birkeland and Viola [3] suggest a heuristic placement method for a peel-away which takes the view direction into account. The method cannot handle complex or non-convex situations, though.

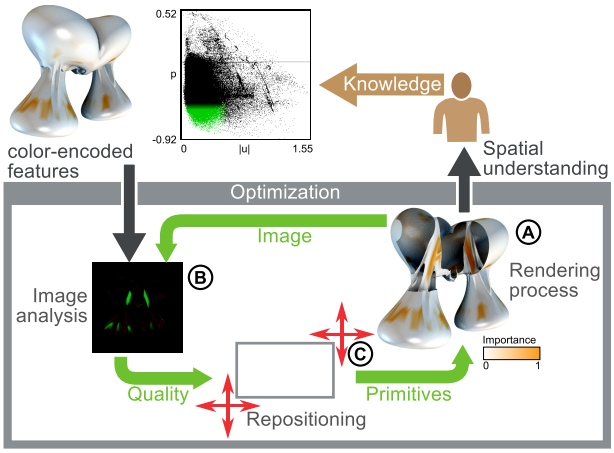

Figure 2: Computer assisted creation of cutaway illustrations. The user knows which features are relevant and can understand a rendering of the data. To place the cutaway geometry optimally in the illustration, an optimization algorithm based on a MC method interacts with the rendering system.

In all of these methods, the shape of the cutaway is controlled by heuristics based on a combination of view vector, depth and user interaction. Furthermore, the handling of complex situations when important regions occlude other important regions is not discussed. To our knowledge, there is not yet a technique to place a cutaway geometry optimally based on a flexible objective function such as the one presented in this paper. Furthermore, all approaches are designed to work with either geometry or volume rendering.

Intractable problems in visual computing A common class of NP-hard problems are various decomposition problems for polygons and polyhedra. For example, the triangulation of three-dimensional polygons is NP-hard [1] and the minimum weight triangulation is NP-hard [26] as well. Furthermore, many segmentation and scene analysis problems are NP-hard [9]. Also related to the problem discussed here is the class of tiling problems, where Khanna et al. show that optimal tilings of images can be NP-hard [18].

3 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. 2. 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. 3.

After the importance is specified, the computer finds the approximation for the best size and position of the primitives without the user's help by cycling through the three stages displayed in Fig. 2: (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

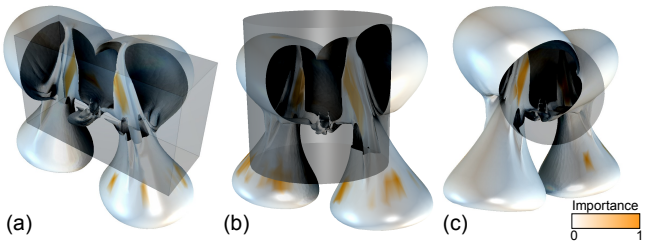

Figure 3: Three primitives used to produce cutaways: (a) cuboid, (b) cylinder, and (c) sphere. Unlike in the real results, primitives are rendered here in order to get an impression of how the method works.

used to evaluate the current cutaway positioning and to perform the optimization algorithm. This 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 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. [31] is used. We extract the isosurface with $\omega = 0.1$ using the algorithm of Schindler et al. [30] and select a range of curvatures to define the region of interest. The examples in Fig. 3 are created using this dataset.

3.2 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 [5] 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.

3.3 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 $|D| = \left(\sum_{i=1}^{n} i\right)^3$, 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.

3.4 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

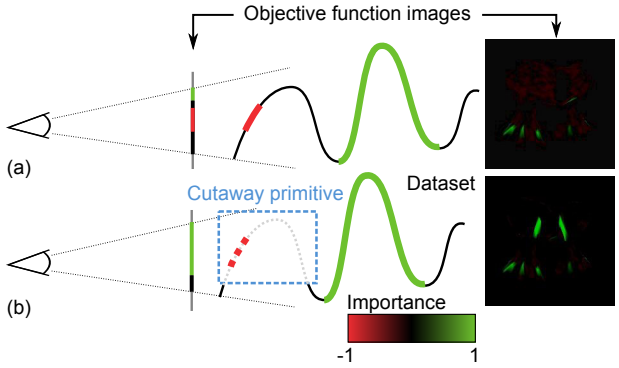

Figure 4: Composition of the objective function images. From the viewpoint of the user, the objective function images are produced by coloring the data according to the importance. Objective function images resulting from the dataset defined in Section 3.1 are displayed on the right side. (a) Without 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.

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. 4, the image is produced using an off-screen renderer that draws an image with the objective function values at each pixel as 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:

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

$\mathbf{A}$ 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.

3.5 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.

4 Optimization Algorithm: 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. [20] showed that SA is applicable to general optimization problems and also pointed out that the method performs well for hard to solve optimization problems [19].

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 [25]:

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

where x is the current configuration and x_{new} is a randomly selected configuration from the environment $E(x)$, defined in Section 3.5. SA can therefore be seen as an extension to Metropolis' algorithm originating from the beginnings of computer simulation.

4.1 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.

4.2 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.

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_S	Number of successful steps that has to be reached at a temperature.
S_{max}	Maximum number of steps at a temperature.

Table 1: Simulated annealing (SA). Definitions used in the description of the SA algorithm.

The temperature is controlled by an adaptive schedule. During initialization, the system is heated up as proposed by Kirkpatrick [19]. 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.

5 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 4 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 3.3 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 3.5 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 3.5 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.

6 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.

6.1 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 [27] provides an OpenGL interoperability feature which enables CUDA to directly process the data produced by OpenGL. As a starting point, the image postprocessing example from the CUDA SDK can be used [28]. 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 [16]. 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.

6.2 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. 5 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.

7 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.

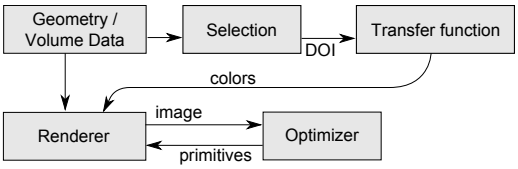

Figure 5: Integration in the rendering framework. The renderer processes the usual inputs (geometry or volume data and colors) plus a list of primitives to be clipped. The optimization process takes an image as input and measures its quality. Then, a new placement of the cut objects is proposed and transmitted to the renderer which produces a new image.

7.1 Piston Engine Dataset

The Piston Engine dataset [12] 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. 6c. This example shows that a cutaway is superior to the transparency based solution when it comes to structure and context of the selected parts.

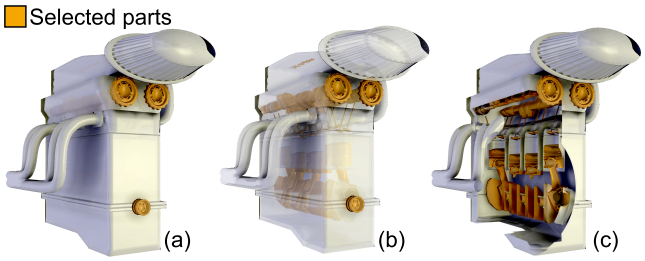

Figure 6: Piston gas engine dataset. (a) Orange parts are selected by the user. (b) Making the unwanted parts transparent makes the important parts visible whereas (c) a cutaway provides more information about spatial structure. In this example, a cuboid and a sphere are used to produce the cutaway shape.

7.2 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. [14] 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 [17]. 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. 7 illustrates two different selection cases where the locations of the important 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.

7.3 Volumetric Data

In order to show that the algorithm also supports volumetric datasets, an example of a cutaway using the CT study of a cadaver head from the Stanford volume data archive

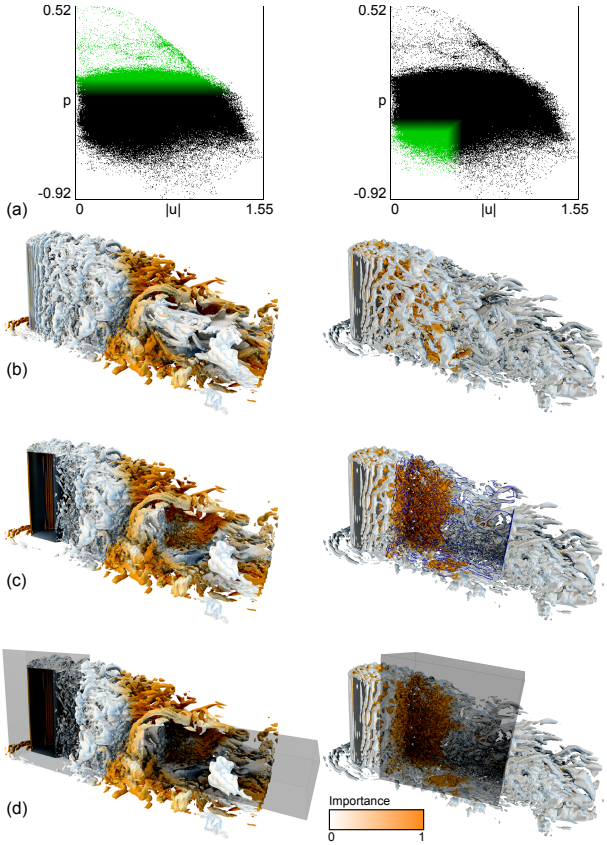

Figure 7: LES cylinder flow. (a) The user selects regions with different physical properties. (b) The data contains the selected features (orange) but they are occluded by other parts of the isosurface. (c) Parts of the data are removed by the cutaway in order to reveal the selected regions. In the right image, the cutting lines are highlighted blue. (d) The resulting cutaway.

(`http://graphics.stanford.edu/data/voldata/`) is provided in Fig. 8. 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.

8 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 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.

8.1 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 and around 15 evaluations per second for the LES cylinder example discussed

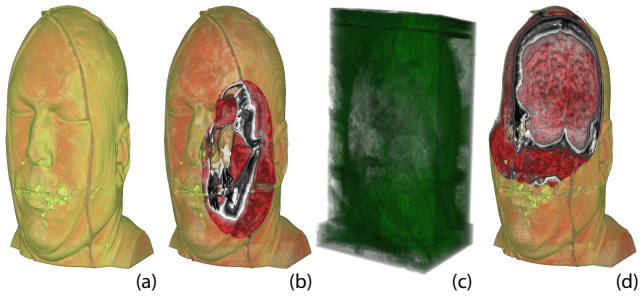

Figure 8: Volumetric dataset. (a) Direct volume rendering. (b) Example of a spherical cut-away. (c-d) Even a rough DOI with the highest values inside the brain leads to the clean cutaway in (d), which maximizes the brain cross-section.

in Section 7.2. As the configuration space defined in Section 3.3 grows with n^6, we use an example with one cuboid and $n = 10$ to perform the evaluation.

8.2 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.

8.3 Evaluation Results

In Fig. 9, 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.

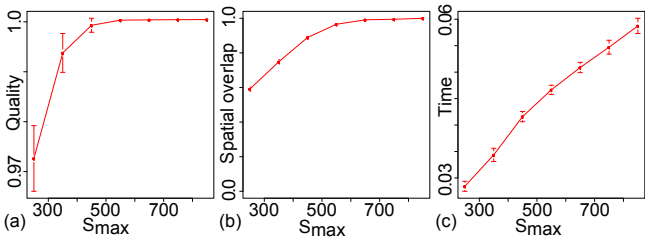

Figure 9: Evaluation results, normalized with respect to the reference run. (a) The objective function approaches the value obtained by the reference run. (b) The convergence measurement shows that the global optimum is reached consistently. (c) SA performs magnitudes better than searching the best position 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.

8.4 Parametrization

Evaluation results from different runs were used to find optimal parameters for SA. It turned out that the maximum number of steps

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. 9.

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 growing 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: $S_S \leq s \leq S_{max}$. Hence, if both of them increase, more samples are taken to find the equilibrium at each temperature and it is reached more likely.

9 Optimal Cutaways Are a Difficult Problem

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. [18] or special segmentation and scene analysis by Cooper [9].

Lichtenstein [22] 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. 10a 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. 10b.

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. 10c. 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. 10d. 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. □

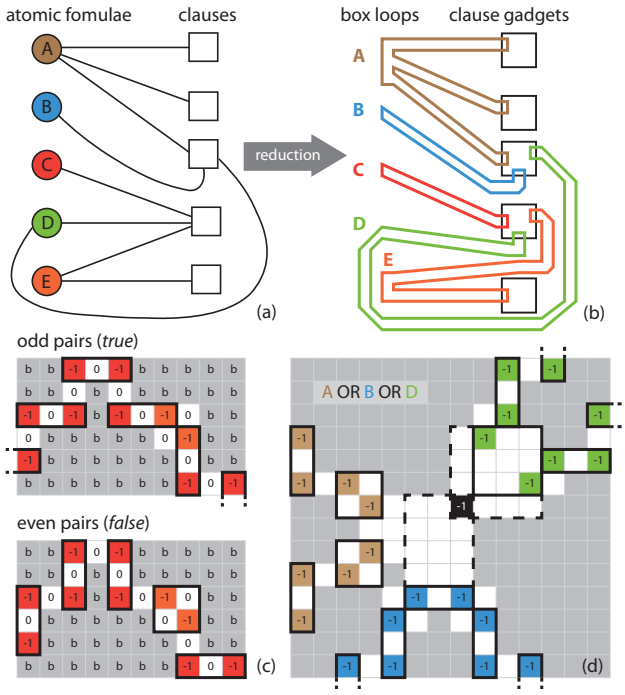

Figure 10: (a) An instance of PLANAR-3SAT. (b) An instance of CUTPOINTS. The array $\mathbf{A}$ is used as a drawing board to draw one closed loop per variable and one *clause gadget* per clause. (c) Part of a variable loop. There are only two ways to cover all negative cells in the loop with a minimal 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.

Corollary 1. *The problem defined in Section 3.4 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 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. □

10 Conclusion

In this paper, we have presented a fully automatic approach to position cutaways based on a degree of interest specified by the user. Even though we do not exploit it in this paper, the method is general enough to solve a wide range of view-dependent optimization problems. One benefit of the approach is the fact that it places simple, comprehensible cutaway shapes such as boxes or spheres and does not need to remove occluding geometry more or less arbitrarily, which can be difficult to perceive. In this paper we speak in favor of a less-interaction-is-better approach, but only in the sense that rendering parameters such as the position of cutaway boxes should not have to be specified by the user. 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 believe that using known geometries eases perception.

We also show that the problem of placing cutaways optimally is NP-hard. It is the general belief today that no efficient algorithm for its solution will be found, even though this has not been proven yet. What are we supposed to do? In many cases it is a good approach to come up with good heuristics and hope that a good solution will arise for many relevant inputs. In this paper we suggest another solution based on a randomization strategy and show that it is able to find 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 as well. 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.

Acknowledgements

This work was funded by the Swiss National Science Foundation under grant 200021_127022, the ETH grant 12 09-3, and grant 226042 (SemSeg) from the Future and Emerging Technologies program of the European Commission.

References

[1] G. Barequet, M. Dickerson, and D. Eppstein. On triangulating three-dimensional polygons. *Computational Geometry: Theory and Applications*, 10(3):155–170, 1998.

[2] R. A. Becker and W. S. Cleveland. Brushing scatterplots. *Technometrics*, 29(2):127–142, 1987.

[3] Å. Birkeland and I. Viola. View-dependent peel-away visualization for volumetric data. In *Proc. Spring Conference on Computer Graphics*, pages 133–139, 2009.

[4] L. Bonanni, X. Xiao, M. Hockenberry, P. Subramani, H. Ishii, M. Seracini, and J. Schulze. Wetpaint: scraping through multi-layered images. In *Proc. 27th International Conference on Human Factors in Computing Systems*, pages 571–574, 2009.

[5] R. Bürger, P. Muigg, M. Ilčík, H. Doleisch, and H. Hauser. Integrating local feature detectors in the interactive visual analysis of flow simulation data. In *Proc. Eurographics/IEEE-VGTC Symposium on Visualization 2007*, pages 171–178, 2007.

[6] M. Burns, M. Haidacher, W. Wein, I. Viola, and E. Gröller. Feature emphasis and contextual cutaways for multimodal medical visualization. In *Proc. Eurographics/IEEE-VGTC Symposium on Visualization 2007*, pages 275–282, 2007.

[7] C.-K. Chen, R. Thomason, and K.-L. Ma. Intelligent focus+context volume visualization. In *Proc. 8th international Conference on Intelligent Systems Design and Applications*, pages 368–374, 2008.

[8] C. Coffin and T. Höllerer. Interactive perspective cut-away views for general 3d scenes. In *Proc. IEEE Symposium on 3D User Interfaces*, pages 25–28, 2006.

[9] M. Cooper. The tractability of segmentation and scene analysis. *International Journal of Computer Vision*, 30(1):27–42, 1998.

[10] C. D. Correa and K.-L. Ma. Visibility histograms and visibility-driven transfer functions. *IEEE Transactions on Visualization and Computer Graphics*, 17:192–204, 2011.

[11] H. Doleisch, M. Gasser, and H. Hauser. Interactive feature specification for focus+context visualization of complex simulation data. In *Proc. Symposium on Data Visualisation 2003*, pages 239–248, 2003.

[12] Engine Model `http://www.turbosquid.com/3d-models/free-cylinder-intake-3d-model/369057` (last visit 1. Aug. 2011).

[13] S. K. Feiner and D. D. Seligmann. Cutaways and ghosting: satisfying visibility constraints in dynamic 3D illustrations. *The Visual Computer*, 8(5):292–302, 1992.

[14] O. Frederich, J. Scouten, D. M. Luchtenburg, and F. Thiele. Large-scale dynamics in the flow around a finite cylinder with a ground plate. *Fluid Dynamics Research*, 43(1):015504:1–015504:22, 2011.

[15] R. Fuchs, V. Welker, and J. Hornegger. Non-convex polyhedral volume of interest selection. *Computerized Medical Imaging and Graphics*, 34(2):105–113, 2010.

[16] M. Harris, S. Sengupta, and J. D. Owens. Parallel Prefix Sum (Scan) with CUDA. In *GPU Gems 3*, chapter 39, pages 851–876. Addison Wesley, 2007.

[17] J. Jeong and F. Hussain. On the identification of a vortex. *Journal of Fluid Mechanics*, 285:69–94, 1995.

[18] S. Khanna, S. Muthukrishnan, and M. Paterson. On approximating rectangle tiling and packing. In *Proc. 9th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, 1998.

[19] S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, 34(5):975–986, 1984.

[20] S. Kirkpatrick, D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[21] W. Li, L. Ritter, M. Agrawala, B. Curless, and D. Salesin. Interactive cutaway illustrations of complex 3D models. *ACM Transactions on Graphics - Proceedings of ACM SIGGRAPH 2007*, 26(3):1–11, 2007.

[22] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

[23] M. J. McGuffin, L. Tancau, and R. Balakrishnan. Using deformations for browsing volumetric data. In *Proceedings of IEEE Visualization 2003*, pages 401–408, 2003.

[24] T. McInerney and P. Crawford. Ribbonview: interactive context-preserving cutaways of anatomical surface meshes. In *Proceedings of the 6th international Conference on Advances in Visual Computing*, pages 533–544. Springer, Berlin, Germany, 2010.

[25] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Computational Physics*, 21:1087, 1953.

[26] W. Mulzer. Minimum weight triangulation is np-hard. In *Proceedings of the 22nd annual Symposium on Computational Geometry*, pages 1–10. ACM, New York, NY, USA, 2006.

[27] NVIDIA CUDA: Compute Unified Device Architecture, 2011.

[28] NVIDIA CUDA SDK: Post-Process in OpenGL, 2011.

[29] C. A. Pelizzari, R. Grzeszczuk, G. T. Y. Chen, R. Heimann, D. J. Haraf, S. Vijayakumar, and M. J. Ryan. Volumetric visualization of anatomy for treatment planning. *International Journal of Radiation Oncology*, 34(1):205 – 211, 1996.

[30] B. Schindler, R. Fuchs, J. Waser, and R. Peikert. Marching correctors – fast and precise polygonal isosurfaces of SPH data. In *Proc. of the 6th International SPHERIC workshop*, pages 125–132, 2011.

[31] W. M. van Rees, A. Leonard, D. Pullin, and P. Koumoutsakos. A comparison of vortex and pseudo-spectral methods for the simulation of periodic vortical flows at high Reynolds numbers. *Journal of Computational Physics*, 230(8):2794–2805, 2011.

[32] I. Viola, A. Kanitsar, and M. E. Gröller. Importance-driven feature enhancement in volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):408–418, 2005.

[33] D. Weiskopf, K. Engel, and T. Ertl. Interactive clipping techniques for texture-based volume visualization and volume shading. *IEEE Trans. on Visualization and Computer Graphics*, 9(3):298–312, 2003.

[34] Y. Wu and H. Qu. Interactive transfer function design based on editing direct volume rendered images. *IEEE Transactions on Visualization and Computer Graphics*, 13:1027–1040, 2007.