Eurographics Conference on Visualization (EuroVis) (2014)
N. Elmqvist, M. Hlawitschka, and J. Kennedy (Editors)

Visibility-Driven Depth Determination of Surface Patches in Direct Volume Rendering

Sergej Stoppel[1,3], Hans-Christian Hege[3] and Alexander Wiebel[2,3]

[1]Freie Universität Berlin, Germany, [2]Coburg University of Applied Sciences, Germany, [3]Zuse Institute Berlin (ZIB), Germany

Abstract

This paper presents an algorithm called surfseek *for selecting surfaces on the most visible features in direct volume rendering (DVR). The algorithm is based on a previously published technique (WYSIWYP) for picking 3D locations in DVR. The new algorithm projects a surface patch on the DVR image, consisting of multiple rays. For each ray the algorithm uses WYSIWYP or a variant of it to find the candidates for the most visible locations along the ray. Using these candidates the algorithm constructs a graph and computes a minimum cut on this graph. The minimum cut represents a visible and typically rather smooth surface. In the last step the selected surface is displayed. We provide examples for results using artificially generated and real-world data sets.*

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

1. Introduction

Direct volume rendering (DVR) [Sab88] is the state-of-the-art method for displaying complex volumetric data in medicine, engineering, and natural sciences [PB07]. DVR gives an excellent overview of the data, but sometimes the perception of distinct features can be hard. Features might cover themselves partly, and the sometimes "foggy" nature of DVR impedes the perception of depth. Motivated by these problems, the goal of this paper is to provide an effective technique that allows the user to select and highlight surfaces of features appearing most visible to the user.

In the following we will introduce an effective algorithm that computes a surface on the most visible features by casting multiple rays through the data. This mimics the original DVR creation process. For each ray the algorithm detects feature boundary points using WYSIWYP [WVFH12] or a similar criterion. Using the detected boundaries, the algorithm constructs a weighted graph and computes its minimal cut, from which it reconstructs the desired surface. We do not aim to replace surface determination techniques acting on the scalar data itself like, e.g. [ONI05], but rather see (*surfseek*) presented here as a complementary method that allows the user to intuitively select the surface that is most visible with the chosen transfer function.

2. Related Work

The visibility criterion of our approach are based on the single point picking technique WYSIWYP by Wiebel et al. [WVFH12]. While WYSIWYP detects the *one* location belonging to the most visible feature along one ray, *surfseek* computes a surface using the information about visually prominent features along multiple rays. A technique that can be considered to be a step in between the two approaches is *VisiTrace* [WPVH13], which determines a 3D line from of a 2D stroke on a DVR image. The line is constructed such that it runs either on top of or inside the most visible features of the DVR. Other related techniques also extracting features from strokes are those by Yu et al. [YEII12], Owada et al. [ONI05] [ONI*08] and Liu et al. [LSS09]. These techniques aim at selecting or segmenting objects in the scalar data itself whereas our technique works on what will be used for rendering after applying the DVR transfer function.

Li et al. [LWCS06] present a surface segmentation using a non-trivial graph construction. Similar to our approach, a min-cut encoding the desired surface is computed on the graph. The skeleton of the graph exhibits a highly regular structure (reflecting voxel columns of the dataset). The positions of surface vertices along the columns can be constrained by a user-chosen smoothness parameter. Thus the smoothness of the computed surface can be controlled very easily, but the algorithm neglects all surfaces, that do not fulfill the prescribed constraint. Moreover, the algorithm does not work with irregular graphs. The latter, however, is needed for the approach we propose. Although also working with graphs, Grady [Gra06] proposes a different approach. His algorithm takes closed contours, computed in 2D slices, as input. Using linear programming he then computes the

DOI: 10.2312/eurovisshort.20141164

minimal surface. The drawback of this approach is the isolated treatment of individual 2D slices.

With *surfseek* we take an approach that is different to these methods. Instead of constructing a graph from all sample points along a ray, we only use the detected feature boundary points as nodes in the graph. This is advantageous as the result is a significantly smaller graph. Furthermore, our approach does not require a regular distribution of nodes.

3. Surfseek

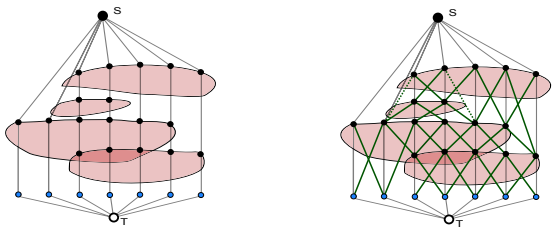
The purpose of the *surfseek* algorithm is to find surfaces on top of the most visible features in a DVR image. The algorithm searches for greatest influences to the resulting DVR pixels in the opacity values of the samples on the rays used for performing the DVR. It consists of the following steps:

1. The user selects an area on the screen. The selected pixels are transformed into world coordinates and parallel viewing rays are cast for each of the stored coordinates.
2. Along each ray the algorithm detects feature boundary points using the WYSIWYP [WVFH12] algorithm (or a similar algorithm). All points and the opacity contributions α_{acc} of the corresponding features are stored.
3. Using the stored values, the algorithm constructs a network-graph with the boundary points being the nodes and constructs the edges according to specific criteria. All edges receive certain weights that depend on the stored opacity contribution α_{acc} and position values. The details of this step are explained in section 3.1.
4. The max-flow algorithm of Boykov and Kolmogorov [BK04] as implemented in the boost library [Die06] is applied to the graph to compute the maximal flow. Using the max-flow-min-cut theorem [EFS56], the minimal cut is computed afterwards.
5. A surface is reconstructed using the nodes of the minimal cut, by creating triangles of adjacent nodes.
6. The computed surface is displayed.

The third step is the heart of *surfseek*, thus we will in the following concentrate on this particular step.

3.1. Graph Construction

In the description below we use the following objects: A vertex set V, an edge set E and an edge weight function $w(e)$: $E \to \mathbb{R}$. The weight function consists of two components: The first component w^{op} reflects the opacity contribution and the second component w^{dist} ensures the smoothness of the surface. When selecting a weight function we have to balance those components. If the influence of the opacity weight is too strong the computed surface becomes jaggy, since always features with the largest opacity contribution are selected and not the surface of one single feature. On the other hand, if the opacity weight is insignificant the algorithm neglects curved or irregular surfaces.



(a) (b)

Figure 1: *Examples of a possible graph built up to the intra-ray-edges (a) and inter-ray-edges (b). The marked regions represent different features in the data, seen from the side.*

We assume the rays to be ordered by indices in x and y direction because they start at pixels of the rendered DVR image. As vertex set V we define the set of all positions lying on the front boundary of a detected feature along a ray. A virtual vertex $\tilde{\mathbf{v}}_{x,y}$ is added to V after the last detected feature on each ray $r_{x,y}$. We explain the definition and the purpose of $\tilde{\mathbf{v}}_{x,y}$ when describing the surface construction. Finally, one source vertex S and one sink vertex T vertex are added to V. Edges in E can be subdivided into two categories: intra-ray-edges and inter-ray-edges. Intra-ray-edges connect vertices that lie on the same ray, inter-ray-edges connect vertices on adjacent rays. Rays are adjacent if they differ in one index by one, i.e. ray $r_{x,y}$ is adjacent to $r_{x\pm1,y}$ and $r_{x,y\pm1}$.

The construction of intra-ray-edges is simple: We connect each vertex $\mathbf{v}_{x,y,k}$ on a ray $r_{x,y}$ with the following one $\mathbf{v}_{x,y,k+1}$, up to $\tilde{\mathbf{v}}_{x,y}$. In order to obtain a connected graph we connect every first vertex $\mathbf{v}_{x,y,0}$ of a ray with the source S and all $\tilde{\mathbf{v}}_{x,y}$ with the sink T. Each intra-ray-edge $e_{x,y,k} = (\mathbf{v}_{x,y,k}, \mathbf{v}_{x,y,k+1})$ receives the weight $w^{op}(e_{x,y,k}) = w^{op}(\mathbf{v}_{x,y,k})$, where

$$w^{op}(\mathbf{v}_{x,y,k}) = 1 - \text{" } \alpha_{acc} \text{ of } k\text{-th feature on } r_{x,y}\text{"}. \quad (1)$$

Note that this is only a first reasonable approximation that will be refined later (see Eq. (2)). The edges that connect S and T with any other vertices of the graph receive an infinite weight. Up to this point the graph looks like shown in Fig. 1(a). The black vertices denote the vertices computed from the boundary position, the blue vertices are the virtual $\tilde{\mathbf{v}}_{x,y}$ at the end of each ray. The bigger vertices represent the source S and the sink T.

The construction of inter-ray-edges requires more effort. The inter-ray-edges will receive a combined weight of opacity contribution and the Euclidean distance of the edge vertices. For an edge $e = (\mathbf{v}_{x,y,k}, \mathbf{v}_{x',y',k'})$ we denote the weight of the opacity contribution as $w^{op}(e)$ and compute it as

$$w^{op}(e) = \frac{w^{op}(\mathbf{v}_{x,y,k}) + w^{op}(\mathbf{v}_{x',y',k'})}{2},$$

with ray $r_{x,y}$ being adjacent to $r_{x',y'}$. The distance weight is

$$w^{dist}(e) = \|\mathbf{v}_{x,y,z} - \mathbf{v}_{x',y',z'}\|.$$

We normalize all weights at the end. The final weight is then

$$w(e) = \left(w^{dist}(e)\right)^m \cdot w^{op}(e)$$

where we have chosen $m = 3$ throughout this paper. Hereby we gave the distance weight a greater leverage in order to increase the smoothness of the computed surfaces. Using this definition we can now construct the inter-ray-edges: For each vertex $\mathbf{v}_{x,y,k}$ on a ray $r_{x,y}$ find for each adjacent ray a vertex $\mathbf{v}_{x',y',k'}$ that minimizes the edge weight, that is

$$w(e_{k,k'}) = w(\mathbf{v}_{x,y,k}, \mathbf{v}_{x',y',k'}) = \min_j \{w(\mathbf{v}_{x,y,k}, \mathbf{v}_{x',y',j})\}.$$

This ensures the vertices on the adjacent rays to lie deeper in the volume (farther from the observer), otherwise they will be ignored. Additionally, if the vertex $\mathbf{v}_{x',y',k'}$ is already *connected* to a vertex with $\mathbf{v}_{x,y,\bar{k}}$ and $\bar{k} < k$ on ray $r_{x,y}$, compare the weights $w(e_{k,k'})$ and delete the edge with the bigger weight. If the vertex $\mathbf{v}_{x',y',k'}$ is *not connected* to ray $r_{x,y}$, add the edge $e = (\mathbf{v}_{x,y,k}, \mathbf{v}_{x',y',k'+1})$ to the graph and assign it the weight $w(e_{k,k'})$. An example of a possible graph is shown in Fig. 1(b). The green edges denote the inter-ray edges, the dotted green edges represent the removed inter-ray edges.

When a ray does not cross any features, the only vertex on this ray, i.e. the virtual vertex, is removed. At this point the graph has the desired vertices and edges. However there may be vertices with no adjacent inter-ray-edges, i.e. only two intra-ray-edges. Clearly, the min-cut algorithm would favour to cut the edges between those vertices, since it would stop the flow. To avoid this, we have to adjust the weights of the intra-ray-edges defined in Eq. (1) to

$$w(e_k) = \frac{1 - w^{op}(\mathbf{v}_k)}{con(\mathbf{v}_k) + 1}, \tag{2}$$

where $con(\mathbf{v}_k) =$ "# of edges from $\mathbf{v}_k$ to adjacent rays".

3.2. Surface Computation

After the graph is ready, the minimal edge-cut is computed using the Boykov-Kolmogorov algorithm [BK04]. The desired surface is reconstructed by going along each ray until an edge $e_{k,k+1}$ is found that is cut. The vertex $\mathbf{v}_k$ is then added to the surface vertices. Here the virtual vertices $\tilde{\mathbf{v}}_{x,y}$ come into play. They result in edges behind the back-most features. Thus making also the vertices of the back-most features available for the surface. A surface can be computed by triangulation of surface vertices lying on adjacent rays.

4. Results

In the following we will show a few examples of surfaces computed by our algorithm. We distinguish between experimental data (CT, computed tomography scans) and synthetic datasets. Note that the views showing the surfaces are rotated from the original view in which the 2D area was specified. This allows to get an impression of the three-dimensional nature of the surfaces which would be impossible in the original view. We use rectangular areas in all our examples because it makes the figures easier to understand, although the method works for any connected 2D region of pixels. To demonstrate the superiority of *surfseek* we will compare its results to surfaces computed using the WYSIWYP criterion only, i.e. without graph optimization.

(a) (b) (c)

Figure 2: *Examples of surfaces computed with* surfseek *on a simple ball without noise (a) and with a random perturbation of 35 % of the ball density (b), and a surface computed without the graph optimization (c).*

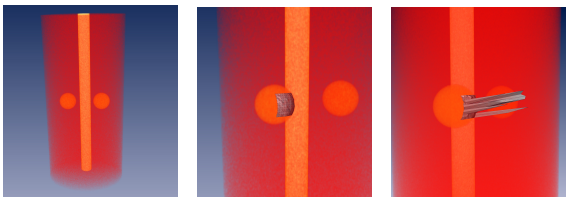
The running time of the algorithm is bounded from above by the worst case complexity of the Boykov-Kolmogorov max-flow algorithm with the pseudo-polynomial bound $O(n^2mC)$ [BK04]. Where n is the number of nodes and m the number of edges. For all examples presented in this paper the computation time lay in the order of seconds.

4.1. Synthetic Datasets

We consider two different synthetic datasets: A simple ball with and without noise, and a "synthetic torso" dataset consisting of two balls in a volume with slightly lower density than the balls. The synthetic torso dataset was perturbed with Gaussian noise and subsequently smoothed to imitate perturbations that occur CT acuqisition.

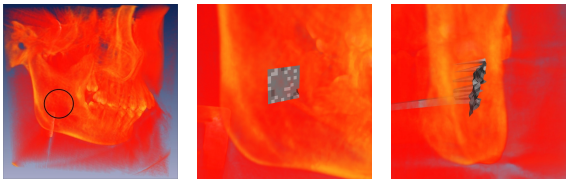
The dataset with the simple ball serves only as a validation that the algorithm in fact can detect a surface. The ball was created with the a diameter of 1 and constant density. The result of the algorithm on this ball is shown in Fig. 2(a). If we add noise to the dataset, the algorithm starts to detect some non-existing features inside of the foggy area, thus the direct picking jumps between those spurious features, see Fig. 2(c). The additive Gaussian noise has a maximal amplitude of 35 percent of the ball density and a standard deviation $\sigma = 1$ (5% of density). The surface obtained using *surfseek* can be observed in Fig. 2(b). The ball is covered in a thick foggy area but it is still the only recognizable feature in the dataset.

The "synthetics torso" can be observed in Fig. 3(a). In this dataset we used similar density values as the human bone and kidney tissue to simulate the backbone and the kidney. The torso was simulated using the average density of the internal organs. The dimension of the torso is $50cm \times 35cm \times 25cm$. Furthermore we added Gaussian noise with a maximal amplitude of 1, ca. 20 % of the bone density,



(a) (b) (c)

Figure 3: *An example of an artificially created dataset inspired by the human torso with noise and blurring (a) and a surface computed in this dataset: (b) computed with* surfseek *and (c) computed without graph optimization.*



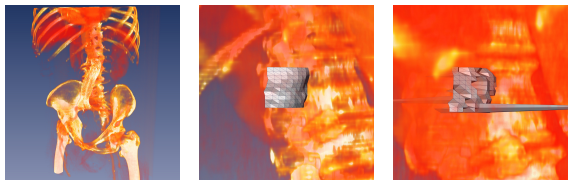
(a) (b) (c)

Figure 4: *Example of DVR image of a CT-scan of a human skull (a) with a computed surface on the jaw: (b) computed with* surfseek *and (c) computed without graph optimization.*

and $\sigma = 1$ mm. Afterwards the dataset has been blurred with $\sigma = 2$ mm. The surface computed by *surfseek* can be observed in Fig. 3(b). A surface computed without graph optimization is depicted in Fig. 3(c). We conclude that *surfseek* handles synthetic data very well.

4.2. Experimental Datasets

We discuss results for two experimental datasets from volvis.org: A CT of a human skull anda torso of a male human. The datasets are shown in Figs. 4(a) and 5(a). We marked the thinner bone tissue of the lower jaw in the skull dataset (see circle in Fig. 4(a)). Since this area is significantly thinner, compared to the surrounding bone tissue, it is harder to detect. The resulting surface can be observed in Fig. 4(b). Looking closely, one can see that the surface has a miscalculated point on the lower right border of the surface patch. In this particular ray the jaw was not detected as a feature. This can happen in a foggy dataset when the features have low or changing density. We will come back to this type of problem in the discussion. In the torso dataset we tried to select the surface of the right kidney. Here, the challenge is that the rib behind the kidney is visible through the kidney, and that the kidney has small regions that are almost completely transparent. The computed surface is shown in Fig. 5(b). The reader can observe that the surface lies on the kidney. It reveals the shape of the kidney in a clearer way than it was possible with just the DVR image. Without the graph optimization the selected surfaces will look like in Fig. 4(c) for the skull dataset. The two peaks in this surface are good representations of the challenges the graph construction had to overcome. The one peak, showing to the user, arises of a small feature with a high opacity contribution and thus is immediately selected by WYSIWYP. The other peak lies in an areas of the kidney that has relatively small opacity contribution, thus the algorithm WYSIWYP selects correctly the better visible bone for this particular ray.



(a) (b) (c)

Figure 5: *Example of DVR image of a CT-scan of a human torso (a) with a computed surface: (b) computed with* surfseek *and (c) computed without graph optimization.*

5. Discussion: Limitations and Outlook

In this paper we presented an effective algorithm that allows the users to intuitively select surfaces on the features that appear most visible to them in a DVR. We showed that the algorithm can handle both Gaussian noise and blurring very well, and showed its application to experimental data.

While the algorithm introduces a completely new type of user interaction for DVR, its applicability is limited as follows: First, when the opacity of a feature becomes very low or the feature becomes very thin, the algorithm may fail to detect its boundary. This can cause the surface to jump to features behind or in front of the desired feature. This may be undesired by some users but it reflects the fact that the feature in the back is more visible. There is always a compromise between selecting the most visible and the smoothest surface. Second, although the algorithm is robust against noise, very "foggy" dataset can cause problems. When we try to compute a surface patch for a vaguely visible feature with a very "foggy" area in front of it, the algorithm can detect some false points. Sometimes the algorithm does not detect the beginning of a feature because the foggy area reaches up to the wanted feature. Finally, if there are strong density changes inside a feature, it is questionable if the feature can be considered a connected "whole". The differences influence the opacity values of the feature. The result is that the algorithm divides the feature into several sub-features and thus computes a surface that can lie on top of a sub-feature instead of on the "whole" feature. A detailed demonstration of the method's tolerance to noise and a discussion of possible alternatives to the WYSIWYP criterion (e.g. [KBKG09]) can be found in a Master's thesis on the topic [Sto13]. The thesis also discusses possible post-filtering methods.

Acknowledgments All visualizations were produced with Amira [SWH05].

References

[BK04] BOYKOV Y., KOLMOGOROV V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 9 (Sept. 2004), 1124–1137. doi:10.1109/TPAMI.2004.60. 2, 3

[Die06] DIEDERICH S.: Boost graph library: Boykov-Kolmogorov maximum flow - 1.55.0. http://www.boost.org/doc/libs/1_55_0/libs/graph/doc/boykov_kolmogorov_max_flow.html, 2006. Accessed 2014-03-03. 2

[EFS56] ELIAS P., FEINSTEIN A., SHANNON C.: A note on the maximum flow through a network. *IRE Transactions on Information Theory 2*, 4 (December 1956), 117–119. doi:10.1109/TIT.1956.1056816. 2

[Gra06] GRADY L.: Computing exact discrete minimal surfaces: Extending and solving the shortest path problem in 3d with application to segmentation. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)* (2006), pp. 69–78. doi:10.1109/CVPR.2006.82. 1

[KBKG09] KOHLMANN P., BRUCKNER S., KANITSAR A., GRÖLLER M. E.: Contextual picking of volumetric structures. In *Proceedings of the IEEE Pacific Visualization Symposium 2009* (May 2009), Eades P., Ertl T., Shen H.-W., (Eds.), IEEE Computer Society, pp. 185–192. 4

[LSS09] LIU J., SUN J., SHUM H.-Y.: Paint selection. *ACM Transactions on Graphics 28*, 3 (July 2009), 69:1–69:7. doi:10.1145/1531326.1531375. 1

[LWCS06] LI K., WU X., CHEN D. Z., SONKA M.: Optimal surface segmentation in volumetrix images - a graph-theoretic approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 1 (January 2006), 119 – 134. doi:10.1109/TPAMI.2006.19. 1

[ONI05] OWADA S., NIELSEN F., IGARASHI T.: Volume catcher. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2005), I3D '05, ACM, pp. 111–116. doi:10.1145/1053427.1053445. 1

[ONI*08] OWADA S., NIELSEN F., IGARASHI T., HARAGUCHI R., NAKAZAWA K.: Projection plane processing for sketch-based volume segmentation. In *International Symposium on Biomedical Imaging* (2008), IEEE, pp. 117–120. doi:10.1109/ISBI.2008.4540946. 1

[PB07] PREIM B., BARTZ D.: *Visualization in Medicine: Theory, Algorithms, and Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007. 1

[Sab88] SABELLA P.: A rendering algorithm for visualizing 3D scalar fields. *SIGGRAPH Computer Graphics 22*, 4 (June 1988), 51–58. doi:10.1145/378456.378476. 1

[Sto13] STOPPEL S.: *Visibility-Driven Depth Determination of Surface Patches in Direct Volume Rendering*. Master's thesis, Fachbereich Mathematik und Informatik, Freie Universität Berlin, 2013. http://research.awmw.org/PDF/stoppelMA.pdf. 4

[SWH05] STALLING D., WESTERHOFF M., HEGE H.-C.: Amira: A highly interactive system for visual data analysis. In *The Visualization Handbook*, Hansen C. D., Johnson C. R., (Eds.). Elsevier, 2005, pp. 749–767. 4

[WPVH13] WIEBEL A., PREIS P., VOS F. M., HEGE H.-C.: 3d strokes on visible structures in direct volume rendering. In *EuroVis Shortpapers* (June 2013), Eurographics Association, pp. 91–95. doi:10.2312/PE.EuroVisShort.EuroVisShort2013.091-095. 1

[WVFH12] WIEBEL A., VOS F. M., FOERSTER D., HEGE H.-C.: WYSIWYP: What you see is what you pick. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (December 2012), 2236–2244. doi:10.1109/TVCG.2012.292. 1, 2

[YEII12] YU L., EFSTATHIOU K., ISENBERG P., ISENBERG T.: Efficient structure-aware selection techniques for 3D point cloud visualizations with 2DOF input. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (December 2012), 2245–2254. doi:10.1109/TVCG.2012.217. 1