Interactive Volume Illustration

Zoltán Nagy, Jens Schneider, Rüdiger Westermann

Scientific Visualization & Imaging Group
University of Technology Aachen*

Abstract

In this paper we describe non-photorealistic rendering techniques for volumetric data sets. First, we outline an automatic approach that generates line drawings to illustrate such data sets and to augment traditional volume rendering techniques. For a number of seed points that are placed appropriately to represent selected volume structures curvature lines are traced and encoded by a sparse set of control points. These curves are finally drawn as hatching strokes modulated by anisotropic lighting and transparency. Second, in addition to line-art drawings we present efficient implementations of volume toon-shading and silhouette rendering using fragment shader hardware. All techniques together allow us to interactively illustrate volumetric data sets and to enhance important features using non-photorealistic rendering techniques.

1 Introduction

In this paper we consider volume illustrations as a class of non-photorealistic rendering (NPR) techniques with the particular characteristic to emphasize important attributes or parts of an object and to communicate relevant information to the viewer in the most effective way. It is quite interesting that this characterization exactly matches the demands on visualization techniques in general. In scientific visualization the grant challenge is to convey the relevant information in the most comprehensible but not necessarily in the most realistic way. With regard to that observation NPR techniques seem to be the predestined rendering tool in visualization applications, and it comes to no surprise that NPR techniques have already attracted the visualization community during the last couple of years.

* University of Technology, Aachen, Seffenter Weg 23, 52056 Aachen, Germany, Phone:+49-(0)241-80-28920, Fax:+49-(0)241-80-22241, e-mail: nagy@sc.rwth-aachen.de

In this paper our emphasis is on proposing interactive approaches to non-photorealistic volume illustration. Our first technique creates hatching fields coinciding with the principal curvature directions along selected volume structures. Creating the hatching field and rendering the strokes is performed in two separate passes: In the first pass higher order differential characteristics of the volumetric data field are computed and encoded in a hierarchical data structure. At run time, based on some user-defined importance criterion a representative set of hatching strokes is computed, each of which is effectively encoded by a connected group of line segments. Strokes are finally displayed as colored and shaded line strips employing OpenGL functionality and the anisotropic model proposed by Banks [1].

Our choice of techniques was mainly driven by two requirements. First, in our opinion line drawings should not be integrated into the volume representation itself as proposed by [17] thus prohibiting flexible and efficient modification of the strokes appearance and arrangement. Second, techniques should not rely on any polygonal representation thus limiting its potential use to geometric objects. Rather than that we aim at proposing methods that are capable of interactively illustrating arbitrary structures in volumetric data sets without the need to generate intermediate surface representations at run time. Moreover we want to demonstrate that interactive non-photorealistic rendering is possible even for large-scale volumetric data sets thus spawning a promising new direction in scientific visualization.

In addition to volume hatching we present non-photorealistic volume rendering techniques via three-dimensional texture maps and fragment shader hardware. Dedicated shaders have been developed that enable iso-surface toon-shading, silhouette rendering and view-dependent opacity enhancement at interactive rates in one single render-

ing pass. high resolution data sets.

The remainder of this paper is organized as follows. In Chapter 2 we review related work with the focus on NPR techniques that are related to our approach. The procedure to compute principal curvature directions in volumetric data sets is described in Chapter 3. Chapter 4 is dedicated to interactive NPR techniques for volumetric data sets. Future directions and alternative issues are subject of Chapter 5.

2 Related work

In the past a number of different approaches have been carried out to simulate the imagery generated by artists and technical writers in an automatic and computer-assisted way. Since in our current work we mainly concentrate on the simulation of line drawings, we will only refer to those attempts that are directly related to ours. For an excellent introduction to and a comprehensive survey of NPR techniques in general let us refer to the book of Gooch and Gooch [12] and to the web-page maintained by Reynolds [29], where many related online resources are given.

An image-based system for generating computer-aided pen-and-ink illustrations using oriented stroke textures was proposed in [33, 34]. In the final version stroke textures conveying color, tone and orientation were generated automatically from a set of representative strokes and a user-controlled direction field. A digital engraving system based on lines coinciding with potential fields in the image was presented in [27]. Rössl and Kobbelt [31] described a semi-automatic image-based technique for generating technical illustrations of 3D models, where the user has to manually select partitions in image space exhibiting coherent curvature direction to generate appropriate hatching fields. A real-time rendering system was described in Lake et al. [22], where a palette of textures representing different tones by different pencil strokes is precomputed and used to cover the object. However, due to the employed viewport mapping to generate texture coordinates from viewport coordinates, the textures seem to stay above the object thus limiting the techniques suitability for animations.

In [5] attributed lines in object space were used to augment traditional renderings with regard to user-defined importance of information. In [39, 8, 9, 15] 3D curves on free-form surfaces and parametric or implicit surfaces were introduced to emphasize geometric properties of 3D models. Winkenbach and Salesin [38] also presented an object-based approach, in which oriented strokes were utilized to simulating different tones. Markosian et al. [25] employed line drawings for interactive NPR rendering. The focus in this work, however, was on enhancing silhouettes and cusps by only placing a sparse set of appropriately selected strokes. Finally, Praun et al. [28] employed lapped multi-textures to achieve real-time hatching. They extended on the concept of art maps described in [20] by using a set of oriented stroke textures each of which represents different tones. In visualization NPR techniques have been considered to substitute or to enhance traditional rendering techniques. Saito [32] described a point-based rendering system to allow for fast previewing of volumetric data sets. Expressive textures have been used in [30, 18] to improve the understanding of the shape of complex structures. Interrante [17] developed techniques to enhanced spatial and shape information of transparent surfaces in volumetric data sets by constructing patterns of thin opaque lines. In order to accelerate volume rendering and to enhance the insight into complex structures Csébfalvi et al. [3] proposed a contour based visualization technique. Kirby et al. [19] described a technique closely related to oil painting, which effectively enhances the information content in 2D images by using multiple layers covering different kinds of features. An image-based approach for simulating pen-and-ink drawings to augment volume rendering was introduced by Treavett and Chen [36]. Ebert and Rheingans [7] proposed a modified volume rendering pipeline that is amenable to a variety of different NPR techniques, which can be used to further enhance traditional rendering methods. The visualization of tensor data by means of brush strokes was illustrated by Laidlaw et al. [21]. Here, a brushing technique was used to stress directional information and to guide the user towards the orientation of electrocardial fields.

3 Preprocess

We now start with a description of the preprocess that has to be performed to generate accurate principal curvature directions in three-dimensional scalar data fields.

3.1 Discrete curvature estimation

Although principal curvature directions can be derived from the extremal values of a quadratic form [6], this technique is impractical in the current scenario due to its numerical complexity. Monga *et al.*[26] derived a considerably faster method for computing the principal directions in three-dimensional scalar fields. First, an orthonormal basis for ε is computed explicitly. Therefore, let $G = \nabla F = (g_0, g_1, g_2)^t$ and $\gamma = \sqrt{g_0^2 + g_1^2}$. Then an orthonormal matrix P can be given that rotates the first basis vector into the direction of G:

$$P = \begin{pmatrix} \frac{g_0}{\|G\|} & \frac{g_1}{\gamma} & \frac{g_2 \cdot g_0}{\gamma \cdot \|G\|} \\ \frac{g_1}{\|G\|} & \frac{-g_0}{\gamma} & \frac{g_2 \cdot g_1}{\gamma \cdot \|G\|} \\ \frac{g_2}{\|G\|} & 0 & \frac{\gamma}{\|G\|} \end{pmatrix} = \left(\frac{G}{\|G\|}, h, f \right)$$

The case $\gamma = 0$ has to be treated separately and results in a standard basis.

Any v in $\{G\}^{\perp}$ can then be expressed as

$$v = h \cdot \cos\theta + f \cdot \sin\theta$$

If θ corresponds to a principal curvature direction then the first derivative of κ_v with respect to θ vanishes. Finally this yields

$$\tan 2\theta = \frac{2 \cdot h^t (D^2 F) f}{h^t (D^2 F) h - f^t (D^2 F) f}$$

which can be easily solved. Special attention has to be paid at umbilical points where the surface is locally flat or spherical. Such points occur if and only if the denominator vanishes. One approach to obtain meaningful principal curvatures at such points is to increase the filter width of the differentiation operator.

Besides the possibility of computing principal curvature directions in three dimensions, Mongas algorithm can be fed with any suitable method to estimate partial derivatives. Unfortunately, this technique (and any other method we know) only solves for $\theta \in (-\frac{\pi}{2}, +\frac{\pi}{2})$. At first glance this does not seem to be a major problem, but it results in curvature fields that are not oriented consistently. As a matter of fact the curvature directions might flip about 180 degrees along a particular field line. Tracing these fields in a globally consistent way, e.g. if oriented strokes should be rendered, becomes impossible without that a heuristic is used to consistently align the directions along the paths. Note that this is not a problem if hatches are generated as proposed in [17] by integrating backward and forward along the curvature field path lines. Then, along each path the flip can be performed based on the current direction of the stroke.

Global methods to continuously orient the entire field, however, fail in general. This is because in many real data sets we find curvature fields containing regions in which curvature streams with opposite direction meet each other, but which smoothly merge into each other in some other region. As a matter of fact, in these cases global methods like sweep-planes or region growing successively flip the directions back and forth but do not converge. On the other hand, optimization schemes such as conjugate gradients as employed in [28] for surface hatching are far too expensive in three dimensions.

A solution to the problem is to use a heuristic to continuously orient curvature directions during the tracing of path lines in the field by using a globally defined reference field, as it was proposed in [10] to flip first and second curvature directions. From a selection of possible choices the user selects the reference field that results in the most pleasent orientation, without that the appropriateness of such a global reference field can be guaranteed in general. In our current work a cylindrical reference field was used, which in general leads to good results.

3.2 Pyramid data structure

The principal curvature directions can now be used to generate hatching strokes that characterize structures in the volume and effectively reveal important shape information. Therefore we have to select appropriately positioned seed points that are used as starting points of the strokes. The seed points have to be arranged statically so that temporal coherence between frames can be guaranteed even if transformations are applied to either the object or the view point.

In order to minimize memory requirements, however, we do not generate any seeds in advance. Generating seed points for all possible volume structures would produce a tremendous amount of points to be encoded, and it would also generate many points that are not going to be used for hatching the currently selected structures. Instead we construct a pyramidal data structure that allows us to efficiently find those cells in the volume where seed points have to be placed at run time.

In this work we employ a pointerless octree representation, where every level has one eighth as many entries as the previous level. The first level of the pyramid is the original data. Entries at level two carry additional information about the region covered by that entry, i.e. the minimal and maximal scalar values, the maximal gradient magnitude and the maximal mean curvature $\frac{1}{2}(\kappa_1 + \kappa_2)$. These attributes are used at run time to efficiently find the set of cells that should host a seed and to determine the seed point density within a region. At levels larger than two the gradient magnitude is not going to be stored any more because a measure of the local homogeneity and the maximal curvature is sufficient to terminate the traversal of the octree. Note that all attributes are quantized to 8 bit thus keeping the additional memory as low as possible.

4 Run time

4.1 Seed point placement

To generate hatching strokes a number of seed points are strayed into the volume. Instead of a random placement we perform a data driven placement which produces an optimal arrangement of seeds. Therefore we assume that at most one seed per volume cell should be issued. On the other hand, within a certain region also fewer seed points might be selected depending on the local characteristics of the data.

When a new hatching process is initiated the octree data structure is traversed in depth-first order starting at the coarsest level. The traversal is terminated if a region turns out to be empty or when an iso-surface should be illustrated but the iso-value is not contained in this region. We also stop the traversal if the maximal mean curvature is below a constant. This is motivated by the observation that on planar or almost planar structures no meaningful principal curvature directions can be computed. Whenever the traversal is terminated on a level greater than one no seed points are placed and thus no strokes are going to be drawn in this region.

In order to determine the number of effectively placed seed points we take into account the normalized gradient magnitude and mean curvature information at level two. From both attributes we compute a value that defines the probability of placing a seed point in any non-empty cell of the 2x2x2 voxel region (κ and $|\nabla_{max}|$ are normalized to (0,1)):

$$p(\kappa, \nabla_{max}) = \begin{cases} 0 & : \quad \kappa \leq C_0 \\ 0 & : \quad |\nabla_{max}| \leq C_1 \\ \frac{1}{2}(\kappa + |\nabla_{max}|) & : \quad otherwise \end{cases}$$

Note that in case of surface illustrations a cell is supposed to be non-empty only if it contains the iso-value. As a consequence of the seeding strategy the number of selected seeds directly correlates to the gradient magnitude and the mean curvature, i.e Figure 1. As a matter of fact less points are placed in homogeneous regions exhibiting low curvature while more points are placed at highly curved boundary surfaces.

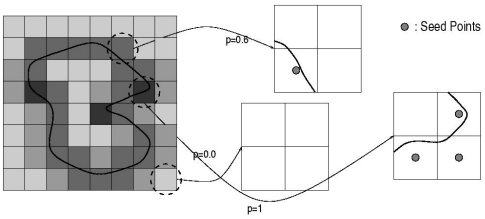

Figure 1: *We illustrate the process of determining seed point density with regard to gradient magnitude and curvature. A region is supposed to be empty if no iso-surface passes through it. Darker color corresponds to higher probability of placing a seed in a volume cells. Seed points are only placed in non-empty cells.*

The most critical step in hatching the volume is the correct placement of seed points within the selected cells. This doesn't pose a problem for transparent structures where strokes might be placed inside the structures, but for opaque objects it has to be guaranteed that the entire stroke lies in front of the object and thus survives the depth test that is applied during rendering. In our terminology a stroke lies in front of an object if the scalar values at each point defining that stroke are less than the selected iso-value. This assumption is in accordance with the use of the alpha-test that is employed in our application to discards all fragments less than the selected threshold during volume rendering (see below).

After a seed point has been selected for hatching at run time it is initially placed right in the center of the corresponding cell. At this point the scalar value and the gray-scale gradient are interpolated and the scalar value is compared with the iso-value. Depending on the result of the comparison the point

is either shifted along the gradient direction or into the opposite direction until it is positioned at a point having a scalar values that is slightly smaller than the iso-value.

4.2 Stroke generation

For each selected seed point we finally generate a stroke that is used for hatching. Strokes are generated by tracing the paths originating at a selected seed point along the maximal curvature directions in object space. The entire stroke is stored as a set of points that are rendered as line strip. At each point we also interpolate the gray-scale gradient from the scalar field in order to simulate lighting effects. From the set of all gradients a cone of normals is derived that can be used effectively to cull back facing strokes. By assuming sufficiently accurate curvature directions each stroke will be placed entirely in front of any selected surface.

In order to numerically integrate the field lines in the curvature field we employ a Runge-Kutta RK4(3) integration method with adaptive step size control. Although we only tri-linearly interpolate the curvature field the RK4(3) method allows us to compute the field lines within high accuracy. This is due to the fact that a stroke is terminated whenever the current hatch direction deviates from the original one by more than a constant. In this way we avoid stepping into regions where the curvature field shows high variations.

By means to the adaptive step size control we considerably reduce the number of intermediate points used to sample the vector field and thus to represent the strokes. As a matter of fact the strokes can be effectively encoded using only a very small number of line segments. In addition to vertex coordinates and gradients we also store texture coordinates for each point. Texture coordinates correspond to the distance of each point to the seed point relative to the total length of the stroke. As will be shown later, these attributes can be used to simulate various drawing options.

4.3 Hatching

Once the hatching strokes have been computed and stored, line art drawings to illustrate the volumetric data set or to augment traditional volume rendering techniques can be simulated. Volume hatching is split into two passes. In the first pass hatching strokes are rendered as line strips starting at the currently selected seed points. In the second pass the volumetric data set is rendered by means of three-dimensional texture maps.

4.3.1 Stroke rendering

Due to the object space representation of strokes as line strips we can render the curves using OpenGL functionality. In particular, the curves can be illuminated by means of the stored gradients and per-vertex colors can be specified to modify the strokes appearance. In addition, the appearance of each curve is modulated by applying the anisotropic shading model for lines as proposed in [1], which was extended towards real-time shading using hardware support in [35, 40, 14].

To produce additional visual cues that help to convey the positions of strokes in space we extend [14] as follows. Since for each stroke the normal of the corresponding iso-surface is known we can determine whether a line belongs to a front or back facing part of the surface. This enables us to use different colors for back and front facing parts thus emphasizing the line positions in space. In order to do so we calculate two different 2D lookup textures by varying diffuse and specular color. The two textures are mapped onto each line and the color contributions are linearly interpolated based on the dot product $N \cdot V$ between the surface normal N and the per-vertex view-vector V. Interpolation is performed using per-fragment shaders, while the dot product is calculated in software. Optionally a self-shadowing term as proposed in [14] can be integrated using the surface normal.

In particular, anisotropic shading of hatching strokes helps to enhance the visual impression in dynamic animations. Spatially coherent bundles of lines having the same orientation can be visually distinguished quite efficiently using this approach. Two-sided lighting using different colors for back and front facing lines also simplifies the visual complexity of hatching fields for high-resolution volumetric structures. If only front facing strokes or strokes on opaque iso-surfaces should be considered, however, the cone of normals can be used for culling purposes. During rendering of hatching strokes the depth test is enabled and values are updated in the depth buffer. Because line drawing is always performed first, transparent volume structures can be overlayed by rendering these struc-

tures in back-to-front order without changing the OpenGL state. Cross-hatches are used in our approach to effectively reveal spatial correlations by darkening those regions that are in shadow. They are generated as follows. For every selected seed point we consider the dot product between its normal and the light source direction to determine whether a cross-hatch should be drawn or not. If the angle is above a constant a cross-hatch is generated by creating an additional stroke that follows the minimal curvature direction. Further on, apart from being shorter than the initial strokes they are handled in exactly the same way.

4.3.2 Volume rendering

Volume rendering via 3D texture maps has become a powerful tool to interactively display scalar data fields [2]. Interpreting volume rendering as the re-sampling of a discrete 3D texture map on appropriately oriented cutting geometries allows one to exploit hardware supported texture interpolation and per-pixel blending to simulate the appearance of semi-transparent media. By rendering the hatching strokes prior to volume rendering arbitrary rendering modes can be applied. By using the depth-test and alpha-blending transparent and opaque regions will be correctly merged with the hatches. This allows us to enhance traditional volume representations or to simulate the hatching of opaque iso-surfaces. This can be achieved by combining the OpenGL alpha- and depth-test during the re-sampling procedure to guarantee that only those texture samples closest to the viewpoint and above/below a selected threshold are rendered [37]. By writing white color into the frame buffer only those hatches that were just drawn in front of the iso-surface remain visible.

4.4 Toon-shading and silhouette rendering

In this section we are going to present a couple of interactive algorithms to illustrate volumetric data sets by exploiting consumer class fragment shader hardware.

In contrast to the work presented in [24] we developed specialized fragment shaders that enable us to render the data sets even more efficiently but using the same principal algorithms. In addition, by exchanging the shaders we can arbitrarily integrate them into our hatch based approach and into the direct volume rendering algorithm via 3D texture maps.

Our first goal is to develop an algorithm that creates toon-shaded images from volumes without any special mark-up - only volume gradients and lighting parameters are required. Alternative approaches include image-space methods [4] and toon-shading algorithms based on ray-tracing, e.g. implemented in CartoonReyes, a commercial cartoon shader [16].

The general idea our algorithm is based upon has been described in [22, 23] for toon-shading polygonal objects. For every vertex the dot product $L \cdot N$ between the normal N and the light source direction L is issued as a one-dimensional texture coordinate. The texture coordinate is then used to look up colors in a toon-shading texture. As texture coordinates of a surface point are interpolated from the values of the vertices the dot product is interpolated across the surface. By using nearest neighbor texture interpolation a discontinuouity between illuminated points and points lying in shadow is achieved.

In our approach, however, we do not have any vertices at all, and shading has to be performed on a per-fragment basis using multitextures and fragment shader hardware. Therefore we utilize a three-dimensional normal map storing the gray-scale gradients in the RGB components.

We proceed by rendering the three-dimensional normal map as described. In the texture combiner the parallel light vector is issued as constant color. Using per-fragment arithmetic the dot product between this vector and the normal vector that is looked up in the three-dimensional normal map is computed. The resulting value is used to trigger a multiplexer that is available in the texture combiner stage. Depending on whether this value is larger or less than 0.5 the multiplexer outputs one of two possible registers. Both registers can be written by the user to specifying the tones that should finally be rendered. Note that an offset can be added to the input to the multiplexer thus enabling an arbitrary mapping of shades to tones.

Using the same approach also allows us to interactively render silhouettes (see [13, 11] for a good introduction). Object- or geometry based approaches for rendering silhouettes attempt to extract those edges of a polygonal mesh that belong to the silhouette. These edges are then rendered in a specific color. Because silhouette edges are connected

to one front- and one back-facing triangle with respect to the current viewpoint, a silhouette edge is defined by

$$((N_1 \cdot (C - V_1))(N_2 \cdot (C - V_2)) \leq 0$$

where N_1, N_2 are the face normals, V_1, V_2 are points that lie on face 1 and 2, respectively, and C is the eye point.

To perform the described method for volumetric data sets we proceed as follows. The three-dimensional normal map is rendered and at each vertex of the rendered polygons the normalized view vector is issued as an additional texture coordinate. In the rasterization stage coordinates are interpolated on a per-fragment basis, but because normalization is not preserved they have to be renormalized. Therefore the coordinates are used to fetch texture values from an additional cube-map. This map simply stores the normalized values for each possible coordinate. Again register combiners are employed to compute the dot product between the normalized view vector and the normal vector. Before the results can be used to trigger the multiplexer they first have to be squared to obtain positive values. Depending on the dot product between view vector and normal the multiplexer outputs black or white. Finally, the result is merged with the colors produced by the toon-shader.

5 Results and discussion

All our results were computed on a Pentium4 processor running at 1.5 GHz and a GeForce4 graphics unit with 128 MB local memory and 3D texture support. Our experiments were run on different real data sets with different illumination models, as illustrated in figures 4-6: (a) the well known engine block (256^2x128), (b) the hydrogen molecule data set (128^3) and (c) an iron proteine molecule (64^3). Over that we applied our toon shader to the aneurysm data set (256^2x128, c.f. 2) as well as to a human foot and a human skull (256^3, c.f. 3). For a 256^3 volume preprocessing the data set roughly took 1.3 minutes including curvature estimation and octree construction.

Preprocessing statistics of the seeding and path tracing procedure differ significantly depending on the number of strokes and their length. The average number of line segments used to represent the strokes in all of our examples was 10. Placing seed points to appropriately represent an iso-surfaces took between 0.5 and 4 seconds in the examples. 20K to 50K strokes were generated. Rendering the data sets (including volume rendering) was always performed with 4 to 8 fps on a 600x600 viewport. As can be seen, even for highly detailed hatches the volume can still be rendered interactively. Also the process of generating the strokes at run time consumes only very few time. Here the algorithm considerably takes advantage of the data driven seed point selection by means of which the generation of an optimal number of points is ensured. This affects both the performance of the generation process as well as the performance of the rendering process, keeping memory requirements as low as possible. In all our examples at most one eighth of the memory consumed by the volume data set was required to store the selected strokes.

Figures 4-6 exemplify various drawing options. In figure 4 the hatching strokes were lit anisotropically and superimposed over an iso-surface that was illuminated using a per-fragment phong model. In figure 5 only the hatching strokes were rendered. To help the user convey the positions of the hatchings in space, front facing lines were colored blue while back facing lines were colored yellow. In the last row (c.f. 6) we demonstrate classical artistic hatching. Cross hatches and intensity modulations are used to emphasize both lighting and shape. In all cases the rendering performance was at least 4 fps, depending on the number of strokes and the resolution of the volume.

Figure 2 illustrates our interactive volume toon shader with silhouette tracing and half-level lighting. Rendering performance was about 7 fps.

6 Conclusion

We have presented an interactive object space technique for the illustration of volumetric data sets by simulating free-hand line art drawing suitable for generating technical illustrations and sketches.

We have described an object space algorithm that generates hatching strokes from volumetric data sets. This allows us to arbitrarily modulate the strokes appearance by means of color and texture and to integrate stroke based rendering into other rendering modes, e.g. surface or direct volume rendering. The major contribution here is that we effec-

tively take advantage of hardware assisted volume and line rendering to generate meaningful images.

By selecting seed points at run time, we are able to minimize the overall memory requirements and the number of line segments to be generated, stored and rendered. This makes it possible to apply the technique to large-scale data sets.

Furthermore we have demonstrated that interactive NPR techniques for volumetric data sets can be achieved by taking advantage of graphics hardware. Even for large-scale data sets we achieve interactive frame rates on consumer class hardware using fragment shader support. In particular we hope to endorse the strategy already proposed by others - to bring NPR techniques and real-time 3D graphics to the visualization community.

References

[1] D.C. Banks. Illumination in diverse codimensions. *Computer Graphics (SIGGRAPH 94 Proceedings)*, pages 327–334, 1994.

[2] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings ACM Symposium on Volume Visualization 94*, pages 91–98, 1994.

[3] B. Csébfalvi, L. Mroz, H. Hauser, A. König, and E. Gröller. Fast visualization of object contours by non-photorealistic volume rendering. In *Computer Graphics Forum (Eurographics '01)*, pages 210–218, 2001.

[4] P. Decaudin. Rendu des scénes 3d imitant le style dessin anim. Technical report, Projet Syntim, 1996.

[5] D. Dooley and M.F. Cohen. Automatic illustration of 3d models: Lines. *Computer Graphics*, 23(2):77–82, 1990.

[6] Eberly, D. *3D Game Engine Design*. Morgan Kaufmann Publishers, 1999.

[7] D. Ebert and P. Rheingans. Volume illustration: Non-photorealistic rendering of volume data. In *IEEE Visualization '2000*, pages 195–203, 2000.

[8] G. Elber. Line art rendering via a coverage of isoparametric curves. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):231–239, 1995.

[9] G. Elber. Interactive line art rendering of freeform surfaces. In *Computer Graphics Forum (Eurographics '99)*, volume 18(3), pages 1–12. The Eurographics Association and Blackwell Publishers, 1999.

[10] A. Girshick, V. Interrante, S. Haker, and T. Lemoine. Line direction matters: an argument for the use of principal directions in 3d line drawings. *NPAR 2000, First International Symposium on Non-Photorealistic Animation and Rendering*, pages 13–20, 2000.

[11] B. Gooch and A. Gooch. Interactive non-photorealistic rendering. ACM Siggraph '99 Course Note, 1999.

[12] B. Gooch and A. Gooch. *Non-Photorealistic Rendering*. A K Peters, 2001.

[13] S. Green. Introduction to non-photorealistic rendering. ACM Siggraph '99 Course Note, 1999.

[14] W. Heidrich and H.-P. Seidel. Efficient rendering of anisotropic surfaces using computer graphics hardware. In B. Girod H. Niemann, H.-P. Seidel, editor, *Image and Multi-dimensional Digital Signal Processing Workshop '98*, pages 315–318. infix, 1998.

[15] A. Hertzmann and D. Zorin. Illustrating smooth surfaces. *Computer Graphics (SIGGRAPH 00 Proceedings)*, pages 517–526, 2000.

[16] Reyes Infografica. CartoonReyes. http://www.reyes-infografica.com/.

[17] V. Interrante. Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution. In *Computer Graphics (SIGGRAPH 97 Proceedings)*, pages 109–116, 1997.

[18] V. Interrante, H. Fuchs, and S. Pizer. Conveying the 3d shape of smoothly curving transparent surfaces via texture. *IEEE Transactions on Visualization and Computer Graphics,*, 3(2):98–117, 1997.

[19] R.M. Kirby, H. Marmanis, and D.H. Laidlaw. Visualizing multivalued data from 2D incompressible flows using concepts from painting. In *IEEE Visualization '99*, pages 333–340, 1999.

[20] A. Klein, W. Li, M. Kazhdan, W.T. Corra, A. Finkelstein, and T. Funkhouser. Non-photorealistic virtual environments. *Computer Graphics (SIGGRAPH 00 Proceedings)*, pages 121–129, July 2000.

[21] D. Laidlaw, E.T. Ahrens, D. Kremers, M. Avalos, R. Jacobs, and C. Readhead. Visualizing diffusion tensor images of the mouse spinal cord. In *IEEE Visualization '98*, pages 127–134, 1998.

[22] A. Lake, C. Marshall, M. Harris, and M. Blackstein. Stylized rendering techniques for scalable real-time 3d animation. *NPAR 2000, First International Symposium on Non-Photorealistic Animation and Rendering*, pages 13–20, 2000.

[23] J. Lander. Shades of disney. *Game Developer Magazine*, March 2000.

[24] E. Lum and K.-L. Ma. Hardware-accelerated parallel non-photorealistic volume rendering. *NPAR 2002, First International Symposium on Non-Photorealistic Animation and Rendering*, 2002.

[25] L. Markosian, M.A. Kowalski, S.J. Trychin, L.D. Bourdev, D. Goldstein, and J.F. Hughes. Real-time non-photorealistic rendering. *Computer Graphics*, 31(Annual Conference Series):415–420, 1997.

[26] O. Monga, S. Benayoun, and O. Faugeras. From partial derivatives of 3d density images to ridge lines. In *IEEE Conference on Computer Vision and Pattern Recognition 92*, pages 354–359, 1992.

[27] Y. Pnueli and A.M. Bruckstein. Digi Dürer – A Digital Engraving System. *The Visual Computer*, 10(5):277–292, 1994.

[28] E. Praun, H. Hoppe, M. Webb, and A. Finkelstein. Real-time hatching. *Computer Graphics (SIGGRAPH 01 Proceedings)*, pages 579–584, 2001.

[29] C. Reynolds. Stylized Depiction in Computer Graphics. http://www.red3d.com/cwr/npr/.

[30] P. Rheingans. Opacity-modulating triangular textures for irregular surface. In *IEEE Visualization '96*, pages 219–226, 1996.

[31] C. Rössl and L. Kobbelt. Line-art rendering of 3d-models. *Pacific Graphics 00*, pages 231–239, 2000.

[32] T. Saito. Real-time previewing for volume visualization. In *ACM Symposium on Volume Visualization '94*, pages 99–106, 1994.

[33] M. Salisbury, S. Anderson, R. Barzel, and D Salesin. Interactive pen-and-ink illustration. *Computer Graphics (SIGGRAPH 94 Proceedings)*, pages 101–108, 1994.

[34] M. Salisbury, M. Wong, J. Hughes, and D. Salesin. Orientable textures for image-based pen-and-ink illustration. *Computer Graphics (SIGGRAPH 97 Proceedings)*, pages 401–406, 1997.

[35] D. Stalling, M. Zöckler, and H.-C. Hege. Fast display of illuminated field lines. *IEEE Transactions on Visualization and Computer Graphics, 3(2)*, pages 118–128, 1997.

[36] S.M.F. Treavett and M. Chen. Pen-and-ink rendering in volume visualisation. In *IEEE Visualization '2000*, pages 203–210, 2000.

[37] R. Westermann and T. Ertl. Efficiently using Graphics Hardware in Volume Rendering Applications. In *Computer Graphics (SIGGRAPH 98 Proceedings)*, pages 291–294, 1998.

[38] G. Winkenbach and D. Salesin. Computer-generated pen-and-ink illustration. *Computer Graphics (SIGGRAPH 94 Proceedings)*, pages 91–100, 1994.

[39] G. Winkenbach and D. Salesin. Rendering parametric surfaces in pen and ink. *Computer Graphics (SIGGRAPH 96 Proceedings)*, pages 469–476, 1996.

[40] M. Zöckler, D. Stalling, and H.-C. Hege. Interactive visualization of 3d-vector fields using illuminated stream lines. *IEEE Visualization '96*, pages 107–113, 1996.

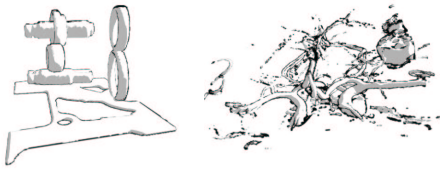

Figure 2: *Results of interactive volume cartoon shaded rendering. Rendering performance was about 7 fps.*

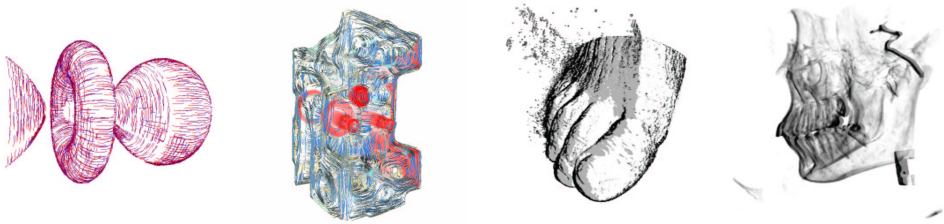

Figure 3: *We present interactive non-photorealistic rendering techniques for three-dimensional scalar fields. These techniques allow us to display volumetric structures in a way that is suitable for technical illustrations and sketches, and it generates additional visual cues providing an effective means for enriching traditional representations.*

Figure 4: *Hatches were lit anisotropically and superimposed over a volume illuminated by per-fragment phong.*

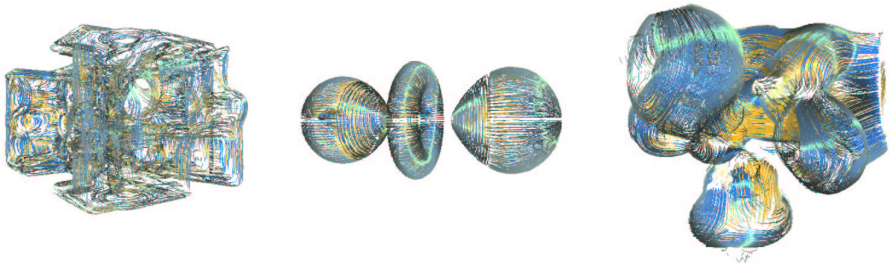

Figure 5: *Anisotropically lit hatches, but without the volume. Back facing parts of the hatches can be identified due to their yellow tone.*

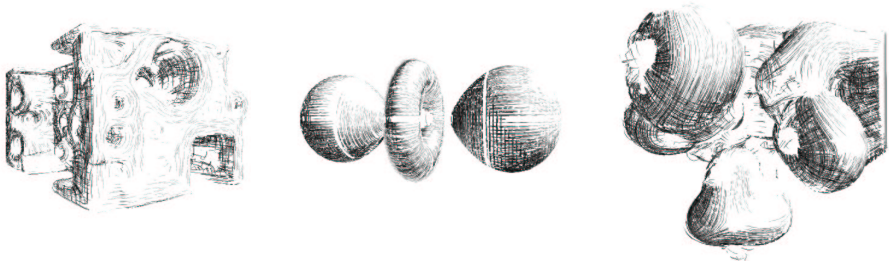

Figure 6: *Artistic hatching over isosurfaces. Cross hatches are used to emphasize both shape and lighting. For the leftmost image our seeding strategy was used. For the other two images seed points were placed in each voxel containing the selected iso-surface in order to gain a dense appearance of hatches on low-resolution datasets.*