Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2004)
O. Deussen, C. Hansen, D.A. Keim, D. Saupe (Editors)

Illustrating Surfaces in Volume

Xiaoru Yuan and Baoquan Chen

Department of Computer Science and Engineering, University of Minnesota at Twin Cities
200 Union Street S.E., Minneapolis, MN 55455, USA
Email:{xyuan, baoquan}@cs.umn.edu

Abstract

This paper presents a novel framework for illustrating surfaces in a volume. Surfaces are illustrated by drawing only feature lines, such as silhouettes, valleys, ridges, and surface hatching strokes, and are embedded in volume renderings. This framework promises effective illustration of both surfaces and volumes without occluding or cluttering each other. A two-step approach has been taken: the first step depicts surfaces; the second step performs volume rendering, at the same time embedding surfaces from the first step.
We introduce Procedurally Perturbed Image Processing (PIP), a new method for enhancing both feature detection and depiction of surfaces. We also present implementation strategies, especially those leveraging modern graphics hardware, for delivering an interactive rendering system. Our implementation results have shown that this mixed form of rendering improves volume visualization and is efficient.

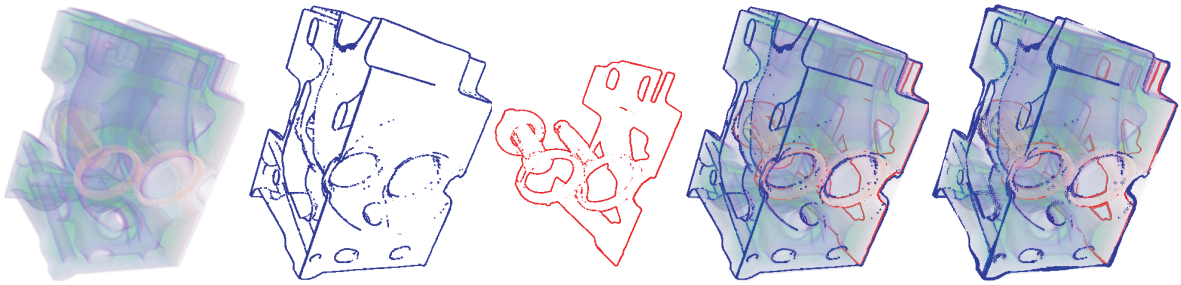
Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Three-Dimensional Graphics and RealismColor, Shading, and Texture; I.4.6 [Image Processing and Computer Vision]: SegmentationSegmentation Edge and feature detection;

1. Introduction

The complex nature of volume data imposes a challenging problem for effective visualization. Because of the data complexity, features and structures have to be selectively extracted and visualized. Surface-based volume renderings have been developed to depict iso-valued surfaces or structure boundaries. In these methods, surface geometry can be either explicitly extracted and represented in polygon mesh [LC87] or implicitly defined by a transfer function [Lev88]. Surface-based volume rendering, however, provides limited structure context because either only surface geometry is depicted (as in [LC87]) or the opaque surfaces occlude volume data behind them. Being able to depict both surfaces and their surrounding volume structures (context) can significantly improve a viewer's understanding of complex data.

There have been several methods developed to visualize both surface geometry and volume. One such method is to make surface geometry transparent. Embedding transparent polygonal surfaces in a volume has been researched by Kreeger and Kaufman [KK99]. There, polygons are drawn with transparency, revealing surrounding structures. While this offers the best possibility for enabling an integrated visualization of both surfaces and volumes, the full three-dimensional shape of a transparent surface can often be difficult to perceive, let alone when the rendering is mixed with a volume. The situation is worsened when multiple surfaces are to be depicted and mixed with volume rendering. In another representative method, Interrante [IFP95] used strokes on a transparent surface help enhance the surface shape.

In this paper, we develop a framework for effectively visualizing multiple surfaces and mixing them with the volume. To clearly convey surface shape and at the same time avoid obstructing volume illustration, surfaces are depicted through strokes representing only geometric features. As shown by various stroke-based non-photorealistic rendering (NPR) methods, strokes can provide more effective shape illustration by emphasizing important geometric features without cluttering them with small or unimportant ones. We then embed these strokes into volume rendering, which provides structure context. Figure 1 illustrates images generated through our methods. Comprehensive visualization (Figure 1(d) and (e)) can be achieved by appropriately mix-



(a) (b) (c) (d) (e)

Figure 1: A CT scan of engine rendered by various methods: (a) conventional volume rendering, (b) iso-surface with NPR illustration, (c) another iso-surface with NPR illustration, (d) mixture of iso-surfaces in (b) and (c) with volume rendering (a), (e) similar image as (d) with iso-surface illustration enhanced by PIP to emulate felt-tip pen drawing style.

ing iso-surfaces rendered in NPR styles with directly rendered volume. While NPR-generated surfaces highlight geometric structures such as silhouettes, creases, and valleys, volume rendering provides context structures surrounding the depicted surfaces. Our new framework features a two-step approach: the first step renders surfaces, and the second step renders volume and mixes it with surface strokes. Since the surfaces and volumes are rendered separately, this approach can bring together advances in each rendering task.

For the first step of surface stroke generation, several choices are possible. Interrante pre-generated evenly distributed strokes through 3D Line Integral Convolution (LIC) in a volume guided by principal curvature direction [Int97] and stored them in a separate texture volume. Stroke-based surface illustration was then generated through volume rendering with texture mapping, which can be expensive since it involves full volume rendering. Here, in contrast, we resort to surface-based NPR methods to generate the required stroke-based surface illustration and separate this from volume rendering to gain flexibility. A straightforward approach would require that a surface geometry be explicitly constructed before existing surface-based NPR methods can be applied; this is especially true for object-space methods that rely on polygon connectivity information to extract geometric features and view-dependent silhouettes (e.g., [IFH*03]). To accelerate this process, we instead detect surface features in image space, performing image processing, such as edge detection, on a geometric buffer (G-buffer) [ST90]. Here any general surface extraction and rendering method can be applied; then a G-buffer is generated through efficient rendering.

For the second step of volume rendering, any existing method can be employed. The key issue is the mixing of volume with surface illustrations. Since surface strokes are usually sparse, we set them to be fully opaque. These strokes occlude only limited volume structures, and hence the representation will not lose overall data context. Mixing opaque strokes with the volume can be efficiently performed through depth testing.

We enhance the feature detection by introducing a novel method we term Procedurally Perturbed Image Processing (PIP), in which the size and sampling positions of the image filter kernel are modified by a procedural perturbation function (hence the name) to impose artistic stylization on the output features. Moreover, in our algorithm and implementation design, we extensively leverage modern graphics hardware by implementing as many operations as possible using vertex and fragment hardware shaders. With this hardware support, multiple layers of surfaces can be interactively rendered and mixed with the volume.

In the rest of this paper, we will first review some recent developments in NPR volume rendering (Section 2), followed by an overview of our framework (Section 3). Then we discuss the two rendering steps, surface illustration and volume rendering and mixing in Section 4 and Section 5, respectively. In Section 6 we will discuss in more detail the Procedural Image Processing (PIP) introduced specifically for surface feature extraction and illustration. Results are demonstrated and discussed in Section 7, and finally concluding remarks are given in Section 8.

2. Related Work

Enhancing volume features in volumetric data visualization has recently become an area of active research. Critical to the success of a volume visualization is the technique for emphasizing important features and avoiding as much visual clutter and distraction as possible. Interrante enhanced transparent surface shape and position using sparse textured ridge and valley lines [IFP95]. Both ridge and valley lines correspond to geometric features of the surface measured on local surface geometry (e.g., normals, curvatures), which are further calculated from local volumes. Rheingans introduced an illustration approach [RE01] in which various visual cues, such as object boundaries, silhouettes, and halos, are evaluated and enhanced within the volume visualization pipeline by modulating the corresponding voxel's color and opacity. As in most other volume rendering enhancement methods [CMH*01, LME*02], the visual cues or features

are evaluated based on local volume characteristics (e.g., gradients). Various NPR styles have been generated, such as pen-and-ink style [TC00] and stippling [LME*02]. Dong et al.[DCLK03] developed a pen-and-ink volume hatching method for surfaces in which strokes are generated by 2D processing of projected surface voxels.

Using only one method of representation and visualization is sometimes insufficient to convey all relevant feature information. Accordingly, some researchers have developed hybrid rendering methods. Hauser et al. [HMBG01, HBH03] developed a two-level volume rendering approach that allows selective use of different rendering techniques for different segments of data. Zhou et al. [ZHT02] applied traditional photorealistic volume rendering to the focal volume region, but otherwise conducted NPR rendering. Lum and Ma presented a method combining several perceptually effective NPR techniques such as tone shading, silhouette illustration, and depth-based color cues in one single volume rendering [LM02]. Different visual cues are generated through separate evaluations and are blended together.

All the above methods perform feature extraction and visualization through volume-based processing. To obtain any individual feature, a separate pass of volume operation has to be performed, which can be very computation intensive. For example, in [LM02], multiple graphics cards spread across a PC cluster have been used to parallelize NPR volume rendering so as to obtain interactive rendering speed. Because our goal here is to highlight features on surfaces, our feature extraction is achieved through a more efficient method of surface rendering, producing a surface illustration that can also be efficiently mixed with volume rendering. Since our method employs image-space feature detection, the detected features are view-dependent, while existing methods usually generate view-independent features since they are calculated in volume space. Nevertheless, it is worth noting that many existing volume illustration methods can be employed to complement our method in providing improved volume visualization.

3. System Overview

Our rendering system takes a two-step approach. The first step generates a surface illustration, while the second step performs direct volume rendering and mixes this with surface illustration. As illustrated in Figure 2, the complete system pipeline consists of five major operations: surface extraction, surface rendering, surface feature detection, surface hatching (optional), and volume rendering and mixing (i.e., embedding surface illustration). The first four of these operations belong to the first step of surface illustration and can be repeated based on the number of surfaces defined. An explanation of each of these operations follows.

- **Surface extraction:** The first operation extracts surfaces from input volume data. Surfaces can be defined based on iso-values or segmented object boundaries. Since surface features will be extracted through image-based approaches, which do not rely on polygon connectivity, a point set is extracted for surface representation and rendering.
- **Surface geometry rendering:** The second operation renders point geometry and generates a geometric buffer (G-buffer) to allow image-based feature detection. The G-buffer includes one or multiple geometric properties such as depth, gradient of isovalue, and/or dot product of gradient and viewing direction [NXYC03]. Points are efficiently rendered by vertex programs.
- **Surface feature detection:** The third operation is to detect and illustrate features by applying various image-detection filters to the G-buffer. Various geometric features, such as silhouettes, ridges, and valleys, can be generated through image filtering and thresholding. We apply hardware-accelerated PIP to detect and stylize geometric features. (PIP will be discussed in more detail in Section 6.) The image filtering operations are performed by fragment programs. If more than one surface is rendered, each surface is rendered and feature-detected separately. The detection results are combined.
- **Surface hatching:** The optional fourth operation conducts hatching on surfaces for better conveying surface shapes. To achieve this, a subset of surface points is selected and hatching strokes are positioned at these points. With encoded 3D geometry information, strokes are oriented and drawn in various styles.
- **Volume rendering and mixing:** The last operation performs volume rendering and mixes it with surface illustration. We have developed strategies for performing this operation efficiently without altering the existing volume rendering pipeline.

4. Surface Illustration

This section explains the operations outlined above for illustrating surfaces extracted from volume.

4.1. Isosurface Extraction

Any isosurface extraction method can be applied in our framework. Here instead of resorting to conventional surface extraction methods [LC87] and generating polygon meshes, we extract a point-based surface model instead. Point-based representation can be more efficiently constructed and rendered than polygon meshes [CHJ03]. Here we directly extract the vertices of corresponding polygon meshes. An iso-point and its attributes are linearly interpolated from two adjacent voxels. Then an oriented ellipse (splat) is defined at that point position. Splats lie on the isosurface and overlap with adjacent ones without leaving holes in between. The extraction time depends on the dataset and number of iso-points. For example, it requires 0.17s for Engine(267K iso-

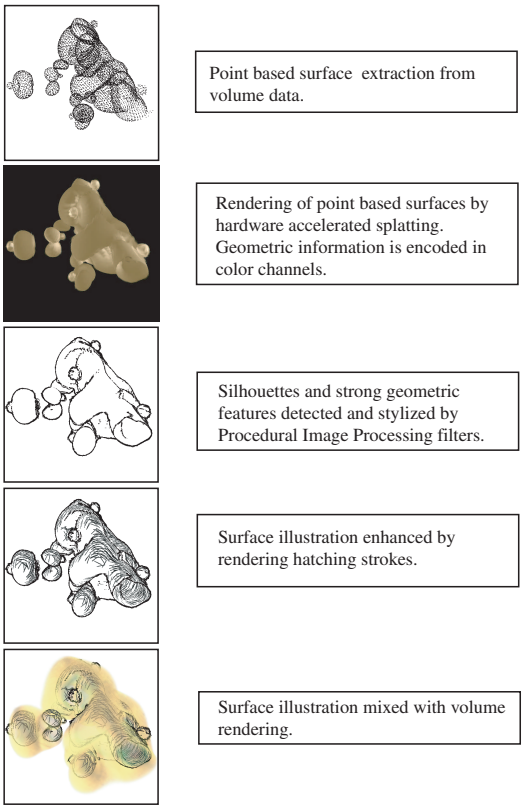


Figure 2: Overview of rendering pipeline.

points), 8.4ms for Neghip(17K isopoints). It is worth noting that methods for speeding polygon-based isosurface extraction can also be applied to accelerate point-based isosurface extraction.

4.2. Hardware Accelerated Surface Rendering

The objective of this operation is to generate a G-buffer for feature detection. Here we generate geometric information such as normal position and/or the dot product of the gradient and view vector for later image-based feature extraction. The gradient can also be used for directing surface orientation during surface hatching. To generate a smooth interpolation of geometric properties between adjacent points, we employ the method developed by [RPZ02], in which a point splat is rendered as a rectangle texture-mapped with a Gaussian alpha texture. Using this method, the surface rendering quality is satisfied even for small dataset. For large ones with more iso-points, using point sprites will greatly accelerate surface rendering while maintain adequate quality. Various G-buffers, such as depth, dot product of surface normal and view vector, can be generated at this stage. We mostly use dot product of normal and view direction since this information shows large variation in the high curvature place [NXYC03].

4.3. Image-Based Surface Feature Extraction

Features are detected by applying image filters [ST90, ND03], such as a Sobel filter for edge detection. The filtered result is then passed through a threshold test to classify the features. This feature detection operation is implemented in a fragment shader. More implementation details about image-based surface feature extraction can be find in [NXYC03].

To gain more control over detected features, such as the width and style, we have developed PIP, which can control the width and 'style' of the extracted features by procedurally modifying a basic filter(Section 6). Perturbation function can be either defined by analytical functions or encoded in textures. Users are able to load different textures and change parameters interactively to obtain different stylizations of the produced features.

4.4. Surface Hatching

The features detected by the procedures described above are geometric features such as silhouettes, ridges, and valleys. To further convey surface shape, we place strokes inside the surface boundaries to illustrate surface geometry and/or illumination. Figure 3(b) shows the depicted shapes by rendering evenly distributed strokes on the edge image Figure 3(a).

We use a subset of surface points to specify where strokes are placed; therefore, changing the size of this point set automatically changes stroke density. Using surface points to control strokes can also facilitate animation coherence [Mei96]. We offer two levels of control on stroke density: a global control and a local control. The global stroke density is controlled by a screen space density factor d, defining the number of strokes at a unit screen area. As pre-processing, we first randomly order the entire isosurface point list. At the run time, an object's screen projection size is estimated. This estimated area is multiplied with the density factor to obtain the number of strokes to be depicted, n. Then the first n number of points from the pre-generated surface point list are retrieved and projected to screen for determining stroke locations. A Similar operation is described in more details in [XC04].

Stroke density is also controlled by illumination. To achieve this effect, all isopoints are pre-assigned a random number; during the run time, a value is calculated for each point and compared against the assigned random number. If the test is successful, the point remains; otherwise, it is discarded. This value can either be geometric curvature or illumination value at the point. When illumination value is used, the stroke density will illustrate the shading tone, as can be seen in Figure 3(c).

The stroke orientation is guided by the geometry curvature at the point [IFP95]. Once the stroke position and orientation are known, a stroke is generated by drawing a textured quadrilateral primitive, similar to what is done in the

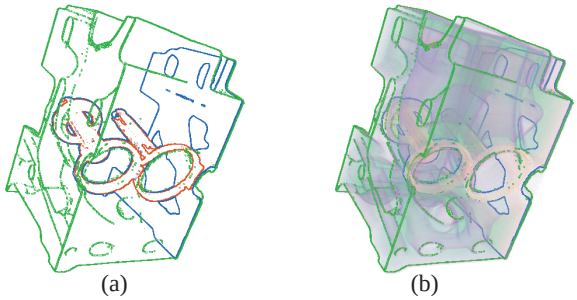
(a) (b) (c)

Figure 3: Surface illustration: (a) profiles only (filter-detected features), (b) with surface hatching, (b) with surface hatching and lighting effects.

previous operation of point-based surface rendering. Various stroke styles can be defined in the texture. The stroke visibility is simply determined by depth test (the depth buffer has been generated at the surface rendering stage).

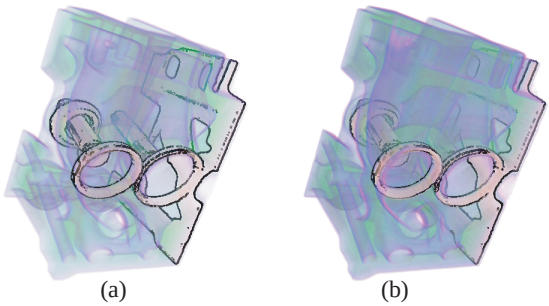
4.5. Multi-Surface Rendering

When multiple surfaces are specified, each surface is rendered separately and combined with others before volume rendering is performed. For each surface, its stroke illustration image and corresponding depth image are stored in textures. After all surfaces have been processed, the corresponding texture images are read and drawn to screen, however, all empty pixels are made transparent. Some operations are performed to achieve correct surface combination. First, at each surface rendering, the rendered image is stored into texture memory (achieved by a fragment program), and then the surface image (now texture) is rendered back into the frame-buffer (by texturing a window-sized rectangle). The fragment program evaluates whether a pixel belongs to a surface stroke; if so, the corresponding depth value is then written to the depth buffer. During this process, depth testing is enabled to ensure correct visibility. Since only strokes are rendered back to frame-buffer, surfaces between strokes will not occlude any strokes so that multiple surface layers can be simultaneously visible. Figure 4(a) shows three iso-surfaces illustrated and combined together. Figure 4(b) shows the surfaces mixed with volume.



(a) (b)

Figure 4: (a) Three isosurfaces (encoded by red, green, and blue colors in electronic version); (b) mixture of surfaces and volume.



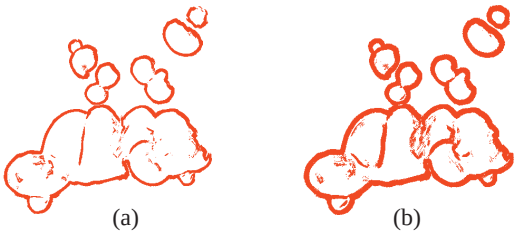
(a) (b)

Figure 5: Setting different global alpha ρ for volume to adjust the visibility of embedded surfaces: (a) ρ = 0.25, (b) ρ = 0.50.

5. Volume Rendering and Mixing with Surfaces

Volume rendering is implemented using hardware accelerated 3D texture rendering [RSEB*00]. Parallel polygons perpendicular to the viewing directions are generated and texture mapped with the volume and are composited together in back-to-front order. To mix surfaces with volume, simply compositing the image of surface rendering with that of volume rendering will not convey the correct depth variations of individual surface strokes. Our approach is to initialize appropriate depth buffer and enable depth testing before volume rendering. The depth initialization is performed by setting *only* depth values of feature or stroke pixels, which is done in multi-surface rendering. During volume rendering, no voxels behind surface strokes contribute to the final image. Voxels in front of strokes will contribute to the final pixels.

When illustrating a surface structure inside a volume, the rendered 3D volume could be too opaque to hide the surface inside. We set a transparency factor ρ to globally modulate voxels' alpha values. Tuning the the entire volume's transparency can achieve a different visibility of the surface illustration. Figure 5 demonstrates this operation by showing images generated using different ρ values.



(a) (b)

Figure 6: Silhouette width variation controlled by different constant perturbation functions: (a) $P(x,y) = 1.0 pixel$, (b) $P(x,y) = 2.0 pixels$.

6. Procedural Perturbed Image Processing (PIP)

In this section, we provide more details about the PIP. First, let us formulate a regular image filter operation. Let the input image F be a pixel array $f(x,y)$; W denotes a $m \times n$ filter kernel applied to the input image and the output image G has a pixel array $g(x,y)$. The output image can be computed by the expression:

$$g(x,y) = F \bigotimes W = \sum_{s=-a}^{s=a} \sum_{t=-b}^{t=b} w(s,t) f(x+s, y+t), \quad (1)$$

where $a = (m-1)/2$ and $b = (n-1)/2$. For PIP filtering, a procedural function $P(x,y)$ is applied to perturb the filter's sampling positions:

$$g'(x,y) = \sum_{s=-a}^{s=a} \sum_{t=-b}^{t=b} w(s,t) f(x+sP(x,y), y+tP(x,y)) \quad (2)$$

Let us draw some intuition here. If $P(x,y) = 1$, the filter becomes a regular filter. If $P(x,y) = 0$, Equation 2 returns the center pixel multiplied by the sum of filter weights. In the case of edge detection filter, the sum of filter weights is zero so no edge is detected. When the perturbation $P(x,y)$ increases, pixels further away from the filtering center will be sampled, therefore, slow edge may get amplified (widened).

Some examples of PIP filters and their results are shown in Figure 6 and 7. In Figure 6(a) and Figure 6 (b), a Sobel filter (3x3 kernel) is perturbed by constant function $P(x,y) = 1.0$ and $P(x,y) = 2.0$, respectively. The large perturbation factor produces thicker edges. In fact, both regular and random perturbation pattern (or function) can be defined. The perturbation patterns can be either defined by an analytical function or pre-generated and stored in an array (image). The latter allows generating more complex and random perturbation patterns. When a perturbation texture is used, it is usually smaller than the image to be filtered; when this is the case, it is usually tiled to cover the entire image. Figure 7 shows results by applying different PIP perturbation textures. The appearances of the detected feature edges are altered. In Figure 7(a), a regular 'dot' matrix texture is used. Smaller texture values (darker texels) reduce the chance of edge being detected. When the 'dot' size is smaller than the underlying edge width, this produces an effect that edges are being stippled or halftoned. On the other hand, when the 'dot' size is increased to be larger than the edge width, the edge can be either thickened or thinned, simulating felt-tip pen drawing style. This is illustrated in Figure 7(c). In Figure 7(b) the perturbation function texture is a Perlin's noise texture with small scale random patterns. Compared with Figure 7(a), the resulting image appears similarly stippled, but the edges appear more random. In Figure 7(d), an irregular 'dot' texture with larger scale patterns is used; the resulting image shows that detected edges are randomly distributed dots.

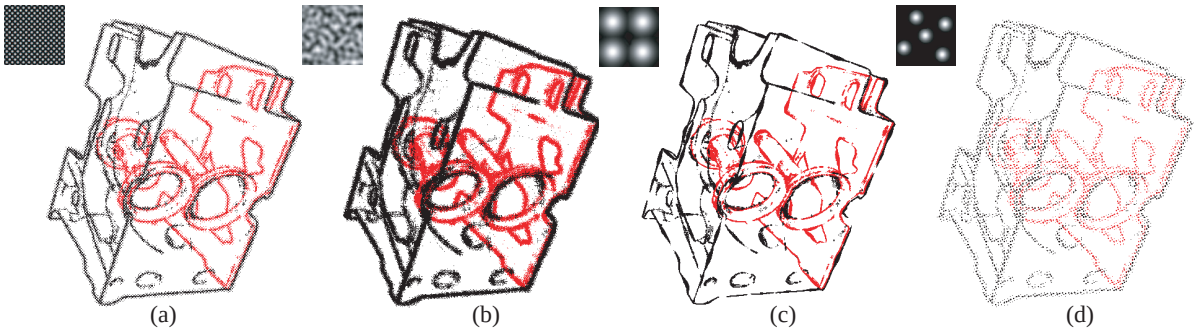
The current programable graphics hardware allow above PIP operations be implemented directly in fragment programs. Since perturbation textures are applied to screen space, it could lead to shower-door effect. However, the effect is not obvious probably due to that only edges or feature lines are illustrated and no large area pattern is involved. We have to point out that the net behavior of the PIP filter depends also on the underlying image content. The goal of the perturbation function design is to make resulted feature illustration appear random on small scales while conform with large scale stylization. This may also make the shower-door artifacts less severe.

7. Results and Discussions

All experiments have been performed on a Dell Precision 530 workstation with single Intel Xeon 2.20G Hz CPU, 1GB RAM, 4×AGP motherboard and a 256MB GeForce FX5900 Ultra graphics card. All images are generated at a screen resolution of 512×512 pixels. We utilize the Cg language [MGAK03] for vertex/fragment hardware programming. NV_vertex_program2 (vp30 profile) and NV_fragment_program (fp30 profile) OpenGL extensions are used in our implementation.

First, we demonstrate rendering performance. Figure 8 shows the averaged rendering times (ms) for some data sets. The surface rendering time includes surface splatting, edge detection, and surface hatching. As illustrated in the figure, for a medium sized data set of 256^3, a frame rate of around $3 \sim 5$ fps has been achieved. The volume rendering step is relatively fast and takes less amount of time as surface rendering. This is because volume rendering here performs neither advanced shading nor a complex feature enhancement operation; instead, merely color lookup, diffuse shading and compositing are conducted.

Next, we demonstrate visualization effects created by our system. Figure 9 and 10 show mixed rendering of surfaces and volume of several volumes, including both simulated volume data, such as silicium, protein and a Bucky ball, and medical volume data set, such as CT scans of a hand, cranium, chest, and foot. For all these examples, one or multiple surfaces are illustrated and are embedded in conventional volume rendering. Surface hatching is also performed in Figure 9. Since surfaces are depicted using strokes, multiple surfaces are visible without occluding



(a) (b) (c) (d)

Figure 7: A variety of stylization achieved by applying different perturbation patterns (top-left small texture images): (a) small 'dot' pattern, (b) Perlin's noise, (c) large 'dot' pattern, (d) irregular 'dot' pattern.

	T_0(ms)	T_1(ms)	T_2(ms)	N_p
Neghip	56.6	51.1	5.5	19.8K
C_{60}	133.05	117.5	15.5	79.8K
Engine	142.6	99.1	43.5	71.0K
Chest	343.4	297.6	45.8	239K

Figure 8: Averaged rendering time for different volume data (Neghip(64^3), C_{60}(128^3), Engine(256^2128)), Chest(256^3). T_0 is the total rendering time for each frame. T_1 is the surface rendering time, T_2 is the volume rendering time. N_p represents the number of isosurface points used.

each other. The stroke illustrated surfaces also allow maximum visibility of the enclosing volume, especially between strokes. Instead of superimposition of hatching surface on top of volumes [NSW02], Different color attenuation on surface strokes also provides depth cues of the strokes, and thus help viewers better comprehend the surface shape.

8. Conclusions and Future Work

We have introduced a framework for rendering multi-layered surfaces and their mixing with volume. In this framework, surfaces are illustrated using strokes highlighting their geometric features. The stroke-based illustration is then embedded in the corresponding volume. This hybrid illustration provides an opportunity to highlight surface features without losing their volumetric context information. Hadwiger's two level volume rendering [HBH03] is complementary to our work. We have also discussed implementation strategies, especially those leveraging modern graphics hardware for achieving interactive visualization and manipulation. Facilitated by a comprehensive GUI design and implementation, various parameters, such as the number of surface, surface values, PIP filters, lighting conditions, stroke textures, and transfer function for volumes, can be changed interactively. The functionalities and interactivity offered by our system can substantially enhance a data exploration experience.

There is room for improvement in our system. First, temporal aliasing can be present due to image-based iso-surface feature detection. We plan to investigate mechanism to address this issue. Next, we will seek to further investigate PIP operation. This paper has shown some effective applications of PIP filters. More principles need to be developed for guiding the design of PIP and realizing its full potential in visualization and stylization. Finally, we intend to integrate other volume illustration methods into our system. Existing non-photorealistic volume visualization methods can complement our approach in providing more available tools for further enriched volume illustration. As indicated in our system introduction, this integration is feasible.

9. Acknowledgements

We would like to thank the anonymous reviewers for their insightful comments and suggestions. This work was supported in part by NSF CAREER ACI-0238486 and by the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAD19-01-2-0014. Its content does not necessarily reflect the position or the policy of this agency, and no official endorsement should be inferred. Other support has included a Computer Science Department Start-Up Grant and Scholarship from the Office of the Vice President for Research and the Dean of the Graduate School, all from the University of Minnesota. Thanks to Amit Shesh and Fang Ye for proofreading.

The CT-scanned chest, hand, cranium, and foot data are downloaded from the web site of the Department of Radiology, University of Iowa. The Bucky ball data set has been created by Dr. Oliver Kreylos at the University of California, Davis. The Engine data is from General Electric. Neghip and Silicium are VolVis distribution of SUNY Stony Brook. Thanks to Michael Meissner for maintaining the volume data repository and providing downloads.

References

[CHJ03] CO C. S., HAMANN B., JOY K. I.: Iso-splatting: A point-based alternative to isosur-

face visualization. *Proc. of Pacific Graphics 2003* (2003), 325–334. 3

[CMH*01] CSÉBFALVI B., MROZ L., HAUSER H., KÖNIG A., GRÖLLER E.: Fast visualization of object contours by non-photorealistic volume rendering. *Computer Graphics Forum 20*, 3 (2001), 452–460. 2

[DCLK03] DONG F., CLAPWORTHY G. J., LIN H., KROKOS M. A.: Nonphotorealistic rendering of medical volume data. *IEEE Computer Graphics and Application 23*, 4 (2003), 44–52. 3

[HBH03] HADWIGER M., BERGER C., HAUSER H.: High quality two-level volume rendering of segmented data sets on consumer graphics hardware. *Proc. of IEEE Visualization '03* (2003), 301–308. 3, 7

[HMBG01] HAUSER H., MROZ L., BISCHI G. I., GRÖLLER M. E.: Two-level volume rendering. *IEEE Transactions on Visualization and Computer Graphics 7*, 3 (2001), 242–252. 3

[IFH*03] ISENBERG T., FREUDENBERG B., HALPER N., SCHLECHTWEG S., STROTHOTTE T.: A developer's guide to silhouette algorithms for polygonal models. *IEEE Computer Graphics and Applications 23*, 4 (2003), 28–37. 2

[IFP95] INTERRANTE V., FUCHS H., PIZER S.: Enhancing transparent skin surfaces with ridge and valley lines. *Proc. of IEEE Visualization '95* (1995), 52–59. 1, 2, 4

[Int97] INTERRANTE V.: Illustrating surface shape in volume data via principal direction-driven 3d line integral convolution. *Computer Graphics, Annual Conference Series* (1997), 109–116. 2

[KK99] KREEGER K., KAUFMAN A.: Mixing translucent polygons with volumes. *Proc. of IEEE Visualization '99* (1999), 24–29. 1

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3-d surface construction algorithm. *Computer Graphics (SIGGRAPH '87) 21* (1987), 163–169. 1, 3

[Lev88] LEVOY M.: Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications 8(5)* (May 1988), 29–37. 1

[LM02] LUM E. B., MA K.-L.: Hardware-accelerated parallel non-photorealistic volume rendering. *NPRA '02* (2002), 67–74. 3

[LME*02] LU A., MORRIS C. J., EBERT D. S., RHEINGANS P., HANSEN C.: Non-photorealistic volume rendering using stippling techniques. *Proc. of IEEE Visualization '02* (Oct. 2002), 211–218. 2, 3

[Mei96] MEIER B. J.: Painterly rendering for animation. *Computer Graphics (SIGGRAPH '96)* (Aug. 1996), 477–484. 4

[MGAK03] MARK W. R., GLANVILLE R. S., AKELEY K., KILGARD M. J.: Cg: a system for programming graphics hardware in a c-like language. *ACM Transactions on Graphics(SIGGRAPH '03) 22*, 3 (2003), 896–907. 6

[ND03] NIENHAUS M., DOELLNER J.: Edge-enhancement – an algorithm for real-time non-photorealistic rendering. *WSCG '03* (2003), 346–353. 4

[NSW02] NAGY Z., SCHNEIDER J., WESTERMANN R.: Interactive volume illustration. *Proc. of VMV '02: Vision, Modeling, and Visualization* (2002). 7

[NXYC03] NGUYEN M. X., XU H., YUAN X., CHEN B.: Inspire: An interactive image assited non-photorealistic rendering system. *Proc. of Pacific Graphics 2003* (2003), 372–376. 3, 4

[RE01] RHEINGANS P., EBERT D.: Volume illustration: Nonphotorealistic rendering of volume models. In *IEEE Transactions on Visualization and Computer Graphics* (2001), vol. 7(3), IEEE Computer Society, pp. 253–264. 2

[RPZ02] REN L., PFISTER H., ZWICKER M.: Object space ewa surface splatting: A hardware accelerated approach to high quality point rendering. *Computer Graphics Forum* (2002), 461–470. 4

[RSEB*00] REZK-SALAMA C., ENGEL K., BAUER M., GREINER G., ERTL T.: Interactive volume rendering on standard pc graphics hardware using multi-textures and multi-stage-rasterization. *Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware '00* (2000), 109–118,147. 5

[ST90] SAITO T., TAKAHASHI T.: Comprehensible rendering of 3-D shapes. *Computer Graphics 24*, 4 (1990), 197–206. 2, 4

[TC00] TREAVETT S., CHEN M.: Pen-and-ink rendering in volume visualisation. *Proc. of IEEE Visualization '00* (July 2000), 203–210. 3

[XC04] XU H., CHEN B.: Stylized rendering of 3d scanned real world environments. *Proc. of NPAR '04* (2004). 4

[ZHT02] ZHOU J., HINZ M., TÖNNIES K. D.: Focal region-guided feature-based volume rendering. *Proc. of the 1st 3DPVT* (2002), 87–90. 3

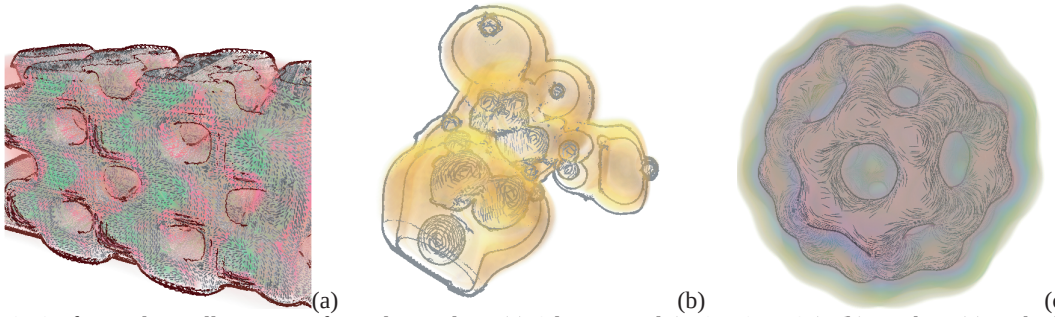


Figure 9: Surface-volume illustration of simulation data: (a) Silicium grid ($128 \times 64 \times 64$), (b) Neghip, (c)Bucky ball(C_{60}). Surface hatching is performed on all the illustrated surfaces except the outer layer surface in (b). Higher stroke density and shorter stroke lenghth are used in (a) compared with (b) and (c). Lighting effect is also enabled in (c).

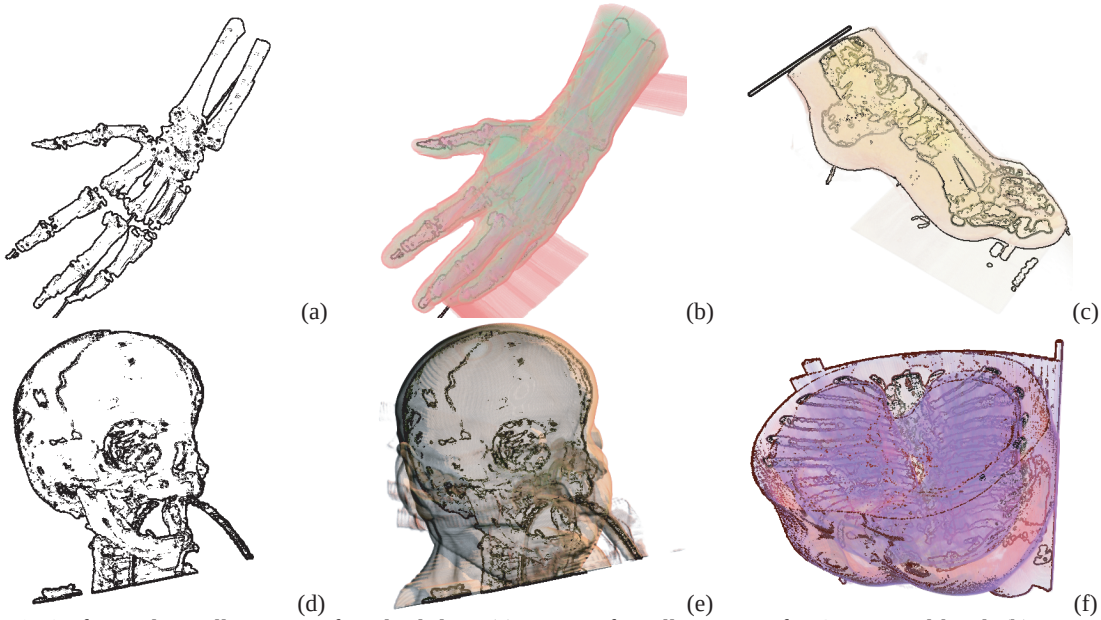


Figure 10: Surface-volume illustration of medical data: (a) NPR surface illustration of a CT scanned hand, (b) mixture of the surfaces in (a) and the corresponding volume, (c) mixture of surfaces and the foot volume, (d) NPR surface illustration of cranium, (e) mixture of the surfaces in (d) and the corresponding volume, (f) mixture of surfaces and the chest volume.