Procedural Image Processing for Visualization

Xiaoru Yuan and Baoquan Chen

Department of Computer Science and Engineering
University of Minnesota at Twin Cities, MN 55455, USA
{xyuan, baoquan}@cs.umn.edu

Abstract. We present a novel Procedural Image Processing (PIP) method and demonstrate its applications in visualization. PIP modulates the sampling positions of a conventional image processing kernel (e.g. edge detection filter) through a procedural perturbation function. When properly designed, PIP can produce a variety of styles for edge depiction, varying on width, solidity, and pattern, etc. In addition to producing artistic stylization, in this paper we demonstrate that PIP can be employed to achieve various visualization tasks, such as contour enhancement, focus+context visualization, importance driven visualization and uncertainty visualization.

PIP produces unique effects that often either cannot be easily achieved through conventional filters or would require multiple pass filtering. PIP perturbation functions are either defined by analytical expressions or encoded in pre-generated images. We leverage the programmable fragment shader of the current graphics hardware for achieving the operations in real-time.

1 Introduction

A key objective of visualization is to effectively deliver information by emphasizing important features while hiding unimportant ones. Image processing methods have been employed as a viable tool for illustration[1] where geometric features such as silhouettes, ridges, and valleys are extracted through image processing (e.g. edge detection) on intermediately generated geometric images. However, while existing image processing algorithms are effective in detecting features, they fall short in stylizing their appearance.

In this paper, we propose a new image filtering operation termed procedural image processing (PIP) that can stylize features *during* the process of detecting them. While a conventional image filter is defined by a kernel matrix with a fixed sampling and weighting pattern, a PIP filter has its sampling positions modulated by a perturbation function. When the perturbation function is designed properly, the detected features can be either enhanced or deprecated to display certain stylization. PIP produces unique effects that often either cannot be easily achieved through conventional filters or would require multiple pass filtering. The PIP filter has been applied to visualize isosurface in volume [2], in this paper, we extend its application in 2D image, 3D geometry visualization and 3D volume visualization. We discuss several perturbation functions or patterns and demonstrate their effectiveness on various visualization tasks, such as contour enhancement, focus+context visualization, and uncertainty or importance driven visualization.

G. Bebis et al. (Eds.): ISVC 2006, LNCS 4291, pp. 50–59, 2006.

The most recent advances in graphics hardware provide the ability to interactively perform image processing on GPU [3]. PIP operation adds little overhead to a conventional image filtering operation. When perturbation patterns encoded in a texture are passed to GPU with a few additional parameters, the added computation for PIP implementation is a few texture lookup operations and texture coordinates calculation. All PIP-based renderings are performed wholly on hardware through fragment programming and real-time performance is achieved.

2 Related Work

Many visualization systems have been striving to achieve improved effectiveness by selectively depicting important data features. For example, to visualize highly complex volume data, gradient operations have been employed to enhance boundaries via opacity modulation [4]. The more recent volume illustration systems have integrated gradient modulation into volume stylization [5]. Gradient information has also been used in high-dimensional transfer function design [6,7].

Image processing operators, including extracting edges, have been employed to achieve non-photorealistic rendering (NPR) of 3D objects. Image processing is performed on intermediately generated geometric images (G-buffer) to depict geometric features such as silhouettes, ridges, and valleys [8,1]. As only 2D processing in image space is required, image-based NPR methods can be more efficient than traditional object-based NPR methods which perform geometric feature extraction operations on 3D geometry.

Generally, an image processing filter defines a weighting function that is usually discretized and stored in a matrix, often called filter kernel. To process a pixel, the filter kernel is centered at the pixel, neighboring pixels are sampled based on the matrix grids and multiplied with matrix values (weights) and are finally summed together. Traditionally a linear edge filter is fixed in terms of both its weights and sampling pattern (matrix grid). Recently, a new filter called bilateral filter [9] has been proposed in which the weight of each filter sample is changed based on the difference between its value and the averaged value of all samples within the filter's footprint. Nevertheless, the output of a conventional filter is in the form of pixelized lines that lack solidity and styles. Since images are processed pixel by pixel and edge information is available only after the entire image processing is done, stylizing pixelized features such as halftoning [10], or overdrawing parameterized strokes on them usually needs additional pass(es) of processing.

In the remainder of the paper, we first introduce the basic elements of Procedural Image Processing (PIP) (Section 3), then describe its various applications(Section 4).

3 Procedural Image Processing (PIP)

An image filter kernel can be represented by a matrix mask W. Let the input image F be a pixel array $f(v)$, where v is the vector representing pixel position (x,y); W denotes a $m \times n$ filter kernel applied to the input image and the output image G has a pixel array $g(v)$. The output image can be computed by the expression:

$$g(v) = F \bigotimes W = \sum_{s=-a}^{s=a} \sum_{t=-b}^{t=b} w(r) f(v+r), \tag{1}$$

where $a = (m-1)/2$, $b = (n-1)/2$ and r denotes a filter element position (s,t). In cases presented in this paper, $m = n = 3$ (i.e., 9 samples for each filter). The filtering convolution of $F \bigotimes M$ at pixel $f(v)$ is obtained as usual by centering the filter matrix at the pixel, multiplying the matrix elements with their corresponding pixels and summing the results. Figure 2(a) illustrates an isosurface rendering image after conventional Sobel edge filtering. Silhouette lines have a one-pixel width.

For PIP filtering, a filter's sampling positions are perturbed by a procedural function $P(v,r)$, which is defined in the entire image domain:

$$g_{pip}(v) = \sum_{s=-a}^{s=a} \sum_{t=-b}^{t=b} w(r) f(v+P(v,r)). \tag{2}$$

Let us draw upon some intuitions here. If $P(v,r) = r$, the equation 2 is equivalent to the equation 1 and the filter becomes a regular filter. When the amplitudes of perturbation function $P(v,r)$ increases, sampling positions are moved away from the center pixel v, and pixels further away from the filtering center will be sampled. This can be considered as enlarging the filter kernel size (but the number of samples is kept the same). Therefore, slow pixel-value variations may be amplified, resulting in thicker edges. To gain more flexible control over the perturbation functions, we decompose a perturbation function into two components: scaling and translation. The resulting perturbation function is then expressed as:

$$P(v,r) = P_{scale}(v)r + P_{trans}(v). \tag{3}$$

The scaling factor is uniform for every direction. P_{trans} represents the translation vector. The perturbation functions can be encoded in textures for GPU implementation.

The effects of applying PIP with scaling and translation perturbations are demonstrated in Figure 1 for edge detection on 1D signals. Figure 1(a) is the original 1D signal with an edge. Figure 1(b) is the output of Figure 1(a) applied with a regular edge detection filter. The edge filter is a 1D matrix $|-1,0,1|$, each sampling position is exactly on the pixel grid. In Figure 1(c), a PIP with $P_{scale} = 1.5$ is introduced. The result edge is wider. We will demonstrated and discuss such edge widening effect in the next section. In Figure 1(d), a translation with offset of 0.5pixel in negative axial direction is applied. The resulting edge is shifting to positive axial direction correspondingly. In above cases, the same PIP perturbation function is applied throughout whole input signal. When applying different PIP perturbation functions in different regions, stylization variation can be introduced to the final rendering. By providing careful user control on such perturbation functions, various visualization effects can be generated.

Figures 2(b) and (c) demonstrate the results after applying only scaling perturbation. Larg constant scaling values are used throughout the entire respective image. Thicker edges than those from regular Soble filter (Figure 2(a)) are obtained. Such edge thickening cannot be achieved by simply decreasing the threshold, as that will introduce unwanted noises (Figure 2(d)).

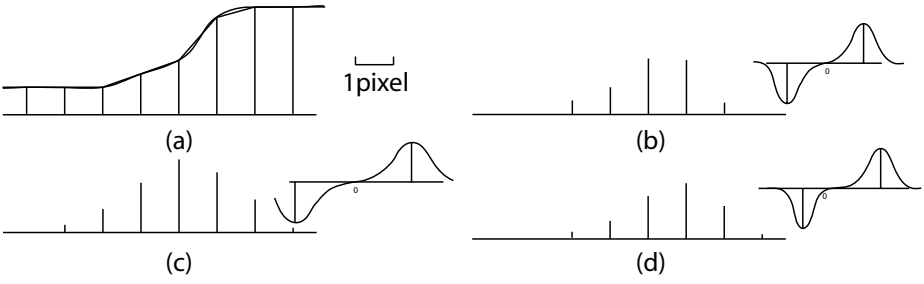

Fig. 1. 1D Procedural Image Processing (Edge Detection). (a) input signal, (b) signal output of a normal edge detection filter, (c) PIP filter ($P_{scale} = 1.5$), (d) PIP filter, shift a normal edge detection filter 0.5 pixel leftwards, ($P_{trans} = -0.5$).

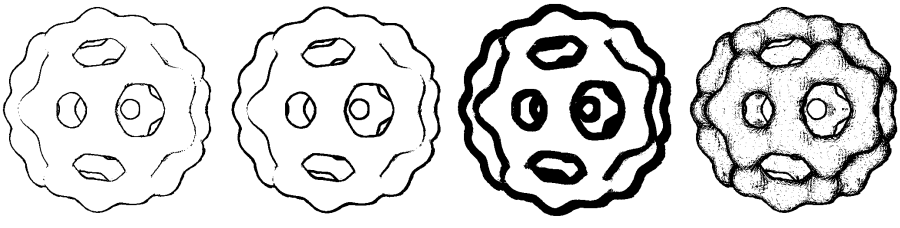

Fig. 2. 2D edge detection of isosurface (the isosurface is extracted from isosurface is extracted from Bucky ball data): (a) regular filter ($P_{scale} = 1.0$), threshold= 0.65; (b) PIP filter ($P_{scale} = 2.0$), threshold= 0.65; (c) PIP filter ($P_{scale} = 8.0$), threshold= 0.65; (d) PIP filter ($P_{scale} = 1.0$), threshold= 0.05

We must point out that the net behavior of the PIP filter also depends on the underlying image contents. The goal of the perturbation function design is to make the resulting feature illustration appearing random at small scales while conforming with large-scale stylizations driven by large-scale perturbation patterns.

The perturbation function can be normalized to the range of $[0, 1]$. Then at the processing time, the looked-up perturbation value needs to be multiplied with a constant scaling factor. Since the perturbation functions are usually periodic, we simply need to store one cycle of the function, which can be encoded in a much smaller texture image. During the processing, the texture image is tiled together to cover the entire image domain. Translation perturbation can be encoded in similar forms.

4 Applications

We demonstrate the capability of PIP in both image processing and visualization.

4.1 2D Image Processing

Various filtering results can be obtained by applying PIP to 2D images. Figure 3 shows several different PIP operations on the Lena image using different perturbation functions.

(a) (b) (c)

Fig. 3. PIP processed results of the Lena image. For each image, the smaller image in the white frame encodes scaling perturbation function.

Only the scaling perturbation is applied in all the examples here. For each image, the smaller image in the top right encodes the scaling perturbation function. In Figure 3(a), a constant scaling factor of 4 is used. Thickened edge lines are obtained. In Figure 3(b), a regular dot pattern is used as the scaling perturbation. The processed edges demonstrate a dot pattern, resembling a halftoning effect. In Figure 3(c), a swirl-like perturbation texture is used; the swirling perturbation patterns are carried over to edge depiction, resembling pen-and-ink drawing. In Figure 3(d), a water caustics texture is used as perturbation function. The resulting image displays beehive patterns resembling cracked painting.

4.2 Geometric Data Visualization

We demonstrate here different visualization effects that PIP can generate for 3D geometric models. The 3D rendered geometric buffers, like those used in image-based NPR [1], can encode additional information such as depth and normal. We utilize this additional information to modulate PIP operation to obtain additional control over data visualization.

In the first example we use depth values to control the PIP operation for modulating edge appearance based on depth, demonstrating a different kind of 'depth cueing'. Figure 4(a) shows such a result. The scaling perturbation function is defined as a constant (encoded in the left grey image in the bottom row). The depth image is on the right in the bottom row, while the regular color image is in the middle and is the one on which the PIP processing operates. Even though the scaling perturbation function is specified as a constant, it is inversely scaled by the corresponding depth value at each pixel. Thus more distant objects get a smaller scaling perturbation, resulting in a thinner detected edge.

In the next example we demonstrate that a different image portion or different part of a 3D object can use a different perturbation function to demonstrate different styles across the image. This style difference can be used to emphasize different aspects or convey certainty/uncertainty in data visualization. In our experiment, we define an importance map with values between $[0,1]$ (1: most important; 0: least important). We

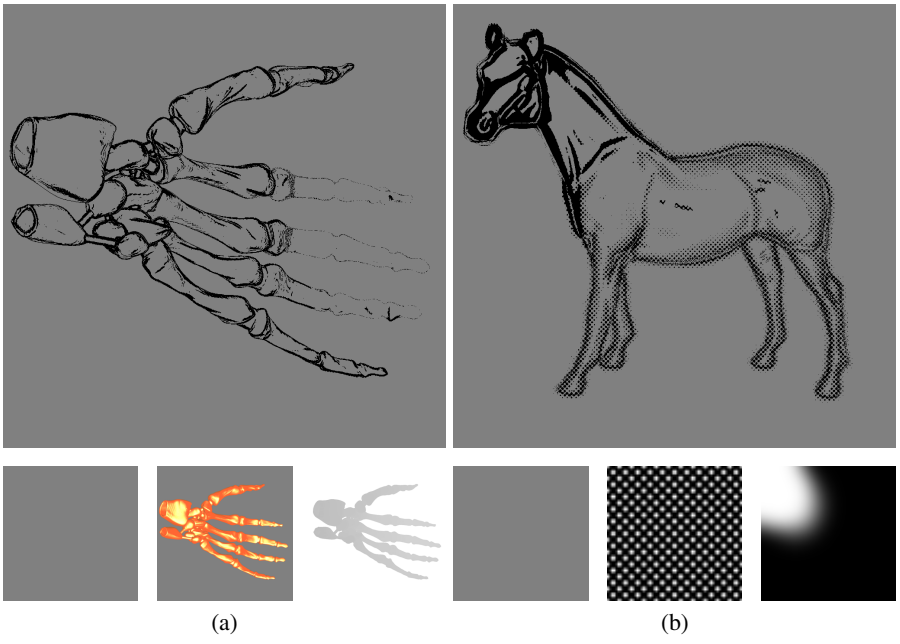

Fig. 4. Various effects generated by PIP: (a) depth cueing and (b) attention depiction. The bottom row of (a) shows (from left to right) the scaling perturbation pattern, the regular color image, the depth image. The bottom tow of (b) shows (from left to right) the two scaling perturbation functions for the most and least importance values and the importance map (bright pixel indicates higher importance).

then define two separate scaling perturbation functions associated with 0 and 1 importance value. For an importance value between 0 and 1, the perturbation function is interpolated between the two pre-defined ones. In Figure 4(b), the first two images in the bottom row illustrate the two pre-defined perturbation patterns; the uniform grey pattern is used for the most important value as it tends to strengthen edges, while the dot pattern will do the opposite. The rightmost image in the bottom row is the importance map. The top left corner of this image (corresponding to the horse head) is specified as the attention region (with high importance values). The resulting image shown on the top conforms with the design – the head is illustrated using thick dark lines and the rest of the body is depicted using small dots. Although in this example the importance map is specified in the image domain, it can be generated dynamically in 3D rendering. For example, when an attention point is specified on a 3D object, its projection on the screen can be the attention center. The attention center can also be obtained through eye or gaze tracking.

4.3 Volume Visualization

We also apply PIP to volume visualization. PIP operations on both isosurface rendering and direct volume rendering are demonstrated in the following examples.

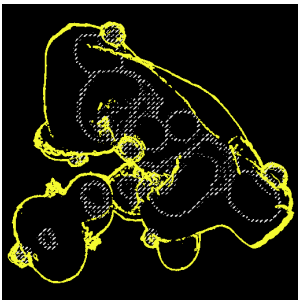

Fig. 5. PIP enhanced silhouettes of two isosurfaces

Isosurface Rendering: The isosurface method renders the surface that is defined by an isovalue in the volume. Same as for 3D polygon models, PIP generates silhouette and contour edges from the intermediate geometric buffers of the isosurface(s). Figure 5 shows two layers of outlines generated by PIP. Silhouettes of the inner isosurface are depicted by dashed lines The rendered silhouettes represent important geometry features of the isosurface; the variation of stylization can be used to depict other associated data information, such as uncertainty.

Direct Volume Rendering: Throughout the examples that we discuss here, we perform gradient modulated volume rendering. The gradient is computed using 3D central difference. The computed gradient magnitudes further modulate the opacity of the sample. Figure 6(a) and (b) show a conventional volume rendering of the Neghip data, without and with gradient modulation, respectively.

Next we illustrate a variety of interesting visualizing effects that can be generated by applying PIP at different stages of different purposed volume rendering. The first example illustrates how PIP can be used for importance driven or focus emphasis visualization. Research has been done to utilize different rendering methods such as direct volume rendering, maximum intensity rendering and NPR rendering to differentiate focus and context regions [11,12]. Here we seek to design perturbation functions in PIP to achieve the same goal. We define a scalar value d to represent the degree of interest. In the region of interest (ROI), $d = 1.0$, otherwise $d < 1.0$. We then use this d value as the scaling factor in PIP kernel. Features (surface boundaries) are indeed much enhanced in the ROI when this PIP is applied to gradient modulated volume rendering (like in Figure 6(b)), while becoming less visible in regions outside the focus center. Figure 6(c) illustrates the result.

The fourth example shows application of PIP to uncertainty visualization [13,14]. To simulate an uncertainty distribution, we assume the data at the ROI is the most accurate, but uncertainty increases when moving away from the ROI. There are several possible ways of achieving this. One intuitive way is to generate a noise function (ranging from 0 to 1) and then directly modulate opacity with this noise value.

We now describe our approaches of using PIP for achieving uncertainty visualization. First, we define a noise function, which is multiplied by $1 - d$ and added with d and is then used as the the scaling function of PIP. Hence, the ROI receives no noise

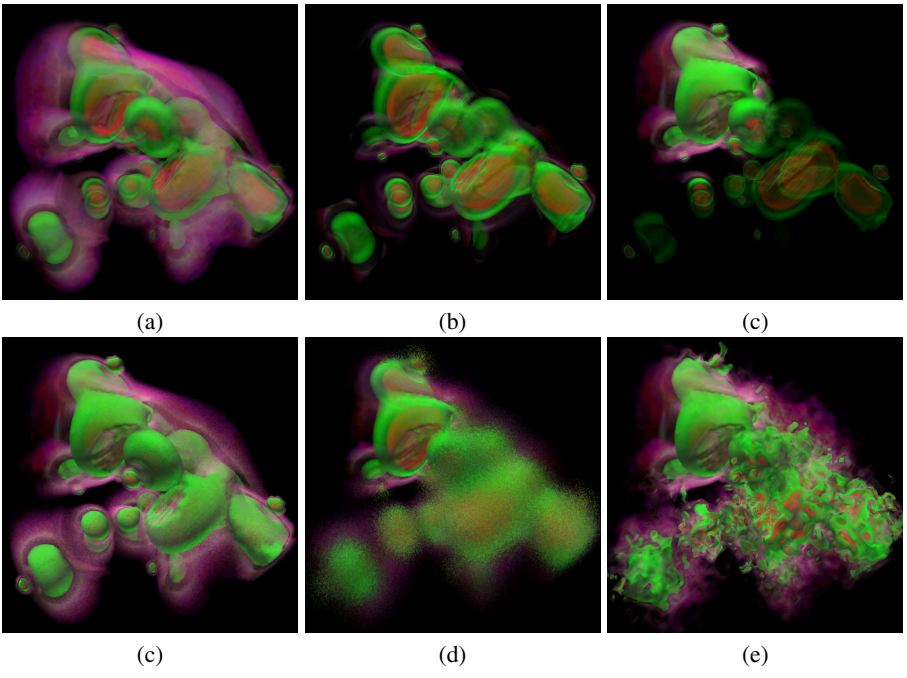

Fig. 6. Various effects of volume rendering: (a) regular volume rendering; (b) gradient enhanced volume rendering (regular filter); (c) importance driven visualization; (d-f) uncertainty visualization ((d) opacity directly modulated by noise, (e) noise as the translation function in PIP, (f) similar to (e), but with lower noise frequency)

while the noise values increase as one moves away from the focus center. The consequence is that the thickness of the boundary is randomly varied (hence appearing rough) in the uncertainty region. Figure 6(d) illustrates this effect. Our method to depict uncertainty is to define a translation function of PIP. The translation direction is constant, but the amount of translation is determined by $(1-d)*noise$. Figure 6(e) shows the resulting effect in which the surface boundaries become cloudy at the uncertain region. In Figure 6(f), we apply a lower frequency noise than that is used in Figure 6(e). In this image, surface boundaries can be seen as being randomly deformed. Through the above examples, PIP provides another set of opportunities for visualizing data uncertainty, especially when surfaces boundaries are to be depicted.

5 Implementation Notes, Discussions and Conclusion

All our experiments have been performed on a Dell Precision 530 workstation with a 256MB GeForce FX5900 Ultra graphics card. We use the newly introduced Cg language [15] for hardware fragment programming. Specifically, NV_fragment_program (fp30 profile in Cg program) OpenGL extension [3] is used in our implementation. All the operations involved in PIP are implemented in hardware. Perturbation functions are encoded in texture images. Texture lookup, sample transformation and interpolation, and

filter convolution are some typical sub-operations in the implementation of the PIP operation. The PIP implementation can be fully integrated with hardware assisted volume rendering, which leverages 3D texture mapping hardware [16]. The PIP filter operation is directly performed in the fragment program where 3D texture lookups are performed. The overhead of PIP operation is almost negligible. Because of the hardware implementation, all rendering demonstrated in the paper is real time. The rendering times for the stylized Lena images in Figures 3 are around 0.15 ms. The 2D image rendering time is close to the PIP processing time which is much smaller comparing with model rendering time of mesh or volume. The rendering time of the horse model (97K triangles) in Figure 4 is 11.2 ms and the hand model (655K triangles) is 68 ms. We didn't observe obvious performance difference when the PIP operation is turned on.

Although PIP represents a simple extension of conventional image processing, a variety of effects can be achieved *during* image filtering without resorting to any pre- or post-processing. The stylization is achieved by perturbation patterns which modulate filters' sampling positions. The effects are unique that often either cannot be achieved through conventional filters or would require multiple pass filtering. Our method works on both 2D images and 3D data and can achieve a wide range of visualization tasks, such as depth cueing, focus+context visualization, importance driven visualization, and uncertainty visualization.

We have identified two immediate avenues for future work. First, we plan to investigate more perturbation functions in hope of achieving additional effects. Along this line, we seek to understand the relationship between perturbation functions and their resulting effects in greater depth. Second, although we have concentrated on edge detection filter so far, we aim to extend the PIP concept to other filters as well, such as unsharp masking filter [17].

Acknowledgements

Support for this work includes University of Minnesota Computer Science Department Start-up funds, University of Minnesota Digital Technology Center Seed Grants 2002-4, NSF ACI-0238486 (CAREER), NSF DMS-0528492, and 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.

The Skeleton hand model is from the Stereolithography Archive at Clemson University. The volumetric CT scanned engine data is from General Electric. The Neghip is a VolVis distribution of SUNY Stony Brook. The Bucky ball data set has been created by Dr. Oliver Kreylos at the University of California, Davis.

References

1. Saito, T., Takahashi, T.: Comprehensible rendering of 3-D shapes. In: Proceedings of SIGGRAPH 1990. Volume 24. (1990) 197–206
2. Yuan, X., Chen, B.: Illustrating surfaces in volume. In: Proceedings of Joint IEEE/EG Symposium on Visualization (VisSym'04), the Eurographics Association (2004) 9–16

3. Nvidia: Nv_fragment_program. NVIDIA OpenGL Extension Specifications for the CineFX Architecture (NV30) (2003)
4. Levoy, M.: Display of surfaces from volume data. IEEE Computer Graphics and Application **8** (1988) 29–37
5. Ebert, D., Rheingans, P.: Volume illustration: Non-photorealistic rendering of volume models. In: Proceedings of IEEE Visualization '00. (2000) 195–202
6. Kindlmann, G., Durkin, J.W.: Semi-automatic generation of transfer functions for direct volume rendering. In: Proceedings of IEEE 1998 Symposium on Volume Visualization. (1998) 79–86
7. Kniss, J., Kindlmann, G., Hansen, C.: Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In: Proceedings of IEEE Visualization '01. (2001) 255–262
8. Hertzmann, A.: Introduction to 3d non-photorealistic rendering:silhouettes and outlines. ACM SIGGRAPH 99 Course Notes (Non-Photorealistic Rendering) (1999)
9. Tomasi, C., Manduchi, R.: Bilateral filtering for gray and color images. Proceedings of the 1998 IEEE International Conference on Computer Vision (1998) 836–846
10. Ostromoukhov, V., Hersch, R.D.: Artistic screening. In: Proceedings of SIGGRAPH 1995, ACM Press (1995) 219–228
11. Hauser, H., Mroz, L., Bischi, G.I., Gröller, M.E.: Two-level volume rendering. IEEE Transactions on Visualization and Computer Graphics **7** (2001) 242–252
12. Zhou, J., Hinz, M., Tönnies, K.D.: Focal region-guided feature-based volume rendering. In: Proceedings of First International Symposium on 3D Data Processing Visualization and Transmission. (2002) 87–90
13. Grigoryan, G., Rheingans, P.: Probabilistic surfaces: Point based primitives to show surface uncertainty. In: Proceedings of IEEE Visualization '02. (2002) 147–154
14. Johnson, C.R., Sanderson, A.R.: A next step: Visualizing errors and uncertainty. IEEE Computer Graphics and Application **23** (2003) 6–10
15. 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 (TOG) **22** (2003) 896–907
16. Van Gelder, A., Kim, K.: Direct volume rendering with shading via three-dimensional textures. In: Proceedings of the 1996 Symposium on Volume visualization. (1996) 23–30
17. Luft, T., Colditz, C., Deussen, O.: Image enhancement by unsharp masking the depth buffer. ACM Transactions on Graphics **25** (2006) 1206–1213