INSPIRE: An Interactive Image Assisted Non-Photorealistic Rendering System

Minh X. Nguyen Hui Xu Xiaoru Yuan Baoquan Chen

Department of Computer Science and Engineering
University of Minnesota at Twin Cities
http://www.cs.umn.edu/~{mnguyen,hxu,xyuan,baoquan}
Email:{mnguyen,hxu,xyuan,baoquan}@cs.umn.edu

Abstract

Two prominent issues in non-photorealistic rendering (NPR) are extracting feature points for placing strokes and maintaining frame-to-frame coherence and density of these feature points. Most of the existing NPR systems address these two issues by directly operating on objects, which can not only be expensive, but also dependent on representation.

We present an interactive non-photorealistic rendering system, INSPIRE, which performs feature extraction in both image space, on intermediately rendered images, and object space, on models of various representations, e.g., point, polygon, or hybrid models, without needing connectivity information. INSPIRE performs a two-step rendering process. The first step resembles traditional rendering with slight modifications, and is often more efficient to render after the extraction of feature points and their 3D properties in image and/or object space. The second step renders only these feature points by either directly drawing simple primitives, or additionally performing texture mapping to obtain different NPR styles. In the second step, strategies are developed to promise frame-to-frame coherence in animation. Because of the small computational overheads and the success of performing all the operations in vertex and pixel shaders using popularly available programmable graphics hardware, INSPIRE obtains interactive NPR rendering with most styles of existing NPR systems, but offers more flexibility on model representations and compromises little on rendering speed.

Keywords: non-photorealistic rendering, interactive 3D graphics, programmable graphics hardware, GPU.

1 Introduction

Recent advances in computer graphics hardware and algorithms have introduced increasing interests in Non-Photorealistic Rendering (NPR). NPR effects such as pen and ink [14], painting [5], stippling [16] and charcoal drawing [8] have been produced. There are two fundamental isssues to address in producing NPR styles: (1) to extract feature points for placing strokes (of various styles); and (2) to maintain frame-to-frame coherence of these feature points during a continuous navigation. Related to the second issue is the problem of maintaining stroke density. Cornish et al. [2] summarize several commonalities in NPR techniques for generating several highly varied artistic effects.

According to where the NPR features are extracted, NPR methods can be categorized into object space and images space methods. Object-space approaches for NPR are popularly used in existing NPR systems. 3D features are directly computed based on 3D geometry [6] and strokes are placed on surfaces. This approach tends to require a well behaved geometric representation of 3D models. Raskar and Cohen employ a method which requires no polygon connectivity information for displaying silhouette edges [12]. However, inspite of using the current graphics hardware, many more primitives than required need to be rendered. Markosian [9] trades accuracy and details for speed by only randomly examining a small fraction of the edges. This randomized method cannot guarantee that all silhouettes be retrieved. To maintain coherence, Meier [10] associates strokes with particles defined on the surface of the object. View-dependent particles can be selected based on view-dependent polygon simplification [2]. Although effective NPR styles have been achieved using object-space approaches, obtaining interactive speeds for large models is often difficult, if not impossible.

Earlier NPR systems take an image-based approach by post-processing intermediate images. Saito et al. [13] employ G-buffers (geometric buffers) that preserve geometric properties of objects in the scene in image space, such as normals, depths, or object identities, for deriving image-space information such as contour lines, which are then combined with the conventionally rendered images. Decaudin [3] enhances Saito's method with an additional normal map. Nienhaus et al. [11] accelerate image space edge filters utilizing modern texture hardware and combine edge features with other conventional rendering through projective texture mapping [17]. However, their 3D models have to be rendered twice. The image-based approach is attractive because there are a wealth of image processing tools, hence it is more flexible and robust. However, several issues related to the image-based approach are unsolved. The first issue is the frame-to-frame coherence – as each frame is processed independently, some weak features (like valleys or ridges) in object space may not be consistently detected in subsequent image frames. Then the detected image features are combined with conventional or stylized renderings. The styles achieved in this approach are limited, and the image features lack 3D continuity.

In this paper, we present an NPR system, dubbed INSPIRE, that extends the image-based approach to achieve most features of object-based NPR systems, i.e., the wide range of styles, the frame-to-frame coherence, and the uniform density control of strokes. This is achieved by a number of novel strategies. Firstly, the content of intermediate images (or G-buffers) is specially designed so that sufficient geometric information is available for directing stroke placement. The detected feature points are classified into strong and weak edge points. This allows us to place different strokes to illustrate strong and subtle features separately. Secondly, to ensure frame-to-frame coherence, we employ a set of seed points on the initial object and project them to the image space as well. These seed points serve as candidate locations for placing strokes so that strokes can appear sticking to object sufaces. Thirdly, these seed points can be further used to illustrate the tone of the surfaces based on interactive lighting changes. Finally, all major operations are directly implemented in graphics hardware, specifically, the latest vertex and pixel shaders, so that interactive NPR is achieved.

INSPIRE hence represents an extended image-based approach that combines object space controls (i.e., seed points) to deliver various NPR styles and coherent navigation. INSPIRE promises

a number of features: model independence, stylization flexibility and efficiency.

1. *Model independence:* All the special features (silhouettes, ridges, valleys) are detected on the fly from the rendered 2D image. This makes our method independent of object representations, e.g., polygons, points, or hybrids [1]. Because of the model independence, this method can be easily merged with almost any existing 3D rendering system with optimizations such as multi-resolution and view-dependent approaches. For the same reason, our method is suitable for generating NPR images of dynamic scenes and animation objects as no connectivity information needs to be maintained.

2. *Stylization flexibility:* Because of a range of ways of specifying feature points and the separation of feature point extraction and stylization, we can support various NPR styles within the same framework.

3. *Efficiency:* Our method does not depend on the complexity of a 3D scene for feature point extraction. Our combination of image and object space techniques also provides an opportunity to utilize hardware acceleration.

The rest of the paper is organized as follows. We first provide an overview of our system (section 2), then discuss about our rendering pipelines in details (section 3), especially our implementation in vertex and pixel shaders. We present our results (section 4), and conclude with a discussion and plan for future work (section 5).

2 System Overview

INSPIRE can illustrate both view independent edge features such as ridges and valleys, and view dependent edges such as silhouettes. INSPIRE can also illustrate tones of surfaces under different lighting conditions. We will first explain procedures of illustrating only object edge features, in which we discuss how to maintain stroke consistency and density uniformity during animation. Then, we will explain the procedures for illustrating tones of shading.

The system can be divided into three steps, although 3D models are rendered only once. The first step of the system is to render 3D models. In this step, seed points, which are pre-selected points on object, are also optionally projected. The second step is to perform image processing (i.e., edge filtering) to extract edge points. Seed points and edge points are combined to form *feature points* in the third step in which strokes are placed. The strokes are drawn either as line segments, or oriented rectangles mapped with stroke textures.

All pixels with non-zero alpha value after edge filtering are candidate edge points, which can be excessive; drawing all of them may lead to a cluttered effect. Therefore, in step 2, we further categorize candidate edge points into "strong" and "weak" edge points by comparing the filtered alpha value with a threshold value. This classification also helps us to apply different strokes to different groups of edge points to emphasize desirable features and achieve different styles. For example, strong edge points such as silhouettes and sharp ridges and valleys are drawn using thick and long strokes, while weak edge points are either discarded or drawn using thin and short strokes, depending on desired styles. Without explicit notice, hereafter we will refer to strong edge points simply as edge points.

The key to the interactivity of this system is a compact hardware implementation of all the key operations of the pipeline. The image filtering step for edge detection is directly performed in pixel shaders similar to that of Nienhaus et al. [11]. Using traditional color images to detect object space edge features may fail to detect some edge features due to the interference of lighting. Therefore, here we generate intermediate images that encode surface normals for edge detection. After image filtering, only edge points are kept. However, normal information is needed to direct stroke orientation for each detected edge point. To keep the normal information and avoid rendering the object multiple times, we carefully encode different information in RGBA channels of the frame buffer. The Alpha channel stores the dot-products of the surface normals and the view vectors. Instead of maintaining all three components of surface normals, we convert object space normals into eye space and then discard the "z" component. We store the other two components in Red and Green channels respectively. The seed points in the first step are projected and their intensities are stored in the Blue channel.

Figure 1 illustrates the pipeline of generating one rendering style, which can be generalized to the generation of other styles. Figure 1(b), (c), and (d), illustrate the eye space normals (red and green channels), intensities of seed points (blue channel) and the dot-products of normal and view vectors (alpha channel) respectively. Figure 1(e) is the image of the alpha channel after edge filtering. Figure 1(f) shows the selected "strong" edge points. Strokes are placed at the edge points in Figure 1(f) and are oriented by their corresponding normals in Figure 1(b); Figure 1(g) is the image showing strong silhouette edges. For non-edge points in Figure 1(e), their corresponding seed points in Figure 1(c) are used as the remaining feature points; strokes of different styles are drawn, again, using their corresponding normals in Figure 1(b). The final image is depicted in Figure 1(i).

3 Implementation Details

3.1 Rendering of 3D Models

This step performs conventional 3D rendering but substitutes the general color attributes with geometric attributes. The objective of this rendering is to obtain image space geometry buffers for further processing. This rendering is independent of model representation. Polygon, point, or hybrid models can be used. In our experiment, we mostly use point-based rendering for improving rendering efficiency without compromising on the final image quality. We assume rendering primitives as points in the next discussion.

Each color component encodes a different geometric attribute. One such attribute is the dot product of the normal vector and the view vector at each point. This scalar value is stored as the alpha component. The normal at each point is projected to screen space and the x and y components constituting the screen space normal are stored as red and green values respectively, while the "z" component is simply discarded. After rendering, the obtained alpha channel will be used for edge detection, while the red and green channel can be used to guide the stroke orientation. Because of this compact representation, there is still a blue channel left, which will be used for storing seed point projection, to be discussed next. At this 3D rendering step, the blue value of each point is set to zero.

These geometric attributes are computed in vertex shaders. The pseudo code (**VertexProgram1**) can be found in Appendix A.

3.2 Rendering of Seed Points

In the first step, pre-selected seed points are also projected and taken as candidate locations for laying strokes for illustrating interiors of the object, in order to maintain coherence between frames when the 3D model dynamically changes. The seed points are a subset of original object points; their number depends on the required density. In fact, the number of seed points needs to be dynamically changed to maintain a constant screen density. The seed points' density correlates to feature points' density, and therefore, a constant on-screen density of seed points leads to uniform density

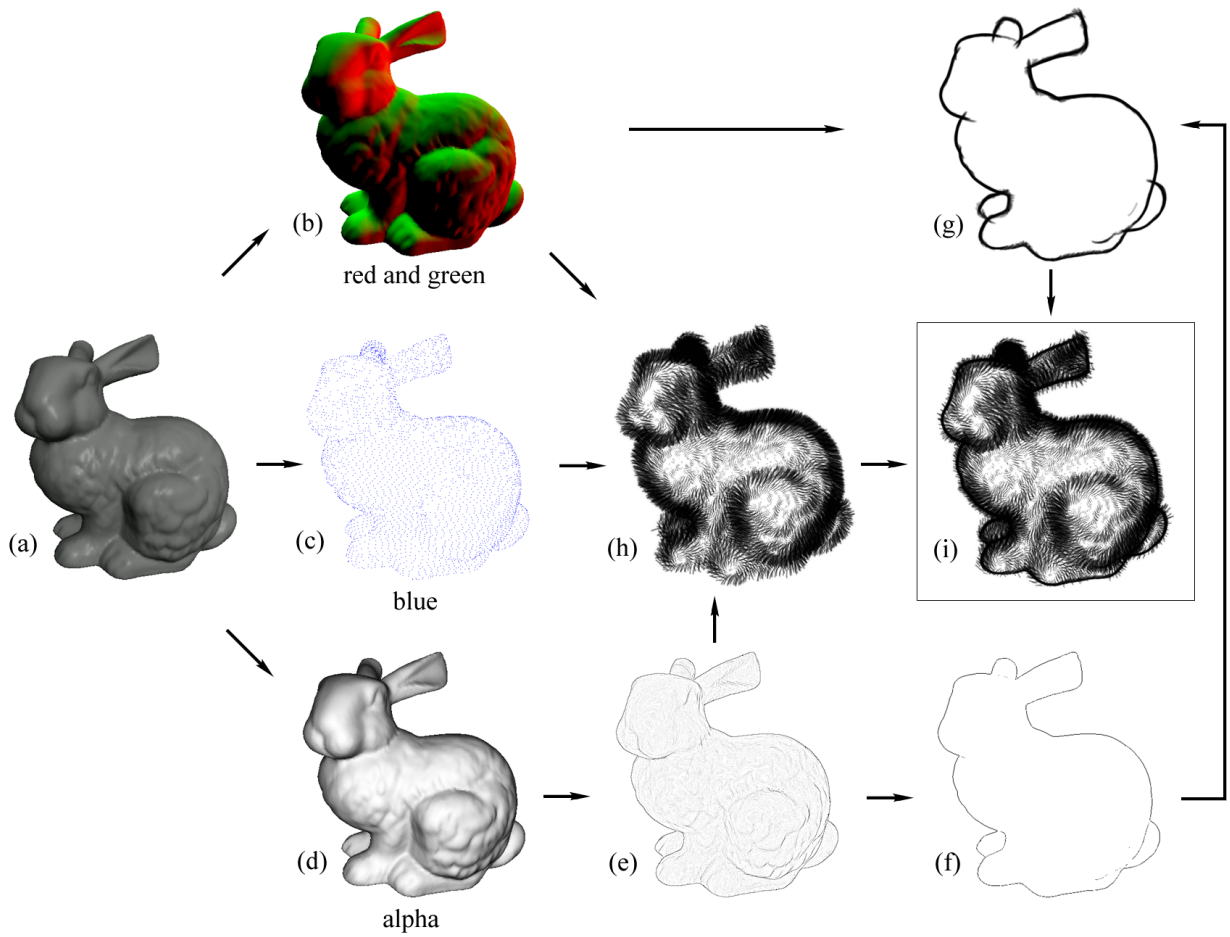

Figure 1: *The general rendering pipeline includes three steps of rendering. Step 1: the object (a) is rendered by encoding different information in different frame buffer channels: (b) red and green channels store eye space normals; (c) blue channel encodes projected seed points (and optionally their shading intensities); (d) alpha channel stores the dot-product between surface normals and viewing directions. Step 2: image filtering is performed on alpha channel (d) to obtain edge points (e), which is further classified into strong (f) and weak edge points. Step 3: rendering strokes at edge points. The strong edge points are rendered using strokes of one style (g) while the weak edge points are rendered using strokes of another style (h). All strokes are oriented based on their corresponding normals in (b). The final image (i) is a combination of image (g) and (h).*

of strokes, hence producing a constant tone of the surfaces [15]. To achieve this, we take a randomized approach in which the initial point set of an object is randomly ordered and stored in a linear buffer. Then at runtime, depending on the number of seed points needed, a point set is fetched from the linear buffer starting from the beginning. This method is successfully used in [18].

When lighting is enabled, we can obtain different tones by controlling the stroke density, and therefore, the density of seed points. We apply half-toning techniques to this seed point selection process. Freudenberg et al. [4] achieve real-time halftoning by comparing a pre-designed halftone texture with the target shading intensity. Inspired by Freudenberg's hardware threshold method, we assign each input primitive (here points) with an evenly distributed random value. The range of random values matches that of the shading intensity. Hardware threshold operation is made by comparing the random value with the illumination value. Thus lesser seed points will be selected for highlight regions while more seed points will be selected for dark regions. Therefore, the shading intensity is conveyed by spatial point density distribution.

The seed points are rendered using OpenGL command GL_POINTS with size 1. We utilize the depth buffer from the previous 3D modeling rendering to do the depth test so that only visible seed points remain in the frame buffer. For each seed point, in a vertex program, the light intensity with respect to the current light source is computed and stored in the blue channel of the output color if it is larger than the pre-stored value, otherwise the vertex's blue channel remains unchanged. At the pixel level, the visible seed points are combined with the 3D model image by a small fragment program. Pseudo codes for both the vertex program (vertex program 2) and fragment program for rendering of seed points are provided in Appendix A

3.3 Feature Points Extraction and Classification

This step involves two operations. Firstly, edge detection is performed on the above obtained alpha channel by applying any edge filter and then, the obtained values are further analyzed and combined with the information from other channel to form feature points.

The edge detection can be efficiently done using the multitexture and texture rectangle features provided by the current commodity graphics hardware. The details can be found in [7] and [11]. A wide range of edge filters can be implemented.

Once the edge detection is done, the entire frame buffer is read back into the main memory. The alpha buffer is scanned through and each pixel is classified as either a strong edge point or a weak edge point based on its alpha channel value and a pre-selected threshold value. If only strong silhouette edges are of interest, weak points are discarded. Otherwise, weak edge points may also be depicted but using a different style. Optionally, their blue components are further tested so that only the weak edge points with non-zero blue channel value are regarded as feature points. For some styles, seed points can also be used directly as feature points independent of their alpha values.

3.4 Feature Points Stylization

Various NPR styles are achieved by applying different textures (e.g., strokes, furs, and dots) on feature points rendered as textured quads. Specifically, for each feature point, we generate four vertices on the view plane representing the four corners of the quad centered at the feature point. Texture coordinates (0,0), (0,1), (1,0), and (1,1) are assigned to the four vertices respectively. The quad is placed in such a direction that the texture orientation follows the perpendicular direction to the projected normal vector (stored in Red and Green channels). Figure 2 shows various styles achieved by INSPIRE using different textures. To fully utilize the efficiency of the GPU computation, we calculate the vertex coordinates of the quad in vertex shaders, similar to [18].

4 Results

We have implemented our NPR pipeline on an 800MHz Pentium III PC with OpenGL under Windows 2000. The main memory is 1GB. We use an nVidia GeForce4 Ti4600 graphics card with 128MB video memory. All images are rendered using 640×640 resolution.

We are able to generate various NPR styles within a unified framework. Figure 2 demonstrates representative styles that are generated by INSPIRE. For comparison, we use the same model for illustrating all the styles. For each style image, the zoom-in image of a part of the complete image indicated by a square is illustrated at the top-left corner, under which the stroke texture is also illustrated.

1. **Cartoon style:** Figure 2(a) illustrates this style, which is done by disabling seed points. The outline is created by rendering the strong edge points with strong stroke texture. The interior of the model is painted with a solid color.

2. **Pencil style:** Figure 2(b) illustrates this style, which is similar to Figure 2(a) with seed points. The threshold for selecting the strong edge points is decreased to a small number so that more strong edge points are selected.

3. **One-Tone style:** Figure 2(c) illustrates this style. Lighting is enabled in this style. Each seed point is drawn as a dot if its light intensity is less than a pre-assigned threshold. The outlines (strong edge points) of the model is drawn with a dot texture.

4. **Two-Tone style:** Figure 2(d) illustrates this style. Instead of one threshold as in the One-Tone style, there are two thresholds, high and low thresholds. The seed points with light intensity larger than the high threshold are colored white, the seed points with light intensity less thn the low threshold are colored black, the remaining seed points are ignored. All of these are placed on top of a gray background.

   For Figure 2(c) and (d), the seed point distribution are increased to about 50% to achieve the effects.

5. **Curly fur style:** Figure 2(e) illustrates this style. Here a "zizzag" texture is drawn at each seed point. To make the texture more visible at each seed point, the basic quad's size drawn at each seed point is increased by a constant of 4.

6. **Fur style:** Figure 2(f) illustrates this style. A hairy texture is placed on strong edge points and seed points. To control the fur density, we can increase or decrease the number of seed points. Alpha blending is enabled to achieve smoothness.

7. **Sketch style:** Figure 2(g) illustrates this style. Here a straight line texture is simply used with alpha blending enabled at each seed point.

8. **Indian-ink style:** Figure 2(h) illustrates this style, which is very similar to One-Tone style, but seed points are drawn with a strong stroke texture if their light intensity is less than preselected threshold.

We have also demonstrated another style for medical object illustration (Figure 3). This style is created by taking weak edge points as feature points. However, to avoid excessive number of feature points, another threshold is specified to filter away weak edge points of extremely small values. No seed points are taken as feature points.

Finally, we demonstrate the rendering efficiency of our system in Table 1. All objects are point-based models. The frame rate is average performance of typical interaction with the models. Performance difference between rendering of different styles is negligible since all styles share the same rendering pipeline.

Table 1: *Rendering performance (image resolution: 640x640).*

Model	# of Points	FPS
Bunny	34,834	20.0
Elephant	155,688	15.1
Dragon	437,645	8.5
Armadillo	172,974	10.4
Hip	530,168	7.41

5 Conclusions and Future Work

We have presented an NPR system that features stylized rendering and interactive navigation in a unified framework. We have achieved various styles. By designing a compact hardware implementation, interactive rendering speed can be obtained. Specifically, to achieve common features of non-photorealistic rendering, we have developed procedures to first extract feature points and and then employing different geometric primitives (or strokes) to illustrate these feature points. By employing different strokes for different feature points through texture mapping, we can obtain a number of artistic effects. We have also developed methods for maintaining consistent animation and screen space stroke density during navigation. We have leveraged modern commodity graphics hardware by deferring almost all operations to the GPU. The results obtained have shown that our NPR system is both *flexible*, in terms of various stylizations, *efficient*, in terms of interactivity, and *consistent* in terms of animation.

The above efforts have created a framework for developing additional artistic styles in the future. Research is already under way in this direction. Currently, since we define seed points as a subset of polygon vertices or points of point-based models, the maximum number of seed points is hence fixed. When the camera gets very close to the object, no new points can be added, hence, the stroke density starts to decrease. We wish to address this issue in future. A potential solution is to either generate new points, or generate texture patterns with more textured strokes. We are investigating efficient approaches for hardware implementation of either approaches. One bottleneck of our system is the frame-buffer read-back operation in step two of the rendering pipeline. For large image resolutions this will have a noticeable effect on rendering speed. We will investigate methods that allow us to generate larger resolution images while keeping the intermediate image resolution moderate. We will also analyze new features of future graphics hardware to hopefully avoid this bottleneck.

6 Acknowledgements

Our thanks to Amit Shesh for proofreading the paper. We also thank Stanford Computer Graphics Laboratory for the bunny, dragon and armadillo models and GRAIL at the University of Washington for the elephant model. The hip and teeth models are from Cyberware sample repository.

Support for this work has included a Computer Science Department Start-Up Grant and a Grant-in-Aid of Research, Artistry, and Scholarship, 2002-2003, Digital Technology Center Seed Grant 2002, all from the University of Minnesota; and NSF CAREER ACI-0238486. This work was supported also in part 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.

References

[1] B. Chen and M. X. Nguyen. POP: A hybrid point and polygon rendering system for large data. *Proc. of IEEE Visualization '01*, pages 45–52, Oct. 2001.

[2] D. Cornish, A. Rowan, and D. Luebke. View-dependent particles for interactive non-photorealistic rendering. In *Proceedings of Graphics Interface 2001*, pages 151–158, 2001.

[3] P. Decaudin. Cartoon-looking rendering of 3d-scenes. Inria technical report 2919, university de technologie de compiegne, france, June 1996.

[4] B. Freudenberg, M. Masuch, and T. Strothotte. Real-time halftoning: A primitive for non-photorealistic shading. In *Proceedings 13th Eurographics Workshop on Rendering Techniques*, pages 227–231, 2002.

[5] A. Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *SIGGRAPH 98 Conference Proceedings*, pages 453–460, July 1998.

[6] A. Hertzmann and D. Zorin. Illustrating smooth surfaces. *Proceedings of SIGGRAPH 2000*, pages 517–526, July 2000. Held in New Orleans, Louisianna.

[7] H. H. M. Hadwiger, Thomas Theul and E. Grller. Hardware-accelerated hiqh-quality filtering on pc graphics hardware. In *Proceedings of Vision, Modeling, and Visualization*, pages 105–112, 2001.

[8] A. Majumder and M. Gopi. Hardware accelarated real time charcoal rendering. In *Non-Photorealistic Animation and Rendering 2002 (NPAR '02), Annecy, France*.

[9] L. Markosian, M. A. Kowalski, S. J. Trychin, L. D. Bourdev, D. Goldstein, and J. F. Hughes. Real-time nonphotorealistic rendering. In *SIGGRAPH 97 Conference Proceedings*, pages 415–420, Aug. 1997.

[10] B. J. Meier. Painterly rendering for animation. In *SIGGRAPH 96 Conference Proceedings*, pages 477–484, Aug. 1996.

[11] M. Nienhaus and J. Doellner. Edge-enhancement – an algorithm for real-time non-photorealistic rendering. pages 346–353, 2003.

[12] R. Raskar and M. Cohen. Image precision silhouette edges (color plate S. 231). In *Proceedings of the Conference on the 1999 Symposium on interactive 3D Graphics*, pages 135–140, Apr. 1999.

[13] T. Saito and T. Takahashi. Comprehensible rendering of 3-D shapes. *Computer Graphics*, 24(4):197–206, 1990.

[14] M. P. Salisbury, S. E. Anderson, R. Barzel, and D. H. Salesin. Interactive pen–and–ink illustration. In A. Glassner, editor, *Proceedings of SIGGRAPH '94 24–29, 1994)*, pages 101–108, July 1994.

[15] M. P. Salisbury, M. T. Wong, J. F. Hughes, and D. H. Salesin. Orientable textures for image-based pen-and-ink illustration. In *SIGGRAPH 97 Conference Proceedings*, pages 401–406, Aug. 1997.

[16] A. Secord. Weighted voronoi stippling. In *Non-Photorealistic Animation and Rendering 2002 (NPAR '02), Annecy, France*, pages 37–43, June 2002.

[17] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli. Fast Shadows and Lighting Effects Using Texture Mapping. In *Computer Graphics (SIGGRAPH 92)*, pages 249–252, July 1992.

[18] H. Xu and B. Chen. Stylized visualization of 3d scanned outdoor environments. In *IEEE Visualization 2003 (under review)*, 2003.

A Vertex and Pixel Shader Pseudo-codes

VertexProgram1

1: INPUT: $Vertex\ P$, $Normal\ N$
2: $o[HPOS] \leftarrow P$ transformed by view projection matrix
3: $P_0 \leftarrow P$ in the eye space
4: $N_0 \leftarrow N$ in the eye space
5: $V_0 \leftarrow$ view direction from P to (0,0,0) in the eye space
6: $o[COL0].red \leftarrow N_0.x$
7: $o[COL0].green \leftarrow N_0.y$
8: $o[COL0].blue \leftarrow 0$
9: $o[COL0].alpha \leftarrow \max(dot(N_0, V_0), 0)$

VertexProgram2

1: INPUT: $Vertex\ P$, $Normal\ N$, $LightDirection\ L$
2: INPUT: pre-generated random value r at P
3: CONSTANT: $EPSILON \leftarrow 1/255$
4: $o[HPOS] \leftarrow P$ transformed by view projection matrix
5: $P_0 \leftarrow P$ in the eye space
6: $N_0 \leftarrow N$ in the eye space

7: $L_0 \leftarrow L$ in the eye space
8: $V_0 \leftarrow$ view direction from P_0 to (0,0,0)
9: $intensity \leftarrow$ light shading using L_0, N_0, V_0
10: **if** $intensity < r$ **then**
11: $\quad intensity \leftarrow EPSILON$
12: **end if**
13: $o[COL0].red \leftarrow 0$
14: $o[COL0].green \leftarrow 0$
15: $o[COL0].blue \leftarrow intensity$
16: $o[COL0].alpha \leftarrow 0$
17: $o[TEX0] \leftarrow 2D$ coordinate of the projection of P

FragmentProgram is applied to each rasterized pixel from **VertexProgram2**. If at **VertexProgram1** level, TEX0 is bound to the frame buffer of Step 1, then at pixel level, $TEX0$ contains the current value of the frame buffer, $COL0$ contains the incoming color value of the pixel.

FragmentProgram

1: $output \leftarrow TEX0 + COL0$

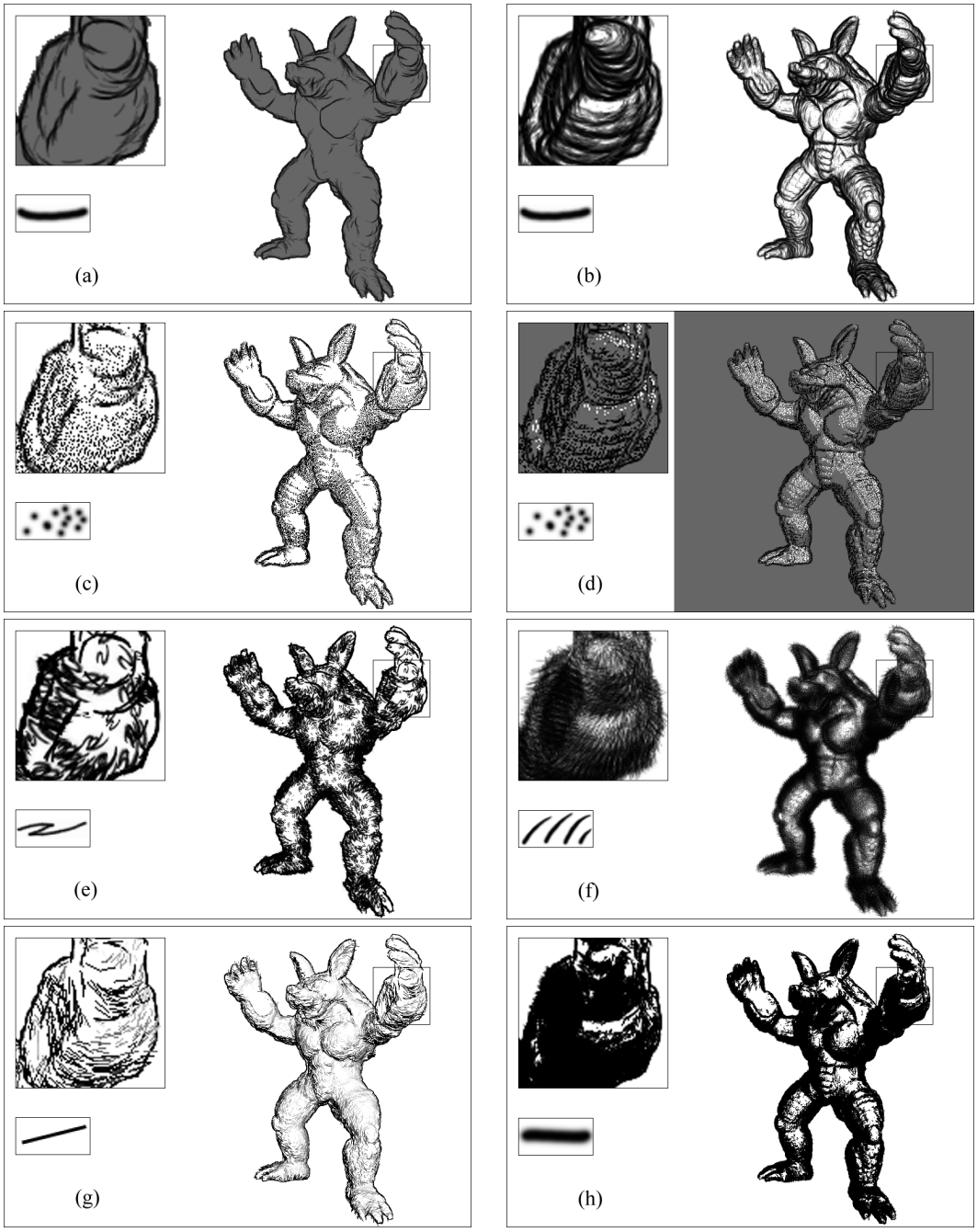

Figure 2: *Rendering of the Armadillo object using different NPR styles (the top-left corner small images are the zoom-ins of the right images; under the zoom-ins are the stroke textures used): (a) Cartoon style (interior solid color; only strokes on strong edge points), (b) Pencil style (similar to (a) without interior color, more strong edge points are selected), (c) One-Tone style (strong edge points drawn using dots texture; lighting enabled; seed points with intensity above threshold drawn as black dots), (d) Two-Tone style (similar to (c), two thresholds used; white and black dots are drawn for seed points above larger threshold and seed points below smaller threshold, respectively, against a gray background), (e) Curly fur style (weak points drawn using "zig-zag" textures in large sizes), (f) Fur style (edge points drawn using hair textures; blending enabled for smoothness), (g) Sketch style (seed points drawn using straight line textures; alpha blending enabled), (h) Indian-ink style (similar to one-tone style; dense seed points with light intensity less than pre-selected threshold are drawn using thick stroke textures).*

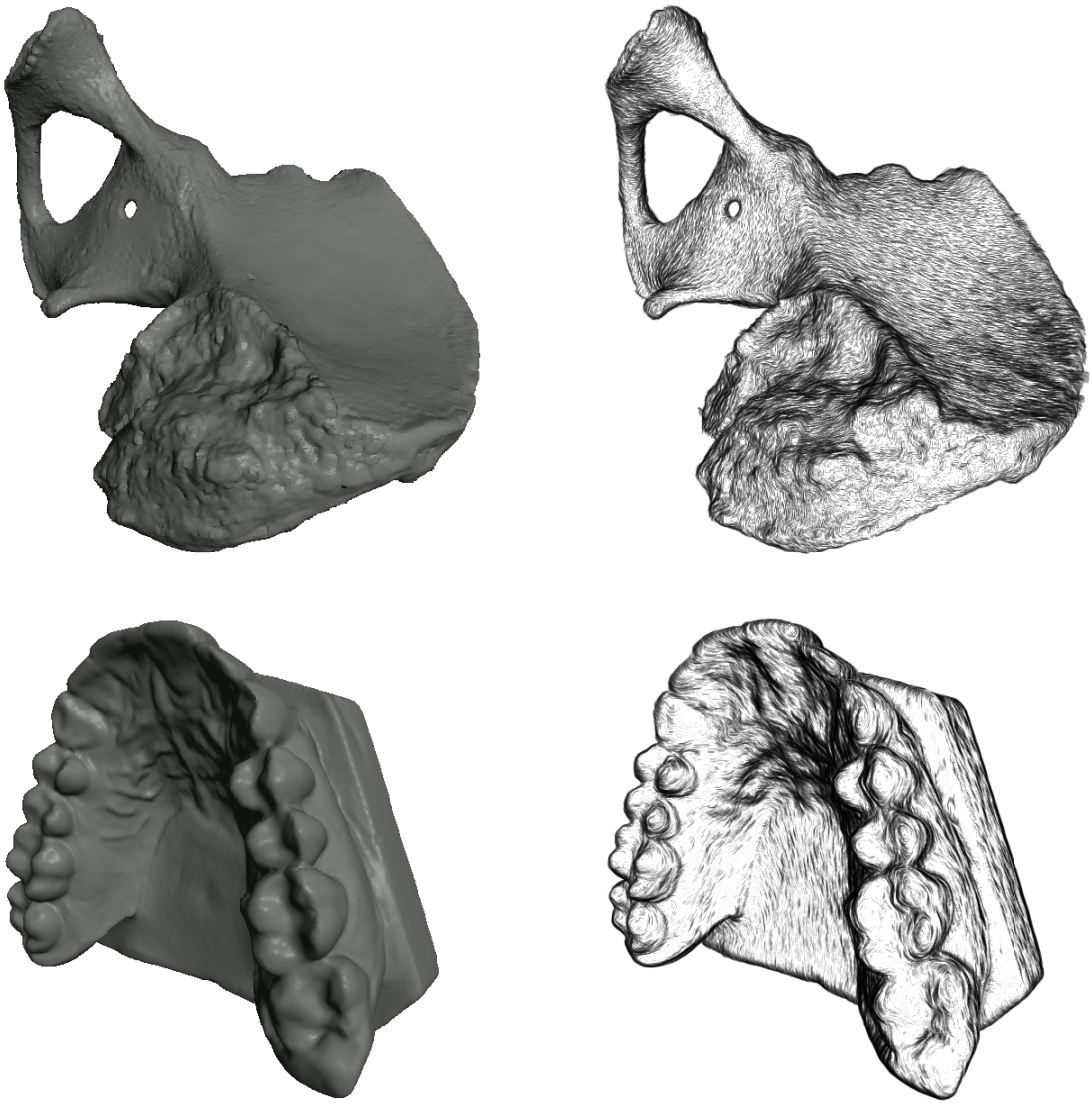

Figure 3: *Another style for medical illustration. The images to the right depict pencil drawings of the corresponding left images, which are generated photorealistically.*