Real-Time Pencil Rendering

Hyunjun Lee* POSTECH
Sungtae Kwon† POSTECH
Seungyong Lee‡ POSTECH

Abstract

This paper presents a real-time technique for rendering 3D meshes in the pencil drawing style. We analyze the characteristics of pencil drawing and incorporate them into the rendering process, which is fully implemented on a GPU. For object contours, we propose a multiple contour drawing technique that imitates trial-and-errors of human in contour drawing. For interior shading, we present a simple approach for mapping oriented textures onto an object surface. To improve the quality of pencil rendering, we generate and map pencil textures that reflect the properties of graphite pencils and paper. We show several rendering examples that demonstrate the high performance of the proposed technique in terms of speed and quality.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms

Keywords: non-photorealistic rendering, pencil drawing, GPU rendering, multiple contour drawing, pencil texture

1 Introduction

A variety of non-photorealistic rendering techniques have been developed to provide software tools that simulate physical materials for drawing and painting, such as pen-and-ink [Winkenbach and Salesin 1994; Winkenbach and Salesin 1996; Lake et al. 2000; Deussen and Strothotte 2000; Wilson and Ma 2004], watercolor [Curtis et al. 1997; Laerhoven and Reeth 2005; Luft and Deussen 2005], and charcoal [Bleser et al. 1988; Majumder and Gopi 2002]. Diverse physical materials used in the real world have inherent characteristics, which determine the features and the flavor of drawing and painting results. A pencil is one of the most available and easy-to-handle tools for drawing. Although pencil drawing might be used as a basis for sophisticated painting afterwards, it has appealed as an interesting art form on its own. Pencil drawings can represent soft impressions of objects with pencil strokes, where the exact tones of strokes are determined by the properties of pencil and paper.

To provide a tool for pencil drawing, a simulation-based approach was proposed to mimic the characteristics of graphite pencils combined with paper effects [Sousa and Buchanan 1999; Sousa and Buchanan 2000]. There exists an image-based approach to obtain pencil drawings which transforms an input image into the pencil drawing style [Mao et al. 2001]. Pen-and-ink illustration can be considered similar to pencil drawing in that the tones, materials, and shapes of objects are mostly represented by strokes. Excellent techniques were developed to generate pen-and-ink illustrations from 3D objects [Winkenbach and Salesin 1994; Lake et al. 2000; Wilson and Ma 2004]. Real-time hatching techniques [Praun et al. 2001; Webb et al. 2002] can also be applied to pen-and-ink illustration of 3D meshes.

Despite these good results, real-time rendering of 3D meshes in the pencil drawing style has not been fully addressed in the graphics literature. Although it can generate very natural and realistic effects, the simulation-based approach for pencil drawing has a limitation in the rendering speed. The 2D image-based approach can be applied to the rendered images of 3D meshes but will not be effective for pencil rendering of 3D meshes in an animation. Pen-and-ink illustration and real-time hatching techniques provide a concrete basis for real-time pencil rendering. However, the inherent characteristics of pencil drawings differ from pen-and-ink illustrations and these techniques cannot be directly applied.

In this paper, we propose a real-time pencil rendering technique, which produces pencil drawing style images of an input 3D mesh in real time. Specifically, this paper presents contributions in the following aspects of pencil rendering;

- **Multiple contour drawing:** When a person draws an object with a pencil, the object contours are usually drawn a few times with slightly different shapes. We present an effective technique to handle this characteristic of pencil drawing (see Figure 3).
- **Simpler interior shading:** The real-time hatching technique [Praun et al. 2001] provides an effective solution for shading an object interior with oriented textures. In this paper, we present a simpler technique that achieves similar effects without using lapped textures [Praun et al. 2000]. Our technique is fully implemented on a GPU and can incorporate paper effects into interior shading.
- **Pencil texture generation and mapping:** We present techniques for generating and mapping pencil textures that reflect characteristics of graphite pencils and paper. Especially, we observe overlapping and white pixel effects from real pencil drawing examples and implement them to improve the quality of pencil rendering.
- **GPU-based rendering framework:** Most of the components in our pencil rendering technique are implemented on a GPU. Consequently, the rendering speed is quite fast; 15 to 60 frames per second for reasonably complicated meshes.

2 Related Work

Sousa and Buchanan [1999; 2000] analyzed and simulated the properties of various materials for pencil drawing. They modeled pencils, paper, erasers, and blenders, and their simulation results show nice effects comparable to real pencil drawings. However, this approach needs heavy computation for simulation, which may prohibit real-time rendering.

*crowlove@postech.ac.kr, http://home.postech.ac.kr/∼crowlove
†zelong@postech.ac.kr, http://home.postech.ac.kr/∼zelong
‡leesy@postech.ac.kr, http://www.postech.ac.kr/∼leesy

Mao *et al*. [2001] proposed an image transformation technique that changes an input image to have the pencil drawing style. They construct a 2D vector field on an image and apply the line integral convolution to the vector field to generate pencil strokes. In this paper, we consider 3D meshes as the input for pencil rendering, instead of 2D images.

Winkenbach and Salesin [1994] developed a rendering technique that generates pen-and-ink illustrations from 3D meshes and showed impressive results. Lake *et al*. [2000] presented real-time rendering techniques in various styles, including pencil sketch shading, with projected textures. Saito and Takahashi [1990] introduced line drawing on an object surface based on a curvature field for comprehensible rendering of 3D shapes. Salisbury *et al*. [1997] compute a direction field on a 2D image and map orientable textures to produce pen-and-ink illustrations. However, in these techniques, the inherent characteristics of pencil drawing are not really investigated.

With the advances of graphics hardware, real-time techniques for non-photorealistic rendering have been developed. Praun *et al*. [2001] presented a real-time hatching technique, which can map stroke textures onto a 3D mesh surface with level-of-detail control. Webb *et al*. [2002] extended this technique with volume rendering hardware and pixel shaders to enhance the control of hatching tones. Majumder and Gopi [2002] introduced a real-time technique for charcoal rendering of 3D objects, which utilizes graphics hardware. In this paper, we present a real-time rendering technique for pencil drawing of 3D meshes, which can handle texture mapping of orientable strokes with a simpler approach than previous work. With the simplicity, our technique is readily implemented by GPU programming.

For stylization of 3D object contours, Mohr and Gleicher [2001] used jittering to mimic the multiple contours in a pencil drawing. Loviscach [2004] demonstrated that multiple contour drawing can be supported by graphics hardware. However, in this paper, we present a more efficient technique for multiple contour drawing, which is implemented by a pixel shader with multi-texturing. Since our technique simply distorts the underlying plane of the contours to produce different trajectories, it can handle arbitrarily complex contours very efficiently.

3 System Overview

In a drawing, objects are usually depicted by contours and interior shading. In our system for pencil rendering of a 3D mesh, the run-time process consists of contour drawing and interior shading parts whose output images are combined to generate the final image (see Figure 1).

In contour drawing, we first render the input mesh to obtain normal and depth images from the shape. The normal and depth images are then used to detect contours in image space. To make the extracted contours look natural, similar to pencil drawing by a human, we draw the contours a few times with slightly distorted trajectories. The pencil tones of the contours are represented by mapping a pencil texture. The normal and depth images are obtained by OpenGL rendering of a mesh, and the succeeding image-based contour detection is performed by a pixel shader. After contours have been obtained, another pixel shader handles multiple contour drawing and pencil texture mapping at the same time.

The interior shading part starts with intensity computation for the input mesh to determine the brightness of pixels in the rendered image. To introduce pencil drawing tones, we adjust the contrast of

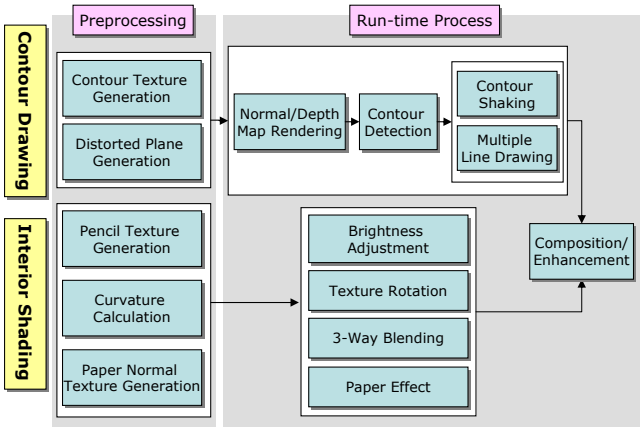


Figure 1: System overview

pixel brightness, similar to charcoal rendering [Majumder and Gopi 2002]. To express pencil strokes, we use texture rotation and 3-way blending for mapping oriented pencil textures in the principal curvature directions. To represent interaction among a pencil and paper, we model paper as a height field and the paper normals are used to control the pixel intensities during the mapping of oriented pencil textures. In interior shading, all steps are performed in a single pixel shader, except that a simple vertex shader is used to project the 3D principal curvature directions at vertices onto the 2D image space.

After contour drawing and interior shading, the resulting images are composed and enhanced to produce the final image. This step is fully performed in a pixel shader using the images from contour drawing and interior shading as texture images mapped onto the whole screen.

In the preprocessing step, we prepare several data that will be used in the run-time process. This step needs to be performed only once for each input model, thus the computation speed is not really important. For contour drawing, we generate a texture to be used for giving pencil tones to the contours. This texture can be simpler than the pencil texture used for interior shading because contours occupy small narrow parts of the rendered image without revealing the texture details. We also construct distorted planes to be used for modifying contour trajectories. For interior shading, we generate a set of pencil textures to represent different intensities of an object interior. The principal curvature directions are computed for mesh vertices to determine the orientations in interior texture mapping. A paper model is constructed as a height field and the normal vector at each pixel is computed to provide paper effects in interior texture mapping.

Once all necessary data have been prepared in preprocessing, the run-time process for pencil rendering of a 3D mesh is fully supported by graphics hardware and performed very fast with varying camera and lighting parameters. In contour drawing, we mimic the trial-and-error of human by multiple contour drawing with distorted trajectories. In interior shading, pencil textures and paper effects that reflect characteristics of graphite pencils provide natural impressions with an object interior. The details of the techniques are described in the following sections.

4 Contour Drawing

4.1 Contour detection

Many algorithms exist for detecting contours from a 3D mesh. Among them, we use an image space approach, which can detect contours in real time using GPU and enables us to apply image processing techniques to detected contours [Isenberg et al. 2003; Nienhaus and Doellner 2003]. This approach detects contours using normal and depth images acquired from a 3D object. Suggestive contours [DeCarlo et al. 2003] can also be extracted in the 2D image space.

The result of contour detection with an image space approach is represented by a black and white or gray scale image. To introduce lighting effects onto the contours, we modify the pixel values on the contours using the normal information at the pixels;

$$I_c' = l_c I_c, \quad (1)$$

where I_c and I_c' denote the original and modified pixel values of contours, respectively. l_c is the light effect factor, which is computed with normal and light directions at each pixel. As a result, the information for detected contours is stored as a gray scale image. We normalize the gray scale values between 0 and 1, where 0 is black (the darkest) and 1 is white (the brightest). Both contour detection and lighting effects on the contours are performed on image pixels and are easily implemented with a pixel shader.

4.2 Contour shaking

Contours acquired by the method in Section 4.1 are artificial and may differ from those drawn by hand. Adding errors to contours can help to imitate hand drawing [Loviscach 2002; Ho and Komiya 2004]. Although a person cannot draw contours without errors, these errors are usually bounded in some limited range. When the drawing error increases, a person will try to stick to the target contour, reducing the error. This implies that shaking patterns of contour drawing can be approximated by the shape of a periodic function.

With this idea, we use a sine function to imitate the shaking errors in contour drawing;

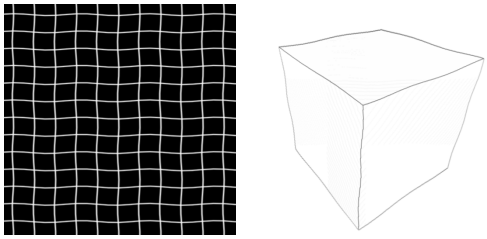
$$y = a\sin(bx+c)+r, \quad (2)$$

where a and b determine the range and period of the error, respectively. c is a shift of a periodic function and r adds randomness to the error value.

We can perturb contours by adding the errors computed with Eq. (2) directly to the contour points. However, this perturbation process will be difficult to implement by a pixel shader because a pixel shader cannot write a value to another pixel except to the current one. We resolve this problem by texturing the contour image onto a distorted plane, instead of shaking contours directly.

In the preprocessing step, we divide the screen space into regularly sized rectangles. We assign the coordinates on the contour image to the texture coordinates of the rectangles so that the contour image may be redrawn by texturing the rectangles with the contour image. Then, distorted contours can be obtained by adding errors to the texture coordinates of the rectangles and performing texture mapping afterwards. The distorted texture coordinates for the rectangles are computed with Eq. (2) in the preprocessing step and repeatedly used in the run-time rendering process.

Figure 2(a) shows an example of the distorted plane with perturbed rectangles. Figure 2(b) shows shaken contours obtained by texturing a contour image onto a distorted plane.



(a) distorted plane (b) shaken contours

Figure 2: Contour shaking

4.3 Multiple contour drawing

In real pencil drawing, we can observe that a contour is usually composed of overlapped multiple lines, not a single one. In our contour drawing, we can easily achieve this effect by composing several distorted contour images. With multi-texturing, we can assign several texture coordinates computed by different sine functions to the rectangles on the screen space. Then, several contours distorted in different ways are obtained by sampling multiple texture values at the same time in a pixel shader. From experiments, we have discovered that the number of overlapping works best at 3-5.

When combining multiple texture values in the pixel shader, we put a darker intensity to a pixel with much overlapping of contours. This strategy imitates a darkening effect induced by overlapping pencil strokes. In addition, we project a contour pencil texture, created in the preprocessing step, onto the contours. This texture mapping introduces impression of a pencil material to the contours and can be performed in the same pixel shader.

Figure 3 shows an example of multiple contour drawing. The figures on the left are contour images created from different sine functions. The right figure is the merging result of the left images. From the zoom-in image on the right, we can see that overlapping contours and pencil materials are successfully implemented.

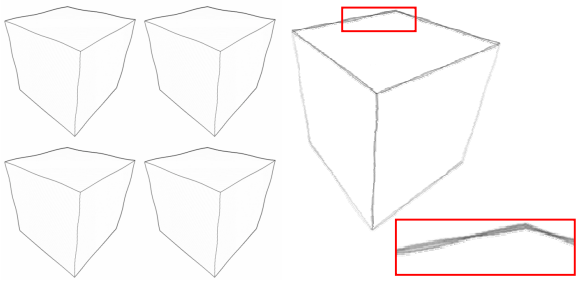


Figure 3: Multiple contour textures and the merged image

5 Interior Shading

In pencil drawing, various effects are considered for the interior of an object, such as the color and material of a pencil, the directions and thickness of strokes, the paper material, and the blending

and eraser effects [Sousa and Buchanan 1999]. In the preprocessing step, we create pencil textures that reflect the color and material of a pencil represented with oriented strokes. We also make a paper texture that mimics the material of paper. In the run-time process, pencil textures are rotated with the principal curvature directions and blended by multi-texturing, representing the directions and brightness of strokes on the object interior. The paper texture is used to implement the interaction of a pencil with paper in interior stroke mapping. The run-time process for interior shading is implemented in a single pixel shader.

5.1 Pencil texture generation

Pencil textures for interior shading should express essential properties of pencil strokes, such as brightness and thickness. In addition, strokes in textures should have a specific direction to depict the voluminousness of an object along the direction. To obtain such pencil textures, we propose a texture generation technique based on [Webb et al. 2002] and use a pixel shader to implement the technique.

In [Webb et al. 2002], the brightness of an object part is represented by the density of strokes. On the other hand, a pencil drawing expresses the brightness of an object using the brightness of a pencil line itself rather than the density. Therefore, in our texture generation technique, pencil strokes are placed uniformly in each texture image, as shown in Figure 4. To obtain strokes with similar orientations, pencil lines are drawn along a specific direction with some angle perturbation. The brightness of a pencil texture is modulated by repeatedly drawing uniformly distributed strokes. That is, to make a texture darker, we draw uniformly distributed strokes with perturbation on top of the current texture, where pixels are darkened by overlapped strokes.

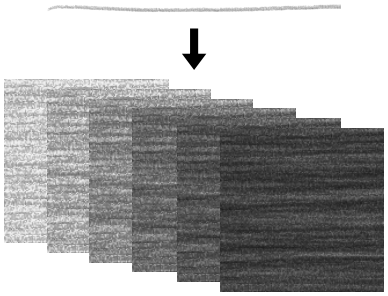


Figure 4: Initial stroke and generated textures

In real pencil drawing, the increase of darkness becomes smaller as pencil strokes are overlapped on the same region repeatedly. To incorporate this observation, the brightness of a pixel in a pencil texture is determined by

$$\begin{aligned} c_t' &= c_t - \mu_b c_a, \\ c_a &= c_t(1.0 - c_s), \end{aligned} \tag{3}$$

where c_t and c_t' are the current and updated texture colors, respectively. c_s is the stroke color and c_a is the maximum amount of darkness that can be increased by the stroke. The texture and stroke colors are represented by values between 0 (black) and 1 (white). The user controlled parameter μ_b determines how much stroke darkness is added to the current texture. From experiments, we found a value between 0.1 and 0.3 is proper for μ_b.

We also observe from real pencil drawing that the regions not stained with black lead at the first stroking tend to remain white even though strokes are added on the paper many times afterwards. To achieve this property, we add a conditional formula at the end of Eq. (3);

$$\text{If } c_t \text{ is white enough, } c_a = \mu_w c_a. \tag{4}$$

With a small value of μ_w, we can preserve white pixels. Empirically, a value between 0.3 and 0.5 is good for μ_w.

We use a 3D texture space to store various tones of pencil textures [Webb et al. 2002]. Since no stroke is needed for the brightest region, a white image is stored as the brightest texture. Including the white image, 32 pencil textures are stored in the 3D texture space. We make the colors of these textures uniformly distributed from the brightest to the darkest when the textures are generated by overlapping strokes.

5.2 Brightness adjustment

The object surface brightness for pencil rendering can be computed by a general shading technique, e.g., Gouraud shading. Such computed brightness may not be sufficient for expressing a voluminous object. Majumder and Gopi [2002] enforce the voluminousness by enhancing the contrast of colors, where a brightness value is replaced by its square.

In pencil rendering, enforcing contrast between bright and dark regions is still desirable. However, in this case, intensity variation in a bright region is more important than the contrast between bright and dark regions. We adjust the brightness by using the square root function to expand bright regions with enforced intensity variation. The brightness adjustment is performed at each pixel in the pixel shader for interior shading.

5.3 Texture rotation

In a drawing, stroke directions generally follow the principal curvature directions [Girshick et al. 2000; Hertzmann and Zorin 2000]. To apply this rule to our interior shading, we compute and store the curvature directions at mesh vertices in the preprocessing step. For curvature direction computation, we adopt the technique based on the tensor field computation used in [Alliez et al. 2003]. In the regions where principal directions are not clearly defined, such as spherical and planar regions, we determine the principal directions by propagating them from neighbor vertices.

In the run-time rendering, pencil textures are mapped onto the object interior in the image space. In the mapping, we rotate a pencil texture at a vertex along the minimum curvature direction and the rotated texture is projected onto neighbor faces of the vertex (see Figure 5). The 3D curvature direction at a vertex can be transmitted to the vertex shader as 3D texture coordinates. In the vertex shader, the 3D curvature direction is projected onto the 2D image space using the OpenGL projection matrix. With the projected direction, we calculate a rotation angle θ in the pixel shader for interior shading. By rotating texture coordinates by $-\theta$, instead of rotating a texture, we can draw a pencil texture along the desired direction. To obtain a constant direction of a pencil texture in the neighbor faces of a vertex, as shown in Figure 5, we assign the same 3D texture coordinates (curvature direction) to the vertex and its 1-ring neighborhood.

5.4 3-way blending

As a result of texture rotation performed at each vertex, each face is assigned three pencil textures with different orientations. The

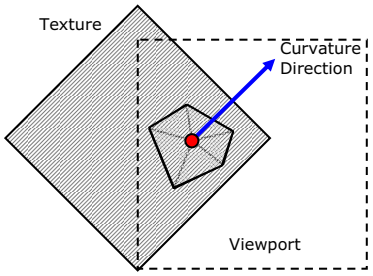
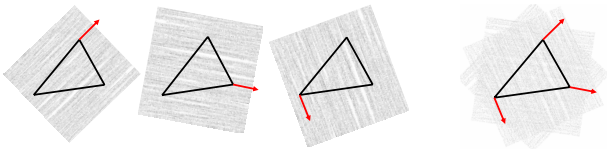


Figure 5: Texture rotation

orientations come from the minimum curvature directions at the three vertices of a face. The actual mapping of three textures onto a face is implemented by multi-texturing with three different 3D texture coordinates for each vertex. In the pixel shader for interior shading, texture rotation is performed three times and three rotated texture values are blended together (see Figure 6).



(a) shading for each vertex (b) 3-way blending

Figure 6: 3-way blending

Usually, the interior strokes for pencil drawing are expressed by lines and very smooth curves. Pencil stokes with different directions are overlapped several times in the regions with complicated shapes. Our 3-way blending of textures effectively handles these effects in interior shading. In a region with a simple shape, such as a cylindrical part, the principal curvature directions are almost constant and 3-way blending generates consistent strokes along the shape. Around an object part with a complex shape, the principal curvature directions change dynamically and 3-way blending gives overlapping effects of differently oriented strokes.

Although texture rotation is independently performed for each face, 3-way blending helps remove most of the discontinuity of pencil strokes among adjacent faces. With 3-way blending, for two faces sharing an edge, two directions for texture rotation are the same. Similarly, two faces sharing a vertex have one common direction for texture rotation. Consequently, pencil strokes look almost continuous along the object surface, as demonstrated in the rendering results shown in Section 7.

In real pencil drawing, we can observe that more lines are drawn with different directions on dark regions. To simulate this effect, we draw additional strokes for dark regions in interior shading. For a dark region, after a pencil texture is mapped with the minimum curvature direction, the same texture but with the maximum curvature direction is mapped on the same region again. Figure 7 shows an example.

5.5 Paper effect

Paper has various material properties, such as roughness and textures, which affect the drawing results. To apply the paper effect to our pencil rendering, we make a paper texture that resembles the surface of the real paper. We adopt the method proposed by Curtis

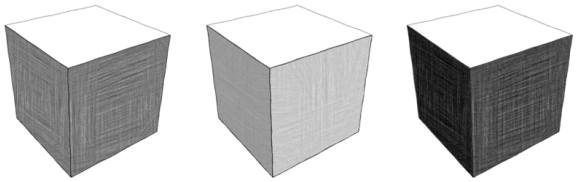


Figure 7: Cross hatching in dark regions

et al. [Curtis et al. 1997], which generates a height map for paper using noise. From the height map, we obtain a normal map in the preprocessing step.

With the normal map, we apply the paper effect to interior shading by considering interaction among the pencil and paper in a stroke. If the stroke direction is opposite to the paper normal, we darken the pencil texture color to simulate more black lead being stained on the paper. Similarly, the texture color is attenuated if the stoke direction and paper normal are similar. See Figure 8 for an illustration.

This paper effect can be implemented by

$$\begin{aligned} c_t' &= c_t - \mu_p(-\mathbf{d}\cdot\mathbf{n}) \\ &= c_t + \mu_p(\mathbf{d}\cdot\mathbf{n}), \end{aligned} \tag{5}$$

where c_t and c_t' are the current and modified texture colors, respectively. $\mathbf{d}$ is the curvature direction and $\mathbf{n}$ is the normal direction from the paper model. μ_p is a weighting factor, empirically, whose value is selected between 0.0 and 0.1.

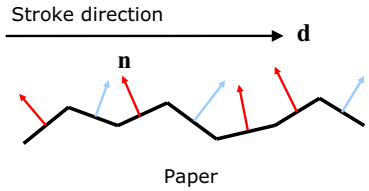


Figure 8: Paper effect

6 Composition and Enhancement

As explained in Section 3, we render contours and the object interior independently and the resulting images are merged to obtain the final pencil rendering image. We can perform the merging by a pixel shader, where the contour and interior images are texture-mapped onto a big rectangle in the screen space.

Since contour drawing usually precedes interior shading in real pencil drawing, we first apply contour image values to the final pixel intensities and then add interior image values. On the regions where the interior shading overlaps with already drawn contours, we attenuate the interior intensities before blending with contour intensities. This attenuation allows contours to remain visible while reflecting interior textures.

In Section 5.2, we described a method to adjust the brightness for interior shading. Similarly, the contrast of the result image can be modulated in the pixel shader after the merging. In this case, we adapt the contrast enhancement proposed by Majumder and Gopi [2002], which makes bright regions brighter while dark regions become darker.

Finally, we blend the paper texture onto the background since the paper is not completely white. The pixel and paper colors are blended by

$$c_i' = c_i c_p, \quad (6)$$

where c_i and c_i' are the original and blended colors, respectively. c_p is a color from the paper texture. In this blending, the paper material is hardly exposed in dark regions, where the pixel colors dominate the final output.

7 Experimental Results

We implemented the proposed pencil rendering technique with OpenGL and OpenGL Shading Language. The pencil rendering examples in this paper were obtained on a Windows PC with Pentium IV 630 with 2G memory. The graphic card is GeForce 7800GTX with 256M video memory.

Figure 9(a) gives a result of contour drawing. We can see that contours are drawn multiple times with slightly different trajectories, which imitates the trial-and-errors in contour drawing by human. Figure 9(b) shows a magnified part of a pencil rendering result. The pencil strokes naturally follow the principal curvature directions and the image intensity smoothly changes with the brightness on the object surface.

In Figure 10, two interior textures generated from sample strokes with different brightness are used to render the same object. Figure 11 shows a comparison of the rendering results with and without additional contrast enhancement in the final composition stage. Figures 10 and 11 demonstrate that the style and impression of the pencil rendering results can be easily controlled in our technique.

Figures 12 and 13 show additional examples of pencil rendering produced by our technique. In these examples, the shapes of the given objects are nicely described with appropriate effects of pencil drawing.

Table 1 summarizes the rendering speed for the examples shown in this paper. Our technique provides real-time rendering for reasonably complex objects, up to meshes with several tens of thousands polygons.

Figure	Model	# polygon	Avg. FPS
Figure 9 (a)	Fandisk	12,946	45
Figure 9 (b)	Globe	9,728	47
Figure 10	Grenade	17,464	44
Figure 11	Venus	1,416	62.5
Figure 12	Drill	41,210	16

Table 1: Rendering performance table

8 Conclusion and Future Work

In this paper, we proposed a real-time technique for rendering 3D meshes in the pencil drawing style. We analyzed the characteristics of pencil drawing and incorporated them into the rendering process. The components of the process are fully supported by graphics hardware and mostly implemented by vertex and pixel shaders with multi-texturing.

In this paper, temporal coherence among animation frames of pencil rendering was not really addressed, while the accompanying video shows a reasonable amount of coherence. Although the texture mapping for interior shading is performed with 2D polygons, we do not have the "shower-door effect" when we rotate the models because the orientations for texture mapping are updated at each frame with 3D principal curvature directions. Nevertheless, a better handling of temporal coherence will reduce possible artifacts in an animation.

An important area of future work is to support more characteristics of pencil drawing, including blender and eraser effects. Other interesting problems include pencil rendering with a more abstract style, such as croquis, and color pencil rendering with the material property of a mesh.

Acknowledgments

The authors would like to thank Yunjin Lee for help in paper writing. The grenade, hydrant, and drill models are courtesy of Amazing3D, and the Venus model is courtesy of Cyberware. This work was supported in part by the Korean Ministry of Education through the BK21 program and the Korean Ministry of Information and Communication through the ITRC support program.

References

ALLIEZ, P., COHEN-STEINER, D., DEVILLERS, O., LEVY, B., AND DESBRUN, M. 2003. Anisotropic polygonal remeshing. *ACM Transactions on Graphics 22*, 3, 485–493.

BLESER, T. W., SIBERT, J. L., AND MCGEE, J. P. 1988. Charcoal sketching: Returning control to the artist. *ACM Transactions on Graphics 7*, 1, 76–81.

CURTIS, C. J., ANDERSON, S. E., SEIMS, J. E., FLEISCHER, K. W., AND SALESIN, D. H. 1997. Computer-generated watercolor. In *Proc. ACM SIGGRAPH 1997*, 421–430.

DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive contours for conveying shape. *ACM Transactions on Graphics 22*, 3, 848–855.

DEUSSEN, O., AND STROTHOTTE, T. 2000. Computer-generated pen-and-ink illustration of trees. In *Proc. ACM SIGGRAPH 2000*, 13–18.

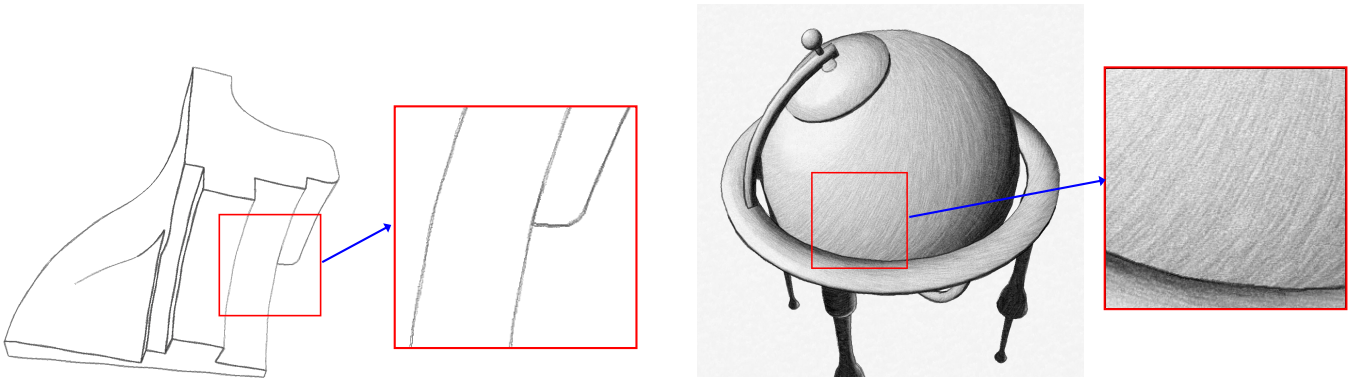
GIRSHICK, A., INTERRANTE, V., HAKER, S., AND LEMOINE, T. 2000. Line direction matters: An argument for the use of principal directions in 3D line drawings. In *Proc. NPAR 2000*, 43–52.

HERTZMANN, A., AND ZORIN, D. 2000. Illustrating smooth surfaces. In *Proc. ACM SIGGRAPH 2000*, 517–526.

HO, S. N., AND KOMIYA, R. 2004. Real time loose and sketchy rendering in hardware. In *Proc. 20th Spring Conference on Computer Graphics*, 83–88.

ISENBERG, T., FREUDENBERG, B., HALPER, N., SCHLECHTWEG, S., AND STROTHOTTE, T. 2003. A developer's guide to silhouette algorithms for polygonal models. *IEEE Computer Graphics and Applications 23*, 4, 28–37.

LAERHOVEN, T. V., AND REETH, F. V. 2005. Real-time simulation of watery paint. *Computer Animation and Virtual Worlds 16*, 3-4, 429–439.



(a) fandisk with contour drawing only

(b) globe with contour and interior shading

Figure 9: Contour and interior shading

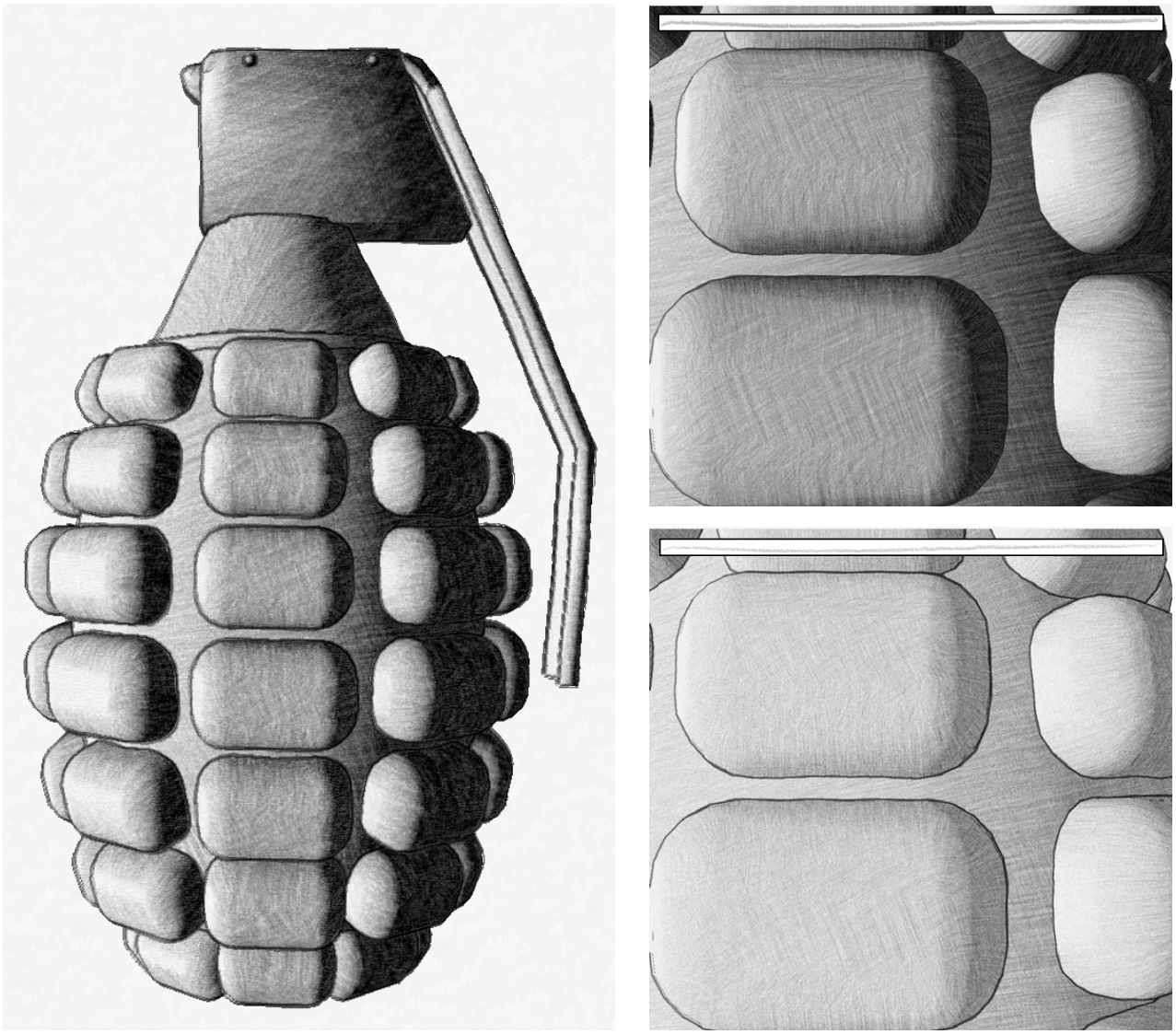


Figure 10: Grenade with two different sample strokes for texture generation: (left) a pencil rendering of Grenade; (top right) a zoom-in of the rendering on the left; (bottom right) a zoom-in of the rendering with a brighter texture

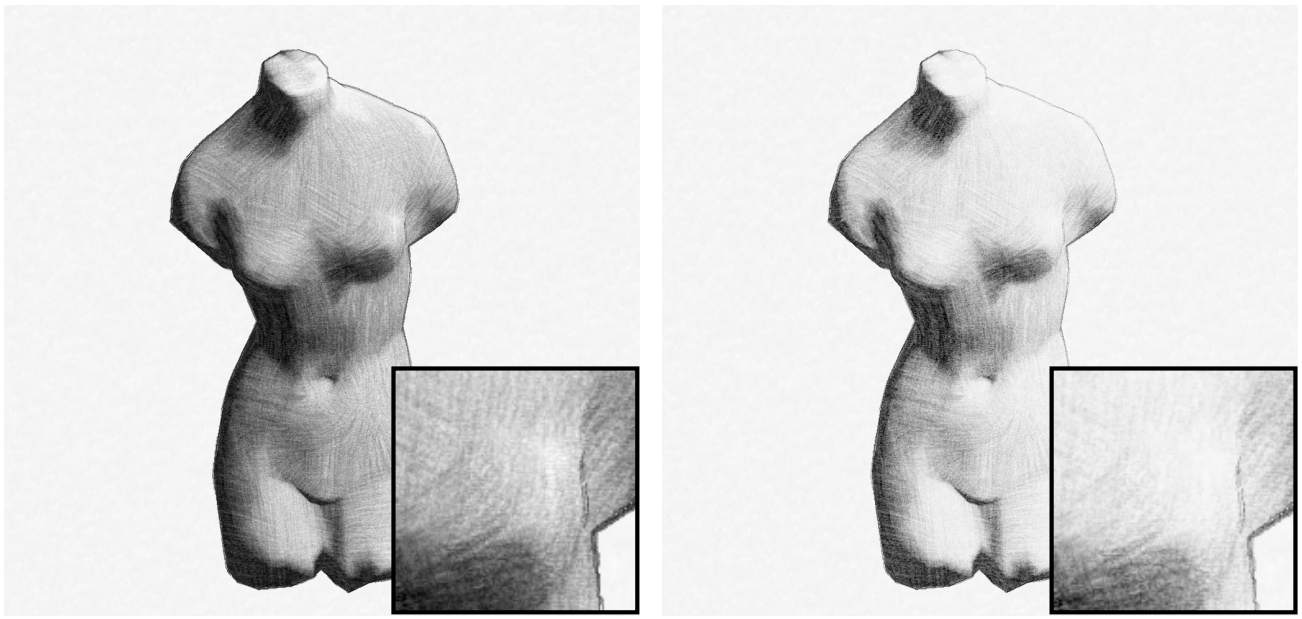


Figure 11: Venus without and with contrast enhancement: (left) a rendering result; (right) result with contrast enhancement

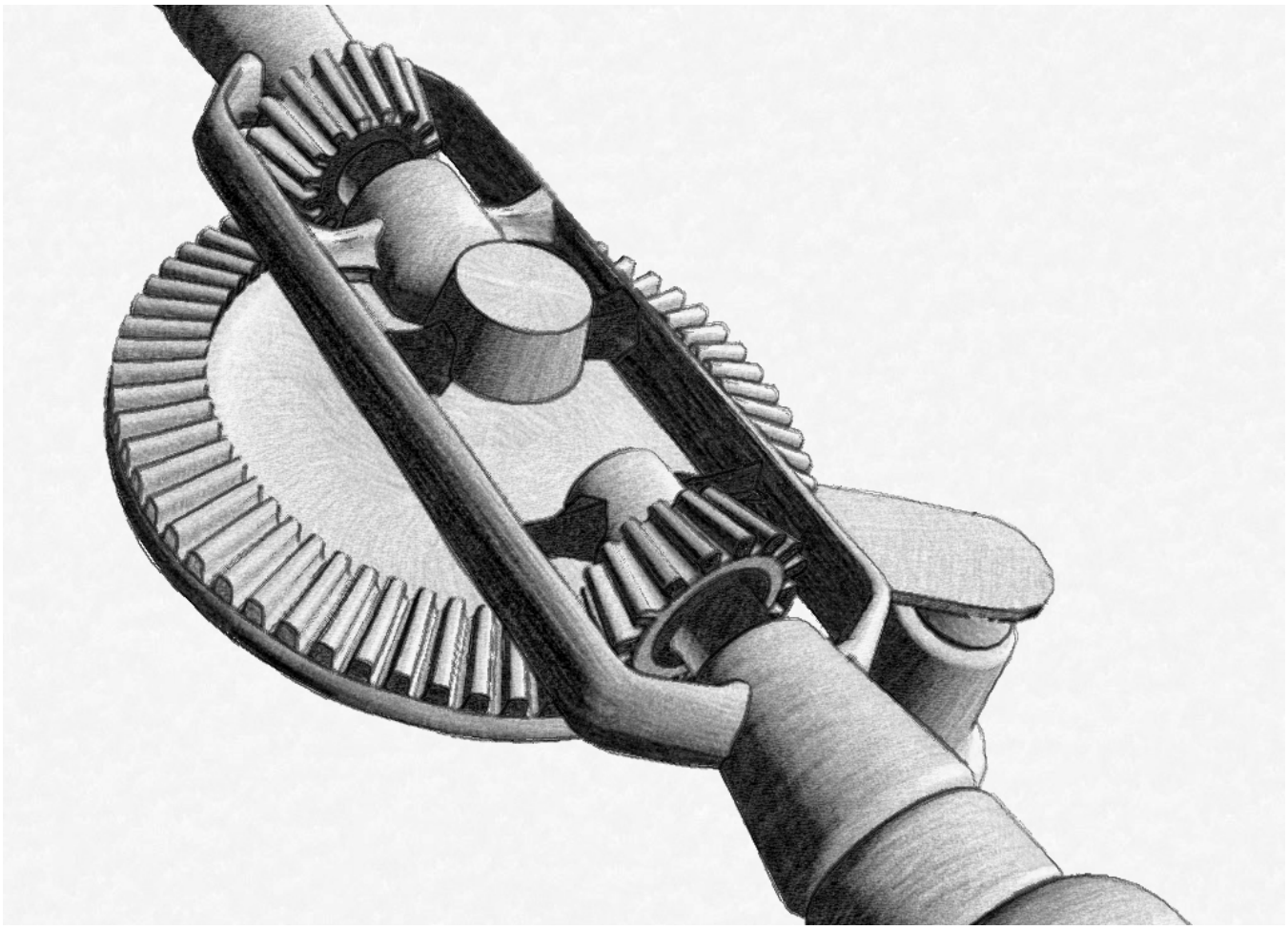


Figure 12: Drill

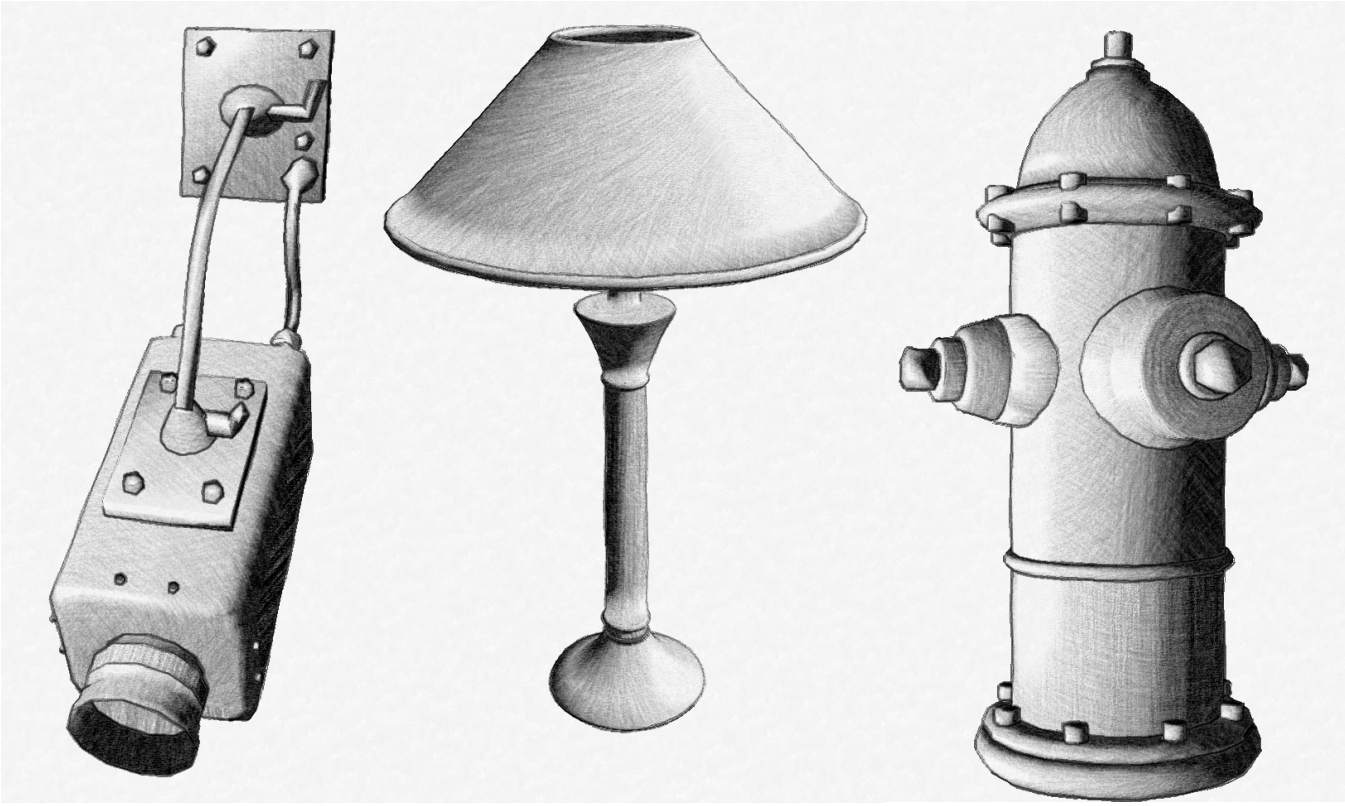


Figure 13: Pencil rendering results: (left) camera; (center) lamp; (right) hydrant

LAKE, A., MARSHALL, C., HARRIS, M., AND BLACKSTEIN, M. 2000. Stylized rendering techniques for scalable real-time 3D animation. In *Proc. NPAR 2000*, 13–20.

LOVISCACH, J. 2002. Rendering artistic line drawings using off-the-shelf 3-D software. *Computer Graphics Forum (Proc. Eurographics 2002) Short Paper*, 125–130.

LOVISCACH, J. 2004. Stylized haloed outlines on the GPU. *ACM Computer Graphics (Proc. SIGGRAPH 2004) Poster Session*.

LUFT, T., AND DEUSSEN, O. 2005. Interactive watercolor animations. In *Proc. Pacific Graphics 2005*, 7–9.

MAJUMDER, A., AND GOPI, M. 2002. Hardware accelerated real time charcoal rendering. In *Proc. NPAR 2002*, 59–66.

MAO, X., NAGASAKA, Y., AND IMAMIYA, A. 2001. Automatic generation of pencil drawing from 2D images using line integral convolution. In *Proc. 7th Int'l Conference on Computer Aided Design and Computer Graphics*, 240–248.

MOHR, A., AND GLEICHER, M. 2001. Non-invasive, interactive, stylized rendering. In *Proc. 2001 ACM Symposium on Interactive 3D Graphics*, 175–178.

NIENHAUS, M., AND DOELLNER, J. 2003. Edge-enhancement an algorithm for real-time non-photorealistic rendering. *Int'l Winter School of Computer Graphics, Journal of WSCG 11*, 2, 346–353.

PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2000. Lapped textures. In *Proc. ACM SIGGRAPH 2000*, 465–470.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. In *Proc. ACM SIGGRAPH 2001*, 579–584.

SAITO, T., AND TAKAHASHI, T. 1990. Comprehensible rendering of 3D shapes. In *Computer Graphics (Proc. ACM SIGGRAPH 1990)*, 197–206.

SALISBURY, M. P., WONG, M., HUGHES, J. F., AND SALESIN, D. H. 1997. Orientable textures for image-based pen-and-ink illustration. In *Proc. ACM SIGGRAPH 1997*, 401–406.

SOUSA, M. C., AND BUCHANAN, J. W. 1999. Computer-generated graphite pencil rendering of 3D polygonal models. *Computer Graphics Forum (Proc. Eurographics'99) 18*, 3, 195–208.

SOUSA, M. C., AND BUCHANAN, J. W. 2000. Observational models of graphite pencil materials. *Computer Graphics Forum 19*, 1, 27–49.

WEBB, M., PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2002. Fine tone control in hardware hatching. In *Proc. NPAR 2002*, 53–58.

WILSON, B., AND MA, K.-L. 2004. Rendering complexity in computer-generated pen-and-ink illustrations. In *Proc. NPAR 2004*, 129–137.

WINKENBACH, G., AND SALESIN, D. H. 1994. Computer-generated pen-and-ink illustration. In *Proc. ACM SIGGRAPH 1994*, 91–100.

WINKENBACH, G., AND SALESIN, D. H. 1996. Rendering parametric surfaces in pen and ink. In *Proc. ACM SIGGRAPH 1996*, 469–476.