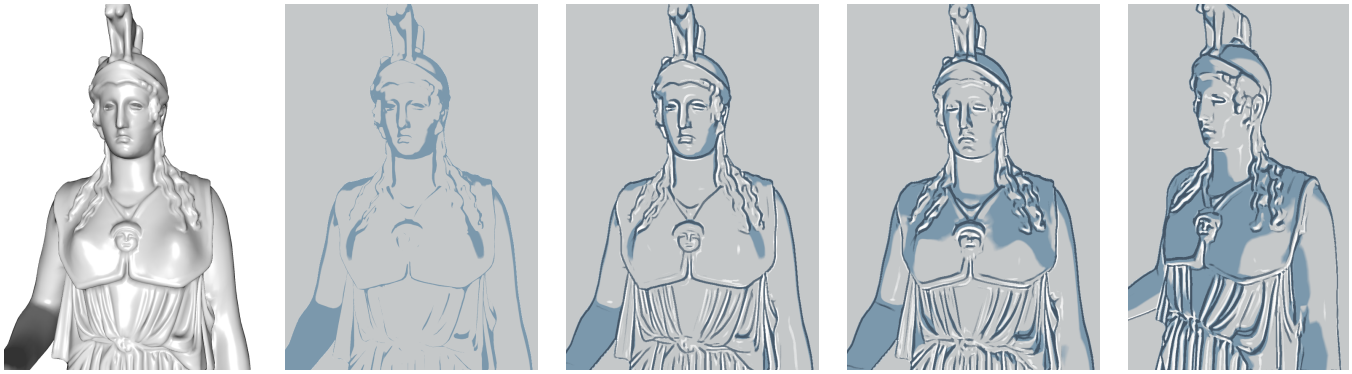
Line drawings via abstracted shading

Yunjin Lee, Lee Markosian
University of Michigan

Seungyong Lee
POSTECH

John F. Hughes
Brown University



(a) tone image (b) toon shading (c) toon shading + lines (d) new lighting (e) new lighting and view

Figure 1: (a) A tone image. (b) Toon shading captures broad tonal regions, and some thin ones. (c) Lines complement toon shading, emphasizing thin tone regions and tone boundaries. (d) Lines respond to new lighting, or (e) to new lighting and viewpoint.

Abstract

We describe a GPU-based algorithm for rendering a 3D model as a line drawing, based on the insight that a line drawing can be understood as an abstraction of a shaded image. We thus render lines along tone boundaries or thin dark areas in the shaded image. We extend this notion to the dual: we render highlight lines along thin bright areas and tone boundaries. We combine the lines with toon shading to capture broad regions of tone.

The resulting line drawings effectively convey both shape and material cues. The lines produced by the method can include silhouettes, creases, and ridges, along with a generalization of suggestive contours that responds to lighting as well as viewing changes. The method supports automatic level of abstraction, where the size of depicted shape features adjusts appropriately as the camera zooms in or out. Animated models can be rendered in real time because costly mesh curvature calculations are not needed.

1 Introduction

Depicting shape by line-drawing is clearly effective and natural, having been used for tens of thousands of years. Lines like silhouettes (where the view direction is tangent to the surface) and creases (where the surface normal changes abruptly) are common, but artists also use other lines that somehow capture more about shape, especially when rendering organic, free-form surfaces.

Recently, DeCarlo *et al.* [2003; 2004] described "suggestive contours," a new type of line that can be combined with silhouettes to produce effective line drawings of smooth shapes. They described two algorithms for rendering these. The first is an object-space algorithm that computes a subset of points on the surface where radial curvature is zero. The second is an image-space algorithm that operates on a diffuse-shaded rendering of the scene using a single point light at the camera. It outputs lines where the tone has a sufficiently sharp local minimum in one direction.

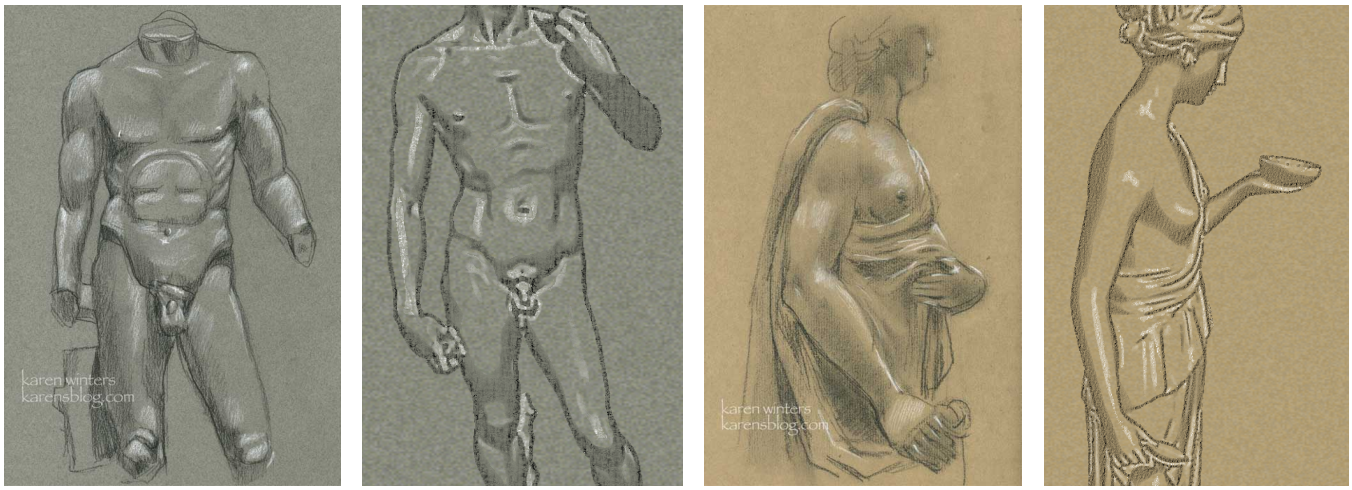
While suggestive contours convey shape well, they have some limitations. The object space algorithm does not account for how large the object appears in the image, and so may depict features that are too large or small for the current view. Suggestive contours do not depend on lighting or material properties, which is undesirable when we want the lines to reinforce existing lighting and material cues, as when some form of shading is used in combination with the lines. (See Figure 1.)

This leads us back to the question: What lines should we draw? We return to the shaded image itself as a more direct motivation for identifying important lines, and observe that a line drawing can be understood as an abstraction of the shaded image. This makes sense: if you want to convey shape effectively, but can only afford to draw a sparse set of lines, a good strategy might be to choose lines that convey the essential features of the *shading*, then rely on our visual system to interpret the shading and infer the shape.

Good candidates for "essential features" are (1) boundaries between dark and light regions, and (2) thin areas of shading that are well-approximated by lines. Indeed, the image-space suggestive contour algorithm [DeCarlo et al. 2003] detects thin dark regions in a diffuse-shaded rendering of the scene with a single point light at the camera. (In a sense, the lines "abstract" this particular shading.)

Based on these observations, we developed the following algorithm. In the first pass, we render a greyscale "tone image" describing how the scene is illuminated, then blur it and save it to texture memory. In the second pass, we use a fragment shader on the GPU to render dark lines in thin areas of dark tone.

ACM Reference Format
Lee, Y., Markosian, L., Lee, S., Hughes, J. 2007. Line Drawings Via Abstracted Shading. *ACM Trans. Graph. 26*, 3, Article 18 (July 2007), 5 pages. DOI = 10.1145/1239451.1239469 http://doi.acm.org/10.1145/1239451.1239469.



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

Figure 2: Comparison with traditional drawings. Drawings by Karen Winters: (a) and (c). Our results: (b) and (d). In both cases, dark and highlight strokes together convey a sense of shape and material properties.

A logical extension is to render light-colored "highlight lines" in thin areas of light tone, a technique used by artists (see Figure 2). This works, though we usually set a higher threshold for detecting these highlight lines, so just the strongest appear. We treat specular highlights specially. These are important features of the shading that convey material cues, but we found they are often masked by broad regions of moderately bright diffuse shading. We thus render specular highlights in a separate channel of the tone image, then extract highlight lines from *both* channels. With this change, specular highlights are captured more reliably, and help convey material shininess. The non-specular highlight lines are also useful – note the "halo" effect where the hand occludes part of the upper body in Figure 2b. (We discuss this further in Section 4.)

An added control lets the user include lines along tone boundaries (the first "essential feature"), which often occur along silhouettes and creases (including rounded ones). The last step in abstracting the shaded image is to combine dark and highlight lines with a toon shaded "base coat" to convey broad areas of dark and light.

The resulting line drawings convey shape well, are temporally coherent in animations, and can be rendered interactively on programmable GPUs. In addition, the algorithm has these benefits:

- Like other image space approaches, it selects lines at appropriate scales automatically (see Figures 6b and 8f).
- It does not require costly mesh curvature information, and so can be applied to animated models in real time.
- It depends on lighting, so lines provide lighting cues.
- It can extract highlight lines, which provide additional cues about shape and material properties.
- It supports limited fading and tapering for smoother and more temporally coherent lines.

The method has limitations as well. Depending on lighting, important shape features may be conveyed poorly. Rendered lines contain some pixel-level artifacts and may look a bit "ratty." There are limited options for stylization, since lines are extracted per-pixel. Additional processing would be needed to extract explicit lines suitable for more sophisticated stylization.

2 Related work

The first paper on suggestive contours [DeCarlo et al. 2003] contains a good review of the prior art in rendering line drawings. Particularly relevant to both suggestive contours and this work are the "formulated silhouettes" of Whelan and Visvalingam [2003], in which terrains are rendered with silhouettes plus lines that represent silhouettes from a lower viewpoint.

Saito and Takahashi [1990] used image processing operations to extract silhouettes, creases, and other lines. Creases were optionally rendered as highlights to suggest a specular quality, a technique later used by Gooch *et al.* [1999] in their work on technical illustration.

Pearson and Robinson [1985] described lines dependent on view and lighting. They seek thin dark regions in an image lit by a single source in the same horizontal plane as the view vector. We generalize to any light source, and detect other features of the tone image as well.

Ni *et al.* [2006] applied the object-space suggestive contour algorithm to a multi-scale mesh that smoothly adapts its detail to suit the viewing conditions. While effective, the multi-scale representation requires preprocessing time to build, plus significantly more memory at run-time than the original mesh.

In these proceedings, Judd *et al.* [2007] present a new object-space definition of lines, "apparent ridges," that extracts ridges using a view-dependent measure of curvature. The lines are related to the shaded image: they appear where the surface is most likely to reveal high shading contrast under arbitrary lighting. Like suggestive contours, the lines are independent of the particular lighting used at run-time. A limitation is that frame rates are relatively low due to the time spent computing view-dependent curvature.

Recently, DeCarlo and Rusinkiewicz [2007] developed an extension to suggestive contours that detects two kinds of highlight lines using an object space algorithm. The lines appear where diffuse shading would yield thin bright areas, using a single point light located at the camera. Our highlights are similar, but correspond to diffuse and specular highlights under arbitrary illumination.

3 Rendering lines

The rendering makes two passes: the first to render the tone image, which is copied from the frame buffer to a texture, and the second to detect ridges and valleys in the tone image, using a GLSL fragment shader [Rost 2006]. Its main job is to compute an opacity used to composite the line color into the frame buffer. Opacity is set to 0 when no line is near. Otherwise, it can depend on several measures of confidence. A simple GUI in our system lets the user choose line color, width, and other parameters described below.

3.1 Ridge detection

The ridge detection method we use has been widely used by others before us. Steger [1998] provides a survey of that and related methods. We explain the details here for completeness.

The lines we wish to capture correspond to ridges and valleys in the tone image, viewing it as a height field. We thus compute principal curvatures, and detect a ridge or valley when the magnitude of one curvature is sufficiently large, and the other is sufficiently small. We use a standard representation of tone: 1 is white and 0 is black. If the height field surface normals point down, then on a ridge (seen as a highlight) the curvature with larger magnitude is negative, while in a valley (seen as a dark line) it is positive.

At each pixel, the fragment shader first fits a degree-2 polynomial $f(x,y) = a_0x^2 + 2a_1xy + a_2y^2 + a_3x + a_4y + a_5$ to the tone values near the pixel, then computes curvature analytically. We use a local parameterization so that $(0,0)$ corresponds to the pixel center, and compute the coefficients of f via least squares as follows.

Given n sample locations $(x_i,\ y_i)$ in a neighborhood of the pixel, with corresponding tone values t_i taken from the image, we construct the $n \times 6$ matrix X made up of rows: $(x_i^2\ 2x_iy_i\ y_i^2\ x_i\ y_i\ 1)$. The n samples determine the set of equations: $XA = T$, where A is the 6×1 column vector of unknown coefficients a_i, and T is the $n \times 1$ vector of tone values t_i. We then solve for A via least squares: $A = (X^TX)^{-1}X^TT$. Because we use a local parameterization, the matrix $H = (X^TX)^{-1}X^T$ is constant and can be computed off-line and hard-coded into the program. To solve for A at run time, we simply multiply the $6 \times n$ matrix H with the $n \times 1$ tone vector T. Denoting the point $(x,\ y)$ by $\mathbf{x}$, we can express $f(\mathbf{x})$ as the quadratic form $Q(\mathbf{x}) = (\mathbf{x}-\mathbf{c})^TM(\mathbf{x}-\mathbf{c})$ plus a constant term, where

$$M = \begin{pmatrix} a_0 & a_1 \\ a_1 & a_2 \end{pmatrix}, \text{ and } \mathbf{c} = -\frac{1}{2}M^{-1}(a_3\ a_4)^T.$$

The eigenvalues and eigenvectors of M are the principal curvatures and directions of f at $\mathbf{c}$. The ridge or valley is the line through $\mathbf{c}$ in the low-curvature direction. If that curvature is 0, M is singular and we can't compute $\mathbf{c}$, but the line is still defined. It lies at distance $\frac{-(a_3\ a_4)\cdot\mathbf{u}}{2\lambda}$ from the pixel in direction $\mathbf{u}$, where λ is the nonzero eigenvalue and $\mathbf{u}$ is the corresponding unit-length eigenvector.

3.2 Ridge searching

In practice we use 9 sample points arranged in a 3×3 grid around the pixel location, with spacing set to half the desired line width w. To reduce sampling artifacts, the preparation of the tone image includes a blurring step using a Gaussian kernel of size w.

Based on the distance to the ridge or valley line and its first principal curvature, the original pixel belongs to one of four cases, as shown in Figure 3: (a) a pixel *on* a ridge or valley, (b) a pixel *near* a ridge or valley, (c) a pixel in a smooth region, and (d) a pixel on an edge. In case (c) we simply set opacity to 0. For the others, we explain only the case of a ridge, since the valley case is symmetric.

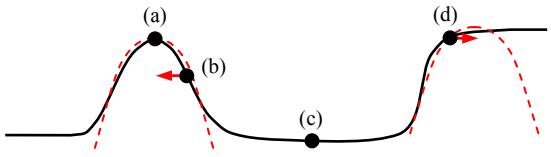
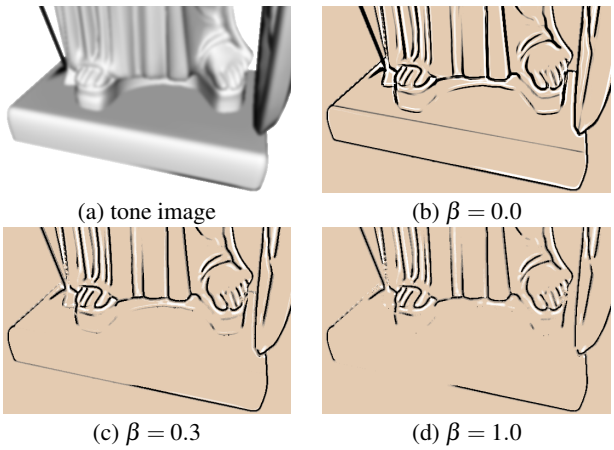


Figure 3: Ridge searching. Red dotted curves show the fitting polynomials. By refitting the polynomials after moving toward the ridge or valley, we can distinguish case (b) from case (d).

In both cases (b) and (d) in Figure 3, the polynomial f (measured at the pixel and shown as red dotted curves in the figure) shows a nearby ridge. We use an iterative search method to distinguish the two cases. We move toward the detected ridge line, then solve for f at the new location and measure curvature and distance to the new ridge. We repeat this process several times (5 in our implementation) and set opacity to 0 if the computed curvature falls below a threshold, or the total distance travelled exceeds half the line width. Otherwise we set opacity to 1 (see Figure 5b).

During this iterative search, if at each step we move the full distance to the ridge, tone edges (case (d)) will not be drawn. If we do not move at all, tone edges will be drawn along with ridges and valleys. (Note that tone edges very often include silhouettes and sharp or rounded creases – see Section 4.) A compromise – moving by a fraction β of the full distance – keeps only strong edges. We give the user control of this fraction $\beta \in [0,1]$ (see Figure 4). For the examples in this paper we usually used values between 0.1 and 0.3.



(a) tone image (b) $\beta = 0.0$

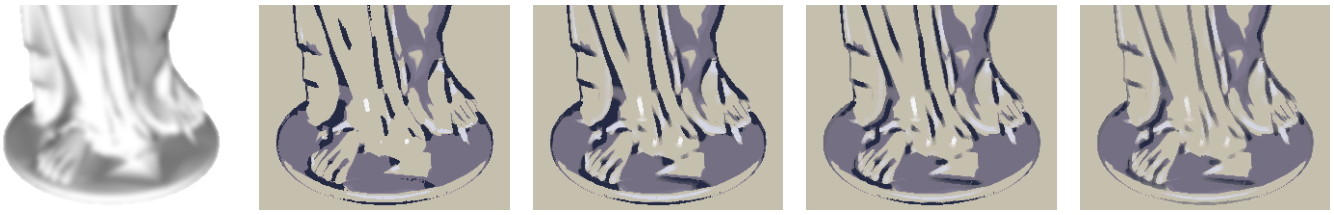
(c) $\beta = 0.3$ (d) $\beta = 1.0$

Figure 4: For small β, we detect tone edges along with ridges. For large β, we detect only ridges. In between, we detect strong edges.

3.3 Control of line opacity

During ridge searching, we accept or reject a pixel by setting the opacity to 1 or 0. To produce smoother lines, and prevent them from appearing abruptly when the viewpoint or lighting changes, we can replace the single threshold with lower and upper thresholds, c_l and c_u. We stop early if $c < c_l$, where c is the magnitude of the larger principal curvature. After the iteration, we set opacity to 0 if $c < c_l$, and to 1 if $c > c_u$. Otherwise, we use $(c - c_l)/(c_u - c_l)$. Figure 5c shows lines drawn with controlled opacity.

We can also vary line opacity with distance to the ridge line. That is, we let opacity drop off as distance to the ridge approaches half the line width. This is shown in Figure 5d.



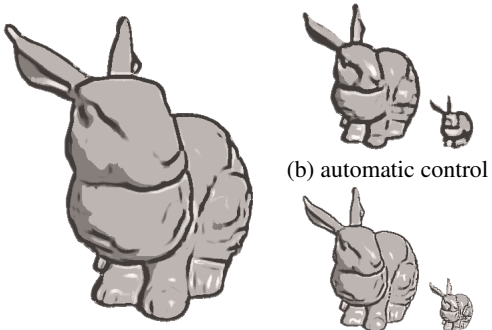
(a) (b) (c) (d) (e)

Figure 5: The tone image (a) is used to render lines with a hard threshold based on curvature (b). A soft threshold yields smoother lines (c), as does fading the line near its edges (d). Counterchange, shown in (e), reduces opacity more or less, depending on the underlying tone.

To enhance dark lines, we draw them lighter in light areas, so they appear comparably darker in dark areas, an effect known as "counterchange" [Ruskin 1971]. To achieve this (and a similar effect for highlight lines), we multiply the opacity by $1-t$ for a dark line and by t for a highlight line, where t is the pixel's tone value. Figure 5e shows this effect. In practice, we usually applied the three effects together (smooth curvature threshold, distance to line, and counterchange), by using the product of the three opacities.

3.4 Level of abstraction

Like other image space approaches, our method selects lines at appropriate scales automatically (see Figure 6b). To provide a depth cue, we can reduce line width for distant surfaces as in Figure 6c, where we also used "abstracted normals" [Barla et al. 2006] to render the tone image and toon base coat. Figure 7 shows line width adjusted by average edge length at each vertex. Thinner lines are used in finely tessellated regions.



(a) initial view (b) automatic control (c) control by depth

Figure 6: Varying levels of abstraction. (a) Initial view. (b) For a given line width, a greater level of abstraction results automatically when zoomed out, since coarser (more "essential") shape features are conveyed. In (c) we reduce line width as depth increases, and use abstracted normals in the tone image and toon base coat.

4 Results and discussion

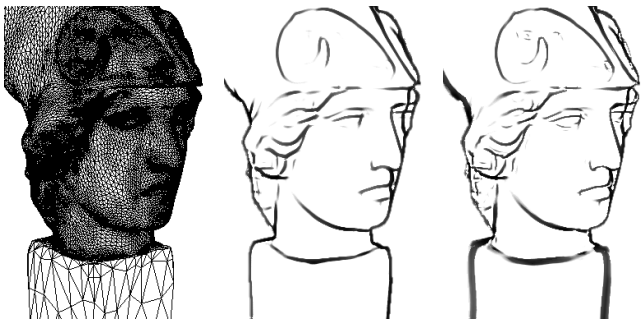
The images in this paper and accompanying video show that our method can produce effective line drawings of complex shapes in a range of styles (see Figure 8) based on relatively simple but effective controls. The lines generalize suggestive contours, and provide cues about shape, lighting, and materials. They work well when combined with toon shading, which often anticipates and extends the lines, and both evolve smoothly and naturally together as the shape, lighting, or camera changes.

The method reliably finds silhouettes and creases (even rounded ones) when a low value for β is used (see Section 3.2), because usually there is a strong change of tone across the silhouette or crease (the tone image background is white). This also explains the "halo" effect mentioned previously and observable in Figure 2b and images in the top row of Figure 8 – highlight lines appear over partially occluded surfaces near the occluding silhouette, providing natural contrast enhancement along those important lines.

The method can render complex models at real-time rates (including animated models, as shown in the video). We tested our shader with an NVIDIA GeForce 7900 GT GPU. With the model filling a quarter to a half of an 800×600 window, frame rates ranged from 20 to 7 fps for models with 60K to 200K triangles, respectively. Much of the time is taken by the preparation of the tone image, especially the blurring step.

Our algorithm does not extract lines explicitly in vector form, so options for stylization are limited. We can incorporate a degree of media simulation using the "paper effect" of Kalnins *et al.* [2002], applied to lines and shading as shown in Figures 2b, 2d, 8c, and 8d.

Dependence on lighting can be a limitation, since important features will not be depicted with some lighting setups. On the other hand, scene designers can customize the lighting to convey desired features clearly. Extracting lines from the tone image adds flexibility and opens the possibility of achieving better line drawings through global illumination and other new ways to compute tone.



(a) (b) (c)

Figure 7: Level of abstraction based on edge length. (a) input mesh. (b) Constant line width. (c) Line width varies with edge length.

Acknowledgments

We thank Karen Winters (karenwinters.com) for permission to use her drawings in Figure 2. This research was supported in part by the ITRC support program, the Korea Research Foundation Grant funded by the Korean Government (MOEHRD) (KRF-2006-214-D00138) in Korea, and the NSF (CCF-0447883).

(a) Hebe, David, and Athena (b) Dama (c) Horse

(d) Bunny (e) Hippo (f) Zoom-in/out of landscape

Figure 8: Line drawings rendered in various styles, depending on line color, width, and other controls provided in our system.

References

BARLA, P., THOLLOT, J., AND MARKOSIAN, L. 2006. X-toon: an extended toon shader. In *Proceedings of NPAR 2006*, 127–132.

DECARLO, D., AND RUSINKIEWICZ, S. 2007. Highlight lines for conveying shape. *Working manuscript*.

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

DECARLO, D., FINKELSTEIN, A., AND RUSINKIEWICZ, S. 2004. Interactive rendering of suggestive contours with temporal coherence. In *Proceedings of NPAR 2004*, 15–24.

GOOCH, B., SLOAN, P.-P. J., GOOCH, A., SHIRLEY, P., AND RIESENFELD, R. 1999. Interactive technical illustration. *1999 ACM Symposium on Interactive 3D Graphics*, 31–38.

JUDD, T., DURAND, F., AND ADELSON, E. H. 2007. Apparent ridges for line drawing. *ACM Transactions on Graphics 26*, 3.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. WYSIWYG NPR: Drawing strokes directly on 3D models. *ACM Transactions on Graphics 21*, 3, 755–762.

NI, A., JEONG, K., LEE, S., AND MARKOSIAN, L. 2006. Multi-scale line drawings from 3D meshes. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, 133–137.

PEARSON, D., AND ROBINSON, J. 1985. Visual communication at very low data rates. *Proc. the IEEE 73*, 4, 795–812.

ROST, R. J. 2006. *OpenGL Shading Language, Second Edition*. Addison Wesley Professional.

RUSKIN, J. 1971. *The Elements of Drawing*. Dover Publications.

SAITO, T., AND TAKAHASHI, T. 1990. Comprehensible rendering of 3D shapes. *Proceedings of SIGGRAPH 90*, 197–206.

STEGER, C. 1998. An unbiased detector of curvilinear structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20*, 2, 113–125.

WHELAN, J., AND VISVALINGAM, M. 2003. Formulated silhouettes for sketching terrain. In *Proc. Theory and Practice of Computer Graphics*, 90–96.