An Approach to Visualizing Transparency in Computer-Generated Line Drawings

Jörg Hamel, Stefan Schlechtweg, Thomas Strothotte
Department of Simulation and Graphics
Otto-von-Guericke University of Magdeburg
Magdeburg, Germany
hamel, stefans, tstr@isg.cs.uni-magdeburg.de

Abstract

This paper builds on principles of depicting transparency in hand-made line drawings, and develops a method to generate similar, but computer-generated, line drawings in a two-step process. In the first step a brightness image is generated which is used as a basis for the line drawing created in the second step. Three different methods derived from traditional drawing techniques are shown.

Keywords: computer-generated line drawings, transparency

1 Introduction

As opposed to photographs and photorealistically rendered pictures, illustrations have the advantage that they can be more selective about their content and detail. Important details can be emphasized while extraneous can be omitted (see Dooley, Cohen [2]). Shapes can be simplified and hidden structures revealed. It has been shown that line drawings can be used to adjust finely the message conveyed by an image [14, 11, 15]. On the other hand, illustrations included in text books often use no colors, or at least very few colors to keep down the cost of the print. Therefore, line drawings have held their place in scientific illustration, despite of the advances of photography and computer graphics in the last century.

The generation of illustrations on the computer has attracted attention in the research community in recent years. In particular, *Saito* and *Takahashi* [6] proposed a method for enhancing rendered images by computing hatching lines based on G-buffers containing information of u-v coordinates of the parametric surface and lines along edges. Computer generated copper plates were the result of a raytracing system by *Leister* [4]. These copper plates can incorporate specular reflection and transparency. However, no special attention was payed to transparency, it was merely something inherited from the raytracing basis of the system. *Winkenbach* and *Salesin* [7] presented a rendering system for pen-and-ink illustrations based on parametric surfaces. Classical rendering techniques are applied to the lines, so shadows, texture, environment and bump mapping result in richer images. *Schofield* developed *Piranesi*, a rendering system for 'non-photorealistic' images, which handles like a paint program [10]. It works on a rendered picture augmented by G-Buffers containing material and depth information. The sketch-renderer described by *Strothotte* et al. [13] provided interaction facilities to design lines and hatchings in a flexible way.

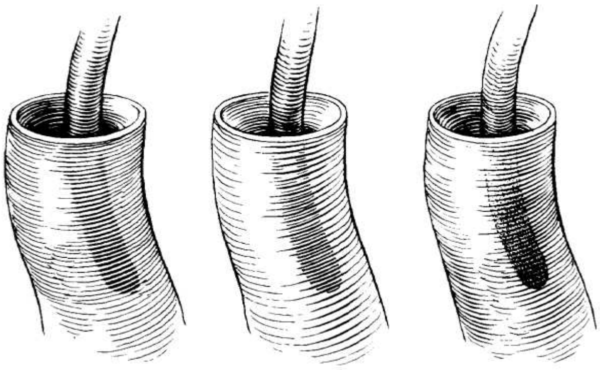
This paper treats a specific problem in computer-generated line drawings: how to visualize transparency. While transparency is well-established in illustrations based on photorealistic rendering (see e.g. Seligman, Feiner [12], who describe the Intent-Based-Illustration System IBIS) it has hardly been used in computer-generated line drawings so far—the exception being the raytracer by *Leister*. Probably, the rare visualization of transparent objects in manual line-drawings is responsible for this. Transparency, however, is essential to communicate the location of occluded objects.

The paper is organized as follows. The section right after this introduction presents the reference line drawings and discusses the techniques which were used to create them. In the third section, two methods for rendering special pixel images are developed. These images are used in Section 4 to render the actual lines. Section 5 describes the rendering system that was used as the basis for this article. The last section concludes the article and gives a preview to future work.

2 Drawing Techniques

This paper concentrates on line drawing techniques depicting transparency as presented in "The Guild Handbook of Scientific Illustration" [3]. In this book, techniques and methods are discussed which illustrators use to create their drawings. Characterizing for the techniques dealing with transparency is that they do not use colors. For colored illustrations, Hodges just recommends to lighten the color of the occluded object.

Examples which inspired our work are shown in Figure 1. The pictures show a stick-like object pointing downward into a semi-transparent tube.



(a) Increasing the thickness of lines

(b) Inserting additional lines

(c) Using a different style, in this case stippling

Figure 1: Three different ways for expressing transparency in a line drawing, drawn by hand by a scientific illustrator (taken from [3, page 101]).

The illustrator used three different ways to show the 'shining through' stick on the surface of the tube. In the first picture, he drew extra thick lines. Notice that the lines at least on the front face of the tube are nowhere as thick as on the 'stick-part'. In the second picture, the stick is shown by additional lines interlacing with lines shading the surface of the tube. Comparing with the other two pictures it is obvious that the overall spacing of the lines is greater here. In the third picture, a different style altogether is used. Here, the part shining through is depicted with another shading technique: stippling. Characterizing for all three pictures is that the way the lower part of the 'stick' is drawn is independent from the upper part and that the lower part fades out the closer it comes to the opening of the 'tube'. The three drawings are appealing and intuitively understandable. For someone who wants to render similar pictures on a computer, however, it would be interesting to know if they are based on a common physical model, as that would keep things simple.

3 Rendering Techniques

To investigate this question—is a physical model used to generate these pictures?—the two objects were modeled using a common 3D-modeler and rendered with a simple, OpenGL-based renderer. The result can be seen in Figure 2(a).

There is more information in this picture than we need. For example we do not need to see the backside of the tube and certainly not the thickness of the wall. Therefore, a simpler blending is applied as a postprocessing step as can be seen in Figure 2(b).

But still this picture is not suitable as basis for line drawings as the above examples. The lower part of the stick is visible for all its length in this picture, but not in the line drawings. Why is this so? Certainly, our picture is more realistic—even though we simplified the transparency calculation and used a quite simple lighting model—the stick does shine through all the way.

The answer is that line drawings seldomly follow physical models but adhere to certain empirical rules. Looking at the reference pictures, you can see for instance that the light source should be somewhere front-left taking into account the dark lines in the inner side of the tube, however, this does not explain why there is so much dark area left on the outside. This can only be explained by the assumption that the illustrator wanted to draw lines on the left as well so that the tube does not lose its spatiality. He also

(a) Transparency from OpenGL

(b) Transparency by postprocessing OpenGL-output

Figure 2: Rendering the scene with OpenGL.

wanted to keep a certain minimal line length. For the transparency of the stick, we do not have to use assumptions as *Hodges* provides us with the rule:

- strengthen the shade where an object enters another object;
- set its intensity to zero at the edge and slowly increase the intensity again.

This is clearly a rather arbitrary way, and should therefore be modeled similarly. The first decision we have to make is, do we want to work in 2D—the image—or in 3D—the scene. Each way has positive and negative aspects, therefore, we will look at both in turn.

Working just on the brightness image is the most straightforward way: an edge detection in the objectID-buffer (another output of the OpenGLRenderer, see Section 5) between the outer and the inner part followed by a vectorization gives a number of lines. These can be used for all pixels of the occluded object—in this case the stick—to determine the smallest distance d_s. This distance is set in relation to the parameters d_{out} or d_{in}—depending on whether the pixel is in- or outside—which define the length of the respective fading areas.

Equation 1 shows the simple blending (a linear interpolation) of the brightness values from the picture without (b_i) and with (b_i^T) transparency applied to the pixels on the inside. In Equation 2, a linear scaling operation is sufficient for the outside part.

$$b_i' = \begin{cases} b_i \cdot \left(1 - \frac{d_s}{d_{in}}\right) + b_i^T \cdot \frac{d_s}{d_{in}} & \text{for } d_s < d_{in} \\ b_i^T & \text{otherwise} \end{cases} \quad (1)$$

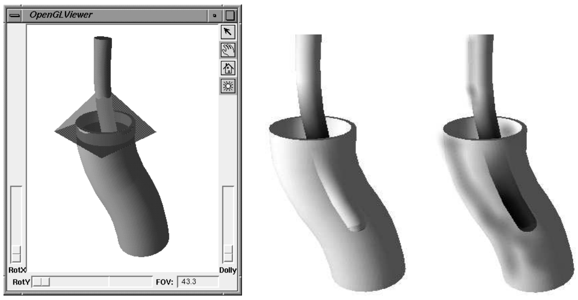
$$b_i' = \begin{cases} b_i \cdot \frac{d_s}{d_{out}} & \text{for } d_s < d_{out} \\ b_i & \text{otherwise} \end{cases} \quad (2)$$

A look on the result in Figure 3 shows that the desired effect was achieved, however, the shape of the fading follows the edge between the inner and outer part. Thus in an example where this edge is formed differently, that is, where the curvature of the edge is more different to the curvature of the object, the result will look less satisfying.
The logical consequence from the latter thoughts is to define the fading analogously to the above 2D solution in 3D, thus implicitly taking perspective and curvature into account. For each area, a start point, a normal vector indicating the direction, and a length has to be defined.

These parameters can be set using the OpenGLView with the help of a pyramid-formed widget (cf. Figure 4(a)). This works quite well as can be seen in Figure 4(b).

Figure 3: Fading the transparent areas in 2D.

Which of the two methods is to be preferred depends on the scene, certainly, in the scene used here, the two-dimensional method produced about the same result as the three-dimensional and since it requires less user interaction it might be preferred in this case.

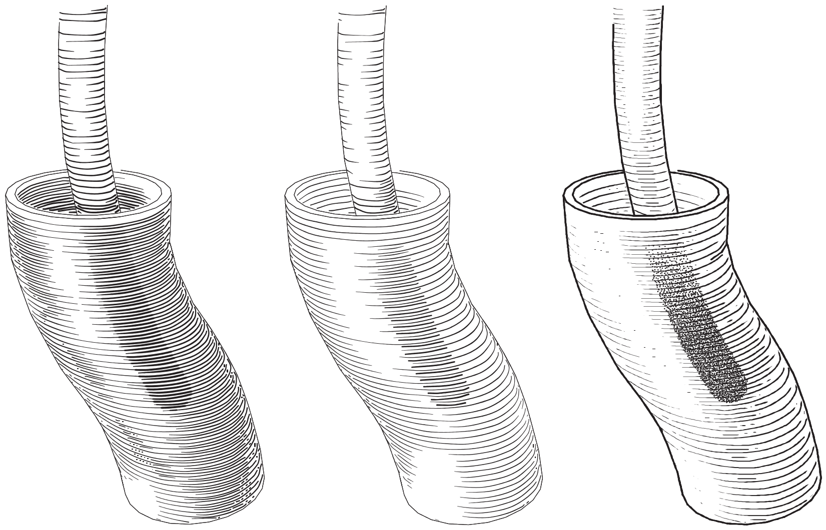


(a) (b) (c)

Figure 4: The definition and look of the 3D fading, the final image

For the final image, b_i^T in Equation 1 is set to zero so that the stick is just black and not showing any shading. Some arbitrary added shading supports the plasticity, and lessens the cold touch of the gouraud shading.

The resulting image shown in Figure 4(c) approximates the brightness distribution of our reference pictures quite well and gives us a working base for the creation of a line drawing.



(a) Thicker lines (b) More lines (c) Different style

Figure 5: Rendering transparency in different ways.

4 Drawing Lines

To generate a line drawing that includes transparency in the way proposed by *Hodges*, we need three different images:

- the 'opaque' image, where all objects are rendered opaque and the outer parts brightness is faded to zero towards the edge between inner and outer part;
- the 'transparent' image, as described in the previous section;
- a mask, which masks out all pixels not concealed in the opaque image (in this case—the lower part of the stick).

The methods that use these images are discussed in the following paragraphs.

Thick Lines

In the first reference image, the stick is shown by increasing the thickness of the lines. To achieve this effect, we intersect our lines with the edge of the mask and draw the part inside with a thickness based on the brightness of the transparent picture. The thickness of the line parts outside the masked area and all other lines is based on the opaque image.

More Lines

The second reference image shows the occluded object by interlacing the lines rendering the transparent object with additional lines. This time we draw only every second line on the tube, based on the opaque picture. The other lines we intersect with the mask edge and draw them based on the transparent image.

Different Style

The third reference image uses a different style altogether, here it is stippling. This case is quite similar to the last one. All we have to do is drawing the lines based on the opaque image, then add the lines—or dots, whatever our different style uses as primitives—drawn based on the transparent image and masked out.

Figure 5 shows the images rendered with these methods. The reader is encouraged to compare them with the reference pictures. The three different techniques—thicker, more lines, and different style—do deliver distinguishably different results.

5 A Rendering System

To understand why the process bringing transparency into line drawings developed in this article is actually two-stepped, it is useful to have a look at the rendering system our work is based on.

5.1 Architecture

The interface of the rendering system consists of four basic elements as shown in Figure 6:

- In the OpenGL-View the scene as rendered with OpenGL is shown. Since OpenGL is hardware-supported on a variety of platforms, rendering is quite fast, enabling interactive rotation and zooming. Attributes of objects that have a three-dimensional representation can be rotated, scaled and translated here as well.
- The Hierarchy-View is a view onto the structure of the scene and provides access to attributes of the objects.
- The Render-View is the front-end for the actual Hybrid Renderer. Results of the rendering process are shown here. A result, that is, a set of lines can be selected and removed from the image or dialogs for rendering and line style can be opened.

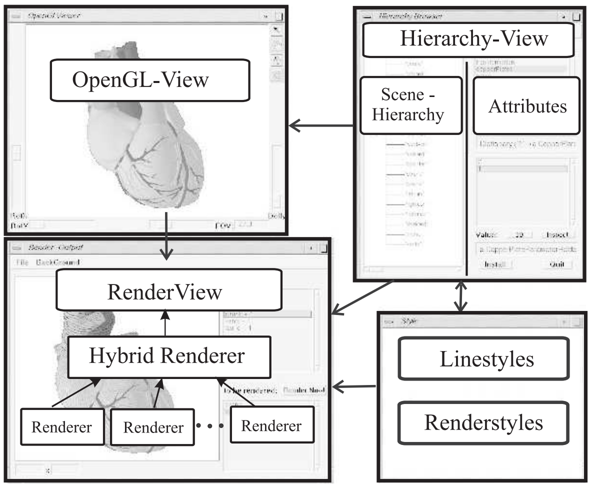


Figure 6: The Rendering System

5.2 Functionality

There are currently five different renderers providing input to the hybrid renderer:

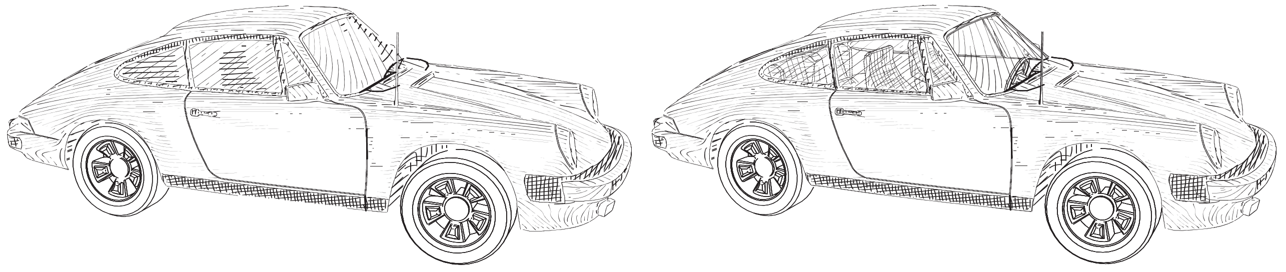
- An OpenGL-Renderer.
- A pixel-oriented renderer developed by *Deussen* [1], which intersects objects explicitly with planes of arbitrary orientation and performs edge detections on the resulting images. The results are copperplate-like lines, which can be used for hatching.
- A pixel-oriented renderer based on the one by *Deussen*, speed optimized for parallel planes only. Using the z-Buffer—an output from the OpenGL-Renderer—the space between the planes is filled with a color encoding the number of the next plane. Because perspective and curvature can get planes in the picture closer together or farther apart as intended, more than one of these pictures is necessary, four usually suffice though. Thus, edge detection has to be performed on only four pictures. The results are then interlaced.
- An analytic renderer working on polygonal meshes by *Raab* [5]. Edges between polygons with angles above a certain threshold are collected and returned. This results in lines describing the silhouette of the object.
- A simple stipple renderer using dithering of a grayscale pixel image.

The hybrid renderer draws the lines that are the output of the three line-renderers using the *linestyles* developed by *Schumann* and *Schlechtweg* [9]. These linestyles allow the encoding of surface related properties (as for example the brightness) in a line's attributes. So lines can change their thickness according to the brightness distribution on an object. The output of the OpenGL-Renderer can be used as a source for the values to be encoded. Other or additional sources are quite useful and yield a rich set of parameters to vary the appearance of the created line drawings, a few examples were shown in Figure 5, for further reference, see [9].

6 Conclusions and Future Work

This article has shown methods to render line drawings containing transparency. The rendering process is divided into two steps: the creation of a brightness image with transparency; and the actual drawing of the lines. For each step, different methods were presented. The efforts were guided by analysis of the reference pictures and rules given by *Hodges* in [3].

An aspect not covered here so far is how to visualize different levels of transparency. Figure 7 demonstrates two levels of transparency. The windows in Figure 7(a) are not very transparent, the seats inside are only visible as shadows—like through a shower curtain. Notice that this is a logical result of the rendering process, when hatching is based on the lines of the transparent object and the transparency just results in a darkening of the transparent surface. Figure 7(a) was rendered with the methods presented in this paper, a 3D-fading in view direction delivers the inten-



(a) Almost opaque windows

(b) Very transparent windows

Figure 7: Two rendered line drawings of a car with different levels of transparency.

sity of the 'shadows'. In Figure 7(b), in contrast, the seats are clearly visible, with contours and perspectively correct hatching drawn on the seats themselves. For this picture, the window was rendered separately and overlayed later, so that there is no transparency used except for this separation.

These two pictures can be envisaged to stand at opposite ends of the spectrum of transparency values. To investigate in between is one area of future work, another is the addition of colors. Last not least the resulting new ways for expressing transparency will have to be systemised and integrated into the rendering system and an appropriate user interface.

References

[1] O. Deussen, "Pixel-Oriented Rendering of Line Drawings" in: [15].

[2] D. Dooley and M. F. Cohen, "Automatic Illustration of 3D Geometric Models: Lines", *Computer Graphics 24*, Vol. 2, pp. 77–82, 1990.

[3] E. R. S. Hodges (Ed.), *The Guild Handbook of Scientific Illustration*, Van Nostrand Reinhold, New York, 1989.

[4] W. Leister, "Computer Generated Copper Plates", *Computer Graphics Forum*, Vol. 13, no. 1, pp. 69–77, 1994.

[5] A. Raab, *Techniken zur Exploration und Visualisierung geometrischer Modelle*, PhD Thesis at the Otto-von-Guericke University Magdeburg, Germany, 1998 (to appear).

[6] T. Saito and T. Takahashi, "Comprehensible Rendering of 3D Shapes", *Computer Graphics* (Proc. Siggraph), Vol. 24, No. 4, pp. 197–206, 1990.

[7] G. Winkenbach and D. H. Salesin, "Rendering Parametric Surfaces in Pen and Ink", *Computer Graphics* (Proc. Siggraph), Vol. 30, pp. 469–476, 1996.

[8] G. Winkenbach and D.H. Salesin, "Computer-Generated Pen-and-Ink Illustration", *Computer Graphics* (Proc. Siggraph), Vol. 28, No. 4, pp. 91–108, 1994.

[9] S. Schlechtweg, B. Schönwälder, L. Schumann, T. Strothotte, "Surfaces to Lines: Rendering Rich Line Drawings", *Conference Proceedings WSCG'98*, Vol. 2, pp. 354–361, 1998.

[10] J. Lansdown and S. Schofield, "Expressive Rendering: A Review of Nonphotorealistic Techniques" , *IEEE Computer Graphics and Applications*, pp. 29–37, May 1995.

[11] J. Schumann, T. Strothotte, A. Raab, S. Laser, "Assessing the Effect of Non-Photorealistic Rendered Images in CAD", *Proceedings of CHI '96*, pp. 35–42, 1996.

[12] D. D. Seligmann and S. K. Feiner, "Automated generation of intent-based 3D-illustrations", *Computer Graphics* (Proc. Siggraph), Vol. 25, pp. 123-132, 1991.

[13] T. Strothotte, B. Preim, A. Raab, J. Schumann, D. R. Forsey, "How to Render Frames and Influence People," *Computer Graphics Forum*, Proc. of Eurographics, Vol. 13, pp. 455–466, 1994.

[14] C. Strothotte, T. Strothotte, *Seeing Between the Pixels: Pictures in Interactive Systems,* Springer-Verlag, Berlin, 1997.

[15] T. Strothotte (ed.), *Abstraction in Interactive Computer Visualization: Exploring Complex Information Spaces,* Springer-Verlag, Berlin-Heidelberg-New York, 1998 (in print).