First publ. as: Paper / Graphics Interface Conference, Kingston, Ontario, June 1999, pp. 175-182

An illustration technique using hardware-based intersections and skeletons

Oliver Deussen[*] Jörg Hamel Andreas Raab Stefan Schlechtweg Thomas Strothotte

Department of Simulation and Graphics, Otto-von-Guericke University of Magdeburg

Abstract

We present a method for generating line drawings of complex geometries in the style of crosshatched illustrations. Hatching lines are generated by intersecting the geometry with a set of planes. Half-toning on the basis of the generated curves is used to represent a given intensity distribution. Computing a geometric skeleton allows us to determine automatically the orientation of the intersection planes for a wide variety of models. By using predefined line styles different types of illustrations can be generated. Applications of the method are discussed, examples are given.

Key words: Non-Photorealistic Rendering, Illustration Techniques, Arts, Graphics Hardware

1 INTRODUCTION

Traditional printing techniques have evolved in our century in a way that realistic images like photographs can be included in every book at affordable costs. But, a look at scientific, technical, or medical documents shows that many of the images are in fact not photographs but illustrations, sketches or other line-oriented drawings [26]. The reason for not using photographs here is that complex information can be conveyed better by using abstract drawings instead of realistic images [23, 27].

The need for generating such drawings has motivated a number of authors to work on non-photorealistic rendering techniques, while most of the work in computer graphics done so far has been dealing with the generation of photorealistic images.

Early work in non-photorealism focussed on how to draw hidden lines in order to enhance the comprehensibility of line drawings [1, 11]. Drawing with different lines styles was introduced by Dooley and Cohen [3], while Strassmann [25] proposed a method for simulating hairy brushes in order to obtain watercolor-like drawings. Guo and Kunii added a method for simulating ink-diffusion on paper [8]. Hsu and Lee [10] extended the set of possible line styles by generating strokes that use textures from arbitrary pictures.

Schofield [13] recognized that the generation of line drawings and other non-photorealistic drawings can be made easier by using 2-D images and additional information about the underlying 3-D model. This idea was introduced earlier but independently by Saito and Takahashi [18] for generating expressive images of landscapes and medical data.

A method working with stroke textures was introduced by Salisbury, Winkenbach and Salesin [20, 29, 19] to create hatching lines that represent texture and tone of the model. They extended their work to resolution-dependent hatchings and developed methods for interactive and automatic definition of stroke directions within the textures.

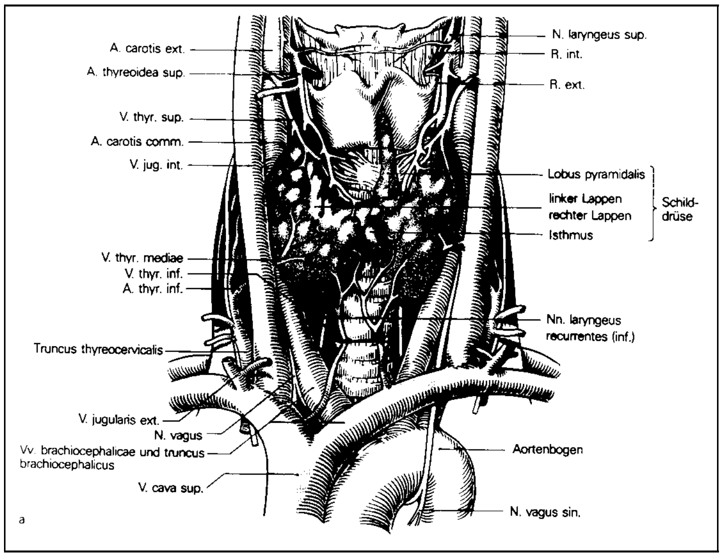
In [30] the approach was applied to parametric surfaces. Elber presented line-art images of NURBS surfaces in [4]. A disadvantage of both approaches is that lines are directed along parameter directions which leads to problems if parametrization changes between patches or, as in the case of triangular surfaces, parametrization is not given explicitly.

Recently, Elber presented a more general but still parameter-based solution for illustrating parametric and implicit forms [5]. Among other techniques, intersections are used to generate line strokes on various objects.

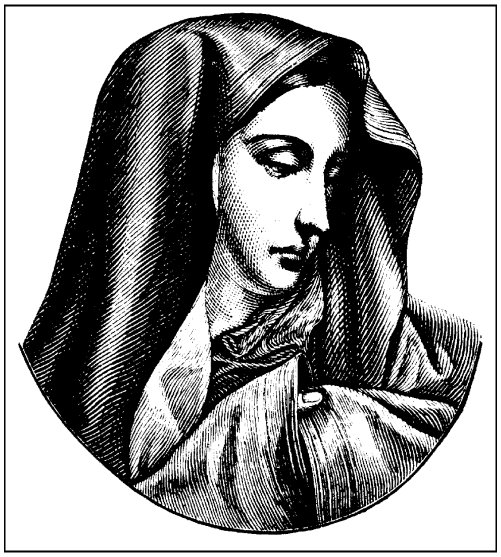
A method for hatching curved surfaces independently to parametrization was proposed by Leister [14]. He uses a special kind of ray tracing in combination with volume textures and image processing operators for the generation of hatching lines. The appearance of the volume texture on the surface of an object is modulated by the light intensity to produce the illusion of hatching lines. Though the images are impressive, aliasing artifacts and numerical inaccuracies occur.

Our approach overcomes some of the restrictions of the previous methods. We are able to process non-parametric surfaces as, for example, polygonal data. A real lighting model including shadows and all other visual effects is used on the basis of a photorealistic shaded image. And

[*]Universittsplatz 2, 39106 Magdeburg, Germany, email: deussen@isg. cs.uni-magdeburg.de, http://isgwww.cs.uni-magdeburg.de/˜deussen



(a)



(b)

Figure 1: Two examples of traditional illustrations using hatching. In both cases, the hatching strokes can be seen as intersection curves with directions chosen in dependence on the geometry.

even more important, the 3-D objects are explicitly intersected with a set of planes.

This approach offers several advantages:

1. Instead of generating hatching lines on a pixel-by-pixel basis, whole curves are generated which are then drawn in a post processing step using a diversity of line styles.

2. The direction of the planes can be chosen automatically for a number of models. This allows controlled-density hatching. Also, artifacts like closed intersection curves can be avoided.

3. Methods that rely on depth-values [18] or intersections between rays and volume textures [14] tend to produce visible artifacts. We avoid this by using a hybrid algorithm that uses analytic clipping and pixel-based post processing. Treating intersection curves as a whole allows us to eliminate remaining artifacts by applying smoothing techniques.

We will demonstrate most of these advantages in the following sections. In Section 2 we review traditional illustration techniques and motivate why the intersection with a set of planes is an appropriate method for hatching curved objects. Section 3 outlines our framework for intersection based hatching.

The basis for all following operations is the generation of intersection curves. This is reviewed in Section 4. Section 5 deals with the definition of such intersection planes in dependence to the model. For a wide range of objects, the orientation of these planes can be generated automatically by computing the geometric skeleton of the object. This will be demonstrated.

Our method of half-toning on the basis of intersection curves is given in Section 6, while Section 7 deals with applying line styles to the illustration.

2 CLASSICAL ILLUSTRATION TECHNIQUES

In Figure 1, two examples of traditional illustrations are given. Figure 1(a) shows an anatomical illustration taken from a textbook, while in Figure 1(b) a picture of a nun is shown which was done by using a copper plate. The hatching lines of both illustrations can be regarded as coming from intersections between parts of the model and a set of planes.

In Figure 1(a) the hatching lines on the veins are drawn in a way that the direction of the intersection planes is always perpendicular to the direction of the veins. This is also the case for the lines on the nun's scarf. It is clear that the lines generated in these examples are not the exact intersection results. They contain some artifacts, noise, and also intentional variations introduced by the artist.

Another observation concerns the appearance of the lines: In the left picture, hatching lines are drawn using a pen, while in the right a copper plate was engraved. These different production processes can be simulated in a computer-generated drawing by using different line styles.

In addition, both pictures approximate the intensity distribution of an imaginative photograph taken from the same scene. Especially the copper plate tries to simulate natural lighting. This is done to imitate the way in which we normally perceive the drawn objects.

3 COMPUTER-GENERATED ILLUSTRATIONS

The above observations lead to an overall scheme for creating illustrations. In this section, the scheme is outlined; the ingredients are described in more detail in the remainder of this paper. The following steps can be distinguished:

1. **Specify the viewing direction, take a snapshot**
   So far, the process of creating hatched illustrations is view-dependend in a way that the intersection lines can be optimally placed only for a small viewing angle.

   We have to specify a viewing direction and then a photorealistic shaded image of the object is to be taken. At this time all kind of global illumination effects like transmission, reflection or shadows can be introduced.

2. **Segment the object**
   Next, the 3-D model has to be segmented into parts that are to be handled by the same line styles and intersection sets. For complex objects dozens of such sets may be used.

   Segmenting the model has to be done manually as it depends on the geometry and the content of the model.

3. **Create the intersection lines**
   For each part of the model the set of intersecting planes is created. The user can define planes interactively while the intermediate planes are interpolated, or the system creates the planes automatically. The number of planes has to be chosen depending on the model, on the viewpoint and on the line style which will be used later.

4. **Calculate the line parameters (Half-toning)**
   In the last step a line renderer is applied to all parts. Each intersection curve is processed in a way that the appearance of the curve approximates the tonal values of the surrounding region. This is a special kind of half-toning based on the intersection curves.

Now suppose that the viewing direction is chosen, the snapshot is taken and the model is segmented in a desirable way. The next step in our framework is to specify and to create the intersections. In the following section the latter is described, in Section 5 two methods of defining intersection planes are given.

4 CREATING INTERSECTION CURVES

The intersection curve between a surface and a plane can be computed either analytically or on a pixel-by-pixel basis. Analytic approaches (as used in [5] for parametric and implicit surfaces) work directly on the model and generate precise results. Each surface description requires its own intersection algorithm and sophisticated implementations are needed to deal with complex objects efficiently.

Pixel-based methods work on a discretization of the object. Every kind of surface can be processed as long as the elements can be discretized. The accuracy depends on the screen resolution and therefore precise results may require a fine discretization.

But how to avoid the aliasing errors that occur in pixel-based algorithms? Fortunately, OpenGL, as the most widespread graphics programming language makes use of a hybrid method that mixes analytic and pixel-based computation. While most of the routines in the graphics pipeline work on the concept of fragments (small parts of objects, in general pixels), clipping is done analytically on the basis of polygons [7]. The user might specify up to six additional clipping planes to the viewing volume.

Such an additional clipping plane that represents the desired intersection plane is used for determining the intersection curve. Our algorithm works as follows:

1. Display model, take snapshot
2. Extract pixels on boundary (Image I_{full})
3. Display model with clipping plane, take snapshot
4. Extract pixels on boundary (Image I_{inters})
5. Generate $I = I_{inters} - I_{full}$
6. Convert I to line segments

The first step is done by drawing the object in white on a black background (cf. Figure 2(a)). Steps two and four are performed on a pixel-by-pixel basis by determing the pixels that have a black neighbour. To avoid fat diagonal lines (see Figure 2(c)) it is usefull to use only the 4-neighbourhood of a pixel. Step five is a simple image operation that can be performed in the accumulation buffer, if available. The conversation from pixels to line segments is done by using least square fitting [16, 24].

A big advantage of the above algorithm is that graphics hardware can be used in steps one, three, and five. Copying of image data and pixel operations are also cheap on

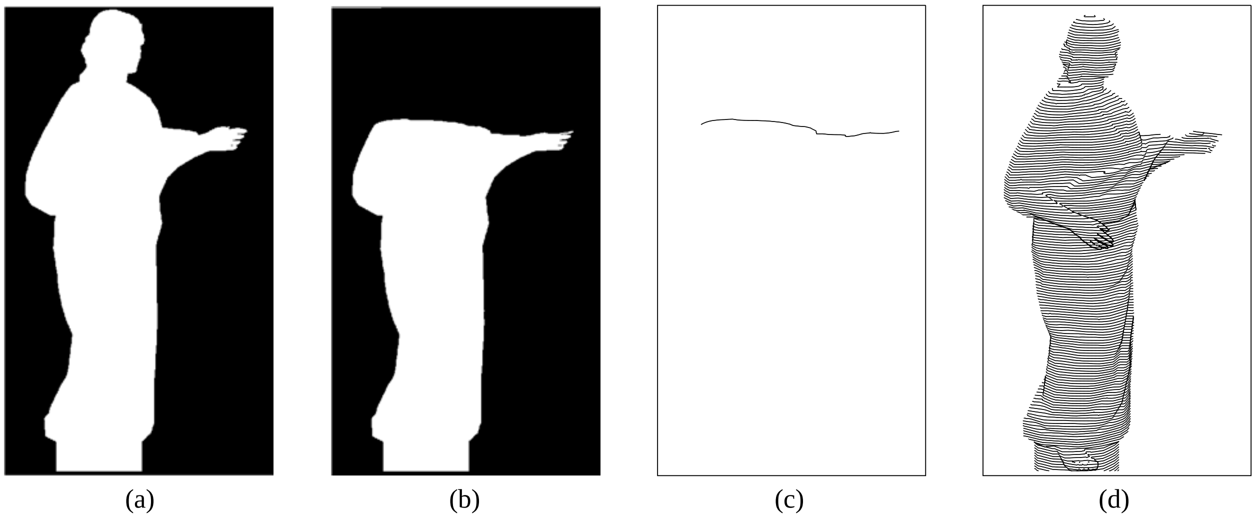


Figure 2: OpenGL-based generation of intersecting lines: (a) Flat shaded image of the model. (b) Image with additional clipping plane. (c) Resulting curve after subtracting the outline of the complete model. (d) The whole set of intersection curves.

modern graphics workstations. This allows to process very complex objects in a reasonable time [2].

The algorithm was also used to generate the silhouettes of the objects to be illustrated. Silhouettes are important ingredients of many line drawings as can be seen below. In this case it is also possible to use an analytic algorithm like the one of Markosian et al. [15].

In Figure 2(c) one important artifact can be seen: The intersection lines in the region of the statue's head and the shoulder are visually unpleasant. If the orientation of the planes is chosen differently, this effect can be even worse. For closed objects like a simple sphere, it is even impossible to find a single set of intersecting planes that does not have such regions somewhere on the surface.

Similar problems occur in NC applications like milling where the trajectories of the cutters have to be determined. A region like the statue's head would introduce a larger error than other parts of the object.

In our case we have the advantage that only one view has to be generated with visually pleasing curves. So we avoid the effect by defining the set of planes individually for each viewing direction and by choosing sets of planes which are not parallel. This resembles what is done by artists in their artwork.

5 DEFINING INTERSECTION PLANES

Section 4 described the computation of an intersection curve if the corresponding intersection plane is given. Like motivated above, the appropriate definition of a set of these planes is crucial for generating pleasant illustrations.

In this section we introduce two methods for defining intersection planes: One is the interactive specification which gives the user full control over the planes but requires some interactive work, the second method determines the planes automatically for a given geometry.

5.1 Interactive Specification

To specify a set of planes interactively, the user has to define a spline curve by positioning control points. Equidistant points between the specified control points are used to define each of the intersection planes. The user types in the number of equidistant points to be used, these points are automatically generated and visualized by the system during movement of the control points. At each intermediate point, the intersection plane is defined by the point itself and the derivative of the spline curve.

The derivative of the curve at the control points is visualized by handles in the form of pyramids. The direction of a pyramid defines the tangent of the spline curve, the control point is the center point of the pyramid's base. The user specifies a number of such pyramids which are drawn semi-transparent. As the spline curve usually has to lie inside the geometry, the geometry is also displayed semi-transparent.

5.2 Automated specification

For many models, the orientation of the intersection planes can be computed automatically. We do this by calculating a skeleton of the model. This skeleton produces a graph of line segments inside the object that represents topological and geometric properties of the object. The topology is stored in the graph structure and the geometrical information is stored in the length and angle of the edges [6]. In Figure 3, an object and its geometric skeleton is shown.

Generating the skeleton is traditionally done by using one of two methods: thinning algorithms based on pixels or voxels are used for unstructured data [12, 28] and Voronoi-based approaches for structured input [6]. However, both methods are inefficient for the complex objects we want to process. Instead, we use the algorithm developed by one of the authors in [17], which works directly on the geometry of the object.

The algorithm assumes that the object is represented by a triangular mesh M^0 with n edges. The basic element is the edge collapse operation introduced by Hoppe et al. [9]. Instead of using the edge collapse for reducing the geometry, our sequence of edge collapses goes much further: The initial mesh is reduced until no regular triangles exist any more.

Degenerated triangles with zero area would also appear in Hoppe's algorithm if the algorithm is not forced to omit places that will generate such triangles. In the present case, we treat the degenerated triangles as line segments and build a graph out of these.

Additionally, in each reduction step, the edges of the mesh are sorted by their length in order to collapse short edges first. This is motivated by the fact that collapsing a short edge produces only a small geometric deformation and therefore the skeleton is located as near as possible to the initial geometry. The outline of the algorithm is as follows:

```
sort all edges of the model according to their length
while faces remain in the model do
    take shortest edge $e(v_1, v_2)$
    collapse edge
    for all adjacent edges $e_v$ of $v_1$ and $v_2$ do
        if $e_v$ has no regular faces
            add $e_v$ to the skeleton
            delete $e_v$ from model
        else
            re-sort edges with $e_v$
```

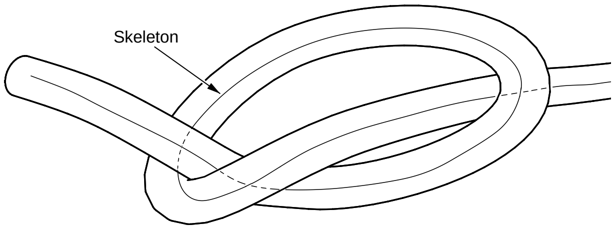


Figure 3: A geometric object and its skeleton

Instead of sorting the edges after each edge collapse, during one step a series of collapses can be performed. To avoid problems with updating the data structure, each collapsed vertex is marked with all adjacent vertices. As stated in [17], this refinement of the algorithm works in the average case with a time complexity of $O(n \log n)$ for n collapses. In the worst case a complexity $O(n^2 \log n)$ is achieved.

After performing the skeleton operation, the resulting line segments can be smoothed or converted to a spline. The resulting curve is now used for defining the intersection planes as described above.

Figure 4 gives an example of a curved object that was hatched by using a skeleton. The appearance of the lines was generated by so called priorized drawing and half-toning. Both techniques are described in the next section.

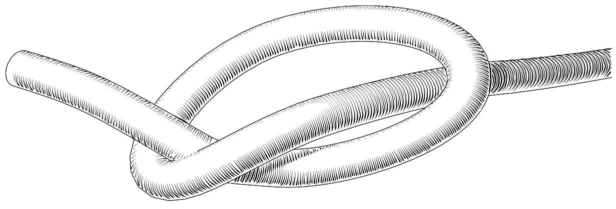


Figure 4: Hatching a knot by using the geometric skeleton.

6 HALF-TONING USING INTERSECTIONS

As mentioned above, classical illustrations often try to simulate natural lighting by the means of hatching lines. To incorporate this into our illustration framework, we use a photorealistic shaded image of our model and perform a half toning step based on the hatching curves.

Each line is responsible for representing the tonal value of that part of the surface which is near the curve, e.g., that is the Voronoi region of the curve with respect to the neighbouring curves.

To compute the appropriate regions efficiently, we generated twice as much intersection curves as needed and used the intermediate curves for determining the regions of each hatching curve. Now the tonal value of the region is calculated along the line and the line thickness at equidistant points is computed in order to represent the gray-scale tone. As a result, the line fills the region by its thickness if the part of the region is black, if the tonal value is 50%, the line width is half the width of the region.

6.1 Drawing with priorized hatching lines

It is well known in non-photorealistic rendering that hatching of curved objects requires adaptive density control [21]. If half-toning should be achieved only by enlarging the thickness of the intersection lines, either ugly fat lines will appear at the outside of a curved object or the tone is too light. In Figure 5(a) the problem can be seen. Here, the knot was hatched using a single set of curves.

Our solution to the problem is to use priorized drawing of the curves. This is done by generating as many curves as needed for nicely hatching the outer parts of the object. Let us assume that we need four times more lines in these regions than for the inner parts of the model. This is sufficient in most cases; if more lines are needed, the method can easily be extended.

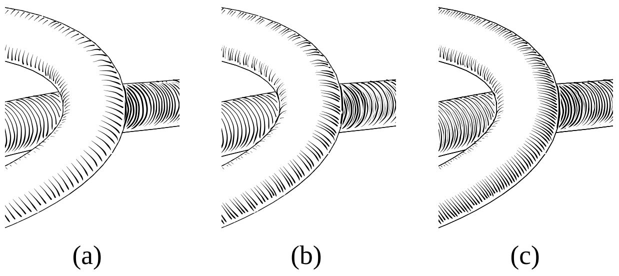


Figure 5: An example of priorized drawing; (a) a single set of lines is used, in (b) and (c) two more line sets are added.

First, every fourth curve is drawn. The line widths are restricted to be below a given maximal width. Consequently, the corresponding regions do not fully partition the object.

On top of these curves every second curve is drawn. In this case the lines are drawn using a white halo corresponding to the region of the surface that is represented by the curve. In the last step, the remaining curves are also drawn haloed. Figure 5 shows the process. The effect that the lines which were drawn first are partly erased by following lines can be seen in Figure 5(b) and (c) on the straight part of the geometry in the background.

6.2 An example

In Figure 6, the whole process is shown as applied to a bust of Beethoven. On the left, the shaded image of the bust is shown. On the right, we see a computer-generated copper plate using our techniques. As described in Section 3, the lines were calculated separately for each part of the model and the line widths were calculated to achieve half-toning.

For the face, two sets of lines were combined. One set was drawn using black lines and one using white lines. By combining the line thickness of both black and white lines appropriately, half-toning is achieved.

7 APPLYING LINE STYLES

After the intersection curves are defined and the line width is calculated, one degree of freedom is still left: the line style that is to be used for drawing the line.

In our system, the intersection curves form the path of each stroke to be drawn [22]. Several attributes like thickness and tone can be added to vary the output. The line styles might introduce some extra curves or simulate the effect of unstable strokes. In the case of scientific illustrations, such line styles are sometimes used for example to visualize motion.

In Figure 7 the bones of a foot are illustrated by four methods. Two images show only the outline which was drawn using different line styles. In the other images intersection lines are added. All images show the model in visualization styles that can be found in traditional scientific illustrations.

8 CONCLUSION AND FURTHER WORK

We presented a method to generate crosshatched illustrations of geometric objects. The method uses intersections of the geometry with a set of planes for generating the hatching curves, the intersections are determined depending to the model. This is done either by interactive specification or by automated generation. The skeleton needed here was generated by a new algorithm that works directly on the surface of the object.

A hybrid analytic and pixel-based intersection algorithm was proposed that allows to process even highly complex objects efficiently. The resulting curves were drawn in order to achieve half-toning. This can also be found in many traditional illustrations. The system allows to generate different line styles in order to achieve a wide range of illustrations.

Future work will include other intersection primitives like cylinders or spheres. This will help to achieve new artistic images but will also require a new process of clipping as these clipping primitives are not supported by the graphics hardware.

Currently, branching objects like veins or trees cannot be hatched automatically as the branches have to be processed in a special way. This problem has to be solved first. Also, other objects like plants or technical devices have to be considered and special hatching methods have to be developed.

9 References

[1] A. Appel, F. J. Rohlf, and A. J. Stein. The haloed line effect for hidden line elimination. *Computer Graphics (SIGGRAPH '79 Proceedings)*, 13(3):151–157, August 1979.

[2] O. Deussen. Pixel-oriented rendering of line drawings. In T. Strothotte, editor, *Computational Visualization: Graphics, Abstraction and Interactivity*, pages 105–120. Springer Verlag, 1998.

[3] D. Dooley and M. Cohen. Automatic illustration of 3D geometric models: Lines. *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, 24(2):77–82, March 1990.

[4] G. Elber. Line art rendering via a coverage of isoparametric curves. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):231–239, September 1995.

[5] G. Elber. Line art illustrations of parametric and implicit forms. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):71–81, 1998.

[6] E. Ferley, M. Cani-Gascuel, and D. Attali. Skeletal reconstruction of branching shapes. *Computer Graphics Forum*, 16(5):283–293, 1997.

[7] C. Frazier. The OpenGL Graphics System: A Specification (Version 1.1). http://www.opengl.org/Documentation/Specs.html, 1992.

[8] Q. Guo and T. Kunii. Modeling the diffuse painting of sumie. In T. L. Kunii, editor, *IFIP Modeling in Computer Graphics*, 1991.

[9] H. Hoppe. Progressive meshes. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[10] S. Hsu and I. Lee. Drawing and animation using skeletal strokes. In A. Glassner, editor, *SIGGRAPH '94 Conference Proceedings*, pages 109–118. ACM SIGGRAPH, ACM Press, July 1994.

[11] T. Kamada and S. Kawai. An enhanced treatment of hidden lines. *ACM Transactions on Graphics*, 6(4):309–323, 1987.

[12] L. Lam, S.-W. Lee, and C.Y. Suen. Thinning methodologies - a comprehensive study. *IEEE PAMI*, 14(9):869–885, 1992.

[13] J. Lansdown and S.Schofield. Expressive rendering: A review of nonphotorealistic techniques. *IEEE Computer Graphics and Applications*, 15(3):29–37, May 1995.

[14] W. Leister. Computer generated copper plates. *Computer Graphics Forum*, 13(1):69–77, 1994.

[15] L. Markosian, M.A. Kowalski, S.J. Trychin, L.D. Bourdev, D. Goldstein, and J.F. Hughes. Real-time nonphotorealistic rendering. In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, pages 415–420. ACM SIGGRAPH, Addison Wesley, August 1997.

[16] J.R. Parker. Extracting vectors from raster images. *Computers & Graphics*, 12(1):75–79, 1988.

[17] A. Raab. *Techniques for Interacting with and Visualization of Geometric Models*. PhD thesis, Otto-von-Guericke Univerity of Magdeburg, 1998.

[18] T. Saito and T. Takahashi. Comprehensive rendering of 3-d shapes. In *Computer Graphics (Proc. SIGGRAPH 90)*, volume 24(4), pages 197–206. ACM SIGGRAPH, ACM Press, 1990.

[19] M. Salisbury, C. Anderson, D. Lischinski, and D. Salesin. Scale-dependent reproduction of pen-and-ink illustrations. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pages 461–468. ACM SIGGRAPH, Addison Wesley, 1996.

[20] M. Salisbury, S. Anderson, R. Barzel, and D. Salesin. Interactive pen–and–ink illustration. In A. Glassner, editor, *SIGGRAPH '94 Conference Proceedings*, pages 101–108. ACM SIGGRAPH, ACM Press, 1994.

[21] M. Salisbury, M. Wong, J. Hughes, and D. Salesin. Orientable textures for image-based pen-and-ink illustration. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*. ACM SIGGRAPH, Addison Wesley, August 1997.

[22] S. Schlechtweg, B. Schönwälder, L. Schumann, and T. Strothotte. Surfaces to Lines: Rendering Rich Line Drawings. In V. Skala, editor, *Proceedings of WSCG'98 (Plzeň, Czech Republic, February 1998)*, volume 2, pages 354–361, 1998.

[23] J. Schumann, T. Strothotte, A. Raab, and S. Laser. Assessing the effect of non-photorealistic images in computer-aided design. In *ACM Human Factors in Computing Systems, SIGCHI '96*, pages 35–41, April 13-15 1996.

[24] J. Sklansky and V. Gonzales. Fast polygonal approximation of digitized curves. *Jorunal of the ACM*, 18(2):255–264, 1979.

[25] S. Strassmann. Hairy brushes. In D.C. Evans and R.J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 225–232, August 1986.

[26] C. Strothotte and T. Strothotte. *Seeing Between the Pixels: Pictures in Interactive Systems*. Springer-Verlag, Berlin-Heidelberg-New York, 1997.

[27] T. Strothotte. *Computational Visualization: Graphics, Abstraction and Interactivity*. Springer-Verlag, Berlin-Heidelberg-New York, 1998.

[28] C.-Y. Suen and P. Wang. *Thinning methodologies for pattern recognition*. World Scientific, 1994.

[29] G. Winkenbach and D. Salesin. Computer–generated pen–and–ink illustration. In A. Glassner, editor, *SIGGRAPH '94 Conference Proceedings*, pages 91–100. ACM SIGGRAPH, ACM Press, 1994.

[30] G. Winkenbach and D. Salesin. Rendering parametric surfaces in pen and ink. In H. Rushmeier, editor, *SIGGRAPH '96 Conference Proceedings*, pages 469–476. ACM SIGGRAPH, Addison Wesley, 1996.

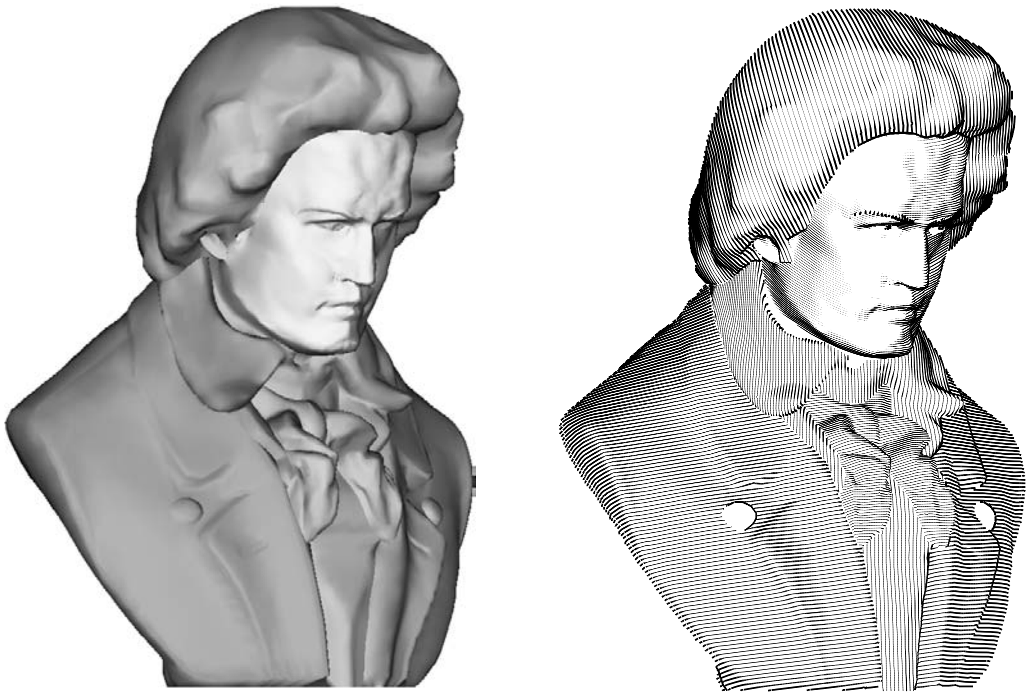


Figure 6: (a) A conventional shaded image of a bust of Beethoven; (b) computer generated copper plate using the image.

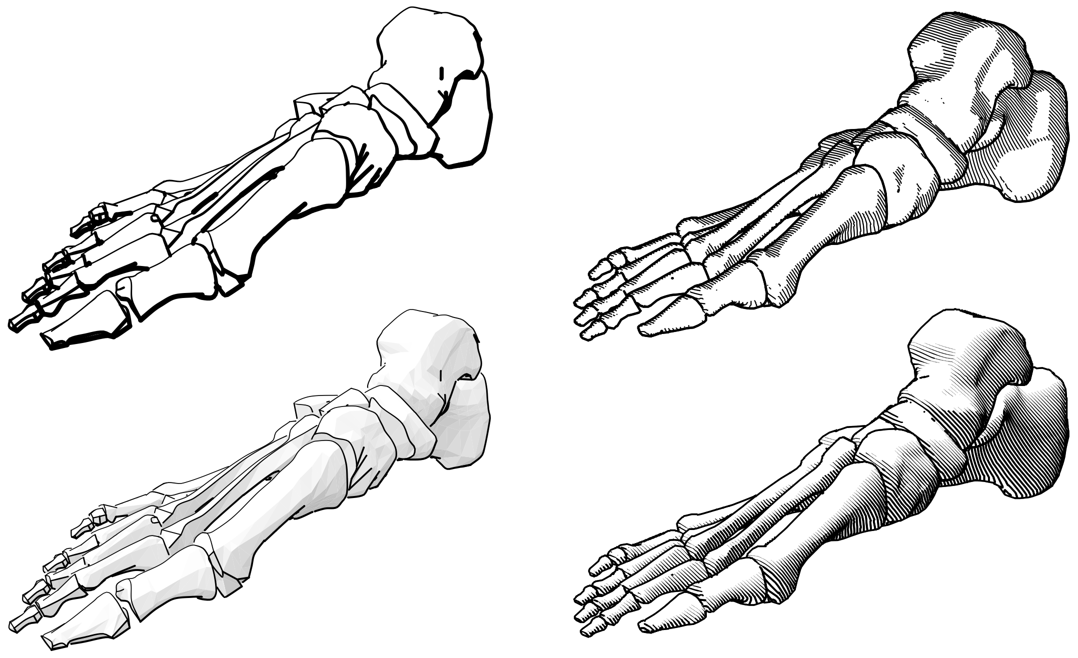


Figure 7: Several illustrations of a foot. Left column: Two different line styles were used to draw the outline of the foot. In the upper image the line thickness is varied according to a virtual light source, in the lower image a uniform line thickness is used. Right column: In the upper image uniform lines were drawn, in the lower image a different light source was used and the hatching lines were varied to achieve half-toning.