Line-Art Rendering of 3D-Models

Christian Rössl Leif Kobbelt

Max-Planck-Institute for Computer Sciences
Stuhlsatzenhausweg 85, 66133 Saarbrücken, Germany
{roessl,kobbelt}mpi-sb.mpg.de

Abstract

We present an interactive system for computer aided generation of line art drawings to illustrate 3D models that are given as triangulated surfaces. In a preprocessing step an enhanced 2D view of the scene is computed by sampling for every pixel the shading, the normal vectors and the principal directions obtained from discrete curvature analysis. Then streamlines are traced in the 2D direction fields and are used to define line strokes. In order to reduce noise artifacts the user may interactively select sparse reference lines and the system will automatically fill in additional strokes. By exploiting the special structure of the streamlines an intuitive and simple tone mapping algorithm can be derived to generate the final rendering.

1. Introduction

In contrast to photorealistic images based on physical simulation of lighting and shading, non-photorealistic rendering techniques offer various advantages for printed illustrations. The higher level of abstraction that they provide helps to put emphasis on the crucial information of an illustration. Abstraction mechanisms available to designers include levels of detail and focus attention to stress the important parts of a drawing or enriching an image with additional visual information and effects.

Typical applications for the use of non-photorealism are e.g. architectural sketches, illustrations in medical text books or technical manuals, and cartoons. Many of these applications prefer a very specific technique: line rendering. Here, a picture consists of monochrome lines or strokes only. An artist may chose between different line styles or vary line attributes like length, thickness or waviness. There is a broad spectrum of drawing styles, that have been developed in fine arts for centuries. Well known examples are silhouette drawings where just the contours of an object are outlined or more sophisticated techniques like engraved copper plates that have formerly been used for printing or pen-and-ink illustrations.

Apart from the advantage of abstraction or artistic features line drawings also provide some technical benefits: they can be stored resolution independent or provide level of detail like features for different resolutions. Besides, such pictures can easily be reproduced on any monochrome media.

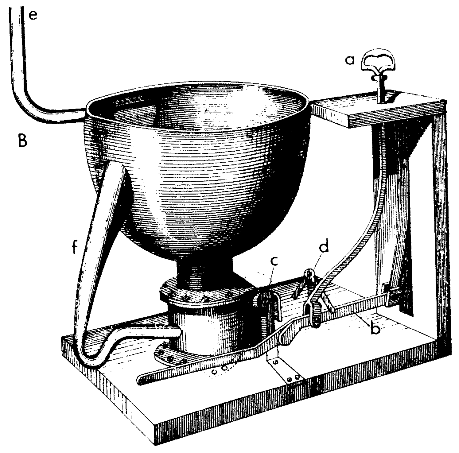

Figure 1. Hand-made line art drawing have been used for centuries. Our example is from 1778. Such illustrations provide a high density of clearly perceptible information while also satifying aesthetic standards.

In this paper we present a tool that assists an artist in manufacturing a line drawing from a 3D model. The geometry of the object is given as a triangle mesh. This allows us to handle a huge class of models ranging from technical constructions in CAD to real world data that is obtained from a range scanner. The system frees the user from drawing individual lines or strokes. Instead curvature data is first sampled from the object and then used to semi-automatically generate lines. Our rendering style produces long single or cross hatched lines of varying thickness that are especially appropriate for technical drawings.

2. Related Work

Over the last years various computer based techniques for producing line drawings semi-automatically or even automatically have been developed. There are two basic approaches: The first one is image based. Here, the user is assisted in converting a digital grey scale image into a line drawing without having to draw individual strokes. [12] define a potential field over an image and draw equipotential lines of varying density. In [15] predefined stroke textures are used to map the tone of the reference image and to produce a pen-and-ink illustration. This method is improved in [16] in a way that the stroke textures are generated automatically from both a set of reference strokes and an interactively modifiable direction field.

The second approach takes into account the 3D geometry of a scene. Here, an early step was the use of haloed lines [1] that give an impression of depth and the use of different line styles for outline and shading [4]. [19] utilizes special stroke textures to render high quality pen-and-ink illustrations from polygonal models. These stroke textures are created for different materials and resemble hand draw strokes. During shading the strokes are mapped to image space together with the corresponding polygons and are used for tone mapping then. This system was extended to process free-form surfaces in [20] where isoparametric curves are used for rendering, following [5]. Such curves look especially well for surfaces of revolution. Elber also employs lines of curvature of parametric surfaces and contour lines of implicit surfaces [6]. By precalculating these stokes and applying a simple shader even interactiveness can be achieved [7].

Most recently, [21] generate a discrete direction field over a triangulated (subdivision) surface. Originally the principal directions are calculated via discrete curvature analysis. In order to reduce noise and to get a pleasant rendering this field is globally optimized in a second step by minimizing some energy functional. For the final picture a silhouette drawing is combined with hatches from the filtered principal directions.

The previously mentioned approaches that respect the geometry of an object are all able to produce "real" stokes, e.g. in Postscript. Besides, there is another class of techniques that are pixel based and produce discrete images. [14] introduce the concept of G-buffers that provide additional information per pixel such as depth value, normal vector or the parameter value of a parametric surface. Non-photorealistic images that resemble line drawings can then be generated by applying well known image processing operators. In contrast, [9] uses a modified ray-tracing algorithm for emulating engraved copper plates. With processing pixel data it is also possible to take advantage of graphics hardware: [3] generate copper plate like images from intersection lines of OpenGL clipping planes with the object.

The approach that is presented in this paper examines the geometry of a model by approximating the principal directions in every vertex of the given triangle mesh. Then this data is interpolated across triangles and sampled per pixel. Here, the G-buffer concept is used, and the graphics hardware is exploited. Once all data is grabbed from image buffers the user may modify these buffers and an additional ("stencil") buffer and interactively place continuous streamlines. Instead of generating all strokes from streamlines, the system lets the user chose a few reference lines and generates strokes by interpolation between these curves. This guarantees that only good, visually appealing strokes that are not disturbed by noise are rendered. For the final Postscript image the silhouette that has been grabbed from an image buffer and converted to a set of polygons is added.

3. Overview of the algorithm

In this section we give a short overview over the various stages of our algorithm. The details are explained in the subsequent sections. The input data is given by an arbitrary triangle mesh viewed from a given perspective and the output is a line art image of the object. For the preprocessing of the surface, we have to assume that the mesh is mostly manifold. Non-manifold artifacts can be handled as long as they do not cover a significant part of the image.

The **first phase** of the algorithm preprocesses the mesh data. For every vertex of the mesh intrinsic geometric attributes like the normal vector and the principal curvature directions are computed. This information is needed to determine the stroke placement and orientation in the later phases. The strokelines or hatches of our line art renderings will follow the (projected) principal curvature directions on the given surface, since those directions characterize the underlying geometry very well and carry a maximum shape information. The user can control the geometric detail considering a smoothed version [8] of the mesh for curvature analysis. Although the geometric attributes are estimated for the vertices only, we can (linearly) interpolate them across the triangles to obtain a set of continuous attribute fields that are defined everywhere on the given surface.

The 3D-object is then rendered together with its attributes into a so-called *enhanced frame buffer* similar to a *G-buffer*. For every pixel this frame buffer contains lots of information about the visible surface point. The pixel data includes the local shading (grey value) which later determines the local stroke density. Additionally, the normal vector is stored for every pixel which provides the necessary information for contour extraction. For the orientation of the strokelines, the projected principal curvature directions are also stored.

In the **second phase** the enhanced pixel information is used to partition the image in regions with homogeneous principal direction fields. Not all criteria for this segmentation are based on pure geometry. In fact, the segmentation is the most artistic part of the line art image generation. Consequently we allow the user to interactively control the segmentation process. The information from the enhanced frame buffer is used to implement an intuitive user interface.

Each separated region can be rendered by a single grid of strokelines in a line art style. For every segment we hence generate a *fishbone* structure which captures all necessary data for the stroke generation. To optimize the quality of the resulting line art image we have to apply different types of filter operations that remove local turbulences in the principal direction fields.

The **third phase** then performs the actual tone mapping, i.e., the translation of directional and shading information into quasi-parallel hatches with appropriate width. This phase defines the actual "style" for the line art rendering. Our goal is to distribute the strokes as uniformly as possible while making their construction as local as possible.

4. Preprocessing the surface geometry

Non-photorealistic renderings of complex shapes usually exploit the differential geometric attributes of the surfaces. Grids of principal curvature directions and geodesic lines provide natural (orthogonal) parameterizations for freeform surfaces and the strokeline orientation can be based on the iso-lines of such parameterizations [5, 6]. For piecewise linear surface representations these concepts have to be generalized. By locally fitting a polynomial patch to a mesh vertex and its adjacent neighbors, we can compute first and second fundamental forms for each vertex [18, 13, 17]. From these we can derive normal and principal curvature directions. For surface points in the interior of the triangular faces these attributes can be computed by barycentric interpolation of the values in the corners.

This interpolation step can be done very elegantly by the underlying rasterization engine which maps the 3D mesh to the frame buffer: vector or scalar valued attributes can be encoded as RGB colors and assigned to the mesh vertices. Rendering the object without any shading then generates a frame buffer pixel matrix which stores the interpolated values. Arbitrary precision can be obtained by appropriate scaling and multipass rendering. For vector valued attributes a concluding normalization step might be necessary. We render each attribute field separately. In the end we obtain a set of images, one of them containing the (grey-scale) shaded view of the object. The other images show the color coded attribute fields, one for the normal vectors and one for each principal direction. We call the collection of all these images (with identical resolution) the *enhanced frame buffer* with many different values stored for each pixel. We prefer this term to *G-buffer* [14] because we interprete pixels as *discrete* samples that are used to (re-)construct *continous* strokes.

Once the given 3D-object is rendered, the enhanced frame buffer contains all the necessary information for the subsequent steps of the algorithm. Hence, phases two and three entirely work in 2D which reduces their computation costs.

5. Image segmentation

After the preprocessing all relevant geometric information about the original 3D-model is encoded in a regular pixel matrix. Hence the following steps of the algorithm do not have to consider the (possibly complex) topology of the original mesh data. Most operations can be expressed and implemented as image processing operators which rely on the regular grid structure.

At first we apply simple Gaussian or Median filters to the frame buffer. This is necessary to remove high frequency noise from the attribute fields which sometimes emerges from sampling artifacts during the rasterization. Later, the principal direction fields are used to generate strokelines by line integration. Low-pass filtering on these direction fields hence have a smoothing effect on the strokelines leading to an improved visual appearance of the line art image.

From the normal direction field we can easily extract the silhouette lines for the object. We do this by dot multiplication of the normal vectors with the viewing direction and extracting the zero-contour.

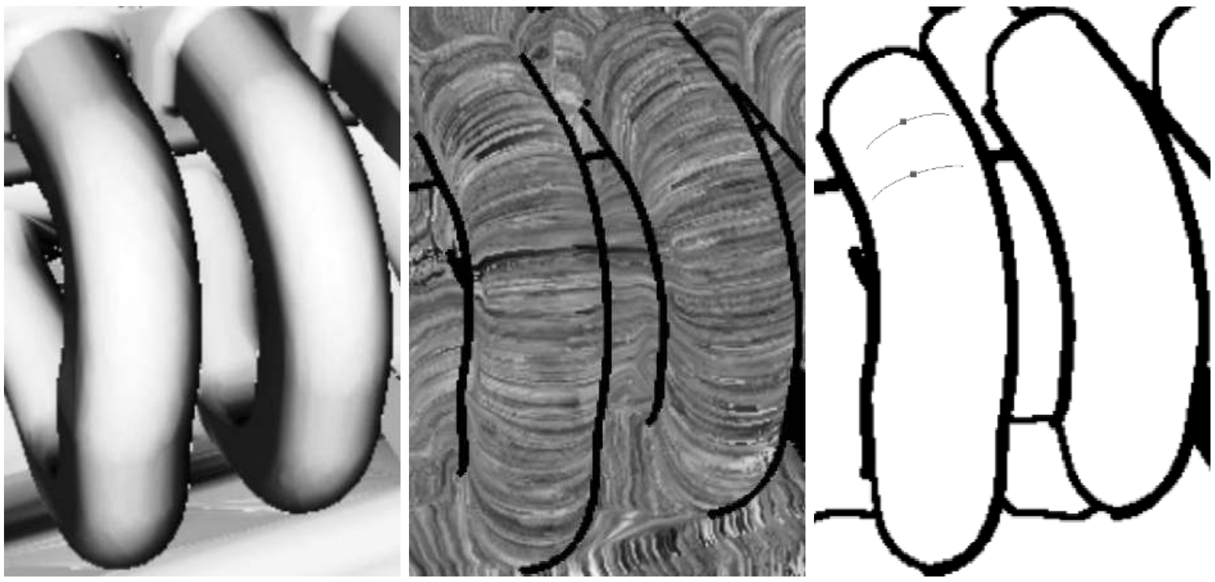

Figure 2. To convert the shaded image on the left into a line art image, we have to partition the image in regions with coherent principal direction fields. A LIC image with overlayed contours (center) helps the user to do the segmentation interactively. In addition the user can probe streamlines (right).

This works for parallel projection as well as perspective projection. An efficient technique to compute a polygonal approximation of that zero-contour is a two-dimensional variant of the marching cubes algorithm (*marching squares*) [10]. The smoothed contour polygons resemble thick, slightly waved hand drawn strokes.

Our goal is to decompose the image into several regions which have a strong coherence in the principal direction fields since these areas are to be rendered by a single set of quasi-parallel hatches. Silhouette lines serve as a reliable detector for the boundary between such regions. In addition, image processing operators can be applied to the sampled data. We believe that automatic algorithms can only provide a rough initial segmentation that is to be refined manually.

The segmentation of the input image is the most sophisticated part of the generation of line art images. Since the segmentation is usually driven by non-geometric attributes such as functional grouping in technical parts or implicit color and texture information, we allow the user to interactively control the partitioning. In our user interface, we display a LIC image [2] based on the maximum curvature direction field overlayed with the automatically extracted silhouettes (which serve as initial segmentation). The LIC image gives a good and intuitive perception of regions with coherent flow. The user can now simply *draw* the segment boundaries on the screen. In practice we observed that the interactive segmentation can be done in several minutes even for moderately complex objects (Fig. 2).

If a surface has too much fine detail relative to the sampling density in the frame buffer, we sometimes find regions in the image where no natural direction field for the strokeline orientation can be defined. In such cases, our interface allows the user to override the directional information in the frame buffer and to locally define a synthetic direction field. With a technique similar to [11] we use the partial derivatives of bi-linear Coons-patches to generate these synthetic direction fields. In the final line art image, this fine detail will be taken into account by the tone mapping.

6. Generating hatches

For every segment with coherent principal direction fields we define a grid of hatches. We build this grid in a *fishbone* fashion by first picking a *backbone* and then arranging the *ribs* in orthogonal direction.

To define the backbone, the user can pick an arbitrary point in the interior of the current segment. Starting at that point we compute the backbone curve by integrating along the *minimum* curvature direction field. We use simple forward Euler for tracing the streamline. On the backbone curve we distribute sam-

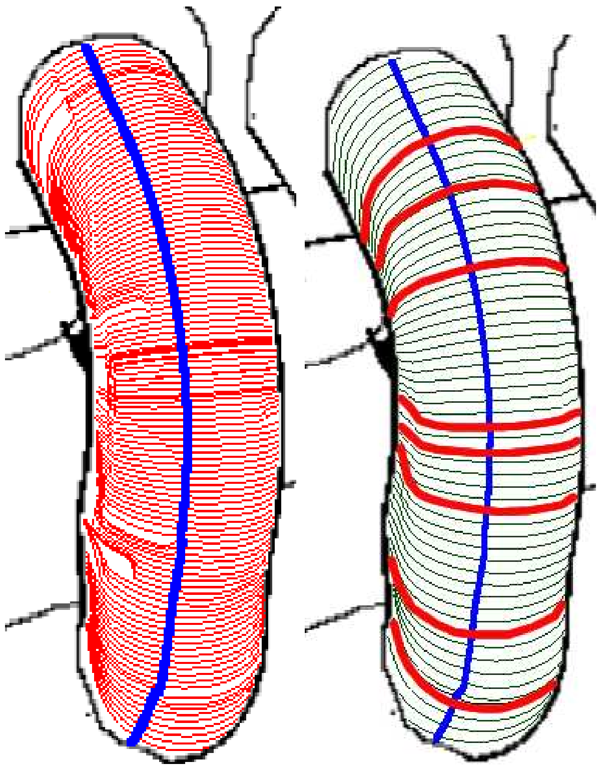

Figure 3. Hatch generation is based on a fishbone structure. The backbone is defined by a curve following the minimum curvature directions and the ribs are computed by tracing along the maximum curvature. In order to reduce the noise in the direction field (left) and to avoid discretization artifacts, the user can define a sparse set of "key-ribs" and a dense set of ribs is constructed by interpolation (right).

ples with constant arc-length distance. The orthogonal ribs are then computed by starting at those samples and integrating along the *maximum* curvature direction.

In some cases, however, this simple rib generation technique fails to produce useful strokelines. Due to the discretization of the direction field, neighboring lines can merge which destroys the global fishbone structure (Fig. 3, left). In order to avoid this effect, the user can manually place a few sample points on the backbone from where the *key-ribs* are traced along the maximum curvature directions. Inbetween these key-ribs we uniformly distribute additional *blended ribs.* The blended ribs result from interpolating between the key-ribs. By properly choosing the starting points for

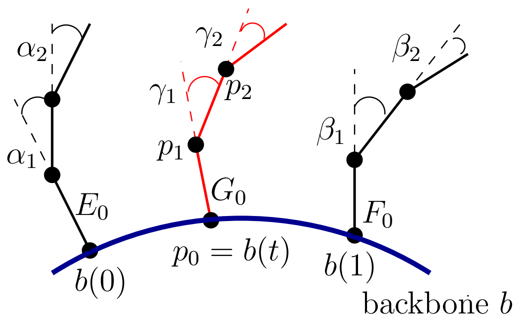

Figure 4. The blending of strokes is based on a decomposition into *orientation* (E_0,F_0) and *characteristic shape* ($\{\alpha_i\}$,$\{\beta_i\}$) and controlled by a parameter $t \in [0, 1]$ with the backbone curve b locally parameterized as shown .

the key-ribs, we generate a high quality set of pseudo-parallel strokelines (Fig. 3, right).

Each strokeline is represented by a polygon with constant edge length h (arc-length parameterization). To uniquely describe the shape of a rib strokeline we need the first polygon edge E_0 and a sequence of angles α_i between successive edges E_i and E_{i+1}. The complete strokeline can then be reconstructed by starting with the first edge E_0 and adding more edges E_{i+1} with the same length in the direction determined by the angles α_i. In the strokeline representation $[E_0, \{\alpha_i\}]$, the edge E_0 determines the *orientation* and the sequence $\{\alpha_i\}$ determines the characteristic shape (Fig. 4).

Assume we are given two key-ribs $[E_0, \{\alpha_i\}]$ and $[F_0, \{\beta_i\}]$ which start on the same backbone. Then for every value $t \in [0, 1]$ we find a new starting point on the backbone arc between the two key-ribs and the corresponding blended rib is given by $[G_0, \{\gamma_i\} = \{(1-t)\,\alpha_i + t\,\beta_i\}]$ where the orientation G_0 is given by an weighted average of E_0 and F_0 and the characteristic shape is a weighted blend of the two key-ribs. Using this blending technique we can generate very good fishbone type strokelines by prescribing only rather few key-ribs.

After the rib generation, the fishbone structure is given by a set of polygons with unit edge length h. For rendering purposes, the kth vertex $\mathbf{p}_k^{(l)}$ of the lth rib $R_l = [G_0, \{\gamma_i\}]$ can be computed by

$$\mathbf{p}_k^{(l)} \;=\; \mathbf{p}_0 + G_0 + h \sum_{i=1}^{k-1} \begin{pmatrix} \cos(\sum_{j=1}^{i} \gamma_j) \\ \sin(\sum_{j=1}^{i} \gamma_j) \end{pmatrix},$$

or

$$\mathbf{p}_k^{(l)} \;=\; \mathbf{p}_0 - G_0 - h \sum_{i=1}^{-k-1} \begin{pmatrix} \cos(\sum_{j=1}^{i} \gamma_{-j}) \\ \sin(\sum_{j=1}^{i} \gamma_{-j}) \end{pmatrix}$$

for negative k.

The organization of the rib vertices $\mathbf{p}_k^{(l)}$ to a sequence of sequences $[[\mathbf{p}_k^{(l)}]_k]_l$ corresponds to a rectilinear matrix type structure where the vertices of one rib form a row $[\mathbf{p}_k^{(l)}]_k$ and the kth vertex for all ribs $[\mathbf{p}_k^{(l)}]_l$ form a column. Hence, it is natural to exploit this structure for the tone mapping. To simplify further processing, we sample the shading values at the locations $\mathbf{p}_k^{(l)}$ and pass a resampled attribute matrix to the tone mapping procedure. In that procedure, a stroke width value $w_k^{(l)}$ is computed for every location $\mathbf{p}_k^{(l)}$. For this we do not need any directional information.

7. Tone mapping

The last step of the algorithm is the translation of grey value shading information into stroke widths. For this translation several aspects have to be taken into account.

First of all the local brightness of the rendered image obviously depends on the ratio between stroke width and distance between neighboring strokes. If strokes lie more densely, the width of the strokes has to decrease to maintain constant shading. Another possibility is to suppress a part of the strokes and leave the others with their original width. The problem with the suppressing of certain strokelines is that disturbing low-frequency patterns can appear in the final image if the lines are chosen according to a periodic rule. Adapting the distance of neighboring strokes to the local grey value is not an option since that value usually changes along the stroke but the distance cannot be controlled freely.

Many sophisticated techniques have been proposed for the tone mapping in line art images [15, 20, 16].

We use a simple and efficient technique which does not use any global information but still has flexibility to adjust stroke widths and suppress strokes.

The idea is to define strokes to have a constant width w but only a certain portion $w' = (1-c)\,w$ is drawn in black where $c \in [c_{\min}, c_{\max}] \subset [0,1]$ is the local grey value (0=black). Restricting the grey values to the interval $[c_{\min}, c_{\max}]$ guarantees a minimum width of the strokes and a minimum width of the white space between strokes. Since strokes have a constant width they partially overlap if neighboring strokes come too

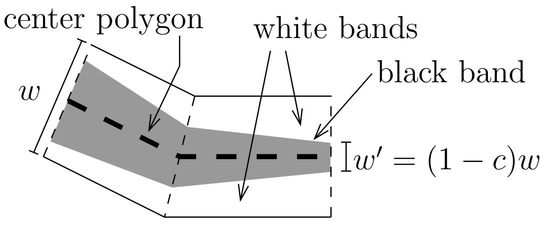

Figure 5. The strokes we use have constant width. Only a certain portion of a stroke is drawn in black while the rest is drawn as two white bands on both sides.

close together. If the strokes are painted one after the other then one stroke can delete parts of its neighbors. In the extreme case two non-neighboring strokes can approach so close that all the strokes between them are completely removed.

The technique described so far controls the local tone mapping by adjusting the stroke widths and automatically removes some strokes if their density increases. The remaining question is how to avoid low-frequency patterns in the distribution strokelines. We solve this problem by drawing the strokes in a special order which guarantees that the right strokes are overpainted and the surviving ones are equally distributed.

A sequence of uniformly distributed strokelines (without low frequency patterns) can be generated by drawing every 2^kth line. If the strokelines become denser we want the strokes from coarser distributions (higher values k) to survive. Hence we have to start drawing the finest level containing the strokes $[2^1\,i]_i$ and then go to coarser and coarser levels $[2^k\,i - 1 + 2^{k-1}]_i$, $k = 2, 3, \ldots$ Here we chose the index offset $-1 + 2^{k-1}$ such that no strokeline appears twice for different values k.

The ordering in which the strokeline have to be drawn can easily be computed from their index: in the jth step the rev(j)th strokeline is drawn where rev(j) is the number which has the reverse binary representation of j, i.e., the sequence of binary digits is reversed.

8. Cross hatches

So far we explained the generation of hatches only along the maximum curvature direction. In some cases the image quality can be improved by also adding hatches in the cross direction since this increases the brightness range (darkest to lightest grey value). Cross hatches are often used to enhance the contrast at shadow boundaries.

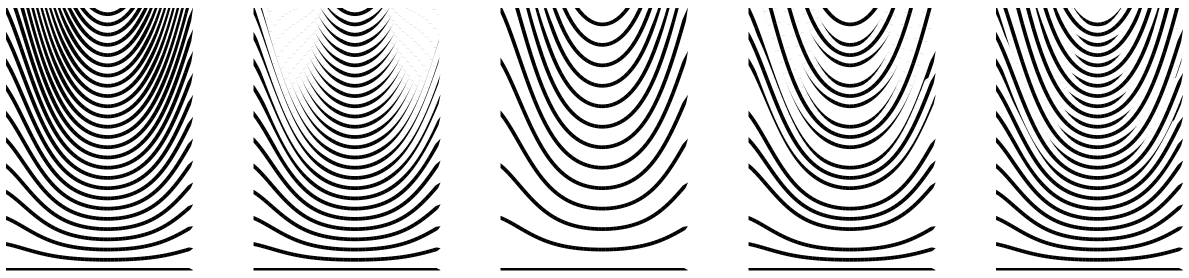

Figure 6. The special order in which the strokelines are drawn, guarantees that the surviving lines (the ones that are not overpainted) do not show a low-frequency pattern. Simply painting black strokes with constant width w does not lead to constant color (far left). Drawing the strokes with white bands and constant black fraction (as shown in Fig. 5) and in sequential top to bottom order leads to low-frequency patterns of suppressed lines (left). The next three images depict our special ordering. The center image shows the lines with index $2\,i$ ($k = 1$) only. In the center right image these lines are partially overpainted by the lines with index $4\,i + 1$ ($k = 2$). In the next step another layer of lines with index $8\,i + 3$ ($k = 3$) is painted (far right). The resulting image has an almost constant shading color which is achieved by suppressing some of the lines in regions where strokelines become denser.

Since the cross hatches follow the minimum curvature direction, they are typically less curved than the original hatches. As a consequence the effects of varying strokeline density are less severe and simple stroke width modulation (without strokeline suppression) is usually sufficient for the tone mapping.

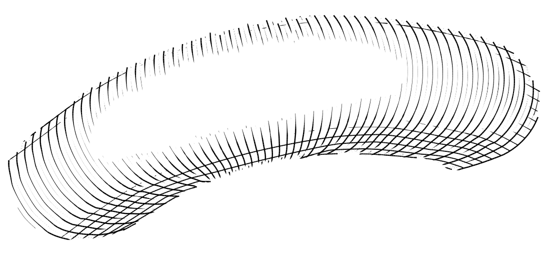

Figure 7. Cross hatches are used to enhance the contrast. Only the black portion of the strokes is drawn to avoid overpainting.

In our implementation we applied cross hatches in regions of the image where the shading value falls below a prescribed threshold $c_{\min}$ (cf. the range restriction in Section 7). Because one set of hatches has already been painted before the cross hatches are added, we base the stroke width computation on the offset grey values $c' = c - c_{\min}$. In order to avoid overpainting the already existing hatches, we only draw the black portion of the cross hatches (no white band, no strokeline suppression). As stated above this simple cross hatching technique works well because minimum curvature lines usually have a rather straight characteristic shape (Fig. 7).

9. Examples

Our examples show a technical model and a toy elk. The original triangle meshes consist of 43k and 30k triangles. The technical part (Fig. 8) nicely shows the suppression of strokes in the tone mapping process. Cross hatched strokes are used in dark areas. The elk model (Fig. 9) was generated from range scans of a wooden toy. The strokes at the sphere-shaped wheels have been generated manually as there are no meaningful principle directions defined there. The two images are identical except that the cross hatch threshold $c_{\min}$ was modified.

10. Conclusions

We presented a new technique for interactively generating line art drawings from triangulated models. The model geometry is sampled in 3D space and we can take advantage of the graphics hardware in this preprocessing step. All subsequent processing is done entirely in 2D which reduces the computation costs. The artistic part of image segmentation and the placing of some few reference streamlines is done manually while the strokes are generated automatically by

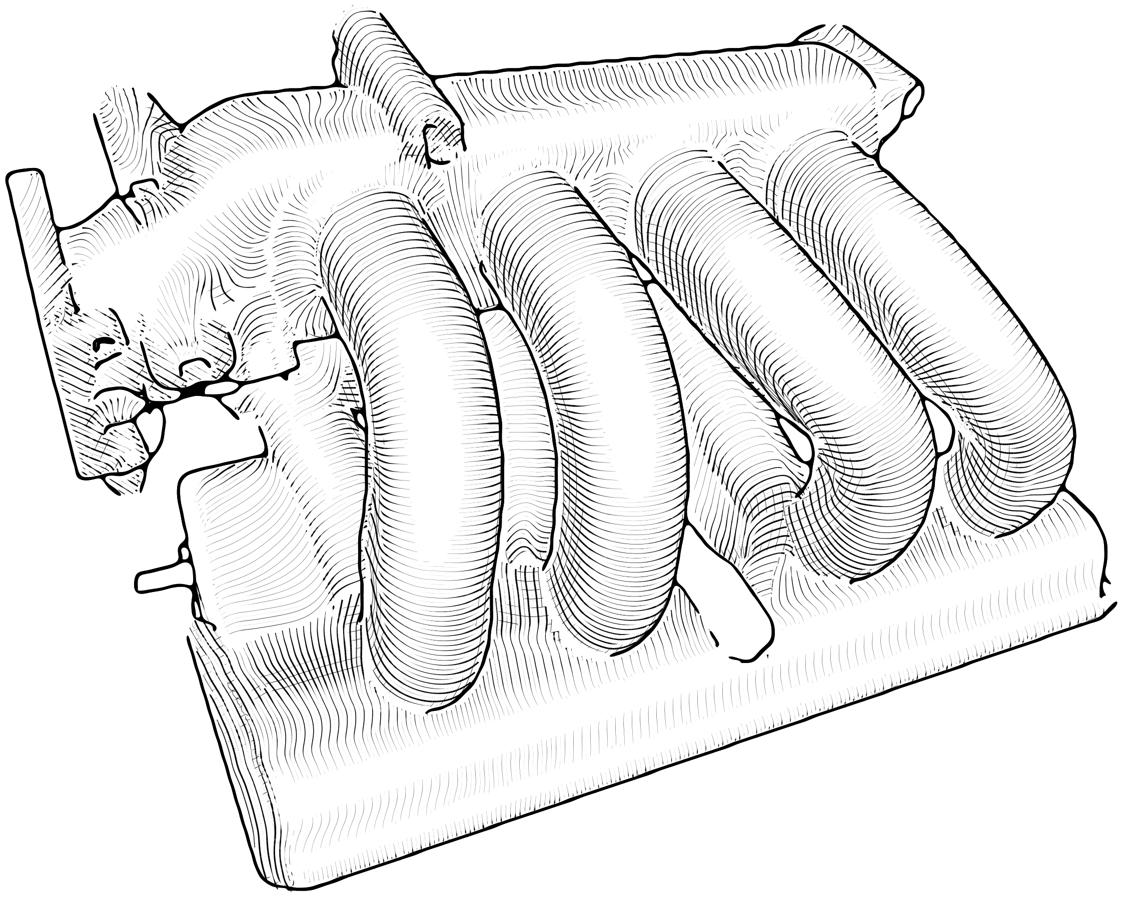

Figure 8. A technical part line art rendered with our system. The view was captured from a model of 43k triangles.

blending reference "key-ribs". Exploiting the special structure of our fishbone curves leads to a intuitive yet effective tone mapping algorithm.

Our algorithm processes unorganzied triangle meshes directly i.e. it does not assume a global parameterization of the surface in contrast to [5, 6, 7, 20]. This enables us to process a huge class of models since such meshes are the most popular surface representation in computer graphics. In addition, the geometric detail can easily be reduced by applying the discrete curvature analysis to a smoothed version of the given mesh thus allowing some kind of multi-resolution functionality.

The previously mentioned algorithms process 3D data that is already segmented into subsurfaces. We require manual work for image segmentation and placing of reference streamlines. Subsurface information – if available – can easily be exploited by performing an additional ID-rendering step into the enhanced frame buffer. Occlusion can also be handled by using the silhouettes as segment boundaries. For rather complex triangle meshes this can only serve as an initial segmentation that is to be refined manually by the artist.

Image based approaches like [16] also need manual segmentation. As we are operating in 2D after geometry sampling, the situation is similar to our approach. The main difference is that we can take great advantage of the sampled data instead of e.g. creating a direction field from scratch. So we focussed our work on the relatively "high-level" fishbone metaphor that

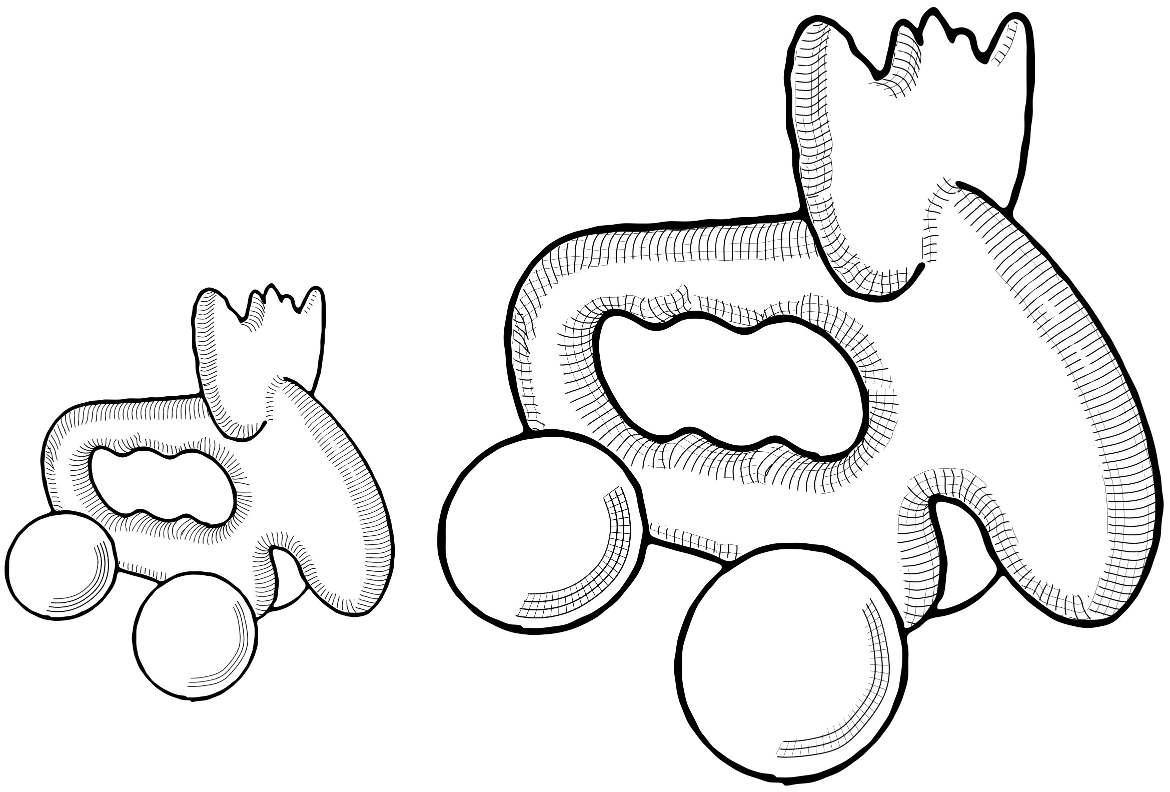

Figure 9. Mama elk and baby-elk (30k triangles) with and without cross hatching.

allows simple and efficient generation of smooth, quasi parallel strokes.

We do not employ predefined stroke textures as in [15, 16, 19, 20]. As a consequence our strokes only reproduce tone and not texture that emulates different surface materials as glass or wood. Nevertheless, our rendering style seems appropriate for technical objects with piecewise smooth surfaces.

Combining our technique with advanced pen-and-ink rendering methods such as texturing and a more sophisticated global tone mapping scheme would clearly be interesting for future work.

Acknowledgement

We would like to thank Kolja Kähler for generating the elk model.

References

[1] A. Appel, F. Rohlf, A. Stein. The Haloed Line Effekt for Hidden Line Elimination. *Computer Graphics (SIGGRAPH '79 Proceedings)*, 1979, pp. 151–157

[2] B. Carbal, L.C. Leedom. Imaging Vector Fields using Line Integral Convolution. *Computer Graphics (SIGGRAPH '93 Proceedings)*, 1993, S. 263–272

[3] O. Deussen, J. Hamel, A. Raab, S. Schlechtweg, T. Strothotte. An illustration technique using hardware-based intersections and skeletons. *Proc. Graphics Interface '99*, 1999, 175–182

[4] D. Dooley, M.F. Cohen. Automatic Illustration of 3D geometric models: Lines. *Computer Graphics*, Vol. 23, No. 2, 1990, pp. 77–82

[5] Gershon Elber. Line Art Rendering via a Coverage of Isoparametric Curves. *IEEE Transactions*

on Visualization and Computer Graphics, Vol. 1, No. 3, September 1995, pp. 231–239

[6] Gershon Elber. Line Art Illustrations of Parametric and Implicit Forms. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 4, No. 1, January-March 1998, pp. 1–11

[7] Gershon Elber. Interactive Line Art Rendering of Freeform-Surfaces. *Computer Graphics Forum (EUROGRAPHICS '99 Proceedings)*, 1999, pp. 1–12

[8] Leif Kobbelt. Discrete Fairing and Variational Subdivision for Freeform Surface Design. *The Visual Computer*, Vol. 16, 2000, pp. 142 – 158

[9] W. Leister. Computer generated Copper Plates. *Computer Graphics forum.* Vol. 13, No. 1, 1994, pp. 69–77

[10] W. Lorensen, H. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 1987, S. 163–169

[11] Victor Ostromoukhov. Digital Facial Engraving. *Computer Graphics (SIGGRAPH '99 Proceedings)*, 1999, pp. 417–424

[12] Yachin Pnueli, Alfred M. Bruckstein. **Dig**$^{\mathbf{i}}_{\mathrm{D}}$ürer – a digital engraving system. *The Visual Computer*, Vol. 10, 1994, pp. 277–292

[13] Christian Rössl, Leif Kobbelt, Hans-Peter Seidel. Line art rendering of triangulated surfaces using discrete lines of curvature. *WSCG '2000 Proceedings*, 2000, pp. 168–175

[14] Takafumi Saito, Tokiichiro Takahashi. Comprehensible Rendering of 3-D Shapes. *Computer Graphics (SIGGRAPH '90 Proceedings)*, 1990, pp. 197–206

[15] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel, David H. Salesin. Interactive pen-and-ink illustrations. *Computer Graphics (SIGGRAPH '94 Proceedings)*, 1994, pp. 101–108

[16] Michael P. Salisbury, Michael T. Wong, John F. Huges, David H. Salesin. Orientable Textures for Image-Based Pen-and-Ink Illustration. *Computer Graphics (SIGGRAPH '97 Proceedings)*, 1997, pp. 401–406

[17] Robert Schneider, Leif Kobbelt. Generating Fair Meshes with G^1 Boundary Conditions. *Geometric Modeling and Processing (GMP 2000 Proceedings)*, 2000, pp. 251–261

[18] William Welch, Andrew Witkin. Free-Form Shape Design Using Triangulated Surfaces. *Computer Graphics (SIGGRAPH '94 Proceedings)*, 1994, pp. 247–256

[19] Georges Winkenbach, David H. Salesin. Computer-generated Pen-and-Ink Illustration. *Computer Graphics (SIGGRAPH '94 Proceedings)*, 1994, pp. 91–98

[20] Georges Winkenbach, David H. Salesin. Rendering Parametric Surfaces in Pen-and-Ink. *Computer Graphics (SIGGRAPH '96 Proceedings)*, 1996, pp. 469–476

[21] Dennis Zorin, Aaron Hertzmann. Illustrating smooth surfaces. to appear in *Computer Graphics (SIGGRAPH '2000 Proceedings)*, 2000