LINE ART RENDERING OF TRIANGULATED SURFACES USING DISCRETE LINES OF CURVATURE

Christian Rössl **Leif Kobbelt** **Hans-Peter Seidel**

Computer Graphics Group
Max-Planck-Institut für Informatik
Im Stadtwald, 66123 Saarbrücken
Germany
{roessl,kobbelt,hpseidel}@mpi-sb.mpg.de

ABSTRACT

In recent years, several techniques have been proposed for automatically producing line-art illustrations. In this paper a new non photo-realistic rendering scheme for triangulated surfaces is presented. In contrast to prior approaches with parametric surfaces, there is no global parameterization for triangle meshes. So a new approach is made to automatically generate a direction field for the strokes. Discrete curvature analysis on such meshes allows to estimate differential parameters. Lines of curvature are then constructed to be used as strokes. Using triangulated surfaces allows to render aesthetically pleasing line drawings from a huge class of models. Besides, experiments show that even real time visualization is possible.

Keywords: non photo-realistic rendering, line art drawings, triangle meshes, discrete curvature analysis

1 INTRODUCTION

1.1 Line art rendering

Computer graphics usually focus on creating photo-realistic pictures of artificial scenes. However, there are numerous applications where abstract line-oriented drawings are preferred to photographs, especially for scientific or technical illustrations. There are a couple of reasons for this: information can be conveyed better by using some level of abstraction, line drawing enables creating sketches that appear less computer oriented and may be easier to be reproduced on black and white displays or printers.

Line rendering techniques have a long tradition in arts, e.g. pen-and-ink illustrations or copper plates with engraved lines, which used to be a common way to print illustrations in books. Over the last years computer based techniques for producing such illustrations have been developed. There are two basic approaches:

The first approach is image based and assists the user in converting a digital grey scale image into a pen-and-ink illustration. This way, the user is freed from drawing individual strokes. In [Salis94] predefined stroke textures are used to map the tone of the reference image, and in [Salis97] the strokes are generated automatically from both a set of reference strokes and an interactively modifiable direction field.

The second approach creates line art illustrations from 3D geometry. An early step in the development of this technique was the use of haloed lines [Appel79] which give an impression of depth. Different line styles such as line width, dashed or dotted lines can be employed for outline and shading [Doole90a, Doole90b]. [Leist94] discusses a ray tracing approach to emulate copper plates. Most recently, [Deuss99] takes advantage of the graphics hardware to achieve similar effects with sets of parallel cutting planes.

In order to produce high quality illustrations from polygonal models [Winke94] uses special stroke textures in addition to the geometry, e.g. bricks of a wall. The strokes are rendered with lines of varying thickness and shape (waviness) emulating the effect of manual drawing. Apart from that, an advanced shader is used, which e.g. generates shadows. After transforming the polygons from 3D to 2D image space a special tone mapping process determines the density of the strokes in 2D image space.

In this paper we will concentrate on line art rendering of 3D geometry that can do *without* any further information about the scene like texture. A direction field for the strokes is generated automatically from the geometry. In contrast to other approaches using parametric surfaces there is no global parameterization available for triangle meshes that can be utilized for producing strokes. Our results will show that rendering can be performed efficiently on such free-form geometry. The preceding survey is continued with focus on automatically generating strokes from 3D geometry.

1.2 Related work

[Elber95] uses a coverage of isoparametric curves of a free-from surface for line art rendering. Therefore strokes are defined as a parallel lines in the parameter domain resulting in isoparametric curves. This works especially well with surfaces of revolution. [Winke96] uses such isoparametric lines in order to produce high quality illustrations like in [Winke94]. Interestingly, for this approach the surface is tessellated to a polygonal mesh for the final tone mapping as well.

Most recently [Elber99] extends these techniques to enable interactive rendering with isoparametric curves, isophotes or lines of curvature on free-form surfaces. For rendering, these stroke curves are approximated by piecewise linear polygons. They are then evaluated up to a certain length determined by the shader. For shading, the surface normals at the seed points of the polygons are used. In order to meet real time demands, all possible strokes are precalculated. Furthermore, the number of polygons involved in the rendering process is effectively reduced by inserting the polygons into buckets. Geometrically, a bucket is a cone outgoing from the origin. The union of all buckets cover the unit sphere. Every bucket contains all normals falling into the corresponding cone. The shader first evaluates the "intensity" for the axis of a cone taken as a representative normal. This way polygons in a bucket only need to be evaluated if the expected intensity is above a certain limit.

Elber prefers strokes generated from isoparametric curves because they can be determined more efficiently than isophotes or lines of curvature. Furthermore they produce visually good results. This is impossible for triangulated surfaces lacking a global parameterization. We use lines of principle curvature which define a natural "flow" over the surface instead. Textures generated from principle curvature directions have recently been used by [Inter97] to visualize volume models.

1.3 Overview

In this paper we present an algorithm to render line-oriented sketches of triangulated surfaces without any (stroke) texture information. We trace lines into the direction of the maximum curvature and use them as strokes, expanding [Elber99] to be utilized for triangle meshes. Such meshes are a universal representation of surfaces and they become more and more popular for geometric modeling. This way our technique allows us to create aesthetically pleasing line art drawings for a wide class of geometrical models.

In order to utilize lines of curvature as strokes for line art rendering, an algorithm to construct these lines is needed. In contrast to parametric surfaces, there are no derivatives or curvatures defined for a piecewise linear triangle mesh. Therefore we use a method for calculating discrete curvature. Given principle curvature values and directions we can compute lines of curvature by integration.

For the rendering of the scene, such lines are uniformly scattered over the surface. Depending on the point of view and the lighting, the strokes are drawn with different lengths. This classification of strokes is part of the shading algorithm.

2 CONSTRUCTION OF LINES OF CURVATURE

Discrete curvature on triangulated surfaces is approximated to get values and directions of principle curvature in every vertex. Then lines of curvature can be integrated from the (discrete) field of directions according to the maximal curvature.

2.1 Approximation of discrete curvature

A triangle mesh is a piecewise linear rather than a smooth surface, so it is not clear how to calculate any derivatives on such a mesh. A common technique generalizes concepts from differential geometry of smooth surfaces. It fits simple geometric primitives e.g. second order surfaces (quadrics) to a vertex and its neighbors. The differential parameters can then be obtained from differentiating those well known primitives.

Our approach uses locally isometric divided difference operators which are derived by fitting a second order Taylor polynomial to a vertex and its neighbors. A locally isometric parameterization leads to simple linear operators since derivatives with respect to such a parameterization have a geometric interpretation. This way we can estimate geometric curvature.

We locally estimate the first and second fundamental form of the surface $F(u, v)$ in every vertex of its triangulation. Deriving surface curvatures like principal curvatures from the fundamental forms is straightforward. An introduction to the basic concepts of differential geometry can be found e.g. in [Farin96, Carmo76].

In this section V denotes the vertex for which the fundamental forms are to be approximated, V_i ($1 \leq i \leq n$, and for convenience $V_{n+1} := V_1$) are its neighbors. Q and Q_i denote the positions of V and V_i in 3D space. Without loss of generality the origin is shifted such that $Q := (0, 0, 0)$. For simplicity, we do not handle vertices on the boundary of the mesh.

2.1.1 Parameterization

As we are interested in curvatures, it is sufficient to estimate partial derivatives up to second order. For approximating the derivatives F_u, F_v, F_{uu}, F_{uv} and F_{vv} in a specific vertex V we need a locally isometric parameterization $F(u_i, v_i) = Q_i$ of its neighborhood with $F(0, 0) := (0, 0, 0) = Q$. A parameterization is *isometric* if $\|F_u\| \approx 1$, $\|F_v\| \approx 1$ and $F_u^\top F_v \approx 0$. The coefficients of the fundamental forms are completely defined by the derivatives up to second order.

The obvious way of constructing such a parameterization is to project the neighborhood of the vertex into a tangent plane at this vertex. The projection plane P is given by averaging the triangle normals around V resulting in the normal vector N_P. By transforming the projected points into an orthonormal basis $\{U_P, V_P, N_P\}$ we get a parameterization $F(u_{P,i}, v_{P,i}) = Q_i$. This projection method suffers from the fact that the ordering of neighbors around V is not necessarily preserved. The ordering can be destroyed if the triangle mesh is not sufficiently flat [Welch94].

An alternative parameterization considers the lengths and the angles between adjacent edges of the triangulated surface. The ordering of neighbors is preserved when using the exponential map [Carmo76]

$$\begin{aligned} &\exp(Q_i) \mapsto \\ &\|Q_i\| \left(\cos(\textstyle\sum_{j=1}^{i-1} \tilde{\alpha}_j), \sin(\textstyle\sum_{j=1}^{i-1} \tilde{\alpha}_j) \right) \end{aligned} \tag{1}$$

where $\tilde{\alpha}_i = \textbf{flat}(\angle(Q_i, Q_{i+1}))$ with $\sum_i \tilde{\alpha}_i = 2\pi$. Therefore **flat** scales the angles between two edges in 3D so that they sum to 2π in 2D. A possible definition for **flat** is

$$\textbf{flat}(\phi_i) = \phi_i \frac{2\pi}{\sum_j \phi_j} \tag{2}$$

This uniformly scales the 3D angle in a straightforward way and will work with any configuration of 3D angles. [Welch94]

2.1.2 Surface fitting

Given this isometric parameterization, the surface can locally be approximated by a quadratic Taylor polynomial

$$\begin{aligned} &F(u, v) = \\ &uF_u + vF_v + \tfrac{u^2}{2} F_{uu} + uvF_{uv} + \tfrac{v^2}{2} F_{vv} \end{aligned} \tag{3}$$

Recall that we shifted the origin and chose our parameterization so that $Q = (0, 0, 0) = F(0, 0)$. Fitting a second order surface to a vertex V and its neighbors is straightforward: By utilizing the parameterization $F(u_i, v_i) = Q_i$ we set up a system of n linear equations

$$\mathbf{VF} = \mathbf{Q} \tag{4}$$

with $\mathbf{V} = (u_i, v_i, \frac{u_i^2}{2}, u_i v_i, \frac{v_i^2}{2})_i$, $\mathbf{F} = (F_u, F_v, F_{uu}, F_{uv}, F_{vv})^\top$ and $\mathbf{Q} = (Q_i)_i^\top$. The (least squares resp. least norm) solution of this linear system is

$$\mathbf{F} = \begin{cases} \mathbf{V}^\top(\mathbf{V}\mathbf{V}^\top)^{-1}\mathbf{Q} & : \; n < 5 \\ \mathbf{V}^{-1}\mathbf{Q} & : \; n = 5 \\ (\mathbf{V}^\top\mathbf{V})^{-1}\mathbf{V}^\top\mathbf{Q} & : \; n > 5 \end{cases} \tag{5}$$

An alternative approach proposed in [Welch94] is to switch to another set of basis functions if the $\mathbf{V}$ matrix is ill-conditioned or $n < 5$.

The resulting vector $\mathbf{F}$ yields approximations of the Taylor coefficients F_u, F_v, F_{uu}, F_{uv}, F_{vv} of the surface. This enables us to estimate further differential parameters at the vertex V of the triangulation as needed.

As we used an isometric parameterization it turns out that the first fundamental form is the identity

$$\begin{pmatrix} F_u^\top F_u & F_u^\top F_v \\ F_u^\top F_v & F_v^\top F_v \end{pmatrix} \approx \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{6}$$

The following terms have been simplified by taking advantage of this fact.

2.2 Integration of lines of curvature

With the approximated first (identity) and second fundamental forms, principal curvature values and directions can be estimated for every vertex.

Let $N := (F_u \times F_v)/\|F_u \times F_v\|$ be the normal vector at a vertex V, thne we use the following notation for the second fundamental form:

$$\begin{pmatrix} l & m \\ m & n \end{pmatrix} := \begin{pmatrix} F_{uu}^\top N & F_{uv}^\top N \\ F_{uv}^\top N & F_{vv}^\top N \end{pmatrix} \tag{7}$$

The normal curvature for a tangent direction $\lambda := dv/du$ in the parameter domain is given by [Farin96]

$$\kappa(\lambda) = \frac{l + 2m\lambda + n\lambda^2}{1 + \lambda^2} \tag{8}$$

The normal curvature $\kappa(\lambda)$ has two extrema in the general case – the principal curvatures κ_1, κ_2. The corresponding principal directions are obtained as roots λ_1, λ_2 of a quadratic polynomial with

$$\lambda_{1,2} = \frac{(n - l) \mp \sqrt{(n-l)^2 + 4m}}{2m} \tag{9}$$

Since strokes should follow the maximum curvature $\kappa_{\max} = \max\{|\kappa_1|, |\kappa_2|\}$ we chose $\lambda_{\max} := \lambda_j$ with $|\kappa_j| = \kappa_{\max}$. By transforming these directions $\lambda_{\max}$ from the parameter domain to 3D space by

$$\vec{e} := (F_u + \lambda_{\max} F_v)/\|F_u + \lambda_{\max} F_v\| \tag{10}$$

we obtain a discrete vector field $i \mapsto \vec{e}_i$ defined for every vertex V_i of the mesh.

This field can be made continuous by interpolating directions across triangles. For a triangle $\triangle := \triangle(V_{i_1}, V_{i_2}, V_{i_3})$ we interpolate linearly using barycentric coordinates. A point $P_\triangle$ in the triangle plane can be expressed by

$$P_\triangle = \sum_{k=1}^{3} \gamma_k V_{i_k}, \quad \sum_{k=1}^{3} \gamma_k = 1 \tag{11}$$

Applying the same barycentric combination to $\vec{e_{i_1}}, \vec{e_{i_2}} \vec{e_{i_3}}$ yields the interpolated direction

$$\vec{e}(P_\triangle) = \gamma_1 \vec{e}_{i_1} + \gamma_2 \vec{e}_{i_2} + \gamma_3 \vec{e}_{i_3} \tag{12}$$

For convenience the resulting $\vec{e}(P_\triangle)$ are normalized to unit length. For triangles at the boundary of the mesh, only the directions of inner vertices are calculated and taken into account for interpolation. We do not handle triangles with three boundary vertices. Except for points in such triangles for any other point P of the surface the direction of the maximum curvature can be estimated.

In oder to trace one single line of curvature $F(k(t))$ starting from a point $P_0 := F(u_0, v_0)$ and streaming along $\vec{e}_{\max}$, we solve the differential equation

$$k'(t) = \vec{e}^{\,p}_{\max}(k(t)), \quad k(t_0) = (u_0, v_0) \tag{13}$$

Here $\vec{e}^{\,p}_{\max}$ is the projection of $\vec{e}_{\max}$ into the plane of the reference triangle $\triangle$. A simple Euler integrator has proven to be sufficient for our purpose. With a given step size h Eq. 13 is then discretized to

$$k_{n+1} = k_n + h\vec{e}^{\,p}_{\max}(k_n), \quad k_0 = P_0 \tag{14}$$

Note, that we do not need a global parameterization. Using the projected directions is necessary in order to guarantee that the resulting stream line does not leave the surface. In addition, we must detect when we leave the current triangle $\triangle$ and enter another one. The barycentric coordinates are used for the in-triangle test. If $\gamma_1, \gamma_2, \gamma_3 \geq 0$ for a point $P_\triangle$ then this point is situated inside $\triangle$. Otherwise a neighbor triangle is entered. Let $i' := i + 1 \mod 3$ and $i'' := i + 2 \mod 3$.

In case one coordinate γ_i is negative, the intersection T of the line segment $[F(k_n), F(k_{n+1})]$ with the opposite border $[V_{i'}, V_{i''}]$ of the triangle is calculated. If two coordinates $\gamma_{i'}, \gamma_{i''}$ are negative we have to determine both intersections T_1, T_2 with the two edges $[V_i, V_{i'}], [V_i, V_{i''}]$. The point $T = T_k$ closest to the starting point with $\|T_k - F(k_n)\| \to \min$ is the position, where the curve leaves the current triangle. It may happen that a *vertex* is (nearly) intersected, i.e. $T_1 \approx T_2$. The integration algorithm must be numerically stable in such that it guarantees to determine the correct triangle that will be entered next.

Figure 1 illustrates the two intersection cases. The point T is used as new vertex of the stream line polygon, thus $F(k_{n+1}) := T$. Furthermore the reference triangle is changed.

Integration is stopped after a maximum path length or when reaching the boundary of the triangle mesh or a triangle with three boundary vertices. Since there are no directions defined in its vertices, integration is stopped as well.

So far, we did not care about the orientation of the $\vec{e}_{\max,i}$ vectors. Integrating Eq. 13 only makes sense if these vectors are consistently oriented. As this is not globally possible, directions are oriented locally. Therefore an initial orientation is chosen at the seed point. Then every time the curve leaves one triangle, the directions at vertices of the newly entered triangle are oriented into the current direction of the curve. Thus, if $\vec{e}_{\text{cur}}$ denotes the current direction all $\vec{e}_i$ are flipped to $-\vec{e}_i$ if the angle $\angle(\vec{e}_{\text{cur}}, \vec{e}_i) > \pi$.

There are sophisticated methods for adjusting the step size h adaptively during numerical integration. We found that varying h according to the shape of current triangle is fast and gives satisfactory results. In our experiments the following heuristic has shown to work well. Let c be the circumference of the triangle and α_i the smallest angle between any two directions $\vec{e}_{\max,i}$. We then choose

$$h_\triangle = \frac{c}{2(1 + \sum_{j=1}^{3} \alpha_j)^2} \tag{15}$$

Of course we do not claim to guarantee any error bounds on the integration as this is not necessary for the reconstruction of the natural flow.

3 RENDERING

After scattering seed points uniformly over the surface, lines of curvature are drawn starting from these points. The shader then determines which parts of the lines are actually rendered.

3.1 Distribution of seed points

In order to cover the surface with lines of curvature we start with uniformly scattering a number of seed points. Drawing curves outgoing from these seed points we expect that the surface will be suitably covered. We provide a function that computes an appropriate number of randomly placed points per triangle.

We start with an approximate number n of points which are to be scattered over the whole surface. In addition a parameter m is chosen as the maximum number of points scattered across a single triangle. In general, m is chosen several magnitudes smaller than n. Now for every triangle $\triangle_i$ the following algorithm is iterated m times.

Chose a random number $r \in [0, 1]$. Let A_i be the area of $\triangle_i$ and $A := \sum_i A_i$ the total surface area. If $r > \frac{A_i}{A} \cdot \frac{n}{m}$ then a point is falling into $\triangle_i$. This point is determined by choosing barycentric coordinates with random $\gamma_1, \gamma_2 \in [0, 1]$ and $\gamma_3 = 1 - \gamma_1 - \gamma_2$.

By repeating this algorithm m times per triangle every triangle has a chance of getting at most m points. Especially for configurations with triangles of very different area the possibility of placing multiple points into a single triangle is important. The probability is chosen in a way that in fact approximately n points are spread over the surface.

For some models it might be appropriate to emphasize detail in distinct regions of the model. In order to achieve this the density of seed points is increased for that region. Regions of detail can be detected from the geometry of the model by thresholding the maximum curvature $\kappa_{\max}$. For triangles in such regions the probability of getting a seed point is increased by some factor. Of course, the generated distribution of points is not uniform anymore. Figure 3 shows such a non

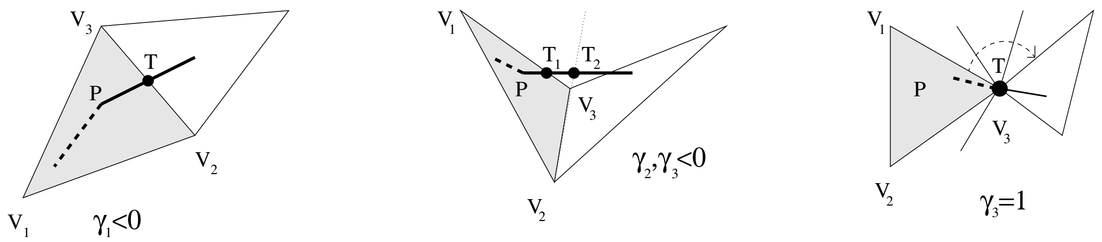

Figure 1: Barycentric coordinates $\gamma_1, \gamma_2, \gamma_3$ can be used for the in-triangle test. There are two cases how the traced path can leave the current triangle. Left: one coordinate γ_1 is negative. Middle: Two coordinates γ_2, γ_3 are negative. Then two intersection points have to be determined. Right: If two coordinates are negative and a vertex is (nearly) intersected then the integration algorithm must be able to find the next triangle.

uniform distribution. Regions of high curvature of the mannequin head are rendered with higher density of strokes.

3.2 Shading and rendering

The line art shader is restricted to black and white only. There are different approaches possible depending on the requested quality of the output. While [Winke94, Winke96] use sophisticated shading in conjunction with a special tone mapping process, our shader is relatively simple much like the one used in [Elber99]. Therefore real time applications are practicable. In general any advanced shader can be used. We are not limited to a particular shading resp. hidden-line removal algorithm.

A set of lines of curvature outgoing in both directions from uniformly spread seed points on the surface is precomputed as polygons. The maximum arc length $s_{\max}$ of these "candidate strokes" is limited to some useful value depending on the total area of the surface. The shader can then affect density and length of strokes to be rendered.

The arc length of strokes is derived from both an illumination and a silhouette intensity term. For illumination only diffuse light is considered. In addition to lighting, the silhouette of the surface should always be visible or be enhanced. For silhouette areas the surface normal is nearly perpendicular to the viewing direction. Further on, the boundary of the surface is defined to be part of the silhouette and thus considered extra. Otherwise the boundary is likely to be invisible since it cannot be respected otherwise.

For both the illumination and the silhouette intensity a truncation level is used so that short lines are not drawn. This prevents highlighted regions from being covered by small spots resulting from short curve segments. Such regions are to be rendered without any strokes instead. The total intensity is obtained as a convex combination of the diffuse and the silhouette intensity.

In order to determine intensities there must be a normal vector defined for every stroke. We use the normal vector of the triangle where the seed point of a stroke is located. This way intensities only have to be calculated once for every triangle that contains any seed points. As a result real time visualization is possible for reasonable sized models without any advanced data structure.

The density can be adjusted by randomly selecting only a fraction $\tau \in [0,1]$ of strokes for rendering. This fraction may vary with the distance of the surface to the viewer. This way a constant brightness level of the 2D image is preserved. This may be achieved with some simplified kind of tone mapping. An easier way is taking only into account the relative distance $d_{\min} \leq d \leq d_{\max}$. Different transfer functions $d \mapsto \tau$ can be used for this purpose. The simplest alternative allows the user to select the density and therefore the number of strokes rendered. This may be the method of choice to achieve real time rendering on a variety of graphics hardware.

Hidden surface removal is trivial if one is using a z-buffer. The scene is rendered twice, first the faces in background color, then the strokes. A common problem here is "z-fighting" caused by nearly equal z-values for surface and strokes, resulting in visual artifacts as dotted strokes. In order to avoid such artifacts the strokes are translated some small amount into z-direction resp. towards the viewer.

The graphics hardware may exploit back face culling for the triangles but not for the strokes. So an extra test in the rendering algorithm is worthwhile. Therefore strokes can be culled if $N_i V > \varepsilon$. Here $\varepsilon \in [0, 1]$ is some small positive constant, i.e. 0.2. As strokes originated at a back face may flow into front faces and thus may be visible, the usual test against 0 is inappropriate.

4 RESULTS

Figures 2-4 show line art renderings of three triangle meshes. Different shading parameters are used. The figures have been rendered by using the above z-buffer algorithm. Both, the synthetic model (cf. fig. 2) and the cat (cf. fig. 4) can be displayed interactively on a SGI O2 (R10000, 225MHz). A maximum of about 10k resp. 25k strokes are precalculated.

Although we do not utilize special data structures or algorithms like grouping strokes into buckets as in [Elber99], we are able to interact with reasonable sized models in real time. By assigning triangle normals to the strokes all strokes outgoing from the same triangle are treated equal. So shading is done per triangle and it can be restricted to those triangles that contain at least one seed point of a stroke.

For this reason the mannequin head (cf. fig. 3) with about nine times more triangles than the other models cannot be displayed interactively anymore. By combining Elber's algorithm with our triangle rendering substantial gains are to be expected for future work.

5 CONCLUSION

We presented a technique that allows to produce aesthetically pleasing line-drawings from triangle meshes by utilizing lines of curvature. The z-buffer rendering used can even display models of reasonable size in real time.

REFERENCES

[Appel79] Appel, A.; Rohlf, F.; Stein, A.: *The Haloed Line Effect for Hidden Line Elimination, Computer Graphics (Proc. SIGGRAPH '79)*, pp. 151-157, 1979

[Carmo76] do Carmo, M.P.: *Differential Geometry of Curves and Surfaces, Prentice Hall*, 1976

[Deuss99] Deussen, O.; Hamel, J.; Raab, A.; Schlechtweg, S.; Strothotte, T.: *An illustration technique using hardware-based intersections and skeletons, Proc. Graphics Interface '99*, pp. 175-182, 1999

[Doole90a] Dooley, D.; Cohen, M.F.: *Automatic Illustration of 3D geometric models: Lines, Computer Graphics*, Vol. 24, No. 2 ,pp. 77-82, 1990

[Doole90b] Dooley, D.; Cohen, M.F.: *Automatic Illustration of 3D geometric models: Surfaces, Proc. Visualization '90*, pp. 307-314, 1990

[Elber95] Elber, G.: *Line Art Rendering via a Coverage of Isoparametric Curves, IEEE Transactions on Visualization and Computer Graphics*, Vol. 1,No. 3, 1995

[Elber99] Elber, G.: *Interactive Line Art Rendering of Freeform Surfaces, Proc. EUROGRAPHICS '99*, Vol. 18, No. 3, pp. 1-12, 1999

[Farin96] Farin, G.: *Curves and Surfaces for Computer Aided Geometric Design. A Practical Guide, Academic Press*, 4. ed.; 1992

[Inter97] Interrante, V.L.: *Illustrating Surface Shape in Volume Data via Principal Direction-Driven 3D Line Integral Convolution, Computer Graphics (Proc. SIGGRAPH '97)*, pp. 109-116, 1997

[Leist94] Leister, W.: *Computer Generated Copper Plates, Computer Graphics Forum*, Vol. 13, No. 1, pp. 69-77, 1994

[Salis94] Salisbury, M.P.; Anderson, S.; Barzel, R.; Salesin, D.: *Interactive Pen-and-Ink Illustration, Computer Graphics (Proc. SIGGRAPH '94)*, pp. 101-108, 1994

[Salis97] Salisbury, M.P.; Wong, M.; Hughes, J.; Salesin, D.: *Orientable Textures for Image-Based Pen-and-Ink Illustration, Computer Graphics (Proc. SIGGRAPH '97)*, pp. 401-406, 1997

[Welch94] Welch, W.; Witkin, A.: *Free-Form Shape Design Using Triangulated Surfaces, Computer Graphics (Proc. SIGGRAPH '94s)*, pp. 247-256, 1994

[Winke94] Winkenbach, G.; Salesin, D.H.: *Computer Generated Pen-and-Ink Illustration, Computer Graphics (Proc. SIGGRAPH '94)*, pp. 91-98, 1994

[Winke96] Winkenbach, G.; Salesin, D.H.: *Rendering Parametric Surfaces in Pen and Ink, Computer Graphics (Proc. SIGGRAPH '96)*, pp. 469-467, 1996

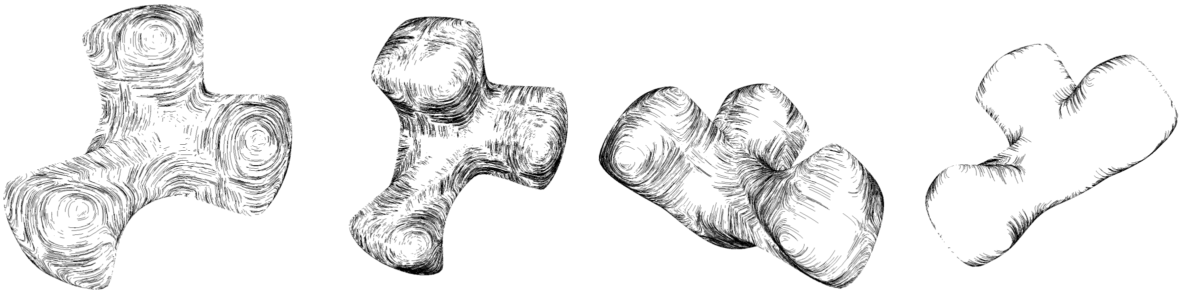

Figure 2: A synthetic model (9 k△) rendered with different shading parameters. The rightmost figure shows the silhouette.

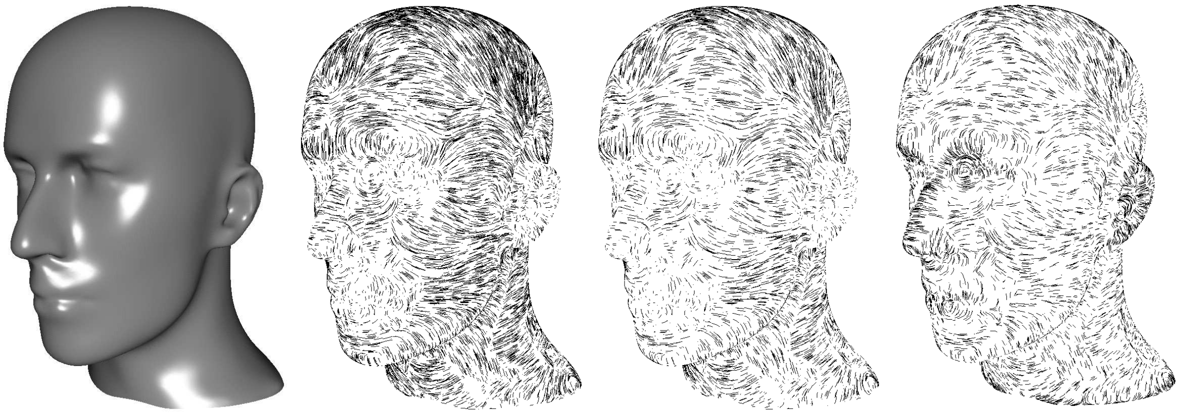

Figure 3: From left: original model of the mannequin head (87 k△); line art illustration; same but using fewer strokes; without shading but enhanced stroke density for regions of high curvature e.g. nose, eyes, ear.

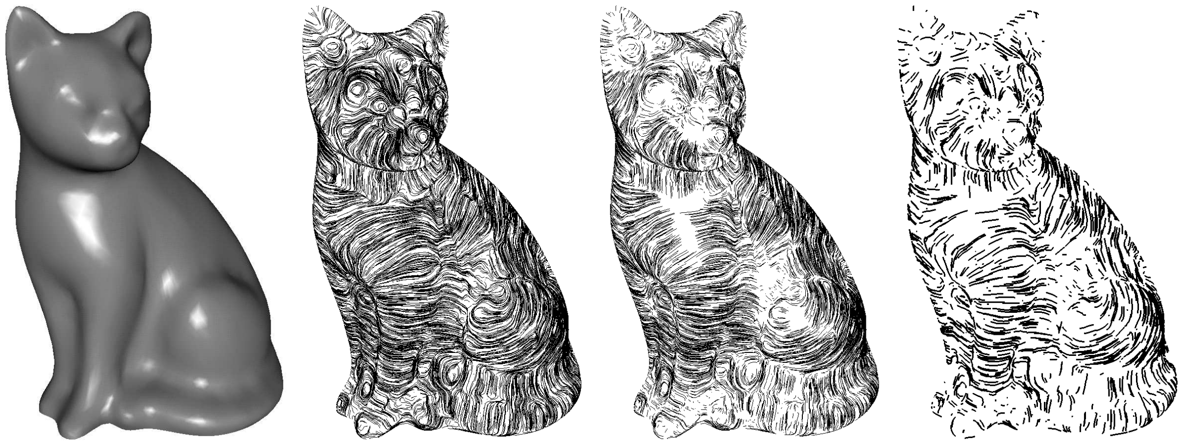

Figure 4: From left: original model of the cat (11 k△); all generated strokes; with shading considered; same but using fewer and thicker strokes.