Illustrating smooth surfaces

Aaron Hertzmann Denis Zorin

New York University

Abstract

We present a new set of algorithms for line-art rendering of smooth surfaces. We introduce an efficient, deterministic algorithm for finding silhouettes based on geometric duality, and an algorithm for segmenting the silhouette curves into smooth parts with constant visibility. These methods can be used to find all silhouettes in real time in software. We present an automatic method for generating hatch marks in order to convey surface shape. We demonstrate these algorithms with a drawing style inspired by *A Topological Picturebook* by G. Francis.

CR Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation– *Display algorithms.*

Additional Keywords: Non-photorealistic rendering, silhouettes, pen-and-ink illustration, hatching, direction fields.

1 Introduction

Line art is one of the most common illustration styles. Line drawing styles can be found in many contexts, such as cartoons, technical illustration, architectural design and medical atlases. These drawings often communicate information more efficiently and precisely than photographs. Line art is easy to reproduce, compresses well and, if represented in vector form, is resolution-independent.

Many different styles of line art exist; the unifying feature of these styles is that the images are constructed from uniformly colored lines. The simplest is the style of silhouette drawing, which consists only of silhouettes and images of sharp creases and object boundaries. This style is often sufficient in engineering and architectural contexts, where most shapes are constructed out of simple geometric components, such as boxes, spheres and cylinders. This style of rendering captures only geometry and completely ignores texture, lighting and shadows. On the other end of the spectrum is the pen-and-ink illustration style. In pen-and-ink illustrations, variable-density hatching and complex hatch patterns convey information about shape, texture and lighting. While silhouette drawing is sufficient to convey information about simple objects, it is often insufficient for depicting objects that are complex or free-form. From many points of view, a smooth object may have no visible silhouette lines, aside from the outer silhouette (Figure 8), and all the information inside the silhouette is lost. In these cases, can be added to indicate the shape of the surface.

The primary goal of our work was to develop rendering techniques for automatic generation of line-art illustrations of piecewise-smooth free-form surfaces. When using conventional photorealistic rendering techniques (e.g. Z-buffer or ray tracing)

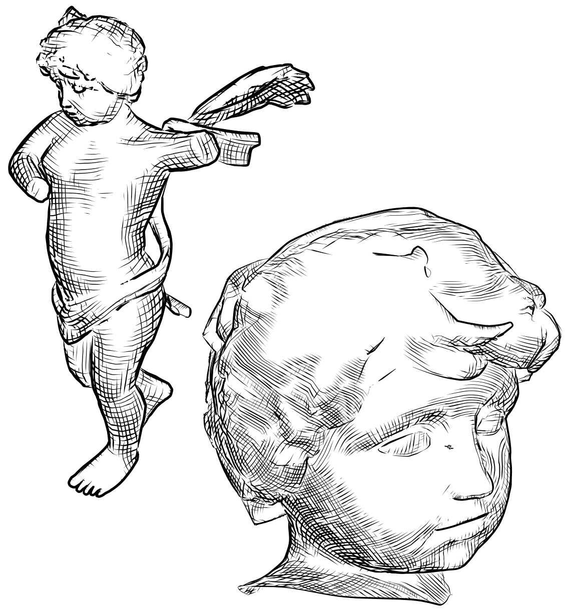


Figure 1: Illustrations of the Cupid mesh.

one can typically replace a smooth surface with a polygonal approximation, and thus reduce the problem to that of rendering polygonal meshes. This no longer true when our goal is to generate line drawings. Some differential quantities associated with the smooth surface must be recovered in order to generate visually pleasing hatch directions and topologically correct silhouette lines. Some of the problems that occur when a smooth surface is replaced by its polygonal approximation are discussed in greater detail in Section 4.

In this paper we address two general problems: computing silhouette curves of smooth surfaces, and generating smooth direction fields on surfaces that are suitable for hatching. The algorithms that we have developed can be used to implement a number of non-photorealistic rendering techniques. Our main focus is on a particular rendering style, which aims to communicate all essential information about the shape of the surface with a limited amount of hatching.

Contributions. *Algorithms.* To support rendering of smooth surfaces, we have developed a number of novel algorithms including:

- An efficient, deterministic algorithm for detecting silhouettes; (Section 4.3). In addition to non-photorealistic applications, this method can be used to accelerate computation of shadow volumes.
- An algorithm for cusp detection and segmentation of silhouette curves into smooth parts with constant visibility (Section 4.2).
- An algorithm for computing smooth direction fields on surfaces, suitable for use in hatching (Section 5). These fields have a wide range of uses, ranging from high-quality pen-and-ink rendering to interactive illustration and hatching.

An important feature of our approach is that any polygonal mesh can serve as input; the smooth surface that we render is inferred from the mesh. We do not assume an explicitly specified parameterization, which make our approach more general than previously developed techniques.

Rendering style. We have developed a new non-photorealistic rendering style based on the techniques of Francis [15], and influenced by the cartoons of Thomas Nast [34] and others.

The rules for drawing in this style are described in Section 6.

2 Previous Work

The methods used in nonphotorealistic rendering can be separated into two groups: image-space and object-space. The image-based approach is general and simple; however, it is not particularly suitable for generating concise line drawings of untextured smooth surfaces. Image-based techniques are presented in [5, 30, 7, 18, 6, 28]; these algorithms exploit graphics hardware to produce image precision silhouette images. Our technique is an object space method; it directly uses the 3D representation of objects, rather than their images. Winkenbach and Salesin [36] describe a method for producing appealing pen-and-ink renderings of smooth surfaces. Parametric lines on NURBS patches were used to determine the hatch directions and silhouette lines were computed using polyhedral approximation to the surface. Their main technical focus is on using the hatch density to render complex texture and lighting effects. Their system relied on a surface parameterization to produce hatch directions; however, such a parameterization does not exist for many types of surfaces, and can often be a poor indicator of shape when it does exist. Elber [12, 13] and Interrante [21] used principal curvature directions for hatching. Curvatures generally provide good hatch directions, but cannot be reliably or uniquely computed at many points on a surface. Our system makes use of the principle curvature directions, and uses an optimization technique to “fill in” the hatching field where it is poorly-defined. Deussen et al. [9] use intersections of the surfaces with planes; while being quite flexible, this approach requires segmentation of the surface into parts, where different groups of planes are used; the plane orientations computed using skeletons relate only indirectly to the local surface properties.

Our work also draws on techniques developed for vector field visualization [8, 22]. It should be noted that relatively little work has been done on generating fields on surfaces as opposed to visualization of existing fields. Elber [12, 13] discuses the relative merits of some commonly-used hatching fields (principle curvature directions, field of tangents to the isoparametric lines, the gradient field of the brightness).

Silhouette detection is an important component of many nonphotorealistic rendering systems. Markosian et al. [25] presented a randomized algorithm for locating silhouettes; this system is fast but does not guarantee that all silhouettes will be found. Gooch et al. [18] and Benichou and Elber [3] proposed the use of a Gauss map to efficiently locate all object silhouettes under orthographic projection. In this paper, we present a new method for silhouette detection that is fast, deterministic, and applicable to both orthographic and perspective projection.

Our method for computing the silhouette lines of free-form surfaces is closely related to the work of [14, 17] in computing silhouettes for NURBS surfaces.

3 Overview

In this section we present a general overview of our algorithms.

Surface representation. The input data for our system is a polygonal mesh that approximates a smooth surface. Polygonal meshes remain the most common and flexible form for approximating surfaces. However, information about differential quantities (normals, curvatures, etc.) associated with the original surface is lost. We need a way to estimate these quantities and compute, if necessary, finer approximations to the original smooth surface. This can be done if we choose a method that allows us to construct a smooth surface from an approximating arbitrary polygonal mesh, and easily compute the associated differential quantities (normals, curvatures, etc.).

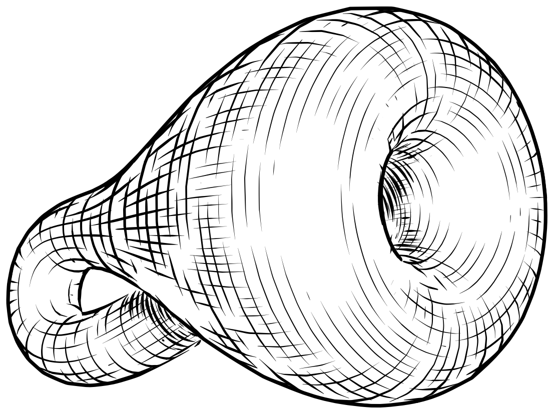


Figure 2: Klein bottle. Lighting and hatch directions are chosen to convey surface shape. Undercuts and Mach bands near the hole and the self-intersection enhance contrast.

We use piecewise-smooth subdivision, similar to the algorithms presented in [20], with an important modification (Appendix A) to make the curvature well-defined and nonzero at extraordinary vertices. However, other ways of defining smooth surfaces based on polygonal meshes can be used, provided that all the necessary quantities can be computed.

Algorithms. Our rendering technique has three main stages: computation of a direction field on the surface, computation of the silhouette lines and generation of hatch lines.

Hatch direction field. This stage defines a view-independent field on the surface that can be used later to generate hatches. Rather than defining two separate directional fields, we define a single *cross field* (Section 5) for hatches and cross-hatches. The main steps of our algorithm are: smooth the surface if necessary; compute an initial approximation to the field in areas of the surface where it is well defined, initialize the directions arbitrarily elsewhere; optimize the directions in places where the cross field was not well defined.

Silhouette curve computation. We compute the curves in several steps (Section 4): compute boundary, self-intersection and crease curves, as well as boundaries of flat areas; compute silhouette curves as zero-crossings of the dot product of the normal with the view direction; find cusps, determine visibility, and segment the silhouette curves into smooth pieces.

Hatch generation. Our hatch generation algorithms follow some of the rules described by Francis [15] (Section 6). The surface is divided into four levels of brightness with corresponding levels of hatching: highlights and Mach bands (no hatching), midtones (single hatching), shadowed regions (cross-hatching), and undercuts (dense cross-hatching). Line thickness varies within each region according to the lighting. Undercuts and Mach bands are used to increase contrast where objects overlap. Lights are placed at the view position or to the side of the object. The hatching algorithm covers all hatch regions with cross-hatches, then removes hatches from the single hatch regions as necessary.

4 Computing Silhouette Drawings

In this section we describe algorithms for generating the simplest line drawings of smooth surfaces, which we call silhouette drawings. A silhouette drawing includes only the images of the most visually important curves on the surface: boundaries, creases, silhouette lines and self-intersection lines. Finding intersections of smooth surfaces is a complex problem, which we do not address in the paper. We find self-intersections of a mesh approximating the surface and assume that self-intersection lines of the mesh approximate the self-intersection curves of the surface sufficiently well. Boundary curves and creases are explicitly represented in the surface; thus, we focus our attention on the problem of computing the silhouette lines. We will refer to the creases, boundaries and self-intersection curves as *feature curves*.

Before proceeding, we recall several definitions[1]. First, we define more precisely what we mean by a piecewise-smooth surface. A piecewise-smooth surface can be thought of as a finite union of a number of smooth surfaces with boundaries. A smooth embedded surface is a subset M of $\mathbf{R}^3$ such that for any point $\mathbf{p}$ of this subset there is a neighborhood $U(\mathbf{p}) = \mathrm{Ball}_\epsilon(\mathbf{p}) \cap M$ and a C^1-continuous nondegenerate one-to-one map $\mathbf{F}(u, v)$ from a domain D in $\mathbf{R}^2$ onto $U(\mathbf{p})$. The domain D can be taken to be an open disk for interior points, and a half-disk (including the diameter, but excluding the circular boundary) for smooth boundary points. It follows from the definition that the normal $\mathbf{F}_u \times \mathbf{F}_v$ is defined and is nonzero everywhere on the surface. The direction of $\mathbf{F}_u \times \mathbf{F}_v$ at any point of the surface is independent, up to a sign, of the local parameterization $\mathbf{F}$ and is denoted $\mathbf{n}(\mathbf{p})$.

The *silhouette set* for the smooth surface is the set of points $\mathbf{p}$ of the surface such that $(\mathbf{n}(\mathbf{p}) \cdot (\mathbf{p} - \mathbf{c})) = 0$, where $\mathbf{c}$ is the viewpoint. The silhouette is in general a union of flat areas on the surface, curves and points. We isolate flat areas and consider them separately. Isolated silhouette points are unstable, and are not relevant for our purposes. For a surface that does not contain flat areas and is C^2, the silhouette for a general position of the viewpoint can be shown to consist of C^1 non-intersecting curves (silhouette curves).

An important role in our constructions is played by the *curvature* of the surface. More specifically, we are interested in principal curvatures and principal curvature directions. The two principal curvatures at a point $\mathbf{p}$ are maximal and minimal curvatures of the curves obtained by intersecting the surface with a plane passing through $\mathbf{p}$ and containing the normal to the surface. The principal curvature directions are the tangents to the curves for which the maximum and minimum are obtained; these directions are always orthogonal and lie in the tangent plane to the surface. The formulas expressing these quantities in terms of the derivatives of $\mathbf{F}$ are standard and can be found, for example, in [4]. The most important property of the principal curvatures that we use can be formulated as follows: *if a surface has principal curvatures κ_1 and κ_2, and the unit vectors along principal directions and the normal are used to define an orthonormal coordinate system (r, s, t), with r and s parameterizing the tangent plane then locally the surface is the graph of a function over the tangent plane*

$$t = \kappa_1 r^2 + \kappa_2 s^2 + o(r^2 + s^2) \tag{1}$$

It follows that principal curvatures and principal curvature directions locally define the best approximating quadratic surface.

4.1 Silhouettes of Meshes and Smooth Surfaces

The simplest approach to computing the silhouette curves would be to replace the smooth surface with its triangulation and find the silhouette edges of the triangular mesh. However, there are significant differences between the silhouettes of smooth surfaces and their approximating polygonal meshes (Figure 4). For polygonal meshes, complex cusps (Figure 3), where several silhouette chains meet, are stable, that is, do not disappear when the viewpoint is perturbed. Singularities of projections of polyhedra were studied in considerable detail (see a recent paper [2] for pointers); a simple classification in the two-dimensional case, which does not appear to be explicitly described elsewhere, can be found in [1]. For smooth surfaces, the only type of stable singularity is a simple cusp, as it was shown in the classic paper by Whitney [35]. As a consequence, silhouette curves on smooth surfaces are either closed loops, or start and end on feature lines, while on polygonal surfaces they may intersect, and their topology is more complex. Moreover, we observe that *no matter how fine the triangulation is, the topology of the silhouette of a polygonal approximation to the surface is likely to be significantly different from that of the smooth surface itself.* This does not present a problem for fixed-resolution images: if the distance between the projected silhouette of the mesh and the projected silhouette of the smooth surface is less than a pixel, the topological details cannot be distinguished. However, if we do want to generate resolution-independent images capturing the essential features of the silhouette of the smooth surface correctly, or apply line styles to the silhouette curves, the polygonal approximation cannot be used. (Figure 4). Similar observations were made in [5].

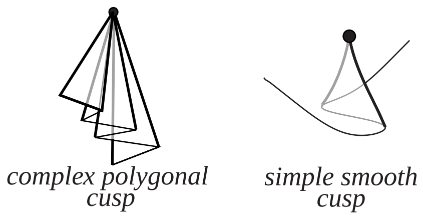


Figure 3: *Left:* Complex cusps are stable on polygonal meshes. *Right:* Only simple cusps are stable on smooth surfaces.

To preserve the essential topological properties of silhouettes, we compute the silhouette curves using an approach similar to the one used in [14, 17] for spline surfaces. Recall that the silhouette set of a surface is the zero set of the function $g(\mathbf{p}) = (\mathbf{n}(\mathbf{p}) \cdot (\mathbf{p} - \mathbf{c}))$ defined on the surface. The idea is to compute an approximation to this function and find its zero set. For each vertex $\mathbf{p}$ of the polygonal approximation, we compute the true surface normal and $g(\mathbf{p})$ at the vertex. Then the approximation to the function $g(\mathbf{p})$ is defined by linear interpolation of the values of the function. As the resulting function is piecewise-linear, the zero set will consist of line segments inside each triangle of the polygonal approximation. Moreover, we can easily enforce the general position assumption by picking arbitrarily the sign of the function $g(\mathbf{p})$ at vertices where it happens to be exactly zero. As a result, the line segments of the zero set connect points in the interior of the edges of the mesh, and form either closed loops or non-intersecting chains connecting points on the feature lines (Figure 4), similar in structure to the actual silhouette curves. We may miss narrow areas on the surface where the sign is different from surrounding areas. It is easy to see, however, that the silhouette curves we obtain by our method will have the same topology as the silhouette curves of some surface obtained by a small perturbation of the original. This means that we are guaranteed to have a plausible image of a surface, but it may not accurately reflect features of size on the order of the size of a triangle of the approximating mesh in some cases. The silhouette algorithm is described in greater detail in [19].

4.2 Cusp Detection

While the silhouette curves on the surface do not have singularities in a general position, the projected silhouette curves in the image plane do; there is a single stable singularity type, aside from terminating points at feature lines: a simple cusp (Figure 3). The most straightforward way to detect these singularities is to examine the

[1] We do not state rigorous mathematical definitions in complete detail; an interested reader can find them in most standard differential geometry texts.

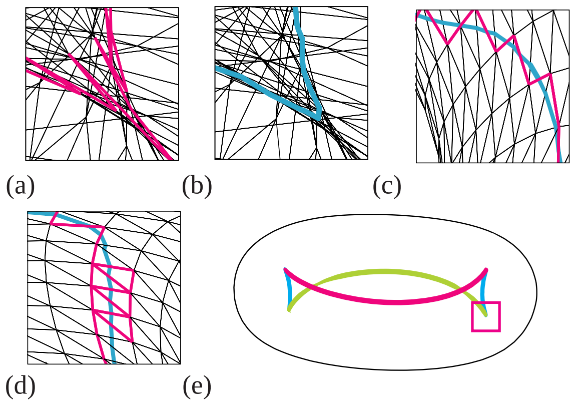


Figure 4: (a) Silhouette edges of a polygonal approximation produce jagged silhouette curves. (b) Our method produces smooth silhouette curves by inferring information about the smooth surface from the polygonal mesh. (c) The same curves shown from another viewpoint and overlayed. (d) A complex cusp occurs in the polygonal approximation when the surface is nearly parallel to the view direction. This does not occur in the smooth silhouette curve. (e) Smooth line drawing of the "smiling torus." The red box shows the location of the curves in (a)-(c).

tangents of the silhouette curves; cusps are the points where the tangent is parallel to the view direction. However, this approach is not numerically reliable, especially if the silhouette curves are approximated by polylines. We propose a new, numerically more robust way to find the cusps, using the following geometric observations.

Consider a silhouette point $\mathbf{p}$ with principal curvature directions $\mathbf{w}_1$ and $\mathbf{w}_2$ and principal curvatures κ_1 and κ_2. Let $\mathbf{c}$ be the viewpoint; since $\mathbf{p}$ is the silhouette point, the viewing direction $\mathbf{v} = \mathbf{c} - \mathbf{p}$ is in the tangent plane. Let $[c_1, c_2, 0]$ be the components of $\mathbf{c}$ with respect to the coordinates (r, s, t) associated with the principal curvature directions, computed by $c_1 = (\mathbf{v} \cdot \mathbf{w}_1)$ and $c_2 = (\mathbf{v} \cdot \mathbf{w}_2)$. As we have observed, $\mathbf{p}$ is a cusp when the tangent to the silhouette at $\mathbf{p}$ is parallel to the viewing direction $\mathbf{v}$. The tangent to the silhouette can easily be expressed in terms of curvature. Approximation (1) yields the following approximation to the normals in a small neighborhood near $\mathbf{p}$: $\mathbf{n}(r, s) = [-2\kappa_1 r, -2\kappa_2 s, 1]$. The equation of the 2nd order approximation to the silhouette curve is an implicit quadratic equation, $g(r, s) = (\mathbf{n}(r, s) \cdot \mathbf{v}(r, s)) = 0$, where $\mathbf{v}(r, s)$ is the viewing direction $\mathbf{c} - \mathbf{p}(r, s) = [c_1 - r, c_2 - s, -\kappa_1 r^2 - \kappa_2 s^2]$. We calculate the vector perpendicular to the silhouette at $\mathbf{p}$ as $\nabla g(0, 0) = [-2\kappa_1 c_1, -2\kappa_2 c_2]$. The resulting condition for the viewing direction to be parallel to the silhouette tangent (or, equivalently, perpendicular to $\nabla g(0, 0)$) to the viewing direction is $\kappa_1 c_1^2 + \kappa_2 c_2^2 = 0$. Therefore, we can define a parameterization-independent scalar function on the surface which we call the *cusp function*:

$$C(\mathbf{p}) = \kappa_1 (\mathbf{v} \cdot \mathbf{w}_1)^2 + \kappa_2 (\mathbf{v} \cdot \mathbf{w}_2)^2$$

where all quantities are evaluated at point $\mathbf{p}$. This function has the following important property: *cusps are contained in the intersection set of the two families of curves: one obtained as the zero set of the function $g(\mathbf{p})$, the other as the zero set of the cusp function $C(\mathbf{p})$* (Figure 5). The zero set of $C(\mathbf{p})$ can be approximated in the same way as the zero set of $g(\mathbf{p})$; each triangle of the polygonal mesh may contain a single line segment approximating the zero set of $C(\mathbf{p})$ and another approximating the zero set of $g(\mathbf{p})$. This allows us to compute approximate cusp locations robustly, without introducing many spurious cusps, and at the same time using relatively coarse polygonal approximations to the smooth surface.

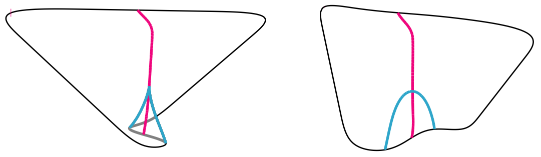


Figure 5: *Left:* Cusps are found as intersections of zero sets of two functions defined on the surface, the dot product of the normal with the viewing direction and the cusp function. The silhouette curve is shown in blue, the cusp zero set in red. *Right:* The same curves; view from a viewpoint different from the one that was used to compute the curves.

4.3 Fast Silhouette Detection

In the previous section, we have presented an algorithm for constructing approximations to the silhouette curves which, when implemented in the simplest way, requires complete traversal of the mesh. Such a traversal is unnecessary; typically, only a small percentage of mesh faces contain silhouettes [25, 23]. For polygonal meshes, a number of fast techniques were developed that allow one to avoid complete traversal. A stochastic algorithm was proposed in [25]. A deterministic algorithm based on the Gauss map was proposed in [3, 18], but is restricted to orthographic projection. We present a new deterministic algorithm for accelerated location of silhouettes, which works for both orthographic and perspective projection. This algorithm is equally suitable for finding silhouettes defined as zero sets, and for finding silhouette edges of polygonal meshes.

Our algorithm is based on the concept of *dual surfaces.* The points of the dual surface M' are the images of the tangent planes to a surface M under a duality map, which maps each plane $Ax + By + Cz + D = 0$ to the homogeneous point $[A, B, C, D]$. More explicitly, M' can be obtained by mapping each point of M to a homogeneous point $\mathbf{N} = [n_1, n_2, n_3, -(\mathbf{p} \cdot \mathbf{n})]$, where $\mathbf{n} = [n_1, n_2, n_3, 0]$ is the unit normal at $\mathbf{p}$. Note that the inverse is also true: each plane in the dual space corresponds to a point in the primal space. Let $\mathbf{C} = [c_1, c_2, c_3, c_4]$ be our viewpoint in the homogeneous form. Then the silhouette of the surface consists of all points $\mathbf{p}$ for which $\mathbf{C}$ is in the tangent plane at that point. For perspective projection, this means that $(\mathbf{C} \cdot \mathbf{N}) = (\mathbf{c} - \mathbf{p}) \cdot \mathbf{n} = 0$. For orthographic projection, the homogeneous formula is the same: $(\mathbf{C} \cdot \mathbf{N}) = (\mathbf{c} \cdot \mathbf{n}) = 0$, where $\mathbf{c}$ is interpreted as the view direction. Our algorithm is based on the following observation: *the image of the silhouette set of the surface with respect to the viewpoint $\mathbf{C}$ under the duality map is the intersection of the plane $(\mathbf{C} \cdot \mathbf{x}) = 0$, with the dual surface.* This fact allows us to reduce the problem of finding the silhouette to the problem of intersecting a plane with a surface (Figure 7), for which many space-partition-based acceleration techniques are available. However, an additional complication is introduced by the fact that some points of the dual surface may be at infinity. This does not allow us to consider only the finite part of the projective space, which can be identified with $\mathbf{R}^3$. However we can identify the whole 3D projective space with points of the unit hypersphere S^3, or, equivalently, of the boundary of a hypercube, in four-dimensional space. As four-dimensional space is somewhat difficult to visualize, we show the idea of the algorithm on a 2D example in Figure 6. In the 2D case, the problem is to compute all *silhouette points* on a curve, that is, the points for which the tangent line contains the viewpoint.

While the geometric background is somewhat abstract, the actual algorithm is quite simple. The input to the algorithm is a polygonal mesh, with normals specified at vertices, if we are computing silhouettes using zero-crossings. The normals are not necessary if we are locating the silhouette edges of the polygonal mesh. There are two parts to the algorithm: initialization of the spatial partition and

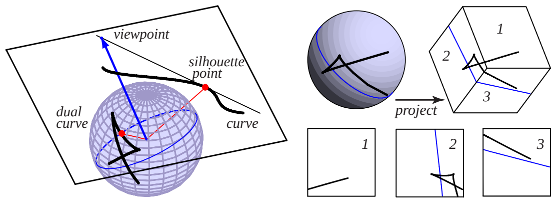


Figure 6: *Left*: Using a dual curve to find silhouette points. The figure shows a curve in the plane $z = 1$ and its dual on a sphere. The blue arrow is the vector c from the origin in 3D to the viewpoint in the plane, the blue circle is the intersection of the plane passing through the origin perpendicular to c with the unit sphere. The red points are a silhouette point and its dual. The silhouette point can be found by intersecting the blue circle with the dual curve and retrieving corresponding point on the original curve. *Right*: Reducing the intersection problem to planar subproblems. The upper hemisphere containing the dual curve is projected on the surface of cube and at most 5 (in this case 3) planar curve-line intersection problems are solved on the faces.

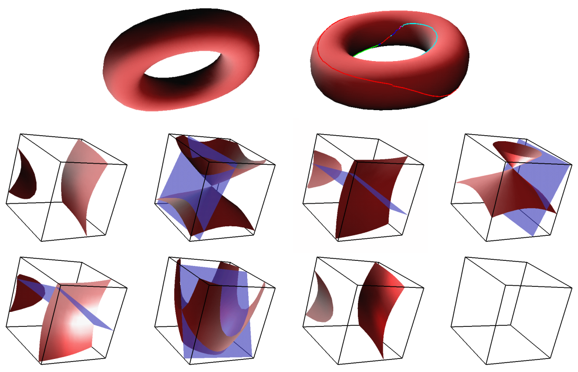


Figure 7: Silhouette lines under the duality map correspond to the intersection curve of a plane with the dual surface. *Top:* Torus shown from camera and side views. *Bottom:* The eight 3D faces of the hypercube, seven of which contain portions of the dual surface. The viewpoint dual is shown as a blue plane. Silhouettes occur at the intersection of the dual plane with the dual surface.

intersection of the dual surface with the plane corresponding to the viewpoint. The second part is fairly standard, so we focus on the first part.

Step 1: For each vertex $\mathbf{p}$ with normal $\mathbf{n}$, we compute the dual position $\mathbf{N} = [n_1, n_2, n_3, -(\mathbf{p} \cdot \mathbf{n})]$. The dual positions define the dual mesh which has different vertex positions but the same connectivity.

Step 2: Normalize each dual position $\mathbf{N}$ using l_∞-norm, that is, divide by $\max(|N_1|, |N_2|, |N_3|, |N_4|)$. After division, at least one of the components N_i, $i = 1..4$, becomes 1 or -1. The resulting four-dimensional point is on the surface of the unit hypercube. The three-dimensional face of the cube on which the vertex is located is determined by the index and sign of the maximal component.

Step 3: Each triangle of the dual mesh is assigned to a list for every three-dimensional face in which it has a vertex.

Step 4: An octtree is constructed for each three-dimensional face, and the triangles assigned to this face are placed into the octtree.

The second step of the algorithm, which is repeated for each frame, uses the octtree to find the silhouette edges for a given camera position by intersecting the dual plane with the dual surface.

We have implemented an interactive silhouette viewer based on the dual space method. In our tests, silhouette tests were performed on twice as many triangles as there were actual triangles containing silhouettes, suggesting that performance is roughly linear in the number of silhouette triangles. This represents a substantial speedup over traversing the entire mesh. Silhouette edge detection and visibility calculations on the three-times subdivided Venus model (∼90,000 triangles) can be performed at approximately 17 frames per second on a 225 MHz SGI Octane, without using graphics hardware, which is similar to the performance of the nondeterministic algorithm of [25].

4.4 Visibility

Before computing visibility, we separate the silhouette curves into segments. Visibility is determined for each segment. The following points are used to separate segments: cusps, silhouette-feature joints, and inverse images of silhouette-feature and silhouette-silhouette intersections in image space. Visibility can change only at these points, thus each segment is either completely visible or invisible.

Determining visibility is fundamentally difficult for smooth surfaces, because it cannot be inferred precisely from visibility of the approximating mesh. Our algorithm can only guarantee that the correct visibility will be produced if the mesh is sufficiently fine, using a theoretically-estimated required degree of refinement. However, the estimate is too conservative and difficult to compute to be practical; in our implementation, we refine the mesh to a fixed subdivision level.

Our visibility algorithm is based on the following observation: at any area on the surface, the rate of change of the normal is bounded by the maximal directional curvature. For a sufficiently fine triangulation, one can guarantee that for any triangle for which $(\mathbf{n} \cdot (\mathbf{p} - \mathbf{c}))$ changes sign, there is a silhouette edge of the polygonal approximation adjacent to a vertex of the triangle. We use the visibility of these edges to compute visibility of the silhouette curves. The visibility of the silhouette edges can be determined using known techniques (e.g. [25]).

For each curve we find visibility of all nearby silhouette edges (which is not necessarily consistent) and use the visibility of the majority of the edges to determine visibility of the chain. It is possible to show that this method will produce correct visibility for sufficiently fine meshes in the following sense: there is a smooth surface for which the precise projection has the same topology as the one computed by our method.

In practice, we have found that the algorithm performs well even without extra refinement near the silhouettes, provided that the original mesh is sufficiently close to the surface. An efficient algorithm with better-defined properties would be useful.

5 Direction Fields on Surfaces

Fields on surfaces. To generate hatches, we need to choose several direction fields on visible parts of the surface. The direction fields are different from the more commonly used vector fields: unlike a vector field, a direction field does not have a magnitude and does not distinguish between the two possible orientations.

The fields can either be defined directly in the image plane as in [31], or defined on the surface and then projected. The advantage of the former method is that the field needs to be defined and continuous only in each separate area of the image. However, it is somewhat more difficult to use the information about the shape of the objects when constructing the field, and the field must be recomputed for each image. We choose to generate the field on the surface first.

A number of different fields on surfaces have been used to define hatching directions. The most commonly-used field is probably the field of isoparametric lines; this method has obvious limitations,

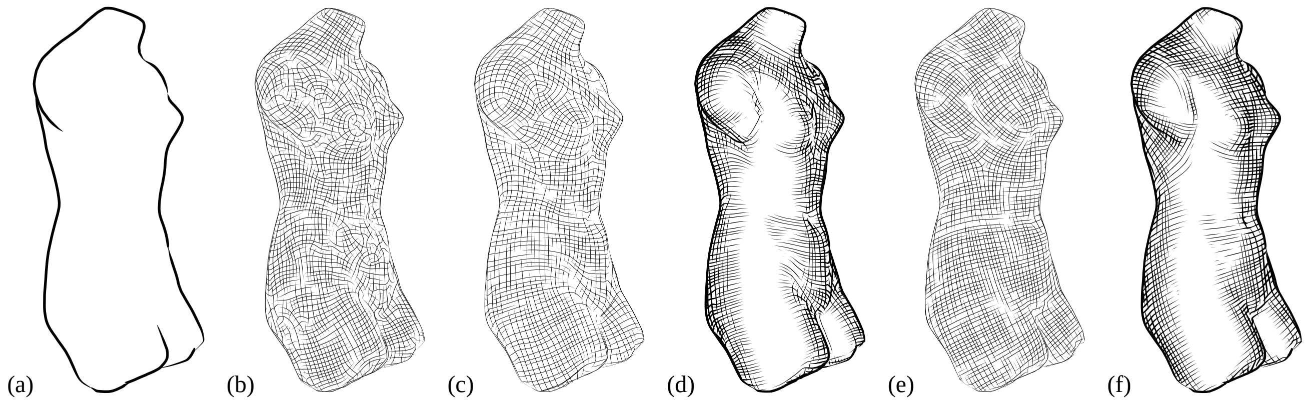


Figure 8: Direction fields on the Venus. (a) Silhouettes alone do not convey the interior shape of the surface. (b) Raw principle curvature directions produce an overly-complex hatching pattern. (c) Smooth cross field produced by optimization. Reliable principal curvature directions are left unchanged. Optimization is initialized by the principal curvatures. (d) Hatching with the smooth cross field. (e) Very smooth cross field produced by optimizing all directions. (f) Hatching from the very smooth field.

as the parameterization may be very far from isometric, and is not appropriate for surfaces lacking a good natural parameterization, such as subdivision surfaces and implicit surfaces. The successes and failures of this approach provide valuable clues for construction of fields for hatching.

The most natural geometric candidate is the pair of principal curvature direction fields [13, 21]. corresponding to the minimal and maximal curvatures[2]. We will refer to the integral lines of these fields as *curvature lines.* These fields do not depend on parameterization, capture important geometric features, and are consistent with the most common two-directional hatching pattern. However, they suffer from a number of disadvantages. All umbilical points (points with coinciding principal curvatures) are singularities, which means that the fields are not defined anywhere on a sphere and have arbitrarily complex structure on surfaces obtained by small perturbations of a sphere. On flat areas (when both curvatures are very small) the fields are likely to result in a far more complex pattern than the one that would be used by a human.

Other candidates include isophotes (lines of constant brightness) and the gradient field of the distance to silhouette or feature lines [25, 12]. Both are suitable for hatching in a narrow band near silhouettes or feature lines, but typically do not adequately capture shape further from silhouettes, nor are they suitable for cross-hatching.

Our approach is based on several observations about successes and failures of existing methods, as well as hatching techniques used by artists.

- *Cylindric surfaces.* Surface geometry is rendered best by principal curvature directions on cylindrical surfaces, that is, surfaces for which one of the principal curvatures is zero (all points of the surface are parabolic). This fact is quite remarkable: psychophysical studies confirm that even a few parallel curves can create a strong impression of a cylindrical surface with curves interpreted as principal curvature lines [32, 24]. Another important observation is that for cylinders the principal curvature lines are also geodesics, which is not necessarily true in general. Hatching following the principal curvature directions fails when the ratio of principal curvatures is close to one.

Deussen et al. [9] uses intersections of the surface with planes to obtain hatch directions; the resulting curves are likely to be locally close to geodesics on slowly varying surfaces.

- *Isometric parameterizations.* Isoparameteric lines work well as curvature directions when a parameterization exists and is close to isometric, i.e. minimizes the metric distortion as described in, for example, [10, 27]. In this case, parametric lines are close to geodesics. Isoparametric lines were used by [36, 11].
- *Artistic examples.* We observe that artists tend to use relatively straight hatch lines, even when the surface has wrinkles. Smaller details are conveyed by varying the density and the number of hatch directions (Figure 9).

[2]It is possible to show that for a surface in general position, these fields are always globally defined, excluding a set of isolated singularities.

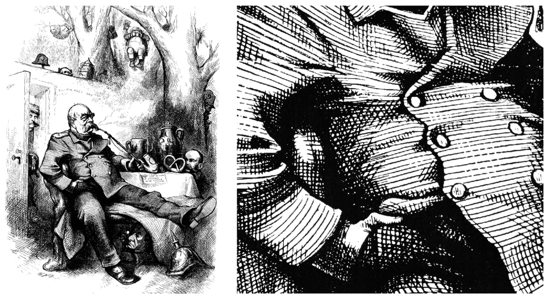


Figure 9: Almost all hatches in this cartoon by Thomas Nast curve only slightly, while capturing the overall shape of the surface. Note that the hatches often appear to follow a cylinder approximating the surface. Small details of the geometry are rendered using variations in hatch density.

These observations lead to the following simple requirements for hatching fields: *in areas where the surface is close to parabolic, the field should be close to principal curvature directions; on the whole surface, the integral curves of the field should be close to geodesic*. In addition, if the surface has small details, the field should be generated using a smoothed version of the surface.

Cross fields. While it is usually possible to generate two global direction fields for the two main hatch directions, we have observed that this is undesirable in general. There are two reasons for this: first, if we would like to illustrate nonorientable surfaces, such fields may not exist. Second, and more importantly, there are natural cross-hatching patterns that cannot be decomposed into two smooth fields even locally (Figure 10). Thus, we consider *cross fields*, that is, maps defined on the surface, assigning an unordered pair of perpendicular directions to each point.

Constructing Hatching Fields. Our algorithm is based on the considerations above and proceeds in steps.

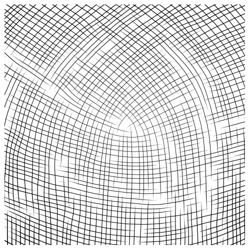


Figure 10: A cross-hatching pattern produced by our system on a smooth corner. This pattern cannot be decomposed into two orthogonal smooth fields near the corner singularity. The analytic expression for a similar field in the plane is $v_1(r,\theta) = [\cos(\theta/4), \sin(\theta/4)]$; $v_2(r,\theta) = [-\sin(\theta/4), \cos(\theta/4)]$. This field is continuous and smooth only if we do not distinguish between v_1 and v_2.

Step 1. Optionally, create a smoothed copy of the original mesh. The copy is used to compute the field. The amount of smoothing is chosen by the user, with regard to the smoothness of the original mesh, and the scale of geometric detail the user wishes to capture in the image. For example, no smoothing might be necessary for a close-up view of a small part of a surface, while substantial smoothing may be necessary to produce good images from a general view; in practice we seldom found this to be necessary.

Step 2. Identify areas of the surface which are sufficiently close to parabolic, that is, the ratio of minimal to maximal curvature is high, and at least one curvature is large enough to be computed reliably. Additionally, we mark as unreliable any vertex for which the average cross field energy of its incident edges exceeds a threshold, in order to allow optimization of vertices that begin singular.

Step 3. Initialize the field over the whole surface by computing principal curvature directions. If there are no quasi-parabolic areas, user input is required to initialize the field.

Step 4. Fix the field in quasi-parabolic areas and optimize the field on the rest of the vertices, which were marked as unreliable. This step is of primary importance and we describe it in greater detail.

Our optimization procedure is based on the observation that we would like the integral lines of our field to be close to geodesics. We use a similar, but not identical, requirement that the field is as close to constant as possible. Minimizing the angles between the world-space directions at adjacent vertices of the mesh is possible, but requires constrained optimization to keep the directions in the tangent planes. We use a different idea, based on establishing a correspondence between the tangent planes at different points of the surface, which, in some sense, corresponds to the minimal possible motion of the tangent plane as we move from one point to another. Then we only need to minimize the change of the field with respect to the corresponding directions in the tangent planes.

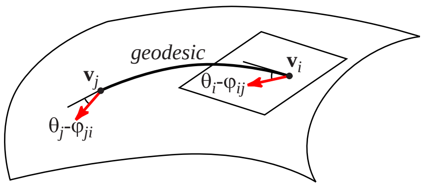


Figure 11: Moving vectors along geodesics.

Given two sufficiently close points $\mathbf{p}_1$ and $\mathbf{p}_2$ on a smooth surface, a natural way to map the tangent plane at $\mathbf{p}_1$ to the tangent plane at $\mathbf{p}_2$ is to transport vectors along the geodesics (Figure 11); for sufficiently close points there is a unique geodesic $\gamma(t)$, $t = 0..1$, connecting these points. This is done by mapping a unit vector $\mathbf{u}_1$ in the tangent plane at $\mathbf{p}_1$ to a unit vector $\mathbf{u}_2$ in the tangent plane at $\mathbf{p}_2$, such that the angle between $\mathbf{u}_1$ and the tangent to the geodesic $\gamma'(0)$ is the same as the angle between $\mathbf{u}_2$ and $\gamma'(1)$. In discrete case, for adjacent vertices of the approximating mesh $\mathbf{v}_i$ and $\mathbf{v}_j$, we approximate the tangents to the geodesic by the projections of the edge $(\mathbf{v}_i, \mathbf{v}_j)$ into the tangent planes at the vertices. Let the directions of these projections be $\mathbf{t}_{ij}$ and $\mathbf{t}_{ji}$. Then a rigid transformation T_{ij} between the tangent planes is uniquely defined if we require that $\mathbf{t}_{ij}$ maps to $\mathbf{t}_{ji}$ and that the transformation preserves orientation. Then for any pair of tangent unit vectors $\mathbf{w}_i$ and $\mathbf{w}_j$ at $\mathbf{v}_i$ and $\mathbf{v}_j$ respectively, we can use $\|T_{ij}\mathbf{w}_i - \mathbf{w}_j\|$ to measure the difference between directions. One can show that the value of this expression is the same as $\|T_{ji}\mathbf{w}_j - \mathbf{w}_i\|$. To measure the difference between the values of the cross field at two points, we choose a unit tangent vector for each point. The vectors are chosen along the directions of the cross field. There are four possible choices at each point. We choose a pair of unit vectors for which the difference is minimal.

We now explicitly specify the energy functional. The cross field is described by a single angle θ_i for each vertex $\mathbf{v}_i$, which is the angle between a fixed tangent direction $\mathbf{t}_i$, and one of the directions of the cross field; we do not impose any limitations on the value of θ_i, and there are infinitely many choices for θ_i differing by $n\pi/2$ that result in the same cross field. Let φ_{ij} be the direction of the projection of the edge $(\mathbf{v}_i, \mathbf{v}_j)$ into the tangent plane at $\mathbf{v}_i$. Using this choice of coordinates, one can show that the quantity $\|T_{ij}\mathbf{w}_i - \mathbf{w}_j\|$ is equal to $\min_k \sqrt{2 - 2\cos\left((\theta_i - \varphi_{ij}) - (\theta_j - \varphi_{ji}) + k\pi/2\right)}$. Minimization of this quantity is equivalent to minimization of $E(i,j) = \min_k\left(-\cos\left((\theta_i - \varphi_{ij}) - (\theta_j - \varphi_{ji}) + k\pi/2\right)\right)$, which is not differentiable. We observe, however, that $E_0(i,j) = -8E(i,j)^4 + 8E(i,j)^2 - 1$ is just $-\cos 4\left((\theta_i - \varphi_{ij}) - (\theta_j - \varphi_{ji})\right)$, and is a monotonic function of $E(i,j)$ on $[\sqrt{2}/2..1]$, the range of possible values of $E(i,j)$. Thus, instead of minimizing $E(i,j)$, we can minimize $E_0(i,j)$. We arrive at the following simple energy:

$$E_{\text{field}} = -\sum_{\text{all edges } (\mathbf{v}_i, \mathbf{v}_j)} \cos 4\left((\theta_i - \varphi_{ij}) - (\theta_j - \varphi_{ji})\right)$$

which does not require any constraints on the variables θ_i. Note that the values φ_{ij} are constant. Due to the simple form of the functional, it can be minimized quite quickly. We use a variant of the BFGS conjugate gradient algorithm described in [37] to perform minimization. For irregularly-sampled meshes, the energy may also be weighted in inverse proportion to edge length. We have not found this to be necessary for the meshes used in this paper. The result of the optimization depends on the threshold chosen to determine which vertices are considered unreliable; in the extreme cases, all vertices are marked as unreliable and the whole field is optimized, or all vertices are marked as reliable and the field remains unoptimized. Figure 8 shows the results for several thresholds.

6 Rendering Style

6.1 Style Rules

Our rendering style is based to some extent on the rules described by G. Francis in *A Topological picturebook* [15], which are in turn based on Nikolaïdes' rules for drawing drapes [26]. We have also used our own observations of various illustrations in similar styles. We begin our style description by defining undercuts and folds. A visible projected silhouette curve separates two areas of the image: one containing the image of the part of the surface on which the curve is located, the other empty or containing the image of a different part of the surface. We call the former area a *fold*. If the

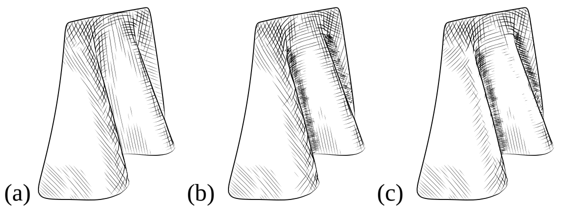


Figure 12: Hatching rules shown on drapes. (a) There are 3 main discrete hatch densities: highlights, midtones, and shadows, corresponding to 0, 1, and 2 directions of hatches. (b) Undercuts. (c) “Mach bands.” Undercuts and Mach bands increase contrast where surfaces overlap.

latter area contains the image of a part of the surface, we call it an *undercut*.

We use the following rules, illustrated in Figure 12.

- The surface is separated into four levels of hatching: highlights and Mach bands (no hatching), midtones (single hatching), shadowed regions (cross-hatching), and undercuts (dense cross-hatching). Inside each area, the hatch density stays approximately uniform. The choice of the number of hatch directions used at a particular area of the surface is guided by the lighting and the following rules:
- If there is an undercut, on the other side of the silhouette from a fold, a thin area along the silhouette on the fold side is not hatched (“Mach band effect”).
- Undercuts are densely hatched.
- Hatches are approximately straight; a hatch is terminated if its length exceeds a maximum, or if its direction deviates from the original by more than a fixed angle.
- Optionally, hatch thickness within each density level can be made inversely proportional to lighting; the resulting effect is rather subtle, and is visible only when the hatches are relatively thick.

6.2 Hatch Placement

The hatching procedure has several user-tunable parameters: basic hatch density specified in image space; the hatch density for undercuts; the threshold for highlights (the areas which receive no hatching); the threshold that separates single hatch regions from cross hatch regions; the maximum hatch length; the maximum deviation of hatches from the initial direction in world space. Varying these parameters has a considerable effect both on the appearance of the images and on the time required by the algorithm. Threshold values are usually chosen to divide the object more or less evenly between different hatching levels.

Once we have a hatching field, we can illustrate the surface by placing hatches along the field. We first define three intensity regions over the surface: no hatching (highlights and Mach bands), single hatching (midtones), and cross hatching (shadowed regions). Furthermore, some highlight and hatch regions may be marked as undercut regions. The hatching algorithm is as follows:

1. Identify Mach bands and undercuts.
2. Cover the single and cross hatch regions with cross hatches, and add extra hatches to undercut regions.
3. Remove cross-hatches in the single hatch regions, leaving only one direction of hatches.

6.3 Identifying Mach Bands and Undercuts

In order to identify Mach bands and undercuts, we step along each silhouette and boundary curve. A ray test near each curve point is used to determine if the fold overlaps another surface. Undercuts and Mach bands are indicated in a 2D grid, by marking every grid cell within a small distance of the fold on the near side of the surface as a Mach band, and by marking grid cells on the far side of the surface within a larger distance as undercuts. (This is the same 2D grid as used for hatching in the next section.)

6.4 Cross-hatching

We begin by creating evenly-spaced cross-hatches on a surface. We adapt Jobard and Lefers’ method for creating evenly-spaced streamlines of a 2D vector field [22]. The hatching algorithm allows us to place evenly-spaced hatches on the surface in a single pass over the surface.

Our algorithm takes two parameters: a desired hatch separation distance d_{sep}, and a test factor d_{test}. The separation distance indicates the desired image-space hatch density; a smaller separation distance is used for undercuts. The algorithm creates a queue of surface curves, initially containing the critical curves (silhouettes, boundaries, creases, and self-intersections). While the queue is not empty, we remove the front curve from the queue and seed new hatches along it at points evenly-spaced in the image. Seeding creates a new hatch on the surface by tracing the directions of the cross-hatching field. Since the cross field is invariant to 90 degree rotations, at each step the hatch follows the one of four possible directions which has the smallest angle with the previous direction. Hatches are seeded perpendicular to all curves. Hatches are also seeded parallel to other hatches, at a distance d_{sep} from the curve. A hatch continues along the surface until it terminates in a critical curve, until the world-space hatch direction deviates from the initial hatch direction by more than a constant, or until it comes near a parallel hatch. This latter condition occurs when the endpoint of the hatch $\mathbf{p}_1$ is near a point $\mathbf{p}_2$ on another hatch, such that the following conditions are met:

- $||\mathbf{p}_1 - \mathbf{p}_2|| < d_{test}d_{sep}$, measured in image space.
- A straight line drawn between the two points in image space does not intersect the projection of any visible critical curves. In other words, hatches do not “interfere” when they are not nearby on the surface.
- The world space tangents of the two hatch curves are parallel, i.e. the angle between them is less than 45 degrees, after projection to the tangent plane at $\mathbf{p}_1$.

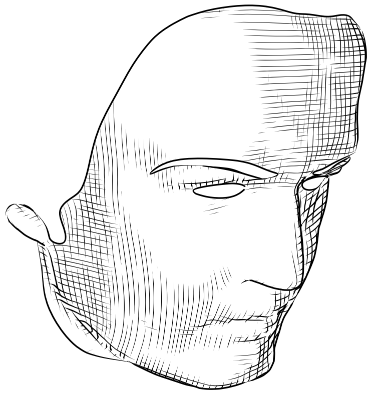
The search for nearby hatches is performed by placing all hatches in a 2D grid with grid spacing equal to d_{sep}. This ensures that at most nine grid cells must be searched to detect if there are hatches nearby the one being traced.

6.5 Hatch Reduction

Once we have cross-hatched all hatch regions, we remove hatches from the single hatch regions until they contain no cross-hatches. By removing hatches instead of directly placing single a hatch direction, we avoid the difficulty inherent in producing a consistent vector field on the surface. Our algorithm implicitly segments the visible single-hatch regions into locally-consistent single hatching fields. This allows us to take advantage of the known view direction and the limited extent of these regions.

The reduction algorithm examines every hatch on the surface and deletes any hatch that is perpendicular to another hatch. In particular, a hatch is deleted if it contains a point $\mathbf{p}_1$ nearby a point $\mathbf{p}_2$ on another hatch such that:

- $\mathbf{p}_1$ and $\mathbf{p}_2$ lie within the single hatch region.
- $||\mathbf{p}_1 - \mathbf{p}_2|| < 2d_{sep}$, measured in image space.
- A straight line drawn between the two points in image space does not intersect any visible critical curve.
- The world space tangents of the two hatch curves are perpendicular, i.e. the angle between them is greater than 45 degrees after projection to the tangent plane at $\mathbf{p}_1$.



Deleting a hatch entails clipping it to the cross-hatch region; the part of the hatch that lies within the cross-hatch region is left untouched.

The order in which hatches are traversed is important; a naïve traversal order will usually leave the single hatch region uneven and inconsistent. We perform a breadth-first traversal to prevent this. A queue is initialized with a hatch curve. While the queue is not empty, the front curve is removed from the queue. If it is perpendicular to another curve in the single hatch region, then the curve is deleted, and all parallel neighbors of the hatch that have not been visited are added to the queue. When the queue is empty, a hatch that has not yet been visited is added to the queue, if any remain. The tests for perpendicular is as described above; the angle condition is reversed for the parallel test.

7 Results and Conclusions

Most of the illustrations in this paper were created using our system. Figures 1, 8 demonstrate the results for relatively fine meshes that define surfaces with complex geometry. Figures 2 and 13 show the results of using our system to illustrate several mathematical surfaces.

The time required to create an illustration varies greatly; while silhouette drawings can be computed interactively, and the field optimization takes very little time, hatching is still time-consuming, and can take from seconds to minutes, depending on hatch density and complexity of the model. Also, for each model the parameters of the algorithms (thresholds for hatching, position of the light sources, hatch density) have to be carefully chosen;

Future work. As we have already mentioned, improvements should be made to the silhouette visibility algorithm. Performance was not our goal for the hatching algorithm. It is clear that substantial speedups are possible. While the quality of fields generated by our algorithms is quite good, it would be desirable to reduce the number of parameters that may be tuned.

A more fundamental problem is the lack of control over the the number, type and placement of singularities of the generated field. As most surfaces of interest have low genus, the number of singularities can be very small for most surfaces.[3] However, the user currently has little control over their placement and additional support must be provided. Furthermore, the hatch reduction algorithm could be made more robust to irregular cross-hatching patterns, and the hatching could be improved reduce hatching artifacts, perhaps by employing the optimization technique of Turk and Banks [33].

[3]The relation between the numbers of singularities of different types is determined by the analogs of Euler formula; such formulas are known for vector and tensor fields; obtaining classification of singularities and a formula of this type for the cross fields described in the paper is an interesting mathematical problem.

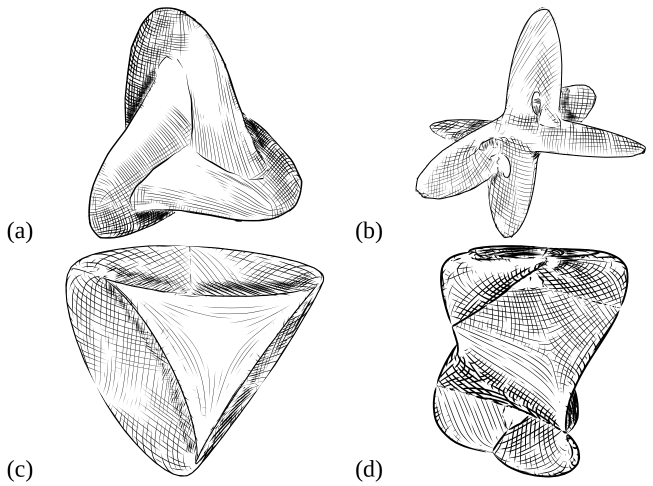


Figure 13: Several surfaces generated using G. Francis' generalization of Apéry's Romboy homotopy [16]. (a) Boy surface; (b) "Ida"; (c) Roman surface; (d) Etruscan Venus.

Acknowledgments

Our special thanks go to Jianbo Peng who implemented the dual surface silhouette detection algorithm. We are grateful to Pat Hanrahan, who suggested this research topic to us. We thank the anonymous reviewers for their comments. Chris Stolte participated in the project in its early stages.

This research was supported in part by the NSF grant DGE-9454173 and the NYU Center for Advanced Technology.

A C^2-surfaces based on subdivision

Commonly used subdivision surfaces, such as variants of Loop subdivision, produce either surfaces with curvatures that do not converge or have zero curvature at extraordinary vertices. There are fundamental reasons for this [29]. This property is rather undesirable, if we would like to compute silhouette curves, as it means either flat points or singular behavior near extraordinary points. We have developed a surface representation based on subdivision that produces surfaces that are everywhere C^2, do not have zero curvature at extraordinary vertices, and agree arbitrarily well with the limit surfaces produced by subdivision. This representation is described elsewhere [38]. However, for our purposes it is sufficient to have a way to compute curvatures for the surface associated with a mesh, and it is not necessary to have a complete surface evaluation algorithm.

The curvature computation that we propose is based on ideas from subdivision and is compatible with the curvature computations for subdivision surfaces in the regular case.

Consider a vertex $\mathbf{v}$ of the initial mesh of valence k. We will regard a part of the smooth surface corresponding to the 1-neighborhood of $\mathbf{v}$ as parameterized over a regular k-gon in the plane. Introduce the polar coordinates (r, φ) in the plane, with $u = r\cos\varphi$ and $v = r\sin\varphi$. then the second-order approximation to the surface can be written as

$$a_0 + (a_{11}\sin\varphi + a_{12}\cos\varphi)r + (a_{20} + a_{21}\sin 2\varphi + a_{22}\cos 2\varphi)r^2$$

A simple calculation shows that the least squares fit to $k+1$ points of the 1-neighborhood $p_0 \dots p_k$ assumed to be values at $(\sin(2\pi i/k), \cos(2\pi i/k))$, $i = 0..k$. with p_0 in the center, leads to

$$a_0 = p_0; \quad a_{20} = -p_0 + \frac{1}{k}\sum_i p_i$$

$$a_{11} = \frac{2}{k}\sum_i p_i \sin\frac{2\pi i}{k}; \quad a_{12} = \frac{2}{k}\sum_i p_i \cos\frac{2\pi i}{k}$$

$$a_{21} = \frac{2}{k}\sum_i p_i \sin\frac{4\pi i}{k}; \quad a_{22} = \frac{2}{k}\sum_i p_i \cos\frac{4\pi i}{k}$$

Note that the formulas for a_{11} and a_{21} coincide with the standard formulas for the tangents to the Loop subdivision surface, and a_{20}, a_{21}, a_{22}, with appropriate variable changes, produce second derivatives in the regular case. To make our calculations compatible with the Loop surface, we replace $a_0 = p_0$ with $a_0 = p_0^{limit}$, the limit position of the control point p_0. As a result, we obtain a set of simple rules for computing the coefficients of an approximating quadratic surface, which, after appropriate change of variables can be used to compute curvatures and is compatible with the Loop subdivision rules. In [38], we show that one can construct a C^2 surface which has precisely these curvatures at the vertices. A similar construction works for the boundary case. We should note that for valences $k = 3, 4$, the coefficients of the quadric are not independent, and thus not all possible local behaviors can be approximated well.

Given known partial derivatives $\mathbf{F}_u, \mathbf{F}_v, \mathbf{F}_{uu}, \mathbf{F}_{uv}, \mathbf{F}_{vv}$ of the local parameterization of the surface, the principal curvature directions and magnitudes can be computed as eigenvalues and eigenvectors of the following matrix:

$$\begin{pmatrix} E & F \\ F & G \end{pmatrix}\begin{pmatrix} L & M \\ M & N \end{pmatrix} \qquad (2)$$

where $E = (\mathbf{F}_u \cdot \mathbf{F}_u)$, $F = (\mathbf{F}_v \cdot \mathbf{F}_u)$, $G = (\mathbf{F}_v \cdot \mathbf{F}_v)$, $L = (\mathbf{F}_{uu} \cdot \mathbf{n})$, $M = (\mathbf{F}_{uv} \cdot \mathbf{n})$, $N = (\mathbf{F}_{vv} \cdot \mathbf{n})$.

References

[1] I. A. Babenko. Singularities of the projection of piecewise-linear surfaces in rspan3. *Vestnik Moskov. Univ. Ser. I Mat. Mekh.*, 1991(2):72–75.

[2] Thomas F. Banchoff and Ockle Johnson. The normal Euler class and singularities of projections for polyhedral surfaces in 4-space. *Topology*, 37(2):419–439, 1998.

[3] Fabien Benichou and Gershon Elber. Output sensitive extraction of silhouettes from polygonal geometry. *Pacific Graphics '99*, October 1999. Held in Seoul, Korea.

[4] William M. Boothby. *An Introduction to Differentiable Manifolds and Riemannian Geometry*. Academic Press, 1986.

[5] Wagner Toledo Corrêa, Robert J. Jensen, Craig E. Thayer, and Adam Finkelstein. Texture Mapping for Cel Animation. In *SIGGRAPH 98 Conference Proceedings*, pages 435–446, July 1998.

[6] Cassidy Curtis. Loose and Sketchy Animation. In *SIGGRAPH 98: Conference Abstracts and Applications*, page 317, 1998.

[7] Philippe Decaudin. Cartoon-Looking Rendering of 3D-Scenes. Technical Report 2919, INRIA, June 1996.

[8] Thierry Delmarcelle and Lambertus Hesselink. The topology of symmetric, second-order tensor fields. In *Visualization '94*, pages 140–147, October 1994.

[9] Oliver Deussen, Jörg Harnel, Andreas Raab, Stefan Schlechtweg, and Thomas Strothotte. An Illustration Technique Using Hardware-Based Intersections. *Graphics Interface '99*, pages 175–182, June 1999.

[10] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution Analysis of Arbitrary Meshes. In *Computer Graphics Proceedings*, Annual Conference Series, pages 173–182. ACM Siggraph, 1995.

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

[12] Gershon Elber. Line Art Illustrations of Parametric and Implicit Forms. *IEEE Transactions on Visualization and Computer Graphics*, 4(1), January – March 1998.

[13] Gershon Elber. Interactive line art rendering of freeform surfaces. *Computer Graphics Forum*, 18(3):1–12, September 1999.

[14] Gershon Elber and Elaine Cohen. Hidden Curve Removal for Free Form Surfaces. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 95–104, August 1990.

[15] George K. Francis. *A Topological Picturebook*. Springer-Verlag, New York, 1987.

[16] George K. Francis. The Etruscan Venus. In P. Concus, R. Finn, and D. A. Hoffman, editors, *Geometric Analysis and Computer Graphics*, pages 67–77. 1991.

[17] Amy Gooch. Interactive Non-Photorealistic Technical Illustration. Master's thesis, University of Utah, December 1998.

[18] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Richard Riesenfeld. Interactive Technical Illustration. In *Proc. 1999 ACM Symposium on Interactive 3D Graphics*, April 1999.

[19] Aaron Hertzmann. Introduction to 3D Non-Photorealistic Rendering: Silhouettes and Outlines. In Stuart Green, editor, *Non-Photorealistic Rendering*, SIGGRAPH Course Notes. 1999.

[20] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Huber Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconstruction. In *Computer Graphics Proceedings*, Annual Conference Series, pages 295–302. ACM Siggraph, 1994.

[21] Victoria L. Interrante. Illustrating Surface Shape in Volume Data via Principal Direction-Driven 3D Line Integral Convolution. In *SIGGRAPH 97 Conference Proceedings*, pages 109–116, August 1997.

[22] Bruno Jobard and Wilfrid Lefer. Creating evenly-spaced streamlines of arbitrary density. In *Proc. of 8th Eurographics Workshop on Visualization in Scientific Computing*, pages 45–55, 1997.

[23] Lutz Kettner and Emo Welzl. Contour Edge Analysis for Polyhedron Projections. In W. Strasser, R. Klein, and R. Rau, editors, *Geometric Modeling: Theory and Practice*, pages 379–394. Springer Verlag, 1997.

[24] Pascal Mamassian and Michael S. Landy. Observer biases in the 3D interpretation of line drawings. *Vision Research*, (38):2817—2832, 1998.

[25] Lee Markosian, Michael A. Kowalski, Samuel J. Trychin, Lubomir D. Bourdev, Daniel Goldstein, and John F. Hughes. Real-Time Nonphotorealistic Rendering. In *SIGGRAPH 97 Conference Proceedings*, pages 415–420, August 1997.

[26] Kimon Nikolaïdes. *The Natural Way to Draw*. Houghton Miffin, Boston, 1975.

[27] Hans Køhling Pedersen. A Framework for Interactive Texturing on Curved Surfaces. *Proceedings of SIGGRAPH 96*, pages 295–302, August 1996.

[28] Ramesh Raskar and Michael Cohen. Image Precision Silhouette Edges. In *Proc. 1999 ACM Symposium on Interactive 3D Graphics*, April 1999.

[29] Ulrich Reif. A degree estimate for polynomial subdivision surfaces of higher regularity. *Proc. Amer. Math. Soc.*, 124:2167–2174, 1996.

[30] Takafumi Saito and Tokiichiro Takahashi. Comprehensible Rendering of 3-D Shapes. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 197–206, August 1990.

[31] Michael P. Salisbury, Michael T. Wong, John F. Hughes, and David H. Salesin. Orientable Textures for Image-Based Pen-and-Ink Illustration. In *SIGGRAPH 97 Conference Proceedings*, pages 401–406, August 1997.

[32] Kent A. Stevens. Inferring shape from contours across surfaces. In Alex P. Pentland, editor, *From Pixels to Predicates*, pages 93–110. 1986.

[33] Greg Turk and David Banks. Image-Guided Streamline Placement. In *SIGGRAPH 96 Conference Proceedings*, pages 453–460, August 1996.

[34] J. Chal Vinson. *Thomas Nast: Political Cartoonist*. University of Georgia Press, Atlanta, 1967.

[35] Hassler Whitney. On singularities of mappings of euclidean spaces. I. Mappings of the plane into the plane. *Ann. of Math. (2)*, 62:374–410, 1955.

[36] Georges Winkenbach and David H. Salesin. Rendering Parametric Surfaces in Pen and Ink. In *SIGGRAPH 96 Conference Proceedings*, pages 469–476, August 1996.

[37] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Software*, 23(4):550–560, 1997.

[38] D. Zorin. Constructing curvature-continuous surfaces by blending. in preparation.