Computational Aesthetics in Graphics, Visualization, and Imaging (2010)
O. Deussen and P. Jepp (Editors)

Suggestive Hatching

Mayank Singh and Scott Schaefer

Texas A&M University

Abstract

We present a method for drawing lines on an object that depict both the shape and shading of the object. To do so, we construct a gradient field of the diffuse intensity of the surface to guide a set of adaptively spaced lines. The shape of these lines reflect the lighting under which the object is being viewed and its shape. When the light source is placed at the viewer's location, these lines emanate from silhouettes and naturally extend Suggestive Contours. By using a hierarchical proximity grid, we can also improve the quality of these lines as well as control their density over the image. We also provide a method for detecting and removing ridge lines in the intensity field, which lead to artifacts in the line drawings.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

The human visual system is adept at understanding the shape of a three-dimensional object from its shaded representation [Ram88]. Often artists depict this shading not only through smooth color gradients, but by using line strokes (referred as *hatching* [HZ00] or *shading strokes* [MKG*97]).

There are two classes of lines that have been used to illustrate shape in Computer Graphics. The first class is a set of sparse lines that are usually placed to highlight some natural discontinuity in the object or its appearance [CGL*08]. Lines such as Suggestive Contours [DFRS03] or Apparent Ridges [JDA07] fall into this category. These lines do not represent the shading of the surface, yet they greatly aid in conveying the shape of the object to the viewer.

The second class of lines are hatching lines that are not designed to indicate natural discontinuities in the shape, but serve a similar purpose. The density of these lines indicates information about the intensity of light reflected from the surface and the path the lines take provides clues as to the shape of the surface and create the illusion of volume by wrapping around the object. Figure 2 shows an example drawing by Albrecht Dürer that demonstrate this effect.

Contributions

These two categories of lines have remained separate for the most part. However, we aim to unify the construction of curves from the first class (namely Suggestive Contours and Suggestive/Principal Highlights) with hatching lines from the second class. In particular, we present a method for drawing lines on the surface to illustrate the intensity of shading as well as the shape of the object. We show how to define a vector field that naturally approximates Suggestive Contours and Suggestive/Principal Highlights. We also describe a hierarchical proximity grid for not only generating long, aesthetically pleasing lines, but also for controlling the density of the lines on the surface to match the intensity of the surface. Finally, we show how to detect ridges in the intensity field, which cause artifacts in the resulting image, and how to eliminate these lines using the proximity grid. Figure 1 shows an example of an abstract shape illustrated with

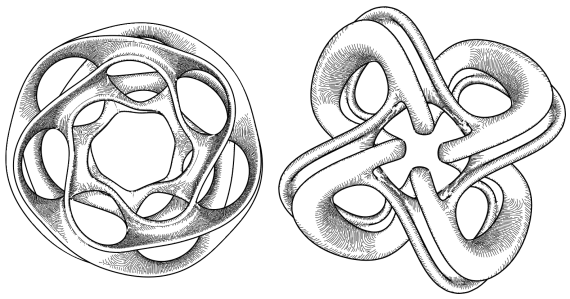

Figure 1: *An example of illustrating shapes using adaptively spaced shading strokes along with the silhouettes.*

Figure 2: *Example of a woodcut relief by Abrecht Dürer.*

lines generated by our algorithm. The results closely resemble wood engravings produced by Albrecht Dürer (Figure 2).

2. Background

There have been numerous methods developed for illustrating surfaces with sparse sets of lines and Rusinkiewicz et al. [RCDF08] provides a comprehensive survey. One notable member of this family is Suggestive Contours [DFRS03, DFR04], which has been shown to account for a large percentage of the lines that artists draw [CGL*08]. These lines correspond to minimums of diffuse intensity in the direction of the projected view vector when the light is co-located with the viewer's position. For some shapes, such as smooth convex objects, these lines may fail to impart additional information about the shape of the object [JDA07]. Our method can illustrate the shape of these objects through our shading strokes and we also approximate Suggestive Contours as a subset of the total set of lines we generate.

The converse of Suggestive Contours are Suggestive/Principal highlights [DR07]. As the authors note, these curves only approximate intensity ridges. We provide a method for detecting and removing these intensity ridges from our line drawings using our proximity grid.

There have also been numerous methods developed to create hatching lines on surfaces. Early work concentrated on simply creating lines on surfaces. Winkenbach et al. [WS96] uses parameter lines on parametric surfaces to create hatch lines, but this technique is limited to parametric surfaces and depends on the choice of parameterization. Deussen et al. [DHR*99] creates curves on surfaces by intersecting the surface with a set of parallel planes. However, these lines provide little information about the shape of the surface.

More recent work has concentrated on defining hatch marks using the direction of Principal curvature [HZ00, GIHL00, PHWF01, ZISS04]. These lines are solely dependent on the surface geometry and independent of the viewing direction and lighting under which an object is viewed. Since robust curvature directions are difficult to compute and may be noisy, Hertzmann and Zorin [HZ00] present a method for optimizing the fairness of the curvature field. Though this process is computationally expensive, the authors precompute the result and generate hatch marks by integrating lines over the surface using this fair vector field.

Praun et al. [PHWF01] describe a similar method to Hertzmann and Zorin [HZ00] except the authors use lapped textures [PFH00] to represent the surface parameterization. The authors then precompute a nested set of hatch strokes, which are stored in a matrix of textures. At runtime, based upon lighting intensity, their method selects a subset of these textures from the matrix and blends them together to create a continuous level of intensity.

Zander et al. [ZISS04] present an algorithm for rendering high quality hatch lines. Like the previous techniques, the authors also rely on precomputing an optimized vector field for their model. Their method renders hatch lines by evenly spacing the lines in model space with the width of the lines modulated to avoid dark regions due to perspective foreshortening. Depending on the size of the polygons, detecting nearby shading strokes to avoid intersections in the model space can be an expensive operation.

However, all of these hatching techniques generate lines whose paths are independent of viewing direction or even shading of the surface. None of these methods create hatch marks that naturally correspond to any sparse line drawing technique [CGL*08]. Moreover, surface regions that are flat or have constant curvature do not have unique principal curvature directions and these curvature dependent hatching methods may fail in these areas. In constrast, our method does not have these ambiguities and extends line drawing techniques [DFRS03, DR07] to draw a set of shading strokes that contain many of these spares lines as a subset.

Our mechanism for generating shading strokes is inspired by techniques commonly used in scientific visualization to visualize vector fields. Jobard and Lefer [JL97] propose a technique that produces aesthetically appealing, long streamlines. Their method creates a set of streamlines that flow up and down the vector field, emanating from a collection of 2D seed points. The authors make the observation that, in order to avoid creating many short streamlines, the separating distance between the seed points and the existing streamlines should be twice as much as the minimum distance between streamlines. Later, the same authors propose an addition to their work by adding a nested property to these streamlines [JL01]. Our hierarchical proximity grid extends Jobard and Lefer's method to create lines; though our goal is not to create evenly spaced lines but lines whose density modulates with the intensity of the surface lighting.

Mebarki et al. [MAD05] present an improvement upon Jobard's method in terms of the placement of streamlines. Their method uses a greedy algorithm for placing successive streamlines in a 2D vector field. The authors construct a constrained Delaunay triangulation of the currently drawn streamlines and place new seeds at the circumcenter of the triangle with the largest circumcircle. This method for seeding points produces streamlines that are long, aesthetically pleasing and farthest apart from each other.

Both the above mentioned techniques draw streamlines on

a 2D vector field. Recently, Spencer et al. [SLZC09] proposed a technique for tracing streamlines over 3D surfaces. Their algorithm essentially projects the surface vector field into the image plane and performs 2D vector tracing with special attention to discontinuities in the z-buffer. Their algorithm is similar to ours in that we too use an image based approach to determine if lines are too close to one another except we use a hierarchical proximity grid instead of a scanline sweep to order seed placement and adaptively control the density of lines over the surface. Our shading strokes are also built on the surface of the 3D model guided by a vector field that resides upon the surface so we need not be concerned with depth discontinuities in the image or perspective foreshortening of the vector field.

3. Shading Strokes

To draw lines on the surface illustrating the shading of the surface, we begin by defining a vector field over the surface. We trace this field by drawing shading strokes that follow the vector field starting from a seed point. We also control the density of lines over the surface by using a proximity check dependent upon the local intensity value of the surface.

3.1. Shading Vector Field

Our vector field is based upon computation of Lambertian reflectance (i.e. diffuse shading) with the light co-located with the viewer's position and follows the gradient construction given by Yu et al. [YZX*04]. Let V be the direction to the viewer from a vertex on the surface. The diffuse light intensity at a vertex on the surface is given by $V \cdot N_i$ where N_i is the unit normal at the i^{th} vertex. We use Gouroud shading to extend these intensities over the triangles of the surface. Given a point x inside the j^{th} triangle of the surface with vertices $p_{j,0}, p_{j,1}, p_{j,2}$ and vertex normals $N_{j,0}, N_{j,1}, N_{j,2}$, the intensity $I(x)$ is given by

$$I(x) = \sum_{k=0}^{2} (V \cdot N_{j,k}) \frac{A(x, p_{j,k+1}, p_{j,k+2})}{A(p_{j,0}, p_{j,1}, p_{j,2})}$$

where $A(a,b,c)$ represents the signed area of the triangle with vertices a,b,c. Now we define a vector field over the surface by computing the gradient of the intensity as

$$\nabla I(x) = \sum_{k=0}^{2} (V \cdot N_{j,k}) \frac{\nabla A(x, p_{j,k+1}, p_{j,k+2})}{A(p_{j,0}, p_{j,1}, p_{j,2})}.$$

Since $\nabla A(x, p_{j,k+1}, p_{j,k+2}) = (p_{j,k+2} - p_{j,k+1}) \times F_j$, where F_j is the unit normal of triangle j, we can rewrite the above equation as

$$\nabla I(x) = \sum_{k=0}^{2} \frac{V \cdot N_{j,k}}{A(p_{j,0}, p_{j,1}, p_{j,2})} (p_{j,k+2} - p_{j,k+1}) \times F_j. \quad (1)$$

This vector field is coplanar with each triangle and piecewise constant over the surface. Note that unlike curvature this is not a view-independent attribute of the surface and needs to be computed for each frame.

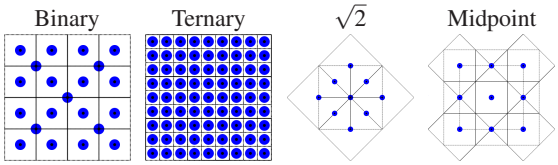

Figure 3: *Comparison of various seeding techniques. Binary subdivision concentrates samples along the diagonal. Ternary subdivision refines space too quickly. $\sqrt{2}$ and midpoint subdivision produce evenly spaced seeds.*

3.2. Building Shading Strokes

Drawing shading strokes that follow the vector field described in Section 3.1 is very similar to visualizing vector fields that arise in physical simulations, and many techniques discussed in Section 2 have been devised to do so. The basic idea is to begin with a seed point somewhere in the domain and follow the vector field in both the positive and negative directions until we reach some predetermined threshold (intensity in our case) or a local maximum/minimum. To do so, we sample the vector field and move in discrete steps corresponding to the size of the triangle in the direction of the vector field at that point. Since the vector field is piecewise constant and lies in the tangent plane, numerical integration of lines over the surface is a straightforward process. We do so using a method similar to Dong et al. [DKG05].

3.3. Seeding and Spacing of Lines

To create shading strokes on the surface, we start with a seed point on the surface and trace shading strokes according to Section 3.2. However, the placement of these seed points can greatly affect the quality and appearance of the resulting strokes. Ideally we should be able to control the density of the shading strokes over a region of the image guaranteeing both that there are no large gaps in the shading strokes and that their density is proportional to the intensity of the light to provide an intensity gradient over the surface.

Notice that we do not control the density of shading strokes in model space. Instead, we control the density in image space. While evenly spaced shading strokes on the surface would provide cues about depth due to foreshortening of the lines under perspective projection, the density of the lines would give the impression that objects farther away are actually darker. Hence we use an image-based algorithm to place shading strokes on the surface.

In the context of 2D vector field visualization, Mebarki et al. [MAD05] showed that seeding the shading strokes at a point farthest away from the existing shading strokes produces longer, more aesthetically pleasing lines. Their solution uses a constrained Delaunay triangulation over the 2D

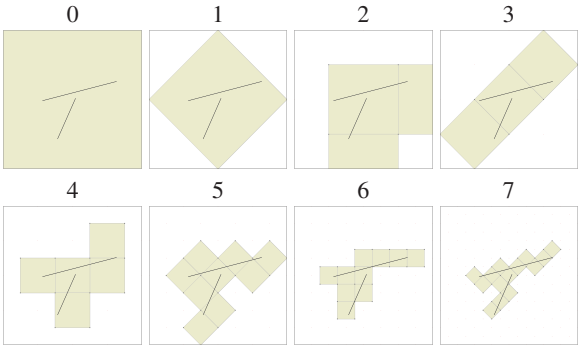

Figure 4: *For each line segment, we rasterize the segment into each level of the proximity grid. Cells that are marked will not generate further seed points.*

image that is rebuilt after each line is drawn to estimate the points furthest away from previously generated lines. Our setting is more difficult as we have boundaries in our image corresponding to the silhouettes of the object. Also, we would like to adapt the density of the lines to the intensity of the surface lighting.

Our solution is to use a hierarchical proximity grid to encode, in a multi-resolution fashion, information about where shading strokes have been previously drawn. This proximity grid is simply a sequence of uniform grids of increasingly smaller cells whose maximum depth is specified by the user. The center of each cell in this grid corresponds to a seed point, which we project onto the surface via a ray intersection and initiate a shading stroke from that point. When generating seed points, we traverse this hierarchy from the coarsest level to the finest level of the tree. For each level, we generate seed points in scanline order for each cell that is not marked. We also rasterize completed scanlines into each level of the hierarchy and mark each cell that the lines cross.

This process also provides proximity information in a multi-resolution fashion over the image. For each empty cell, we know that a shading stroke is no closer than half the width of that cell from its potential seed point. The cell width also decreases by a constant scale factor at each level of the hierarchy. Hence, this process tends to generate seed points as far as possible from previously generated shading strokes. Notice that this process is not as exact as the method described by Mebarki et al. [MAD05]. However, our method is faster as a Delaunay triangulation does not have to be regenerated over the image every time a shading stroke is drawn.

To build this hierarchy of grids, we consider several different types of quadrilateral refinement patterns that have been explored before in the context of surface subdivision ([PR97] and [LMB04]). Figure 3 shows four different refinement patterns. Binary subdivision is perhaps the most obvious refinement method. However, binary subdivision does not produce a uniform seeding pattern and concentrates samples along diagonals. On the other hand, ternary subdivision

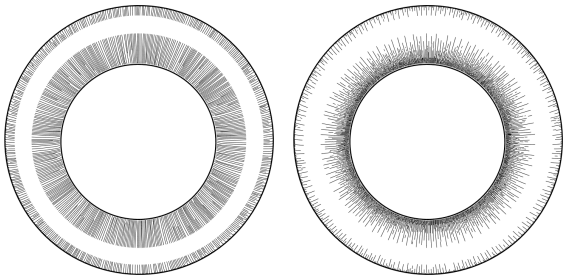

Figure 5: *Example of a torus rendered with strokes with uniform maximum depth (left) and adaptive depth (right).*

does create uniform sampling but refines the grid quickly, which will limit our ability to finely control the density of lines to provide a shading effect on the surface.

Ideally, we would like the size of the cells to refine as slowly as possible to provide a better approximation to the farthest shading stroke and provide finer control over the local density of these strokes. Both $\sqrt{2}$ and midpoint subdivision refine the cells slower than binary subdivision. However, we prefer midpoint refinement because, at every level of the hierarchy, the grid corresponds to a voronoi diagram of the previously generated seed points. Figure 4 shows an example of how lines are written into the hierarchical proximity grid using midpoint subdivision. Cells that are marked will not generate future seed points.

We also use the proximity grid to control the density of shading strokes in a particular region. As we are tracing the shading strokes using the method in Section 3.2, we check the proximity grid at a depth dependent on the value of $I(x)$ at each point along the line to see if the line has entered a marked cell. If so, we truncate that stroke at that point.

To convey the shading of the surface, we would like the density of lines in a cell to match the intensity of the surface over that cell. We can approximate this effect by computing the ratio of the width of a line (1 pixel) to the width of a cell. Assuming we use midpoint subdivision for the proximity grid, the density of a single line compared to a cell is

$$1/\left(\frac{w}{\sqrt{2}^d}\right) = \frac{\sqrt{2}^d}{w}$$

where w is the width of the image and d is the depth of the grid cell. Since a line darkens the image, we set this equation to $1-I(x)$ and solve for the depth.

$$d = 2\log_2(w(1-I(x))) \tag{2}$$

Using this adaptive grid we can enable the lines to come close to each other in darker regions, while increasing their spacing in lighter regions of the surface. Notice that this method produces discrete intensity levels corresponding to each level of the grid. By using a grid that refines slowly

Figure 6: *Portion of drawing by Albrecht Dürer showing strokes terminated near an intensity ridge.*

we obtain more intensity levels and fewer banding artifacts associated with these discrete levels.

Figure 5 shows an example of using a uniform maximum depth (independent of $I(x)$) versus adapting the depth of the proximity check for truncating lines to the intensity of the surface. While the shading strokes are much shorter using an adaptive grid, the vast majority of these shorter strokes occur in darker regions of the surface and are necessary to provide the illusion of variable intensity along the surface.

4. Ridges and Valleys

Since we trace the gradient field to form the shading strokes on the surface, ridges and valleys in $I(x)$ attract shading strokes. The valleys of the intensity field approximate Suggestive Contours [DFRS03] as Suggestive Contours are minimums of the intensity field with respect to the projected view vector. Therefore, the shading strokes will naturally emanate from and extend Suggestive Contours.

Ridges of $I(x)$, like Suggestive Contours, attract shading strokes as well. Unlike Suggestive Contours, these ridges do not tend to correspond to feature lines that an artist would draw since these ridges are high intensity regions of the surface and lines darken the image. Therefore we consider these ridges artifacts of our shading algorithm and would like remove them from the image. A similar effect can be observed in drawings created by Albrecht Dürer (see figure 6). In the pillow beneath the woman's head, the lines are terminated as soon as a ridge in the intensity function is encountered.

Figure 7 shows that we cannot remove these ridge artifacts by simply lowering the intensity threshold at which we stop tracing the lines. While lowering the intensity threshold can remove the ridge artifacts, we also lose shading strokes over large portions of the object, which leaves little visual cues as to the shape of the object, as seen in the middle image. Removing ridges allows us to create shading lines of the entire surface while avoiding the artifacts caused by ridges.

To remove these ridge artifacts, we detect the intensity ridges on the surface and rasterize them into the proximity

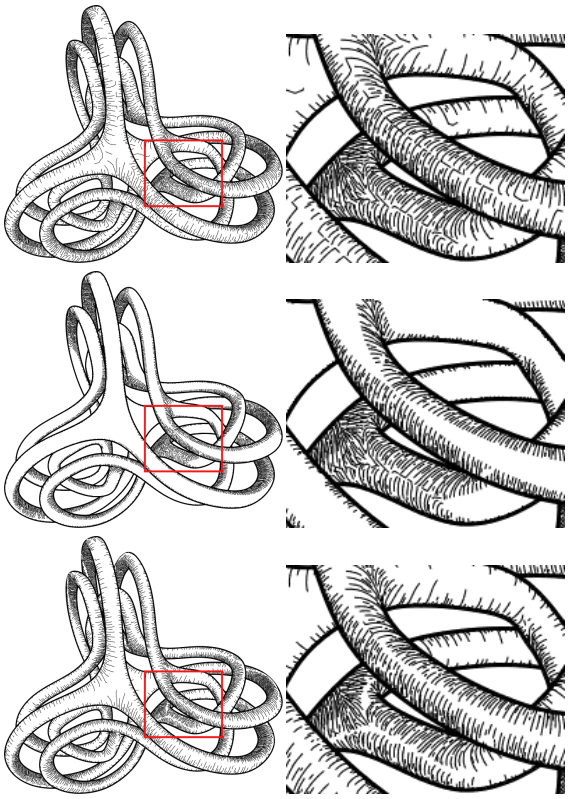

Figure 7: *From top to bottom: no intensity threshold without ridge removal, 0.65 intensity threshold without ridge removal, and no intensity threshold with ridge removal. Right: a zoomed portion to highlight the effect of intensity ridges.*

grid from Section 3.3 at each level before drawing shading strokes effectively stopping our lines from getting too close to the ridge. Note that these ridges are similar to the lines produced by Suggestive/Principal Highlights, which are local maximums of intensity in the direction of the projected viewing vector and its orthogonal counterpart in the tangent plane. The union of both sets of lines approximates intensity ridges well over most of the surface (when the ridge direction roughly aligns with the projected viewing vector or its complement). However, Suggestive/Principal Highlights do not always align with intensity ridges.

We can detect ridges in the intensity function $I(x)$ by looking for a set of points such that the first derivative in the direction orthogonal to the ridge is zero (i.e. a maximum or minimum) and the second derivative in the same direction is negative [Ebe96]. Our solution is to find points on the edges of the polygons that satisfy these tests

Since our underlying gradient function $\nabla I(x)$ is defined as piecewise constant over the faces of the mesh, we detect if an edge has a zero first derivative by estimating gradient vectors at the end-points of each edge. This gradient vector

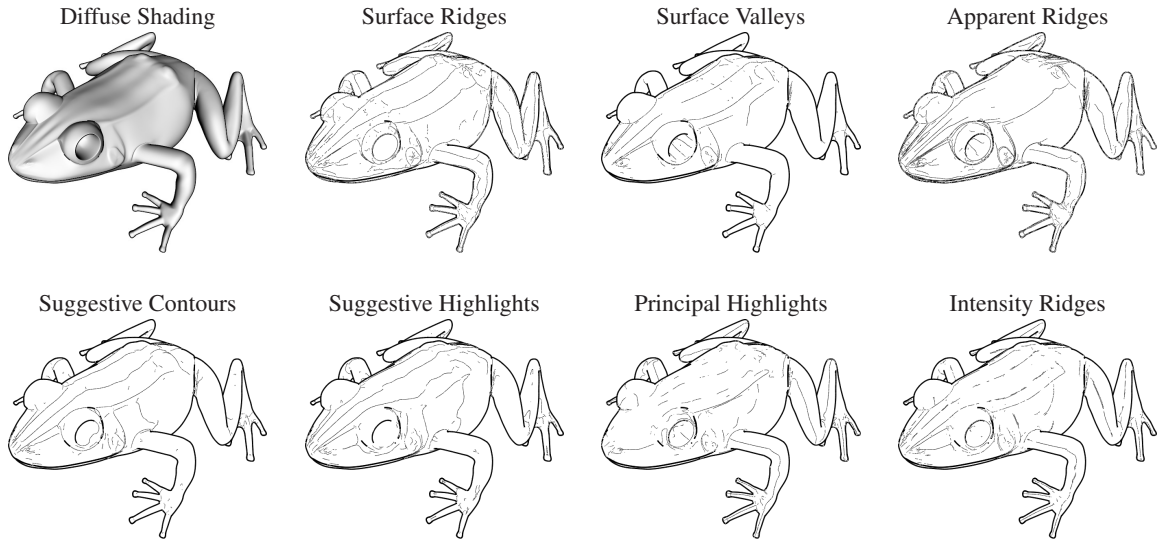

Figure 8: *Side-by-side comparison of ridges detected in the gradient field by our algorithm to other line drawing methods.*

G_i can be estimated as

$$G_i = \frac{\sum_{j=0}^{n} w_{i,j} * \nabla I_j}{\sum_{j=0}^{n} w_{i,j}}$$

where this sum is over the n adjacent faces in 1-ring neighborhood of the i^{th} vertex, ∇I_j is the gradient inside the j^{th} face and the weight $w_{i,j}$ for each face is based upon the angle subtended by the face as suggested by Max [Max99]. However, this computation for the vertex gradient vector becomes numerically unstable when the face gradients point in opposite directions. Such cases occur very frequently near the ridges where the face gradient vectors in the local neighborhood point in opposite directions. Since these cases are precisely those we wish to test, we need a more stable method for computing these gradient vectors.

In order to robustly compute a stable gradient direction for a given vertex, we compute the dominant eigenvector E_i of the covariance matrix built using the face gradients weighted by $w_{i,j}$ in 1-ring neighborhood of the vertex. These eigenvectors provide a stable, unoriented direction vector for the vertex gradient. However, this vector does not yet have a direction or a magnitude. Therefore, we compute the final vertex gradient at the vertex p_i by projecting G_i onto E_i.

$$\nabla I(p_i) = (G_i \cdot E_i) E_i$$

Given an edge with end-points p_1 and p_2, we estimate a potential ridge point p_r as suggested by Ohtake et al. [OBS04]

$$p_r = \frac{|\nabla I(p_2)| p_1 + |\nabla I(p_1)| p_2}{|\nabla I(p_2)| + |\nabla I(p_1)|}.$$

If the angle between $\nabla I(p_1)$ and $\nabla I(p_2)$ is less than 20 degrees, we exclude this point as a weak ridge point since the gradient vectors at the end-points are nearly in the same direction. For all other points, we must verify that these points are indeed part of a ridge by estimating the direction perpendicular to the ridge and computing the first and second derivatives of $I(x)$ in that direction.

Our solution is to check two different directions to see if either direction satisfies the first and second derivative tests. The first direction is given by the dominant eigenvector of the covariance matrix constructed from $\nabla I(p_1)$ and $\nabla I(p_2)$. The second direction is given by the dominant eigenvector of the covariance matrix constructed from $N_1 \times \nabla I(p_1)$ and $N_2 \times \nabla I(p_2)$. For each of these directions, we use a method similar to that in Judd et al. [JDA07] to verify a ridge exists.

Given a direction vector, we project the vector into the plane of each polygon containing the edge and intersect the ray with the edges of the triangles. Let p_f and p_b be the two intersected points. For each of these intersections, we linearly interpolate the vertex gradients at the end-points of the intersected edge to find the gradient at these points $\nabla I(p_f)$ and $\nabla I(p_b)$. We then numerically compute the second derivative as

$$\frac{\nabla I(p_f) \cdot (p_f - p_r) - \nabla I(p_b) \cdot (p_r - p_b)}{|p_f - p_r| + |p_r - p_b|}.$$

If this quantity is negative, then we check whether a zero crossing of the first derivative is possible in this direction by verifying that $\left(\nabla I(p_f) \cdot (p_f - p_r)\right)\left(\nabla I(p_b) \cdot (p_r - p_b)\right) < 0$. If either of the two direction vectors satisfy these derivative tests, then p_r is classified as a ridge point. We then draw ridges by processing each triangle and connecting the ridge points (if any) on the edges together with lines. Figure 8 shows an example of several types of lines commonly used to illustrate the shape of an object. While none of the lines match our intensity ridges, the union of Suggestive and Principal highlights closely approximate these ridges.

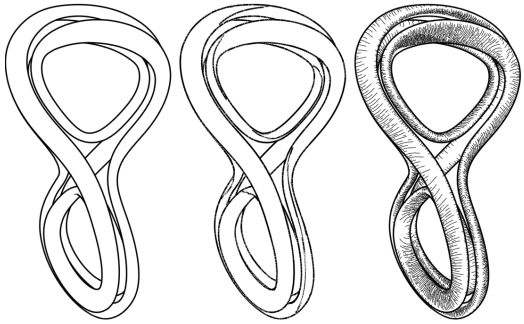

Figure 9: *Comparison of Suggestive Contours (left), Apparent Ridges (middle) and our shading strokes (right).*

5. Results

For convex, abstract shapes such as those in Figure 9, line drawing techniques such as Suggestive Contours [DFRS03] or Apparent Ridges [JDA07] may not be sufficient for the viewer to infer the shape of the object. Shading gives strong clues as to the objects shape and our method, through the adaptive placement of lines and orientation of those lines, helps impart this information to the viewer. For example, in Figure 10 (left), it is difficult to perceive which folds are convex or concave. Our shading strokes extends Suggestive Contours and illuminates the distinct folds in the cloth. Valleys of $I(x)$ closely approximate Suggestive Contours and, in the absence of a proximity check, our line drawing algorithm naturally creates these lines and emanate from them as shown in Figures 10. Figure 11 shows two additional examples of our method and also demonstrates that our technique can handle planar regions of the surface where principal curvature directions are not unique and may cause curvature-based methods to fail.

When the light is not colocated with the viewer, ridges/valleys of our gradient field no longer correspond to any sparse line drawing techniques. Moreover, $I(x)$ represents diffuse intensity and is typically clamped to zero when negative, which produces a gradient field of zero in shadow regions. We can modify our method to handle these situations by allowing $I(x)$ to be negative to define a consistent gradient field over the surface, even in shadow regions. For the purposes of determining density of lines in Equation 2, we simply set $I(x) = 0$. This modification allows us to move the light to arbitrary positions, but our lines will have a constant density in these shadow regions. This constant density may produce a noticeable banding effect and, hence, we prefer to keep the light at the viewer's position. Figure 12 demonstrates the effect of moving the light position.

Table 1 shows the timing of our method in seconds on an Intel Core 2 6700 and an Nvidia 8800 GTX using various models. Each of these examples were rendered on an 800×800 image with a maximum proximity depth of 20. Updating and truncating the lines using the proximity grid accounts for a very small fraction of the total time needed to draw these shading strokes since line rasterization is fast.

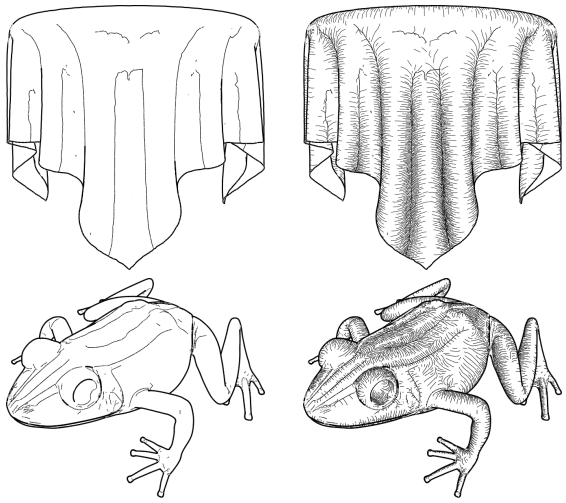

Figure 10: *Two examples of a draped table cloth (top) and a frog (bottom) emphasizing how shading strokes tend to emanate from suggestive contours.*

Model	Faces	Shading Strokes w/o ridge detection	Shading Strokes
Torus	4800	0.72	0.81
Cubehole	6491	0.80	0.82
Table Cloth	44636	0.74	0.88
Frog	82880	1.25	2.52
Brain	294012	1.04	2.53
Heptoroid	573440	1.67	2.44
Foot bone	735424	1.17	2.52

Table 1: *Timings for Shading Strokes measured in seconds. The seeding depth is limited to 18 and the proximity grid depth is 20. Intensity threshold is set to 1.0*

6. Future Work

Temporal coherence is an issue with our method that we have not addressed. Like Suggestive Contours, intensity ridges and valleys are relatively coherent as the object moves around. Our line drawing algorithm draws lines in a nested fashion with longer lines appearing first and shorter lines later to fill in darker regions of the object. We believe that by tracking the seed points of these longer lines between frames that we can maintain temporal coherence and would like to explore this area in the future.

Also, the speed of our method does not produce interactive frame rates for large models. Rasterizing lines into the proximity grid is fast. However, our ridge detection algorithm is about twice as slow as detecting both Suggestive/Principal Highlights and we may be able to modify the method to increase its speed.

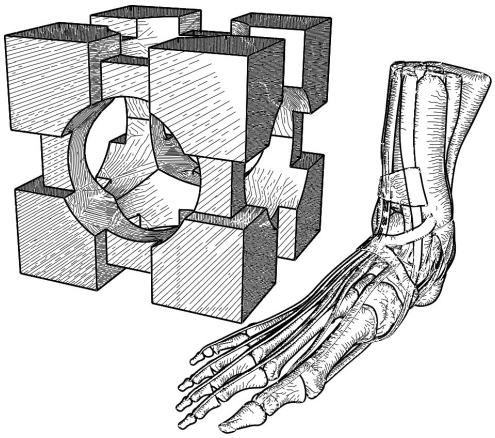

Figure 11: *Examples of models rendered with our method.*

Acknowledgments

We would like to thank Ergun Akleman and Forrester Cole for the models used in the paper. This work was supported in part by NSF grant CCF-07024099.

References

[CGL*08] COLE F., GOLOVINSKIY A., LIMPAECHER A., BARROS H. S., FINKELSTEIN A., FUNKHOUSER T., RUSINKIEWICZ S.: Where do people draw lines? *ACM Trans. Graph. 27*, 3 (2008). 1, 2

[DFR04] DECARLO D., FINKELSTEIN A., RUSINKIEWICZ S.: Interactive rendering of suggestive contours with temporal coherence. In *Proceedings of NPAR* (2004), pp. 15–145. 2

[DFRS03] DECARLO D., FINKELSTEIN A., RUSINKIEWICZ S., SANTELLA A.: Suggestive contours for conveying shape. *ACM Trans. Graph. 22*, 3 (July 2003), 848–855. 1, 2, 5, 7

[DHR*99] DEUSSEN O., HAMEL J., RAAB A., SCHLECHTWEG S., STROTHOTTE T.: An illustration technique using hardware-based intersections and skeletons. In *Proceedings Graphics interface* (1999), pp. 175–182. 2

[DKG05] DONG S., KIRCHER S., GARLAND M.: Harmonic functions for quadrilateral remeshing of arbitrary manifolds. *Comput. Aided Geom. Des. 22*, 5 (2005), 392–423. 3

[DR07] DECARLO D., RUSINKIEWICZ S.: Highlight lines for conveying shape. In *Proceedings of NPAR* (2007), pp. 63–70. 2

[Ebe96] EBERLY D.: *Ridges in Image and Data Analysis*. Springer, 1996. 5

[GIHL00] GIRSHICK A., INTERRANTE V., HAKER S., LEMOINE T.: Line direction matters: an argument for the use of principal directions in 3d line drawings. In *Proceedings of NPAR* (2000), pp. 43–52. 2

[HZ00] HERTZMANN A., ZORIN D.: Illustrating smooth surfaces. In *Proceedings of SIGGRAPH* (2000), pp. 517–526. 1, 2

[JDA07] JUDD T., DURAND F., ADELSON E. H.: Apparent ridges for line drawing. *ACM Trans. Graph. 26*, 3 (2007), 19. 1, 2, 6, 7

[JL97] JOBARD B., LEFER W.: Creating Evenly-Spaced Streamlines of Arbitrary Density. In *Proceedings of Eurographics Workshop on Vis. in Sci. Computing* (1997), pp. 45–55. 2

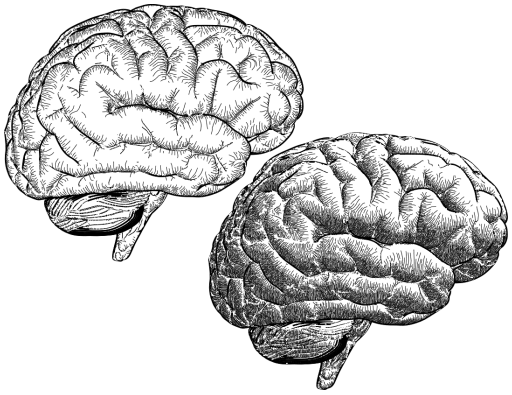

Figure 12: *Model rendered with the light colocated with the viewer (top left) and in arbitrary position (bottom right).*

[JL01] JOBARD B., LEFER W.: Multiresolution flow visualization. In *Proceedings of WSCG* (2001), pp. 33–37. 2

[LMB04] LI G., MA W., BAO H.: $\sqrt{2}$ subdivision for quadrilateral meshes. *The Visual Computer 20*, 4 (2004), 180–198. 4

[MAD05] MEBARKI A., ALLIEZ P., DEVILLERS O.: Farthest point seeding for efficient placement of streamlines. *Proceedings of IEEE Vis.* (Oct. 2005), 479–486. 2, 3, 4

[Max99] MAX N.: Weights for computing vertex normals from facet normals. *J. Graph. Tools 4*, 2 (1999), 1–6. 6

[MKG*97] MARKOSIAN L., KOWALSKI M. A., GOLDSTEIN D., TRYCHIN S. J., HUGHES J. F., BOURDEV L. D.: Real-time nonphotorealistic rendering. In *Proceedings of SIGGRAPH* (1997), pp. 415–420. 1

[OBS04] OHTAKE Y., BELYAEV A., SEIDEL H.-P.: Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph. 23*, 3 (2004), 609–612. 6

[PFH00] PRAUN E., FINKELSTEIN A., HOPPE H.: Lapped textures. In *Proceedings SIGGRAPH* (2000), pp. 465–470. 2

[PHWF01] PRAUN E., HOPPE H., WEBB M., FINKELSTEIN A.: Real-time hatching. In *Proceedings of SIGGRAPH* (2001), pp. 581–586. 2

[PR97] PETERS J., REIF U.: The simplest subdivision scheme for smoothing polyhedra. *ACM Trans. Graph. 16*, 4 (1997), 420–431. 4

[Ram88] RAMACHANDRAN V. S.: Perception of shape from shading. *Nature* (1988), 163 – 166. 1

[RCDF08] RUSINKIEWICZ S., COLE F., DECARLO D., FINKELSTEIN A.: Line drawings from 3d models. In *ACM SIGGRAPH course notes* (2008), pp. 1–356. 2

[SLZC09] SPENCER B., LARAMEE R., ZHANG E., CHEN G.: Evenly-spaced streamlines for surfaces: An image-based approach. In *Computer Graphics Forum* (2009). 3

[WS96] WINKENBACH G., SALESIN D. H.: Rendering parametric surfaces in pen and ink. In *Proceedings of SIGGRAPH* (1996), pp. 469–476. 2

[YZX*04] YU Y., ZHOU K., XU D., SHI X., BAO H., GUO B., SHUM H.-Y.: Mesh editing with poisson-based gradient field manipulation. In *ACM Trans. Graph.* (2004), pp. 644–651. 3

[ZISS04] ZANDER J., ISENBERG T., SCHLECHTWEG S., STROTHOTTE T.: High quality hatching. In *Computer Graphics Forum* (2004), vol. 23, pp. 421–430. 2