Interactive Rendering of Suggestive Contours with Temporal Coherence

Doug DeCarlo
Rutgers University

Adam Finkelstein
Princeton University

Szymon Rusinkiewicz
Princeton University

Abstract

Line drawings can convey shape using remarkably minimal visual content. Suggestive contours, which are lines drawn at certain types of view-dependent surface inflections, were proposed recently as a way of improving the effectiveness of computer-generated line drawings. This paper extends previous work on static suggestive contours to dynamic and real-time settings. We analyze movement of suggestive contours with respect to changes in viewpoint, and offer techniques for improving the quality of strokes rendered for a moving camera. We describe practical algorithms for rendering drawings with contours and suggestive contours at interactive rates. Finally, we discuss techniques for improving the visual appearance of suggestive contours, in both the static and dynamic cases.

Keywords: non-photorealistic rendering, line drawings, silhouettes, contours, differential geometry, graphics hardware

1 Introduction

Line drawings based on feature lines such as contours and creases form the basis for many styles of non-photorealistic rendering (NPR). DeCarlo et al. [2003] recently augmented the suite of available line types by introducing *suggestive contours*—feature lines drawn along zero crossings of radial curvature. As shown in Figure 1(a), suggestive contours blend visually with true contours and help convey 3D shape.

One way to think of suggestive contours is that they are *contours in nearby views*. We might therefore deduce that they enjoy a natural temporal coherence under dynamic viewing conditions: suggestive contours in the current view are likely to resemble those in nearby views. For example, the suggestive contours in Figures 1(a) and (b) are similar because the views are similar. If this is generally true, they should be amenable to rendering with a changing camera, making them suitable for both offline animation and real-time NPR. However, to date there has been no formal analysis of the stability of suggestive contours in the dynamic setting, so we do not know the conditions under which parts of suggestive contours are more or less stable. Furthermore, current algorithms require traversal of the entire model in order to detect and render them—a potential challenge for real-time applications.

This paper presents theoretical and algorithmic modifications that extend suggestive contour rendering to dynamic and real-time settings. Section 2 offers an analysis of the stability of suggestive contours under dynamic viewing conditions, together with observations about where they are most likely to be located on the surface. We show that most visible regions of suggestive contours are stable under animation: they drift slowly over the surface for small camera changes, as seen in Figure 1(c). However, some regions are fleeting: they move quickly over the surface for small camera changes. These regions are obvious candidates to be discarded or faded away.

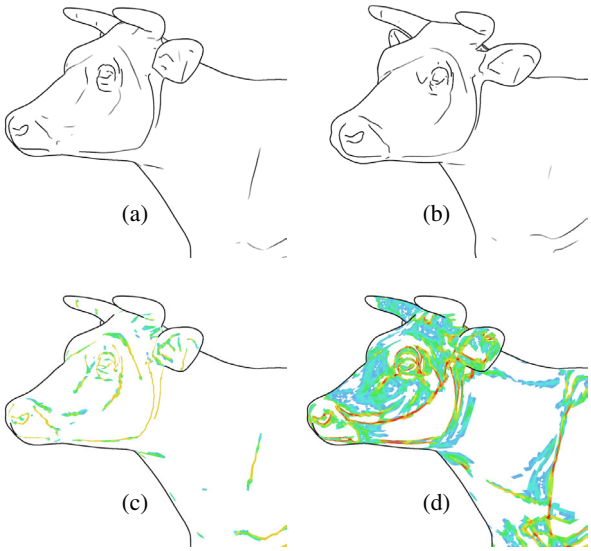


Figure 1: Suggestive contours in a dynamic scene: (a) contours and suggestive contours seen from view A; (b) seen from view B; (c) distribution of faces touched by suggestive contours as camera moves from A to B; and (d) distribution of faces touched for views uniformly distributed over the sphere.

Informed by this analysis, Section 3 steps through the suggestive contour extraction pipeline, describing algorithmic modifications that improve speed and quality. These techniques are largely adapted from or inspired by methods already described in the literature for efficient computation or rendering of true contours, for example by stochastic search [Markosian et al. 1997], hierarchical culling [Gooch et al. 1999; Hertzmann and Zorin 2000; Sander et al. 2000], or hardware approaches (e.g. [Raskar and Cohen 1999]). Finally, Section 4 concludes with some areas for future work.

2 Stability and Distribution

A key challenge for various styles of NPR in dynamic settings is maintaining *temporal coherence*. In this section we analyze the stability of suggestive contours with respect to changes in viewpoint, with the intent of understanding when renderings of suggestive contours will exhibit good coherence. We present both qualitative and quantitative observations about the conditions under which suggestive contours move slowly, and observe that there is a strong correlation between the (view-dependent) locations of suggestive contours and the (view-independent) locations at which the surface's Gaussian curvature is zero. We conclude the section with statistical surveys of the distribution of suggestive contours.

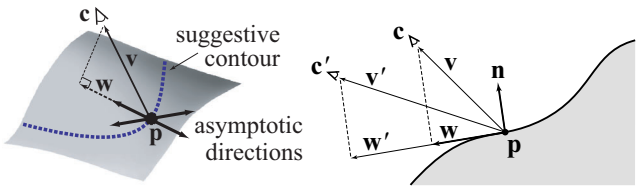


Figure 2: (a) $\mathbf{w}$ is the projection of the view direction $\mathbf{v}$ onto the tangent plane. At a suggestive contour, $\mathbf{w}$ points in one of the asymptotic directions of the surface—directions in which the curvature is zero. (b) Suggestive contours do not move when the camera motion is in the radial plane (from $\mathbf{c}$ to $\mathbf{c}'$) because the direction of $\mathbf{w}$ does not change (only the length of $\mathbf{w}'$ is different, not its direction)

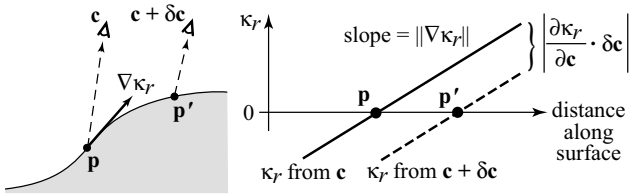


Figure 3: Suggestive contours and camera motion. (a) In a normal slice of the surface that contains $\nabla\kappa_r$ (but not the camera), as the camera moves from $\mathbf{c}$ to $\mathbf{c}+\delta\mathbf{c}$, a point on the suggestive contour slides from $\mathbf{p}$ to $\mathbf{p}'$. (b) The distance $\mathbf{p}$ travels along the surface in the direction of $\nabla\kappa_r$ depends on two factors: κ_r is offset by $\partial\kappa_r/\partial\mathbf{c}\cdot\delta\mathbf{c}$ due to the camera motion $\delta\mathbf{c}$; and the slope of these lines denotes the local gradient magnitude of κ_r. These factors are the numerator and denominator, respectively, of equation (1).

The remainder of this paper assumes a basic familiarity with suggestive contours—a summary is presented in Appendix B. The first definition of suggestive contours is the most relevant: locations where the radial curvature is zero ($\kappa_r = 0$) and its derivative along the projected view direction $\mathbf{w}$ is positive ($D_\mathbf{w}\kappa_r > 0$). Portions of this section also make use of concepts from the differential geometry of surfaces, briefly described in Appendix A, but our main conclusions do not require a detailed understanding of this mathematical background.

2.1 Motion of Suggestive Contours

Suggestive contours are view dependent; they can slide along the surface when the viewpoint changes. This, however, occurs only for some camera motions. For example, in Figure 2(b), as the camera moves from $\mathbf{c}$ to $\mathbf{c}'$ the suggestive contour at $\mathbf{p}$ does not move because the direction of $\mathbf{w}$ does not change. In contrast, any motion of the camera out of the plane of this picture—which changes $\mathbf{w}$—causes the suggestive contour to move.

When the suggestive contour moves, how do we associate points on the curve before the motion with those on the curve afterward? (This question is explored for true contours by Kalnins et al. [2003].) Of all possible ways of tracking suggestive contour curves over time, we begin by assuming that the velocity of points on the curve is perpendicular to the curve itself (on the surface). Our rationale is that on the interior of the curve, the tangential component of velocity will not be observable from one frame to the next and can therefore be safely ignored. Since the curves lie along the zero set of κ_r, its gradient $\nabla\kappa_r$ will be perpendicular to the curve and lie in the tangent plane at such points. Therefore, the direction of movement is along $\nabla\kappa_r$. (Although $\nabla\kappa_r$ can be computed numerically, analytic expressions for it are derived in Appendix C.) Next we find the speed (signed magnitude of velocity).

We first discuss qualitatively how the local geometry contributes to this speed. Consider a point $\mathbf{p}$ located on a suggestive contour. As shown in Figure 2(a), this can only occur when the surface at $\mathbf{p}$ is viewed along one of its asymptotic directions. As the view changes, the suggestive contour will move to a nearby point at which one of the asymptotic directions points along the new projected view direction. If asymptotic directions vary slowly across the surface, the suggestive contour will tend to travel *further* to reach a point with the required asymptotic direction. We therefore expect that the speed of suggestive contours will be highest in areas where the asymptotic directions vary slowly.

2.2 Speed of Motion

Let us now quantify the intuition developed above by writing down the formula for the speed of a moving suggestive contour. We derive the speed using the implicit function theorem [Munkres 1991], which explains how an implicit function varying over time induces motion in its iso-contours. In our case, the implicit function is κ_r, and we are interested in the motion of its zeros. The velocity at a point $\mathbf{p}$ for a small camera motion $\delta\mathbf{c}$ can be shown to be

$$v_{\mathrm{sc}} = -\left(\frac{\partial\kappa_r}{\partial\mathbf{c}}\cdot\delta\mathbf{c}\right)\Big/\|\nabla\kappa_r\|. \tag{1}$$

The numerator of this expression is the change in κ_r given a camera motion $\delta\mathbf{c}$: if κ_r changes more quickly, we expect the velocity of motion to be higher. The denominator[1] is the "steepness" of the zero crossing (Figure 3): a shallow zero crossing will cause *more* motion of the suggestive contour. This observation corroborates the intuitive argument in the end of Section 2.1.

While it would be possible to compute this velocity numerically (for a particular $\delta\mathbf{c}$), we prefer the use of analytic expressions for the derivatives in the above; they are derived in Appendix C. In particular, $\partial\kappa_r/\partial\mathbf{c}$ is a vector in the tangent plane that is perpendicular to $\mathbf{w}$. Thus, we see that for the speed of a suggestive contour to be nonzero, $\delta\mathbf{c}$ must not lie in the radial plane; and the greatest speed will be achieved when $\delta\mathbf{c}$ is perpendicular to both $\mathbf{w}$ and $\mathbf{n}$.

If we are interested in the maximum magnitude of v_{sc} over any unit-length direction of camera change, the speed is simply:

$$\max|v_{\mathrm{sc}}| = \left\|\frac{\partial\kappa_r}{\partial\mathbf{c}}\right\|\Big/\|\nabla\kappa_r\| \tag{2}$$

Multiplying this quantity by $\|\mathbf{v}\|$ gives the maximum speed for an angular camera change (about $\mathbf{p}$). Scaling it by $\cos\theta$ (where θ is the angle between $\mathbf{n}$ and $\mathbf{v}$) gives the projected velocity.

One could determine with a similar computation the speed of the ending points of suggestive contours (where $D_\mathbf{w}\kappa_r = 0$) in a the direction of $\mathbf{w}$ (which is tangent to the curve at such points). While the effects of such growing and lengthening of suggestive contours are indeed visible in animations, computing the speed of this motion would involve fourth derivatives of a surface, which are difficult to use effectively. Therefore, as described in Section 3.4, we fade strokes out gradually as $D_\mathbf{w}\kappa_r$ approaches zero to remove any visual abruptness that might result from this kind of motion.

[1] Had we in Section 2.1 defined the velocity of points on the suggestive contour to be along $\mathbf{w}$ (in the radial plane) rather than along $\nabla\kappa_r$, the denominator would instead be $|D_\mathbf{w}\kappa_r|/\|\mathbf{w}\|$.

Faces	8k	22k	97k	250k
Percentage of faces with some $K < 0$	50%	33%	57%	65%
Percentage with s. c. in some view	37%	24%	45%	60%

Table 1: Performance of (view-independent) rejection of faces with positive Gaussian curvature ($K > 0$) at all vertices.

Faces	8k	22k	97k	250k
Avg. percentage of faces with s. c.	2%	2%	4%	10%
Avg. percentage of faces with frontfacing s. c.	0.2%	0.5%	1.6%	4%

Table 2: Effect of (view-dependent) backface culling on suggestive contour extraction, averaged over many (evenly-distributed) viewpoints. Notice that a surprisingly large fraction of faces containing suggestive contours (60-90%) are backfacing, suggesting that backface culling is strongly effective and should be performed as early as possible in the suggestive contour extraction algorithm.

2.3 Suggestive Contours and Parabolic Lines

We find that stable suggestive contours are often located near the parabolic lines—curves where the Gaussian curvature K is zero. In Figure 4 left, note how the presence of suggestive contours is most likely (considered across all viewpoints) near parabolic lines (dark grey lines in the right image). The suggestive contour actually touches parabolic lines at points where $\mathbf{w}$ aligns with the principal curvature direction whose curvature is zero. Furthermore, at such points, the suggestive contour is tangent to the parabolic line, and is very stable (its speed is zero). See Appendix D for details.

2.4 Distribution of Suggestive Contours

When considering whether a given face can contain a piece of a suggestive contour, there are a few tests we can apply to reject many faces without performing expensive computation. Here we consider two simple examples: rejecting locations where $K > 0$ and backface culling. The benefit of such strategies becomes clear by studying the distribution of suggestive contours on typical models.

One of the simplest strategies is to trivially reject all faces that have positive Gaussian curvature at all three vertices. The reasoning is simple: because radial curvature κ_r must be between the principal curvatures, zeros of κ_r can only occur if the principal curvatures have opposite sign. Thus, areas of positive Gaussian curvature (for which the principal curvatures have the *same* sign) may not contain suggestive contours. Such areas are colored white in Figure 4, right. Because this test is view-independent, it is possible as a preprocess to extract only those faces of the mesh that have negative Gaussian curvature at some vertex, and only loop over those at each frame.

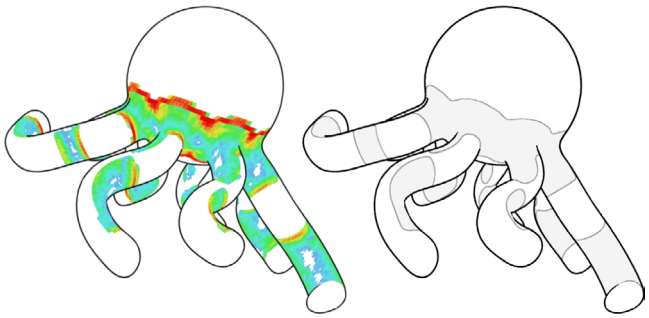


Figure 4: Comparison of density of suggestive contours considered over all viewpoints (*left*) with lines of zero Gaussian curvature, drawn in dark grey (*right*). Colors on left correspond to legend in Figure 1. Light grey shading on right indicates regions of negative Gaussian curvature. Note that suggestive contours can only occur in areas with $K < 0$, but most frequently appear where K is near 0. White regions (never touched) correspond with areas where $K > 0$.

Table 1 shows the benefit of rejecting faces where the Gaussian curvature is positive: we are able to trivially reject (offline or upon startup) between one third and one half of the faces of the model.

A second trivial test is to reject faces oriented away from the viewer. In contrast with contours, suggestive contours are always drawn on faces clearly oriented towards the viewer. It is therefore safe to perform backface culling and, as shown in Table 2, it effectively eliminates a large number of faces from consideration. Surprisingly, while we expect half the faces in the model to be backfacing, it turns out that a larger fraction (60-90%) of faces with suggestive contours are backfacing, so the payoff for this test is substantial. Figure 5 shows an extreme example: roughly 90% of the suggestive contours on this torus model are backfacing. A potential avenue for future work is understanding the class of 3D shapes for which such biases occur.

3 Efficient and High-Quality Extraction

We now consider the problem of visualizing suggestive contours on large models at interactive rates. The baseline algorithm considers each face in the mesh independently, evaluates κ_r at each vertex, interpolates to find zero crossings, and applies appropriate cutoff tests [DeCarlo et al. 2003]. We present modifications to each stage of this algorithm that improve the efficiency and quality of extracted and rendered strokes. Specifically, we consider techniques for not searching over all faces (Section 3.1), modifications to how lines are extracted within a face (Section 3.2), different strategies for trimming the extracted lines (Section 3.3), and a technique for fading lines to improve coherence (Section 3.4). Finally, we consider a fundamentally different rendering approach that exploits graphics hardware (Section 3.5).

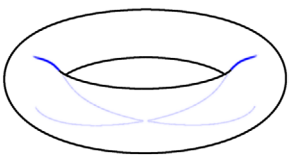


Figure 5: On many objects the backfacing suggestive contours are more prevalent than frontfacing ones. On this torus, the backfacing suggestive contours (shown in light blue) are collectively about ten times the length of the frontfacing ones (dark blue).

3.1 Testing Fewer Faces

The brute force method for extracting true contours is to traverse the entire mesh looking for zero crossings of $\mathbf{n} \cdot \mathbf{v}$. Researchers have investigated a variety of stochastic or hierarchical techniques for accelerating this process. For suggestive contours we also search for zero crossings (of κ_r in our case), but the same general strategies may be applied. This section compares a few approaches.

Randomized techniques: Markosian et al. [1997] proposed a stochastic algorithm for finding zero crossings of $\mathbf{n} \cdot \mathbf{v}$ on a mesh. Their method searched for zero crossings on edges of the mesh, but it was later adapted by Kalnins et al. [2002] to search over faces (producing cleaner loops). The same approach can be applied to suggestive contours. First, we choose a random face in the mesh and check for a zero crossing of κ_r. If one is found, we traverse the mesh, following the loop until we find a face that has already been visited. Next we repeat, choosing another face at random. This is repeated for a small, constant fraction of the faces of the mesh.

Markosian et al. proved that for any acceptably-small probability of missing a face, a constant fraction of the number of faces of the mesh may be tested (and that this fraction scales with the square root of the number of faces in the mesh). Thus, for sufficiently-large meshes the extra cost of traversing the mesh by following loops will be outweighed by the benefit of not visiting all faces. Their proof applies in the case of suggestive contours as well. They also propose a simple improvement that greatly improves the performance of the algorithm: in addition to checking random faces, also check a subset of the faces for which (suggestive) contours were found in the previous frame. This works well for suggestive contours as we have found that they are quite stable on the mesh from one frame to the next.

In addition, we have found that since more than half of the suggestive contour is backfacing, we get better performance if we discontinue traversal when a loop turns to be backfacing, in which case we have to walk both directions along the zero-set because we do not know whether we will make it around the entire loop. (This test cannot be applied to contours as they lie on the boundary of front and backfacing portions of the mesh.)

Finally, we have found that we get a slight performance benefit if, when testing random faces, we "walk" in the direction in which we would expect to find a suggestive contour. This is achieved as follows. Suppose we find that a randomly chosen face (or one chosen from the previous frame) fails to contain the zero crossing. In that case all three of the vertices have either positive or negative κ_r. Without loss of generality assume they are positive. We know that the zero crossing must be somewhere "downhill" and that the steepest descent will be in the direction of $-\nabla\kappa_r$. We want to walk from face to face in this direction, which we coarsely estimate by checking κ_r on the three opposite vertices on neighboring faces and choosing the face with the smallest value. We continue until either we find the zero crossing or we have exceeded a maximum traversal count.

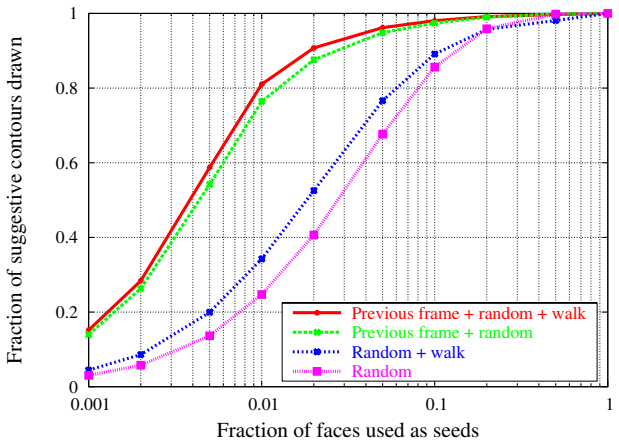


Figure 6: Randomized techniques can reduce the number of faces that need to be tested for suggestive contours.

Figure 6 contains a plot showing how many of the suggestive contours are drawn (not missed) when a given fraction of the overall number of faces are checked. This data is for a rotation of the cow model in which consecutive frames are roughly 1 degree of rotation apart. The *random* curve is the basic randomized method with no embellishments. Adding the *walk* improves the result slightly, but a much greater improvement comes from checking the suggestive contours from the *previous frame*. This is not surprising, since the cow is rotating fairly slowly and therefore the suggestive contours have strong temporal coherence. When we move the cow quickly the coherence is not as strong and the improvement from checking previous frames is reduced; however, in such cases it is harder to observe temporal coherence anyway. Furthermore, even when a suggestive contour is missed in one frame it is likely to be discovered in the next one or two frames, so the "failure" case is that the suggestive contour appears one or a couple of frames late, which is difficult to notice. Finally, the combined effects are the topmost curve. Based on these data the benefit of walking probably does not justify the cost (of either the computation or the implementation) though there may be other situations where the tradeoffs favor walking.

Hierarchical techniques: Several hierarchical culling methods have been proposed for avoiding traversal of the entire mesh when looking for contours (e.g. [Gooch et al. 1999; Hertzmann and Zorin 2000; Sander et al. 2000]). These approaches may be adapted to work for suggestive contours as well.

We implemented a variation of the cone tree hierarchy used by Sander et al. [2000] to store cones of normals (which can then be used to quickly check if there might be a contour in a given subtree). In our case, rather than storing normals we store the asymptotic directions (Appendix B), so at each node we store a *pair* of cones. To aggregate nodes in order to build nodes interior to the tree we associate them by checking which pairing is closest (i.e., provides the narrowest cones).

In our experiments we found that typical meshes are modeled so coarsely that a significant fraction of the faces contain zero crossings of κ_r. In such cases the brute force approach (which simply traverses arrays in memory and achieves very good cache performance) is faster than the hierarchical method (which suffers from poor cache performance when traversing the tree). Therefore we abandoned this approach after attempting only a fairly naive implementation. Several strategies might ameliorate this problem. First, Sander et al. constructed trees where fairly large subtrees were collapsed into a single node using a compact data structure that produced good cache behavior at the leaves; perhaps such an approach would work with suggestive contours as well. Second, we believe that since significant portions of the loops are backfacing, perhaps a hybrid structure that allows for fast testing for *both* the asymptotic directions and whether these areas are front or back facing would help.

3.2 Interpolating for Better Approximations

Many of the computations for suggestive contours involve computing quantities at vertices of the mesh, and interpolating those values across the polygons. Examples include locating the points where $\kappa_r = 0$ and determining the value of $D_{\mathbf{w}}\kappa_r$ at such points. As these quantities do not vary linearly across the triangle, barycentric in-

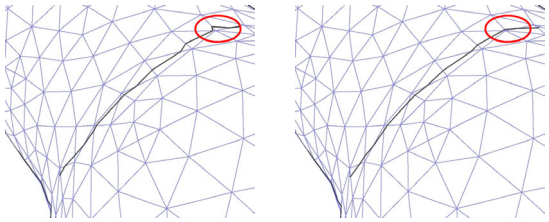


Figure 7: Low-resolution model with suggestive contours found using linear (left) or cubic Hermite (right) interpolation. The latter approach uses not only the value but also the gradient of κ_r at each vertex to improve the estimated locations of $\kappa_r = 0$.

terpolation methods will introduce errors. This section describes methods we found to be effective in alleviating such errors.

First is a more accurate approach for determining the mesh locations where $\kappa_r = 0$. This is particularly relevant when the triangles are large relative to the curvature variation. Determining these locations is easily accomplished for each mesh triangle by computing κ_r at each vertex, and assuming that it varies linearly across the triangle to find the zeros. This approach has been used to locate contours by interpolating values of $\mathbf{n} \cdot \mathbf{v}$ [Hertzmann and Zorin 2000]. Because suggestive contours are not viewed edge-on like contours are, errors introduced by the linearization can become quite visible. Derivatives of κ_r provide more information for how κ_r varies inside each triangle. To keep things simple, we still only find zeros on the polygon edges, and connect them with line segments. Furthermore, we only search for potential zero crossings on an edge when the signs of κ_r differ at its vertices. (This retains the ability to trivially reject polygons by checking for sign changes in κ_r across their vertices; but this causes problems that are discussed below.)

We start with an edge consisting of vertices $\mathbf{p}_0$ and $\mathbf{p}_1$. In terms of an interpolation parameter α, we compute the curvature $\kappa(\alpha)$ along this edge so that it agrees with κ_r and its derivatives as follows:

$$\begin{aligned} \kappa(0) &= \kappa_r(\mathbf{p}_0) \\ \kappa(1) &= \kappa_r(\mathbf{p}_1) \\ \kappa'(0) &= \nabla\kappa_r(\mathbf{p}_0) \cdot (\mathbf{p}_1 - \mathbf{p}_0) \\ \kappa'(1) &= \nabla\kappa_r(\mathbf{p}_1) \cdot (\mathbf{p}_1 - \mathbf{p}_0) \end{aligned}$$

We interpolate using Hermite interpolation, and determine its zeros using a binary search (i.e. the bisection method). Comparative results are shown in Figure 7, with a notable difference marked by a circle.

This approach has limitations, however. Hermite interpolation uses cubic basis functions, which might have three zero crossings, so that a triangle might contain more than one part of a suggestive contour. In our implementation, we sidestep this issue. When an edge is crossed three times, we only find the zero in the middle. This approach entirely misses cases with two zero crossings, as the signs at the vertices will not differ in that case. In practice, these limitations cause the loops of $\kappa_r = 0$ to move discontinuously—they might skip across one triangle. We have observed that this is infrequent, and typically happens where $D_{\mathbf{w}}\kappa_r$ is nearly zero, and so may have been discarded. Fixing this limitation is straightforward, but the implementation is complex (as it has many special cases).

Once the locations where $\kappa_r = 0$ are determined, certain quantities must be computed at these locations, such as $D_{\mathbf{w}}\kappa_r$ or the speed from equation (2), in order to isolate which of them are in fact suggestive contours. These quantities are computed at the vertices, and interpolated. We prefer not to use differential information for this—that would require fourth derivatives. Instead, care is taken to use expressions that will linearly interpolate effectively.

The derivations in Appendix C are general–they provide expressions for any value of κ_r. However, we are only interested in computing quantities such as $D_{\mathbf{w}}\kappa_r$ *on* a suggestive contour, where $\kappa_r = 0$. κ_r typically occurs in these expressions, and we can cancel the corresponding terms and interpolate the simplified versions. When we do this, the results improve in locations where the suggestive contours were heavily influenced by noise. We describe and analyze further possible simplifications based on the mathematics of suggestive contours in Appendix C. This approaches improves the results in noisy situations, but by no means eliminates the problem.

3.3 Trimming Moving Suggestive Contours

Fast derivative computation: One of the basic tests that must be applied to trim the full $\kappa_r = 0$ loops to just the suggestive contours is testing that $D_{\mathbf{w}}\kappa_r > 0$. That is, we must evaluate how the radial curvature changes in the direction $\mathbf{w}$. A simple, but slow, method is to evaluate this derivative numerically at each vertex, by comparing the computed κ_r to the value computed at the neighboring vertices [DeCarlo et al. 2003]. We propose a faster and more direct method based on evaluating $D_{\mathbf{w}}\kappa_r$ in terms of the derivative-of-curvature tensor $\mathbf{C}$, described in Appendix A.

Note that $\mathbf{C}(\mathbf{w},\mathbf{w},\mathbf{w})$ by itself is not the value we need: this gives the directional derivative in direction $\mathbf{w}$ of the normal curvature in the direction $\mathbf{w}$. However, it does not take into account the fact that $\mathbf{w}$ itself varies over the surface; thus, the expression for $D_{\mathbf{w}}\kappa_r$ has an additional term depending on the derivative of the projected view direction, introduced by the chain rule.

In Appendix C, it is shown that at suggestive contours, the derivative of radial curvature is just

$$\frac{D_{\mathbf{w}}\kappa_r}{\|\mathbf{w}\|} = \frac{\mathbf{C}(\mathbf{w},\mathbf{w},\mathbf{w})}{\|\mathbf{w}\|^3} + 2K\cot\theta \qquad (\text{where } \kappa_r = 0), \qquad (3)$$

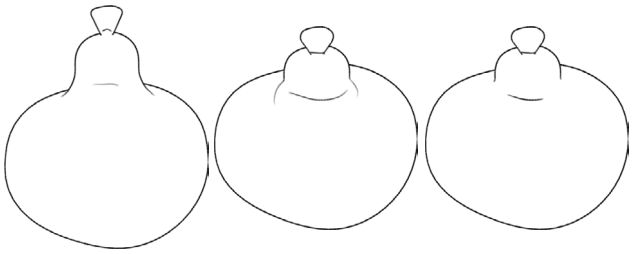
which depends only on $\mathbf{C}$, the Gaussian curvature $K = \kappa_1\kappa_2$, and θ—the angle between $\mathbf{n}$ and $\mathbf{v}$. The curvatures and θ are available, since they are necessary for contour and suggestive contour extraction. The tensor $\mathbf{C}$ is computed during preprocessing, using an algorithm analogous to that for computing curvature: just as $\mathbf{II}$ is estimated by considering the variation of the normal in different directions on the surface, $\mathbf{C}$ is estimated by considering the variation of $\mathbf{II}$ in different directions [Rusinkiewicz 2004]. Thus, $D_{\mathbf{w}}\kappa_r$ can be evaluated using a simple, local (per-vertex) computation at run time, and does not require an iteration over the neighboring vertices.

Other strategies for trimming suggestive contours: As mentioned earlier, the original implementation of suggestive contours uses two formulas for trimming the loops obtained as zeros of κ_r: a test on $D_{\mathbf{w}}\kappa_r/\|\mathbf{w}\|$ and a test on θ—the angle between $\mathbf{n}$ and $\mathbf{v}$. We have investigated other possible formulas for trimming suggestive contour loops. In particular, we examine the effects of combining the tests on $D_{\mathbf{w}}\kappa_r/\|\mathbf{w}\|$ and θ into a single test. We also investigate using a threshold on the speed of suggestive contour motion.

Before we begin, we can produce thresholds that work across a range of objects by normalizing for object size—curvatures are not scale invariant. We determine an approximate object scale by computing s as the median length of an edge in the mesh. If the mesh has been constructed economically, then s will indicate the "feature size" of the object. All curvatures (such as κ_r) are scaled by s, and derivatives of curvature (such as $D_{\mathbf{w}}\kappa_r$ or $\nabla\kappa_r$) by s^2.

Although the underlying justification for using thresholds on $D_{\mathbf{w}}\kappa_r/\|\mathbf{w}\|$ and θ are distinct, they can be productively combined into a single test, where suggestive contours are *retained* when:

$$\sin^2\theta\,\frac{D_{\mathbf{w}}\kappa_r}{\|\mathbf{w}\|} > t_d. \qquad (4)$$



(a) (b) (c)

Figure 8: Effect of adding cutoff based on stroke speed. (a) Contours and suggestive contours drawn from one view, using a cutoff based only on $D_{\mathbf{w}}\kappa_r$. (b) Another view: note that the suggestive contours have moved considerably relative to the model. (c) The same view as (b), but a cutoff based on suggestive contour speed has been applied.

The separate test on θ is simply encapsulated in the $\sin^2\theta$ term:

$$\sin^2\theta = \frac{\mathbf{w}\cdot\mathbf{w}}{\mathbf{v}\cdot\mathbf{v}} = 1 - \left(\frac{\mathbf{n}\cdot\mathbf{v}}{\|\mathbf{v}\|}\right)^2.$$

Empirically, we have found that no important control is lost with a single threshold—separating them is not required for obtaining cleaner results. When multiplied by $(\mathbf{v}\cdot\mathbf{v})\|\mathbf{w}\|$, this combined test is precisely equal to second derivatives of $\mathbf{n}\cdot\mathbf{v}$ towards $\mathbf{w}$ on a suggestive contour:

$$D_{\mathbf{w}}\left(D_{\mathbf{w}}(\mathbf{n}\cdot\mathbf{v})\right) = D_{\mathbf{w}}\left((\mathbf{w}\cdot\mathbf{w})\kappa_r\right) = (\mathbf{w}\cdot\mathbf{w})D_{\mathbf{w}}\kappa_r \quad (\text{where } \kappa_r = 0)$$

(see [DeCarlo et al. 2003] and Appendix C for details). This is quite relevant—recall how the second definition of suggestive contours identifies them as minima of $\mathbf{n}\cdot\mathbf{v}$ in the direction $\mathbf{w}$. When κ_r is scaled by $\mathbf{w}\cdot\mathbf{w}$, the resulting quantity is defined when $\mathbf{n}$ and $\mathbf{v}$ line up. At such locations, $(\mathbf{w}\cdot\mathbf{w})\kappa_r$ is zero. Nearby these locations, which is where θ is close to zero, the derivative of $(\mathbf{w}\cdot\mathbf{w})\kappa_r$ is likely to be small as $\|\mathbf{w}\|$ is small—this suggests the two tests can be merged effectively.

We further discard suggestive contours that move too quickly across the surface. This is easily accomplished by applying a threshold to equation (2), so that suggestive contours are retained when:

$$\|\mathbf{v}\| \max|v_{\text{sc}}| < t_v \tag{5}$$

The additional factor of $\|\mathbf{v}\|$ removes the dependence of v_{sc} on the distance to the camera, so that only angular motion of the camera matters. Using equation (2) indicates we are removing suggestive contours that move quickly across the surface under *any* camera motion. This is because any suggestive contour which is unstable with respect to small camera movements is not likely to convey shape effectively. Figure 8 shows the effectiveness of this extra cutoff at removing quickly-moving portions of suggestive contours.

It is also possible to take into account the actual camera motion $\delta\mathbf{c}$ (by using equation (1), for instance). However, this tends to produce isolated segments of suggestive contours—where $\mathbf{w}$ happens to be parallel to the camera motion. It also introduces another coherence problem whereby more suggestive contours appear as the camera decreases in speed.

Recall how suggestive contours are drawn from loops on the surface which are the solution of $\kappa_r = 0$. As the camera moves, these loops slide along the surface (with velocity given by equation (1)). The speed of these loops tends to be high in locations influenced by noise (the test in equation (2) typically discards these). Aside from this, the speed of points where the loops are splitting apart or merging together tends to be quite high as well. It is the suggestive contours at these locations that equation (5) eliminates. The end result is simply that suggestive contours are not drawn while splitting and merging, which would otherwise be a distraction in an animation.

3.4 Fading Strokes on Suggestive Contours

The previous section described tools for discarding suggestive contours that are unstable (with respect to small changes in viewpoint)—this greatly improves coherence in animations of suggestive contours. Its output is a series of *strokes*—sets of line segments that collectively form the suggestive contour. However, the use of a single threshold causes breaking-up and flickering of strokes over time; there are still problems with coherence.

Broken strokes (which can appear as dotted lines in extreme cases) were addressed by DeCarlo et al. [2003] using a post-processing phase that fills in small gaps and removes isolated segments using a method similar to hysteresis thresholding in Canny edge detection. While this produces reasonable results, it is not temporally coherent—large strokes appear or disappear across a single frame. One possible solution keeps track of strokes across viewpoint changes [Kalnins et al. 2003]. A much simpler approach varies the thickness or opacity of strokes over time [Freudenberg et al. 2002; Kowalski et al. 1999; Markosian et al. 2000; Praun et al. 2001].

The following describes a simple stroke fading scheme suitable for rendering suggestive contours. It computes a positive weight w for every point on the stroke—this weight is zero when the stroke should be drawn invisibly. Our approach varies the darkness of a stroke in a way that matches the tests in equation (4) and equation (5). For each stroke point, we compute how much it exceeded the threshold for each of the two tests–i.e., the difference between the right and left sides of equations (4) and (5). Then, w is simply the product of these values. The locations of where the cutoffs have been applied will be invisible, as they have $w = 0$. (We didn't find it necessary to introduce parameters to control this.)

Our implementation does not vary the thickness of the stroke—just its color. We compute the adjusted stroke color as blending between the background color (white) $[1,1,1]$ and the stroke color $[r,g,b]$ using the weight w:

$$\frac{[1,1,1] + w\cdot[r,g,b]}{1+w}$$

Figure 10 demonstrates unweighted (left) and weighted (center) renderings.

It would also be possible to ensure continuity in stroke weight at locations where a suggestive contour smoothly extends a contour. One simply constrains $w = \infty$ when $\mathbf{n}\cdot\mathbf{v} = 0$. We haven't found this to be necessary; the suggestive contours that cross the contours are quite stable at that location, and w is already sufficiently large.

3.5 Using Graphics Hardware

As an alternative to rendering suggestive contours by extracting them as strokes, we have investigated a rendering algorithm that uses the texture mapping capability of graphics hardware. To do this, we first set up a texture map as shown in Figure 9. The texture map is mostly white, with a thin stripe of black texels in the center. At run time, we compute texture coordinates such that the horizontal dimension is indexed by κ_r, with $\kappa_r = 0$ accessing texels near the center. The vertical dimension is indexed by $(\sin^2\theta)D_{\mathbf{w}}\kappa_r/\|\mathbf{w}\|$, which effectively implements the single-test cutoff described in Section 3.3. Note that the different mipmap levels for the texture map are constructed with a black stripe of the same width in each level, rather than by successively filtering down a single high-resolution texture map. This ensures that the rendered lines will have approximately constant width on the screen.

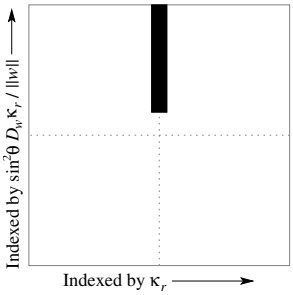


Figure 9: Texture map used to draw suggestive contours. We pass in κ_r and $(\sin^2\theta)D_{\mathbf{w}}\kappa_r/\|\mathbf{w}\|$ as the texture coordinates at each vertex, so that the black texels are used for κ_r near zero and $(\sin^2\theta)D_{\mathbf{w}}\kappa_r/\|\mathbf{w}\| > \varepsilon$.

The effect of our texture-mapped suggestive contour rendering (together with an analogous algorithm for rendering contours) is shown in Figure 10, right. For comparison, we show renderings of the same mesh using constant-weight and faded strokes, as described earlier. Because the bottleneck in all cases is computation of radial curvature and its derivative, we have observed rendering performance to be roughly equivalent in both cases. Running on a PC equipped with a 1.8 GHz P4 CPU and NVidia GeForce 3 graphics card we found the rendering frame rate for all the models we tested ranged between 400fps for the torus (8k faces) and 10fps for the David's head (250k faces).

Although we have implemented only the most basic and portable algorithm for graphics hardware-assisted suggestive contour rendering, we believe that variants that take full advantage of the capabilities of modern graphics cards would improve performance:

- Multitexturing could be used to combine the rendering of contours and suggestive contours, which are currently rendered in separate passes.
- Programmable vertex shaders could be used to compute radial curvature and its derivative in the GPU, given the location of the viewer and the precomputed (view-independent) values of **II** and **C**.
- Programmable fragment shaders could eliminate the need for texture lookup by directly comparing the interpolated κ_r and $D_{\mathbf{w}}\kappa_r$ to the correct thresholds. This might also provide better control over the screen-space width of the rendered lines.

4 Conclusions and Future Work

As presented previously, suggestive contours complement true contours as informative components for single-frame line drawings made from 3D models. In this work we analyze their stability with regard to a moving camera, and offer ways to improve temporal coherence and rendering performance in this dynamic setting. This investigation points toward several areas for future work:

Deforming geometry: Section 2.1 analyzes the stability of suggestive contours in view of a moving camera. This approach might be adapted to account for deforming geometry, a critical step in the use of suggestive contours for rendering animation. Furthermore, to implement a real-time system for drawing animated suggestive contours, algorithms remain to be found for efficient processing of the entire range of motion of the model over time.

Faster extraction and rendering: While we offer performance improvements for extraction and rendering of suggestive contours in interactive applications, a number of opportunities remain for further accelerating this process. Section 3.1 touches on a few algorithms for efficient extraction but also mentions some possible areas for improvement over these algorithms. Likewise, Section 3.5 describes our scheme for rendering suggestive contours using texture mapping and also points out some opportunities to move computation from the CPU into programmable hardware.

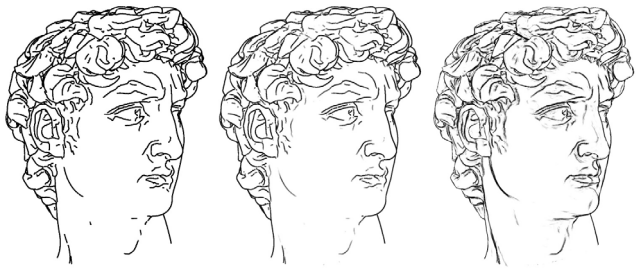


Figure 10: Different drawing styles produced by our interactive suggestive contour viewer. Left: constant-weight strokes. Center: faded strokes. Right: texture-mapped rendering. We find that the fading of strokes provided either explicitly (as at center) or implicitly due to interpolation within texture maps (as at right) helps the perceived frame-to-frame coherence.

Acknowledgments

Thanks to Spike Hughes, Rob Kalnins, Anthony Santella and Matthew Stone. This material is based upon work supported by National Science Foundation under the grant SGER 0227337 and by Intel Research Labs (AIM program).

References

CIPOLLA, R., AND GIBLIN, P. J. 2000. *Visual Motion of Curves and Surfaces*. Cambridge University Press.

DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive contours for conveying shape. *ACM Transactions on Graphics 22*, 3 (July), 848–855.

DO CARMO, M. P. 1976. *Differential Geometry of Curves and Surfaces*. Prentice-Hall.

FREUDENBERG, B., MASUCH, M., AND STROTHOTTE, T. 2002. Real-time halftoning: A primitive for non-photorealistic shading. In *Rendering Techniques 2002: 13th Eurographics Workshop on Rendering*, 227–232.

GOOCH, B., SLOAN, P., GOOCH, A., SHIRLEY, P., AND RIESENFELD, R. 1999. Interactive technical illustration. In *Proc. of the 1999 symposium on Interactive 3D graphics*, 31–38.

GRAVESEN, J., AND UNGSTRUP, M. 2002. Constructing invariant fairness measures for surfaces. *Advances in Computational Mathematics 17*, 67–88.

HERTZMANN, A., AND ZORIN, D. 2000. Illustrating smooth surfaces. In *Proceedings of ACM SIGGRAPH 2000*, 517–526.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. WYSIWYG NPR: Drawing strokes directly on 3D models. *ACM Transactions on Graphics 21*, 3 (July), 755–762.

KALNINS, R. D., DAVIDSON, P. L., MARKOSIAN, L., AND FINKELSTEIN, A. 2003. Coherent stylized silhouettes. *ACM Transactions on Graphics 22*, 3 (July), 856–861.

KOENDERINK, J. J., AND VAN DOORN, A. J. 1998. The structure of relief. *Advances in Imaging and Electron Physics 103*, 65–150.

KOENDERINK, J. J. 1984. What does the occluding contour tell us about solid shape? *Perception 13*, 321–330.

KOENDERINK, J. J. 1990. *Solid Shape*. MIT press.

KOWALSKI, M. A., MARKOSIAN, L., NORTHRUP, J. D., BOURDEV, L., BARZEL, R., HOLDEN, L. S., AND HUGHES, J. 1999. Art-based rendering of fur, grass, and trees. In *Proceedings of ACM SIGGRAPH 1999*, 433–438.

MARKOSIAN, L., KOWALSKI, M. A., TRYCHIN, S. J., BOURDEV, L. D., GOLDSTEIN, D., AND HUGHES, J. F. 1997. Real-time nonphotorealistic rendering. In *Proceedings of SIGGRAPH 1997*, Computer Graphics Proceedings, Annual Conference Series, 415–420.

MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., HOLDEN, L. S., NORTHRUP, J. D., AND HUGHES, J. F. 2000. Art-based rendering with continuous levels of detail. In *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering*, 59–66.

MORETON, H., AND SÉQUIN, C. 1992. Functional optimization for fair surface design. In *Proceedings of ACM SIGGRAPH 1992*, vol. 26, 167–176.

MUNKRES, J. 1991. *Analysis on Manifolds*. Addison-Wesley.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 579–584.

RASKAR, R., AND COHEN, M. F. 1999. Image precision silhouette edges. In *1999 ACM Symposium on Interactive 3D Graphics*, 135–140.

RUSINKIEWICZ, S. 2004. Estimating curvatures and their derivatives on triangle meshes. Tech. Rep. TR-693-04, Princeton University, Department of Computer Science.

SANDER, P. V., GU, X., GORTLER, S. J., HOPPE, H., AND SNYDER, J. 2000. Silhouette clipping. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 327–334.

WHELAN, J. C., AND VISVALINGAM, M. 2003. Formulated silhouettes for sketching terrain. In *Theory and Practice of Computer Graphics 2003*, 90–96.

A Differential Geometry

This section provides the necessary background from differential geometry to understand the formulas and derivations in this paper. For more details, consult [Cipolla and Giblin 2000; do Carmo 1976; Koenderink 1990]. Consider a smooth and closed surface S and a point $\mathbf{p} \in S$ sitting on the surface.

First-order structure The first-order approximation of this surface around this point is the tangent plane there; the unit normal vector $\mathbf{n}$ at $\mathbf{p}$ is perpendicular to this plane. (We use outward-pointing normal vectors.) Directions in the tangent plane at $\mathbf{p}$ can be described with respect to three-dimensional basis vectors $\{\mathbf{s}_u, \mathbf{s}_v\}$ that span the tangent plane. Generally, the three-dimensional vector $\mathbf{x}$ that sits in the tangent plane is written in local coordinates as $[u\ v]^{\mathrm{T}}$, where $\mathbf{x} = u\mathbf{s}_u + v\mathbf{s}_v$.

Second-order structure The unit normal $\mathbf{n}$ is a first-order quantity; it turns out that the interesting second-order structures involve derivatives of normal vectors. A *directional derivative* of a function defined on the surface (the unit normal being one possible function) specifies how that function changes as you move in a particular tangent direction. For instance, the directional derivative $D_{\mathbf{x}}\mathbf{n}$ at $\mathbf{p}$ characterizes how $\mathbf{n}$ "tips" as you move along the surface from $\mathbf{p}$ in the direction $\mathbf{x}$. (This same relationship can be conveyed by the differential $d\mathbf{n}(\mathbf{x})$, which describes how $\mathbf{n}$ changes as a function of a particular tangent vector $\mathbf{x}$.) Since derivatives of unit vectors must be in perpendicular directions, the derivatives of $\mathbf{n}$ lie in the tangent plane. This directional derivative can be written as:

$$D_{\mathbf{x}}\mathbf{n} = n_u\mathbf{s}_u + n_v\mathbf{s}_v, \tag{6}$$

where

$$\begin{bmatrix} n_u \\ n_v \end{bmatrix} = \begin{bmatrix} L & M \\ M & N \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}, \tag{7}$$

where the entries L, M and N depend on the local surface geometry—see [Cipolla and Giblin 2000] for details. Note that $D_{\mathbf{x}}\mathbf{n}$ depends linearly on the length of $\mathbf{x}$.

Now, suppose we have two tangent vectors $\mathbf{x}_1 = u_1\mathbf{s}_u + v_1\mathbf{s}_v$ and $\mathbf{x}_2 = u_2\mathbf{s}_u + v_2\mathbf{s}_v$. The *second fundamental form* $\mathbf{II}$ at $\mathbf{p}$ is a symmetric bilinear form specified by:

$$\begin{aligned} \mathbf{II}(\mathbf{x}_1, \mathbf{x}_2) &= (D_{\mathbf{x}_1}\mathbf{n}) \cdot \mathbf{x}_2 = (D_{\mathbf{x}_2}\mathbf{n}) \cdot \mathbf{x}_1 \\ &= \begin{bmatrix} u_1 & v_1 \end{bmatrix} \begin{bmatrix} L & M \\ M & N \end{bmatrix} \begin{bmatrix} u_2 \\ v_2 \end{bmatrix}. \end{aligned} \tag{8}$$

(This differs in sign from [do Carmo 1976] due to our choice of *outward* pointing normals.) Since $\mathbf{II}$ is symmetric, we use $\mathbf{II}(\mathbf{x})$ as a shorthand to indicate only one vector product has been performed; so $\mathbf{II}(\mathbf{x}) = D_{\mathbf{x}}\mathbf{n}$, and $\mathbf{II}(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{II}(\mathbf{x}_1) \cdot \mathbf{x}_2 = \mathbf{II}(\mathbf{x}_2) \cdot \mathbf{x}_1$.

The *normal curvature* of a surface S at a point $\mathbf{p}$ measures its curvature in a specific direction $\mathbf{x}$ in the tangent plane, and is defined in terms of the second fundamental form. The normal curvature, written as $\kappa_n(\mathbf{x})$, is:

$$\kappa_n(\mathbf{x}) = \frac{\mathbf{II}(\mathbf{x}, \mathbf{x})}{\mathbf{x} \cdot \mathbf{x}}.$$

Notice how the length and sign of $\mathbf{x}$ do not affect the normal curvature. On a smooth surface, the normal curvature varies smoothly with direction $\mathbf{x}$, and ranges between the principal curvatures κ_1 and κ_2 at $\mathbf{p}$. These are realized in their respective principal curvature directions $\mathbf{e}_1$ and $\mathbf{e}_2$, which are perpendicular and unit length.

The Gaussian curvature K is equal to the product of the principal curvatures: $K = \kappa_1\kappa_2$, and the mean curvature H is their average: $H = (\kappa_1 + \kappa_2)/2$. Wherever K is strictly negative (so that only one of κ_1 or κ_2 is negative), there are two directions along which the curvature is zero. These directions, called the *asymptotic directions*, play a central role in where suggestive contours are located.

The vector $D_{\mathbf{x}}\mathbf{n}$ can be broken into two components; in the direction of $\mathbf{x}$, and perpendicular to it. (This will be a useful tool in simplifying the analytic expressions derived in Appendix C.) The length of the component in the direction of $\mathbf{x}$ is simply the normal curvature κ_n. The length of the perpendicular component is known as the *geodesic torsion* τ_g, and describes how much the normal vector tilts to the side as you move in the direction of $\mathbf{x}$. If we define its perpendicular $\mathbf{x}_\perp$ as:

$$\mathbf{x}_\perp = \mathbf{n} \times \mathbf{x}$$

then we have:

$$D_{\mathbf{x}}\mathbf{n} = \mathbf{II}(\mathbf{x}) = \kappa_n(\mathbf{x})\mathbf{x} + \tau_g(\mathbf{x})\mathbf{x}_\perp \tag{9}$$

It follows that $\tau_g(\mathbf{x}) = \mathbf{II}(\mathbf{x}, \mathbf{x}_\perp)/\mathbf{x} \cdot \mathbf{x}$. Furthermore, when $\mathbf{x}$ is an asymptotic direction, $D_{\mathbf{x}}\mathbf{n}$ is perpendicular to $\mathbf{x}$ and it can be shown that $\tau_g^2(\mathbf{x}) = -K$ [Koenderink 1990].

Principal coordinates Using the principal directions $\{\mathbf{e}_1, \mathbf{e}_2\}$ as the local basis leads to *principal coordinates*. In principal coordinates, the matrix in equations (7) and (8) is diagonal with the principal curvatures as entries:

$$\begin{bmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{bmatrix}.$$

Given $[u\ v]^{\mathrm{T}} = [\cos\phi\ \sin\phi]^{\mathrm{T}}$, where ϕ is the angle measured between a particular direction and $\mathbf{e}_1$, this leads to the well-known Euler formula for normal curvature:

$$\kappa_n(\phi) = \kappa_1 \cos^2\phi + \kappa_2 \sin^2\phi$$

and the following for the geodesic torsion:

$$\tau_g(\phi) = (\kappa_2 - \kappa_1) \sin\phi \cos\phi$$

(where the sign of τ_g depends on our definition of $\mathbf{x}_\perp$, above).

Third-order structure In describing how suggestive contours move across the surface, we will need additional notation that describes *derivatives of curvature*. We use this to derive analytic expressions for the third-order quantities $D_{\mathbf{w}}\kappa_r$ and $\nabla\kappa_r$ in Appendix C. The gradient $\nabla\kappa_r$ is a three-dimensional vector in the tangent plane that locally specifies the magnitude and direction of maximal change in κ_r on the surface.

In the following, we use principal coordinates, as the third-order derivatives are much simpler to state. In this case, finding derivatives of normal curvatures involves taking the directional derivative of $\mathbf{II}$ in a particular tangent direction $\mathbf{x}$. The result is written in terms of a symmetric trilinear form $\mathbf{C}$, built from a $2\times 2\times 2$ (rank-3) tensor whose entries depend on the third derivatives of the surface [Gravesen and Ungstrup 2002]. Such derivatives have been ingredients in measures of fairness for variational surface modeling [Gravesen and Ungstrup 2002; Moreton and Séquin 1992].

We write $\mathbf{C}$ with either two or three arguments—indicating how many times a vector is multiplied onto the underlying tensor. Thus, $\mathbf{C}(\mathbf{x},\mathbf{x})$ is a vector and $\mathbf{C}(\mathbf{x},\mathbf{x},\mathbf{x})$ is a scalar. The order of the arguments do not matter, as $\mathbf{C}$ is symmetric. In principal coordinates, the tensor describing $\mathbf{C}$ has 4 unique entries [Gravesen and Ungstrup 2002]:

$$P = D_{\mathbf{e}_1}\kappa_1,\ Q = D_{\mathbf{e}_2}\kappa_1,\ S = D_{\mathbf{e}_1}\kappa_2,\ \text{and}\ \ \mathrm{T} = \mathrm{D}_{\mathbf{e}_2}\kappa_2$$

This leads to the first-order approximation of the matrix in equation (8) towards $\mathbf{x} = u\mathbf{e}_1 + v\mathbf{e}_2$ as:

$$\begin{bmatrix} L & M \\ M & N \end{bmatrix} \approx \begin{bmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{bmatrix} + u\begin{bmatrix} P & Q \\ Q & S \end{bmatrix} + v\begin{bmatrix} Q & S \\ S & T \end{bmatrix} \quad (10)$$

(written with the tensor expanded into two matrices on the right to avoid cumbersome notation, already multiplied once by $[u\ v]^{\mathrm{T}}$.) Finally, we note how to compute the gradient and directional derivative of the normal curvature κ_n using $\mathbf{C}$:

$$\nabla\kappa_n(\mathbf{x}) = \frac{\mathbf{C}(\mathbf{x},\mathbf{x})}{\mathbf{x}\cdot\mathbf{x}} = \frac{g_u\mathbf{e}_1 + g_v\mathbf{e}_2}{\mathbf{x}\cdot\mathbf{x}},$$

where

$$\begin{bmatrix} g_u \\ g_v \end{bmatrix} = \begin{bmatrix} Pu^2 + 2Quv + Sv^2 \\ Qu^2 + 2Suv + Tv^2 \end{bmatrix}$$

and

$$\frac{D_{\mathbf{x}}\kappa_n(\mathbf{x})}{\|\mathbf{x}\|} = \frac{\mathbf{C}(\mathbf{x},\mathbf{x},\mathbf{x})}{\|\mathbf{x}\|^3} = \frac{Pu^3 + 3Qu^2v + 3Suv^2 + Tv^3}{\|\mathbf{x}\|^3}.$$

B Review of Suggestive Contours

This section provides a brief overview of suggestive contours; for a more complete exposition see DeCarlo et al. [2003]. Suggestive contours are view-dependent linear features on the surface of an object that effectively convey its shape in a line drawing. Suggestive contours are related to formulated silhouettes [Whelan and Visvalingam 2003] and are classified as cliff curves [Koenderink and van Doorn 1998].

Of particular relevance here is the *radial curvature* $\kappa_r(\mathbf{p})$—the normal curvature of the surface at $\mathbf{p}$ in the direction of $\mathbf{w}$ [Koenderink 1984; DeCarlo et al. 2003]. As shown in Figure 2(a), $\mathbf{w}$ is the (unnormalized) projection of the view vector $\mathbf{v}$ onto the tangent plane at $\mathbf{p}$. This defines the *radial plane*, which contains $\mathbf{p}$, $\mathbf{n}$ and $\mathbf{v}$ (and $\mathbf{w}$). From its definition it is easy to see that κ_r is view-dependent, and that it is undefined wherever $\mathbf{v}$ and $\mathbf{n}$ are parallel.

DeCarlo et al. offer three equivalent definitions for suggestive contours. The first definition is that suggestive contours are portions of the zero set of radial curvature κ_r where $D_{\mathbf{w}}\kappa_r > 0$. The zero set $\kappa_r = 0$ forms a set of closed loops on the surface, as shown in Figure 11(b). Suggestive contours, the portions of these loops satisfying the derivative test $D_{\mathbf{w}}\kappa_r > 0$, are shown in blue in the figure. They meet up with true contours (shown in green in the figure) at *ending contours*—where the contour turns away from the camera and becomes invisible. Where contours meet suggestive contours, they align with geometric tangent continuity in the image plane, as in Figure 11(a). Note that since $\kappa_r = 0$ at suggestive contours, the projected view direction $\mathbf{w}$ will align with one of the asymptotic directions, as in Figure 2(a).

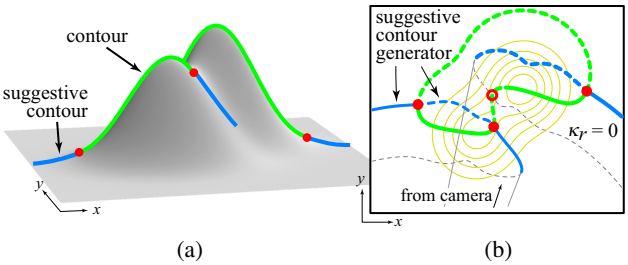


Figure 11: (a) Suggestive contours (shown in blue) extend the actual contours of the surface (shown in green). (b) A topographic view showing how the suggestive contour generators cross contours at the ending contours. The visible portions of the contour and suggestive contour are drawn solid.

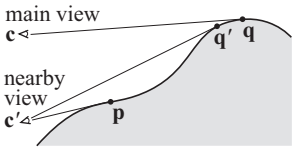


Figure 12: A situation showing both contours ($\mathbf{q}$) and suggestive contours ($\mathbf{p}$), as seen by the main viewpoint $\mathbf{c}$. As the viewpoint moves to $\mathbf{c}'$, a contour suddenly appears at $\mathbf{p}$, while the contour at $\mathbf{q}'$ slides along the surface from $\mathbf{q}$.

The second definition of suggestive contours is that they are positive minima of $\mathbf{n}\cdot\mathbf{v}$ in the view direction. This definition is motivated by the fact that true contours appear where this dot product is zero, so suggestive contours appear where the dot product is almost but not quite zero. We revisit this definition when trimming suggestive contours in Section 3.3.

The third definition is that suggestive contours are *contours in nearby views* that are not in correspondence with points on contours in any (radially) closer view. This is perhaps best understood with a picture, shown in Figure 12. The camera $\mathbf{c}$ sees a true contour at $\mathbf{q}$. From a nearby view $\mathbf{c}'$, however, the camera would see a contour at $\mathbf{p}$. Therefore, from the point of view of $\mathbf{c}$ we say that there is a suggestive contour at $\mathbf{p}$. Note that there is not a suggestive contour at $\mathbf{q}'$ because it is in direct correspondence with $\mathbf{q}$, whereas $\mathbf{p}$ is not in correspondence with a contour in any closer camera.

The above two definitions motivate two observable properties of suggestive contours in relation to true contours. First, suggestive contours visually *extend* true contours in the image plane, as shown in Figure 11(a). Second, suggestive contours *anticipate* true contours (that would appear in nearby views). These properties are of particular importance under animation, since suggestive contours appear in anticipation of contours that will appear soon, and visually blend with the true contours when they appear.

DeCarlo et al. [2003] propose two filters for suggestive contours in order to provide greater stability. First, recall that the radial curvature is undefined wherever the vectors $\mathbf{v}$ and $\mathbf{n}$ are parallel. When these vectors are close to parallel, the suggestive contour becomes unstable. Therefore, we discard regions of suggestive contour where the angle between these vectors is less than a

threshold θ_c. Furthermore, estimating the curvatures themselves can be subject to noise in the mesh and can lead to spurious zero crossings of κ_r. Therefore, the second filter applies a small positive threshold in the derivative test, ensuring $D_{\mathbf{w}}\kappa_r/\|\mathbf{w}\| > t_d$. We have found empirically that these two filters taken in combination provide a great deal of stability to suggestive contours in the dynamic setting. Nonetheless, a proper analysis of the stability of suggestive contours is merited, and can lead to even greater temporal coherence.

C Derivations

In order to derive equation (3) and the components of equation (1), we begin with the definitions of the view direction $\mathbf{v}$ and its projection onto the tangent plane $\mathbf{w}$, in terms of a point $\mathbf{p}$ and camera position $\mathbf{c}$:

$$\begin{aligned}\mathbf{v} &= \mathbf{c}-\mathbf{p}\\ \mathbf{w} &= \mathbf{v}-\mathbf{n}(\mathbf{n}\cdot\mathbf{v})\end{aligned}$$

We may now find derivatives of these in an arbitrary direction $\mathbf{x}$ in the tangent plane; this just involves applying the product rule:

$$\begin{aligned}D_{\mathbf{x}}\mathbf{v} &= -\mathbf{x}\\ D_{\mathbf{x}}\mathbf{w} &= -\mathbf{x}-(D_{\mathbf{x}}\mathbf{n})(\mathbf{n}\cdot\mathbf{v})-\mathbf{n}\big((D_{\mathbf{x}}\mathbf{n})\cdot\mathbf{v}-\mathbf{n}\cdot\mathbf{x}\big)\\ &= -\mathbf{x}-(\mathbf{n}\cdot\mathbf{v})\mathbf{II}(\mathbf{x})-\mathbf{n}\big(\mathbf{II}(\mathbf{w},\mathbf{x})\big)\end{aligned}$$

Now, using the definition of κ_r, we have:

$$\begin{aligned}D_{\mathbf{x}}\kappa_r &= D_{\mathbf{x}}\frac{\mathbf{II}(\mathbf{w},\mathbf{w})}{\mathbf{w}\cdot\mathbf{w}}\\ &= \frac{(D_{\mathbf{x}}\mathbf{II})(\mathbf{w},\mathbf{w})+2\mathbf{II}(\mathbf{w},D_{\mathbf{x}}\mathbf{w})}{\mathbf{w}\cdot\mathbf{w}}-\frac{\mathbf{II}(\mathbf{w},\mathbf{w})}{(\mathbf{w}\cdot\mathbf{w})^2}D_{\mathbf{x}}(\mathbf{w}\cdot\mathbf{w})\\ &= \frac{\mathbf{C}(\mathbf{w},\mathbf{w},\mathbf{x})+2\mathbf{II}(\mathbf{w})\cdot D_{\mathbf{x}}\mathbf{w}-2\kappa_r\mathbf{w}\cdot D_{\mathbf{x}}\mathbf{w}}{\mathbf{w}\cdot\mathbf{w}}\end{aligned}$$

After substituting, simplifications are accomplished by using equation (9) where we define $\tau_r=\tau_g(\mathbf{w})$ as the *radial geodesic torsion*, and can use the following:

$$\mathbf{II}(\mathbf{w})=\kappa_r\mathbf{w}+\tau_r\mathbf{w}_\perp \tag{11}$$

to obtain

$$D_{\mathbf{x}}\kappa_r = \frac{\mathbf{C}(\mathbf{w},\mathbf{w},\mathbf{x})-2\tau_r(\mathbf{n}\cdot\mathbf{v})\mathbf{II}(\mathbf{w}_\perp)\cdot\mathbf{x}-2\tau_r\mathbf{w}_\perp\cdot\mathbf{x}}{\mathbf{w}\cdot\mathbf{w}}$$

Now, $\mathbf{II}(\mathbf{w}_\perp)$ can be expressed as:

$$\mathbf{II}(\mathbf{w}_\perp)=(2H-\kappa_r)\mathbf{w}_\perp+\tau_r\mathbf{w}$$

since the sum of any two curvatures measured in perpendicular directions is $2H$ (and hence $\kappa_n(\mathbf{w}_\perp)=2H-\kappa_r$). This leads to:

$$D_{\mathbf{x}}\kappa_r=\frac{\mathbf{C}(\mathbf{w},\mathbf{w})-2\tau_r^2(\mathbf{n}\cdot\mathbf{v})\mathbf{w}-2\tau_r\left(1+(2H-\kappa_r)\mathbf{n}\cdot\mathbf{v}\right)\mathbf{w}_\perp}{\mathbf{w}\cdot\mathbf{w}}\cdot\mathbf{x}$$

We may now substitute in $\mathbf{w}$ for $\mathbf{x}$ to obtain

$$\begin{aligned}\frac{D_{\mathbf{w}}\kappa_r}{\|\mathbf{w}\|} &= \frac{\mathbf{C}(\mathbf{w},\mathbf{w},\mathbf{w})}{\|\mathbf{w}\|^3}-2\tau_r^2\frac{\mathbf{n}\cdot\mathbf{v}}{\|\mathbf{w}\|}+2(0)\\ &= \frac{\mathbf{C}(\mathbf{w},\mathbf{w},\mathbf{w})}{\|\mathbf{w}\|^3}-2\tau_r^2\cot\theta\end{aligned}$$

where $\theta=\cos^{-1}(\mathbf{n}\cdot\mathbf{v}/\|\mathbf{v}\|)$ is the angle between $\mathbf{n}$ and $\mathbf{v}$. When $\kappa_r=0$, we also have $\tau_r^2=-K$ and this simplifies to (3):

$$\frac{D_{\mathbf{w}}\kappa_r}{\|\mathbf{w}\|} = \frac{\mathbf{C}(\mathbf{w},\mathbf{w},\mathbf{w})}{\|\mathbf{w}\|^3}+2K\cot\theta$$

To find the speed of motion of suggestive contours, we need to know the magnitude of $\nabla\kappa_r$ and the derivative of radial curvature with camera motion $\partial\kappa_r/\partial\mathbf{c}$. Since the directional derivative can be also written as $D_{\mathbf{x}}\kappa_r=\nabla\kappa_r\cdot\mathbf{x}$, and $\nabla\kappa_r$ sits in the tangent plane, we have:

$$\nabla\kappa_r=\frac{\mathbf{C}(\mathbf{w},\mathbf{w})-2\tau_r^2(\mathbf{n}\cdot\mathbf{v})\mathbf{w}-2\tau_r\left(1+(2H-\kappa_r)\mathbf{n}\cdot\mathbf{v}\right)\mathbf{w}_\perp}{\mathbf{w}\cdot\mathbf{w}}$$

which when $\kappa_r=0$ reduces to:

$$\nabla\kappa_r=\frac{\mathbf{C}(\mathbf{w},\mathbf{w})+2K(\mathbf{n}\cdot\mathbf{v})\mathbf{w}-2\sqrt{-K}\big(1+2H(\mathbf{n}\cdot\mathbf{v})\big)\mathbf{w}_\perp}{\mathbf{w}\cdot\mathbf{w}} \tag{12}$$

The remaining piece of (1) is the derivative of radial curvature with respect to camera motion, $\partial\kappa_r/\partial\mathbf{c}$. This is an ordinary derivative (not a directional derivative or differential) with respect to a vector—a Jacobian. In this case, since κ_r is a scalar, this derivative is a vector. We start with the following:

$$\begin{aligned}\frac{\partial\mathbf{v}}{\partial\mathbf{c}} &= \mathbf{1}\\ \frac{\partial\mathbf{w}}{\partial\mathbf{c}} &= \mathbf{1}-\mathbf{n}(\mathbf{n}^{\mathrm{T}}\mathbf{1})=\mathbf{1}-\mathbf{n}\mathbf{n}^{\mathrm{T}},\end{aligned}$$

where $\mathbf{1}$ is the identity matrix. Notice that $\partial\mathbf{w}/\partial\mathbf{c}$ is the projection matrix for the tangent plane. We can now determine $\partial\kappa_r/\partial\mathbf{c}$, again simplifying using equation (9) to obtain:

$$\begin{aligned}\frac{\partial\kappa_r}{\partial\mathbf{c}} &= 2\frac{\frac{\partial\mathbf{w}}{\partial\mathbf{c}}\mathbf{II}(\mathbf{w})}{\mathbf{w}\cdot\mathbf{w}}-\frac{\kappa_r}{\mathbf{w}\cdot\mathbf{w}}\frac{\partial(\mathbf{w}\cdot\mathbf{w})}{\partial\mathbf{c}}\\ &= \frac{2\kappa_r\mathbf{w}+2\tau_r\mathbf{w}_\perp}{\mathbf{w}\cdot\mathbf{w}}-\frac{2\kappa_r}{\mathbf{w}\cdot\mathbf{w}}\frac{\partial\mathbf{w}}{\partial\mathbf{c}}\mathbf{w}\\ &= \frac{2\tau_r}{\mathbf{w}\cdot\mathbf{w}}\mathbf{w}_\perp\end{aligned}$$

and further that:

$$\left\|\frac{\partial\kappa_r}{\partial\mathbf{c}}\right\| = \frac{2\tau_r}{\|\mathbf{w}\|}$$

and when $\kappa_r=0$:

$$\left\|\frac{\partial\kappa_r}{\partial\mathbf{c}}\right\| = \frac{2\sqrt{-K}}{\|\mathbf{w}\|}. \tag{13}$$

D Points where $\kappa_r=0$ and $K=0$

Section 2.3 described a situation where the suggestive contours touch the parabolic lines at a point. This occurs wherever $\mathbf{w}$ lines up with the principal direction $\mathbf{e}_2$ (assuming here that $|\kappa_2|<|\kappa_1|$ and so $\kappa_1\neq 0$ and $\kappa_2=0$).

First, we prove that these two curves are tangent at such points by showing that their gradient directions are the same (not counting sign). We can compute ∇K using a method described in [Gravesen and Ungstrup 2002], which finds K using equation (10) and evaluates its gradient at $u=0$ and $v=0$. This results in:

$$\nabla K=(\kappa_1 S+\kappa_2 P)\mathbf{e}_1+(\kappa_1 T+\kappa_2 Q)\mathbf{e}_2,$$

and when $\kappa_2=0$, $\nabla K=\kappa_1(S\mathbf{e}_1+T\mathbf{e}_2)$. We now find $\nabla\kappa_r$ using equation (12). Given $K=0$ and that $\mathbf{w}$ lines up with $\mathbf{e}_2$, we find that $\nabla\kappa_r=\mathbf{C}(\mathbf{w},\mathbf{w})/\mathbf{w}\cdot\mathbf{w}=(S\mathbf{e}_1+T\mathbf{e}_2)$. Since the two gradients line up, these curves must be tangent at such points.

Furthermore, at these points we have $D_{\mathbf{w}}\kappa_r=T$; so provided that κ_2 passes through zero quickly (i.e. T is large), there will be suggestive contours at this location. We see from equations (2) and (13) that the speed of this point is zero (of course, higher derivatives will not be zero), which means this point will be stable.