

# Apparent Ridges for Line Drawing

by

Tilke Judd

Submitted to the Department of Electrical Engineering and Computer  
Science

in partial fulfillment of the requirements for the degree of

Masters of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Feb 2007

© Massachusetts Institute of Technology 2007. All rights reserved.

Author .....

Department of Electrical Engineering and Computer Science

Feb 2, 2007

Certified by .....

.....

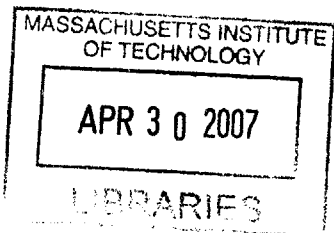
Frédo Durand  
Associate Professor  
Thesis Supervisor

Accepted by .....

.....

Arthur C. Smith

Chairman, Department Committee on Graduate Students



**BARKER**



# Apparent Ridges for Line Drawing

by

Tilke Judd

Submitted to the Department of Electrical Engineering and Computer Science  
on Feb 2, 2007, in partial fulfillment of the  
requirements for the degree of  
Masters of Science in Computer Science and Engineering

## Abstract

Non-photorealistic line drawing depicts 3D shapes through the rendering of feature lines. A number of characterizations of relevant lines have been proposed but none of these definitions alone seem to capture all visually-relevant lines. We introduce a new definition of feature lines based on two perceptual observations. First, human perception is sensitive to the variation of shading, and since shape perception is little affected by lighting and reflectance modification, we should focus on normal variation. Second, view-dependent lines better convey the shape of smooth surfaces better than view-independent lines. From this we define *view-dependent curvature* as the variation of the surface normal with respect to a viewing screen plane, and *apparent ridges* as the locus points of the maximum of the view-dependent curvature. We derive the equation for apparent ridges and present a new algorithm to render line drawings of 3D meshes. We show that our apparent ridges encompass or enhance aspects of several other feature lines.

Thesis Supervisor: Frédo Durand  
Title: Associate Professor



## Acknowledgments

I would like to thank my professor Frédo Durand for his support, his ideas and his advice - much of which I didn't come to understand and appreciate until some time after. I'd like to thank the graphics group as a whole for creating a stimulating environment to do research in. I'd like to thank my family for their positive encouragement and for ensuring that I get a great education. I'd like to thank Yann Le-Tallec for his critiques and revisions of this work and for keeping me happy always.



# Contents

<b>1</b>	<b>Introduction</b>	<b>15</b>
<b>2</b>	<b>Related Work</b>	<b>23</b>
2.1	Silhouettes and Contours . . . . .	23
2.2	Ridges and Valleys . . . . .	28
2.3	Suggestive Contours . . . . .	30
2.4	Other feature lines . . . . .	31
2.5	Discussion . . . . .	32
<b>3</b>	<b>2D Case: Line drawing in flatland</b>	<b>33</b>
3.1	Differential geometry in flatland . . . . .	33
3.2	Derivations in flatland . . . . .	35
3.3	Analysis . . . . .	38
<b>4</b>	<b>Apparent Ridges in 3D</b>	<b>41</b>
4.1	Geometric Properties of Surfaces . . . . .	41
4.2	View-dependent curvature . . . . .	45
4.3	Apparent Ridges . . . . .	49
4.4	Discussion . . . . .	52
<b>5</b>	<b>Adaptation to Meshes</b>	<b>53</b>
<b>6</b>	<b>Results and Analysis</b>	<b>57</b>
6.1	Silhouettes and Contours . . . . .	57

6.2	Ridges and Valleys . . . . .	62
6.3	Suggestive Contours . . . . .	63
6.4	Comparison to edge detection . . . . .	64
6.5	Performance evaluation . . . . .	65
<b>7</b>	<b>Conclusions and Future Work</b>	<b>67</b>



# List of Figures

1-1	A sampling of different Non-Photorealistic Rendering line drawings. (Courtesy of DeCarlo et al, Gooch, Buchin et al.) . . . . .	15
1-2	Here it is evident that a line illustration of an engine is more informative and useful than a photograph. We can clearly see the spicket and the tubes in the line drawing. Reprinted with permission, courtesy of Raskar et al. . . . .	17
1-3	The Max Planck model with several different feature lines. Contours alone don't give enough information to understand the shape, ridges and valleys are excessively sharp, and suggestive contours give a passable, but not great, drawing. We introduce apparent ridges on the right by adding a notion of view-dependency to ridges and valleys. Apparent ridges produce pleasing and informative lines. . . . .	18
1-4	<b>Difference between curvature and view-dependent curvature.</b> Curvature is defined as the change in the normal as one moves a small distance in direction $d$ along the <i>object</i> . On the other hand, view-dependent curvature is defined as the change in normal as one moves a small distance $d$ along the <i>screen</i> . Note that the view-dependent curvature at point $b'$ is much larger than point $a'$ . . . . .	19
2-1	(Left) Silhouettes are defined where the normal of the surface is perpendicular to the view vector (Reprinted with permission, courtesy of Hertzmann). (Right) Location of a ridge line on a surface. . . . .	24

2-2	Image space silhouette detection based on edge detection operators on the z-buffer. (a) Original shaded image, (b) z-buffer depth image, (c) detected edges, (d) combined image. (Courtesy of Saito and Takahashi 1990.) . . . . .	25
2-3	Image space silhouette detection using edge detection operators on the z-buffer and the normal buffer. (a) z-buffer depth map, (b) extracted edges from z-buffer, (c) normal buffer, (d) extracted edges from the normal buffer, and (e) combination of both edge buffers. (Reprinted with permission, courtesy of Aaron Hertzmann 1999.) . . . . .	25
2-4	Back-facing polygons are enlarged to achieve wide silhouette lines in a pipelined hardware implementation. (Courtesy of Raskar 1999) . . . . .	26
2-5	Left: Phong shaded model. Right: Gooch's rendering of the model with black and white silhouette and crease lines in the style of technical illustrations. (Courtesy of Gooch 1999) . . . . .	27
2-6	Computation of a subpolygon silhouette for (a) a single triangle and (b) a mesh. Plus signs at vertices mean the dot product between the normal vector and view direction is positive, negative signs mean the dot product is negative. Between the positive and negative vertex, linear interpolation is used to find the silhouette. (Reprinted with permission, courtesy of Hertzmann.) . . . . .	28
2-7	In the WYSIWYG NPR system of Kalnins et al., Artists directly annotated the same 3D teacup model to produce four distinct rendering styles. The system renders the scene from any new viewpoint and maintains the original look. (Courtesy of Kalnins 2002) . . . . .	28
2-8	An example showing the expressiveness added by suggestive contours. The left image has contours alone, the right image has both contours and suggestive contours. (Courtesy of DeCarlo et al. 2003) . . . . .	31

3-1	<b>Flatland setup</b> Here $\alpha$ is the parameterization of $u$ onto the curve, and $p$ is the projection of the curve onto a line in screen space. There is a tangent (or velocity) vector $\mathbf{t}$ and a normal vector $\mathbf{n}$ are defined at each point of the curve. . . . .	34
3-2	<b>Silhouettes and contours translated to flatland.</b> Silhouettes and contours are defined as locations where the normal of the curve is perpendicular to the view vector. When these locations are projected onto the screen space line, the indicated 4 points become the projection of the silhouettes and contours. . . . .	34
3-3	Difference between curvature and view-dependent curvature. . . . .	35
3-4	A flatland drawing of a bezier curve. . . . .	39
4-1	Setup for the 3D case . . . . .	43
4-2	<b>Ridges</b> On this surface, the dotted line is a principle curvature line; at each point on the line, the curvature changes most along the line direction. The red dot represents the max curvature along the principle curvature line and is a ridge point. The series of all the ridge points together creates the red ridge line. . . . .	45
4-3	The maximum apparent curvature at $b'$ is much larger than at $a'$ uniquely because of the perspective projection effect. . . . .	48
5-1	How we approximate the apparent curvature derivative, locate zero crossings on mesh and trim minima. For each edge, we flip the $\mathbf{t}_A$ in the direction of the positive derivative so that it points towards increasing apparent curvature. Then, if the two $\mathbf{t}_A$ along the edge point in opposite directions, there is a zero crossing. To test for minima, we drop a perpendicular from each vertex to the zero crossing line. If the positive $\mathbf{t}_A$ at each vertex makes an acute angle with the perpendicular, then the zero crossing is a maxima (as in c). If not, the zero crossing is eliminated as a minima (as in d). . . . .	54

6-1	Tablecloth model. Notice how the apparent ridges convey both the rim of the table and the drapery of the tablecloth. Note that the missing occluding contour on the back of the cloth is an artifact of our software and not of the line definitions. The difference figures on the right highlight the differences between the suggestive contours and apparent ridges. . . . .	57
6-2	Bust model. Note that the nose of the ridge and valley drawing is unnaturally sharp. Suggestive contours don't present a pleasing drawing on the right side of the image. . . . .	58
6-3	Brain model. Note the double lines that are drawn with suggestive contours. Both suggestive contours and apparent ridges produce pleasing drawings. . . . .	58
6-4	Column model. Suggestive contours do well on the center of the pillar, but apparent ridges do better on the base and on the face of the angel.	59
6-5	Given a slice of an object, suggestive contours are drawn at the gray locations, while apparent ridges are drawn at the black. This leads suggestive contours to draw two lines in areas that surround a valley, but lead apparent ridges to draw two lines in areas that surround an inflection point. . . . .	59
6-6	Flowerpot model. Notice how different lines need to be thresholded at different levels. The second row has a higher thresholding for all lines.	60
6-7	Head model. Apparent ridges make an elegant line drawing of this model. . . . .	60
6-8	Max Planck model. Apparent ridges render the facial features of the model with more pertinent detail than do the suggestive contours. Ridge and valley lines are overly sharp. . . . .	61
6-9	Vase model. Both ridge and valley and apparent ridge lines convey the details of the vase, and particularly the base, more cleanly and directly than do suggestive contours. . . . .	61

6-10	Sine wave mesh with ridges and valleys (purple and brown) and apparent ridges (red). Note displacement due to the perspective. Lines are collocated at head on views but shift as the object turns away from the viewer. . . . .	62
6-11	Gaussian bump from left to right: shaded view with occluding contour, suggestive contours and apparent ridges. . . . .	63
6-12	Experiments with Canny edge detection. (a) Average edge image for a single light source and Lambertian shading. (b) Average edge image for 10 random light sources and Lambertian shading. (c) Apparent ridge drawing. . . . .	66
6-13	Comparison between the average Canny edge detection on 120 photographs with different lighting and our apparent ridges. . . . .	66



# Chapter 1

## Introduction

Most research in computer graphics has been concerned with producing images of photographic realism. The computer's ability to display ever-increasing detail and complexity has given rise to new problems: communicating this information in a comprehensible way. Some form of visual abstraction is required. This has been well studied in graphic design and traditional illustration where it has long been understood that photographs are not always the best choice for presenting visual information. A simplified line drawing is often preferred when an image is required to delineate and explain (see Fig 1-2). The field of non-photorealistic rendering (NPR) emerged from this motivation and strives to make simple but *comprehensible* pictures of complicated objects by employing an economy of line.

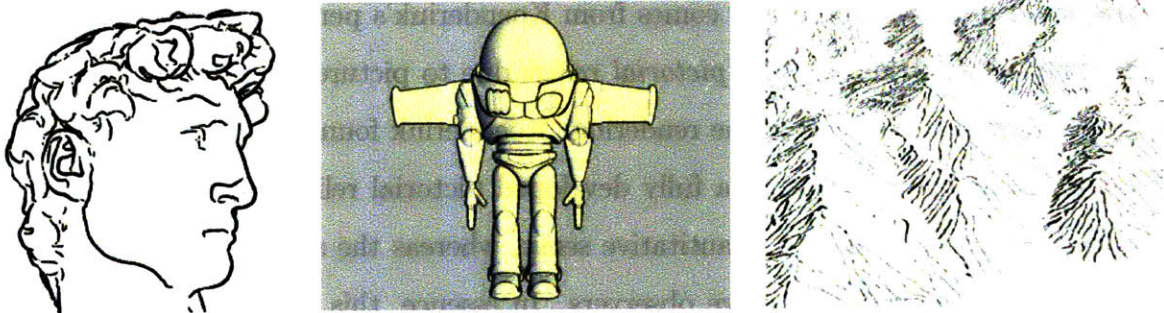


Figure 1-1: A sampling of different Non-Photorealistic Rendering line drawings. (Courtesy of DeCarlo et al, Gooch, Buchin et al.)

**“Less in a drawing is not the same as less of a drawing” [30]** The advantages of line illustration are numerous. Illustrations can convey information better by omitting extraneous detail, by focusing attention on relevant features, by clarifying and simplifying shapes, or by exposing parts that are hidden. In addition, illustrations compress well and are easy to reproduce and transmit. Illustrations also provide a natural way of conveying information at different levels of detail.

The benefits of illustrations over photographs are well-recognized in many practical contexts. For example, medical texts very often use illustrations in place of photographs, since they allow tiny and hidden structures to be much better described. In addition, most assembly, maintenance, and repair manuals of mechanical hardware use illustrations rather than photographs because of their clarity. What is the use of a photograph to mechanics when they already have the real thing in front of them? At Boeing, even when CAD database of airplane parts exist, all high-quality manuals are still illustrated by hand in order to provide more effective diagrams than can be achieved with either photorealistic rendering or simple hidden line drawings [20]. Strothotte et al.[31] note that architects often trace over computer rendering of their initial designs to create a sketchier look, to avoid giving their clients a false impression of completeness.

**Koenderink et al. Experiment** Further evidence that line drawings are effective for communicating shape comes from Koenderink’s perception experiment [17]. He studied what he calls the *pictorial reliefs* due to pictures of a single object in a single view as a function of the rendering. Koenderink found that a cartoon rendering of linear features lead to a fully developed pictorial relief and that full shading added relatively little in a quantitative sense, whereas the silhouette rendering lead to impoverished relief for naive observers. In essence, this experiment showed that line drawings are almost as good as fully shaded photographs at conveying important shape information. This is strong verification that line drawings are powerful perceptual informers.



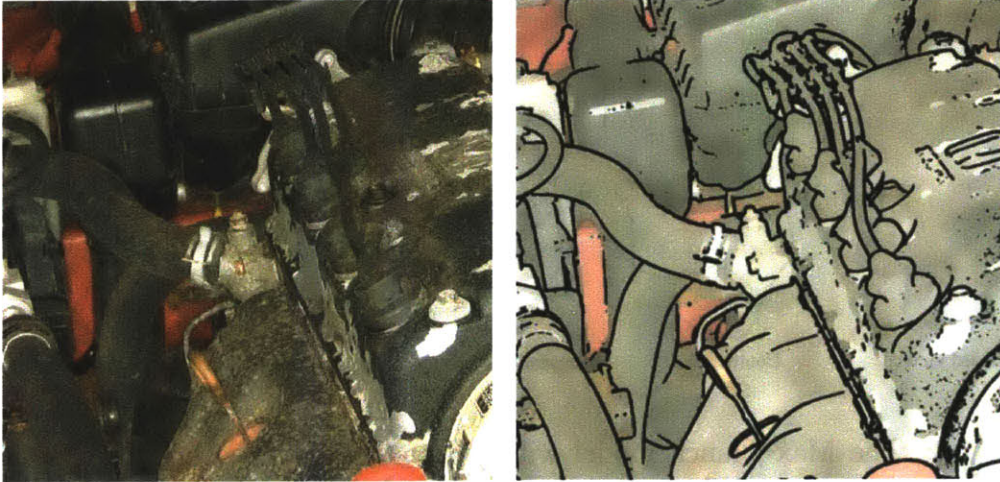


Figure 1-2: Here it is evident that a line illustration of an engine is more informative and useful than a photograph. We can clearly see the spicket and the tubes in the line drawing. Reprinted with permission, courtesy of Raskar et al.

**Feature Lines** Making line drawings on the computer requires precise definitions about where lines should be drawn given a 3D model. Characterizing and mathematically defining where these feature lines should be placed is a primary focus of NPR line drawing. In general, these lines correspond to points where a differential property vanishes. They can be distinguished by their order of derivation, but also by whether they take the viewpoint into account. The most significant feature lines that have been studied and used are listed here (see Fig 1-3).

*Contours* (sometimes called silhouettes) are lines where the normal is orthogonal to the view vector. These pervasive and important lines separate visible and invisible parts of the object. *Ridges and Valleys* are view-independent feature lines that are defined at the extrema of the object's curvature. These curves capture important object properties, but do not make natural looking drawings. They are locked to the surface of the object and do not slide along it when the viewpoint changes. The objects portrayed by ridges and valleys tend to look overly sharp. *Suggestive Contours* are view-dependent feature lines that correspond to contours in nearby views. The radial curvature in the direction of the view vector is zero. Suggestive contours convey some, but not all, of the desirable lines that comprise a drawing. For example, they

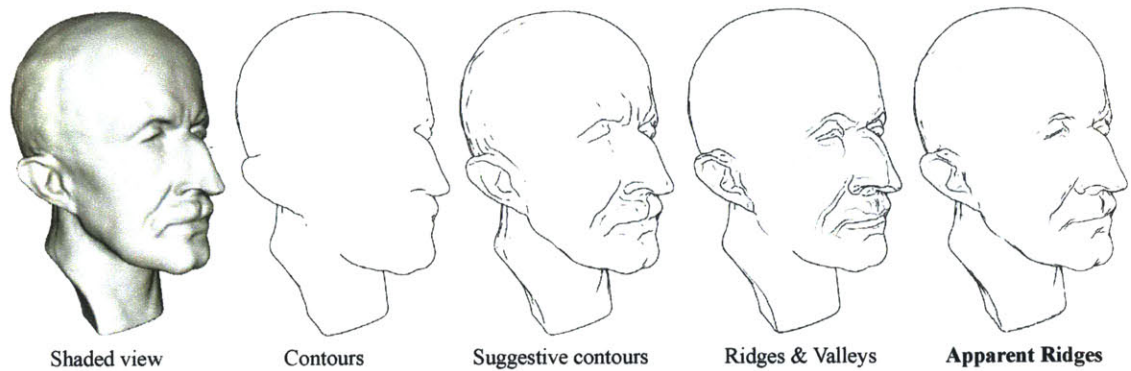


Figure 1-3: The Max Planck model with several different feature lines. Contours alone don't give enough information to understand the shape, ridges and valleys are excessively sharp, and suggestive contours give a passable, but not great, drawing. We introduce apparent ridges on the right by adding a notion of view-dependency to ridges and valleys. Apparent ridges produce pleasing and informative lines.

fail to draw lines on a rounded cube and other convex objects.

While the previous lines all capture important visual features, none of these definitions alone seem to capture all visually-relevant lines. In our work, we show how simple perceptual observations lead to a definition that captures these lines.

**Motivation and Key Idea** We base our work on the observation that human visual perception is sensitive to shading, which is intimately tied to the normals of surfaces. In addition, the human visual system is sensitive to variations of image intensity rather than absolute values. This puts a heavy emphasis on normal derivatives, or the object curvature. Ridges successfully capitalize on this perceptual sensitivity by defining lines where the variation of the normal, or the curvature, at a point in the principle direction is maximum. However, ridges are view-independent and fixed on the surface of an object. They do not take into consideration that humans perceive objects and the object's shading from a specific viewpoint.

We introduce *view-dependent curvature* and *apparent ridges* to take this viewpoint into account. Instead of measuring how surface normals vary on the surface, we measure how surface normals are seen to vary from a given viewpoint. Our lines take into account that, given a viewpoint, foreshortening on a surface can lead to increased

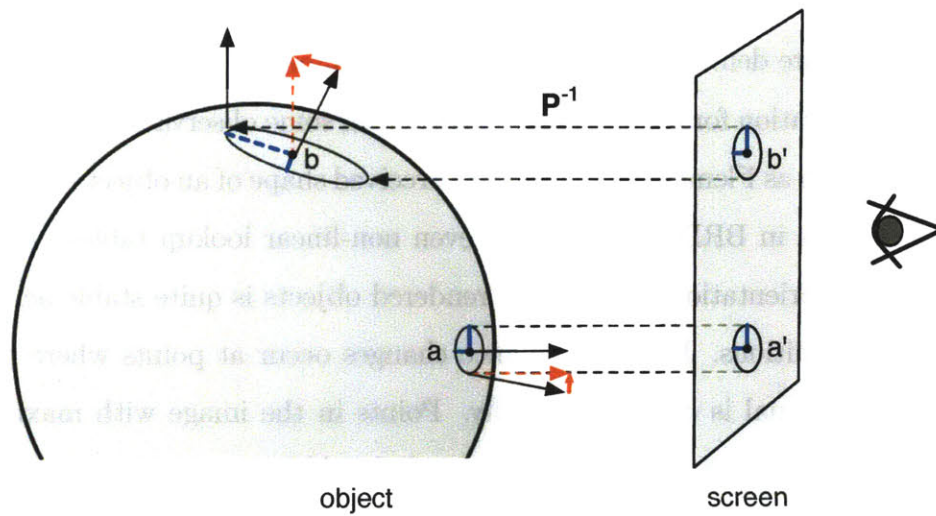


Figure 1-4: **Difference between curvature and view-dependent curvature.** Curvature is defined as the change in the normal as one moves a small distance in direction  $d$  along the *object*. On the other hand, view-dependent curvature is defined as the change in normal as one moves a small distance  $d$  along the *screen*. Note that the view-dependent curvature at point  $b'$  is much larger than point  $a'$ .

perceived variation in the normal especially near the edges of the object and as the object turns away from the viewer (see Fig 4-3). Using this intuition, we define view-dependent curvature as *the variation of the surface normal with respect to the viewing plane*, and apparent ridges as the locus points *where the maximum view-dependent curvature in the corresponding principle direction reaches a maximum*. In essence, our definition of view-dependent curvature measures the change in normal as seen from our viewpoint as opposed to the actual change in normal on the surface, and locates apparent ridge lines at the maximum. Both view-dependent curvature and apparent ridges are defined on the viewing screen plane.

Further motivation for our definition comes from some observations by researchers in perception such as Fleming et al [9]. The perceived shape of an object is surprisingly robust to changes in BRDF, lighting, or even non-linear lookup tables. Fleming et al note that the orientation structure of rendered objects is quite stable across these variations in conditions. Rapid luminance changes occur at points where the angle of the surface normal is changing rapidly. Points in the image with maximal view dependent curvature (i.e., our apparent ridges) should be marked by lines since they will usually contain maximal luminance gradients. Similarly, Durand et al. [8] showed that for photorealistic rendering, such points have highest local frequency content in an image.

**Overview and Contributions** In chapter 2, we look in detail at the previous work on feature lines: how they are defined, why they were introduced and how they have been used. In the following chapters, this paper makes the following contributions.

**Intuitive characterization** We characterize new feature lines, which we call apparent ridges, based on simple perceptual observations.

**Apparent ridge equation** We derive a mathematical characterization of feature lines based on these observations. In chapter 3, we derive the equation for the 2D case, because it is easier to interpret and provides important intuition. We show that this equation captures the phenomena of most previously-known

feature lines. In chapter 4 we extend these equations to 3D, where objects are solid shapes and line drawings are 2D.

**Computation on meshes** In chapter 5 we adapt the equations for triangular 3D meshes. We introduce an algorithm to compute, trim, and threshold apparent ridges on these meshes.

**Relationship to other lines** In chapter 6 we show results of apparent ridge line drawings and compare their performance with other line drawing methods.

**Experimental validation** We show that apparent ridges agree well with the average edges extracted by a Canny edge detector for images under random illumination.

Finally, in chapter 7 we draw conclusions.



# Chapter 2

## Related Work

Line drawing is a major part of artistic expression. Motivated by this, researchers in computer graphics have focused on characterizing and defining important feature lines and generating ways to render them as an image. The majority of work on feature lines focuses on silhouettes or contours, ridges and valleys, and more recently suggestive contours.

### 2.1 Silhouettes and Contours

Silhouettes and contours of a free form object are defined as the set of points on the object's surface where the normal is perpendicular to the view vector. Mathematically, the dot product of the normal  $\mathbf{n}$  with the view vector at a position  $p$  is zero:  $S = \{P : 0 = \mathbf{n} \cdot (p - c)\}$ , with  $c$  being the center of projection. In case of orthographic projection,  $(p - c)$  is replaced with the view direction vector  $\mathbf{v}$ . These lines separate visible and invisible parts of the object and provide a very strong shape cue for smooth objects [19]. Silhouettes form the closed outline around the projection. Contours comprise the other subset of this group and may be disjoint and can fall within the projective boundary. In the 2D image projection, contours divide one portion of the object from another of the same object. Silhouettes and contours are used so frequently in 2D line drawings and illustration that it is difficult to imagine a line drawing that doesn't include these curves. Since silhouettes are view dependent,

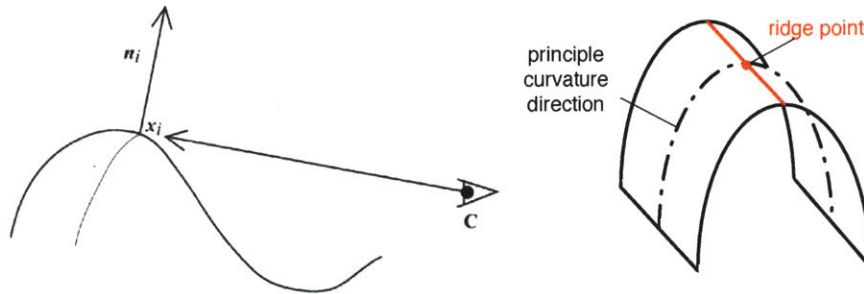


Figure 2-1: (Left) Silhouettes are defined where the normal of the surface is perpendicular to the view vector (Reprinted with permission, courtesy of Hertzmann). (Right) Location of a ridge line on a surface.

they need to be recomputed for every frame of an animation. Finding an efficient way to do this is non-trivial. Indeed, there has been much research in this area and there now exists a variety of different algorithms that compute silhouettes for geometric objects.

Silhouette detection methods can either be image space algorithms which operate only on the image buffer, or object based algorithms that perform manipulations in object space on an implicit, parametric or mesh based representation of a 3D object.

**Image-based algorithms** Saito and Takahashi first proposed the computation of lines through edge detection on an image buffer that encodes geometric information such as depth [28]. Decaudin later adapted this technique and used edges extracted from a normal map [6]. Similar to our work, these approaches can capture zero crossing of the normal variation and use view-dependent information, but their goal is to extract sharp creases between facets, while we focus on smooth features. However, in his course notes, Hertzmann used a similar approach to render the Beethoven bust with edges of a normal buffer and exhibited compelling results. See Fig 2-3 for further results using this approach.

Raskar [25] proposed a one-pass hardware implementation that basically adds little borders around triangles in ways that they appear only at the wanted feature line locations (see Fig 2-4). Because no connection information of the mesh or pre-processing is needed, the method is possible in pipelined hardware implementation.



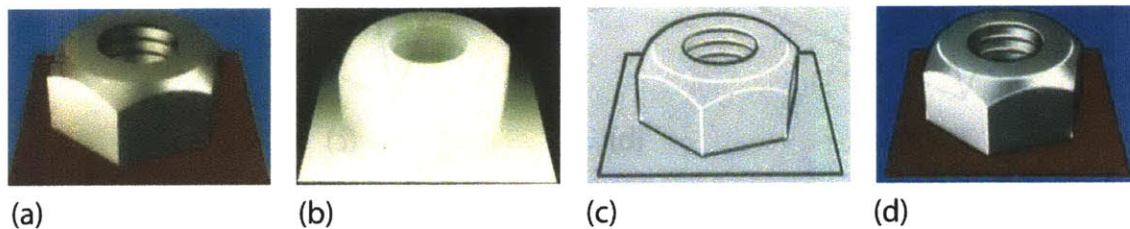


Figure 2-2: Image space silhouette detection based on edge detection operators on the z-buffer. (a) Original shaded image, (b) z-buffer depth image, (c) detected edges, (d) combined image. (Courtesy of Saito and Takahashi 1990.)

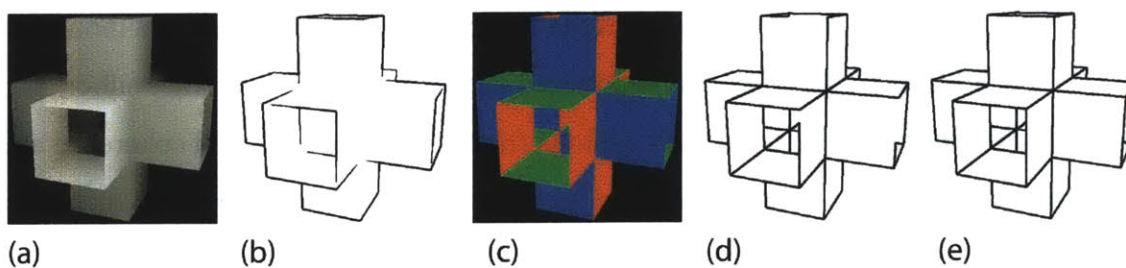


Figure 2-3: Image space silhouette detection using edge detection operators on the z-buffer and the normal buffer. (a) z-buffer depth map, (b) extracted edges from z-buffer, (c) normal buffer, (d) extracted edges from the normal buffer, and (e) combination of both edge buffers. (Reprinted with permission, courtesy of Aaron Hertzmann 1999.)

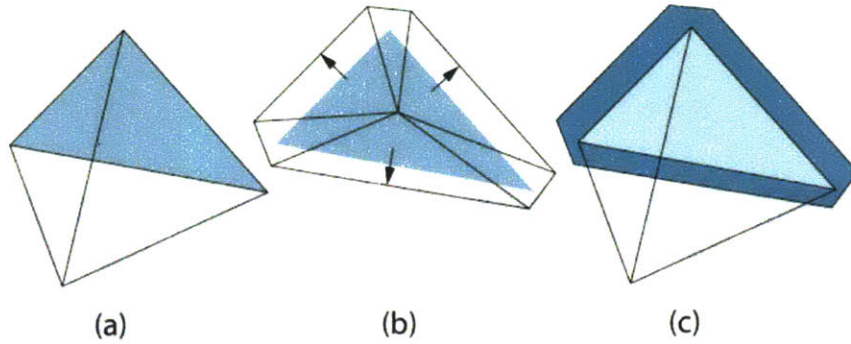


Figure 2-4: Back-facing polygons are enlarged to achieve wide silhouette lines in a pipelined hardware implementation. (Courtesy of Raskar 1999)

The main advantage of image space algorithms is that they can use existing graphics hardware and rendering packages to generate the buffers needed for the algorithms. Implementation is easy. The disadvantage of image based algorithms is that these pixel based representations give the user little control over the resulting line's attributes.

**Object-based algorithms** The added dimension of the object space algorithms allows for greater information which results in better approximations of silhouettes. [Markosian 1997] introduced a fast randomized algorithm to find and trace silhouettes on polygonal surfaces. Their method exploits frame-to-frame temporal coherence and uses random probes to locate silhouette seed points and trace silhouettes along the surface. In addition, by improving and simplifying Appel's hidden-line algorithm, they were able to trade accuracy and detail for speed, they were able to make a real-time renderer.

Inspired by this work, Gooch et al. [11] were able to take the next step to create a system which incorporates fast deterministic silhouette and crease finding algorithms with artistic shading and shadowing with the goal of making technical illustrations (see Fig 2-5). By preprocessing the normals they removed the need to check if every face was back or front facing at each frame and thus increased the speed of their silhouette computation.

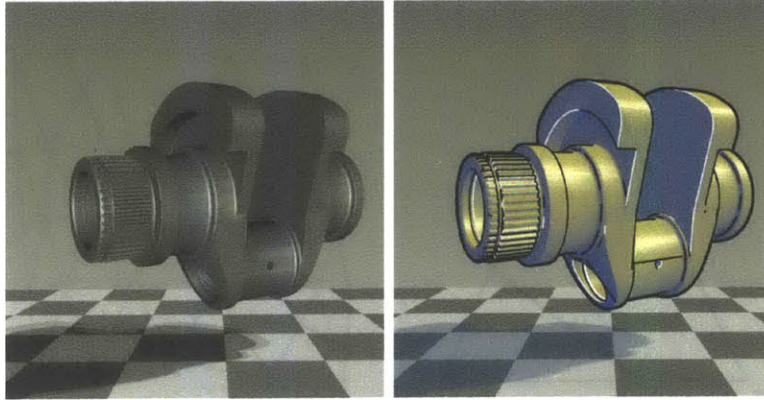


Figure 2-5: Left: Phong shaded model. Right: Gooch’s rendering of the model with black and white silhouette and crease lines in the style of technical illustrations. (Courtesy of Gooch 1999)

Sander [29] presented a fast silhouette computation algorithm based on a precomputed search tree of the original mesh which was used to clip the rendering of coarse geometry to the exact silhouette of the original model. This allowed a low-resolution geometry to be rendered with a high-resolution silhouette. This effectively hides the loss of information in the mesh because the silhouette is a much stronger visual cue of the shape of an object.

In order to get more exact results, Hertzmann and Zorin calculated silhouettes of a free-form surface that approximates the polygonal mesh. To find this silhouette, they recompute vertex normals from the approximated free-form surface and then compute its dot product with the respective viewing direction. Then, for every edge where the sign of the dot product is opposite on either side, they linearly interpolate along the edge to find the point where it is zero (see Fig 2-6). These points connect to result in a piecewise subpolygon silhouette line with far fewer artifacts. Because this result is much closer to the real silhouette, it is well suited for later stylization of lines.

To apply stylization to lines, an analytic representation of the silhouette is needed. Kalnins et al. [16] find analytic representations of silhouettes and use them to create an interactive NPR system which allows annotated brushstrokes of an artist to be consistently rendered across multiple frames and viewpoints (see Fig 2-7). Later they

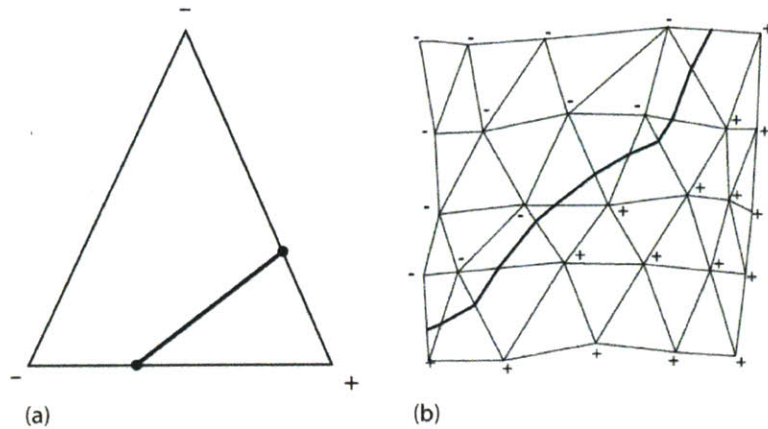


Figure 2-6: Computation of a subpolygon silhouette for (a) a single triangle and (b) a mesh. Plus signs at vertices mean the dot product between the normal vector and view direction is positive, negative signs mean the dot product is negative. Between the positive and negative vertex, linear interpolation is used to find the silhouette. (Reprinted with permission, courtesy of Hertzmann.)

furthered this work by describing a way to render stylized silhouettes of animated 3D models with temporal coherence [15].

## 2.2 Ridges and Valleys

As defined by Koenderink [18], ridge and valley lines are the locus of points at which the maximum (minimum) curvature assumes a local maximum (minimum) in the corresponding principle direction (see Fig 2-1). (Note that ridges and valleys have

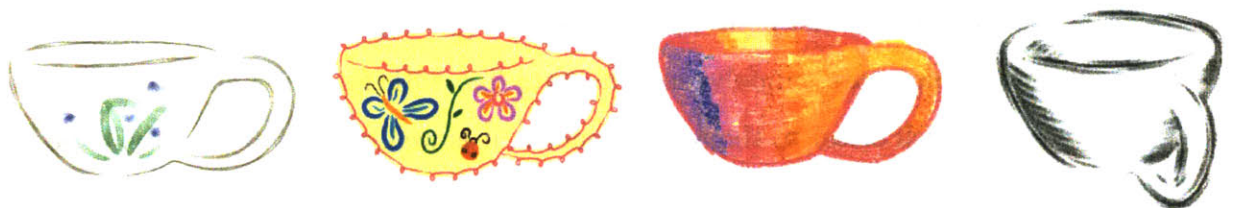


Figure 2-7: In the WYSIWYG NPR system of Kalnins et al., Artists directly annotated the same 3D teacup model to produce four distinct rendering styles. The system renders the scene from any new viewpoint and maintains the original look. (Courtesy of Kalnins 2002)

an alternate height field definition in topology). They consist of a sparse set of descriptive lines which are view-independent and therefore remain fixed on a surface under dynamic viewing conditions. Both ridge and valley lines are often associated with sharp changes in surface intensity due to the locally rapid rate of change of the surface normal direction. While valleys generally correspond to darker portions of the surface, ridges are more likely to reflect a specular highlight.

Both ridge and valley lines correspond to geometric features of the surface, and as such they can be computed automatically from local measures of the surface's differential geometry. Monga et al. [22] assume that the surface is defined locally as an iso-intensity contour and calculate directly the curvatures and characterize the local extrema from the first, second, and third derivatives of the grey level function. They use this method for data registration and automatic atlas generation.

Interrante et al [14] used the same definition of ridge lines as Monga et al, but eliminated the computation of the third derivative in favor of a very stable and simple approximation that tests for the presence of a local curvature maximum in a subvoxel region. They were more concerned with the display of perceptually relevant features as an aide to transparent isosurface visualization than with locating exact ridge lines. Lines are extracted from an isosurface and opacity is modulated by principal curvature.

Lopez et al [21] presented a survey of different formalizations of ridges and valleys. They examine definitions under height conditions, curvature conditions, as watershed and watercourses, and as drainage patterns. Subsequently, they evaluate these characterizations with respect to a list of desirable properties and their purpose in the context of representative image analysis tasks.

Raskar's paper [25] introduces a very fast way of finding ridges similar to the way he found silhouettes. A black quad is placed on each edge of every front facing polygon. The quads are oriented at an angle  $\theta$  with respect to the polygon such that when the polygons are rendered using the traditional depth buffer, only the quads at dihedral angles above the threshold are shown. These extra quads illuminate the ridge lines.

Pauly et al. [24] presents a technique for extracting line-type features on point-

sampled geometry. Given a point cloud, they apply principle component analysis to classify points that likely belong to a feature. The feature classification is based on surface variation estimation using covariance analysis of local neighborhoods. Then they use hysteresis thresholding to get initial approximation of feature lines, model each component of the graph as a snake to do feature smoothing, and finally render a line art drawing.

Ohtake et al. [23] also use the definitions laid out in [22] to estimate curvature and curvature derivatives of 3D objects using implicit function fits and extracting high-quality ridge and valley lines from these estimates. Curvature based filtering is used to keep the most significant lines.

All the papers described here, except for Raskar’s paper, use some form of estimation of the principle curvature lines in order to locate ridges and valleys. Some work has also been done to create line drawings using these curvature lines themselves. Hertzmann and Zorin [13] use the principle direction lines as a basis for their cross hatching textures, and Girshick et al. [10] describe artistic drawings made by principle direction lines that show the flow of curvature over the surface.

Differences between ridge and valley finding methods mostly come from differences in object surface structure. In his textbook [7], DoCarmo estimates principle curvature directions on parametric surfaces, Monga et al use the Hessian of iso intensity surfaces in 3D volume data, and Interrante used a similar technique based on Gaussian weighted finite-differencing. Pauly finds features on point based meshes. Still others fit approximate quadric patches to the mesh and calculate derivatives analytically from them. Girshick and Rusinkiewicz [27] estimate principle curvature lines directly from a polygonal surface mesh. Independent of how they are found, they have proven to be essential in conveying important shape information.

## 2.3 Suggestive Contours

Recently, a new type of line has been suggested. DeCarlo et al. [5] go beyond silhouettes, contours, ridges and valleys by developing the new lines they call suggestive

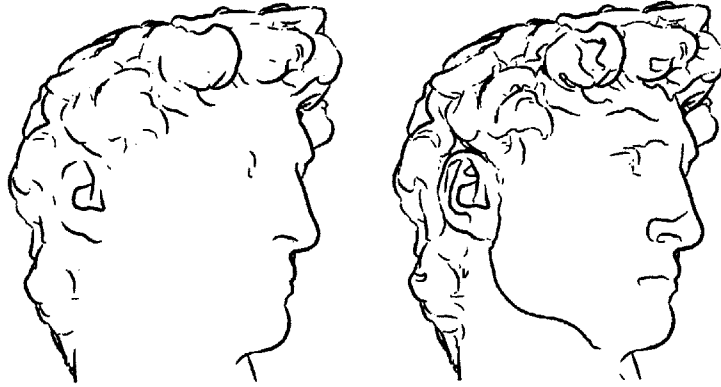


Figure 2-8: An example showing the expressiveness added by suggestive contours. The left image has contours alone, the right image has both contours and suggestive contours. (Courtesy of DeCarlo et al. 2003)

contours. Suggestive Contours are lines drawn of visible parts of the surface, where a true contour would appear with a small change in viewpoint. In this way suggestive contours complement contours in depicting shape by anticipating and extending them. Mathematically, these lines are defined at point with radial curvature, or curvature in the direction of the view vector projected onto a points tangent plane, are zero. The paper provides both image-space and object-space extraction algorithms.

In 2004, DeCarlo et al. [4] extended this work on static suggestive contours to real-time and dynamic settings. They analyzed movement of suggestive contours under changes in viewpoint, and offered techniques for improving stroke quality rendered for a moving camera.

Burns et al. [1] extended line extraction of suggestive contours to volume data because of its important applications in medicine and scientific simulation.

## 2.4 Other feature lines

The majority of work on line drawings focus on silhouettes, ridge and valley, and suggestive contours. However, sometime other line classes are discussed in the literature and they are defined quickly here. **Creases** are sharp lines, such as the edges of a cube, which correspond to discontinuities of the mesh normals. They are a subset of

ridges and valleys and can typically be identified by comparing the angle between its adjacent polygons with a certain threshold. **Border lines** appear in models where the mesh is not closed and are those edges with only one adjacent polygon. **Self intersection lines** are where two parts of a model intersect. These aren't necessarily edges of a mesh, but are important for shape recognition. **Parabolic lines** are lines where one of the principal curvatures vanishes [17]. They separate hyperbolic and elliptic regions of the surface.

## 2.5 Discussion

While all these lines capture important features of an object, none of the definitions alone captures all visually relevant lines. While contours alone provide very strong shape information on the edge of the object, they don't provide enough detail on the insides. While ridges and valleys convey important features defined by extremal curvature, because they don't take viewpoint into consideration they are fixed on an object and create unpleasing images. While suggestive contours create nice drawings in many cases, they don't draw any features on convex objects.

Motivated by the fact that the variation of object normals convey strong shape information, and that lines are best drawn as view dependent features, we characterize apparent ridges as lines which build on and enhance the previously defined lines. Apparent ridges take both curvature and view-dependent foreshortening into account to create pleasing images not seen before.



# Chapter 3

## 2D Case: Line drawing in flatland

Flatland is a simplified 2D world. Objects are 2D curves and a “drawing” of the curve is a set of discrete points on a line. We look at the view-dependent curvature and apparent ridges in flatland first because it is easier to understand and provides good intuition. We first review elements of differential geometry, define objects in flatland, and then derive the equation for apparent ridges.

### 3.1 Differential geometry in flatland

**Definitions and Review** In flatland, a curve is represented by a function  $\alpha$  which takes a one dimensional parametric variable  $u$  and maps it onto a curve in two-dimensional space (Fig. 3-1). Each point on the curve has tangent (or velocity) vector  $\mathbf{t}$  parallel to the curve and a normal vector  $\mathbf{n}$  perpendicular to the tangent. The tangent is obtained by deriving  $\alpha$  with respect to  $u$ . We assume without loss of generality that the magnitude of  $d\alpha/du$  is one, that is, the curve is parameterized by its arclength (other situations can be reduced to this one by reparameterization).

Curvature  $\mathbf{k}$  is defined as the derivative of the normal (see Fig 4-3). It is a vector in the direction of  $\mathbf{t}$  with magnitude inversely proportional to the radius of the circle tangent to the curve.  $\mathbf{k}$  is orthogonal to  $\mathbf{n}$  because  $\mathbf{n}$  is unit-length. (Note that curvature in 2D is usually expressed as the derivative of the tangent, but we stick to curvature as a derivative of the normal because it is more consistent with the notion

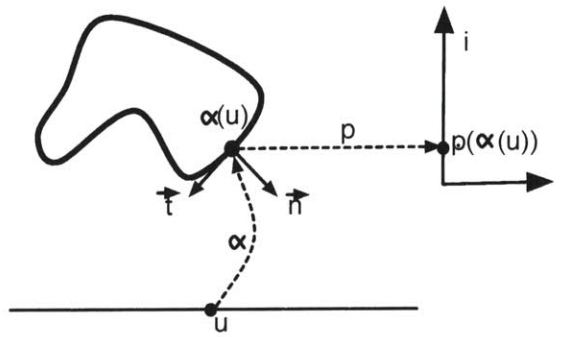


Figure 3-1: **Flatland setup** Here  $\alpha$  is the parameterization of  $u$  onto the curve, and  $p$  is the projection of the curve onto a line in screen space. There is a tangent (or velocity) vector  $\mathbf{t}$  and a normal vector  $\mathbf{n}$  are defined at each point of the curve.

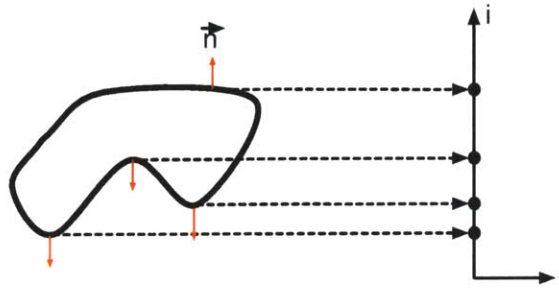


Figure 3-2: **Silhouettes and contours translated to flatland.** Silhouettes and contours are defined as locations where the normal of the curve is perpendicular to the view vector. When these locations are projected onto the screen space line, the indicated 4 points become the projection of the silhouettes and contours.

of shape operator in 3D).

We define “drawings” representing our 2D curve as orthographic projections of feature points onto a line. The mapping  $p$  projects points from the curve to the line. Without loss of generality, we assume that this line is along the coordinate axis  $\mathbf{i}$ . The projection direction or view direction is  $\mathbf{j}$  (see Fig 3-1).

**Traditional feature lines translated to flatland** Silhouettes and contours correspond to points where the normal is orthogonal to the view direction, that is,  $\mathbf{n} \cdot \mathbf{j} = 0$  (Fig. 3-2). Ridges are extrema of the curvature. The closest representation of sugges-

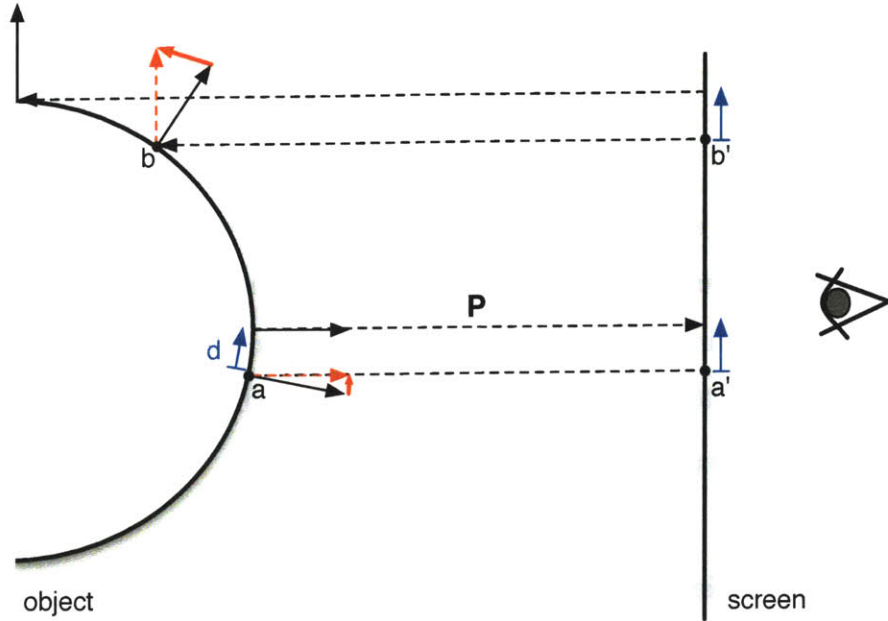


Figure 3-3: Difference between curvature and view-dependent curvature.

tive contours are points where curvature is zero. In 2D, this corresponds to inflection points.

## 3.2 Derivations in flatland

To locate apparent ridges, we need to find the maxima of the view-dependent curvature. For this, we first derive the normal  $\mathbf{n}(u)$  with respect to the screen space variable  $i$  to get the view-dependent curvature.

$$\frac{d\mathbf{n}(u)}{di} \tag{3.1}$$

where

$$u = (p \circ \alpha)^{-1}(i) = g^{-1}(i) \tag{3.2}$$

Because our view-dependent curvature turns out to be a vector, we define the maxima to be the extrema of the magnitude of the vector as it changes. In general, we know

that the maximum values of a vector  $\mathbf{v}$  are the zeros of

$$\frac{d(\mathbf{v} \cdot \mathbf{v})}{dt} = 2 \frac{d\mathbf{v}}{dt} \cdot \mathbf{v}, \quad (3.3)$$

so we need the vector and its derivative as well. For this we compute the derivative of the view-dependent curvature with respect to the screen (second derivative =  $\frac{d^2 \mathbf{n}(u)}{d^2 i}$ ). When we have the first and second derivative, we simply multiply them together. The zeros of the resulting expression locate our apparent ridges.

We analyze the derivations sequentially.

**First derivative: view-dependent curvature** We find the first derivative of the normal of the curve with respect to the screen projection. Because we are taking the derivative of the normal  $\mathbf{n}(u)$  with respect to the screen space variable  $i$ , we use the chain rule to move between the two variables, following the variable mapping defined in equation 3.2.

$$\frac{d\mathbf{n}(u)}{di} = \frac{d\mathbf{n}(u)}{du} \cdot \frac{d(g^{-1}(i))}{di} \quad (3.4)$$

For the first part of the chain we get

$$\frac{d\mathbf{n}(u)}{du} = \mathbf{k}(u) \quad (3.5)$$

where  $\mathbf{k}(u)$  is a vector representing the scalar curvature multiplied with the tangent vector. The second term in the chain rule is calculated as

$$\frac{d(g^{-1}(i))}{di} = \frac{-1}{\mathbf{n}(u) \cdot \mathbf{j}} \quad (3.6)$$

Putting these two together, we find that the first derivative is

$$\frac{d\mathbf{n}(u)}{di} = \frac{-\mathbf{k}(u)}{\mathbf{n}(u) \cdot \mathbf{j}} \quad (3.7)$$

Now it becomes clear that the difference between the the vector curvature  $\mathbf{k}(u)$  and the view-dependent curvature is the  $\mathbf{n}(u) \cdot \mathbf{j}$  in the denominator. This denominator

accounts for the foreshortening of a curve as seen from a specific viewpoint. As we move towards the contour where  $\mathbf{n}(u) \cdot \mathbf{j} = 0$ , the view-dependent curvature goes to infinity (although it is undefined at the exact contour point where the denominator is zero).

**Second derivative: Apparent Ridges** In order to calculate the second derivative,

$$\frac{d}{di} \frac{-\mathbf{k}(u)}{\mathbf{n}(u) \cdot \mathbf{j}} \quad (3.8)$$

we need the derivative of the numerator and the derivative of the denominator of the fraction in order to piece together the whole derivative.

The derivative of the numerator is

$$-\frac{d\mathbf{k}(u)}{di} = \frac{d\mathbf{k}(u)}{du} \frac{1}{\mathbf{n}(u) \cdot \mathbf{j}}, \quad (3.9)$$

and the derivative of the denominator is

$$\frac{d(\mathbf{n}(u) \cdot \mathbf{j})}{di} = \frac{\mathbf{k}(u)}{\mathbf{n}(u) \cdot \mathbf{j}} \cdot \mathbf{j}. \quad (3.10)$$

Therefore the complete second derivative is

$$\frac{d^2\mathbf{n}(u)}{d^2i} = \frac{1}{(\mathbf{n}(u) \cdot \mathbf{j})^2} \left[ \frac{d\mathbf{k}(u)}{du} + \frac{\mathbf{k}(u)^2}{\mathbf{n}(u) \cdot \mathbf{j}} \cdot \mathbf{j} \right]. \quad (3.11)$$

In order to locate our desired maxima, we follow equation 3.3 and multiply the first derivative by the second derivative and set it to zero. This gives us

$$-\frac{1}{(\mathbf{n}(u) \cdot \mathbf{j})^3} \left[ \frac{d\mathbf{k}(u)}{du} \cdot \mathbf{k}(u) + \frac{\mathbf{k}(u)^2}{\mathbf{n}(u) \cdot \mathbf{j}} \mathbf{k}(u) \cdot \mathbf{j} \right] = 0 \quad (3.12)$$

Once again we notice that the equation has singularities at  $\mathbf{n}(u) \cdot \mathbf{j} = 0$ . Except for at these points, we can multiply every term by  $(\mathbf{n}(u) \cdot \mathbf{j})^4$  and rearrange the equation to be more readable:

$$\underbrace{-(\mathbf{n}(u) \cdot \mathbf{j}) \cdot \frac{d\mathbf{k}(u)}{du} \cdot \mathbf{k}(u)} - \underbrace{\mathbf{k}(u)^3 \cdot \mathbf{j}} = 0 \quad (3.13)$$

We call the first term of the equation the *feature* term and the second term the *offset* term. Note that  $\mathbf{k}(u)^3 \cdot \mathbf{j}$  is a shortcut for the multiplication of the two dot products  $\mathbf{k}(u)^2$  and  $\mathbf{k}(u) \cdot \mathbf{j}$ .

### 3.3 Analysis

The resulting expression encapsulates aspects of several different feature lines.

Ridge and valley lines are defined as the locations at which the curvature of the object is at an extrema. This is expressed in the feature term of equation when the derivative of the curvature  $d\mathbf{k}(u)/du$  is zero.

Suggestive contours are the locations where the radial curvature of the curve is zero. In 2D, these correspond to the inflection points, or the locations where the curvature is zero, and are seen in the feature term when  $\mathbf{k}(u)$  is zero.

Silhouette or contour lines are defined as the locations where the normal is orthogonal to the view vector. This is expressed in the feature term of the equation when  $(\mathbf{n}(u) \cdot \mathbf{j})$  is zero. Because we divided our equation by  $(\mathbf{n}(u) \cdot \mathbf{j})^4$  to get this formulation, these are singularities of our equation. However, if we consider the value of the view-dependent curvature in the limit as it approaches the silhouette, we see that the view-dependent curvature tends toward infinity. This is clearly a maximum of the view-dependent curvature.

We have shown that the feature term of the equation goes to zero at the ridge and valley, silhouette and suggestive contours of a curve. The second term of the equation essentially offsets the zeros of the equation by moving them slightly away from the zeros of the feature term. So, in theory the zeros of our entire equation are shifted from the true values of the mentioned feature lines by a term related to the viewpoint. In our experiments however, the offset term is much smaller than the feature term

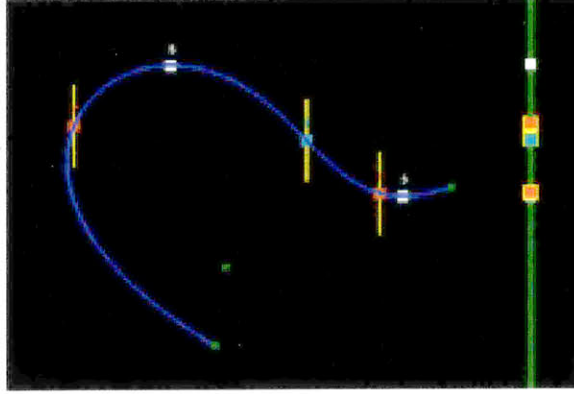


Figure 3-4: A flatland drawing of a bezier curve.

(on the order of  $1/100,000$  the size) and plays a small role in the location of the zero crossings.

We created a piece of software which creates 1D point drawings of 2D bezier curves. Figure 3-4 shows a screenshot of the program. We calculate the major feature points on a 5-control point bezier curve and project them onto the green viewing screen on the right. The ridges and valleys (red 'r'), silhouettes (white 's'), and inflection/suggestive contour points (blue 'i') are indicated. The apparent ridge points found from our equation are indicated by the yellow lines and align almost exactly with the ridge and valley and inflection points. Our equation for apparent ridges have a singularity point at the silhouettes and so are not drawn by our program. However, since view-dependent curvature is a maximum at the silhouette points, apparent ridges are located there as well. This program allowed us to visually verify that our equation captures many of the major feature points.

We proceed by deriving the equations for view-dependent curvature and apparent ridges in 3D.





# Chapter 4

## Apparent Ridges in 3D

In this section we extend our equations for view-dependent curvature and apparent ridges to 3D. First we describe our geometric setup and look closely at the first and second derivatives of the surface normal with respect to the surface. These correspond to the curvature and ridges of the surface. Then, to define view-dependent curvature and ridges, we calculate the same first and second derivative of the surface normal with one key change: we take the derivative with respect to a screen plane instead of the surface. These equations for view-dependent curvature and apparent ridges in 3D define the properties needed to create apparent ridge line drawings of 3D geometry.

### 4.1 Geometric Properties of Surfaces

**Object space** Fix a smooth, closed surface  $\mathcal{S}$  in  $\mathbb{R}^3$ . We denote  $\mathbf{n}(M)$  to be the outward facing normal to the surface at point  $M$ . The tangent plane at  $M$  is a local linear approximation to the surface at  $M$  and the normal  $\mathbf{n}(M)$  is perpendicular to the tangent plane. We let  $M$  be the origin of the tangent plane such that we can identify a point  $N$  in the plane as a vector  $\mathbf{r} = \overrightarrow{MN}$ . Given orthonormal basis  $\{\mathbf{t}_1, \mathbf{t}_2\}$  of the tangent plane, we can write any vector  $\mathbf{r}$  as  $\mathbf{r} = r_1\mathbf{t}_1 + r_2\mathbf{t}_2$ .

**Curvature** Intuitively, curvature of a surface represents how a surface bends, or equivalently it describes how the normal changes from point to point on a surface.

Table 4.1: Table of Notation

$M$	point on smooth, closed surface
$\mathbf{n}$	normal to surface at point $M$
$\mathbf{r} = r_1\mathbf{t}_1 + r_2\mathbf{t}_2$	vector on tangent plane given basis $\{\mathbf{t}_1, \mathbf{t}_2\}$
$D_{\mathbf{r}}\mathbf{n}$	directional derivative which describes how the normal at point $M$ changes as one moves along the tangent vector $\mathbf{r}$ .
$K(\mathbf{r})$	linear curvature operator at point $M$ in the tangent direction $\mathbf{r}$ where $K(\mathbf{r}) = D_{\mathbf{r}}\mathbf{n}$ ( $K$ is also called the shape operator)
$\lambda_{\max}, \lambda_{\min}$	eigenvalues of $K$ , and max and min principal curvatures at point $M$
$\mathbf{t}_{\max}, \mathbf{t}_{\min}$	eigenvectors of $K$ , and max and min principal curvature directions at point $M$
$M'$	point on the screen plane
$\mathbf{s} = s_x\mathbf{u}_x + s_y\mathbf{u}_y$	vector on screen plane given basis $\{\mathbf{u}_x, \mathbf{u}_y\}$
$P$	projection operator at point $M$ which takes points from the tangent plane to the screen plane. $P(M) = M'$ and $P(\mathbf{r}) = \mathbf{s}$
$\tilde{\mathbf{n}}(M')$	given point $M'$ on the screen, $\tilde{\mathbf{n}}$ describes the normal on the corresponding point on the object. $\tilde{\mathbf{n}}(M') = \mathbf{n}[P^{-1}(M')]$
$D_{\mathbf{s}}\tilde{\mathbf{n}}$	directional derivative which describes how the normal of the object changes as one moves along the screen vector $\mathbf{s}$ . $D_{\mathbf{s}}\tilde{\mathbf{n}} = D_{\mathbf{r}}\mathbf{n}$ where $\mathbf{r} = P^{-1}\mathbf{s}$ .
$Q(\mathbf{s})$	linear view-dependent curvature operator at point $M'$ in the screen direction $\mathbf{s}$ where $Q(\mathbf{s}) = K(P^{-1}(\mathbf{s}))$
$\lambda_{A\max} = \lambda_A, \lambda_{A\min}$	eigenvalues of $Q^T Q$ , max and min view-dependent curvatures
$\mathbf{t}_{A\max} = \mathbf{t}_A, \mathbf{t}_{A\min}$	eigenvectors of $Q^T Q$ , max and min principal view-dependent curvature directions

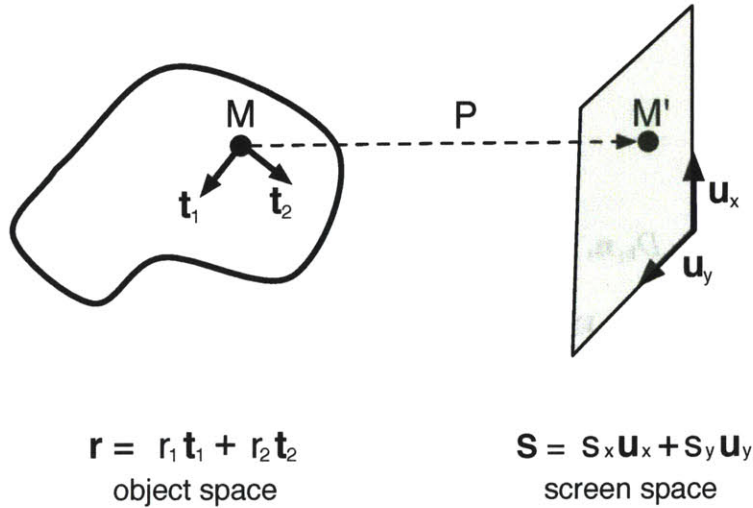


Figure 4-1: Setup for the 3D case

We define the **curvature operator**  $K$  at point  $M$  as

$$K(\mathbf{r}) = D_{\mathbf{r}}\mathbf{n}$$

where  $D_{\mathbf{r}}\mathbf{n}$  is the directional derivative of the normal in the direction  $\mathbf{r}$  in the tangent plane.  $K$  is also known as the *Weigarten map* or the *shape operator* (often denoted as  $S$ ).  $K(\mathbf{r})$  describes the curvature at point  $M$  in direction  $\mathbf{r}$ .

If surface  $\mathcal{S}$  is a plane, the normal  $\mathbf{n}(A)$  is constant, so the curvature operator of a plane is identically zero at all points of the plane. For a nonplanar surface the surface normal  $\mathbf{n}$  will twist and turn from point to point, and  $K$  will be nonzero.

The curvature operator  $K$  at each point  $M$  is a linear map from the tangent of surface at  $M$  into the tangent on the Gaussian sphere. The Gaussian sphere is the unit sphere consisting of all unit vectors in 3-space. Since  $\mathbf{n}$  and its derivative are orthogonal, if we regard  $\mathbf{n}$  as a point of the Gaussian sphere, we can think of the derivative of  $\mathbf{n}$  as a tangent vector to the Gaussian sphere. It turns out that the tangent plane of the surface at  $M$  and the tangent of the Gaussian sphere are the same plane and can both be represented with respect to the basis  $\{\mathbf{t}_1, \mathbf{t}_2\}$ .

The curvature matrix at point  $M$  given a basis  $\{\mathbf{t}_1, \mathbf{t}_2\}$  of the tangent plane is

defined by

$$K = \begin{pmatrix} L & M \\ M & N \end{pmatrix}$$

where

$$L = \mathbf{t}_1 \cdot D_{\mathbf{t}_1} \mathbf{n}, \quad M = \mathbf{t}_1 \cdot D_{\mathbf{t}_2} \mathbf{n} = \mathbf{t}_2 \cdot D_{\mathbf{t}_1} \mathbf{n}, \quad N = \mathbf{t}_2 \cdot D_{\mathbf{t}_2} \mathbf{n} \quad (4.1)$$

See [3] for further details. For all  $\mathbf{r}$  in the tangent plane,

$$K \begin{pmatrix} r_1 \\ r_2 \end{pmatrix} = \begin{pmatrix} L & M \\ M & N \end{pmatrix} \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}.$$

For a given point on the surface, we define the maximum and minimum principal curvatures  $\lambda_{\max}$  and  $\lambda_{\min}$  as the eigenvalues of  $K$ . The associated eigenvectors  $\mathbf{t}_{\max}$  and  $\mathbf{t}_{\min}$  are the maximum and minimum principle curvature directions.

We can write  $K$  with respect to its principal curvature directions,  $\mathbf{t}_1 = \mathbf{t}_{\max}$  and  $\mathbf{t}_2 = \mathbf{t}_{\min}$ , which leads to a diagonal matrix with principal curvatures as entries:

$$K = \begin{pmatrix} \lambda_{\max} & 0 \\ 0 & \lambda_{\min} \end{pmatrix}.$$

**Derivative of curvature** Just as one can describe curvature as the change in normal with motion along the surface, one can describe the *derivative of curvature* at a point as the change in the curvature operator with motion along the surface. Given the principle coordinate basis  $\{\mathbf{t}_1, \mathbf{t}_2\} = \{\mathbf{t}_{\max}, \mathbf{t}_{\min}\}$  of the tangent plane, we define the **curvature derivative tensor** as a  $2 \times 2 \times 2$  rank-3 tensor written here as a vector of matrices

$$\mathbf{C} = (D_{\mathbf{t}_1} K \quad D_{\mathbf{t}_2} K) = \left( \left( \begin{pmatrix} P' & Q' \\ Q' & S' \end{pmatrix} \quad \begin{pmatrix} Q' & S' \\ S' & T' \end{pmatrix} \right) \right) \quad (4.2)$$

where  $P' = D_{\mathbf{t}_1} \lambda_{\max}$ ,  $Q' = D_{\mathbf{t}_2} \lambda_{\max}$ ,  $S' = D_{\mathbf{t}_1} \lambda_{\min}$ , and  $T' = D_{\mathbf{t}_2} \lambda_{\min}$

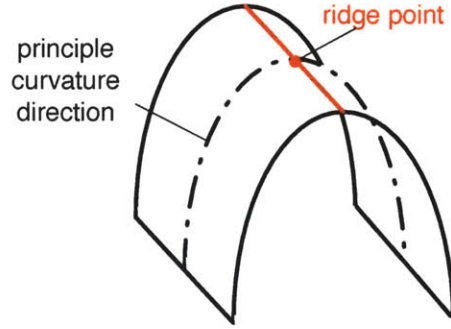


Figure 4-2: **Ridges** On this surface, the dotted line is a principle curvature line; at each point on the line, the curvature changes most along the line direction. The red dot represents the max curvature along the principle curvature line and is a ridge point. The series of all the ridge points together creates the red ridge line.

See [12] [27] for further details. For all  $\mathbf{r}$  in the tangent plane,  $\mathbf{C}(\mathbf{r})$  gives a matrix equal to the directional derivative of  $K$  in direction  $\mathbf{r}$ .

**Ridges** As defined by Koenderink [18], ridges are the locus of points at which the maximum curvature  $\lambda_{\max}$  assumes a local maximum in the principal direction  $\mathbf{t}_{\max}$ . See (Fig. 4-2). They are found where

$$D_{\mathbf{t}_{\max}} \lambda_{\max} = 0.$$

Note that this is the first element  $P'$  of the curvature derivative tensor defined above. The procedure here would not work properly near the umbilical points where ( $\lambda_{\max} = \lambda_{\min}$ ) since the principal directions are not defined at the umbilicals. Fortunately it can be easily shown that the loci of maxima of  $\lambda_{\max}$  and the minima of  $\lambda_{\min}$  along their corresponding curvature lines do not pass through the typical umbilics. Therefore the ridge-valley lines do not approach the generic umbilical points [23].

## 4.2 View-dependent curvature

To define view-dependent curvature, we still consider a smooth closed surface with a normal  $\mathbf{n}$  and tangent plane with basis  $\{\mathbf{t}_1, \mathbf{t}_2\}$  defined at point  $M$ . We introduce the

new concepts of a screen space and a projection function and use these to develop the notion of view-dependent curvature.

**Screen space** Consider a viewing screen plane which looks onto the surface defined by a reference point in space and a screen space basis  $\{\mathbf{u}_x, \mathbf{u}_y\}$ . This is the surface on which we will “draw” a line drawing of the surface. A vector  $\mathbf{s}$  on this surface is described as  $\mathbf{s} = s_x\mathbf{u}_x + s_y\mathbf{u}_y$ .

**Projection to screen space** We define the orthographic projection  $P$  from 3D space to the screen. For a point  $M$  on the object, we will consider the restriction of  $P$  from the tangent plane at  $M$  to the screen. Given the choice of basis  $\{\mathbf{t}_1, \mathbf{t}_2\}$  for the object tangent plane and  $\{\mathbf{u}_x, \mathbf{u}_y\}$  for the screen plane, we can express this mapping as a  $2 \times 2$  matrix, defined by

$$P = \begin{pmatrix} \mathbf{t}_1 \cdot \mathbf{u}_x & \mathbf{t}_2 \cdot \mathbf{u}_x \\ \mathbf{t}_1 \cdot \mathbf{u}_y & \mathbf{t}_2 \cdot \mathbf{u}_y \end{pmatrix}.$$

Any vector in the object tangent plane  $\mathbf{r} = r_1\mathbf{t}_1 + r_2\mathbf{t}_2$  is projected to  $\mathbf{s}$  in the screen plane where  $\mathbf{s} = s_x\mathbf{u}_x + s_y\mathbf{u}_y$ , as

$$\begin{pmatrix} \mathbf{t}_1 \cdot \mathbf{u}_x & \mathbf{t}_2 \cdot \mathbf{u}_x \\ \mathbf{t}_1 \cdot \mathbf{u}_y & \mathbf{t}_2 \cdot \mathbf{u}_y \end{pmatrix} \begin{pmatrix} r_1 \\ r_2 \end{pmatrix} = \begin{pmatrix} s_x \\ s_y \end{pmatrix}.$$

Note that  $P$  has an implicit dependency on  $M$ ; the projection matrix is different at every point  $M$  on the object. Note also that in general a projection is not invertible. However, since we are restricting  $P$  to be defined on the tangent plane of the object at point  $M$ , it is invertible. This is true for all points except where the tangent plane is perpendicular to the screen plane, namely the silhouette points.  $P^{-1}$  is not defined at these points.

**Normals as seen from the screen** A normal  $\mathbf{n}(M)$  is defined at all points  $M$  on the surface of the object. We define the notion of the normal on the corresponding

screen point  $M'$ , denoted by  $\tilde{\mathbf{n}}(M')$ , to be the normal of the surface at point  $M$  where  $M = P^{-1}(M')$ . Formally, we write  $\tilde{\mathbf{n}}(M') = \mathbf{n}[P^{-1}(M')]$ .

**View-dependent curvature** Intuitively, view-dependent curvature is how much the surface is seen to bend from a viewpoint. It takes both the curvature of the object and the foreshortening of the object due to one's viewpoint into consideration. Formally, we define the **view-dependent curvature operator**  $Q$  at point  $M'$  as

$$Q(\mathbf{s}) = D_{\mathbf{s}}\tilde{\mathbf{n}}$$

where  $D_{\mathbf{s}}\tilde{\mathbf{n}}$  is the directional derivative of the normal in the direction  $\mathbf{s}$  in the screen plane. The view-dependent curvature operator  $Q$  at each point  $M'$  is a linear map from the screen plane into the tangent on the Gauss sphere. Note that  $Q$  is not a symmetric linear map.

To relate  $Q$  to functions that we are more familiar with, we observe that

$$D_{\mathbf{s}}\tilde{\mathbf{n}} = D_{\mathbf{r}}\mathbf{n}$$

where  $\mathbf{r} = P^{-1}(\mathbf{s})$ . Therefore  $Q(\mathbf{s}) = D_{\mathbf{s}}\tilde{\mathbf{n}} = D_{\mathbf{r}}\mathbf{n} = K(\mathbf{r}) = K[P^{-1}(\mathbf{s})]$  or

$$\boxed{Q = KP^{-1}} \tag{4.3}$$

$KP^{-1}$  is the 3D expression for view-dependent curvature analogous to the 2D expression  $\frac{-\mathbf{k}(u)}{\mathbf{n}(u) \cdot \mathbf{j}}$ , and is the first derivative of the normal with respect to screen space.  $K$  is the first derivative of the normal with respect to the object space, and  $P^{-1}$  is a result of the chain rule used to change bases. This change of basis causes  $Q$  to be a non-symmetric matrix. Note, that like the 2D equivalent, this matrix is undefined when the view direction is perpendicular to the tangent plane normal  $\mathbf{n}$  because  $P^{-1}$  is undefined at that point.

$Q(\mathbf{s})$  is a vector that describes how the surface normal changes when one moves in direction  $\mathbf{s}$  in the screen. Since we are interested in extrema of the *magnitude* of

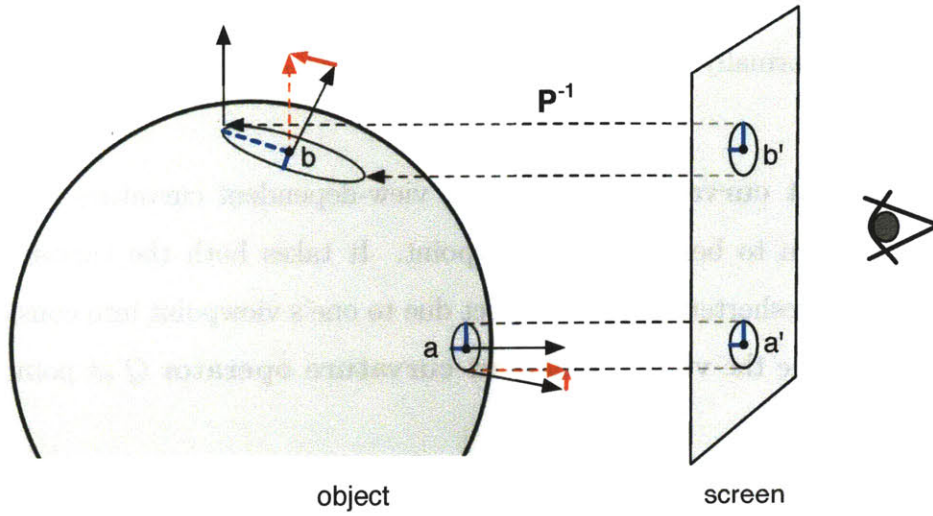


Figure 4-3: The maximum apparent curvature at  $b'$  is much larger than at  $a'$  uniquely because of the perspective projection effect.

the change, we define the maximum view-dependent curvature as

$$\lambda_{A \max} = \max_{\|\mathbf{s}\|=1} \|Q(\mathbf{s})\|^2.$$

The maximum view-dependent curvature  $\lambda_{A \max}$  is equal to the largest eigenvalue of the matrix  $Q^T Q$  and is achieved when  $\mathbf{s}$  is the maximum eigenvector of  $Q^T Q$ . This eigenvector, denoted by  $\mathbf{t}_{A \max}$ , is the maximum view-dependent principal direction. Equivalently, we can define the maximum view-dependent curvature as

$$\lambda_{A \max} = \max_{\|P(\mathbf{r})\|=1} \|K(\mathbf{r})\|^2$$

which is the maximum of the squared normal of the object curvature over an ellipsoid elongated in the view direction (see Fig 4-3).



### 4.3 Apparent Ridges

Following the definition of ridges on the surface, we define **apparent ridges** as the locus of points at which the maximum view-dependent curvature  $\lambda_{A \max}$  assumes a local maximum in the principal view-dependent curvature direction  $\mathbf{t}_{A \max}$ . For simplicity, we will now write  $\mathbf{t}_A$  for  $\mathbf{t}_{A \max}$  and  $\lambda_A$  for  $\lambda_{A \max}$ . Apparent ridges occur where

$$D_{\mathbf{t}_A} \lambda_A = 0.$$

In order to compute  $D_{\mathbf{t}_A} \lambda_A$ , we express  $\lambda_A$  in terms of the eigenexpression

$$\lambda_A = \mathbf{t}_A^T Q^T Q \mathbf{t}_A.$$

Differentiating this equation with respect to  $\mathbf{t}_A$  in screen space, we get

$$D_{\mathbf{t}_A} \lambda_A = (D_{\mathbf{t}_A} \mathbf{t}_A)^T Q^T Q \mathbf{t}_A + \mathbf{t}_A^T (D_{\mathbf{t}_A} Q^T) Q \mathbf{t}_A + \mathbf{t}_A^T Q^T (D_{\mathbf{t}_A} Q) \mathbf{t}_A + \mathbf{t}_A^T Q^T Q (D_{\mathbf{t}_A} \mathbf{t}_A) \quad (4.4)$$

The first and fourth terms of the right hand side go to zero for the following reason: First note that  $(D_{\mathbf{t}_A} \mathbf{t}_A)^T Q^T Q \mathbf{t}_A$  can be rewritten as  $(D_{\mathbf{t}_A} \mathbf{t}_A)^T \lambda_A \mathbf{t}_A$ . Also note that  $2(D_{\mathbf{t}_A} \mathbf{t}_A)^T \cdot \mathbf{t}_A = D_{\mathbf{t}_A} \|\mathbf{t}_A\|^2 = 0$  since  $\|\mathbf{t}_A\|^2 = 1$ . Therefore,  $(D_{\mathbf{t}_A} \mathbf{t}_A)^T \lambda_A \mathbf{t}_A = (D_{\mathbf{t}_A} \mathbf{t}_A)^T Q^T Q \mathbf{t}_A = \mathbf{t}_A^T Q^T Q (D_{\mathbf{t}_A} \mathbf{t}_A) = 0$ . So, the above equation (4.4) becomes

$$D_{\mathbf{t}_A} \lambda_A = 2\mathbf{t}_A^T Q^T (D_{\mathbf{t}_A} Q) \mathbf{t}_A \quad (4.5)$$

The most difficult part is to find the directional derivative of  $Q = KP^{-1}$ . To calculate this we use the product rule as follows:

$$D_{\mathbf{t}_A} Q = D_{\mathbf{t}_A} (KP^{-1}) = (D_{\mathbf{t}_A} K)P^{-1} + K(D_{\mathbf{t}_A} P^{-1}).$$

Intuitively,  $D_{\mathbf{t}_A} K$  and  $D_{\mathbf{t}_A} P^{-1}$  represent how the object curvature and the projection function change as one moves in direction  $\mathbf{t}_A$ . We now look more closely at exactly how to calculate both of these derivatives.

**Finding  $D_{\mathbf{t}_A} K$ .** If we break down  $\mathbf{t}_A$  into its two basis vector components, we get

$$D_{\mathbf{t}_A} K = \mathbf{t}_{Ax} \frac{\partial K}{\partial s_x} + \mathbf{t}_{Ay} \frac{\partial K}{\partial s_y}.$$

Since  $\frac{\partial K}{\partial s_x}$  can be written as  $\frac{\partial K}{\partial \mathbf{r}} = \mathbf{C}$  times change of basis elements, we can write

$$D_{\mathbf{t}_A} K = \mathbf{t}_{Ax} \mathbf{C} \begin{pmatrix} \frac{\partial r_1}{\partial s_x} \\ \frac{\partial r_2}{\partial s_x} \end{pmatrix} + \mathbf{t}_{Ay} \mathbf{C} \begin{pmatrix} \frac{\partial r_1}{\partial s_y} \\ \frac{\partial r_2}{\partial s_y} \end{pmatrix} = \mathbf{C} \left[ \mathbf{t}_{Ax} \begin{pmatrix} \frac{\partial r_1}{\partial s_x} \\ \frac{\partial r_2}{\partial s_x} \end{pmatrix} + \mathbf{t}_{Ay} \begin{pmatrix} \frac{\partial r_1}{\partial s_y} \\ \frac{\partial r_2}{\partial s_y} \end{pmatrix} \right].$$

Since  $P^{-1}$  can be written as

$$P^{-1} = \begin{bmatrix} \frac{\partial r_1}{\partial s_x} & \frac{\partial r_1}{\partial s_y} \\ \frac{\partial r_2}{\partial s_x} & \frac{\partial r_2}{\partial s_y} \end{bmatrix}$$

the equation can be reduced to

$$\boxed{D_{\mathbf{t}_A} K = \mathbf{C} P^{-1} \mathbf{t}_A.} \quad (4.6)$$

**Finding  $D_{\mathbf{t}_A} P^{-1}$ .** We can relate the derivative of  $P^{-1}$  to the derivative of  $P$  as

$$D_{\mathbf{t}_A} P^{-1} = P^{-1} (D_{\mathbf{t}_A} P) P^{-1}$$

see <sup>1</sup>. In order to define  $D_{\mathbf{t}_A} P$ , we note that

$$D_{\mathbf{t}_A} P = D_{\mathbf{t}_A} \begin{pmatrix} \mathbf{t}_1 \cdot \mathbf{u}_x & \mathbf{t}_2 \cdot \mathbf{u}_x \\ \mathbf{t}_1 \cdot \mathbf{u}_y & \mathbf{t}_2 \cdot \mathbf{u}_y \end{pmatrix}$$

by the definition of  $P$ . Now, because  $u_x$  and  $u_y$  are constant, we write

$$D_{\mathbf{t}_A} P = \begin{pmatrix} u_x \\ u_y \end{pmatrix} (D_{\mathbf{t}_A} \mathbf{t}_1, D_{\mathbf{t}_A} \mathbf{t}_2).$$

---

<sup>1</sup>Differentiating both sides of  $P \cdot P^{-1} = I$  gives us  $(D_{\mathbf{t}_A} P^{-1})P + P^{-1}(D_{\mathbf{t}_A} P) = 0$ . Rearranging terms we get  $D_{\mathbf{t}_A} P^{-1} = P^{-1}(D_{\mathbf{t}_A} P)P^{-1}$ .

By change of basis we get

$$D_{\mathbf{t}_A}P = \begin{pmatrix} u_x \\ u_y \end{pmatrix} (D_{\mathbf{r}}\mathbf{t}_1, D_{\mathbf{r}}\mathbf{t}_2) P^{-1}\mathbf{t}_A = \begin{pmatrix} u_x \\ u_y \end{pmatrix} \begin{pmatrix} \frac{\partial \mathbf{t}_1}{\partial \mathbf{r}_1} & \frac{\partial \mathbf{t}_2}{\partial \mathbf{r}_1} \\ \frac{\partial \mathbf{t}_1}{\partial \mathbf{r}_2} & \frac{\partial \mathbf{t}_2}{\partial \mathbf{r}_2} \end{pmatrix} P^{-1}\mathbf{t}_A$$

where the elements  $\frac{\partial \mathbf{t}_1}{\partial \mathbf{r}_1}, \frac{\partial \mathbf{t}_1}{\partial \mathbf{r}_2}, \frac{\partial \mathbf{t}_2}{\partial \mathbf{r}_1}, \frac{\partial \mathbf{t}_2}{\partial \mathbf{r}_2}$  describe how the object basis vectors change as we move along the object in direction  $\mathbf{r}$ . These elements can be shown to equal  $\frac{\partial \mathbf{t}_1}{\partial \mathbf{r}_1} = \frac{Q}{\Delta\lambda}\mathbf{t}_2, \frac{\partial \mathbf{t}_1}{\partial \mathbf{r}_2} = \frac{S}{\Delta\lambda}\mathbf{t}_1, \frac{\partial \mathbf{t}_2}{\partial \mathbf{r}_1} = \frac{-Q}{\Delta\lambda}\mathbf{t}_1, \frac{\partial \mathbf{t}_2}{\partial \mathbf{r}_2} = \frac{-S}{\Delta\lambda}\mathbf{t}_2$  where  $\Delta\lambda = \lambda_{A\max} - \lambda_{A\min}$  and  $Q$  and  $S$  are elements of the curvature derivative tensor  $\mathbf{C}$ . Because vectors  $\mathbf{t}_1$  and  $\mathbf{t}_2$  are orthogonal and lie in the same plane, it makes sense that the derivative of one is in the direction of the other. Now we can simplify  $D_{\mathbf{t}_A}P$  to

$$D_{\mathbf{t}_A}P = \begin{pmatrix} u_x \\ u_y \end{pmatrix} \left[ \frac{1}{\Delta\lambda} \begin{pmatrix} \mathbf{t}_2 Q & \mathbf{t}_2 S \\ -\mathbf{t}_1 Q & -\mathbf{t}_1 S \end{pmatrix} \right] P^{-1}\mathbf{t}_A$$

or

$$\boxed{D_{\mathbf{t}_A}P = \begin{pmatrix} \mathbf{u}_x \\ \mathbf{u}_y \end{pmatrix} \left[ \frac{1}{\Delta\lambda} \begin{pmatrix} \mathbf{t}_2 \\ -\mathbf{t}_1 \end{pmatrix} C(\mathbf{t}_1, \mathbf{t}_2) \right] P^{-1}\mathbf{t}_A} \quad (4.7)$$

Geometrically, this equation takes a vector  $\mathbf{t}_A$ , maps it from the screen space to the object space through  $P^{-1}$ , then evaluates how the object basis vectors change as we move in this object direction, and then maps the change back to screen space by writing it in terms of  $u_x$  and  $u_y$ .

**Final Equation for Apparent Ridges** Putting the entire equation for apparent ridges together, we get

$$\boxed{D_{\mathbf{t}_A}\lambda_A = 2\mathbf{t}_A(KP^{-1})^T ((CP^{-1}\mathbf{t}_A)P^{-1} + KP^{-1}(D_{\mathbf{t}_A}P)P^{-1})\mathbf{t}_A} \quad (4.8)$$

where  $D_{\mathbf{t}_A}P$  is described in equation 4.7. The zeros of this function represent the locations where the value of the view-dependent curvature is at an extremum. These locate our apparent ridge lines.

## 4.4 Discussion

We have derived equations for view-dependent curvature and apparent ridges. To do this, we changed the domain with respect to which we took the derivatives of the surface normal: instead of deriving with respect to an object, we derived with respect to a screen plane.

In essence, we added view dependency to the traditional definition of ridges. On the front facing parts of an object where the object normal points towards to the screen plane, traditional curvature and the view-dependent curvature are the same. Where the object turns away from the screen plane, the view-dependent curvature becomes much larger than the curvature and the view-dependent principle direction is shifted towards the view vector.

In the next chapters we see how the equations of apparent ridges create pleasing line drawings.

# Chapter 5

## Adaptation to Meshes

We adapt view-dependent curvature computations to discrete triangular meshes and present algorithms for the extraction and trimming of apparent ridges.

**Estimating view-dependent curvature** View-dependent curvature is a product of object curvature and an inverse projection. We leverage standard techniques of discrete differential geometry to estimate the curvature  $K$  at each point on the mesh. In practice, we use Rusinkiewicz’s technique [27] but other methods could be used. We then multiply by the inverse projection to obtain  $Q = KP^{-1}$ . We use a weak perspective model to approximate  $P$  at each mesh vertex.

From this matrix  $Q$  we define our maximum apparent curvature  $\lambda_A$  and a maximum apparent curvature direction  $\mathbf{t}_A$  at each point.

**Estimating the view-dependent curvature derivative** The definition of apparent ridges is based on  $D_{\mathbf{t}_A} \lambda_A$ . We estimate this derivative using a finite difference approach as illustrated in Fig. 5-1(a). To compute the derivative at a mesh vertex  $\mathbf{p}$ , we compute the apparent curvature at two points  $\mathbf{w}$  and  $\mathbf{w}'$  on the edges of triangles adjacent to  $\mathbf{p}$  and in the direction of  $\mathbf{t}_{\max}$  (more formally, in the plane defined by  $\mathbf{t}_{\max}$  and the normal at  $\mathbf{p}$ ). Apparent curvature at a point  $\mathbf{w}$  on an edge is obtained by linear interpolation between the two vertices  $\mathbf{v}$  and  $\mathbf{u}$ . We then take the difference between the apparent curvature at  $\mathbf{p}$  with that at  $\mathbf{w}$  and  $\mathbf{w}'$ .

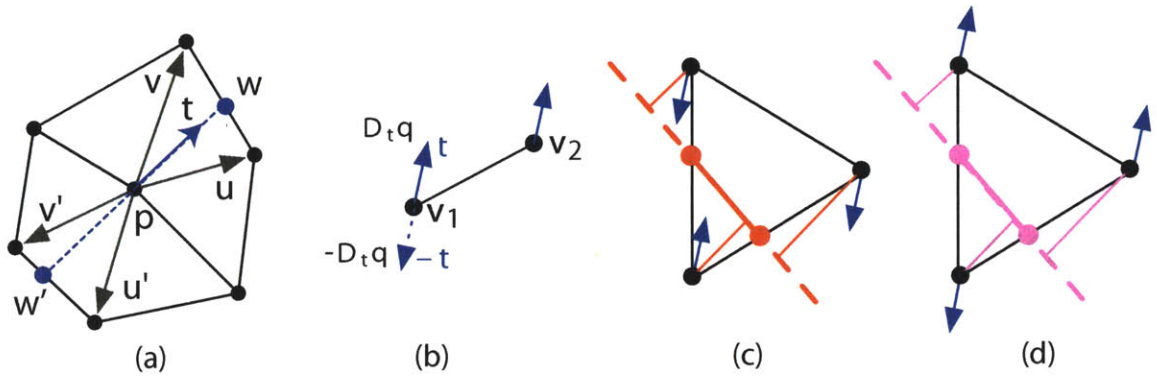


Figure 5-1: How we approximate the apparent curvature derivative, locate zero crossings on mesh and trim minima. For each edge, we flip the  $\mathbf{t}_A$  in the direction of the positive derivative so that it points towards increasing apparent curvature. Then, if the two  $\mathbf{t}_A$  along the edge point in opposite directions, there is a zero crossing. To test for minima, we drop a perpendicular from each vertex to the zero crossing line. If the positive  $\mathbf{t}_A$  at each vertex makes an acute angle with the perpendicular, then the zero crossing is a maxima (as in c). If not, the zero crossing is eliminated as a minima (as in d).

**Finding a consistent  $\mathbf{t}_{\max}$  field** Note that  $\mathbf{t}_{\max}$  is a vector which runs along a line from the vertex in one of two opposite directions (see Fig. 5-1(b)). To make this field consistent across the mesh, we flip  $\mathbf{t}_{\max}$  to point in the direction of the positive derivative, where view-dependent curvature is increasing.

**Locating zero crossings** We use a method inspired by Ohtake et al. [23] to locate the zero crossings of the view-dependent curvature derivative on a mesh given the consistent  $\mathbf{t}_{\max}$  field.

If the  $\mathbf{t}_A$  of both vertices along an edge point in approximately the same direction, then there is no zero crossing. If they point in different directions (defined as  $> 90$  degrees) then there is a zero crossing (see Fig 5-1(c) and (d)). We interpolate the location of the zero crossing using the values of the derivatives at each vertex in a manner similar to Hertzmann and Zorin [13].

**Trimming** The zero crossings locate both the minima and the maxima of the apparent curvature derivatives, but we only want to draw lines at the maxima. To detect

if the zero crossing along an edge in a triangle is a minimum, we drop a perpendicular from each vertex to the zero crossing line. See Fig. 5-1 (c) and (d). If the positive  $\mathbf{t}_A$  at each vertex makes an acute angle with the perpendicular, then the zero crossing is a maxima. If not, the zero crossing is eliminated as a minima.

This test is more robust than that suggested by [23] who approximate the perpendicular by the edge direction.

**Thresholding** Even after minima are trimmed, a lot of lines remain. This is because our method finds both local maxima and global maxima. Informally, not only do we want points where the magnitude of the view-dependent curvature is a maximum, but we also want this maximum to be high. We therefore threshold the value of the view-dependent curvature. This threshold is scaled by the feature size of the mesh (average edge length) to make it dimensionless. All line definitions require a similar thresholding.

**Implementation** In practice, we build on the Princeton's Real Time Suggestive Contour (RTSC) software with the trimesh2 libraries in order to implement apparent ridges and compare them to other lines. The RTSC software computes normals, curvature and the derivative of curvature, and displays many types of lines on 3D triangular meshes. We added all the necessary computations and tests in order to draw apparent ridge line drawings.





# Chapter 6

## Results and Analysis

We now present a closer look at how apparent ridges compare to the other major feature lines used for line drawing. Specifically, we compare drawings of apparent ridges with drawings of suggestive contours and contours, and drawings of ridges and valleys and contours. In order to ensure a fair comparison, we thresholded each image to match the number of gray pixels per image.

### 6.1 Silhouettes and Contours

Silhouettes and the more general contours are located where the normal is perpendicular to the view direction. As we move towards the contour, the view-dependent

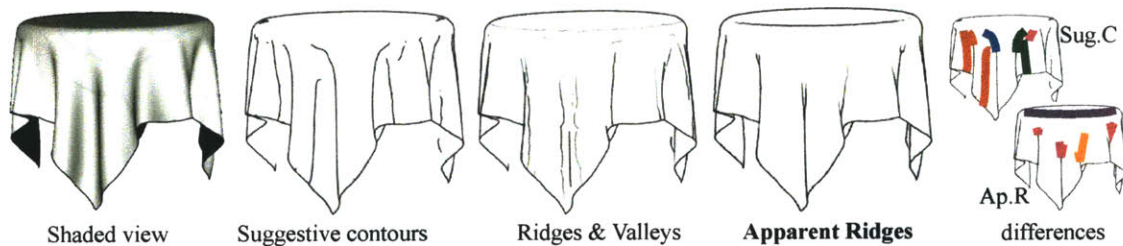
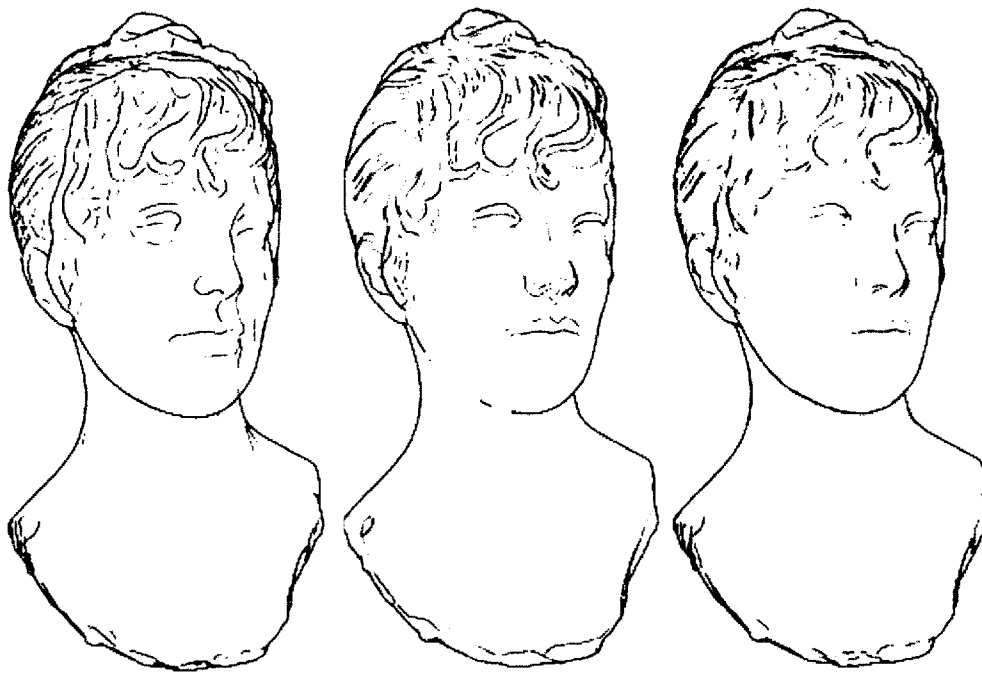


Figure 6-1: Tablecloth model. Notice how the apparent ridges convey both the rim of the table and the drapery of the tablecloth. Note that the missing occluding contour on the back of the cloth is an artifact of our software and not of the line definitions. The difference figures on the right highlight the differences between the suggestive contours and apparent ridges.

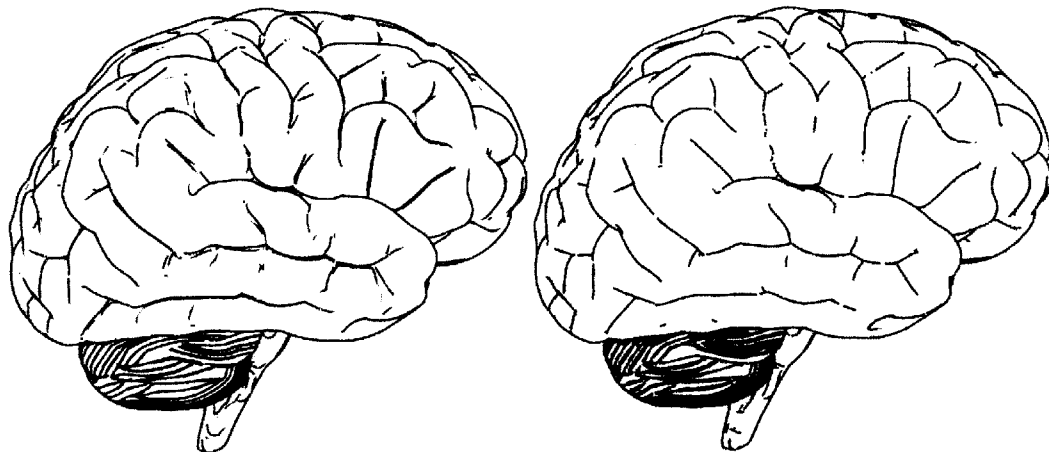


**Suggestive contours**

**Ridges & Valleys**

**Apparent Ridges**

Figure 6-2: Bust model. Note that the nose of the ridge and valley drawing is unnaturally sharp. Suggestive contours don't present a pleasing drawing on the right side of the image.



**Suggestive contours**

**Apparent Ridges**

Figure 6-3: Brain model. Note the double lines that are drawn with suggestive contours. Both suggestive contours and apparent ridges produce pleasing drawings.

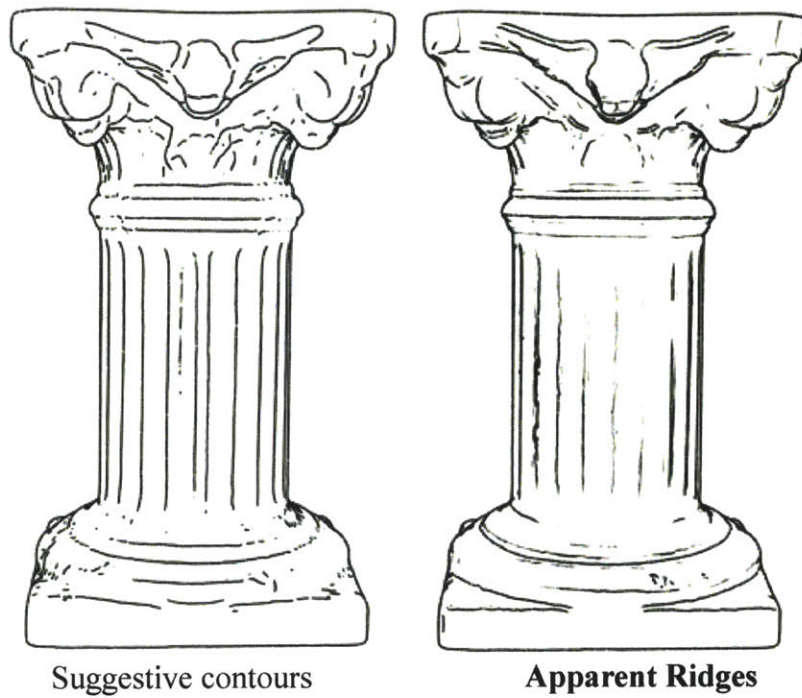


Figure 6-4: Column model. Suggestive contours do well on the center of the pillar, but apparent ridges do better on the base and on the face of the angel.

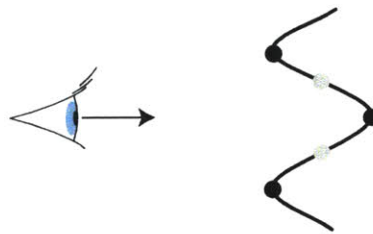


Figure 6-5: Given a slice of an object, suggestive contours are drawn at the gray locations, while apparent ridges are drawn at the black. This leads suggestive contours to draw two lines in areas that surround a valley, but lead apparent ridges to draw two lines in areas that surround an inflection point.

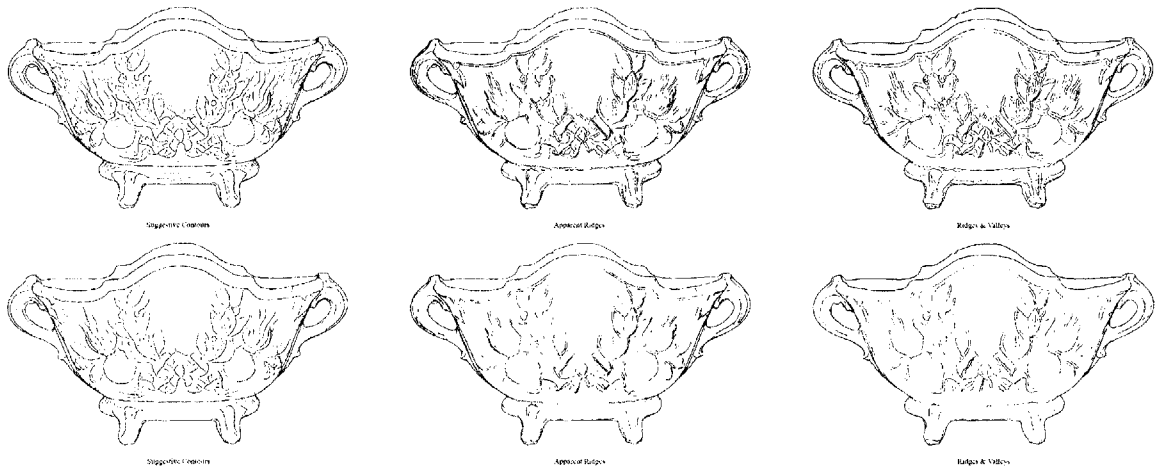


Figure 6-6: Flowerpot model. Notice how different lines need to be thresholded at different levels. The second row has a higher thresholding for all lines.

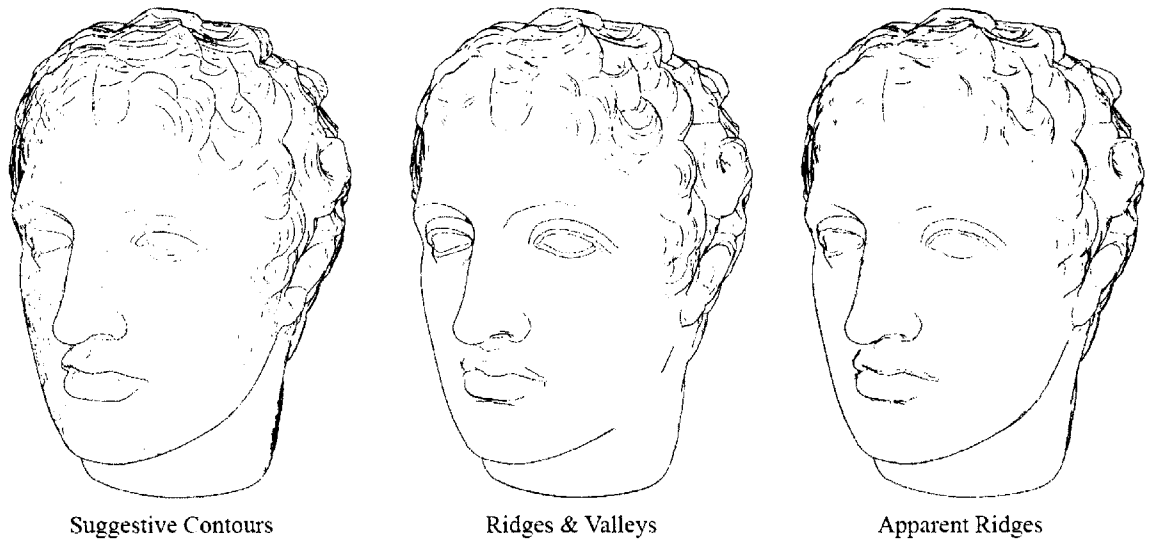


Figure 6-7: Head model. Apparent ridges make an elegant line drawing of this model.

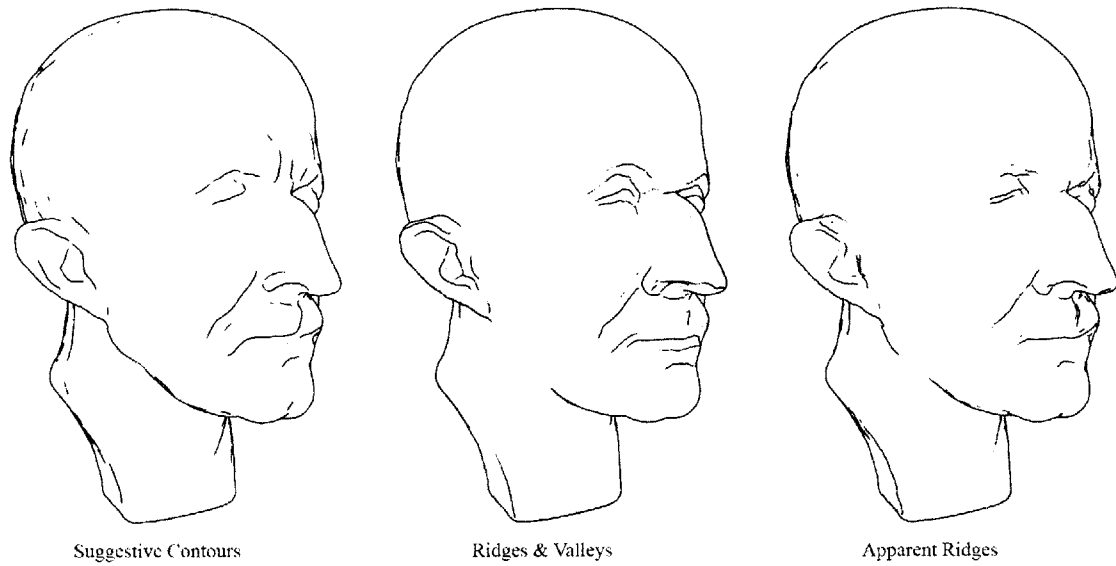


Figure 6-8: Max Planck model. Apparent ridges render the facial features of the model with more pertinent detail than do the suggestive contours. Ridge and valley lines are overly sharp.

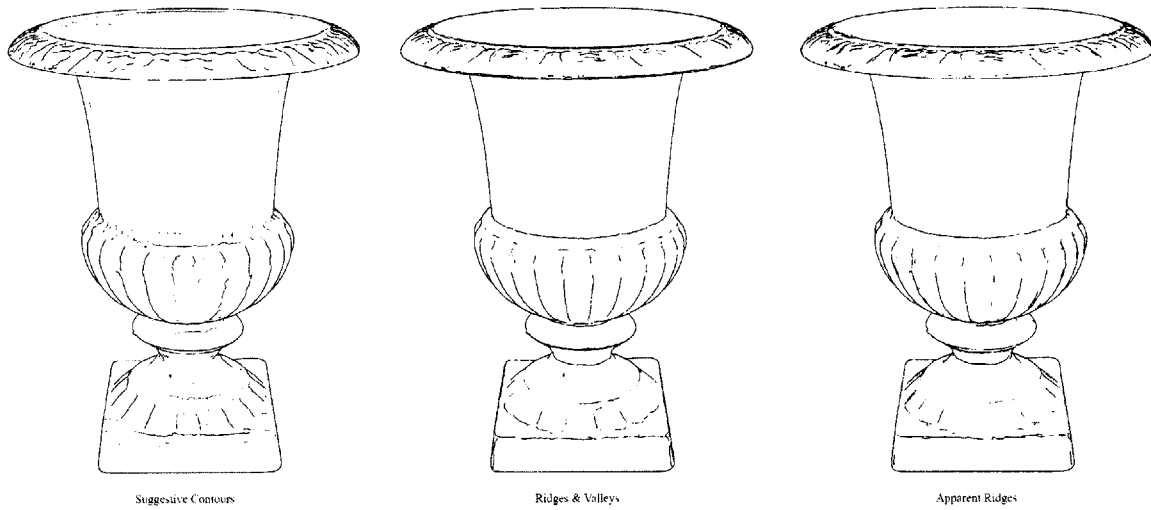


Figure 6-9: Vase model. Both ridge and valley and apparent ridge lines convey the details of the vase, and particularly the base, more cleanly and directly than do suggestive contours.

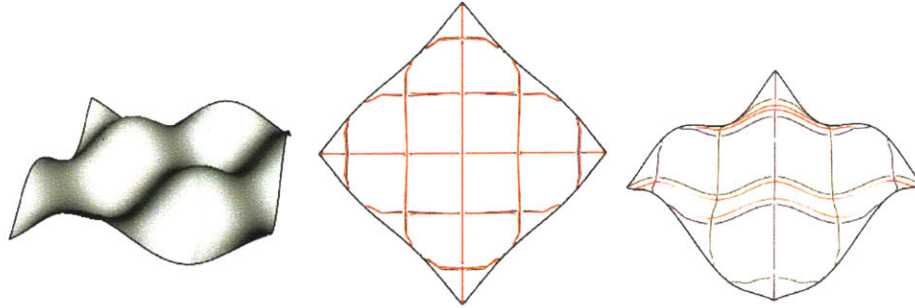


Figure 6-10: Sine wave mesh with ridges and valleys (purple and brown) and apparent ridges (red). Note displacement due to the perspective. Lines are collocated at head on views but shift as the object turns away from the viewer.

curvature approaches infinity because of the perspective effect. Hence apparent ridges include contours (even though the apparent curvature has a singularity at the exact contour.) This is interesting because it allows apparent ridges to stand alone, whereas other lines must combine with contours.

## 6.2 Ridges and Valleys

Mathematically, apparent ridges are very related to traditional ridges: they share the same definition modified by a projection from the object to the screen. By taking this perspective into account however, apparent ridges are perceptually more pertinent than ridges and valleys.

When the effect of the projection is small, at front facing parts of the model where the normal points towards the viewer, ridges and apparent ridges are similar. On parts of the object that turn away from the viewer, the apparent ridges and ridges differ. This can be seen in the example of the sinewave mesh in (Fig. 6-10).

Ridges capture important features of a model, but they need to be adjusted to be more perceptually pleasing. As seen in the tablecloth (Fig. 6-1), ridges successfully capture the important rim of the table, but they also capture all the arbitrary maximal folds of the cloth which have little perceptual importance. Apparent ridges merge these ridges into the much more important nearby contour. Because ridges and valleys are fixed on a object, they can appear as artificial surface markings and produce boxy

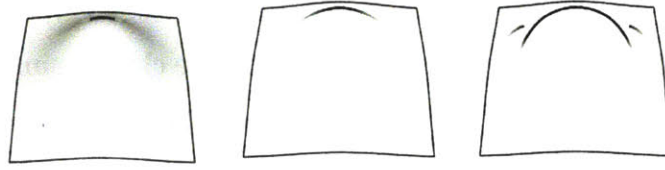


Figure 6-11: Gaussian bump from left to right: shaded view with occluding contour, suggestive contours and apparent ridges.

looking nose and mouth renderings of the Bust (Fig. 6-2) and Max Planck (Fig. 1-3) models. Apparent ridges create more appealing drawings in these examples.

Apparent ridges are also defined in some cases where ridges are ill-defined. On a symmetric gaussian bump (Fig. 6-11), the apparent ridge lines extend the contour, but because there is no maximum of curvature on the symmetric bump, ridge lines don't exist.

### 6.3 Suggestive Contours

Apparent ridges and suggestive contours are fundamentally different lines. Mathematically, suggestive contours are located where the radial curvature is zero, whereas apparent ridges are located at the maxima of the view-dependent curvature in its principal direction.

Intuitively, suggestive contours look at an extremum of the *normal*, where we look at the extremum of the *normal variation*. Suggestive contours look at curvature in the direction of the *view vector*, where we look at curvature in the direction of  $\mathbf{t}_A$ , the *highest normal variation as seen from the screen plane*. These directions have different definitions, but sometimes they align: foreshortening happens along the view direction, thus inflating the view-dependent curvature in that direction and making it more likely to be the direction of  $\mathbf{t}_A$ . However, even when these directions are the same, suggestive contours find where the curvature in that direction is *zero*, where we find where the curvature is *maximum*.

Suggestive contours and apparent ridges trade off situations where they draw

single or double lines. In the model of the brain (Fig. 6-10), suggestive contours create double lines at all the crevices, while we provide cleaner single lines. We locate the valley of the crevice, while suggestive contours locate the two inflection points surrounding the valley. However, in the column model, (Fig. 6-4), we provide two apparent ridge lines for the beveled wing surface (at the ridge and valley), whereas suggestive contours provide a simpler one (at the inflection).

One of the strong attributes of suggestive contours is that they extend contour lines. Apparent ridges do as well (see (Fig. 6-11)). Apparent ridges lengthen the contour because nearby places of “almost contour” are locations where the perspective foreshortening effect is highest, leading to high view-dependent curvature values.

Suggestive contours miss important features in convex regions of an object. For example, no lines are drawn on the rim of the table in Fig. 6-1, whereas apparent ridges successfully locate important features in these examples.

Given that the suggestive contours and apparent ridges are so different, it is hard to say that one is clearly better than the other. Rather, both provide an interesting choice of lines with different strengths and weaknesses.

## 6.4 Comparison to edge detection

Given head-on illumination with a single light source at the viewpoint and lambertian shading, suggestive contours are drawn in the shaded areas of an object. As seen in the tablecloth example, dark shaded areas correspond directly to the suggestive contour lines. These lines only make sense given a certain shading setup; if the light is moved, the suggestive contour lines would seem arbitrary. On the other hand, apparent ridge lines are drawn where shading edges are detected as an average over many light sources, suggesting that apparent ridges are drawn at locations that are important independent of the light direction.

In Fig. 6-12, we show a Monte-Carlo experiment where a diffuse surface is rendered from a given viewpoint with thousands of random lighting configurations. The average output of a Canny edge detector [2] on those thousands of images matches remarkably



well the lines extracted with our technique. We have performed similar experiments with real photographs with flash illumination and compared Canny edge detection to a drawing performed on a scan of the object (Fig. 6-13). While the viewpoints are not matched perfectly, we can see that the two extraction approaches agree, which confirms that our lines are related to edges in shaded views. This also draws interesting relation to Raskar et al.'s NPR camera [26].

## 6.5 Performance evaluation

Ridges and valleys and suggestive contours are quick to compute. Though the computations require information about the curvature and the curvature derivative, these values do not change with each viewpoint and can be precomputed for an object. On the other hand, apparent ridges rely on the view-dependent curvature and its derivative which change, and must be recomputed, for each viewpoint. With our unoptimized code on a 2.33 GHz Intel Core 2 Duo Macintosh, apparent ridges are computed in real time for small meshes,  $\sim 1.5$  seconds on 50,000 polygon meshes, and  $\sim 9$  second for 250,000 polygon meshes.

In addition to performance, one limitation of apparent ridges compared to suggestive contours is that they involve higher-order derivatives, which makes them more prone to numerical noise in digital meshes.

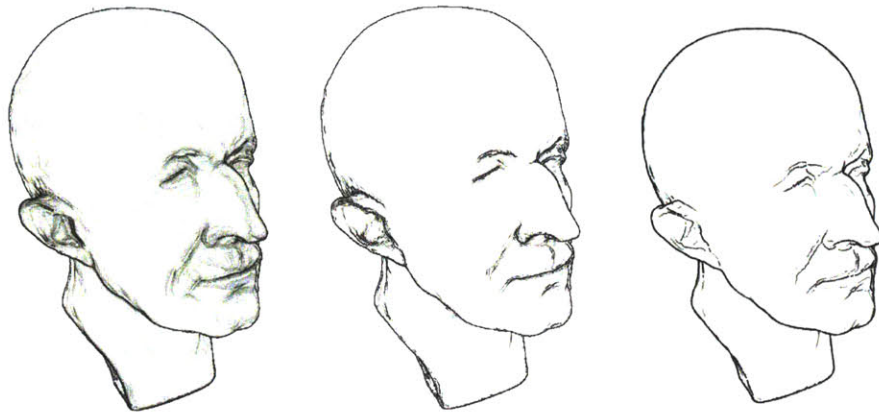


Figure 6-12: Experiments with Canny edge detection. (a) Average edge image for a single light source and Lambertian shading. (b) Average edge image for 10 random light sources and Lambertian shading. (c) Apparent ridge drawing.

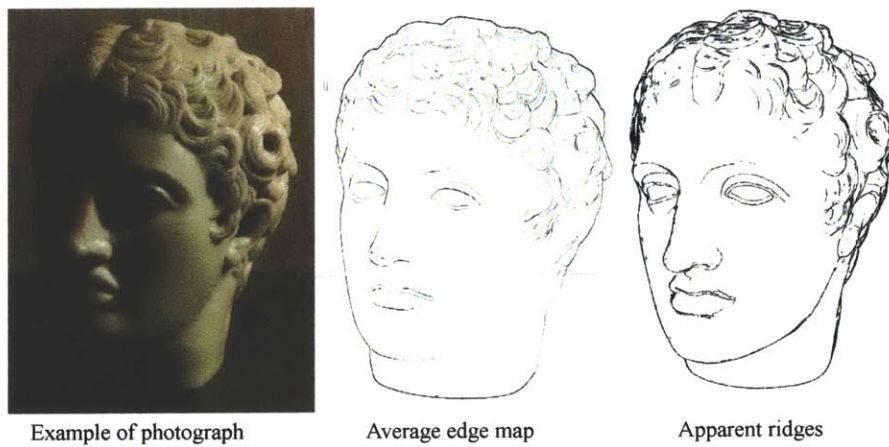


Figure 6-13: Comparison between the average Canny edge detection on 120 photographs with different lighting and our apparent ridges.

# Chapter 7

## Conclusions and Future Work

We have introduced apparent ridges for non-photorealistic line drawings. Apparent ridges produce visually pleasing line drawings, and capture important information about an object's shape. A variety of line types, such as silhouette contours and the sharp ridges of a polyhedron, are special cases of our single definition. Apparent ridges are related to, but distinct from, suggestive contours. Both are based on a combination of view dependent and view independent factors. Both produce pleasing images, but in many cases we find that the apparent ridge images are more natural in appearance. Finally, apparent ridges enhance ridge and valley lines to be more perceptually pertinent. Traditional and apparent ridges are based on the curvature of the object surface, but in addition apparent ridges account for foreshortening of the object when it is viewed from a specific location. This creates a smoother and richer picture than the sharp ridges and valleys alone.

Future work includes doing perceptual experiments to verify whether apparent ridges alone better convey shape information than other feature lines. Adding a notion of lighting into the equations and images, and incorporating notions of scale and line importance are also exciting future challenges. The fact that apparent curvature maxima are closely defined by the viewpoint is encouraging for animation because it could eliminate the flickering of contour lines which only appear briefly. Our initial findings are that apparent ridges work very well for an animated camera. We also would like to try more sophisticated thresholding techniques to get better drawings,

and to explore the use of view-dependent curvature to describe mesh saliency. Finally, we strive to make computations of apparent ridges possible in real-time.

# Bibliography

- [1] Michael Burns, Janek Klawe, Szymon Rusinkiewicz, Adam Finkelstein, and Doug DeCarlo. Line drawings from volume data. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 512–518, New York, NY, USA, 2005. ACM Press.
- [2] J. Canny. A computational approach to edge detection. In *RCV87*, pages 184–203, 1987.
- [3] Roberto Cipolla and Peter Giblin. *Visual motion of curves and surfaces*. Cambridge University Press, New York, NY, USA, 2000.
- [4] Doug DeCarlo, Adam Finkelstein, and Szymon Rusinkiewicz. Interactive rendering of suggestive contours with temporal coherence. In *NPAR 2004*, pages 15–24, June 2004.
- [5] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Transactions on Graphics*, 22(3):848–855, July 2003.
- [6] Philippe Decaudin. Cartoon looking rendering of 3D scenes. Research Report 2919, INRIA, June 1996.
- [7] Manfredo P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976. 503 pages.
- [8] Frédo Durand, Nicolas Holzschuch, Cyril Soler, Eric Chan, and François X. Sillion. A frequency analysis of light transport. *ACM Transactions on Graphics*, 24(3):1115–1126, August 2005.

- [9] R. W. Fleming, A. Torralba, and E. H. Adelson. Specular reflections and the perception of shape. *Journal of Vision*, 4(9):798-820, 2004.
- [10] A. Girshick, V. Interrante, S. Haker, and T. Lemoine. Line direction matters: an argument for the use of principal directions in 3d line drawings, 2000.
- [11] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter S. Shirley, and Rich Riesenfeld. Interactive technical illustration. In *1999 ACM Symposium on Interactive 3D Graphics*, pages 31-38, April 1999.
- [12] J. Gravesen and M. Ungstrup. Constructing invariant fairness measures for surfaces. *Advances in Computational Mathematics*, 17:67-88, 2002.
- [13] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 517-526, July 2000.
- [14] Victoria Interrante, Henry Fuchs, and Stephen Pizer. Enhancing transparent skin surfaces with ridge and valley lines. In *VIS '95: Proceedings of the 6th conference on Visualization '95*, page 52, Washington, DC, USA, 1995. IEEE Computer Society.
- [15] Robert D. Kalnins, Philip L. Davidson, Lee Markosian, and Adam Finkelstein. Coherent stylized silhouettes. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 856-861, New York, NY, USA, 2003. ACM Press.
- [16] Robert D. Kalnins, Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes, and Adam Finkelstein. Wysiwyg npr: Drawing strokes directly on 3d models. *ACM Transactions on Graphics*, 21(3):755-762, July 2002.
- [17] J. J. Koenderink, A. J. van Doorn, C. G. Christou, and J. S. Lappin. Shape Constancy in Pictorial Relief. *Perception*, 25(2):155-164, February 1996.
- [18] Jan J. Koenderink. *Solid shape*. MIT Press, Cambridge, MA, USA, 1990.

- [19] J.J Koenderink. What does the occluding contour tell us about solid shape? *Perception*, 13:321–330, 1984.
- [20] J Lewis. Personal communication. *Boeing Computer Services, Seattle, Washington*, November 1993.
- [21] Antonio M. Lopez, Felipe Lumbreras, Joan Serrat, and Juan J. Villanueva. Evaluation of methods for ridge and valley detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4):327–335, 1999.
- [22] O. Monga, S. Benayoun, and O.D. Faugeras. From partial derivatives of 3-d density images to ridge lines. *Proceedings CVPR*, pages 354–359, June 1992.
- [23] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph.*, 23(3):609–612, 2004.
- [24] M. Pauly, R. Keiser, and M. Gross. Multi-scale feature extraction on point-sampled surfaces, 2003.
- [25] Ramesh Raskar. Hardware support for non-photorealistic rendering. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 41–47, New York, NY, USA, 2001. ACM Press.
- [26] Ramesh Raskar, Kar-Han Tan, Rogerio Feris, Jingyi Yu, and Matthew Turk. Non-photorealistic camera: depth edge detection and stylized rendering using multi-flash imaging. *ACM Transactions on Graphics*, 23(3):679–688, August 2004.
- [27] Szymon Rusinkiewicz. Estimating curvatures and their derivatives on triangle meshes. In *Symposium on 3D Data Processing, Visualization, and Transmission*, 2004.
- [28] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, volume 24, pages 197–206, August 1990.

- [29] Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, and John Snyder. Silhouette clipping. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 327–334, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.
- [30] M.C. Sousa and P. Prusinkiewicz. A few good lines: Suggestive drawing of 3d models. *Computer Graphics Forum (Proc. of EuroGraphics '03)*, 22(3):xx–xx, 2003.
- [31] Thomas Strothotte and Stefan Schlechtweg. *Non-Photorealistic Computer Graphics. Modelling, Animation, and Rendering*. Morgan Kaufmann, 2002.