

# **Line Drawings from 3D Models**

SIGGRAPH 2008 Class

Tuesday, 12 August, 8:30 AM – 12:15 PM

## **Organizer**

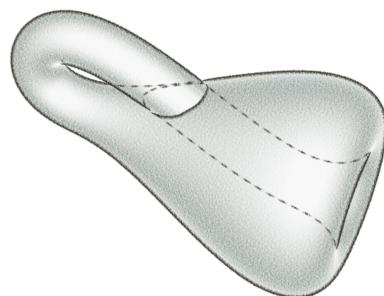
Szymon Rusinkiewicz  
Princeton University

## **Lecturers**

Forrester Cole  
Princeton University

Doug DeCarlo  
Rutgers University

Adam Finkelstein  
Princeton University



## Course Description

Nonphotorealistic rendering techniques, including line drawings, can be remarkably efficient at conveying shape and meaning with a minimum of visual distraction. This class will describe techniques for automated rendering of 3D models using a number of sparse line drawing styles, for both artistic and illustrative purposes. We will mathematically define lines such as silhouettes, contours, creases, suggestive contours and highlights, and apparent ridges and valleys. We then describe algorithms for finding lines efficiently, including object- and image-space methods, and discuss methods for stylization and level-of-detail control. Finally, we provide a brief introduction to concepts of visual perception, including the information content of line drawings and the effects of abstraction and detail on attention.

## Prerequisites

Basic familiarity with the graphics pipeline and some knowledge of calculus and linear algebra.

## Instructor Bios

**Forrester Cole** is a fourth-year Ph. D. student in the graphics group at Princeton University. His research interests are mainly in non-photorealistic rendering, particularly interactive NPR algorithms and algorithms for artistic stylization and abstraction. He is currently investigating improved algorithms for interactive, stylized line drawing. Prior to coming to Princeton, Forrester was a programmer at Pandemic Studios, where he wrote engine code for PlayStation 2 and Xbox games.

**Doug DeCarlo** received BS degrees in computer science and computer engineering from Carnegie Mellon in 1991, and his Ph. D. in computer science from the University of Pennsylvania in 1998. He is currently an associate professor in the Department of Computer Science with a joint appointment in the Center for Cognitive Science at Rutgers University. His research in computer graphics explores how accounts of human perception and communication can inform the design of computer systems that engage in effective visual communication.

**Adam Finkelstein** is an associate professor in the Computer Science Department at Princeton University. His current research focuses on non-photorealistic rendering and animation. Adam received a doctorate from the University of Washington in 1996. From 1987 to 1990, he worked at Tibco developing software for people who trade stock. He received a BA in 1987 from Swarthmore College.

**Szymon Rusinkiewicz** is an associate professor of Computer Science at Princeton University. His work focuses on acquisition and analysis of the 3D shape and appearance of real-world objects, including the design of capture devices and data structures for efficient representation. He also investigates algorithms for processing complex datasets of shape and reflectance, including registration, matching, completion, symmetry analysis, and sampling. In addition to data acquisition, his research interests include real-time rendering and perceptually-guided depiction. He obtained his Ph. D. from Stanford University in 2001.

## Acknowledgements

Thanks to Rob Kalnins, Lee Markosian, Anthony Santella, and especially Mike Burns. Partially supported by the National Science Foundation through grants HLC 0308121, CCF 0347427, and CCF 05041185. Some photographs in slides for Part VII courtesy <http://philip.greenspun.com>.

## **Course Syllabus**

- I. Introduction to the study of lines (:15 — Finkelstein)
  - Non-photorealistic rendering
  - What is conveyed? (shape, markings, material, shading)
  - Approaches to the study of line drawings
- II. Artists' line drawings (:20 — Cole)
  - Studying what lines artists draw
  - Evaluation
- III. Mathematical description of lines (:45 — Rusinkiewicz)
  - Silhouettes and occluding contours
  - Basics of differential geometry for surfaces in 3D
  - Suggestive contours and highlights; principal highlights
  - Surface ridges and valleys
  - Apparent ridges and valleys
  - Ridges and valleys of illumination
  - Other lines (creases, surface markings, material boundaries, topo (isoelevation) lines, principal curvature lines, etc.)
- IV. Perception of line drawings (:25 — DeCarlo)
  - How much information is there in drawings?
  - Ambiguities (bas-relief, line labeling, etc.)
  - Psychophysical studies (gauge figures, etc.)
- Break — (:15)
- V. Algorithms for extracting lines (:30 — Rusinkiewicz)
  - Image-space algorithms
  - Brute force, "marching triangles"
  - Randomized methods
  - Hierarchical methods
  - Methods based on the Gauss map and other mappings
  - Determining line visibility
- VI. Stylization of line drawings (:25 — Finkelstein)
  - Parameterization
  - Temporal coherence
  - User interaction
- VII. Abstraction and evaluation (:30 — DeCarlo)
  - Abstraction in NPR
  - Evaluation using eye tracking
- VIII. Controlling detail and attention (:20 — Cole)
  - Control of line density
  - Stroke simplification
  - Stylized focus

# Notes on Differential Geometry

Defining and extracting suggestive contours, ridges, and valleys on a surface requires an understanding of the basics of differential geometry. Here we collect the necessary background, specialized to the case of surfaces in 3D and considering only orthonormal bases. For more details, consult [Cipolla and Giblin 2000, do Carmo 1976, Koenderink 1990].

We begin by considering a smooth and closed surface  $S$  and a point  $\mathbf{p} \in S$  sitting on the surface.

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

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

$$D_{\mathbf{x}}\mathbf{n} = n_u \mathbf{s}_u + n_v \mathbf{s}_v, \quad (1)$$

where

$$\begin{bmatrix} n_u \\ n_v \end{bmatrix} = \begin{bmatrix} L & M \\ M & N \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}, \quad (2)$$

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

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

$$\begin{aligned} \mathbf{II}(\mathbf{x}_1, \mathbf{x}_2) &= (D_{\mathbf{x}_1}\mathbf{n}) \cdot \mathbf{x}_2 = (D_{\mathbf{x}_2}\mathbf{n}) \cdot \mathbf{x}_1 \\ &= \begin{bmatrix} u_1 & v_1 \end{bmatrix} \begin{bmatrix} L & M \\ M & N \end{bmatrix} \begin{bmatrix} u_2 \\ v_2 \end{bmatrix}. \end{aligned} \quad (3)$$

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

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

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

Notice how the length and sign of  $\mathbf{x}$  do not affect the normal curvature. On a smooth surface, the normal curvature varies smoothly with direction  $\mathbf{x}$ , and ranges between the principal curvatures  $\kappa_1$  and  $\kappa_2$  at  $\mathbf{p}$ . These are realized in their respective principal curvature directions  $\mathbf{e}_1$  and  $\mathbf{e}_2$ , which are perpendicular and unit length.

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

The vector  $D_{\mathbf{x}}\mathbf{n}$  can be broken into two components; in the direction of  $\mathbf{x}$ , and perpendicular to it. The length of the component in the direction of  $\mathbf{x}$  is simply the normal curvature  $\kappa_n$ . The length of the perpendicular component is known as the *geodesic torsion*  $\tau_g$ , and describes how much the normal vector tilts to the side as you move in the direction of  $\mathbf{x}$ . If we define the perpendicular direction  $\mathbf{x}_{\perp}$  as:

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

then we have:

$$D_{\mathbf{x}}\mathbf{n} = \mathbf{II}(\mathbf{x}) = \kappa_n(\mathbf{x}) \mathbf{x} + \tau_g(\mathbf{x}) \mathbf{x}_{\perp} \quad (4)$$

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

**Principal coordinates:** Using the principal directions  $\{\mathbf{e}_1, \mathbf{e}_2\}$  as the local basis leads to *principal coordinates*. In principal coordinates, the matrix in equations (2) and (3) is diagonal with the principal curvatures as entries:

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

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

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

and the following for the geodesic torsion:

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

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

**Third-order structure:** In order to define ridges and valleys, and to analyze how suggestive contours move across a surface, we will need additional notation that describes *derivatives of curvature*. The gradient  $\nabla \kappa_r$  is a vector in the tangent plane that locally specifies the magnitude and direction of maximal change in  $\kappa_r$  on the surface.

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

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

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

This leads to the first-order approximation of the matrix in equation (3) towards  $\mathbf{x} = u\mathbf{e}_1 + v\mathbf{e}_2$  as:

$$\begin{bmatrix} L & M \\ M & N \end{bmatrix} \approx \begin{bmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{bmatrix} + u \begin{bmatrix} P & Q \\ Q & S \end{bmatrix} + v \begin{bmatrix} Q & S \\ S & T \end{bmatrix} \quad (5)$$

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

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

where

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

and

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

## References

- CIPOLLA, R., AND GIBLIN, P. J. 2000. *Visual Motion of Curves and Surfaces*. Cambridge University Press.
- DO CARMO, M. P. 1976. *Differential Geometry of Curves and Surfaces*. Prentice-Hall.
- GRAVESEN, J., AND UNGSTRUP, M. 2002. Constructing invariant fairness measures for surfaces. *Advances in Computational Mathematics* 17, 67–88.
- KOENDERINK, J. J. 1990. *Solid Shape*. MIT press.
- MORETON, H., AND SÉQUIN, C. 1992. Functional optimization for fair surface design. In *Proceedings of ACM SIGGRAPH 1992*, vol. 26, 167–176.

# Published Research on Line Drawings from 3D Data

The following list contains papers describing algorithms for producing line drawings from 3D data, as well as research on the perception of line drawings. Papers marked with an asterisk (\*) are the most closely related to this course, and are recommended reading. Detailed bibliographical information follows this annotated list.

## Before 1990

- [Appel 1967] Introduces an efficient algorithm for discovering hidden lines in drawings of 3D shapes by propagation of *quantitative invisibility*.
- [Waltz 1975] Applies the technique of constraint satisfaction to search for consistent labelings of lines in line drawings of polyhedral scenes. It works from exhaustive catalogs of all possible configurations of line junctions.
- [Barrow 1981] Describes algorithms for the reconstruction of surfaces from line drawings by making a range of assumptions (i.e. about surface smoothness).
- [Stevens 1981] A description of how repeated arrangements of lines in line drawings can give rise to shape percepts. Parallel lines (cutting lines), geodesics and lines of curvature are investigated.
- [Koenderink 1982] A discussion of what lines artists might be drawing. It includes a proof that ending contours must terminate in a concave way, yet points out that artists often draw convex endings.
- \* [Koenderink 1984] Seminal paper establishing the relationship between the curvature of the contour in the image (the apparent curvature) and the Gaussian curvature of the corresponding point on the surface.
- [Pearson 1985] Examined the use of image valleys for producing line drawings from photographs.
- [Canny 1986] Classic method for edge-detection in images.
- [Malik 1987] The generalization of line labeling [Waltz 1975] to curved surfaces. It also includes an algorithm for pruning away unlikely interpretations.

## 1990

- [Dooley 1990] Describes automatic generation of boundary, silhouette, discontinuity, and iso-parametric lines to depict 3D objects. Main focus is on stylization with an importance, line type, and hiddenness matrix.
- [Elber 1990] Defines boundary, silhouette, discontinuity, and iso-parametric curves mathematically. Describes curve extraction, intersection, and visibility methods for B-spline surfaces. Visibility propagates quantitative invisibility.
- [Koenderink 1990] Well-known and well-respected book on geometric and perceptual aspects of shape.

- \* [Saito 1990] Uses 2D image processing algorithms to draw discontinuities, edges, and iso-parametric lines over 2D renderings of 3D models.

## 1992

- [Knill 1992] Describes perceptual experiments that suggest our perceptual systems assume that surface markings (i.e. reflectance boundaries) lie along geodesics on the surface.
- [Rossignac 1992] Introduces a method of hidden-line removal for contours using the hardware z-buffer.

## 1994

- [Hsu 1994] Introduces a comprehensive framework for describing and rendering heavily stylized complex 2D strokes.
- [Saito 1994] Proposes interactive visualization of volumetric data by hierarchically sampling areas of interest from the data and rendering oriented lines.
- \* [Salisbury 1994] Presents interactive user-driven methods for creating line drawings by painting texture and tone over a 2D reference image, with optional extracted edges.
- \* [Winkenbach 1994] Describes general principles for creating computer generated line drawings from 3D models. Introduces stroke textures to apply resolution-dependent texture and tone to 3D polygonal surfaces. Describes *indication*, a technique for elision of detail in drawn textures to enhance comprehensibility of line drawings.

## 1995

- [Elber 1995a] Describes generation of line-art drawings using iso-parametric curves to cover freeform surfaces. A lighting model affects the density of curves and noise is added to avoid a synthetic look.
- [Elber 1995b] Presents several ideas for the improvement of line drawings from 3D surfaces. Suggests performing depth cuing by modulating line width and intensity, drawing thin light strokes for background lines. Trimming of background lines near intersections is also suggested.
- [Interrante 1995] Suggests using ridge and valley lines as an aide to transparent isosurface visualization. Lines are extracted from an isosurface and opacity is modulated by principal curvature.

## 1996

- [Interrante 1996] Describes rendering of transparent isosurfaces using short strokes directed along principal curvature with length modulated by principal curvature magnitude.
- [Koenderink 1996] A psychophysical study that measures and compares shape percepts from (external) silhouettes, line drawings and shaded images. It uses the same psychophysical tasks as [Koenderink 2001].

- [Salisbury 1996] Suggests representing pen-and-ink drawings with a grayscale image, a set of discontinuity segments, and associated stroke textures for the purposes of producing consistent drawings at any scale and resolution. A novel edge-reconstruction algorithm allows low-resolution grayscale images to be up-sampled while maintaining sharp discontinuities. Line drawings are created by rendering the stroke textures on the grayscale image.

- [Thirion 1996] Introduces the marching lines algorithm for extraction of line loops along intersections of two closed surfaces in 3D space. Describes drawing crest lines as an aide to visualizing isosurfaces by intersecting a maximal curvature surface with the isosurface.

- [Winkenbach 1996] Describes rendering line-art drawings from parametric freeform surfaces. Details controlled-density hatching to modulate line width based on proximity to other lines. Also introduces a method to construct a planar map for a parametric surface and a method of rendering shadows with strokes.

## 1997

- [Ma 1997] Argues for the benefit of extracting feature lines (contours, ridges and valleys) from unstructured triangular meshes. Presents some results in the context of augmenting fluid flow visualization.
- \* [Markosian 1997] Introduces a fast randomized algorithm for finding and tracing contours on polygonal surfaces. Exploits frame-to-frame temporal coherence and uses random probes to locate contour seed points and trace contours along the surface. Uses an accelerated quantitative invisibility propagation method to differentiate hidden lines.
- [Masuch 1997] Describes a system for creating animated line-drawings from 3D polygonal objects.
- [Rieger 1997] Presents comprehensive derivations and analysis of intensity ridges and valleys, edges, and other lines defined on images.

## 1998

- [Belyaev 1998] Details and derives formulas for ridges and valleys on implicit surfaces. Uses these to generate ridges and valleys on an implicit surface as the intersection curves of a ridge and valley surface with the implicit surface.
- [Bremer 1998] Describes a method of extracting silhouettes from implicit surfaces using a seed-and-traverse method. Seeding is performed by ray-surface intersection and walking along the surface to a silhouette. Traversal is done by Euler integration with a penalty to avoid iterative drift.
- [Elber 1998] Describes a method for creating line art renderings from freeform parametric surfaces based on a uniform point sampling of the surface that is independent of the surface parameterization.
- [Gooch 1998] Proposes an NPR color shading model to augment line drawings in technical illustrations.
- [Hamel 1998] Introduces transparency in line drawings by modifying line width, density, or style for occluded surfaces. Uses several static 2D renderings of a scene with transparency enabled and disabled as input to a line drawing algorithm.

- [Koenderink 1998] A comprehensive overview of the structure of relief (i.e. images, height fields, etc...) that covers everything from the historical development of the concepts of ridges and valleys, to differential structure, to linear features such as the occluding contour and cliff curves (closely related to suggestive contours). Note: this was written for physicists, so expect math.

- [Masuch 1998a] Extends the daLi! system for creating animated line-drawings from 3D objects with frame-to-frame coherence.
- [Masuch 1998b] Describes an application of the daLi! line-drawing system in visualizing ancient architecture.

## 1999

- [Barequet 1999] Introduces a method for efficiently detecting silhouette edges in 3D meshes by solving a dual problem of intersecting a plane with line segments.
- [Belhumeur 1999] A theoretical development of the ambiguity present in shaded imagery. The generalized bas-relief transformation also preserves contours and shadow boundaries.
- [Benichou 1999] Details a method that allows for real-time selection and rendering of a complete set of silhouette edges from a 3D mesh after a preprocessing of the mesh.
- [Elber 1999] Introduces an interactive method for rendering of silhouette strokes from freeform models. Generates silhouette-oriented strokes in a preprocessing step and uses an accelerated data structure to selectively render them in real-time.
- [Gooch 1999] Outlines aspects of an interactive technical illustration system for 3D polygonal models. Describes hardware and software methods of drawing silhouettes as well as NPR shading models and shadows.
- [Raskar 1999] Describes a hardware-accelerated method for rendering silhouette edges. It renders back-facing polygons offset towards the camera, such that they show through front-facing polygons.

## 2000

- [Deussen 2000] Details an algorithm for the creation of pen-and-ink drawings of trees. Uses silhouettes and hatching to render trunks and branches. Combines individual leaves into larger groups represented by abstract shapes to render foliage.
- [Ebert 2000] Introduces NPR rendering techniques for volume illustration. Topics covered include boundary and silhouette enhancement, feature halos, and tone shading.
- [Girshick 2000] Argues for the merit of using lines along principal directions for conveying 3D shape. Presents psychological evidence to support the importance of principal direction in perception. Describes the creation of principal direction vector fields on surfaces and the tracing of strokes through the vector field to create line drawings.
- \* [Hertzmann 2000] Describes line drawing rendering techniques for smooth mesh surfaces. Includes algorithms for fast deterministic detection of silhouettes using a dual surface intersection, cusp detection, visibility, and computation of smooth direction fields suitable for hatching strokes.

- [Lake 2000] Describes drawing silhouette edges with special strokes based on underlying curvature, and screen-space stroke texturing for hatching and shading.
- [Northrup 2000] Details a set of rendering methods for lines extracted from 3D geometry. Describes how to smooth self-intersecting silhouette edges in image space and create longer, cleaner silhouette paths using an ID reference image. Suggests rendering line paths as texture-mapped triangle strips to achieve various artistic brush stroke effects.
- [Rossi 2000] Creates a specific style of line-art drawings from 3D models using hatching along principal directions. The object is rendered with surface information (normal, curvature) to image space where user-guided segmentation is performed. Hatching is applied to segmented components along their principal directions.
- [Sander 2000] Contains a section on efficient silhouette detection using a hierarchical cone structure.
- [Treavatt 2000] Introduces 3D and multi-pass 2D methods for pen-and-ink rendering of volumes using procedural textures in a standard volume rendering pipeline. Discusses integration of the NPR steps with traditional rendering steps to generate images that are mixed photorealistic and NPR.

## 2001

- [Csébfalvi 2001] Presents a fast, automatic algorithm for visualization of contours in volumes by rendering areas of high gradient magnitude modulated by a view-dependent term. Details optimizations required to achieve interactive rendering.
- [Gooch 2001] Book describing a variety of non-photorealistic rendering techniques and effects.
- [Knill 2001] Connects the information provided by geodesics on surfaces with homogenous texture “flow”.
- [Koenderink 2001] A fantastic development that connects the bas-relief ambiguity [Belhumeur 1999] in shaded imagery with a series of psychophysical experiments that demonstrated that percepts arising from photographs of simply shaded objects respect this ambiguity. The psychophysical tasks described here could be used to measure percepts of NPR imagery.
- [Page 2001] Extracts ridge and valley lines from 3D objects using curvature estimated with a normal voting approach.
- [Praun 2001] Describes a hardware-accelerated method of rendering 3D polygonal models with light-dependent hatched shading in real-time. Textures are oriented along principal directions such that hatching marks convey surface shape.
- [Raskar 2001] Adapts previous two-pass hardware-assisted rendering techniques to single-pass rendering of silhouettes and creases using modern graphics hardware.
- [Watanabe 2001] Detects ridge and valley triangles on meshes by looking for curvature extrema on the “focal surface.” The extreme regions are thresholded and thinned.

## 2002

- [Isenberg 2002] Describes an algorithm that simplifies self-intersecting silhouette edges from 3D meshes to create longer, cleaner paths suitable for stylized rendering. Differs from previous approaches by using z-buffer techniques and avoiding the use of an ID reference image.

- [Kalnins 2002] Details an interactive NPR rendering system for 3D meshes which allows for artist annotated strokes and brush styles to be consistently rendered across multiple frames and viewpoints. Artist input from one viewpoint is synthesized and propagated for other viewpoints and several models of scale-varying hatching are introduced.
- [Lu 2002] Presents an interactive NPR volume rendering technique using stippling. Includes provisions for enhancement of silhouette and boundary regions and a method of sketching silhouette curves.
- [Lum 2002] Describes a hardware-accelerated parallelized NPR volume rendering method for large volumes. Renders with tone shading, silhouettes and depth cuing at interactive frame-rates on computer clusters.
- [Martin 2002] Introduces a method for creating flat silhouettes from 3D meshes suitable for stylized rendering. Results in an ordered stack of 2D filled polygons which together compose a line drawing of the 3D model.
- [Strothotte 2002] Provides an introduction and overview of NPR rendering methods and areas of application. Explains 2D and 3D algorithms for pen-and-ink, pencil-sketch, painterly-effects, and other NPR effects.
- [Webb 2002] Builds on [Praun 2001] by introducing refined control over hatching strokes, allowing for light areas to be textured with a few strong strokes.

## 2003

- \* [DeCarlo 2003] Describes a class of lines defined on a surface, *suggestive contours*, which complement the set of surface contours in depicting shape. Suggestive contours provide interior detail to line drawings and are defined at points with radial curvature of zero. Both object-space and image-space extraction algorithms are provided.
- [Dong 2003] Introduces new non-interactive silhouette and hatching algorithms for volumetric data sets. Hatching strokes for a surface are generated from the surface as well as from data throughout the volume enclosed by the surface.
- \* [Isenberg 2003] Surveys a wide variety of silhouette extraction algorithms for polygonal models.
- \* [Kalnins 2003] Describes a method for maintaining frame-to-frame temporal coherency for stylized silhouettes by propagating line stroke parameterizations between frames.
- [Kindlmann 2003] Transfer function-based rendering of contour and ridge and valley lines in volume data. Contour thickness is controlled using estimated curvatures, and ridges and valleys are emphasized by thresholding principal curvatures.
- \* [Pauly 2003] Extracts ridge and valley lines from unstructured 3D point clouds. Performs a scale-space analysis to keep the most important lines at multiple scales.
- [Phillips 2003] Demonstrates that people can consistently mark ridges and valleys in shaded images. Also includes discussion on the presence of ridges and valleys in line drawings.
- [Sousa 2003a] Details a fast but non-interactive method for creating precise ink drawings from highly tessellated 3D models. A fraction of mesh edges are selected to become strokes and are drawn with width modulated by surface curvature.

- [Sousa 2003b] Describes a complete system for generating line drawings from 3D meshes. Extracts silhouettes, boundaries, ridges, and valleys; chains lines together; fits curves to paths; and renders paths sparsely with varying line width.
- [Whelan 2003] Introduces a method for augmenting silhouettes for rendering terrain, by including silhouettes computed from viewpoints different from the actual camera position.

## 2004

- [Ashikhmin 2004] Presents a simple image-based approach to rendering silhouettes from 3D meshes without any preprocessing of the mesh surface.
- [Brosz 2004] Introduces an improved method of silhouette selection by considering the stability of silhouettes over potential silhouette edges. Resulting silhouettes can be stylized and drawn with frame-to-frame coherence.
- \* [DeCarlo 2004] Discusses the motion and stability of suggestive contours on surfaces and describes an algorithm for interactive suggestive contour extraction and rendering.
- [Gooch 2004] An evaluation of methods for facial illustrations and caricatures that compares photographs and drawings. It turns out that illustrations are learned faster than photographs in recognition tasks.
- [Grabli 2004a] Describes a Python-based programmable architecture for the creation of NPR line drawings. Includes capabilities for line-type selection, line chaining, and stylization as well as descriptions of the requisite data structures and support algorithms.
- [Grabli 2004b] Introduces a method for measuring density in line drawings and a simplification method based on such density information. Differentiates between density measured in a completed drawing versus density measured during the drawing process. Discusses indication as well as stroke prioritization.
- [Kalnins 2004] Discusses the previously introduced WYSIWYG NPR system in more depth, with sections about stroke representation, visibility, media simulation, temporally-coherent line stylization, and annotated hatching.
- [McGuire 2004] Describes GPU hardware methods for extracting and rendering stylized silhouettes from 3D meshes. Creates an edge mesh with edge information stored at its vertices, allowing for the vertex shader to operate on edge information.
- [Nagy 2004] Describes a GPU hardware method for rendering silhouettes on volumes. Uses texturing hardware and the fragment shader to generate thin image-space lines, which are then broadened and rendered.
- \* [Ohtake 2004] Estimates curvature and curvature derivative of 3D objects using implicit function fits, then extracts ridge and valley lines using these estimates. Curvature-based filtering is used to keep the most significant lines.
- [Raskar 2004] Introduces a camera and multi-flash setup for acquisition of silhouettes of real-world objects.
- [Schein 2004] Presents a method for extracting silhouettes from volume data by modeling the data with trivariate tensor product B-spline functions and extracting silhouettes from the implicit function representation.

- [Wilson 2004] Introduces a method for rendering line drawings from complex 3D geometry in a comprehensible expressive manner. Areas of high complexity are rendered with abstraction by using a hybrid 2D/3D rendering approach.
- [Xu 2004] Presents an algorithm for real-time rendering of silhouettes from unstructured 3D point clouds.
- [Zakaria 2004] Introduces an algorithm for interactive extraction of stylized silhouettes from unstructured 3D point clouds.
- [Zander 2004] Describes an algorithm for high-quality rendering of hatching on 3D meshes by selectively rendering streamlines derived from surface curvature.

## 2005

- [Barla 2005] Introduces a clustering method to reduce screen-space line complexity. Similar lines are clustered together and replaced by a representative line.
- [Belyaev 2005] Extracts ridge and valley lines from noisy range data, including de-noising of normals via nonlinear diffusion and Canny-like nonmaximum suppression.
- [Burns 2005] Introduces an interactive method for probabilistic extraction and rendering of contours and suggestive contours from volume data.
- [Jeong 2005] Controls line density at different scales by computing a progressive-mesh representation of the model, then extracting lines on a mesh with an appropriate view-dependent level of detail.
- [Lum 2005] Uses an example based approach to allow the user to select which lines to render from a large candidate set. The user's examples are used to train a machine learning classifier.

## 2006

- [Coconu 2006] Presents a method for pen-and-ink illustration of scenes including large numbers of trees. Trees and other objects are represented by hierarchical set of spheres or other abstract shapes.
- \* [Cole 2006] Presents an interactive system for placing visual emphasis in stylized renderings. Includes a method for temporally coherent line density control. Analyzes the effect of the emphasis using an eye-tracking experiment.
- [Isenberg 2006] Proposes a framework for storing multiple attributes, such as visibility, with a stroke throughout the rendering pipeline, and applying rendering effects based on these attributes just before the final drawing step.
- [Lee 2006] Describes a technique for rendering in a pencil drawing style. Contours are drawn with error and over-sketching, and hatching textures imitate pencil strokes.
- [Ni 2006] Controls line density at different scales by pre-computing multiple levels of smoothing for the target model. These different smoothed models are then interpolated at run-time, and lines extracted from the interpolated result.

## 2007

- [Breslav 2007] Defines a way to map screen-space patterns onto 3D models so that the patterns move closely with the model, but remain parameterized in screen-space. Related to [Kalnins 2003], but with 2D patterns instead of line textures.

- \* [DeCarlo 2007] Introduces new object-space line definitions called *suggestive highlights* and *principal highlights*, which abstract the bright regions in a headlit image much as suggestive contours abstract the dark regions.
- [Goodwin 2007] Proposes adjusting stroke width based on density of nearby isophotes. As a result, strokes are wider in areas where a shaded image would be dark.
- \* [Judd 2007] Describes a new object-space line definition called *apparent ridges*. Apparent ridges are geometric ridges when the surface normal points towards the camera, but become occluding contours when the surface normal becomes perpendicular to the camera.
- [Kaplan 2007] Presents an algorithm for determining the number of polygonal layers occluding a 3D line. This information can be used for, among other things, drawing occluded lines in varying styles.
- \* [Lee 2007] Presents an image-space approach that creates line drawings by finding edges and ridges in a shaded image. The lines respond to lighting and viewing changes. Toon shading is used to augment the lines in areas of broad color.
- [Neumann 2007] Presents a variety of emphasis effects based on adjusting camera and style parameters inside an area of effect. Examples include locally changing line color and width, and spatially distorting the model.
- [Xie 2007] Introduces a method for extracting lines from a 3D model by examining lighting in object-space. Allows for the user to add and manipulate lights to control the resulting lines.
- [Yoshizawa 2007] Presents a method for finding ridge and valley lines on meshes using differential identities on surfaces and focal surfaces.

## 2008

- [Cole 2008a] Introduces partial visibility for stylized lines, along with improvements of the itembuffer technique [Northrup 2000] to reduce visibility aliasing.
- [Cole 2008b] Presents a study of artists' drawings designed to support analysis of the local surface features under the artists' lines and a comparison with current line drawing algorithms.
- [Shesh 2008] Presents a fast and temporally coherent stroke simplification scheme using an advanced datastructure called a deformable spanner.

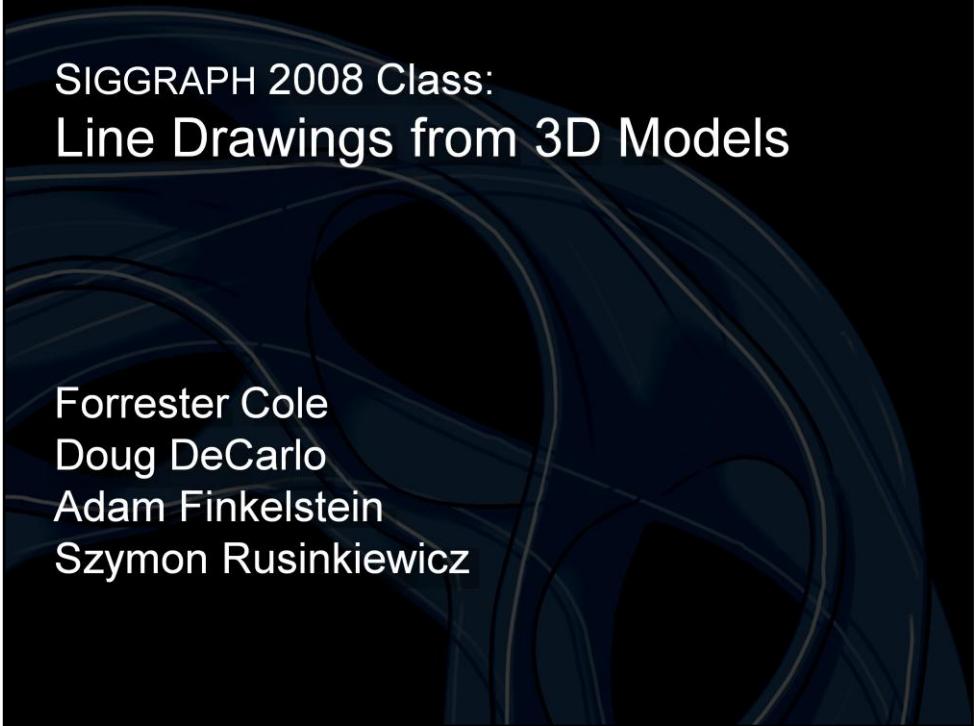
## References

- AGRAWALA, M., AND STOLTE, C. 2001. Rendering Effective Route Maps: Improving Usability Through Generalization. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 241–250.
- APPEL, A. 1967. The notion of quantitative invisibility and the machine rendering of solids. In *Proceedings of the 1967 22nd national conference*, ACM Press, New York, NY, USA, 387–393.
- ASHIKHMIN, M. 2004. Image-space silhouettes for unprocessed models. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 195–202.
- BAREQUET, G., DUNCAN, C. A., GOODRICH, M. T., KUMAR, S., AND POP, M. 1999. Efficient perspective-accurate silhouette computation. In *SCG '99: Proceedings of the fifteenth annual symposium on Computational geometry*, ACM Press, New York, NY, USA, 417–418.
- BARLA, P., THOLLOT, J., AND SILLION, F. X. 2005. Geometric Clustering for Line Drawing Simplification. In *Rendering Techniques 2005: 16th Eurographics Workshop on Rendering*, 183–192.
- BARROW, H., AND TENENBAUM, J. 1981. Interpreting Line Drawings as Three-Dimensional Surfaces. *Artificial Intelligence* 17, 75–116.
- BELHUMEUR, P. N., KRIEGLAN, D. J., AND YUILLE, A. L. 1999. The Bas-Relief Ambiguity. *International Journal of Computer Vision* 35, 1, 33–44.
- BELYAEV, A., PASKO, A., AND KUNII, T. 1998. Ridges and ravines on implicit surfaces. In *Computer Graphics International* 98, 530–535.
- BELYAEV, A., AND ANOSHKINA, E. 2005. Detection of Surface Creases in Range Data. In *Eleventh IMA Conference on The Mathematics of Surfaces*.
- BENICHOU, F., AND ELBER, G. 1999. Output Sensitive Extraction of Silhouettes from Polygonal Geometry. In *PG '99: Proceedings of the 7th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC, USA, 60.
- BREMER, D., AND HUGHES, J. 1998. Rapid approximate silhouette rendering of implicit surfaces. In *Implicit Surfaces 98*, 155–164.
- BRESLAV, S., SZERSZEN, K., MARKOSIAN, L., BARLA, P., AND THOLLOT, J. 2007. Dynamic 2D patterns for shading 3D scenes. *ACM Trans. Graph.* 26, 3, 20.
- BROSZ, J., SAMAVATI, F., AND SOUSA, M. C. 2004. Silhouette rendering based on stability measurement. In *SCCG '04: Proceedings of the 20th spring conference on Computer graphics*, ACM Press, New York, NY, USA, 157–167.
- BURNS, M., KLAWE, J., RUSINKIEWICZ, S., FINKELSTEIN, A., AND DECARLO, D. 2005. Line Drawings from Volume Data. In *SIGGRAPH '05: Proceedings of the 32nd annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA. To appear.
- CANNY, J. 1986. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 8, 6, 679–698.
- COCONU, L., DEUSSEN, O., AND HEGE, H.-C. 2006. Real-time pen-and-ink illustration of landscapes. In *NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, 27–35.
- COLE, F., DECARLO, D., FINKELSTEIN, A., KIN, K., MORLEY, K., AND SANTELLA, A. 2006. Directing Gaze in 3D Models with Stylized Focus. *Eurographics Symposium on Rendering* (June), 377–387.
- COLE, F., AND FINKELSTEIN, A. 2008. Partial Visibility for Stylized Lines. In *NPAR 2008*.
- COLE, F., GOLOVINSKIY, A., LIMPAECHER, A., BARROS, H. S., FINKELSTEIN, A., FUNKHOUSER, T., AND RUSINKIEWICZ, S. 2008. Where Do People Draw Lines? *ACM Transactions on Graphics (Proc. SIGGRAPH)* 27, 3 (Aug.).
- CSÉBFALVI, B., MROZ, L., HAUSER, H., KÖNIG, A., AND GRÖLLER, E. 2001. Fast Visualization of Object Contours by Non-Photorealistic Volume Rendering. In *Eurographics 01*.
- DECARLO, D., AND SANTELLA, A. 2002. Stylization and Abstraction of Photographs. *ACM Transactions on Graphics* 21, 3 (July), 769–776.
- DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive contours for conveying shape. *ACM Trans. Graph.* 22, 3, 848–855.
- DECARLO, D., FINKELSTEIN, A., AND RUSINKIEWICZ, S. 2004. Interactive rendering of suggestive contours with temporal coherence. In *NPAR '04: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, ACM Press, New York, NY, USA, 15–145.

- DECARLO, D., AND RUSINKIEWICZ, S. 2007. Highlight lines for conveying shape. In *NPAR '07: Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, 63–70.
- DEUSSEN, O., AND STROTHOTTE, T. 2000. Computer-generated pen-and-ink illustration of trees. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 13–18.
- DONG, F., CLAPWORTHY, G. J., LIN, H., AND KROKOS, M. A. 2003. Nonphotorealistic Rendering of Medical Volume Data. *IEEE Comput. Graph. Appl.* 23, 4, 44–52.
- DOOLEY, D., AND COHEN, M. F. 1990. Automatic illustration of 3D geometric models: lines. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics*, ACM Press, New York, NY, USA, 77–82.
- DURAND, F., OSTROMOUKHOV, V., MILLER, M., DURANLEAU, F., AND DORSEY, J. 2001. Decoupling Strokes and High-Level Attributes for Interactive Traditional Drawing. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, 71–82.
- EBERT, D., AND RHEINGANS, P. 2000. Volume illustration: non-photorealistic rendering of volume models. In *VIS '00: Proceedings of the conference on Visualization '00*, IEEE Computer Society Press, Los Alamitos, CA, USA, 195–202.
- ELBER, G., AND COHEN, E. 1990. Hidden curve removal for free form surfaces. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 95–104.
- ELBER, G. 1995. Line Art Rendering via a Coverage of Isoparametric Curves. *IEEE Transactions on Visualization and Computer Graphics* 1, 3, 231–239.
- ELBER, G. 1995. Line illustrations in computer graphics. *The Visual Computer* 11, 6, 290–296.
- ELBER, G. 1998. Line Art Illustrations of Parametric and Implicit Forms. *IEEE Transactions on Visualization and Computer Graphics* 4, 1, 71–81.
- ELBER, G. 1999. Interactive Line Art Rendering of Freeform Surfaces. *Computer Graphics Forum* 18, 3, 1–1.
- GIRSHICK, A., INTERRANTE, V., HAKER, S., AND LEMOINE, T. 2000. Line direction matters: an argument for the use of principal directions in 3D line drawings. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, ACM Press, New York, NY, USA, 43–52.
- GOOCH, A., GOOCH, B., SHIRLEY, P., AND COHEN, E. 1998. A non-photorealistic lighting model for automatic technical illustration. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 447–452.
- GOOCH, B., SLOAN, P.-P. J., GOOCH, A., SHIRLEY, P., AND RIESENFELD, R. 1999. Interactive technical illustration. In *SI3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, ACM Press, New York, NY, USA, 31–38.
- GOOCH, B., AND GOOCH, A. 2001. *Non-Photorealistic Rendering*. A K Peters.
- GOOCH, B., REINHARD, E., AND GOOCH, A. 2004. Human facial illustrations: Creation and psychophysical evaluation. *ACM Trans. Graph.* 23, 1, 27–44.
- GOODWIN, T., VOLLICK, I., AND HERTZMANN, A. 2007. Isophote distance: a shading approach to artistic stroke thickness. In *NPAR '07: Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, 53–62.
- GRABLI, S., TURQUIN, E., DURAND, F., AND SILLION, F. 2004. Programmable Style for NPR Line Drawing. In *Eurographics Symposium on Rendering '04*.
- GRABLI, S., DURAND, F., AND SILLION, F. 2004. Density Measure for Line-Drawing Simplification. In *Proceedings of Pacific Graphics*.
- HAMEL, J., SCHLECHTWEG, S., AND STROTHOTTE, T. 1998. An Approach to Visualizing Transparency in Computer-Generated Line Drawings. In *IV '98: Proceedings of the International Conference on Information Visualisation*, IEEE Computer Society, Washington, DC, USA, 151.
- HERTZMANN, A., AND ZORIN, D. 2000. Illustrating smooth surfaces. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 517–526.
- HERTZMANN, A. 2001. Paint by relaxation. In *Computer Graphics International 2001*, 47–54.
- HSU, S. C., AND LEE, I. H. H. 1994. Drawing and animation using skeletal strokes. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 109–118.
- INTERRANTE, V., FUCHS, H., AND PIZER, S. 1995. Enhancing Transparent Skin Surfaces with Ridge and Valley Lines. In *VIS '95: Proceedings of the 6th conference on Visualization '95*, IEEE Computer Society, Washington, DC, USA, 52.
- INTERRANTE, V., FUCHS, H., AND PIZER, S. 1996. Illustrating transparent surfaces with curvature-directed strokes. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, IEEE Computer Society Press, Los Alamitos, CA, USA, 211–ff.
- ISENBERG, T., HALPER, N., AND STROTHOTTE, T. 2002. Stylizing Silhouettes at Interactive Rates: From Silhouette Edges to Silhouette Strokes. *Computer Graphics Forum* 21, 3, 249–249.
- ISENBERG, T., FREUDENBERG, B., HALPER, N., SCHLECHTWEG, S., AND STROTHOTTE, T. 2003. A Developer's Guide to Silhouette Algorithms for Polygonal Models. *IEEE Comput. Graph. Appl.* 23, 4, 28–37.
- ISENBERG, T., AND BRENNCKE, A. 2006. G-strokes: A concept for simplifying line stylization. In *Computers & Graphics*, vol. 30, 754–766.
- ITTI, L., AND KOCH, C. 2000. A Saliency-based search mechanism for overt and covert shifts of visual attention. *Vision Research* 40, 1489–1506.
- JEONG, K., NI, A., LEE, S., AND MARKOSIAN, L. 2005. Detail control in line drawings of 3D meshes. In *The Visual Computer*, vol. 21, 698–706.
- JUDD, T., DURAND, F., AND ADELSON, E. 2007. Apparent ridges for line drawing. *ACM Trans. Graph.* 26, 3, 19.
- KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. WYSIWYG NPR: drawing strokes directly on 3D models. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 755–762.
- KALNINS, R. D., DAVIDSON, P. L., MARKOSIAN, L., AND FINKELSTEIN, A. 2003. Coherent stylized silhouettes. *ACM Trans. Graph.* 22, 3, 856–861.
- KALNINS, R. D. 2004. *WYSIWYG NPR: Interactive Stylization for Stroke-Based Rendering of 3D Animation*. PhD thesis, Princeton University.
- KAPLAN, M. 2007. Hybrid quantitative invisibility. In *NPAR '07: Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, 51–52.
- KINDEMANN, G., WHITAKER, R., TASDIZEN, T., AND MOLLER, T. 2003. Curvature-based transfer functions for direct volume rendering: methods and applications. In *IEEE Visualization 2003*, 513–520.
- KNILL, D. C. 1992. Perception of surface contours and surface shape: from computation to psychophysics. *Journal of the Optical Society of America A* 9, 9, 1449–1464.
- KNILL, D. C. 2001. Contour into texture: information content of surface contours and texture flow. *Journal of the Optical Society of America A* 18, 1, 12–35.

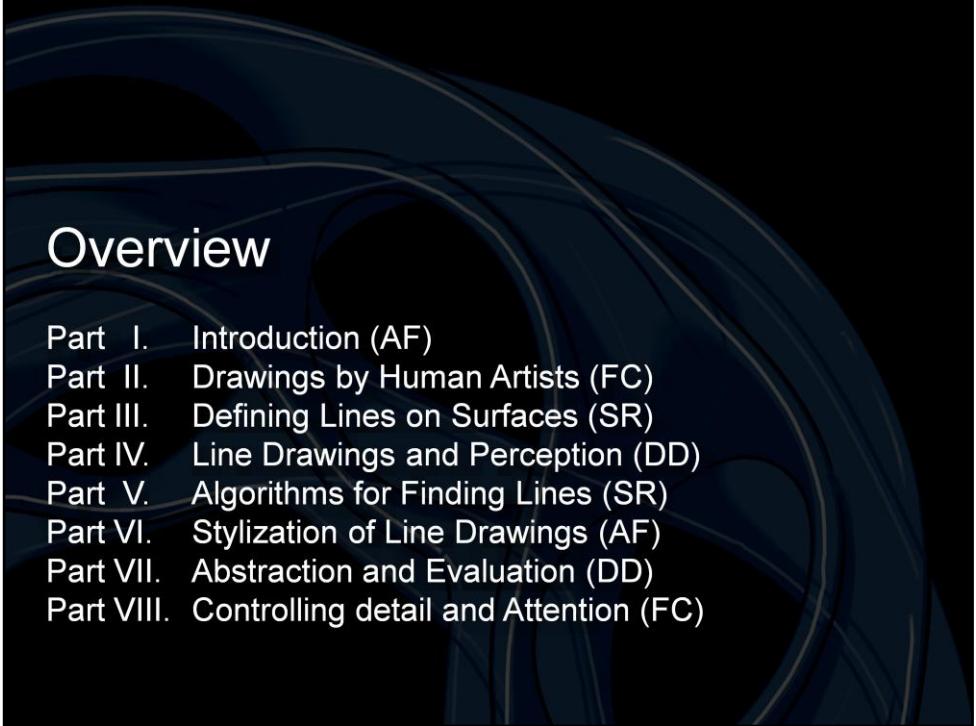
- KOENDERINK, J. J., AND VAN DOORN, A. J. 1982. The Shape of Smooth Objects and the Way Contours End. *Perception* 11, 129–137.
- KOENDERINK, J. J. 1984. What does the occluding contour tell us about solid shape? *Perception* 13, 321–330.
- KOENDERINK, J. J. 1990. *Solid Shape*. MIT press.
- KOENDERINK, J. J., VAN DOORN, A., CHRISTOU, C., AND LAPPIN, J. 1996. Shape constancy in pictorial relief. *Perception* 25, 155–164.
- KOENDERINK, J. J., AND VAN DOORN, A. 1998. The Structure of Relief. *Advances in Imaging and Electron Physics* 103, 65–150.
- KOENDERINK, J. J., VAN DOORN, A. J., KAPPERS, A. M., AND TODD, J. T. 2001. Ambiguity and the ‘mental eye’ in pictorial relief. *Perception* 30, 431–448.
- LAKE, A., MARSHALL, C., HARRIS, M., AND BLACKSTEIN, M. 2000. Stylized rendering techniques for scalable real-time 3D animation. In *NPAR ’00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, 13–20.
- LEE, H., KWON, S., AND LEE, S. 2006. Real-time pencil rendering. In *NPAR ’06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, 37–45.
- LEE, Y., MARKOSIAN, L., LEE, S., AND HUGHES, J. F. 2007. Line drawings via abstracted shading. *ACM Trans. Graph.* 26, 3, 18.
- LU, A., MORRIS, C. J., EBERT, D. S., RHEINGANS, P., AND HANSEN, C. 2002. Non-photorealistic volume rendering using stippling techniques. In *VIS ’02: Proceedings of the conference on Visualization ’02*, IEEE Computer Society, Washington, DC, USA, 211–218.
- LUM, E. B., AND MA, K.-L. 2002. Hardware-accelerated parallel non-photorealistic volume rendering. In *NPAR ’02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, ACM Press, New York, NY, USA, 67–ff.
- LUM, E. B., AND MA, K.-L. 2005. Expressive line selection by example. *The Visual Computer* 21, 8-10, 811–820.
- MA, K.-L., AND INTERRANTE, V. 1997. Extracting feature lines from 3D unstructured grids. In *VIS ’97: Proceedings of the 8th conference on Visualization ’97*, IEEE Computer Society Press, Los Alamitos, CA, USA, 285–ff.
- MALIK, J. 1987. Interpreting Line Drawings of Curved Objects. *International Journal of Computer Vision* 1, 1, 73–103.
- MARKOSIAN, L., KOWALSKI, M. A., GOLDSTEIN, D., TRYCHIN, S. J., HUGHES, J. F., AND BOURDEV, L. D. 1997. Real-time nonphotorealistic rendering. In *SIGGRAPH ’97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 415–420.
- MARTIN, D., FEKETE, J., AND TORRES, J. C. 2002. Flattening 3D objects using silhouettes. *Computer Graphics Forum* 21, 3, 239–239.
- MASUCH, M., SCHLECHTWEG, S., AND SCHNWLDER, B. 1997. dali! – Drawing Animated Lines! In *Simulation and Animation* 97, 87–96.
- MASUCH, M., SCHUHMANN, L., AND SCHLECHTWEG, S. 1998. Animating Frame-To-Frame-Coherent Line Drawings for Illustrated Purposes. In *Simulation and Animation* 98, 101–112.
- MASUCH, M., AND STROTHOTTE, T. 1998. Visualising Ancient Architecture using Animating Line Drawings. In *IV ’98: Proceedings of the International Conference on Information Visualisation*, IEEE Computer Society, Washington, DC, USA, 261.
- MC GUIRE, M., AND HUGHES, J. F. 2004. Hardware-determined feature edges. In *NPAR ’04: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, ACM Press, New York, NY, USA, 35–147.
- NAGY, Z., AND KLEIN, R. 2004. High-Quality Silhouette Illustration for Texture-Based Volume Rendering. In *WSCG ’04*.
- NEUMANN, P., ISENBERG, T., AND CARPENDALE, M. S. T. 2007. NPR Lenses: Interactive Tools for Non-Photorealistic Line Drawings. In *Smart Graphics*, 10–22.
- NI, A., JEONG, K., LEE, S., LEE, S., AND MARKOSIAN, L. 2006. Multi-scale line drawings from 3D meshes. In *I3D ’06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, 133–137.
- NORTHRUP, J. D., AND MARKOSIAN, L. 2000. Artistic silhouettes: a hybrid approach. In *NPAR ’00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, ACM Press, New York, NY, USA, 31–37.
- OHTAKE, Y., BELYAEV, A., AND SEIDEL, H.-P. 2004. Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph.* 23, 3.
- PAGE, D. L., KOSCHAN, A., SUN, Y., PAIK, J., AND ABIDI, A. 2001. Robust Crease Detection and Curvature Estimation of Piecewise Smooth Surfaces from Triangle Mesh Approximations Using Normal Voting. In *Proc. Conference on Computer Vision and Pattern Recognition*.
- PAULY, M., KEISER, R., AND GROSS, M. 2003. Multi-Scale Feature Extraction on Point-Sampled Models. In *Proc. Eurographics*.
- PEARSON, D., AND ROBINSON, J. 1985. Visual Communication at Very Low Data Rates. *Proc. IEEE* 4 (Apr.), 795–812.
- PHILLIPS, F., TODD, J. T., KOENDERINK, J. J., AND KAPPERS, A. M. 2003. Perceptual representation of visible surfaces. *Perception and Psychophysics* 65, 5, 747–762.
- PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. In *SIGGRAPH ’01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 581.
- RASKAR, R., AND COHEN, M. 1999. Image precision silhouette edges. In *SI3D ’99: Proceedings of the 1999 symposium on Interactive 3D graphics*, ACM Press, New York, NY, USA, 135–140.
- RASKAR, R. 2001. Hardware Support for Non-photorealistic Rendering. In *Proc. Graphics Hardware*.
- RASKAR, R., HAN TAN, K., FERIS, R., YU, J., AND TURK, M. 2004. Non-photorealistic Camera: Depth Edge Detection and Stylized Rendering using Multi-Flash Imaging. *ACM Trans. Graph.* 23, 3.
- RIEGER, J. H. 1997. Topographical Properties of Generic Images. *International Journal of Computer Vision* 23, 1, 79–92.
- ROSSIGNAC, J., AND VAN EMMERIK, M. 1992. Hidden Contours on a Framebuffer. *Proc. of 7th Workshop on Computer Graphics Hardware*.
- ROSSI, C., AND KOBELT, L. 2000. Line-art rendering of 3D-models. In *Pacific Graphics 00*, 87–96.
- SAITO, T., AND TAKAHASHI, T. 1990. Comprehensible rendering of 3-D shapes. In *SIGGRAPH ’90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 197–206.
- SAITO, T. 1994. Real-time previewing for volume visualization. In *VVS ’94: Proceedings of the 1994 symposium on Volume visualization*, ACM Press, New York, NY, USA, 99–106.
- SALISBURY, M. P., ANDERSON, S. E., BARZEL, R., AND SALESIN, D. H. 1994. Interactive pen-and-ink illustration. In *SIGGRAPH ’94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 101–108.
- SALISBURY, M., ANDERSON, C., LISCHINSKI, D., AND SALESIN, D. H. 1996. Scale-dependent reproduction of pen-and-ink illustrations. In *SIGGRAPH ’96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 461–468.
- SANDER, P. V., GU, X., GORTLER, S. J., HOPPE, H., AND SNYDER, J. 2000. Silhouette clipping. In *SIGGRAPH ’00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 327–334.
- SANTELLA, A., AND DECARLO, D. 2004. Visual interest and NPR: an evaluation and manifesto. In *NPAR 2004*, 71–78.

- SCHEIN, S., AND ELBER, G. 2004. Adaptive extraction and visualization of silhouette curves from volumetric datasets. *Vis. Comput.* 20, 4, 243–252.
- SHESH, A., AND CHEN, B. 2008. Efficient and Dynamic Simplification of Line Drawings. *Computer Graphics Forum (Proc. Eurographics 2008)* 27, 2, 537–545.
- SOUZA, M. C., FOSTER, K., WYVILL, B., AND SAMAVATI, F. 2003. Precise Ink Drawing of 3D Models. *Computer Graphics Forum* 22, 3, 369–369.
- SOUZA, M. C., AND PRUSINKIEWICZ, P. 2003. A Few Good Lines: Suggestive Drawing of 3D Models. *Computer Graphics Forum* 22, 3, 381–381.
- STEVENS, K. A. 1981. The Visual Interpretation of Surface Contours. *Artificial Intelligence* 17, 47–73.
- STROTHOTTE, T., AND SCHLECHTWEG, S. 2002. *Non-Photorealistic Computer Graphics*. Morgan Kaufmann.
- THIRION, J.-P., AND GOURDON, A. 1996. The 3D Marching Lines Algorithm. *Graphical Models and Image Processing* 58, 6 (Nov.), 503–509.
- TREAVETT, S. M. F., AND CHEN, M. 2000. Pen-and-Ink rendering in volume visualisation. In *VIS '00: Proceedings of the conference on Visualization '00*, IEEE Computer Society Press, Los Alamitos, CA, USA, 203–210.
- WALTZ, D. L. 1975. Understanding Line Drawings of Scenes with Shadows. In *The Psychology of Computer Vision*, P. Winston, Ed. McGraw-Hill, 19–92.
- WATANABE, K., AND BELYAEV, A. G. 2001. Detection of Salient Curvature Features on Polygonal Surfaces. *Computer Graphics Forum (Proc. Eurographics 2001)* 20, 3.
- WEBB, M., PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2002. Fine tone control in hardware hatching. In *NPAR '02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, 53–ff.
- WHELAN, J., AND VISVALINGAM, M. 2003. Formulated silhouettes for sketching terrain. *Theory and Practice of Computer Graphics, 2003. Proceedings*, 90–96.
- WILSON, B., AND MA, K.-L. 2004. Rendering complexity in computer-generated pen-and-ink illustrations. In *NPAR '04: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, ACM Press, New York, NY, USA, 129–137.
- WINKENBACH, G., AND SALESIN, D. H. 1994. Computer-generated pen-and-ink illustration. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 91–100.
- WINKENBACH, G., AND SALESIN, D. H. 1996. Rendering parametric surfaces in pen and ink. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 469–476.
- XIE, X., HE, Y., TIAN, F., SEAH, H.-S., GU, X., AND QIN, H. 2007. An Effective Illustrative Visualization Framework Based on Photic Extremum Lines (PELs). *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (Nov./Dec.), 1328–1335.
- XU, H., NGUEN, M., YUAN, X., AND CHEN, B. 2004. Interactive Silhouette Rendering for Point-Based Models. In *Proc. Eurographics Symposium on Point-Based Graphics*.
- YOSHIZAWA, S., BELYAEV, A., YOKOTA, H., AND SEIDEL, H.-P. 2007. Fast and Faithful Geometric Algorithm for Detecting Crest Lines on Meshes. In *Pacific Graphics*.
- ZAKARIA, N., AND SEIDEL, H.-P. 2004. Interactive stylized silhouette for point-sampled geometry. In *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and Southe East Asia*, ACM Press, New York, NY, USA, 242–249.
- ZANDER, J., ISENBERG, T., SCHLECHTWEG, S., AND STROTHOTTE, T. 2004. High Quality Hatching. *Computer Graphics Forum* 23, 3, 421–421.



# SIGGRAPH 2008 Class: Line Drawings from 3D Models

Forrester Cole  
Doug DeCarlo  
Adam Finkelstein  
Szymon Rusinkiewicz



# Overview

- Part I. Introduction (AF)
- Part II. Drawings by Human Artists (FC)
- Part III. Defining Lines on Surfaces (SR)
- Part IV. Line Drawings and Perception (DD)
- Part V. Algorithms for Finding Lines (SR)
- Part VI. Stylization of Line Drawings (AF)
- Part VII. Abstraction and Evaluation (DD)
- Part VIII. Controlling detail and Attention (FC)

## What's new?!?

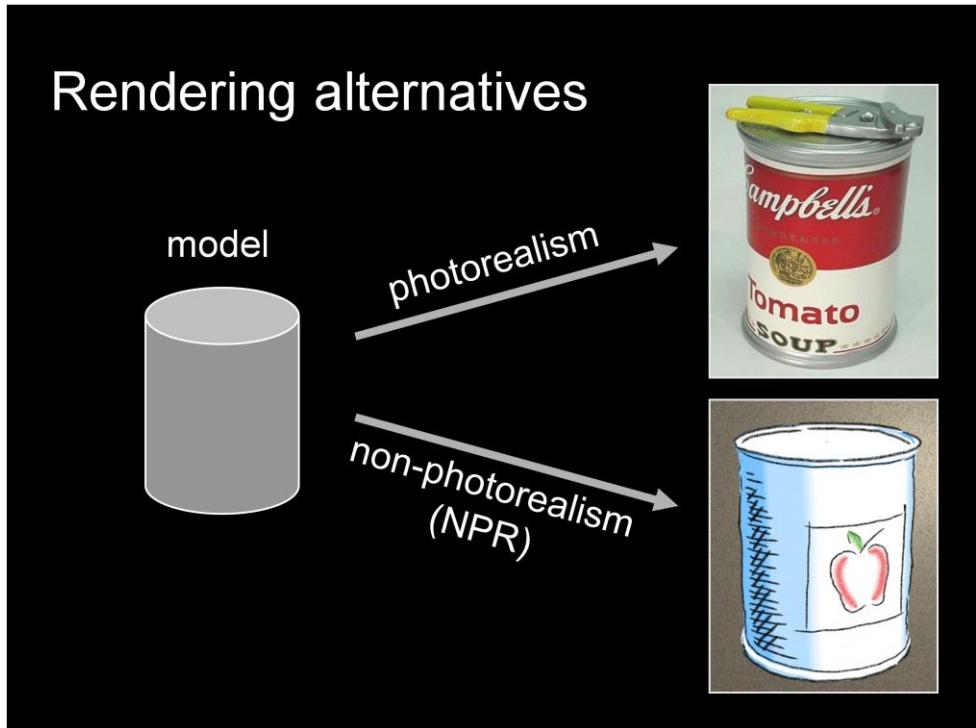
Similar course at SIGGRAPH 2005

Developments since then:

- New algorithms
- New studies
- Version 2.0

We taught a similar course at Siggraph 2005, so it's natural for those of you who attended it to wonder what's new this time? Indeed, there is substantial overlap in this offering. However there have also been a number of advances in the field since then and we'll try to highlight this new material. In particular, there have been several new algorithms proposed recently for drawing lines from smooth 3D shapes. In addition, there have been some studies on how people make line drawings. Finally, there is the usual version 2.0 overhaul.

# Rendering alternatives



When it comes to making pictures with computer graphics there are a couple alternatives. Most traditional work in computer graphics has been striving towards realistic imagery. One reason is that it provides a way to accurately simulate our experiences in the real world. There is a terrible cost to this, unfortunately, which is that you have to get all the details just right, which imposes a great burden on the modeler.

Fortunately there is an alternative. There is a branch of computer graphics that has been looking for more than a dozen years at how we can exploit principles of abstraction known to artists for many centuries to avoid having to worry about a myriad of perhaps unnecessary details.

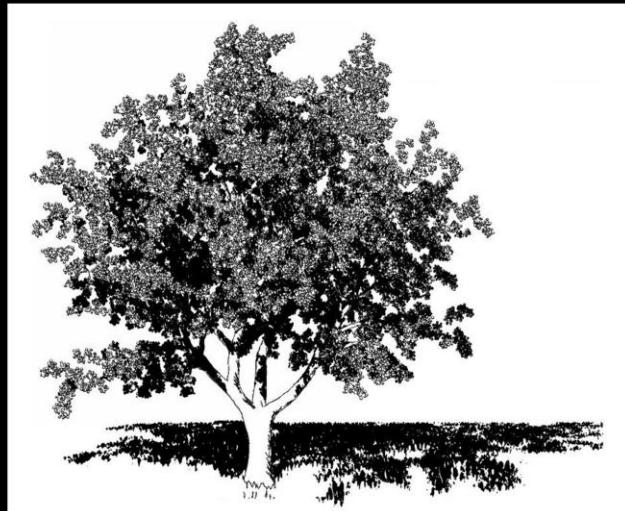
## Realistic modeling and rendering



[Deussen 99]

Here is a rendering of a realistic outdoor scene due to Deussen and others. It highlights the tradeoff that I was just mentioning. The model is extremely complex -- the number of polygons would not fit in the memory of most computers today, let alone those of 1999 when it was created.

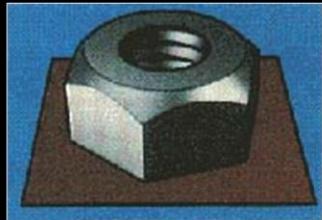
## Non-photorealistic rendering



[Deussen 2000]

Here is a different rendering of an outdoor scene by Deussen and others. Obviously, there is a lot less complexity there, and yet depending on the application it may be just as appropriate for conveying a particular sense of a place.

## Simulating various media (offline)



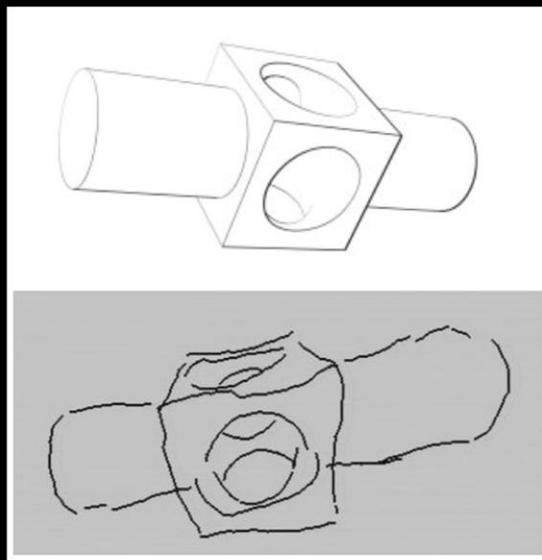
Technical Illustration [Saito 90]



Pen & Ink [Winkenbach 94]

In the 1990s, NPR emerged as an alternative to photorealism. Research efforts largely focused on ways to simulate various media such as technical illustration or pen-and-ink. The goal was to make an image that was believably created by hand in one of these traditional media. These systems were mostly-automatic: the program attempted to mimic what an artist/illustrator might do. As in the traditional media, line drawing played a key role. These methods were largely offline, meaning you wait and after a while you get an image.

## Real-time NPR



[Markosian97]

In the late 90's Markosian showed that stylized line drawings could be generated in real time, responding to changes in camera.

# Temporal coherence

Lack of coherence draws attention



"The New Chair"  
[Curtis 98]

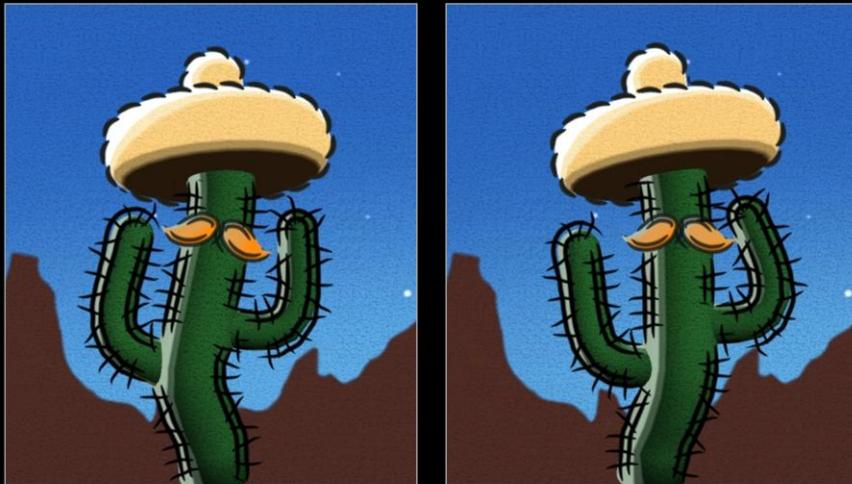
We can also use non-photorealism to guide the viewer's eye towards important features in a scene and away from unimportant fixtures. So, for example here, by using lines that have a crazy temporal quality, Curtis gave the standing figure a very energetic quality that draws your eye.

## Artistic silhouettes



Northrup and Markosian continued to work on temporal coherence and line stylization, producing imagery like this in real time by the beginning of this decade.

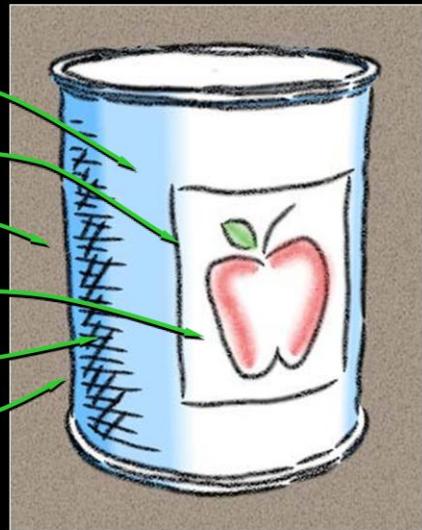
## Coherent stylized silhouettes



[Kalnins03]

## Elements of stylized rendering

Toon shading  
Stylized strokes  
Paper Effect  
Detail Marks  
Hatching  
Outlines



A single illustration can incorporate many different technical elements that combine to form a cohesive image. So there are many different tools that have emerged from the NPR community to be able to make such illustrations with computers. For example, we have cartoon shading, stylized strokes, effects for interaction of media with paper, the ability to draw specific marks onto surfaces, hatching, and automatic outlining tools.

# Line drawings

Elements of line outlines...



In this course we focus on line drawings, in part because they tend to be the simplest form of rendering from 3D models and are often used as a component of more sophisticated rendering schemes. And within line drawings, most of our emphasis will be on outlines such as those shown here. There are three kinds of lines that contribute to the outlines.

# Line drawings

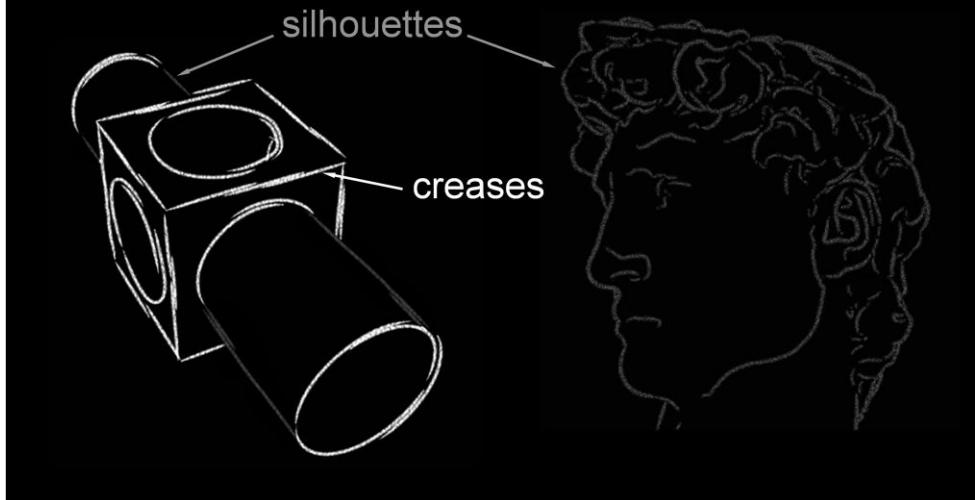
Elements of outlines...



First are the silhouettes (equivalently called “contours” by some researchers) They separate front-facing from back-facing regions of the surface, as a function of view point...

# Line drawings

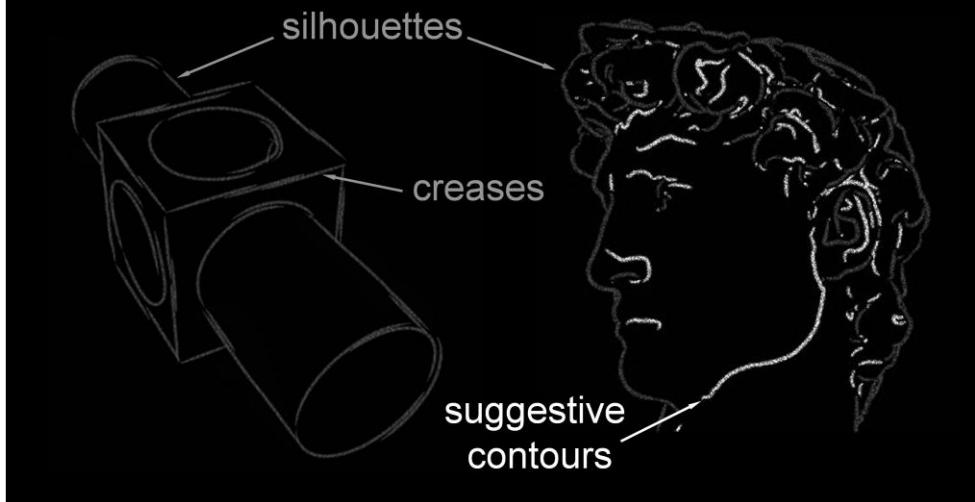
Elements of outlines...



Creases are path which are defined statically on the mesh surface generally representing sharp features, such as the those present on this mechanic part on the left. Other features behave similarly, such as marks drawn directly onto the surface denoting for example the boundary between two different surface textures.

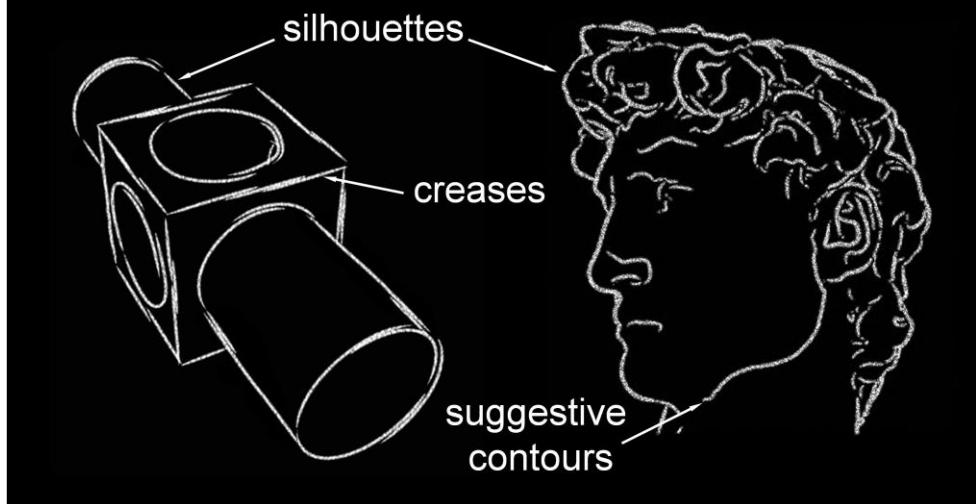
# Line drawings

Elements of outlines...



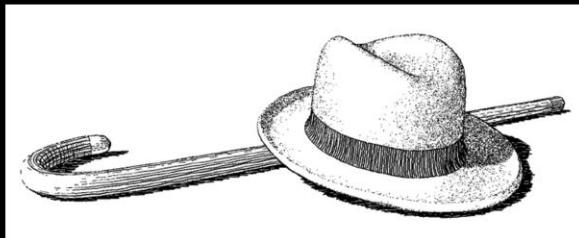
Third, in 2003 we introduced “suggestive contours” which are dynamic features similar to silhouettes that are view dependent and help denote the shape. It turns out that suggestive contours may be thought of as places on the shape that would be silhouettes from nearby views. Since then there have been a number of related line definitions proposed, such as suggestive highlights or apparent ridges, and we will discuss these shortly.

## Line drawings

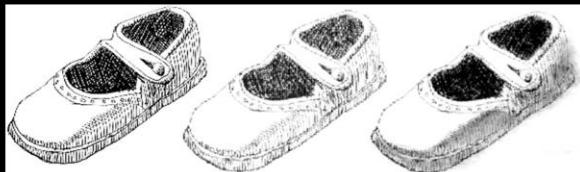


These three elements, though not exhaustive, can produce a wide range of line drawings, and often contribute to more complex illustrations.

## Hatching



[Winkenbach 94, 96]



[Sousa 99]

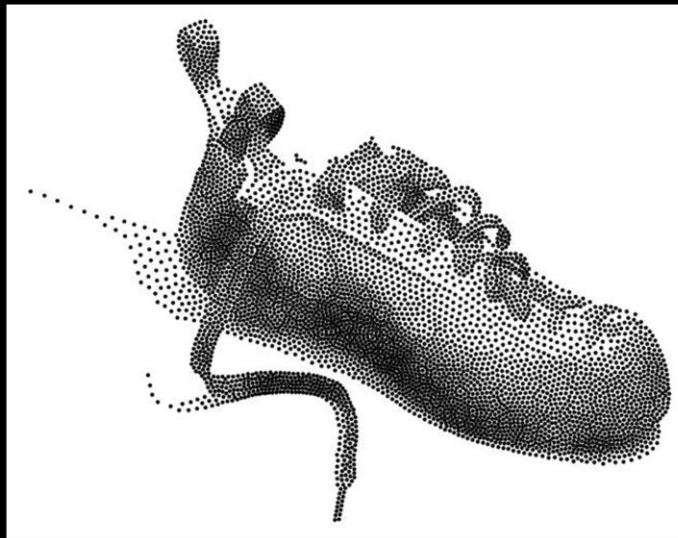


[Hertzmann 2000]

For the purposes of this course we will be ignoring more sophisticated line drawing methods used by artists (and that have been explored by the NPR community). For example, hatching lines can suggest material and tone and are often used in conjunction with basic line drawings.

## Stippling

[Secord02]



I should also mention stippling, which can use a collection of small dots to convey tone. This is an important drawing technique and may be used in conjunction with lines. In contrast with hatching, these marks do not follow the curvatures of the shape and generally do not have the expressiveness to suggest surface texture.

## Basic line drawings



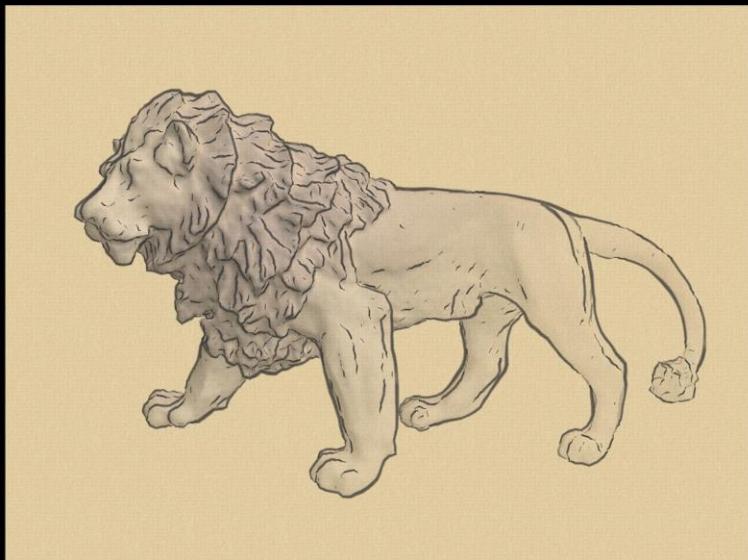
This course will dwell largely on these basic line drawing elements, because they give us plenty to talk about.

## Part II: Drawings by Human Artists

Forrester Cole

Line Drawings from 3D Models  
SIGGRAPH 2008

# Motivation



Computer-Generated Line Drawing

This is an example of a fairly state of the art computer generated line drawing. It uses suggestive contours, slightly stylized strokes, and some visual emphasis effects, and it's a pretty nice, effective drawing.

# Motivation



Artist's Rendering (Rembrandt)

However, it's pretty obvious that the work of human artists, at least the best of them, is still in a different class. Somehow this drawing manages to say a lot more than the CG version, even though the amount of shading and lines on the page is roughly equivalent. Of course, this comparison is unfair – the cg model contains much less information than a view of a real lion.

## A Sobering Comparison



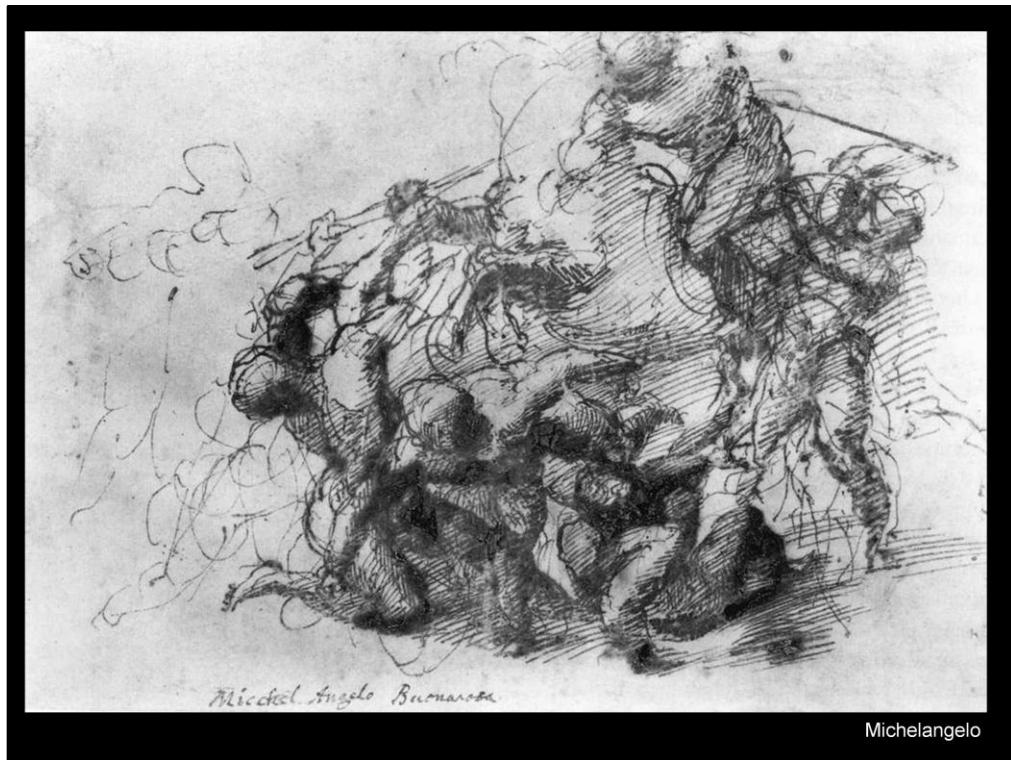
- What do artists have that CG lacks?
- Can we formalize it?

Additionally, there is no shame in not being able to draw as well as Rembrandt. There are maybe a few people in history who could. Still, we'd like to learn what the best human artists are doing differently from our current algorithms, and hopefully simulate it. And to simulate it, we need to formalize it.

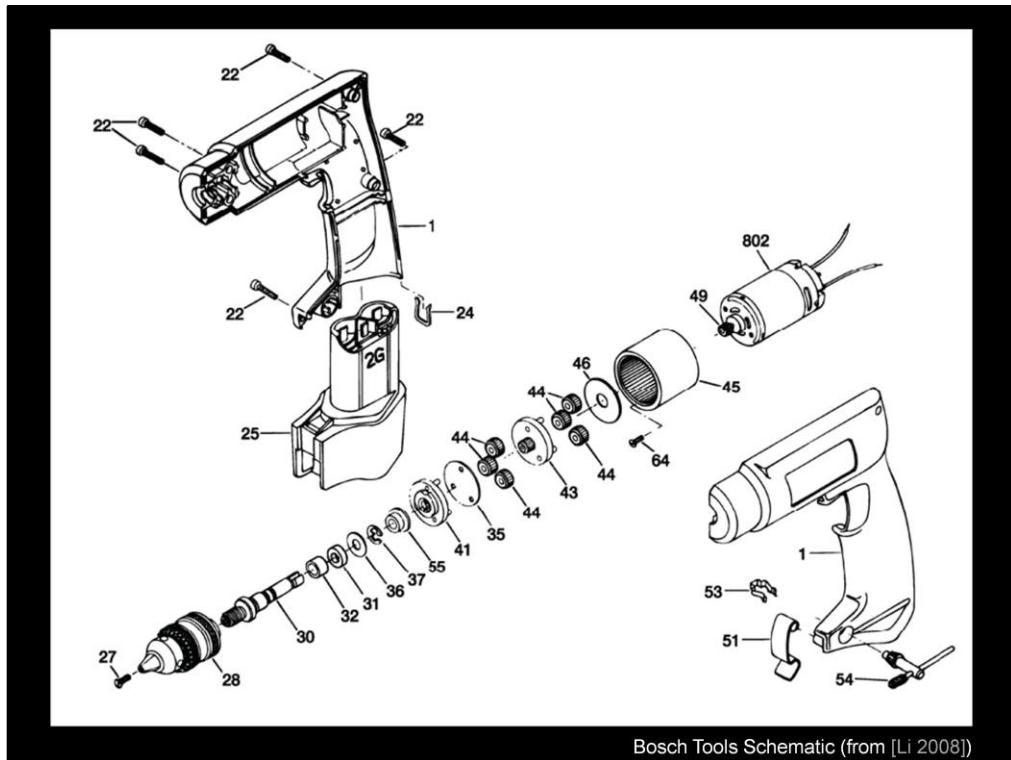
# The Problem

- "Line drawing" is not a single style
- Immense range of approaches
  - Emotional to explanatory
  - Loose to controlled
  - Sparse to developed
  - Veridical to abstract

This course is concerned with line drawings. "Line drawing" is often used as a shorthand for a particular kind of simple, sparse, monochrome drawing, but from the point of view of a computer algorithm, human line drawings include a huge set of variations that need to be examined.



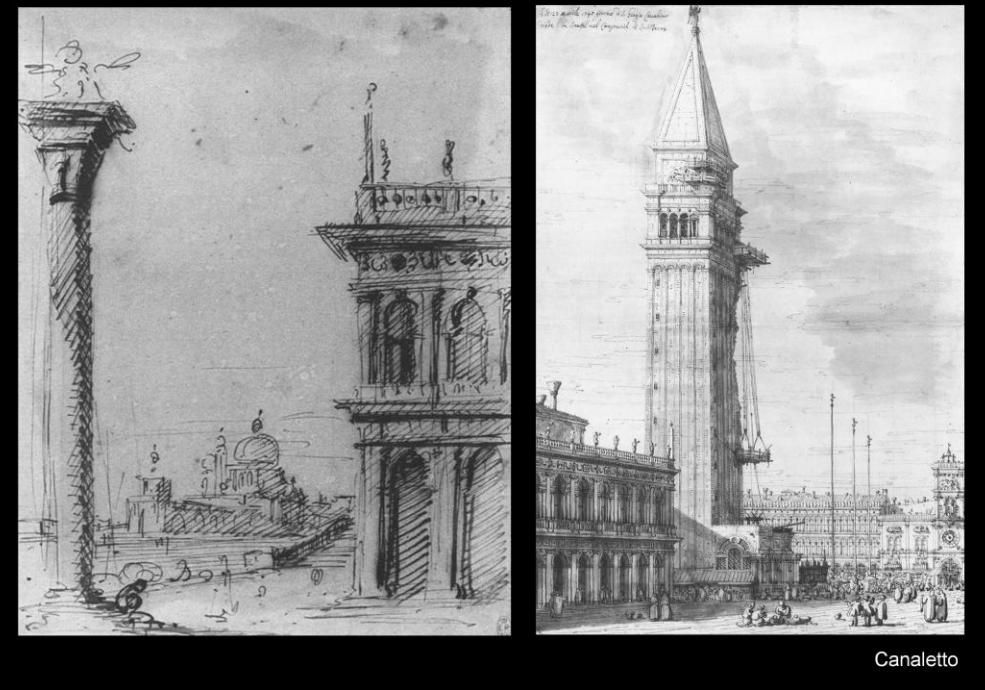
Here is a drawing that would likely be quite difficult to recreate using current computer algorithms. It's a sketch for a later painting, but Michelangelo signed it, so he must have thought that it was reasonably complete. It's hard even for a human to see what exactly the drawing is depicting – something about men struggling, maybe in battle. The point of the drawing is in the way it shows the motion, the emotional content.



Bosch Tools Schematic (from [Li 2008])

By contrast, here is a drawing that is entirely explanatory. It exists to show the parts exactly, and part of the purpose of the drawing is that the viewer can see completely unambiguously which parts are depicted. The contrast between the two drawings is almost painful; you can feel your brain changing gears when you compare this drawing and the last.

## Loose vs. Controlled



Beyond the basic purpose of a drawing, a major distinction is in how tightly controlled and precise the drawing is.

This isn't just the difference between drawing quickly and taking your time, though the drawing on the left must have been much quicker than the one on the right. A precise, controlled style is best for unambiguously showing detail (you can count the number of people in the courtyard in the drawing on the right). However, a loose style also conveys information; it says that there is more there than you are showing right now, the artistic equivalent of waving hands and saying "et cetera."

## Sparse vs. Developed



John Singer Sargent



Piranesi

All line drawings ask the viewer to use their imagination, and a skilled artist uses this as a great point of strength. The artist can choose, however, how far to guide the viewer's imagination. The artist can leave almost everything unsaid, as in the left drawing (where only the basic outline of a classical interior can be made out), or detail almost every inch of the paper, as in the right (where you can look for a long time and still not catch all the tiny human figures).

## Veridical vs. Abstract



Gustave Moreau



Picasso

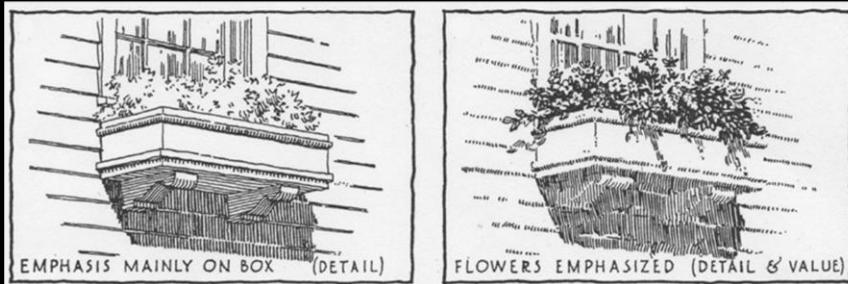
This is a different type of abstraction from using a loose or sparse style – it is a careful depiction of an abstraction of the subject. An extreme example would be a cartoon character. Computer algorithms have so far dealt with this type of abstraction only in very specialized cases.

## The Good News...

- Some techniques fairly well understood
  - Simple abstraction
  - Hatching
  - Shading
  - more...
- Described by art instruction books
- Roughly correspond to algorithms

There are a number of techniques used by human artists that are fairly well understood, such as simple abstraction, hatching, and shading. These techniques are described in detail by books on art instruction, and these descriptions can be translated more or less directly into algorithms.

# Abstraction



Arthur Leighton Guptill

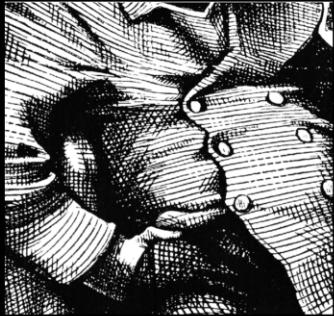


[Deussen 2000]

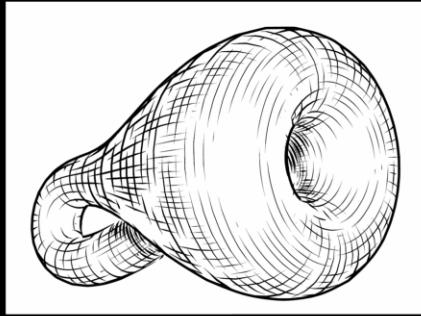
For example, this paper by Deussen et al. simulates quite closely the changes in style laid out by Guptill. The plants are made abstract by rounding the shapes, making the lines more sparse, and making the overall tone lighter. The plants are made more detailed by doing the opposite: adding detail leaf shapes and increasing the overall density of lines.

# Hatching

- Four conventional tone levels [Hertzmann 2000]
  - Highlight, hatching, cross-hatching, undercut
- Fairly constant tone in each hatching area
- Hatching lines mostly straight



Thomas Nast

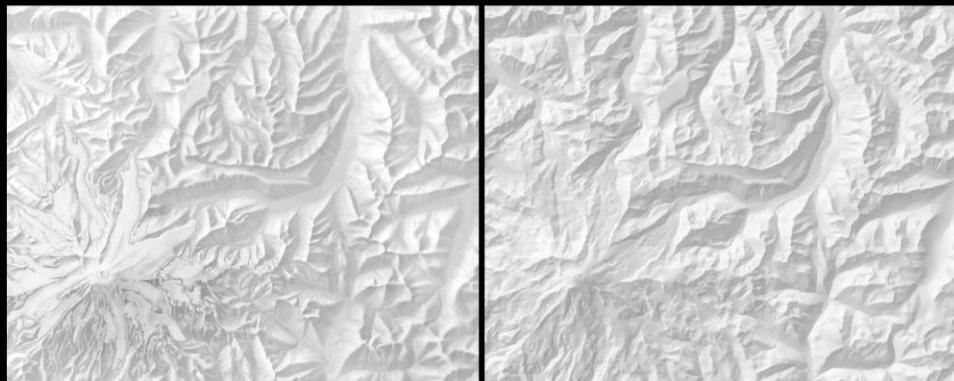


[Hertzmann 2000]

Hatching is a technique for creating shading and suggesting shape using strokes of roughly similar width and length. Rules for effective shading are well laid out in the art literature. Four more or less constant levels of tone are commonly used: highlights, or essentially blank spaces, such as on the hand or the folds in cloth; single hatching, as on the body of the jacket; cross-hatching, as in the shadows of the folds; and undercuts, essentially blacked out areas in deep shadow. An important point is that hatching lines are usually rather straight, though tending to bend somewhat with the underlying surface. Hatching lines that follow the surface too closely for too long tend to appear as markings on the surface itself. Hertzmann's 2000 paper does a nice job of working these observations into an effective algorithm, with an example result on the right.

# Shading

- False light and shade emphasize shape
- Fine scale details are exaggerated



U.S. National Park Service

[Rusinkiewicz 2006]

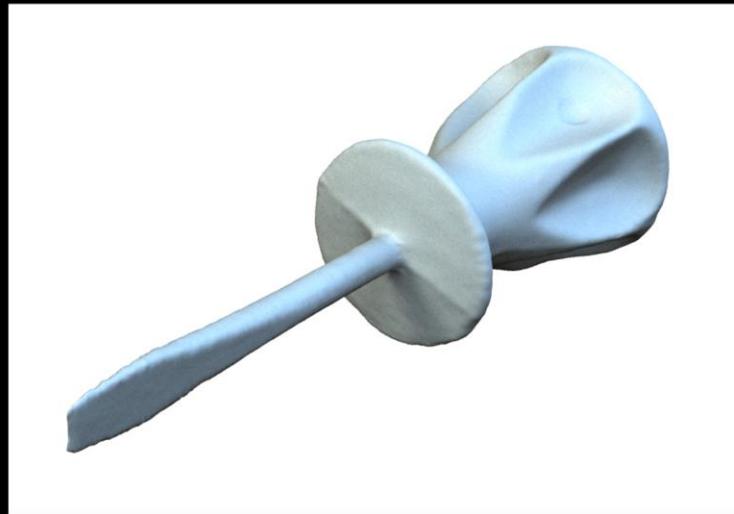
Shading style varies widely, but it is common for artists to exaggerate shading and use false light to bring out the shape of their subject. In the case of terrain relief, rules for shading are codified well enough that an artist's shading (left) can be reproduced quite closely (right). This algorithm uses a multi-resolution approach to locally manipulate the light direction much as an artist might, giving small details extra prominence and the curving sides of valleys roughly similar levels of shade.

## ... And The Bad News

- Not everything so well explained
- Even “simple” drawings are quite complex

Unfortunately, not all the interesting effects are so well explained in the art literature. Even the simplest of line drawings by human artists contain effects that are difficult to effectively describe with our current knowledge.

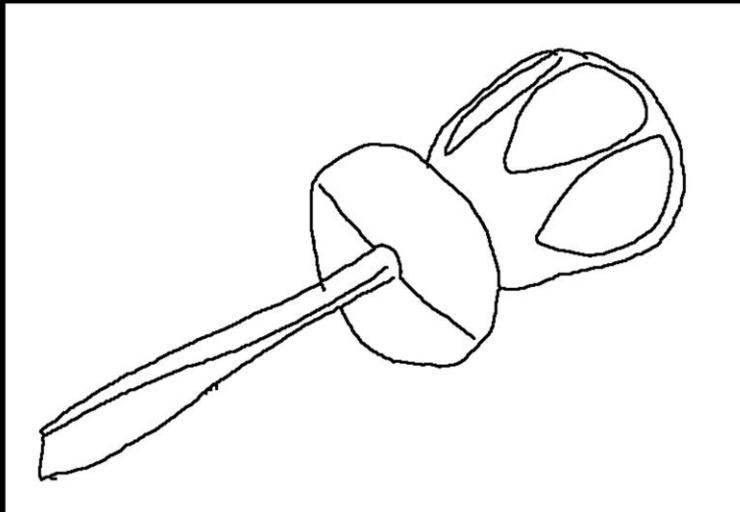
# How to Draw this Shape?



[Cole 2008] (*model by Cyberware, Inc.*)

For example, consider making a simple, pure line drawing of this shape, with the object of conveying the shape as well as possible. Ignore abstraction, hatching, or shading entirely.

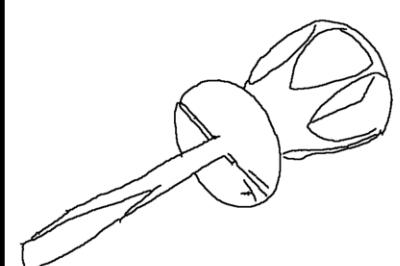
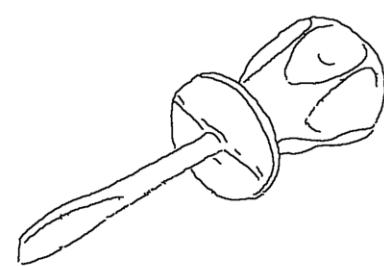
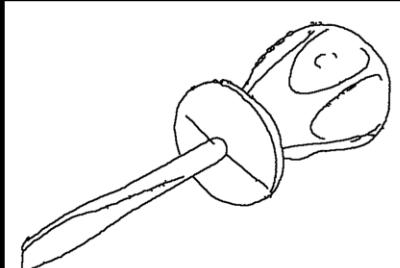
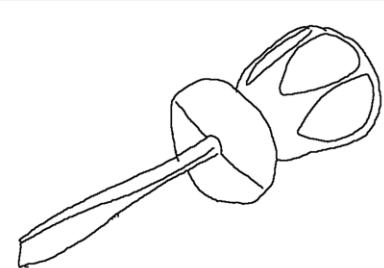
# One Answer



[Cole 2008]

This is one answer to that prompt from an actual artist. Note that the drawing has been scanned and filtered so that each line is constant thickness and strength. This artist made some fairly obvious decisions, such as drawing all the silhouettes, but also some idiosyncratic decisions, such as extending the crease line at the front of the screwdriver far along the shaft.

## Several Answers



[Cole 2008]

Other artists made similar decisions in many cases, but each drawing is unique. One shared property between all the drawings is the way the rear of the screwdriver is represented by three independent components, though some artists chose to complete each loop and some chose to leave the rest of the loop implied. Each drawing has its strengths and weaknesses, but they all get the important bits pretty much right. Lets look at how we might try to formalize these lines.

## Feature Line Models

- Use CG drawing algorithms as models
- Known algorithms
  - Image edges
  - Image ridges and valleys
  - Geometric ridges and valleys
  - Extension and anticipation of contours

There are several line drawing algorithms that can provide models for artists' lines. These algorithms will be described in detail later on in the course, but for now I'll just explain briefly.

# Image Edges

- Image edges perceptually important
- Most simple algorithm we know
- Various implementations exist
  - [Canny 1986]
  - [Kang 2007]



[Kang 2007]

We know that edges in an image are perceptually important. Extracting edges from an image is also simple, in fact, it is probably the simplest line drawing algorithm we know. Variations on the theme include Canny edge detection and more sophisticated methods such as Kang 2007.

# Image Ridges and Valleys

- Represent areas of dark and light
- Special cases describable in object space

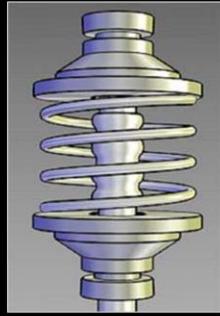


[Lee 2007]

Besides image edges, we can look at ridges and valleys of the image. There is some reason to believe that artists sometime abstract shading with lines, which would correspond to making lines along the image intensity ridges and valleys. Although these lines are most obviously image space features, later on we will see how to describe some special cases in object space.

# Geometric Ridges and Valleys

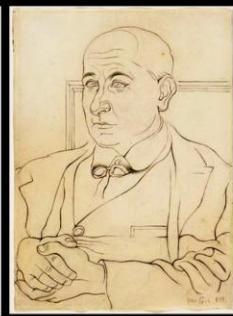
- “Smoothed creases”
- Semantically important features
- Not dependent on lighting



[Gooch 1998]



Matisse



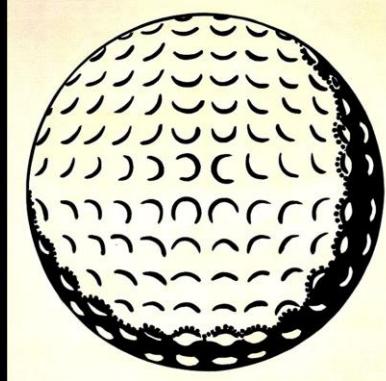
Juan Gris

from [Judd 2007]

Creases (lines of normal discontinuity) are obviously important lines to draw, as shown by the left image. Geometric ridges and valleys are essentially smoothed creases, i.e., lines where the normal is changing rapidly in one direction. These lines are intrinsic properties of the shape, and are therefore not dependent on lighting. Some examples of artists renderings using ridges are shown at right. The most prominent ridge-like features in both are along the nose.

# Extensions of Contours

- Lines that are contours in a nearby view
- Not dependent on lighting



Roy Lichenstein



[DeCarlo 2007]

Occluding contours are known to be important lines to draw. Artists also seem to extend these contour lines, and draw lines where no true contour exists, but a contour would exist if the viewpoint were slightly changed. These lines are also not dependent on lighting – note that the lines on the lower right of the golf ball (closer to the shade) are the same as the lines on the upper left (closer to the light). Later in the course you'll see how to formalize these lines to produce images such as the one on the right.

## Which Model Is Best?

- All seem to work sometimes
- But which, where, and when?

We are left with the question of which model of lines to use. All the models appear to work in some cases, but it is not clear when and where to use each one.

# Problems with Modeling

- Shape unknown for existing drawing
  - Shape usually inferred *from the drawing itself*
- No ground truth for known shapes
  - Hard to tell how well model predicts



John Flaxman

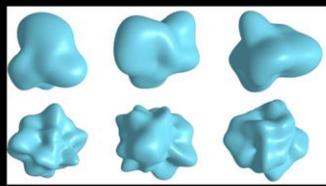


[DeCarlo 2003]

There are greater problems with the modeling approach, however. Even if we come up with a model that produces very similar lines to a given drawing, we have no way to evaluate how well the model truly matches. The shape that the artist considered when making the original drawing is unknown, and in fact, we often beg the question by inferring the shape from the drawing itself. Further, even when we do know the exact shape (such as when we have the 3D model), we cannot make a direct comparison back to the original drawing, since the resulting drawings are completely different. We need a way to directly compare an artist's drawing with our models.

# Direct Comparison

- Artists draw known shapes
- Drawing prompt possibilities
  - A physical 3D model with known properties
  - An animated 3D rendering of a model
  - A set of still images of a model



[Phillips 2005]

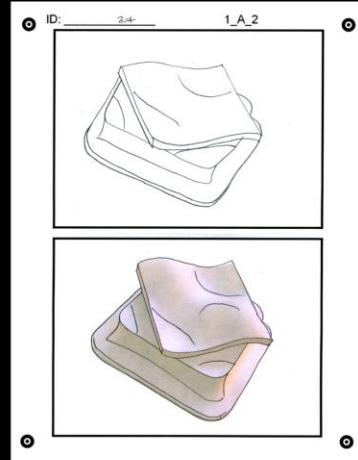


[Cole 2008]

Recently a few studies have attempted to capture drawings of known shapes. The artists are asked to draw under controlled conditions, and their results captured. There are at least three options for drawing prompts: a physical 3D model, and animated rendering, and a set of still images. Phillips 2005 uses all three, while one of our current papers does only the last.

# Experiment Setup

- Drawings must be registered to 3D model
- Two possibilities
  - Artists draw freely, algorithm registers [Phillips 2005]
  - Artists register drawing themselves [Cole 2008]

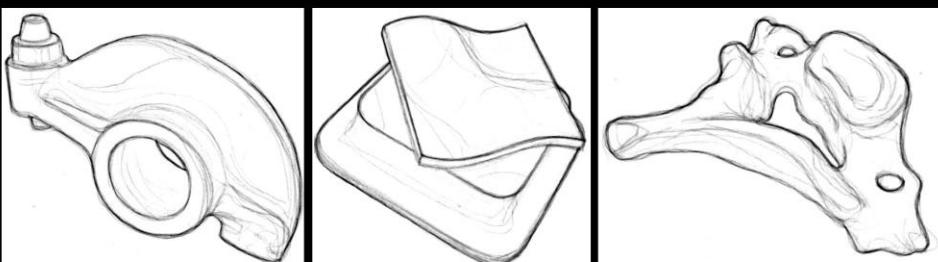
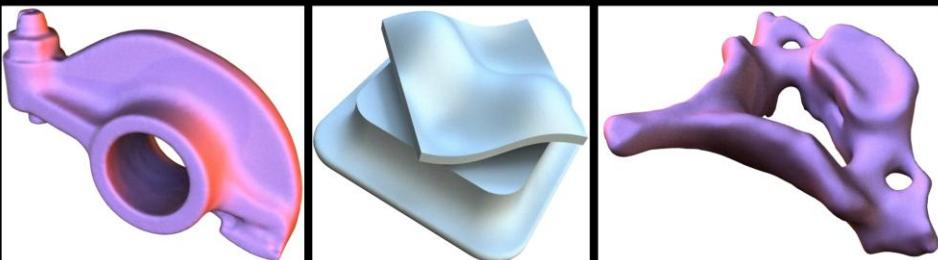


[Cole 2008]

Of course, for the drawings to be useful they must be registered somehow to the 3D model. Phillips allows the artists to draw freely, and then uses a matching algorithm to register the drawings to the model. While this method works for any prompt and provides the artist with maximum freedom, it does not offer good accuracy and the matching problem is hard and ambiguous.

Another option is to ask the artists to register the drawings themselves. The artists first make a freehand drawing, as in the top of the page, and then copy their drawing on to a faint shaded version of the prompt. This method gives good accuracy and works around the matching problem by having the artists solve it for us.

## Experiment Results

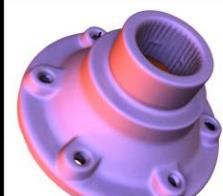


[Cole 2008]

Here are some example results from our recent study. At the top you see the prompt image, and at bottom the average of many different artists drawings of that prompt. The artists seem generally guided by image space shading features, especially edges, which confirms the intuition that artists "draw what they see." Occluding contours are obviously very important, but features such as the circular ridge of the rockerarm are also highly agreed upon by different artists.

# Complex Effects

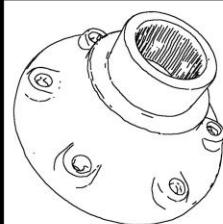
- Image space is not the whole story
- Known definitions explain most lines
  - But not at high precision



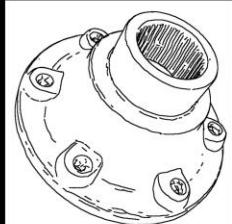
Prompt image



Artists' composite



Ridges (high)



Ridges (low)

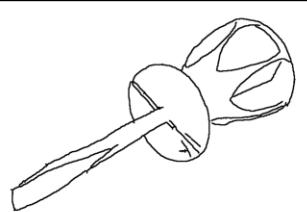
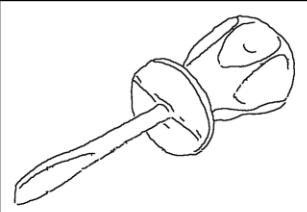
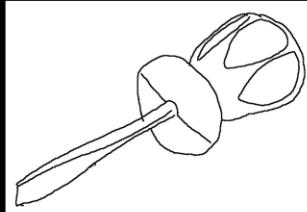
[Cole 2008]

Image space features are not the whole story, however. The average drawing for this shape seems to be best described by the geometric ridges and valleys, because the circular lines remain relatively constant through changes in the image intensity.

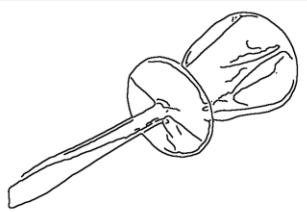
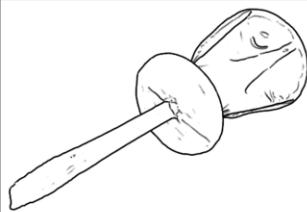
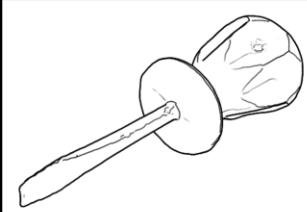
However, while all the artists' lines can be classified as ridges or valleys, just selecting ridges and valleys by strength does not give us a matching to the artists' drawings. The artists made some complex editing decisions to leave out the lines along the top of the mounting holes, while leave in the lines around the whole circumference of the shape. As you can see from the two lower images, using a high enough threshold to exclude the short valley lines excludes the long ones as well, while using a low enough threshold to include the long lines includes the short lines as well.

# Artists vs. CG

Human Artists



CG Algorithms



Ridges and Valleys

Sug. Contours and Highlights

Image Edges

This brings us back to the screwdriver example. While not mentioned before, the screwdriver drawings were made by artists in our study. We therefore have the 3D model and view, and can create CG line drawings for a direct comparison. At top are three of the humans' drawings, and below are three CG drawings, roughly selected to match the humans' drawings.

In the first pair, the shaft line is extended all the way to the hilt, and the loop features at the rear are completed. Note, however, that in the ridge and valley image, lines between loops are shared, unlike in the artist's drawing.

In the second pair, the dimple in the top of the screwdriver is recorded, and the lower section of a loop in left implied as in the artist's drawing. However, the shaft line is missing, as is the back of the top loop.

In the third pair, the crooked line in the middle of the side loop is present, along with the shaft line and some of the messy lines at the front of the screwdriver. However, the top loop is almost omitted in the image edges drawing, while it is completely present in the artist's drawing.

## Summary

- Artists still ahead of CG
- CG close behind in well defined areas
- Future work may require experiments

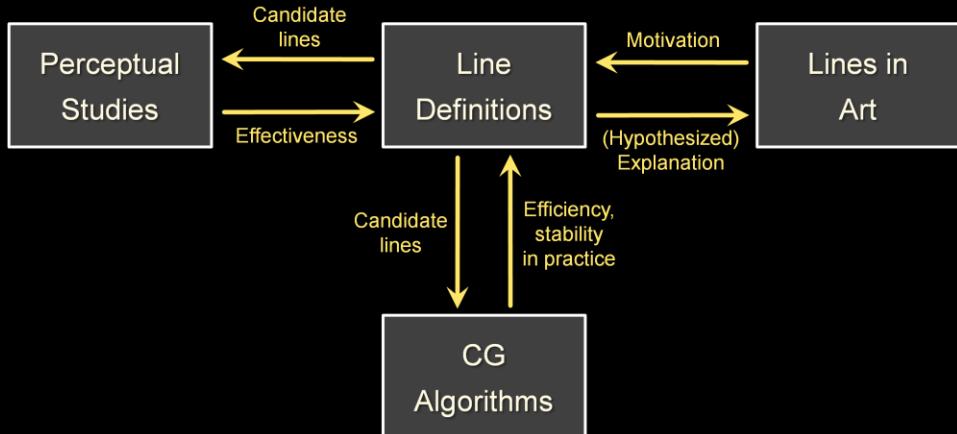
In conclusion, human artists are still ahead of CG algorithms in many respects. However, CG is close behind in the case where the artistic technique is well explained and can be easily formalized. Since no well known formalizations exist for much of artistic technique, however, future study of human line drawings may require experiments with artists in order to make progress.

# Part III: Mathematical Description of Lines

Szymon Rusinkiewicz

Line Drawings from 3D Models  
SIGGRAPH 2008

# Mathematical Description of Lines



In this section of the class, we will look at mathematical definitions of linear features on surfaces. This is an important component in our study of line drawings, since it serves to formalize the intuitions we get from looking at art. Moreover, the mathematical definitions can be turned into algorithms for producing candidate line locations on surfaces or images; this serves as the backbone of NPR line-drawing systems. Finally, as we will see later, we can ask questions about what shaped is perceived, given each different line definition.

# How to Describe Shape-Conveying Lines?

- Image-space features
- Object-space features
  - View-independent
  - View-dependent



[Flaxman 1805]

Here's a hand-drawn illustration by John Flaxman that illustrated a 19th century translation of the Odyssey. Notice how there are a variety of lines illustrating various effects, including things like shading, but in particular there are many lines that convey shape. When a viewer looks at these lines, these lines are naturally interpreted as indicating shape: they are not perceived as lines drawn on the surface!

In studying these shape-conveying lines, people have proposed three categories of mathematical definitions. First, there are “image-space” lines corresponding to features extracted from an image. In the case of lines from 3D models, the images are usually renderings with Lambertian reflection, in most cases using a headlight as the only light source. Next, there are “object-space” features computed directly on the 3D surface. Finally, there are object-space features whose definition depends on the view position.

# Image-Space Lines

- + Intuitive motivation; well-suited for GPU
- Difficult to stylize

Examples:

- Isophotes (toon-shading boundaries)
- Edges (e.g., [Canny 1986])
- Ridges, valleys of illumination  
[Pearson 1985, Rieger 1997,  
DeCarlo 2003, Lee 2007, ...]

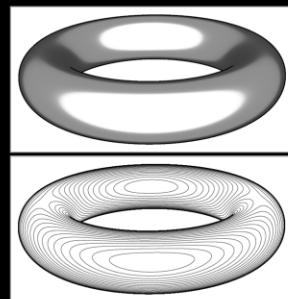


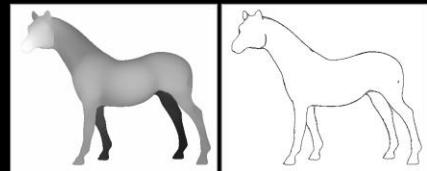
Image-space lines have a very intuitive motivation and a strong connection to art: artists are frequently taught to “draw what they see”. Another big advantage is that they translate into efficient CG algorithms, because they can often be implemented in graphics hardware. On the other hand, such GPU-computed lines can be difficult to stylize, since they are often extracted as collections of pixels rather than as complete curves.

As an example of image-space lines, isophotes are curves of constant illumination, which also correspond to toon shading boundaries.

# Image Edges and Extremal Lines

Edges:

Local maxima of  
gradient magnitude,  
in gradient direction



Ridges/valleys:

Local minima/maxima of  
intensity, in direction of  
max Hessian eigenvector



Another definition for image-space lines is edges. Using the definition of the popular Canny edge detector, these are locations where the image gradient magnitude has a local maximum, when looking in the gradient's direction. (In practice, the gradient direction is sometimes quantized to 45-degree increments, to simplify finding the local maxima.) Canny's algorithm also includes a sophisticated system of hysteresis thresholding, to keep the most important lines.

A final very important class of image-space lines are the image intensity “ridges” and “valleys”. These are curves on the surface of locally maximal/minimal intensity, and are very naturally drawn in white/black on a mid-tone color, as in this example from Lee et al. There are several different definitions of image ridges and valleys, many of which were originally developed in the course of analyzing watercourses on terrain (see, for example, Rieger’s paper). The definition used by Lee et al. looks for local maxima/minima of intensity, in a direction determined by fitting a paraboloid locally and looking for the direction of highest curvature. In other words, the direction is the eigenvector corresponding to the largest-magnitude eigenvalue of the image Hessian (matrix of second partial derivatives).

## View-Independent Object-Space Lines

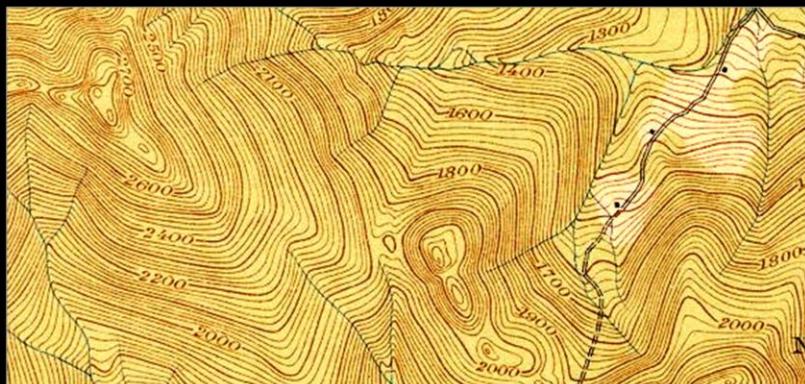
- + Intrinsic properties of shape;  
can be precomputed
- Under changing view, can be  
misinterpreted as surface markings

For the next class of lines, we consider “object-space” definitions that look directly at the 3D shape of a surface. View-independent lines are those that are defined only based on the shape, without considering the viewer’s position. This is somehow intuitively satisfying, since these lines depend only on the surface shape, and a view-independent definition means that these lines can be precomputed and then re-used for multiple viewer positions.

However, it has been observed that, under changing view, these lines are more naturally interpreted as markings on the surface, rather than as lines depicting the shape of the surface. We believe that further perceptual studies will certainly be necessary to investigate this.

# View-Independent Object-Space Lines

Topo lines: constant altitude

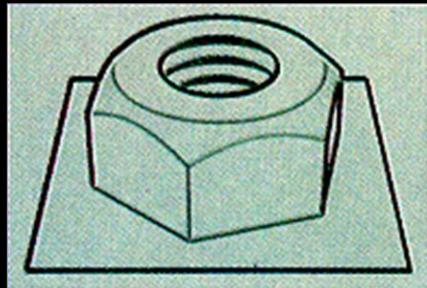


[USGS]

One example of view-independent lines are the constant-altitude lines found on topographic maps. They certainly are effective at conveying shape, both through the shape of the lines and their density.

# View-Independent Object-Space Lines

Creases: infinitely sharp folds



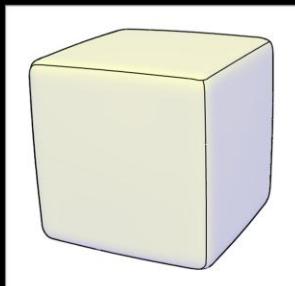
[Saito & Takahashi 90]

Another example, appropriate for objects with sharp folds (i.e., discontinuities in the normal), is lines at those creases of the surface. This works well for polyhedral objects, and is a frequent ingredient in technical drawings. The definition is very simple: just look for a dihedral angle (that is, the angle between two faces connected along an edge) smaller than a threshold.

# View-Independent Object-Space Lines

Ridges and valleys (crest lines)

- Local maxima of curvature
- Sometimes effective, sometimes not



[Thirion 92, Interrante 95, Stylianou 00, Pauly 03, Otake 04 ...]

Unfortunately, the natural generalization for smooth surfaces, ridge and valley lines, leads to mixed results. For example, the ridge lines on the rounded cube at left successfully convey its shape. If you look at the picture on the right, you can see that some of the ridge and valley lines, such as those around the eyes, do a good job of marking features. However, other lines look like surface markings and no good artist would include them in a hand-made drawing.

The definition of ridge and valley lines is slightly complex: they are local maxima of principal curvature, in the corresponding principal direction. (There will be a description of differential-geometry principles later, which should clarify what this means!)

## View-Dependent Object-Space Lines

- + Seem to be perceived as conveying shape
- Must be recomputed per frame

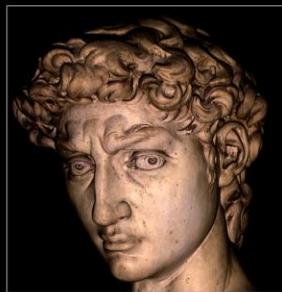
This brings us to the third class of mathematical line definitions, and the one on which we will spend the most time. These are lines defined on the 3D surface, but taking the view position into account. (To be precise, sometimes we think of an orthographic viewer, and so talk about a view direction rather than a viewer position.)

These are the lines that appear to be most effective at conveying shape, and lend themselves naturally to many stylization techniques. On the other hand, they do have to be recomputed whenever view changes, so they aren't necessarily the most efficient (though there has been some progress in GPU-based implementations).

# What Lines to Draw?

Silhouettes:

- Boundaries between object and background



First, we start with the silhouette: the boundary between the object and the background. Here, we draw the silhouette on the right, superimposed over a contrast-reduced version of the photograph on the left (just so we can see what's going on). Silhouettes are obviously very important, and are an essential ingredient in any line drawing. However, as you can see here, they're clearly not enough.

# What Lines to Draw?

Occluding contours:

- Depth discontinuities
- Surface normal perpendicular to view direction

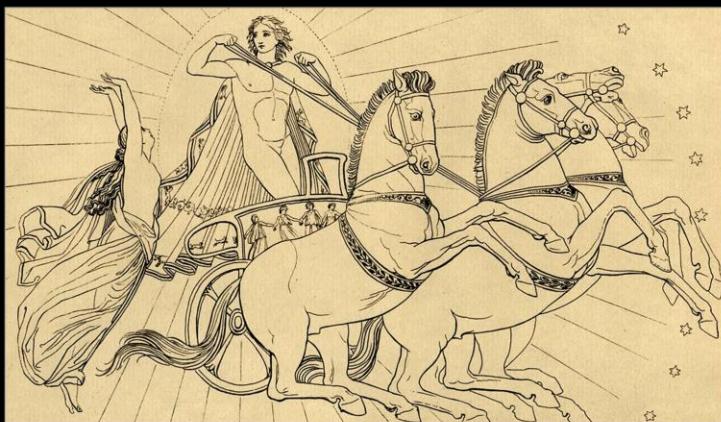


[Saito & Takahashi 90, Winkenbach & Salesin 94, Markosian et al 97, ...]

Another common definition is a generalization of silhouettes called occluding contours (or sometimes “interior silhouettes”). These mark any depth discontinuities, not just those against the background. As seen on the right, this adds a lot of important detail to the drawing, but still does not convey shallow features, particularly those viewed head-on. Still, these are a common component in NPR systems.

# What Lines to Draw?

There are other lines...

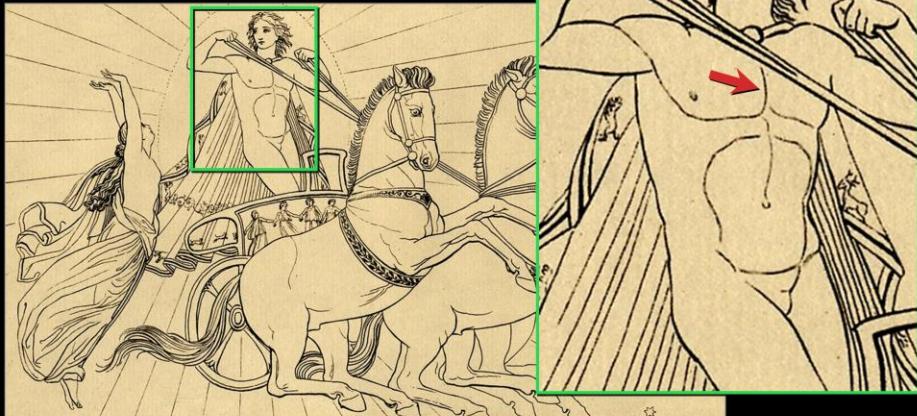


[Flaxman 1805]

So, to complement silhouettes and contours we need another line definition. If we go back and look at a real line drawing, we see that there are, in fact, more lines that pretty clearly are not contours (and not ridges or valleys).

# What Lines to Draw?

There are other lines...

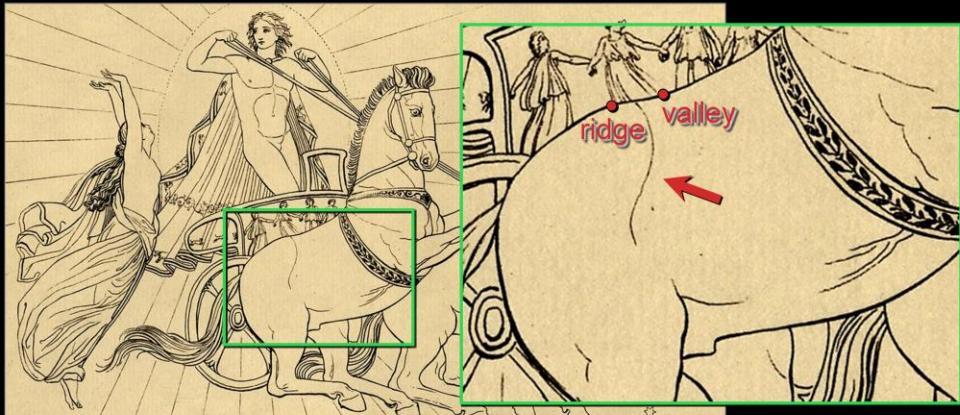


[Flaxman 1805]

For example, there's this line on the chest that's clearly on the right side of the torso, not in the middle of the “valley”.

# What Lines to Draw?

There are other lines...



Hypothesis: some are “almost contours”

[Flaxman 1805]

Another example is the line on the back of the horse. See how it misses the ridges and valleys on the back of the horse. Also, it's certainly not a contour.

We hypothesize that these lines are “almost contours”: if you moved your head a bit to the left, this line would in fact become a contour. We call these lines “suggestive contours”, and we'll later see how to formalize what they are and how to find them on a surface.

# Suggestive Contours

“Almost contours”:

- Points that become contours in nearby views



contours



contours +  
suggestive contours

So, here’s what suggestive contours look like on the David. You can see that they complement the contours nicely (in fact, you can prove that they line up with the contours in the image). Also, they include a lot of the detail that’s missing in the contours-only drawing.

# Differential Geometry

Many shape-conveying lines based on surface differentials

First-order: surface normals

- Example: occluding contours occur where normal perpendicular to view direction

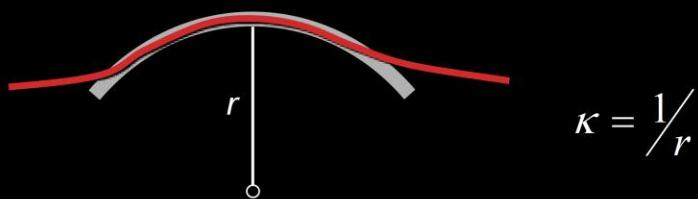
Other lines need higher-order derivatives

It turns out that in order to formalize different families of 3D lines on a surface, we'll need some math from a field called differential geometry. This is the field that concerns itself with what it means to take "derivatives" of curves and surfaces. You are already familiar with a first-order differential quantity of surfaces: the normal. In fact, as has already been mentioned, occluding contours critically depend on the normal: they are zeros of the dot product between the normal and the view direction. In a very similar way, different kinds of lines, like suggestive contours, will have definitions that depend on higher-order derivatives.

# Differential Geometry

Many lines based on curvatures

- Second-order differential properties of surface
- For a curve: reciprocal of radius of circle that best approximates it locally

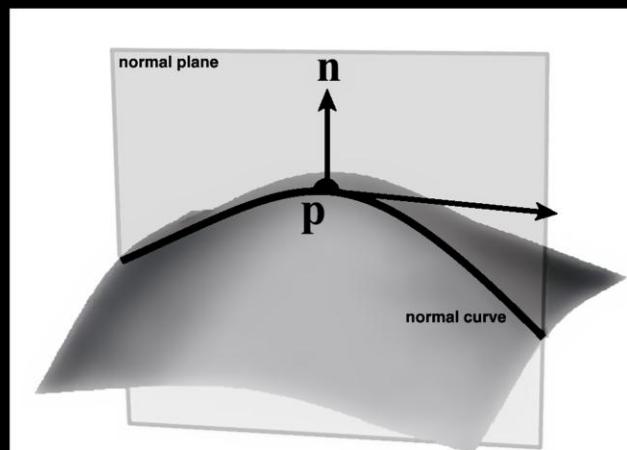


- For a surface: ?

So, let's move on to exploring second-order derivatives, or curvatures. To start with, let's recall the familiar definition of the curvature of a curve: at each point, it is the reciprocal of the radius of a circle that best approximates the curve locally. The sharper the bend in the curve, the higher the curvature. Curvature has units of one-over-length: if you scale an entire curve up by a factor of two, all the curvatures are halved.

# Normal Curvature

Curvature of a normal curve

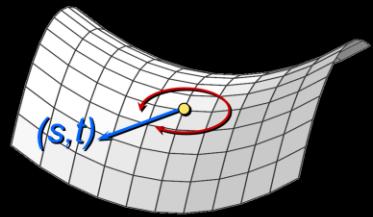


For a surface, we can talk about the curves formed by intersecting the surface with any plane containing the normal. These are called “normal curves”, and their curvature is “normal curvature”. So, for each point on the surface, there are many different curvatures, corresponding to all the different normal planes passing through that point.

# Curvature on a Surface

Normal curvature varies with direction,  
but for a smooth surface satisfies

$$\begin{aligned}\kappa_n &= (s \ t) \begin{pmatrix} e & f \\ f & g \end{pmatrix} \begin{pmatrix} s \\ t \end{pmatrix} \\ &= (s \ t) \mathbf{II} \begin{pmatrix} s \\ t \end{pmatrix}\end{aligned}$$



for a direction  $(s,t)$  in the tangent plane  
and a symmetric matrix  $\mathbf{II}$

There is something interesting that happens, though. For a smooth surface, the variation of normal curvature with direction can't be arbitrary – it has a very specific form. Imagine setting up a local orthonormal coordinate system in the tangent plane at a point on a surface. For any direction  $(s,t)$ , expressed in terms of that coordinate system, we can find the normal curvature in that direction in terms of a simple formula involving a symmetric matrix  $\mathbf{II}$ . This matrix is known as the “second fundamental tensor”, and as we’ll see is related to how much the surface is bent. Note that if you were to expand this formula you’d get terms quadratic in  $s$  and  $t$ : this whole expression is therefore just a fancy way of writing a quadratic form.

## Interpretation of $\mathbf{II}$

Second-order Taylor-series expansion:

$$z(x, y) = \frac{1}{2}ex^2 + fxy + \frac{1}{2}gy^2$$

“Hessian”: second partial derivatives

$$\mathbf{II} = -\begin{pmatrix} \mathbf{s}_{uu} \cdot \mathbf{n} & \mathbf{s}_{uv} \cdot \mathbf{n} \\ \mathbf{s}_{uv} \cdot \mathbf{n} & \mathbf{s}_{vv} \cdot \mathbf{n} \end{pmatrix}$$

Derivatives of normal

$$\mathbf{II} = \begin{pmatrix} \mathbf{n}_u \cdot \hat{\mathbf{u}} & \mathbf{n}_u \cdot \hat{\mathbf{v}} \\ \mathbf{n}_v \cdot \hat{\mathbf{u}} & \mathbf{n}_v \cdot \hat{\mathbf{v}} \end{pmatrix}$$

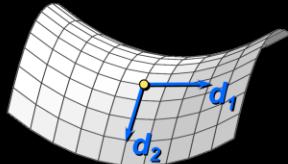
What exactly is  $\mathbf{II}$ ? What are its elements? It turns out that there are many formulas for them, all involving some notion of the second derivative of the surface. For example, they are precisely the second-order terms in a Taylor series expansion of the surface (assuming that  $z$  is oriented along the surface normal). Equivalently, they are the derivatives of the normal as you move along the surface.

Incidentally, if you look for formulas like this in various textbooks, there's a good chance you may see them written with the opposite sign. This is because when writing the formulas you need to establish the conventions of whether normals are considered to point into or out of the surface, and in addition whether convex surfaces are taken to have positive or negative curvature. We assume that convex surfaces have positive curvature, and we use the usual graphics convention of outward-pointing normals, leading to the signs used here.

# Principal Curvatures and Directions

Can always rotate coordinate system  
so that  $\mathbf{II}$  is diagonal:

$$\mathbf{II} = \mathbf{R}^T \begin{pmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{pmatrix} \mathbf{R}$$



$\kappa_1$  and  $\kappa_2$  are *principal curvatures*, and are minimum and maximum of normal curvature

Associated directions are *principal directions*

Eigenvalues and eigenvectors of  $\mathbf{II}$

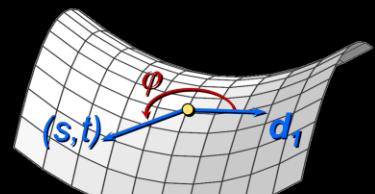
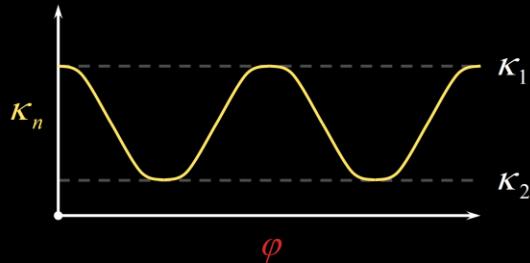
In many situations it is convenient to rotate the local coordinate system to make the matrix  $\mathbf{II}$  diagonal. It turns out you can always do this: the new coordinate axes are the eigenvectors of  $\mathbf{II}$ . (You might recall a neat theorem from linear algebra that the eigenvalues of symmetric matrices are guaranteed to be real: here's a real-life application that relies on this fact.)

Once you've done this change of coordinates, the new axes are known as the principal directions, and the corresponding curvatures are the principal curvatures. If we plug in the new form of  $\mathbf{II}$  into the formula for normal curvature, we see that all normal curvatures have to lie between the principal curvatures. So, the principal curvatures are the minimum and maximum curvatures for any direction (at that point on the surface).

# Euler's Formula

This lets us write normal curvature differently:

$$\kappa_n = \kappa_1 \cos^2 \varphi + \kappa_2 \sin^2 \varphi$$



This leads to something called Euler's formula for normal curvature, which expresses the curvature in any direction as a function of the principal curvatures.

## Gaussian and Mean Curvature

The Gaussian curvature  $K = \kappa_1 \kappa_2$

The mean curvature  $H = \frac{1}{2} (\kappa_1 + \kappa_2)$

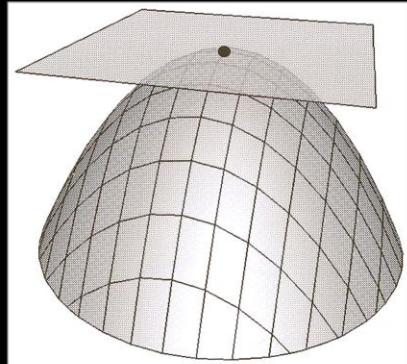
Equal to the determinant and half the trace,  
respectively, of the curvature matrix

Enable qualitative classification of surfaces

When talking about curvatures, there are a couple more terms that often crop up: Gaussian curvature and mean curvature. These are equal to the product and average of the principal curvatures, and can also be computed directly from  $\mathbf{II}$  (expressed in terms of any coordinate system), as the determinant and trace. Notice one interesting feature about Gaussian curvature: it has units of “curvature squared”, which is different from all the other flavors of curvature we’ve talked about.

Gaussian and mean curvature are very useful for qualitatively talking about the shape of a surface.

## Positive Gaussian Curvature: Elliptic Points

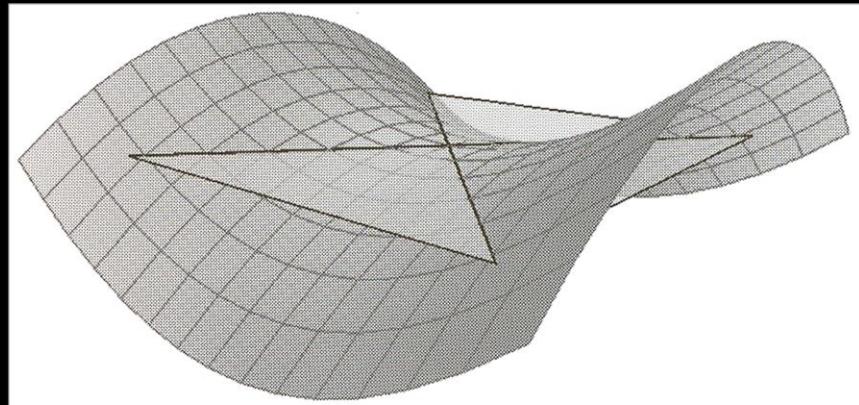


Convex/concave depending on sign of  $H$

Tangent plane intersects surface at 1 point

The most basic classification looks at the sign of Gaussian curvature. If it is positive, then the principal curvatures are either both positive or both negative (and you can tell which one by looking at the sign of the mean curvature). Points of positive Gaussian curvature are known as elliptic points, and are either convex or concave regions.

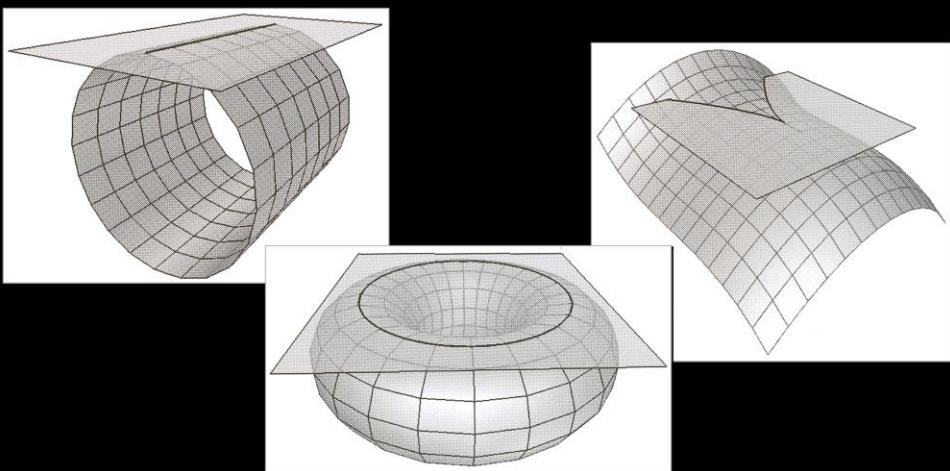
## Negative Gaussian Curvature: Hyperbolic Points



Tangent plane intersects surface along 2 curves

If the Gaussian curvature is negative, we have what are known as hyperbolic points, at which the surface is saddle-shaped. So, if you look in one direction the surface is convex, while in the perpendicular direction the surface is concave.

## Zero Gaussian Curvature: Parabolic Points



Tangent plane intersects surface along 1 curve

Finally, we have parabolic points, at which one of the principal curvatures is zero. The most basic shape with zero Gaussian curvature is a cylinder, but there are many more complex surfaces at which  $K=0$  as well. In general, except for degenerate cases like cylinders, the parabolic points will form curves on the surface (known as parabolic lines), separating regions of positive and negative Gaussian curvature.

## Historical Note

Mathematician Felix Klein was convinced that parabolic lines held the secret to a shape's aesthetics, and had them drawn on the Apollo of Belvedere...

He soon abandoned the idea...



[Hilbert & Cohn-Vossen]

As long as we're talking about parabolic lines in a course about line drawings, it would be remiss not to relate an anecdote about the mathematician Felix Klein, who thought that parabolic lines might, in fact, be interesting lines to draw on a surface. He had them drawn (probably by some poor grad student) on the Apollo of Belvedere.

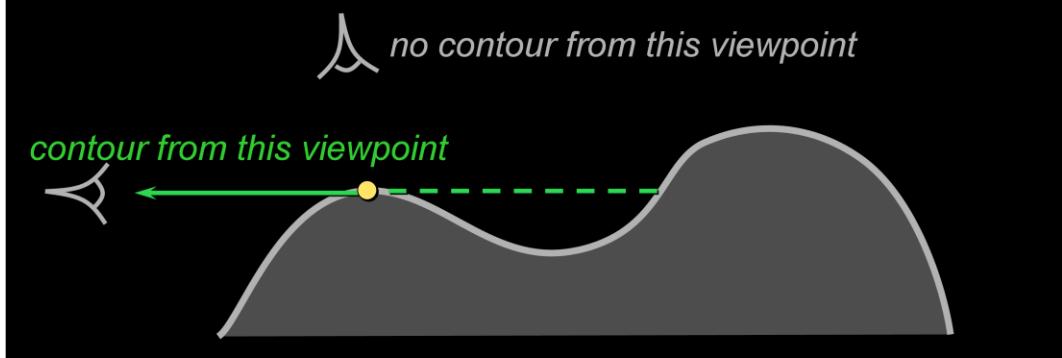
Unfortunately, the experiment didn't turn out that great.

A little later, we'll see that Klein wasn't entirely wrong: it is possible to select a subset of the parabolic lines that look OK...

# Occluding Contours

For any shape: locations of depth discontinuities

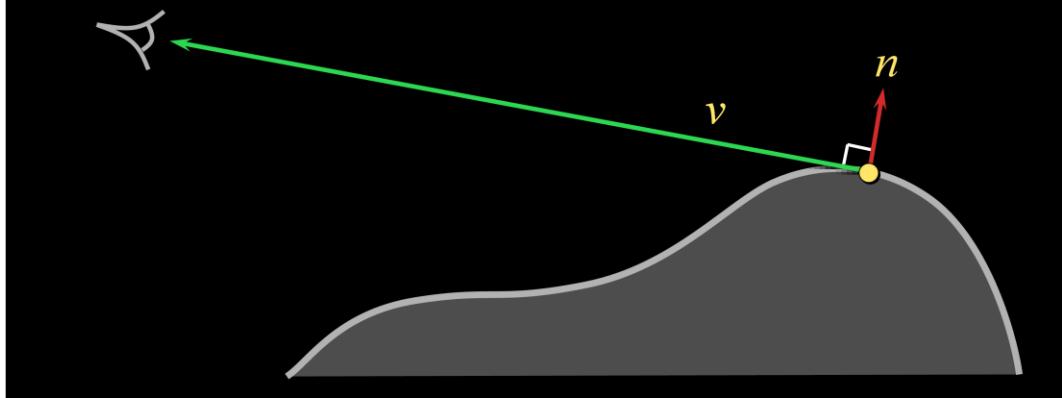
- View dependent
- Also called “interior and exterior silhouettes”



Now that we have some math under our belts, let us look in more detail at the different types of lines. We begin with occluding contours, sometimes also called “interior and exterior silhouettes.” There are a few different ways of defining these, of which a very straightforward definition is simply those locations at which, from the current viewpoint, there is a depth discontinuity. Note that these are view-dependent lines, which implies both benefits and drawbacks. On the plus side, the view dependence makes it much more likely that these lines are interpreted as conveying shape, rather than as surface markings. On the other hand, this means that the lines will have to be recomputed for each frame, and potentially makes it harder to do things like line drawings in stereo.

## Occluding Contours

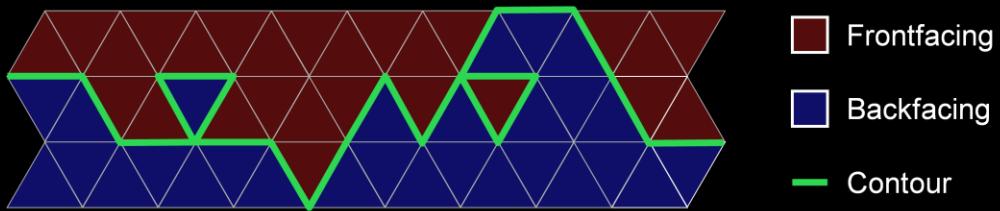
For smooth shapes: points at which  $n \cdot v = 0$



On smooth surfaces, there is another definition of contours that is useful: contours are those surface locations where the surface normal  $n$  is perpendicular to the viewing direction  $v$ . That is, places where  $n$  dot  $v$  is equal to zero.

# Occluding Contours on Meshes

Applying either definition on polygonal meshes  
can result in messy lines

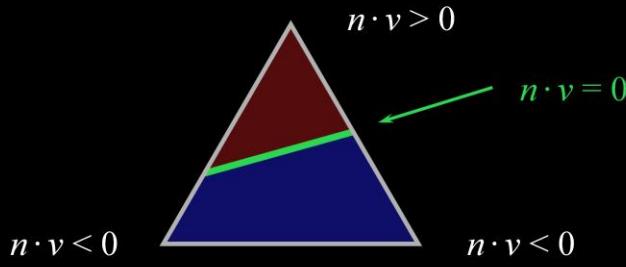


We can apply the same principles to polyhedra (i.e., polygonal meshes), but there's a problem. With only a tiny bit of noise, we can run into a situation where polygons along the boundary are "just barely" front- or back-facing, and the boundary between them is not a simple curve: it can entirely surround certain faces. This isn't necessarily a problem if the only thing you're doing is drawing the curve, since it will be viewed edge-on. However, if you are doing any further processing on the curve, such as trying to draw them with stylization, this can lead to big problems.

# Occluding Contours on Meshes [Hertzmann 00]

Alternative: interpolate normals within faces

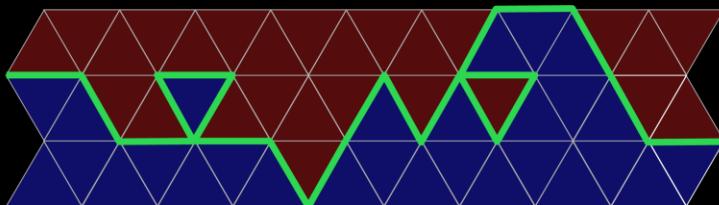
- Start with per-vertex normals
- Interpolate per-face (same as Phong shading)
- Compute  $n \cdot v$  at each point, find zero crossings
- Potential snag: visibility



So, there's a frequently-used alternative for polygonal meshes that tends to produce much nicer curves. It is a bit similar to Phong shading, in that it involves starting with per-vertex normals and interpolating them across a face. Once you know  $n$  at each point, you can find  $n$  dot  $v$ , and locate the curve on the face that corresponds to  $n \cdot v = 0$ . A slightly simpler variant of this is to just compute  $n$  dot  $v$  at the vertices, interpolate across the face, and figure out where the zero crossing is.

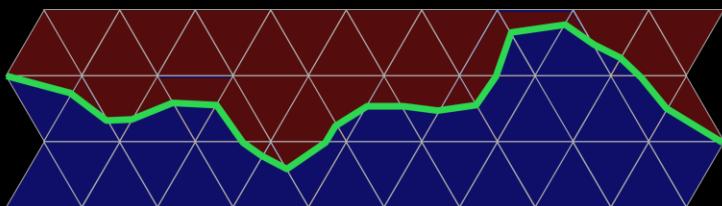
# Occluding Contours on Meshes

Contours along edges



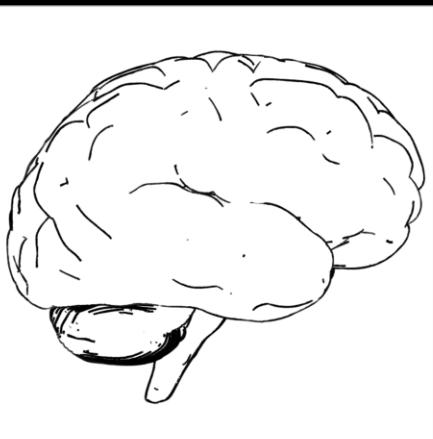
- Frontfacing
- Backfacing
- Contour

Contours within faces

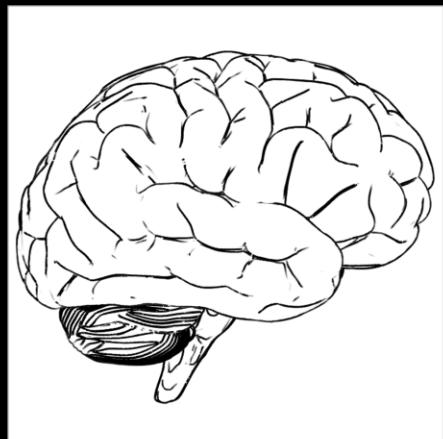


We can apply the same principles to polyhedra (i.e., polygonal meshes), but there's a problem. With only a tiny bit of noise, we can run into a situation where polygons along the boundary are "just barely" front- or back-facing, and the boundary between them is not a simple curve: it can entirely surround certain faces. This isn't necessarily a problem if the only thing you're doing is drawing the curve, since it will be viewed edge-on. However, if you are doing any further processing on the curve, such as trying to draw them with stylization, this can lead to big problems.

## Results...



contours



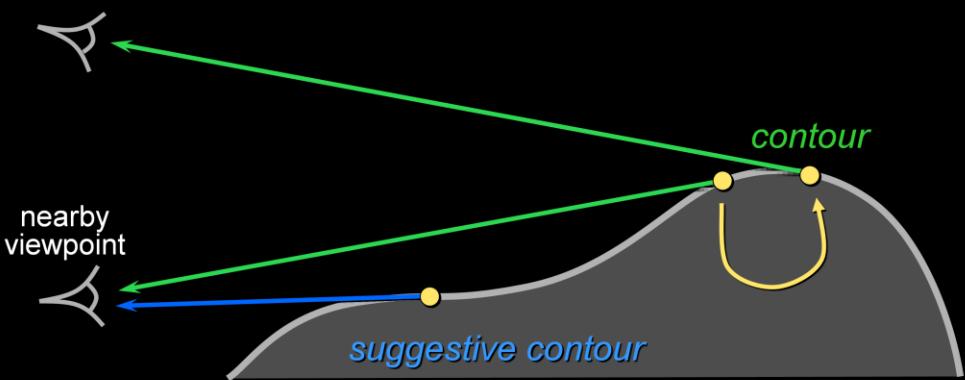
contours +  
suggestive contours

Here is your brain on contours. Here is your brain on suggestive contours. Any questions?

# Suggestive Contours: Definition 1

Contours in nearby viewpoints

(not corresponding to contours in closer views)

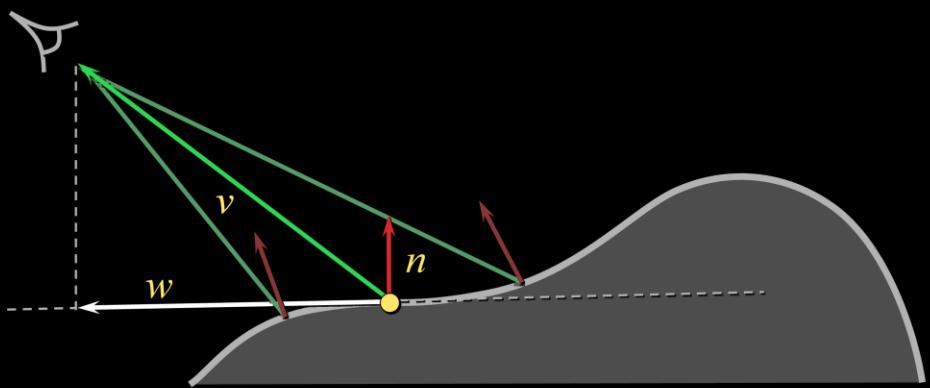


Now that we've seen contours, let's move on to defining suggestive contours. Here's the first definition: contours in nearby views. What happens if we start with the viewpoint at top (which produces a contour), then move the viewpoint down a little bit? First, the contour slides along a surface to a new location where its surface normal is perpendicular to the new view direction. But something else happens here too. A new contour appears that does not correspond to any in a closer viewpoint. This is a suggestive contour from the original viewpoint. The other contour corresponds to that in the original viewpoint, and is not a suggestive contour. Adding this qualification to our definition completes it.

## Suggestive Contours: Definition 2

$n \cdot v$  not quite zero, but a local minimum

(in the projected view direction  $w$ )



While intuitive, the first definition doesn't really lead to efficient algorithms for computing suggestive contours. So, let's look at a second definition (which can be proven equivalent to the first one). The idea is that suggestive contours are places where  $n \cdot v$  doesn't quite make it to zero (at which point we'd have a contour), but is a local minimum on the surface. That is, the location of a suggestive contour from this viewpoint (assumed to be distant) is where the normal is locally closest to perpendicular to the view direction, as you consider points along this normal slice of the surface. This involves moving in the direction "w", which we define to be the projection of  $v$ , the view direction, into the local tangent plane of the surface.

## Minima vs. Zero Crossings

**Definition 2:** Minima of  $n \cdot v$

Finding minima is equivalent to:  
finding zeros of the derivative  
checking that 2<sup>nd</sup> derivative is positive

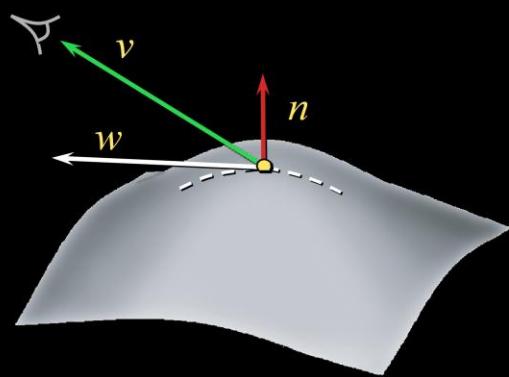
This leads to **definition 3**.

Derivative of  $n \cdot v$  is a form of curvature...

While definition 2 is better from the point of view of computation, we can transform it into yet another form that is still more convenient. The basic idea is that we're looking for local minima, so we use the usual definition that minima are places where the derivative is zero, and the next higher-order derivative is positive (which distinguishes them from maxima). Now, it turns out that derivatives of  $n \cdot v$  are related to curvature.

## Radial Curvature $\kappa_r$

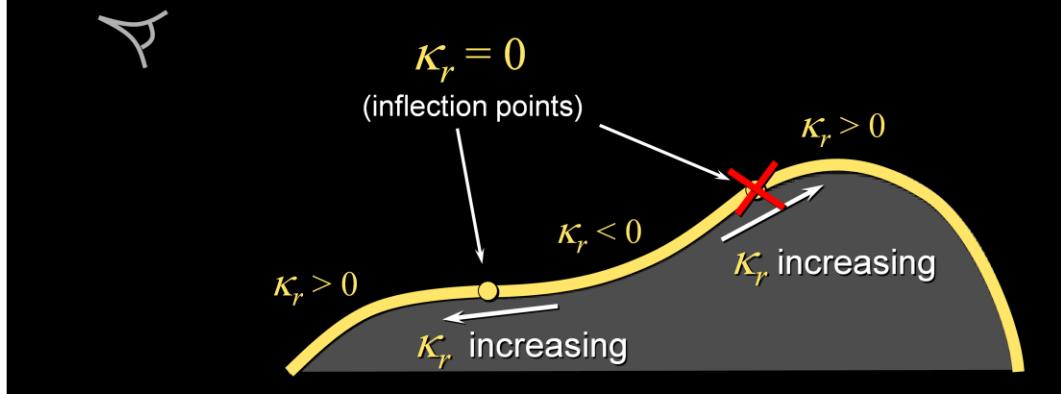
Curvature in projected view direction,  $w$ :



In particular, the derivative of  $n \cdot v$  has the same zeros as a quantity called “radial curvature”, which is just the curvature in the direction  $w$  (which, you’ll recall, is the projection of the view direction).

## Suggestive Contours: Definition 3

Points where  $\kappa_r = 0$  and  $D_w \kappa_r > 0$



So, our third definition of suggestive contours is that they are zeros of radial curvature, subject to a derivative test. This test needs to enforce that the “directional derivative” of radial curvature, in the direction  $w$ , is positive. To figure out what that is, we’ll need to go back and look at the next higher order of surface differentials.

## Finding Suggestive Contours

Finding  $\kappa_r$ :

$$\kappa_r = \mathbf{II}(\hat{w}, \hat{w})$$

Finding  $D_w \kappa_r$ :

?

So, to recap, the most computationally convenient definition of suggestive contours involves finding the zeros of radial curvature, which you compute by multiplying  $\mathbf{II}$  by  $w$  twice, then checking the sign of the directional derivative of radial curvature, which you get by multiplying the  $C$  tensor by  $w$  three times (there's also an extra term due to the chain rule, which accounts for the change of  $w$  itself as you move in the  $w$  direction).

## Derivative of Curvature

Just as  $\mathbf{II} = \begin{pmatrix} \frac{\partial n}{\partial u} & \frac{\partial n}{\partial v} \end{pmatrix}$  can define  $\mathbf{C} = \begin{pmatrix} \frac{\partial \mathbf{II}}{\partial u} & \frac{\partial \mathbf{II}}{\partial v} \end{pmatrix}$

$\mathbf{C}$  is a rank-3 tensor or “cube of numbers”

Symmetric, so 4 unique entries:  $\mathbf{C} = \begin{bmatrix} P^Q & Q^S \\ Q^S & S^T \end{bmatrix}$

Multiplying by a direction **three** times gives  
(scalar) derivative of curvature

Once we know about curvatures, derivatives of curvature are really nothing special. The only really interesting thing is that, as opposed to the normal (which was a vector) and the second fundamental matrix  $\mathbf{II}$ , the derivative of curvature is now a three-dimensional tensor, which can be thought of as a vector of matrices or as a “cube of numbers”. In order to get the derivative of curvature in a particular direction, you multiply this tensor by that direction three times.

## Finding Suggestive Contours

Finding  $\kappa_r$ :

$$\kappa_r = \mathbf{II}(\hat{w}, \hat{w})$$

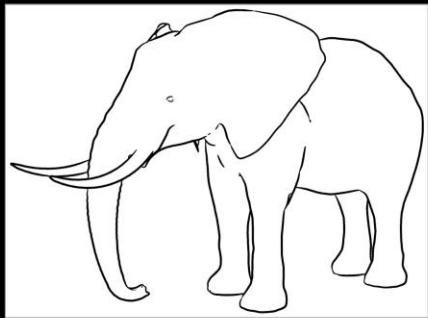
Finding  $D_w \kappa_r$ : (extra term due to chain rule)

$$D_{\hat{w}} \kappa_r = \mathbf{C}(\hat{w}, \hat{w}, \hat{w}) + 2K \cot \theta,$$

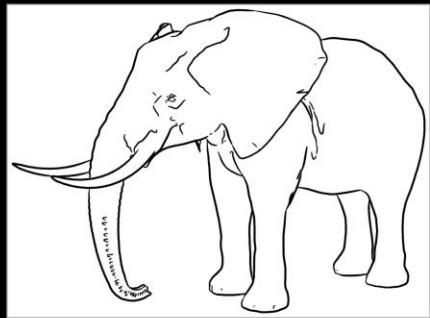
$$\text{where } \kappa_r = 0$$

So, to recap, the most computationally convenient definition of suggestive contours involves finding the zeros of radial curvature, which you compute by multiplying  $\mathbf{II}$  by  $w$  twice, then checking the sign of the directional derivative of radial curvature, which you get by multiplying the  $\mathbf{C}$  tensor by  $w$  three times (there's also an extra term due to the chain rule, which accounts for the change of  $w$  itself as you move in the  $w$  direction).

## Results...



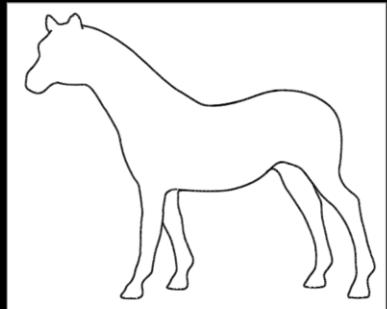
contours



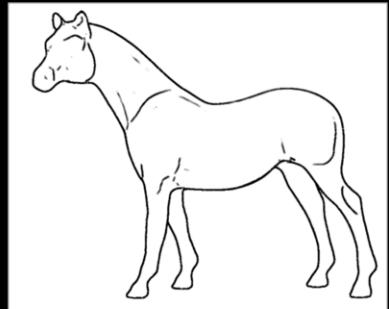
contours +  
suggestive contours

Here is your elephant on contours... (rest of not-a-joke omitted in the interests of good taste)

## Results...



contours

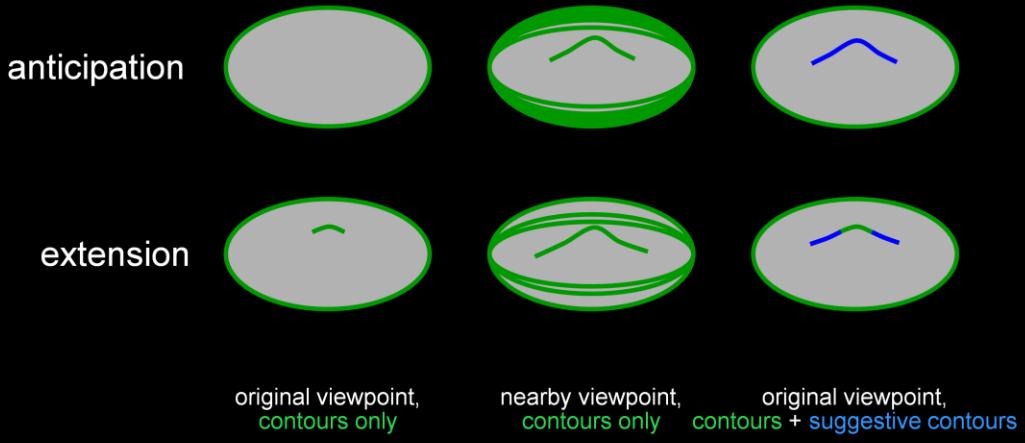


contours +  
suggestive contours

Here's another example...

# Qualitative Structure

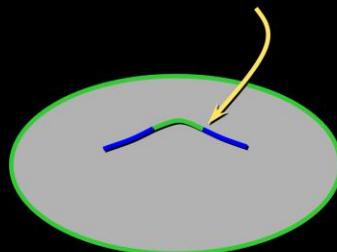
Suggestive contours have two behaviors:



It turns out that suggestive contours have some nice properties that let them complement contours very nicely. First, they can either “anticipate” contours by showing up in nearby viewpoints (i.e., definition 1), or “extend” contours in one view. Here we use the color convention that contours are green while suggestive contours are drawn in blue.

## Continuity of Extensions

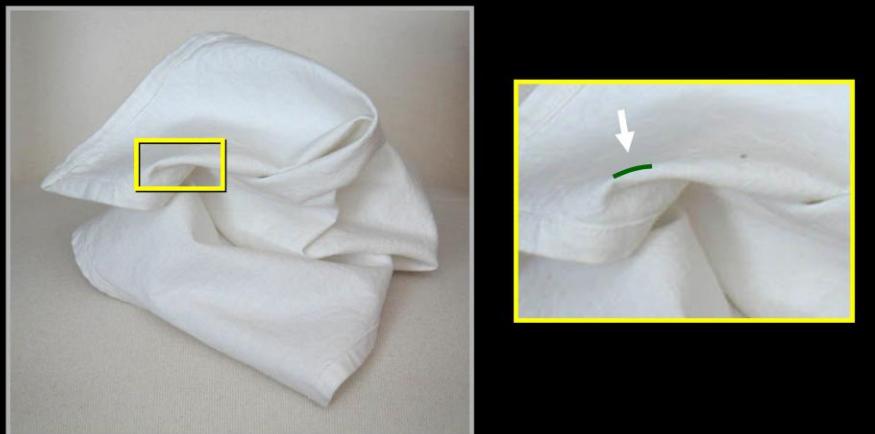
Suggestive contours line up with contours  
in the image



Moreover, in the case of extension, the suggestive contours line up (with  $G^1$  continuity) with contours in the image.

## Ending contours

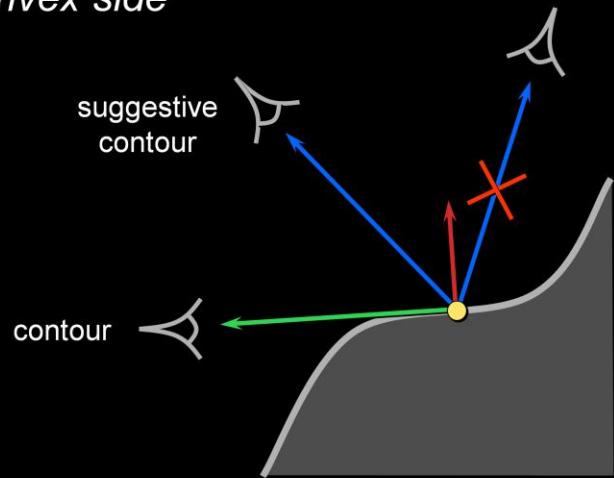
Difficult to localize in real images



The above property is useful, because in real-world images it can in fact be difficult to tell exactly where a contour ends. The fact that suggestive contours extend contours smoothly means that there is a single line that corresponds well to the behavior visible in such cases.

# Viewpoint Dependence

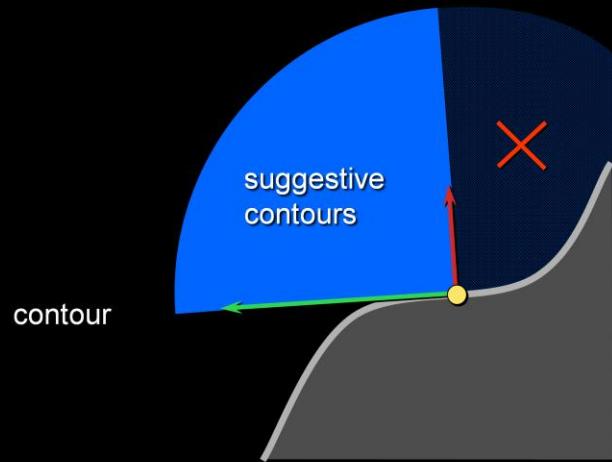
Suggestive contours appear at inflections  
viewed from *convex side*



Another property that becomes apparent from definition 3 is that contours show up at inflections (of normal curves) on the surface, but only when viewed from the convex side. In this case, the derivative test eliminates the suggestive contour at the rightmost viewpoint.

# Viewpoint Dependence

Suggestive contours appear at inflections  
viewed from *convex side*

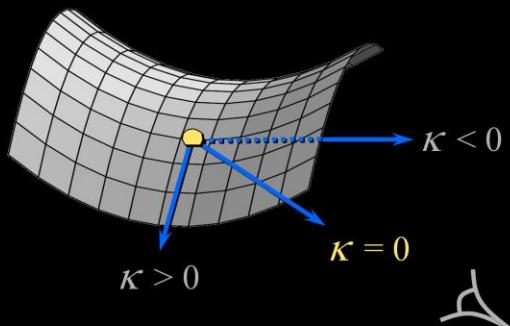


So, considering an inflection on a surface, there is some region of viewpoints from which suggestive contours get drawn, a region where they don't, and a threshold direction at which you start getting contours.

# Viewpoint Dependence

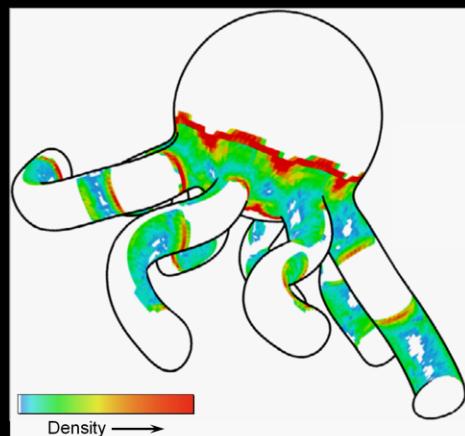
Suggestive contours move across surface

- At a typical point, inflections exist only when viewed from specific directions



Moving the viewpoint out of the plane, we see that suggestive contours can only happen when a surface is viewed from a very particular direction such that the curvature is zero. If you rotated the viewpoint one way, you'd get positive curvatures, and negative curvatures if you rotated the other way. Note also that having a direction for which the curvature is zero implies that the principal curvatures (which are the minimum and maximum limits for normal curvature) can't be both positive or both negative. This, in turn, implies that in order to get a suggestive contour, the Gaussian curvature must be negative (or at worst zero).

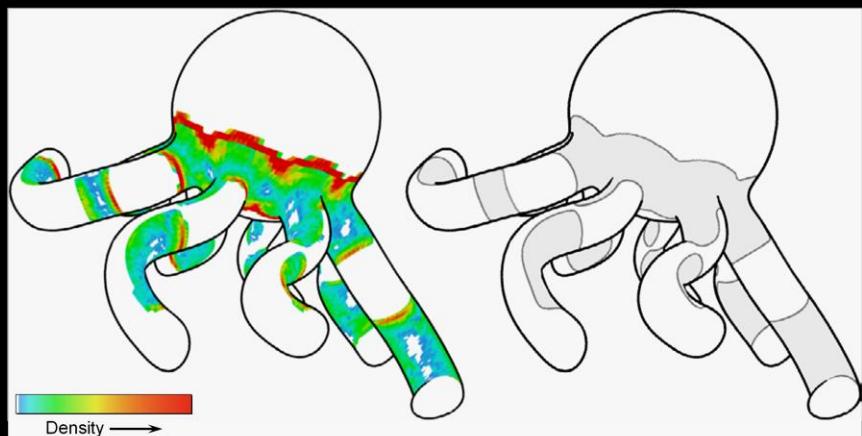
## Distribution of Suggestive Contours



Suggestive contour  
density over all views

This property can be illustrated empirically as well. Here we've taken a model and plotted a histogram of how many views have suggestive contours at each point on the surface.

## Distribution of Suggestive Contours



Suggestive contour  
density over all views

Regions where  
 $K < 0$

Comparing this to regions of negative Gaussian curvature, we see complete agreement. We also see the surprising fact that suggestive contours tend to hug the lines of zero Gaussian curvature (i.e., our friends the parabolic lines). We'll see later how to show this mathematically, but meanwhile let's think back to Klein's experiment. Even if the suggestive contours were always close to the parabolic lines, there's still a big difference between drawing them and our definition of suggestive contours: the derivative test.

## Zeros of $\kappa_r$ , H, and K



$$\kappa_r = 0$$



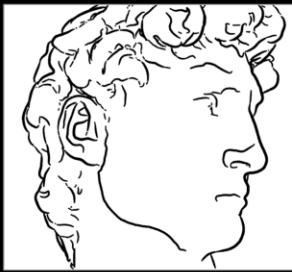
$$H = 0$$



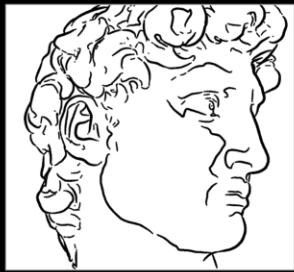
$$K = 0$$

In fact, if we didn't apply the derivative test, the lines of zero radial curvature would look pretty bad: just as bad as drawing all the parabolic lines. (Here we also show zeros of mean curvature for the sake of completeness.)

## Zeros of $\kappa_r$ , H, and K (with derivative tests)



$$\kappa_r = 0$$
$$D_w \kappa_r > 0$$



$$H = 0$$
$$D_w H > 0$$

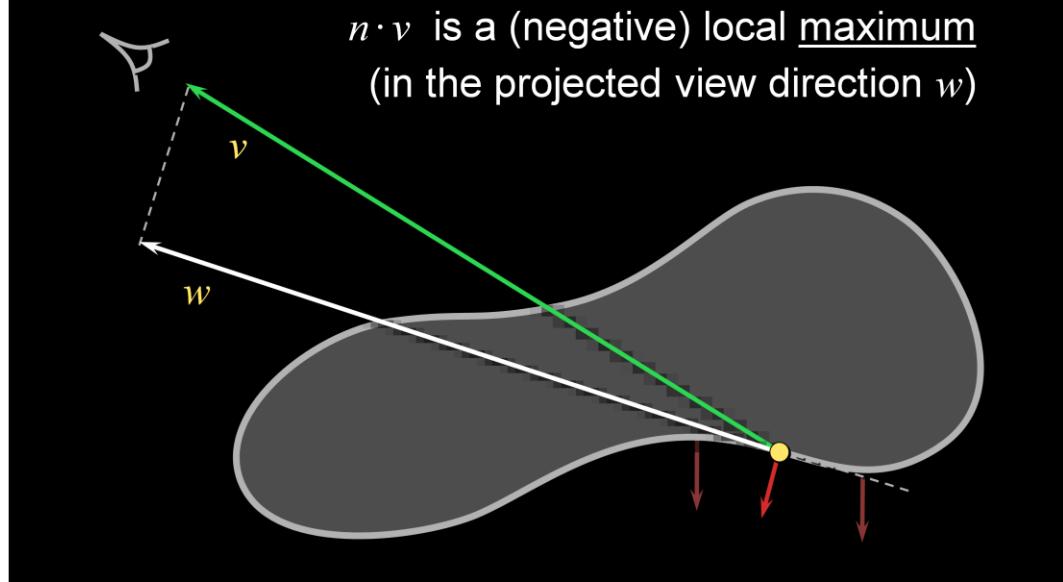


$$K = 0$$
$$D_w K > 0$$

If we add a derivative test, you can see that parabolic lines suddenly don't look so bad, though in general the suggestive contours still look better (and have the other properties of lining up with contours, etc.)

# Backfacing Suggestive Contours

[Burns 2005]



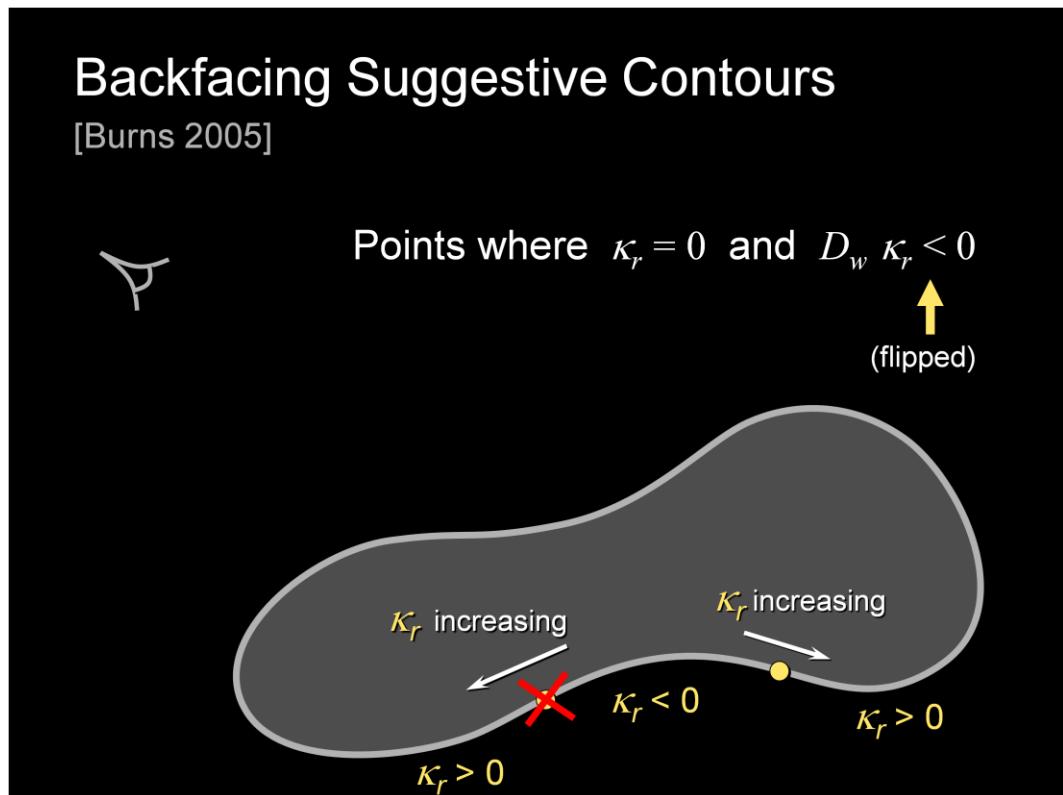
We also need to consider what happens with suggestive contours on backfacing parts of the object; such lines are an effective ingredient in transparent renderings. In fact, this was worked out in a paper on volumetric line drawings at Siggraph 2005.

The first definition of suggestive contours still applies: contours in nearby viewpoints.

We need to change definition 2 a little bit. When looking at how values of  $n \cdot v$  change across the surface, we are still looking for places where  $n \cdot v$  is almost but doesn't quite reach zero. The difference is that we're now looking for *maxima* of  $n \cdot v$ : negative maxima.

# Backfacing Suggestive Contours

[Burns 2005]



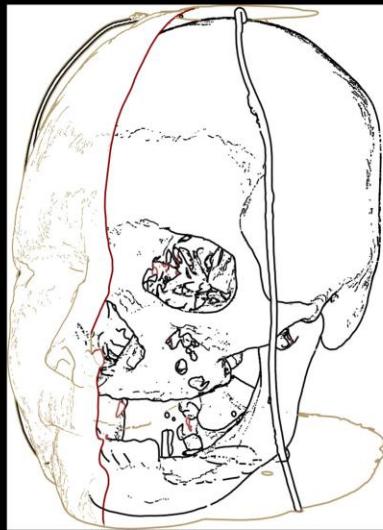
For the third definition, we are now looking for places where the radial curvature is zero where the radial curvature is increasing AWAY from the camera. So the sign of our derivative test gets flipped.

Perhaps a good way of thinking about this is that for backfaces, we're just considering what the inside-out version of the surface looks like (where the normals and curvatures are negated).

Of course, these suggestive contours still smoothly extend transparently rendered contours.

## Line Drawings from Volume Data

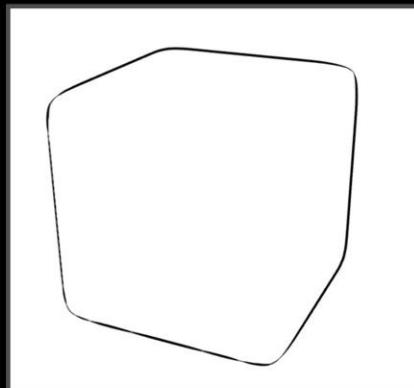
[Burns 2005]



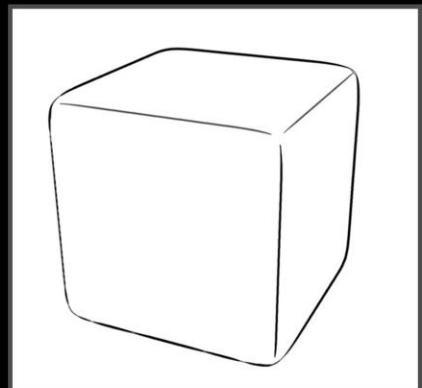
So, here's an example of contours and suggestive contours (together with cutting-plane intersections) produced from volume data.

# Suggestive Contours

No lines in convex regions



contours  
(no suggestive contours)



contours and ridges

Although (it is our belief that) suggestive contours are useful, there are shapes for which even this is not enough. For example, convex surfaces such as this rounded cube have no suggestive contours, yet probably need some more lines to be conveyed clearly.

## Line Drawings with Shading



from Frank Miller's *Sin City* (1991)

Another situation in which it is not clear that suggestive contours provide the right answer are two-tone comics, such as this example by Frank Miller.

## Line Drawings with Shading



from Frank Miller's *Sin City* (1991)

Zooming in, we see that the black line in the white region, which might be a suggestive contour, does not line up with the white line in the black region. Therefore, we hypothesize that there is something else going on with those white lines.

# Intensity Valleys in Images



suggestive  
contours

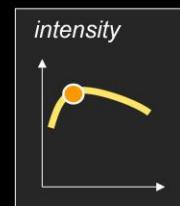
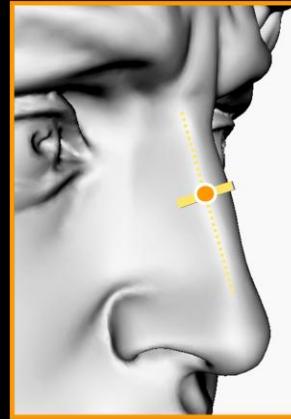
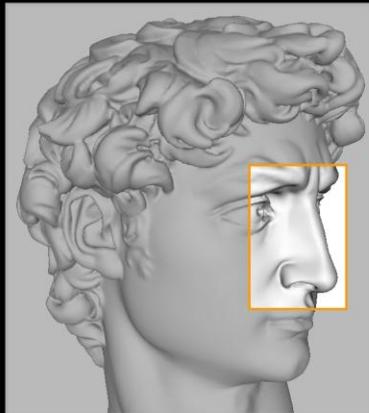


image  
valleys

To see how to add more lines, it is useful to step back a moment and compare suggestive contours with some image-space lines. For example, starting with the head-lit diffuse-shaded view at left, we see that there is a remarkable match between the valleys of illumination and the drawing containing contours and suggestive contours.

(Incidentally, there isn't a perfect match, especially where the surface is twisting in weird ways, and there is ongoing research to characterize the exact conditions under which these two families of lines match.)

# Intensity Ridges in Images



So, in looking for more families of lines to complement contours and suggestive contours, it makes sense to look at the other flavor of image intensity extrema: ridges of illumination.

# Intensity Ridges in Images

Assume:

- Light at camera
- Lambertian material

What lines on the surface correspond to intensity ridges?

- Depends on how a ridge is defined  
(Saint-Venant, principal curvature extrema, ...)
- Exact answer very messy

Unfortunately, even if we set up the problem in a restricted way (headlight, diffuse shading), the answer turns out to be very “messy” mathematically.

## Intensity Ridges in Images

Instead look for maxima of  $n \cdot v$  along view-dependent directions  $w$  and  $w_{\perp}$

- Analog of intensity valleys in images as suggestive contours (minima of  $n \cdot v$  along  $w$ )

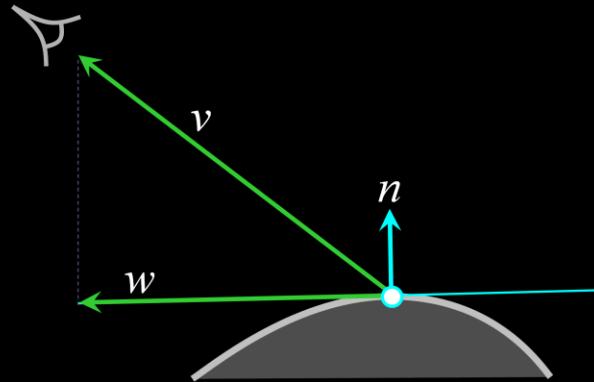
Appears to be a good approximation

Instead, we will look for simpler line definitions, which nevertheless qualitatively match intensity ridges well (just as suggestive contours qualitatively match intensity valleys). To do this, we will look at local maxima of  $n \cdot v$ , in the projected view direction  $w$  and its perpendicular. This corresponds to the definition of suggestive contours as local minima of  $n \cdot v$  in the direction  $w$ .

(Incidentally, we have also examined local *minima* of  $n \cdot v$  in the direction perpendicular to  $w$  – they don't appear to be especially interesting...)

## Surface Coordinates: $w$ and $w_{\perp}$

$w$  is the projected viewing direction



$$w_{\perp} = n \times w \quad (\text{comes out of the screen})$$

To review,  $w$  is just the projection of the view direction into the tangent plane of the surface.  $w_{\perp}$  is also in the tangent plane, and perpendicular to  $w$ . In the above drawing,  $w_{\perp}$  would be pointing towards you.

# Highlight Lines

## Suggestive highlights

- Maxima of  $n \cdot v$  along  $w$

## Principal highlights

- Maxima of  $n \cdot v$  along  $w_{\perp}$

## Lines are drawn in white

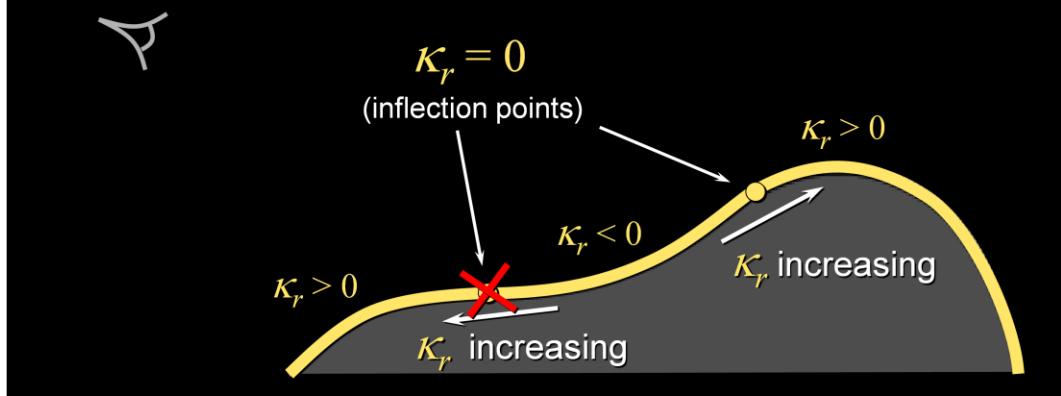
- In practice only draw strong maxima

Now we can define the two families of “highlight” lines corresponding to the above definitions. Suggestive highlights are local maxima of  $n$  dot  $v$  (corresponding to image intensity under a headlight) in the  $w$  direction, while principal highlights are the local maxima in the  $w_{\perp}$  direction.

The styles that we will examine draw principal highlights in white (as opposed to the contours and suggestive contours, which are drawn in black). We find that this makes it easier for the visual system to interpret them.

## Suggestive Highlights

Points where  $\kappa_r = 0$  and  $D_w \kappa_r < 0$



Let's look at suggestive highlights. These are just the inflections we threw away (with the derivative test) when finding suggestive contours.

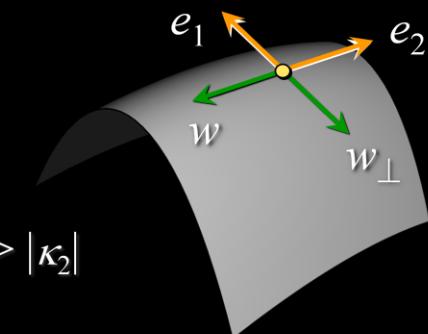
# Principal Highlights

(Strong) maxima of  $n \cdot v$  along  $w_\perp$

$$-D_{w_\perp} n \cdot v \sim \tau_r \quad (\text{radial torsion})$$

– Zeros of  $\tau_r$  occur where

$w$  and  $w_\perp$  are  
principal directions



$$|\kappa_1| > |\kappa_2|$$

Principal highlights are another matter entirely. If we go through the math, we find that just as the derivatives of  $n \cdot v$  in the  $w$  direction were related to the radial curvature, the derivatives in the  $w_\perp$  direction are related to another quantity called “radial torsion”. Intuitively, torsion represents the “twisting” of the normal direction as we move along the surface. Surfaces of zero radial torsion (corresponding to the maxima of  $n \cdot v$ ) are the ones that don’t exhibit this twist, in the view direction.

This turns out to happen precisely when the view direction is aligned with one of the principal directions.

## Principal Highlights

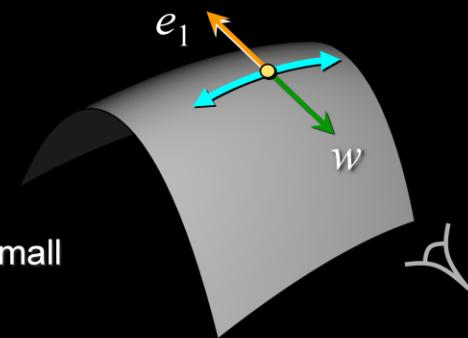
(Strong) maxima of  $n \cdot v$  along  $w_\perp$

$$-D_{w_\perp} n \cdot v \sim \tau_r \quad (\text{radial torsion})$$

– Zeros of  $\tau_r$  occur where

$w$  and  $w_\perp$  are  
principal directions

$$|D_{w_\perp} n \cdot v| \text{ small}$$



We now move to a test in the principal highlight definition designed to keep only the strong intensity maxima in the  $w_\perp$  direction.

The intuition is that when the view direction is aligned with the higher-curvature principal direction  $e_1$ , the surface is not curving very quickly in the perpendicular direction.

## Principal Highlights

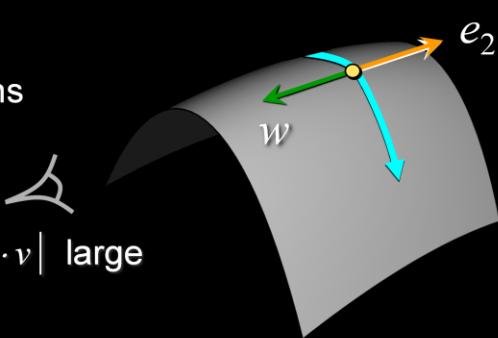
(Strong) maxima of  $n \cdot v$  along  $w_\perp$

$$-D_{w_\perp} n \cdot v \sim \tau_r \quad (\text{radial torsion})$$

– Zeros of  $\tau_r$  occur where

$w$  and  $w_\perp$  are  
principal directions

$$\left| D_{w_\perp} n \cdot v \right| \text{ large}$$

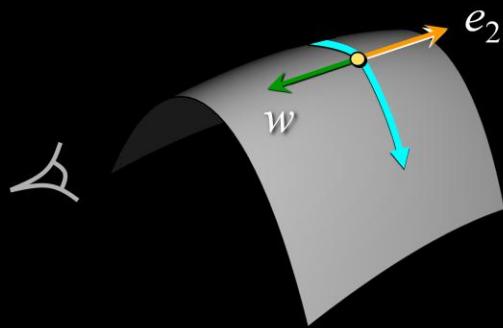


The opposite case, when we are looking along the ridge (and  $w$  is aligned with  $e_2$ , the weaker principal direction), leads to strong intensity maxima in the perpendicular direction, because that's the direction in which the normal is changing quickly.

# Principal Highlights

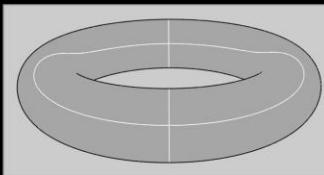
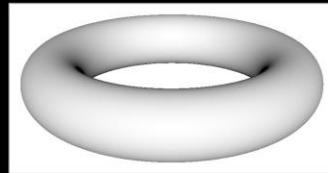
(Strong) maxima of  $n \cdot v$  along  $w_{\perp}$

- Just keep those where  $w = \pm e_2 \Rightarrow w \cdot e_1 = 0$
- Maxima when  $D_{w_{\perp}} \tau_r < 0$

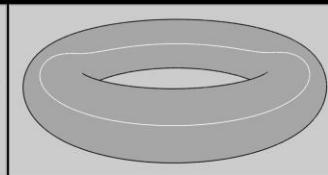


So, we keep the locations where  $w$  is along  $e_2$ , which means that it is perpendicular to  $e_1$ . So, instead of basing our definition for principal highlights on torsion, we adopt  $w \cdot e_1 = 0$  as the primary definition of principal highlights. As with suggestive highlights, there is a derivative test necessary to keep only local maxima, not minima.

## Principal Highlights

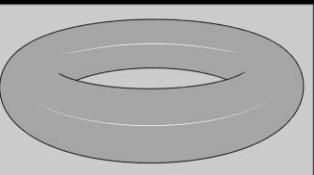


$$\tau_r = 0$$



$$w \cdot e_1 = 0$$

(potentially strong extrema)



$$w \cdot e_1 = 0$$

and

$$D_{w\perp} \tau_r < -\varepsilon < 0$$

(strong maxima)

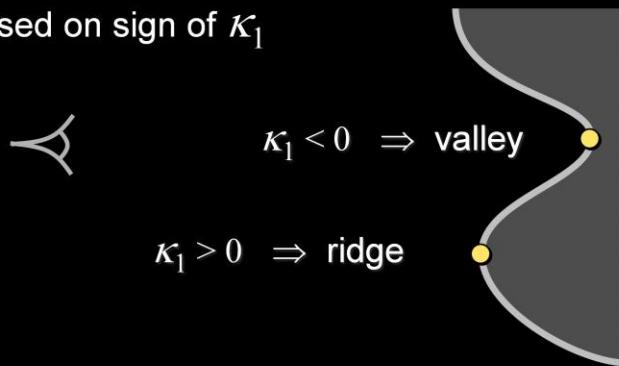
Here's what the different definitions look like for a simple model.

At left, we see all zeros of radial torsion. At center, we keep only those locations where  $w$  lines up with  $e_2$ , and at right we apply a derivative test (with a small but non-zero threshold).

## Principal Highlights

Points where  $w \cdot e_1 = 0$  and  $D_{w\perp} \tau_r < 0$

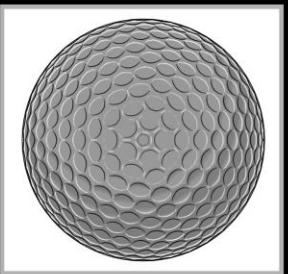
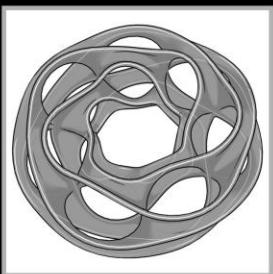
- equivalent to Saint-Venant creases in depth
- classify based on sign of  $\kappa_1$



It is possible to show that principal highlights correspond exactly to converting the depth map to a range image, then looking for extrema (illumination ridge and valley lines) according to a particular definition due to Saint-Venant. In other words, they are one possible view-dependent analog to the view-independent crest lines. (Another possible view-dependent analog is apparent ridges, which we will see later.)

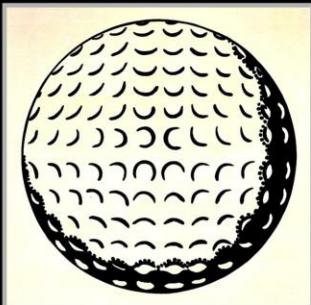
By looking at the sign of the first principal curvature, it is possible to classify these lines as ridge-like or valley-like, and use this additional information to stylize them differently or omit one or the other family.

## Results



Here are some examples with both suggestive contours and suggestive and principal highlights. (Drawn together with a gray background and subtle toon shading.)

## Comparisons

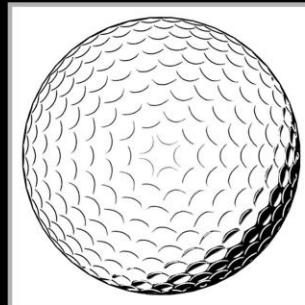


*Golf Ball* (1962)  
Roy Lichtenstein

© Estate of Roy Lichtenstein



SC + SH



SC + invert(SC)

Here is a slightly different style, where we still draw the lines in black and white, but draw the shape with a black/white toon shader (so that only suggestive contours are visible in the white regions, and only suggestive highlights are visible in the dark regions). This corresponds to the style of the Frank Miller comic we saw earlier, as well as this painting of a golf ball by Roy Lichtenstein.

Looking at this painting, we note that the direction of the half-round strokes in the dark region corresponds well with our suggestive-highlight rendering. In contrast, simply inverting the suggestive contours in the black toon region (as shown at right) gives lines that face the wrong way, and don't match the Lichtenstein painting any more.

## Apparent Ridges [Judd et al. 2007]

View-dependent variant of ridge and valley lines

Motivation: look for rapid screen-space normal variation

- Different from ridge and valley lines because of (view-dependent) foreshortening
- Includes occluding contours, and other lines that smoothly join to them
- Approach ridges and valleys when looking head-on

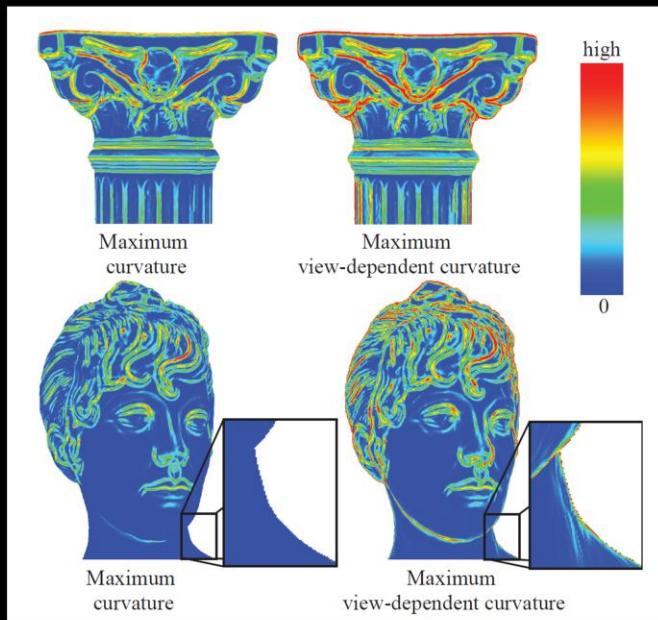


Matisse

There is one final recently-introduced family of lines that we will look at, namely apparent ridges. These were motivated by some drawings, such as this one by Matisse, that seem to combine ridge-like features with contours.

The approach of apparent ridges is to apply the standard ridge and valley line definition (local maxima of principal curvature, in the corresponding principal direction), but replace the use of standard surface curvature with a view-dependent quantity that takes foreshortening into account. Specifically, the curvatures in the projected view direction are divided by  $n \cdot v$ , to account for the fact that normals vary more rapidly with respect to screen-space location where the surface is tilted away from the viewer.

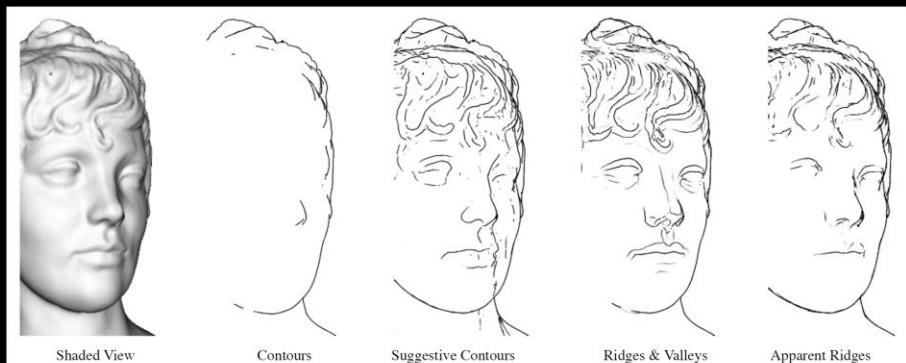
## View-Dependent Curvature



Here is a visualization of view-dependent curvature, as compared to standard curvature. It is obvious that view-dependent curvature grows quickly near occluding contours, leading to a strong tendency for the technique to place lines near those locations.

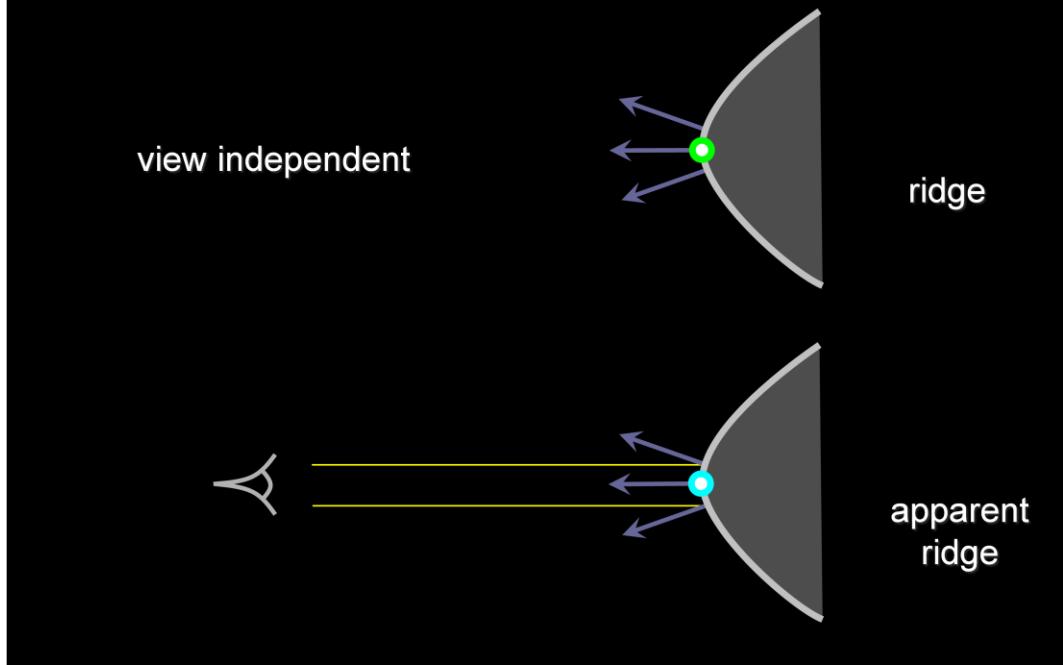
# Apparent Ridges

Local maxima of view-dependent curvature  
(in view-dependent principal direction)



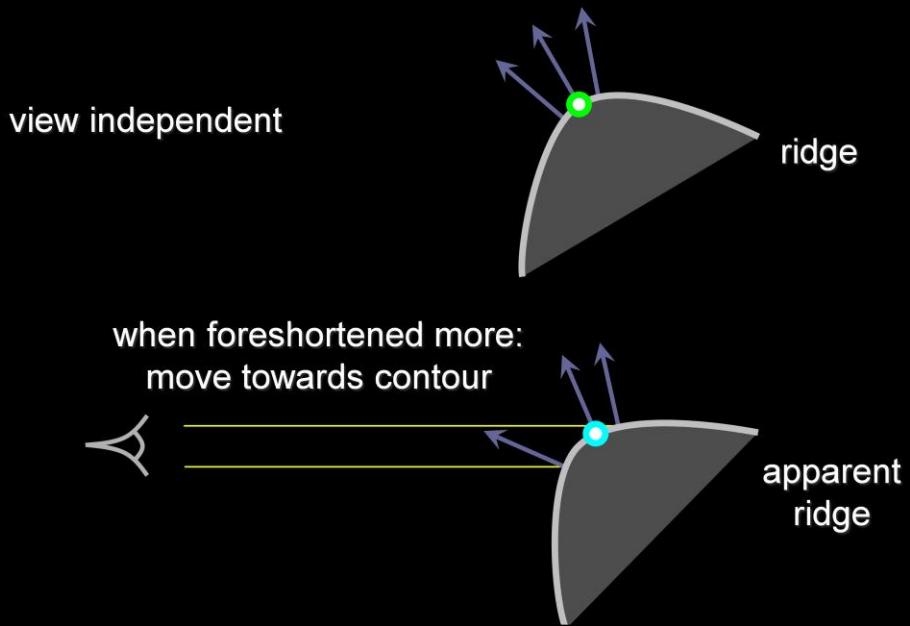
Here is a comparison of apparent ridges (right) to renderings with suggestive contours and view-independent ridge/valley lines.

## Apparent Ridges vs. Ridges



Looking at the qualitative behavior of apparent ridges, we see that they match standard ridges and valleys when viewed head-on.

## Apparent Ridges vs. Ridges



As the view becomes more oblique, they smoothly slide along the surface until they reach the contours. In fact, they connect up to the contours smoothly (just like suggestive contours).

# Lines Summary

Derivative Order	Image-Space	View-Independent Object-Space	View-Dependent Object-Space
0 <sup>th</sup>	Isophotes	Topo-lines	Cutting planes
1 <sup>st</sup>		Isophotes	Occluding contours
2 <sup>nd</sup>	Edges, extremal lines	Parabolic lines	Suggestive contours, suggestive highlights, principal highlights
3 <sup>rd</sup>		Crest lines (ridges and valleys)	Apparent ridges

In summary, we have seen three classes of mathematical line definitions: image-space, view-independent object-space, and view-dependent object-space. Although the jury's still out, it appears that the latter category is the most interesting because it yields shape-conveying lines, while being amenable to sophisticated NPR stylization algorithms.

This table classifies the lines we've seen according to the order of derivatives used in their definition. (Incidentally, we don't *think* there are especially interesting line definitions in the empty boxes, but we certainly could be wrong...)

# Part IV: Line Drawings and Perception

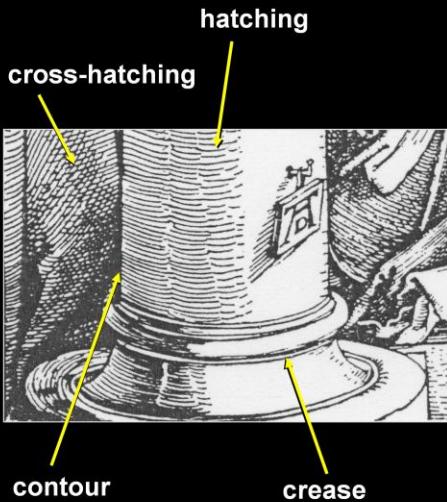
Doug DeCarlo

Line Drawings from 3D Models  
SIGGRAPH 2008

You've heard about the different types of lines that can appear in line drawings.

Now we're ready to talk about how people perceive line drawings.

# Line drawings



[Dürer 1505]

Line drawings bring together an abundance of lines to yield a depiction of a scene.

Take a look at this print by Dürer.

It uses different types of lines that convey geometry and shading in a way that's compatible with our visual perception.

We seem to interpret this scene easily and accurately.

\*\*\*

Some of the lines in this drawing only convey geometry.  
But the fullness of this drawing comes from Dürer's use of hatching and cross-hatching.

These patterns of lines convey shading through their local density and convey geometry through their direction.

## Line drawings



[Flaxman 1805]

Other drawings rely on little or no shading.

In this drawing by Flaxman,  
shading is limited to the cast shadows on the floor.

The detail in the cloth here is conveyed with lines  
such as contours, creases,  
and maybe other lines such as suggestive contours,  
or ridges and valleys.

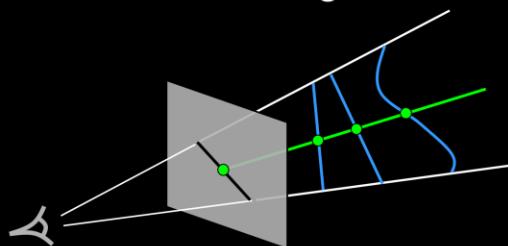
While artists can make drawings like this, they can't really explain  
what they're doing.

They rely on their training,  
and use their own perception  
to judge the effects of their decisions.

# Ambiguity of lines in images

The ambiguity of projection

- an infinity of 3D curves project to the same line in the image



It's actually a little surprising that line drawings are effective at all.

At first, line drawings just seem to be too ambiguous.

An infinite number of 3D curves  
can project to the same line in the image.

All images have this ambiguity, but in photographs,  
there are many other visual cues, such as shading and texture,  
that help to indicate shape.

Here, we're just looking at individual lines.

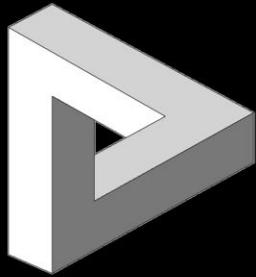
But it turns out that individual lines contain  
a wealth of information about shape.

This information is typically local in nature.

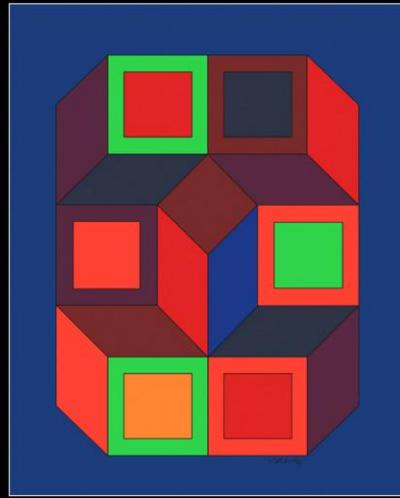
But our perception is somehow able to bring all of this together  
into a coherent whole.

Well, sort of.

# Impossible line drawings



The Penrose triangle [1958]



[Vasarely 1973]

Line drawings of impossible 3D objects show us that this coherence is NOT global.

The Penrose triangle, which was inspired by the work of Escher, is perhaps the simplest of the impossible figures.

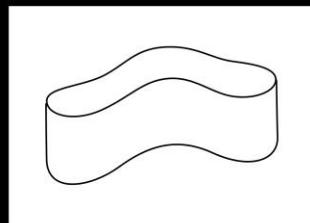
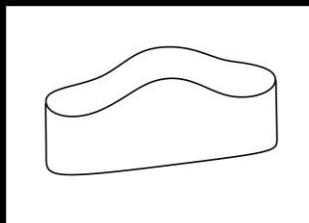
When you first look at it, it seems to be an ordinary object.

Closer inspection is a little unsettling, and its inconsistencies are easily revealed.

Vasarely pushed this idea even further, and made pictures such as this one that encourage us to explore several different inconsistent interpretations at the same time.

# Interactions between lines

- Line drawing interpretation depends on non-local context.



after [Barrow 1981]

Although you might think the Penrose triangle shows that there are no global effects for visual inference, it's not that easy.

Take a look at these two drawings.

The figure on the left appears to be raised in the center, while the figure on the right appears to have a flat top, and bends along its length.

If we compare these two drawings line by line, the only difference is the line along the bottom.

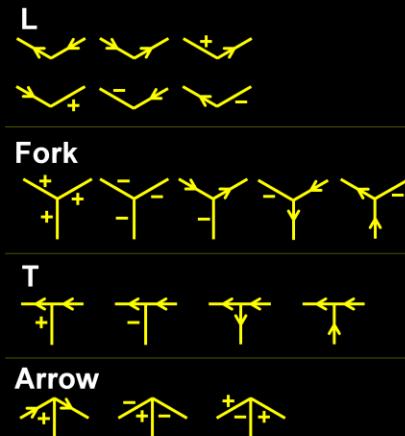
Nobody knows whether we see this difference because we consistently integrate local information, or perform certain types of non-local inference.

# Interpretation of line drawings

## Labeling polyhedral scenes

- catalog of junction labels
- find consistent labelings using constraint satisfaction

[Waltz 1975]



adapted from [Waltz 1975]

Use of non-local inference is plausible.

Algorithms exist for searching among the space of possibilities.

Waltz's method for line-labeling starts with catalogs of all possible line junctions, which are places where two or more lines meet.

Here's the catalog of 18 junctions that lets you classify any trihedral vertex in a polyhedral scene.

CONVEX lines are labeled with a PLUS,  
CONCAVE lines with a MINUS.

Arrows mark visual occlusions,  
where the closer surface is to the RIGHT of the arrow.

Algorithms for constraint satisfaction compute all possible configurations of junctions for a particular picture.

For an impossible figure, this set is empty.

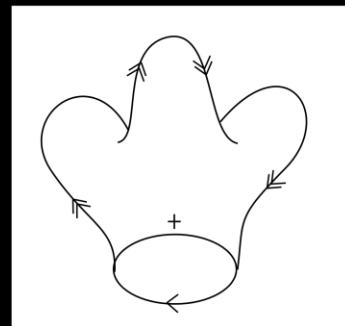
# Interpretation of line drawings

Labeling for more general scenes

- larger catalogs of junctions
- prune unreasonable or unlikely configurations

[Barrow 1981, Malik 1987]

after [Malik 1987]



This idea can be extended for line drawings that contain smooth surfaces.

First, you need a more comprehensive junction catalog.

Then, you need methods that can prune away large numbers of unreasonable interpretations, to prevent a combinatorial explosion.

These algorithms only label lines with a type.

They don't infer geometry.

Furthermore, existing algorithms are restricted to lines from contours and creases, and sometimes lines from shadows.

# Interpretation of line drawings

Unfortunately, we do not know how **humans** process line drawings.

Nevertheless, much is known about the **information** humans might be using.

While these algorithms suggest that exhaustive search might be a viable method for scene interpretation, they don't say anything directly about how PEOPLE interpret line drawings.

In fact, not very much is known about that.

Even so, we can still be very specific about what INFORMATION is available in a line drawing.

This is the information that our perceptual systems are probably using.

# Interpretation of line drawings

Each line **constraints** the depicted shape

- depending on the type of line

The type can sometimes be inferred from context (within the drawing)

Ambiguity always remains

- some interpretations are more likely than others

Essentially, each line in a drawing places a constraint on the depicted shape.

In the end, the geometry that results is never unique.

But our perceptual systems excel at uncovering the most reasonable and most likely interpretations.

So now let's go through the kinds of information that different types of lines provide.

# Information in line drawings

Lines can mark **fixed** locations on the shape

- creases (sharp folds)
- ridges and valleys
- surface markings (texture features, material boundaries, ...)
- hatching lines (although density is lighting-dependent)

Lines can mark **view-dependent** locations on the shape

- contours (external and internal silhouettes)
- suggestive contours
- apparent ridges

Lines can mark **lighting-dependent** locations on the shape

- isophotes (boundaries of attached shadows or in cartoon shading)
- edges (i.e. boundaries of cast shadows)

First, we'll consider lines that mark FIXED locations on a shape, such as creases, ridges and valleys, and surface markings.

Then, we'll consider VIEW DEPENDENT lines.

The most important is the CONTOUR, which lets us infer surprisingly rich information about the shape.

There are also lines whose locations are lighting-dependent, such as edges of shadows, but I'm not going to be discussing those.

Of all these lines, creases and contours are well understood.

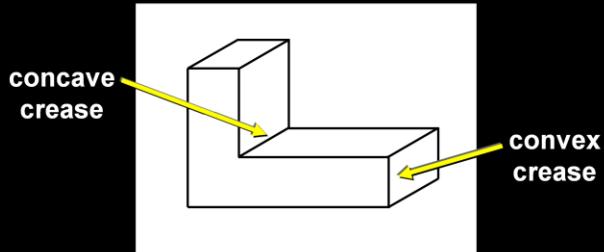
Research on the information other types of lines provide is ongoing.

# Information in creases

Creases mark discontinuities in surface orientation

– often a luminance discontinuity in shaded imagery

Convex vs. concave: cannot be determined locally



Creases mark discontinuities in surface orientation,  
and are typically visible in a REAL image  
as a discontinuity in tone.

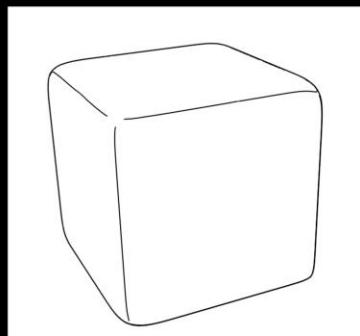
The crease can be concave or convex.

But local information doesn't let us determine which.

The algorithms for line labeling I mentioned earlier  
proceeded by considering every possibility,  
and then enforced consistency across the whole drawing.

# Information in ridges and valleys

- Mark locally rapid changes in surface orientation
- one possible extension of creases to smooth surfaces
  - like creases, locally indistinguishable



Ridges and valleys mark locally maximal changes in surface orientation.

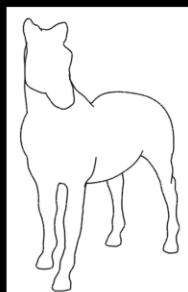
In real images, they can appear  
as smooth but sudden changes in tone.

The ridges on this rounded cube are particularly effective  
at conveying its shape, when drawn with the contours.

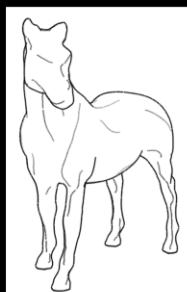
# Information in ridges and valleys

Their use is still unresolved

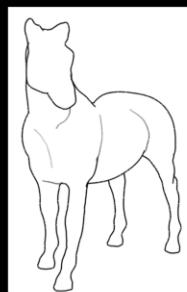
- many seem to convey shape
- others seem to convey surface markings
- humans can locate them in shaded imagery [Phillips 2003]



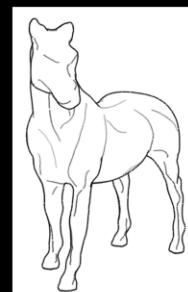
contours



with ridges



with valleys



with both

Research on the use of ridges and valleys in line drawings is ongoing.

When used alongside contours, ridges and valleys can produce an effective rendering of a shape.

The valleys on the side of the horse are quite convincing.

In other cases, they look like surface markings, such as the ridges on its head.

Ridges and valleys are reasonable candidates for line drawings, as there is psychological evidence that viewers can reliably locate them in realistic images.

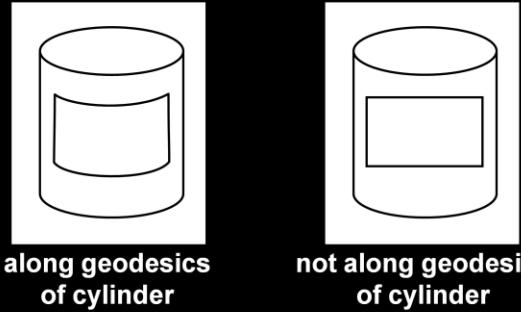
# Information in surface markings

Convey shape when they lie along *geodesics*  
(locally shortest paths on the surface)

[Stevens 1981, Knill 1992]

Related to perception of texture

[Knill 2001]



Markings on a surface can appear as arbitrary lines inside the shape.

However, for a certain type of line known as a geodesic, they can also convey shape.

Geodesics are simply lines on the surface that are locally shortest paths.

Stevens points out that for many fabricated objects, surface markings are commonly along geodesics.

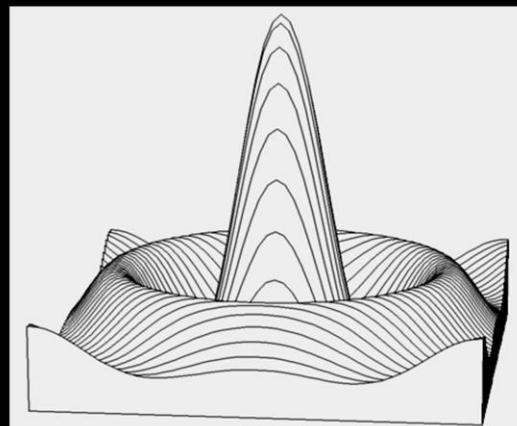
Take for instance the label on the cylinder on the left.

For a more general class of surfaces, Knill draws connections between texture patterns and sets of parallel geodesics.

# Information in surface markings

Parallel lines in space can also convey shape

[Stevens 1981]



When used in repeating patterns,  
other curves can be effective as well.

Sets of parallel lines,  
which are often used to construct plots of 3D functions,  
are one notable example.

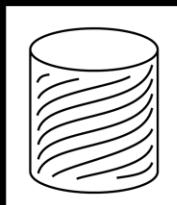
The images that result are analogous to using  
a periodic solid texture.

Stevens points out that all one needs to do  
to infer the shape is  
to build correspondences between adjacent lines,  
matching up points with equal tangent vectors.

# Information in hatching

Conveys shape through direction of hatching lines

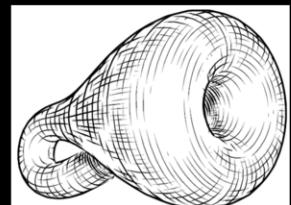
- more effective when drawn along geodesics  
[Stevens 1981, Knill 2001]
- lines of curvature are particularly effective  
[Girshick 2000, Hertzmann 2000]



along geodesics



along lines of curvature



[Hertzmann 2000]

The use of repeating patterns of lines forms the basis of hatching.

These lines convey shape in two different ways; they convey shape directly when they are drawn along geodesics.

And they convey shape indirectly through careful control of their density, which can be used to produce a gradation of tone across the surface.

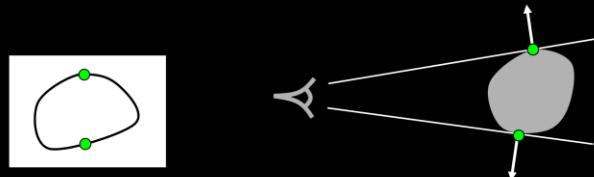
Particularly effective renderings are obtained when lines of curvatures are used, which align with the principal directions of the surface. These also happen to be geodesics.

So that's it for lines whose locations are FIXED on the shape.

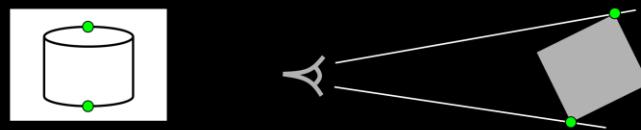
# Information in contours

Contours emerge in two situations:

1. Smooth parts of the shape where  $n \cdot v = 0$



2. Creases separating front- and back-faces



Next are lines whose location depends on the viewpoint.

The contour is the most notable example.

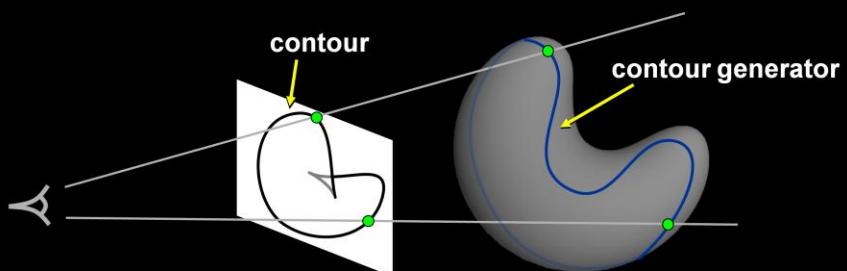
There are two situations when contours are formed.

On a smooth surface, contours are produced when the surface is viewed edge-on.

On an arbitrary surface, contours can also appear along a crease.

# Information in contours

The *contour generator* is the curve sitting on the surface that projects to the occluding contour



In either case,  
sitting on the surface  
is a 3D curve known as the **CONTOUR GENERATOR**.

This curve marks all local changes in visibility across the shape.

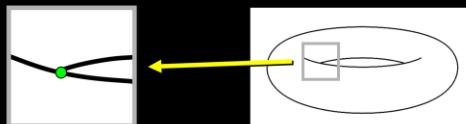
For a typical viewpoint, the contour generator  
consists of a set of isolated loops.

It projects into the image to become the contour.  
So not all parts of the contour are visible.

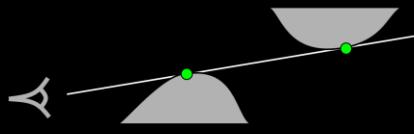
# Visibility of contours

Not all parts of the contour generator are visible

- First case: “non-local” occlusions
  - T-junctions



- viewing direction grazes surface at two locations



Let's consider the different cases of visibility for contours.

On a smooth surface,  
the first case is when one part of the shape  
occludes another more distant part.

This appears in the image as a T-junction,  
where the contour goes behind another part of the shape.

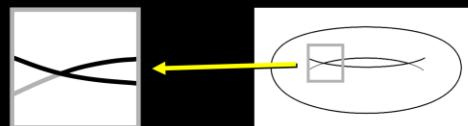
\*\*\*\*

At the location where the visibility changes,  
the visual ray is tangent to the surface in two places.

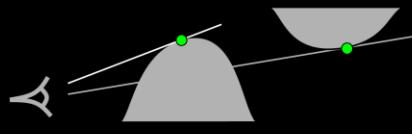
# Visibility of contours

Not all parts of the contour generator are visible

- First case: “non-local” occlusions
  - The contour continues in back



- and is simply occluded



The contour then continues behind the shape,

\*\*\*\*

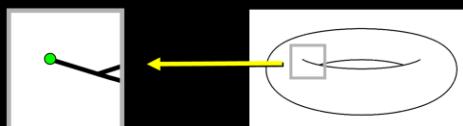
and is occluded.

This can be seen in this transparent line-drawing of a torus.

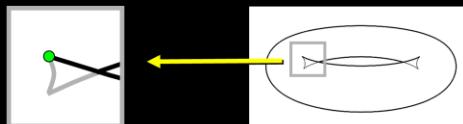
# Visibility of contours

Not all parts of the contour generator are visible

- Second case: ending contours
  - the visible part of the contour ends



- at a cusp in the projection of the contour generator



The second case occurs  
where the contour comes to an end in the image.  
An ending contour.

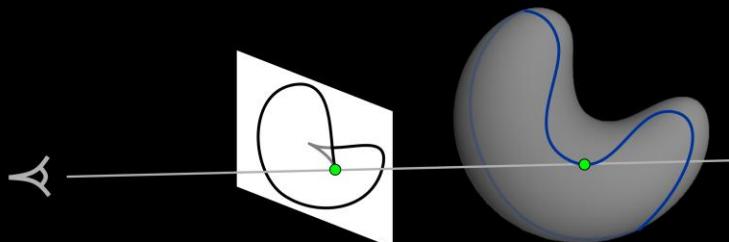
\*\*\*\*

When the occluded part of the contour continues,  
it does so at a cusp in the contour.

# Visibility of contours

Not all parts of the contour generator are visible

- Second case: ending contours
  - the tangent of the contour generator lines up with the viewing direction

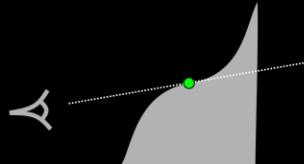


This cusp occurs because the contour generator lines up with the viewing direction, so that its tangent projects to a point.

# Visibility of contours

Not all parts of the contour generator are visible

- Second case: ending contours
  - the radial curvature is zero

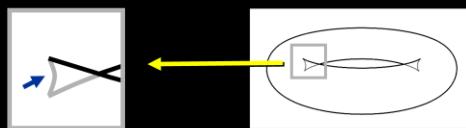


At an ending contour, the radial curvature is zero, which means that we're looking along an inflection; an asymptotic direction of the surface.

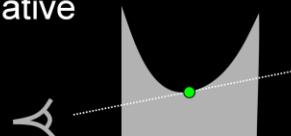
# Visibility of contours

Not all parts of the contour generator are visible

- Third case: local occlusions (“inside contours”)
  - these parts of the contour generator are never visible; the surface always blocks them



– the radial curvature is negative



The last case is a local occlusion;  
places where the surface has no choice but to occlude itself.

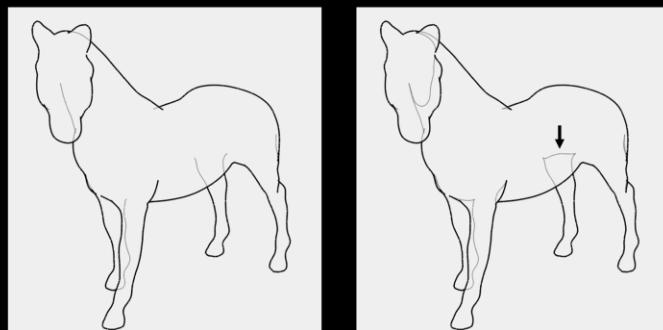
\*\*\*\*

These are locations where the radial curvature is negative.

# Visibility of contours

Not all parts of the contour generator are visible

- Third case: local occlusions (“inside contours”)
  - can be confusing in transparent renderings



In transparent renderings of contours,  
one typically does not draw the local occlusions,  
as the results can be confusing.

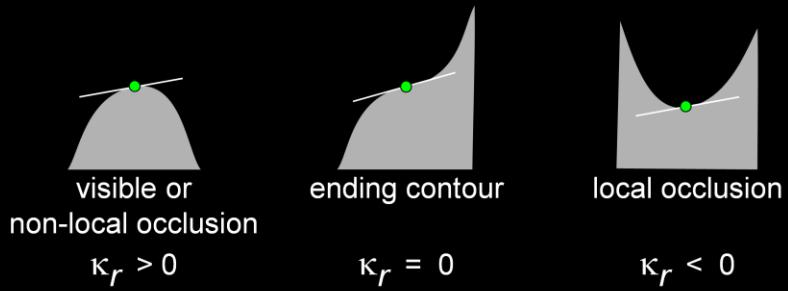
The image on the right here draws these contours.

One is marked with an arrow.

These curves actually correspond to regular contours  
for an inside-out version of the surface.

# Visibility of contours

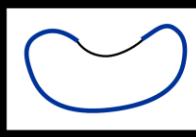
The three cases (on smooth surfaces):



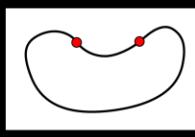
Here are the three cases, all together.

# Apparent curvature of contours

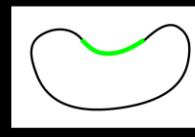
The apparent curvature  $\kappa_{app}$  is the curvature of the contour in the drawing (or image)



$$\kappa_{app} > 0$$



$$\kappa_{app} = 0$$



$$\kappa_{app} < 0$$

At the cusp of an ending contour,  $\kappa_{app}$  is infinite

Now, let's consider what the contours look like in the image.

The apparent curvature is simply the curvature of the contour in the drawing.

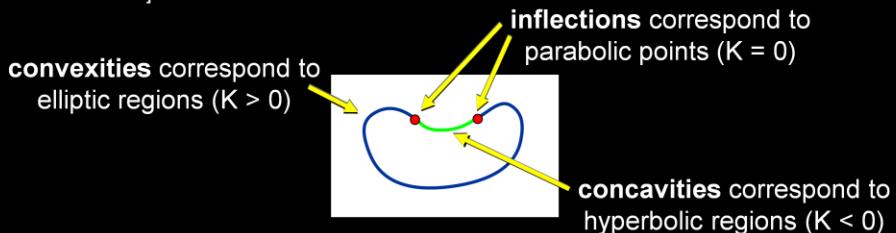
The convex parts of the contour have positive apparent curvature,  
the concave parts have negative apparent curvature,  
and it's zero at the inflections.

At the ending contours,  
the apparent curvature is infinite due to the cusp.

# Apparent curvature of contours

At a point on the contour (of a smooth surface):

- sign of apparent curvature  $\kappa_{app}$  = sign of Gaussian curvature  $K$   
[Koenderink 1984]



- Koenderink proves  $K = \kappa_r \frac{\kappa_{app}}{d}$ 
  - $d$  is the distance to the camera
  - $\kappa_r \geq 0$  for visible points on the contour

Koenderink proved a surprising and important relationship between the apparent curvature and the Gaussian curvature.

Specifically, for visible parts of the contour on a smooth surface, they have the same sign.

This means we can infer the sign of the Gaussian curvature simply by looking at the contour.

\*\*\*\*

CONVEX parts of the contour correspond to locations where the Gaussian curvature is POSITIVE: elliptic regions.

\*\*\*\*

INFLECTIONS on the contour correspond to locations where the Gaussian curvature is ZERO.

\*\*\*\*

CONCAVE parts of the contour correspond to locations where the Gaussian curvature is NEGATIVE: saddle-shaped regions.

\*\*\*\*

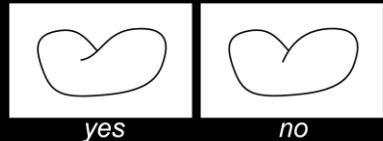
Koenderink gives a formula that connects these two quantities, that involves the distance to the camera and the radial curvature.

# Apparent curvature of contours

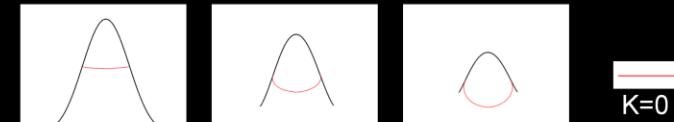
- Contours must end in a concave way

[Koenderink 1982]

- they only occur where  $K < 0$



- However, the concave ending might be hard to see



a Gaussian bump viewed from the side towards the top

A related result is that since ending contours only occur where the Gaussian curvature is negative, the contours must end in a concave way, approaching their end with negative apparent curvature.

\*\*\*\*

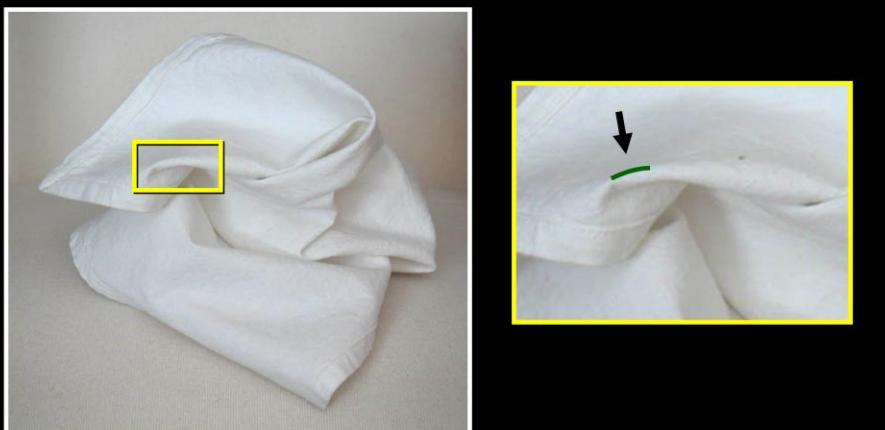
But Koenderink and van Doorn also noticed that artists tend to draw lines that are missing these concave endings.

It turns out this concave ending can be difficult to discern, as is the case for this Gaussian bump.

DEMO

# Ending contours in images

Difficult to localize



Contours are typically easy to detect in real images, at least when the lighting is right.

And there are many studies that demonstrate how people use them for visual inference.

However, in many cases, it's not easy to determine where a contour ends.

Here's an example photograph of napkin.

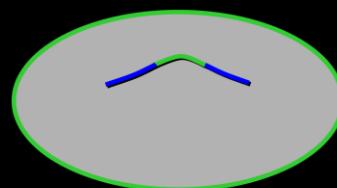
Even if we zoom in, it's still not clear whether the surface occludes itself or whether it's simply heavily foreshortened.

Observations like this make sense of line types that extend ending contours: suggestive contours and apparent ridges.

# Information in suggestive contours

Suggestive contours line up  
with ending contours

[DeCarlo 2003]



Suggestive contours are another type of line to draw, and whether they are in fact detected and represented by our perceptual processes is still an open question.

They do seem to produce convincing renderings of shape in many cases.

The fact that suggestive contours smoothly line up with contours in the image is encouraging.

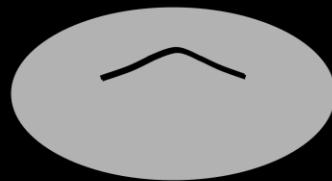
\*\*\*\*

In fact, if the lines aren't color coded, it's difficult to tell where one starts and the other ends.

# Information in suggestive contours

Suggestive contours line up  
with ending contours

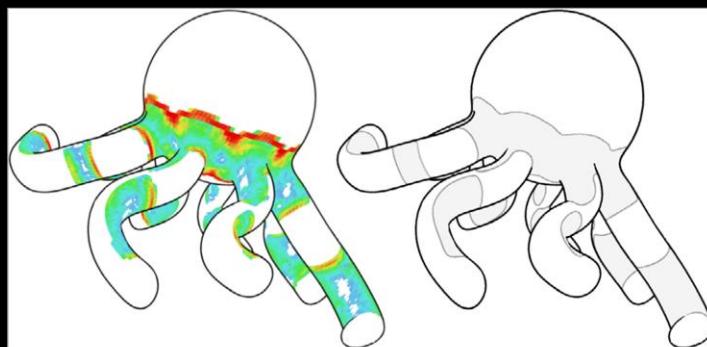
[DeCarlo 2003]



This makes it hard to tell where contour ends

# Information in suggestive contours

Can only appear in hyperbolic regions ( $K < 0$ )



Suggestive contour  
density over all views

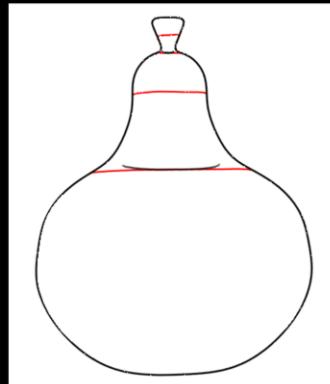
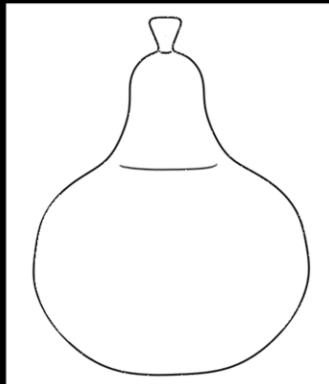
Regions where  
 $K < 0$

We can say something about what information they provide.

Recall from earlier how suggestive contours  
can only appear  
where the Gaussian curvature is negative.

# Information in suggestive contours

Away from the contour, they approach  
parabolic lines ( $K = 0$ ) [DeCarlo 2004]



In many cases, the suggestive contours approach the parabolic lines away from the contour.

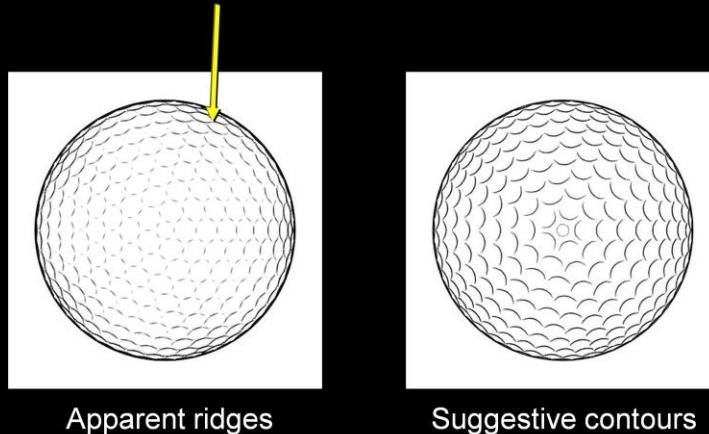
On this pear, we see how the suggestive contour skims along the parabolic line.

DEMO

We hope to be able to say more about this in the future.

# Information in apparent ridges

Near the contour, approach suggestive contours



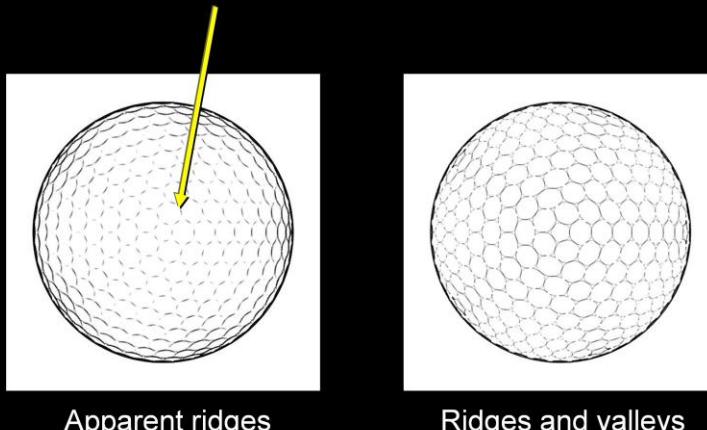
We can make similar statements about apparent ridges.

Near the contour, apparent ridges behave like suggestive contours.

They extend ending contours.

# Information in apparent ridges

Where front-facing, approach ridges and valleys



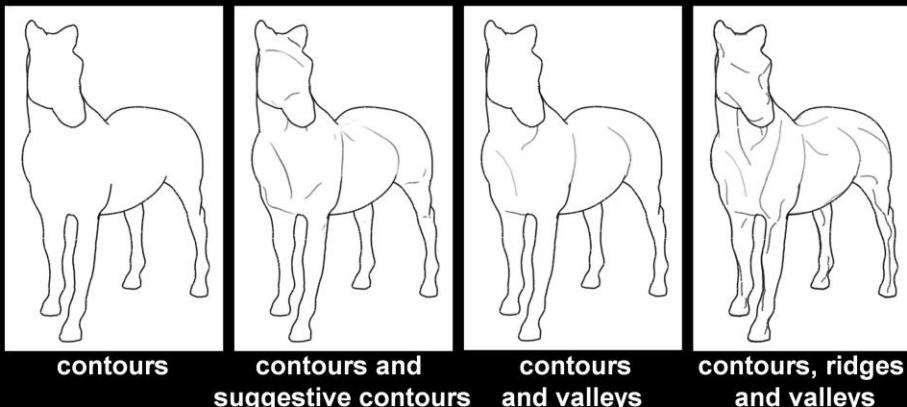
As the surface faces more towards the viewer,  
the location of apparent ridges  
approaches ordinary ridges and valleys.

And of course, in both of these cases,  
apparent ridges are surface locations  
where the normal vector is changing maximally.

# Interior shape features

What lines to draw?

– still not resolved, really



We can compare renderings with ridges and valleys to renderings with suggestive contours.

On the horse from this viewpoint, the rendering with just valleys is actually quite convincing.

As noted earlier, many of the ridges appear as surface markings here.

For the valley rendering, some features are missing, but the more salient features on the side of the horse are depicted.

Note the slight differences between the lines from suggestive contours, and from valleys.

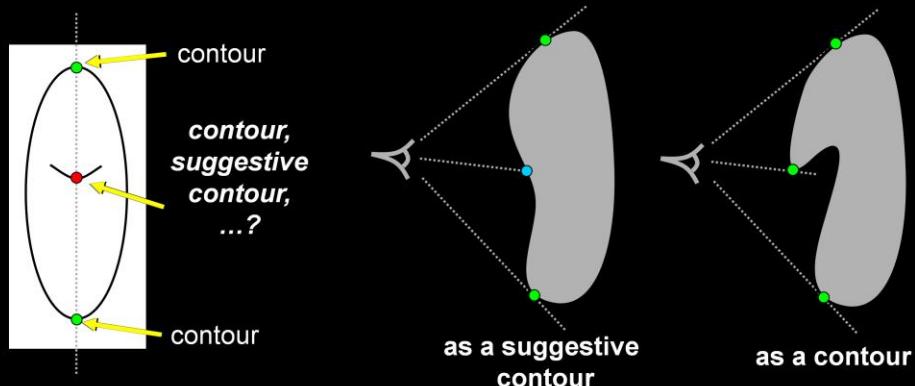
The shapes they convey appear to be a little different.

Clearly there is a lot of interesting work to do here.

This concludes our discussion of what information particular lines provide.

# Line labeling ambiguity

Different assignments of line labels correspond to different surfaces



Of course, this information can only be used if we know the TYPES of the lines when we're given a drawing.

Earlier we discussed algorithms for line drawing interpretation; approaches like this are reasonable to consider for this purpose.

But even if we do use these algorithms, there are often several different labelings that are consistent.

Given the line drawing on the left which depicts an elliptical shape with a bump, we can successfully label the green points as contours.

The red point, however, can be either a contour or suggestive contour.

Two possible shapes that match these labelings are shown on the right.

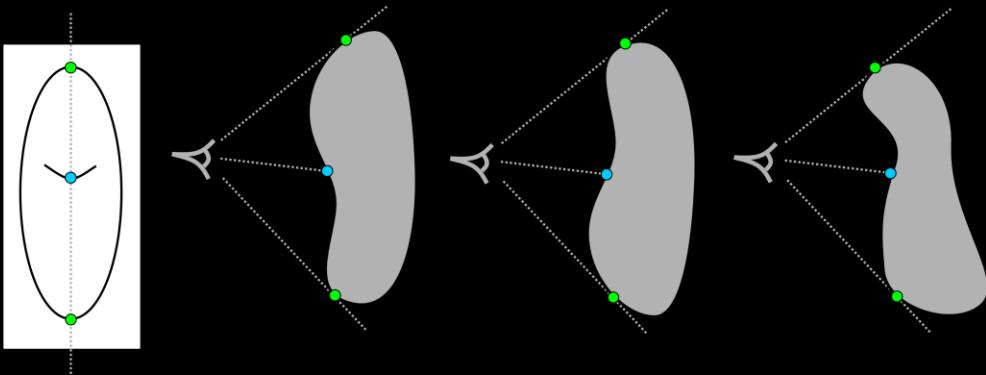
Presumably this problem cannot be solved in general.

There will always be ambiguity.

It's possible that when artists make line drawings, they're careful to shape the remaining ambiguity so it won't be a distraction.

# Projective ambiguity

Even given a line labeling, an infinity of shapes correspond to that drawing



And even with a line labeling,  
there is the ambiguity of projection.

These three interpretations have the same line labeling,  
but different geometries.

At first, this seems hopeless.

Yet sketching interfaces like Igarashi's Teddy  
seem to be quite successful by using inflation.

How can this be?

Well, there are reasonable constraints on smoothness  
that we can expect of the underlying shape.

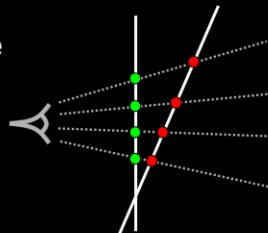
We also presume  
that the artist has drawn all of the important lines,  
so that no extra wiggles remain.

These issues are the source of one crucial challenge  
for sketch-based shape modeling.

# Bas-relief ambiguity

The projective ambiguity that preserves planarity

- ambiguity in Lambertian shaded imagery  
[Belhumeur 1999]
- preserves contours, shadow boundaries,  
(relative) signs of curvature
- has perceptual significance  
[Koenderink 2001]



We can be more specific with regard to this ambiguity.

For real images, there are well defined ambiguities for particular types of imagery.

One notable example is the ambiguity that remains when viewing a shape under Lambertian illumination.

There is a group of shape distortions that can be applied to a shape, that with an corresponding transformation of the lighting positions, approximately produce the same image.

This is the three-dimensional projective mapping known as the generalized bas-relief transformation.

As shown here, it moves points along visual rays and preserves planes.

It also preserves contours, boundaries of shadows, and the relative signs of curvature on the shape.

Perhaps most interestingly is that when you ask people to describe the shapes they see in shaded imagery, they answer consistently modulo this ambiguity transformation.

# Evaluation of line drawings

When is an automatic line drawing effective?

1. compare it to drawings by skilled artists
2. psychophysical measurement
  - perceived shape should be consistent with the original (modulo ambiguity)
  - bas-relief ambiguity may be appropriate [Koenderink et al. 1996, Belhumeur et al. 1997]
  - such methods may yield global inconsistencies [Li and Pizlo 2006]

So how can we be sure that a line drawing we make is perceived accurately?

As you saw earlier, one possible path is to compare that line drawing to those made by skilled artists.

Another way, based in psychophysics, is to simply ask the viewer questions about the shape they see. If this is done right, you can reconstruct their percept and compare it to the original shape, given the appropriate ambiguity transformation.

Koenderink and colleagues already performed a study like this on a single line drawing.

Their results suggest that the bas-relief ambiguity might be the appropriate one to consider here.

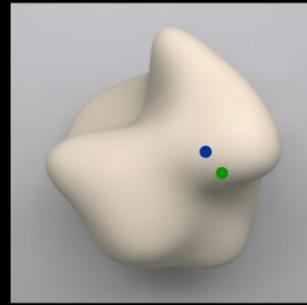
However, this ambiguity may only be resolved locally, where different parts of the shape are locally consistent, but not necessarily in a global sense.

# Psychophysical studies

## Measurement of perceived shape

[Koenderink 2001]

- **depth probing**
  - asks the viewer which point is closer  
(green or blue)



So what kinds of questions can you ask viewers?

In psychophysics, the answer is:  
very simple ones, and lots of them.

Koenderink describes a set of psychophysical methods  
for obtaining information  
about what shape a viewer perceives.

The first they describe is called RELATIVE DEPTH PROBING.

The viewer is shown a display like this one,  
and is simply asked which point appears to be closer.

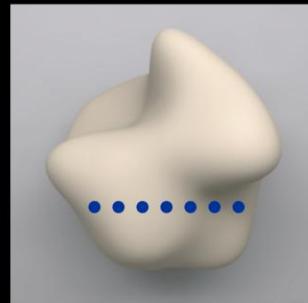
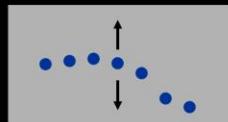
They are asked this question for many pairs of points.

# Psychophysical studies

## Measurement of perceived shape

[Koenderink 2001]

- depth profile adjustment
  - viewer adjusts points until they match the profile of a particular cross-section



Another method is known as DEPTH PROFILE ADJUSTMENT.

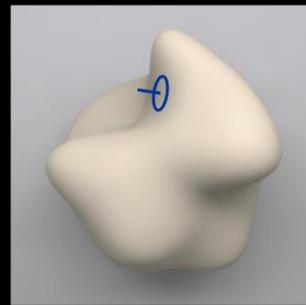
Here, the viewer adjusts points  
to match the profile  
of a particular marked cross-section on the display.

# Psychophysical studies

## Measurement of perceived shape

[Koenderink 2001]

- gauge figure adjustment
  - viewer adjusts a disk until it appears to sit on the tangent plane of the surface



Their third method is known as GAUGE FIGURE ADJUSTMENT.

Here, the viewer uses a trackball  
to adjust a small figure that resembles a thumbtack,  
so that it looks like its sitting on the surface.

All of these methods are successful.

But gauge figure adjustment seems to give  
the best information given a fixed number of questions.

# Summary

- Specific information is conveyed by each type of line
- Psychophysical evidence suggests humans use this information
- How exactly humans interpret line drawings is still unknown

So in summary,

Each type of line in a line drawing conveys specific information about shape.

There is a fair amount of evidence that people use this information.

But how exactly people use this information is still unknown.

We're very encouraged by how a combination of computer graphics and psychophysics can lead to answers to these questions.

# Part V: Algorithms for Extracting Lines

Szymon Rusinkiewicz

Line Drawings from 3D Models  
SIGGRAPH 2008

# Classes of Algorithms

## Image-space:

- Render some scalar field, perform signal processing (thresholding, edge detection, etc.)

## Object-space:

- Extract lines directly on surface

## Other:

- Alternative representations (e.g. geometry images)
- Also, some graphics hardware tricks

Based on [Isenberg 2003]

There are two major classes of algorithms for extracting most kinds of lines from 3D meshes. First, there are image-space algorithms that render something (such as a depth map or cosine-shaded model), then extract lines by doing some sort of image processing on the framebuffer (for simple operations such as thresholding, there are often ways of achieving the same effect using texture mapping, or vertex or pixel shaders). The advantage of this kind of algorithm is that it can be fast, easy to implement, and provides some notion of view-dependent level of detail. A major disadvantage is that it makes it difficult to control the appearance and stylization of the resulting lines.

A second class of algorithm operates in object space – on the model directly. These algorithms tend to be a little more complex, and it is more difficult to adapt them to take advantage of graphics hardware. On the other hand, they provide good control over stylization.

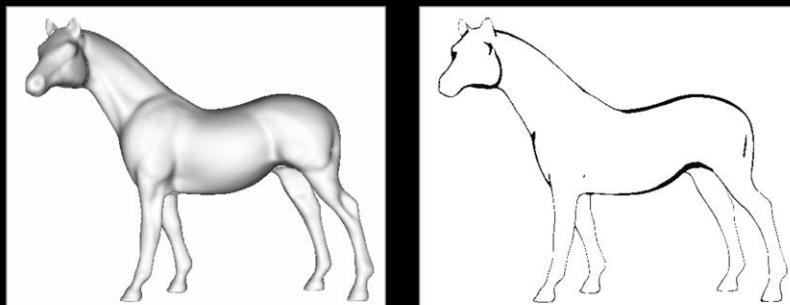
Finally, there are hybrid (usually multi-pass) algorithms, which perform a bit of processing in object space, but the lines ultimately show up only in the frame buffer. These are much less general than the other kinds of algorithms, and are specialized for e.g. contours.

# Contours: Image-Space Algorithm

Recall: occluding contours = zeros of  $n \cdot v$

Simple algorithm: render  $n \cdot v$  as color, apply threshold

- Variant: index into texture based on  $n \cdot v$
- More variants: environment map, pixel shader

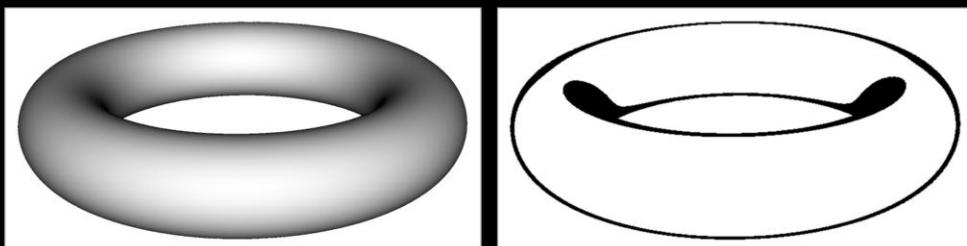


Let's start with occluding contours (or interior and exterior silhouettes), and look at image-space algorithms. A very simple technique is to render a lit version of the model (without color), then perform a thresholding step: any region darker than a threshold is set to black (or the line color), and anything above the threshold is set to the background color. There are many ways to do this thresholding step as part of the rendering, using pixel shaders, texture mapping, environment mapping, etc.

## Line Thickness

Drawback: line thickness varies

- Thicker lines in low-curvature regions

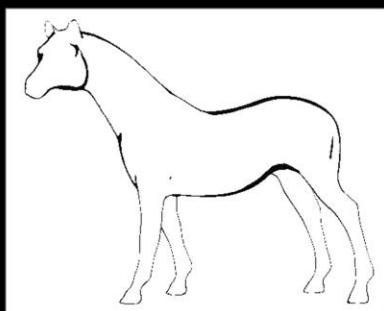


One major problem with this algorithm is that the thickness of the lines can vary, sometimes quite a bit. There are a few tricks to get around this problem.

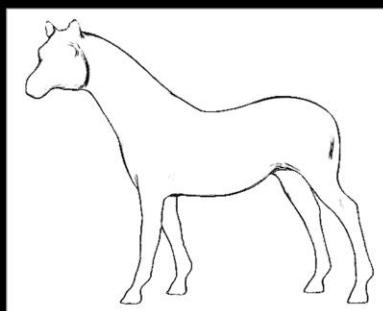
# Line Thickness Control

Solution #1: mipmap trick

- Load same-width line into each mipmap level



Original



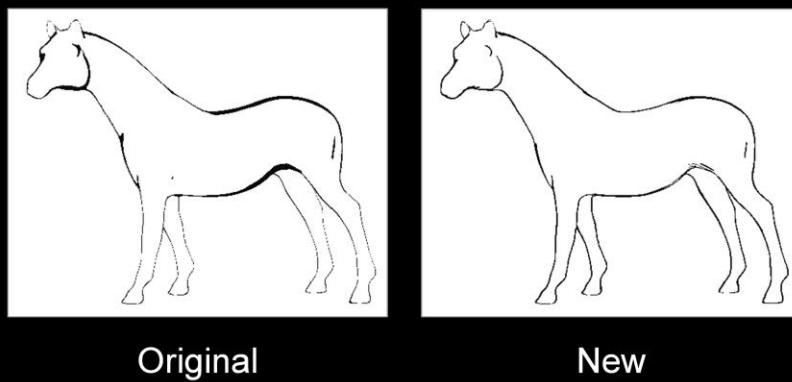
New

First, you can use a texture map indexed by  $n \cdot v$ , but use mipmapping. The trick is to make the width of the black region in the texture map the same width in all mipmap levels.

# Line Thickness Control

Solution #2: curvature-dependent threshold

$$- \text{Test } n \cdot v < \varepsilon \sqrt{\kappa_r}$$

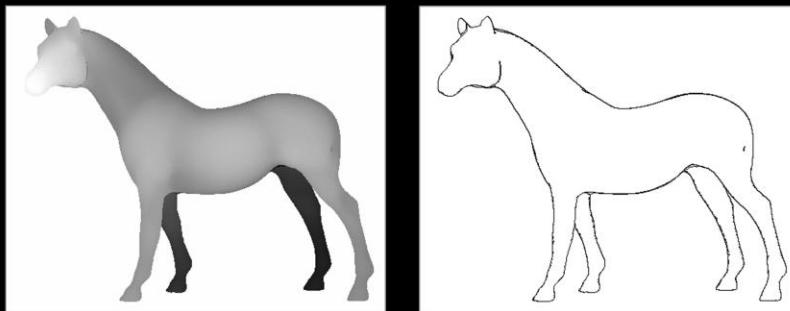


Another solution is to take advantage of the fact that, for a constant-curvature region, you can determine how thick the lines will be as a function of radial curvature. Then, the approximation is to change the threshold depending on the square root of radial curvature.

## Contours: 2<sup>nd</sup> Image-Space Algorithm

Render depth image, find edges

- Simpler rendering: no normals
- More complex image processing:  
edge detector vs. thresholding



There's a second, completely different, image-space algorithm that's possible. Now, instead of rendering  $n \cdot v$ , we render a color that depends on depth (or just look at the depth buffer instead of the color buffer). The image processing operation we have to do here is more complicated: edge detection instead of just thresholding. This is an interesting tradeoff: we have made the rendering simpler, but the image processing more complex.

## Contours: Object-Space Algorithm

Main advantage over image-based algorithms:  
can explicitly stylize lines

Algorithm depends on definition used:  
edges between front/back-facing triangles vs.  
zeros of interpolated  $n \cdot v$

Let's now move to object-space algorithms for contour extraction. Recall that we talked about two possible definitions of contours on polygonal meshes: contours along the mesh edges (separating front-facing and back-facing faces), or contours within faces (zeros of interpolated  $n \cdot v$ ). For the first definition, a simple brute-force algorithm is just to loop over all edges, and check whether each has one adjacent frontface and one adjacent backface.

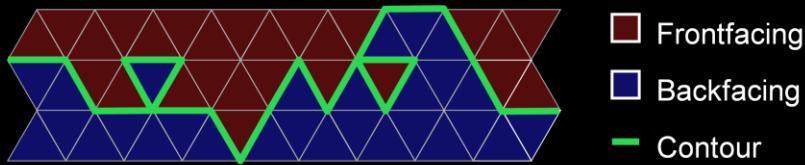
# Contours: Object-Space Algorithm

For first definition: **loop over all edges**

- Test adjacent faces
- If one frontfacing, one backfacing, draw edge
- Can be done in hardware [McGuire 2004]

Disadvantage: can get self-intersecting paths

- Makes stylization difficult



For the first definition, a simple brute-force algorithm is just to loop over all edges, and check whether each has one adjacent frontface and one adjacent backface.

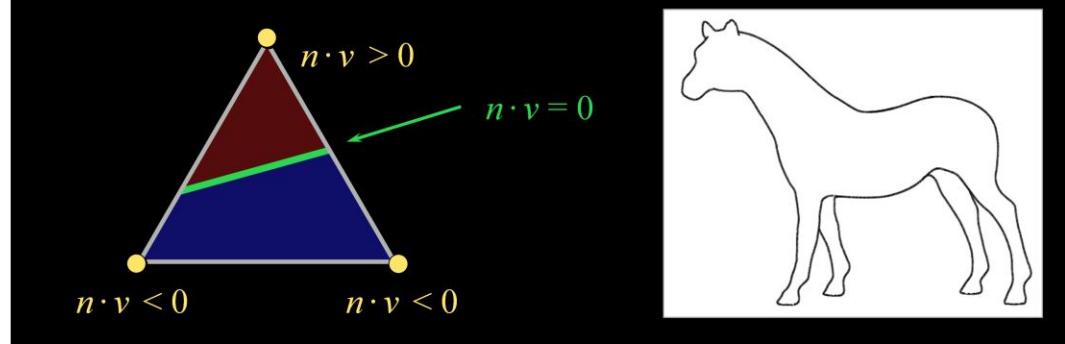
This has the disadvantage that the contour, when viewed as a path along mesh edges, can form loops.

# Contours: Object-Space Algorithm

[Hertzmann 2000]

Second definition: within-face lines

- For each vertex: compute  $n \cdot v$
- For each face: if signs not the same, interpolate to find zero crossing within face



The other definition involves computing  $n \cdot v$  at each vertex, then looping through all faces of the mesh. For each face, you first ask whether  $n \cdot v$  has a different sign at some vertex. If so, you interpolate along edges connecting positive- $(n \cdot v)$  vertices and negative- $(n \cdot v)$  vertices to find zeros, then connect the two points with a segment.

# Acceleration Techniques

Goal: avoid touching all vertices, faces, or edges

Normal cone bounding hierarchy [Sander 2000]

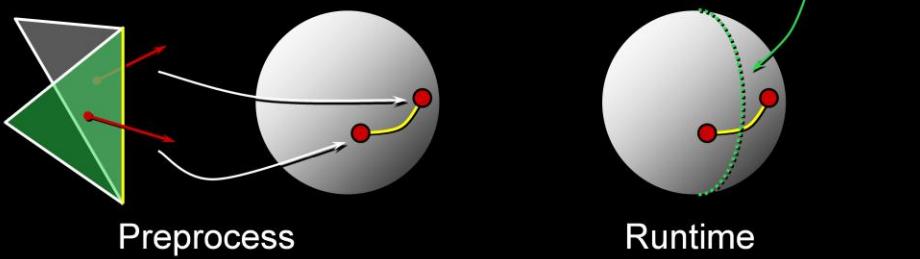
- Nodes contain cone of normals of children
- Leaf node for each face
- Traverse tree checking view direction against cone

Both of these object-space algorithms are brute-force: they require looping over all the edges, vertices, and/or faces of the model. There is a large body of work on acceleration techniques that try to reduce running time. For the contours-within-faces case, one popular technique is to construct a hierarchical data structure, where each node stores a bounding cone of the normals below it. At run time, the tree is traversed, and any nodes for which the cone is entirely frontfacing or entirely backfacing can be pruned.

# Acceleration Techniques

Gauss map [Gooch 1999, Hertzmann 2000]

- Discretize space of normals
- For each edge, find path between normals of faces
  - Store pointer to edge in normal “bucket”
- At run time, check edges corresponding to circle of normals



Another interesting acceleration technique involves the Gauss map. As a preprocess, a data structure is built that represents the space of possible directions (the space of directions conceptually corresponds to a sphere, but usually a cubemap is easiest to work with). For each edge, we compute an arc (shown in yellow) between the directions corresponding to the normals of the two faces touching that edge. Each direction intersected by this arc gets a pointer back to the edge. At run time, we check all directions corresponding to the normals perpendicular to the view: any arc that intersects that circle of directions (shown in green) represents an edge that is part of the silhouette (in practice, a superset of edges is generated because of the discretization of the Gauss map, so candidate edges must be verified).

# Acceleration Techniques

Randomized seed-and-traverse [Markosian 1997]

- Pick random edges
- If found contour, walk along it to extract whole loop
- Use contour edges from previous frame as seeds
- Not guaranteed to find all contours, but likely to have found all big ones after a few frames

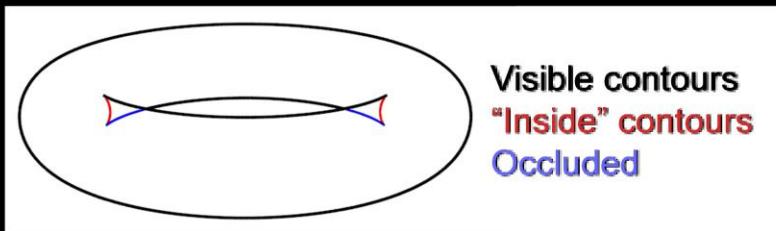
A very different sort of acceleration technique, most suited to interactive systems, relies on randomization. We pick random faces on the mesh, and check whether they contain a contour. If so, we follow the contour by walking to adjacent faces, eventually extracting an entire contour loop. In order to improve the efficiency of the random testing, we can test those faces that contained a contour in the previous frame before resorting to the random testing.

This algorithm, of course, is not guaranteed to find all the contours unless we test all faces. However, it is very likely that all significant contours will be found, and the reliance on temporal continuity means that it is very likely that after a few frames it will find everything.

# Visibility

Any object-space algorithm must handle visibility

- Local occlusions: “inside” contours (have  $\kappa_r < 0$ )
- Non-local occlusions



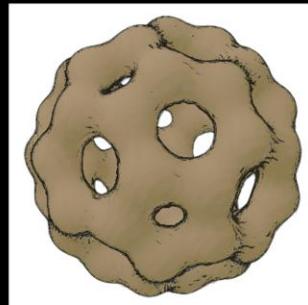
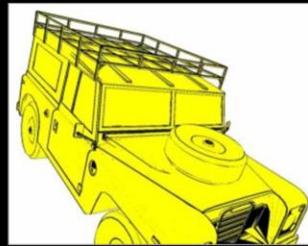
Regardless of the details, all object-space contour finding algorithms must deal with the problem of visibility. Although we'll look at some strategies for this later on, for now let us emphasize the fact that there are two ways in which a contour can be invisible: it can be occluded by a distant portion of the mesh, or it can be occluded locally. The latter pieces of the contour can be identified simply by checking the sign of the radial curvature, so at least part of the visibility problem can be solved locally. Full visibility is usually resolved using an algorithm such as ray tracing or z-buffering.

# Hybird Algorithm for Finding Contours

## Two-pass algorithm

- Draw frontfaces,  
offset towards viewer
- Draw enlarged backfaces  
[Raskar 1999, 2001]

- Does not need a mesh:  
also works for  
point/surfel clouds [Xu 2004]



Let's look at one algorithm in the "hybrid" category. Imagine doing a standard rendering pass, then keep the z-buffer on and render just the backfaces slightly enlarged (which can be done by actually changing the geometry, or by rendering the backfaces using thick outlines). Around the contours, the second rendering pass will "peek out" from behind the geometry rendered on the first pass. This is a nice algorithm because it can be very fast (modern graphics hardware can do it in one pass), and requires neither additional data structures nor image processing. However, just as with image-space algorithms, there is no control over stylization.

## Silhouettes in Geometry Images [Yuan 2005]

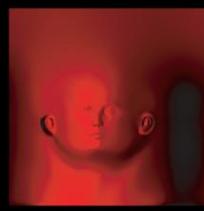
Work directly on parameterized geometry images



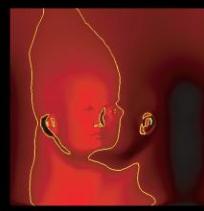
Mesh



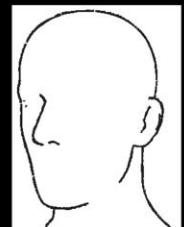
Geometry  
Image



$n \cdot v$  map

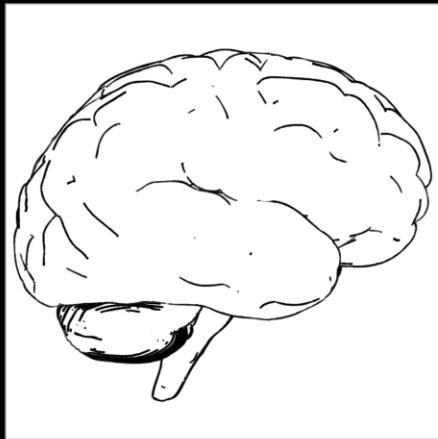


$n \cdot v = 0$

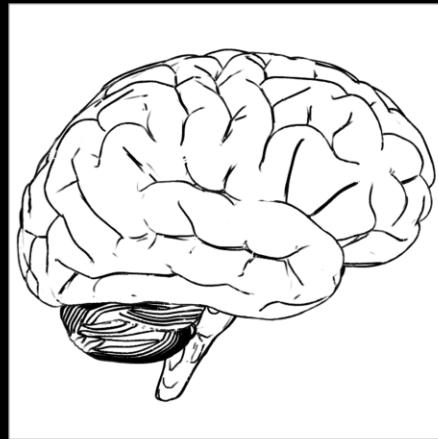


Drawing

## Moving on to Suggestive Contours...



contours



contours +  
suggestive contours

Here is your brain on contours. Here is your brain on suggestive contours. Any questions?

## Algorithms for Suggestive Contours

Definition 1: contours in nearby views

Definition 2: local minima of  $n \cdot v$

Definition 3: zeros of radial curvature

Let's move on to algorithms for suggestive contours. There are three different definition, and each gives rise to a different algorithm. The first definition, "contours in nearby views", is difficult to work with and requires a search over viewpoints.

## Algorithms for Suggestive Contours

Definition 1: contours in nearby views

→ search over viewpoints

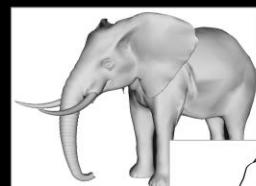
Let's move on to algorithms for suggestive contours. There are three different definition, and each gives rise to a different algorithm. The first definition, "contours in nearby views", is difficult to work with and requires a search over viewpoints.

# Algorithms for Suggestive Contours

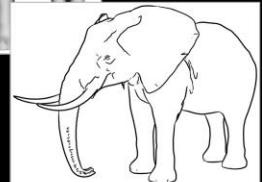
Definition 2: local minima of  $n \cdot v$

→ image processing to detect minima

Render diffuse-shaded image



Filter to detect valleys in intensity



The second definition, “local minima of  $n \cdot v$ ”, gives rise to an image-space algorithm in which an  $(n \cdot v)$ -shaded image is rendered, and a “valley detection” filter is used to detect valleys of intensity.

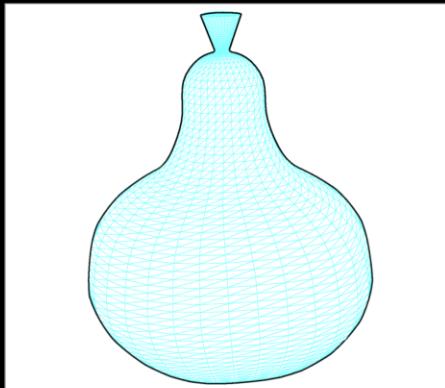
## Algorithms for Suggestive Contours

Definition 3: zeros of radial curvature

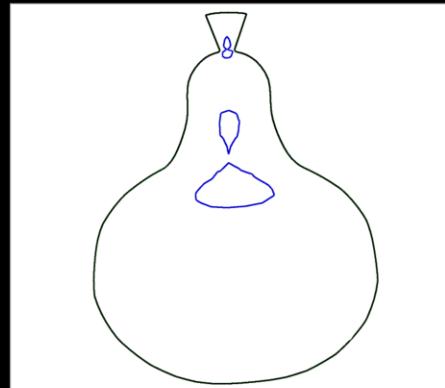
→ object-space curve extraction

Finally, the third definition, “zeros of radial curvature (subject to a derivative test)” naturally leads to an object-space algorithm.

## Suggestive Contours as Zeros of $\kappa_r$



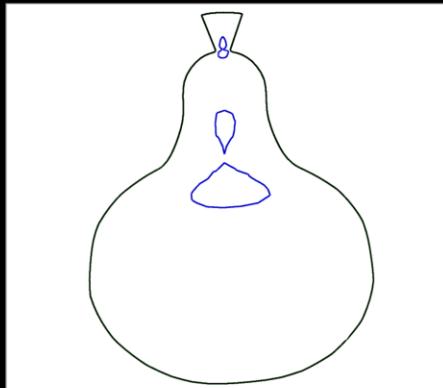
Mesh



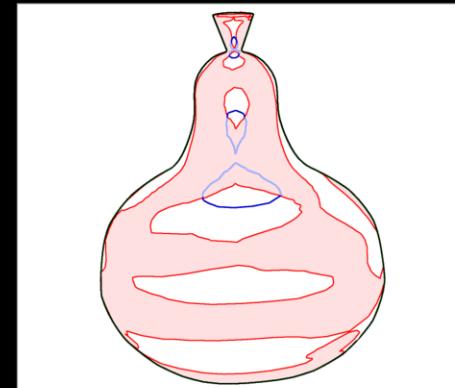
$\kappa_r = 0$

This algorithm extracts loops where radial curvature is zero, using either a brute-force approach or one of the acceleration techniques we talked about...

## Suggestive Contours as Zeros of $\kappa_r$



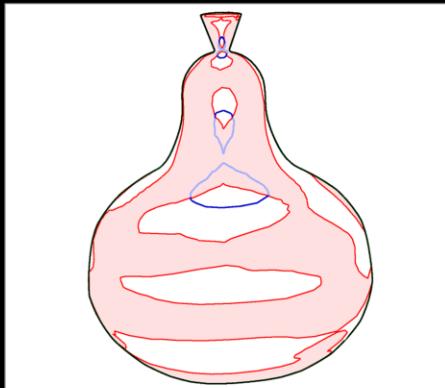
$$\kappa_r = 0$$



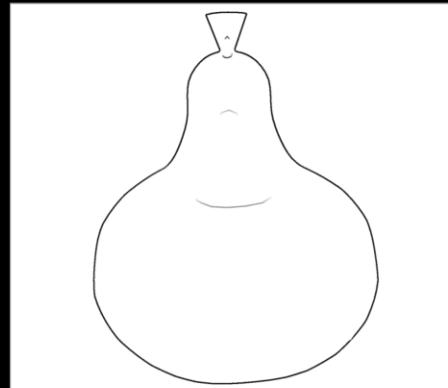
Reject if  $D_w \kappa_r < 0$

Then, the derivative (in the projection of the view direction, which we've been calling w) of the radial curvature is tested at each point along the curve, and we reject regions where it's negative.

## Suggestive Contours as Zeros of $\kappa_r$



Reject if  $D_w \kappa_r < 0$



Suggestive contours

Finally, we can stylize the lines however we want, such as this style that fades out strokes as the derivative of curvature approaches zero.

# Stability

Some curves unstable

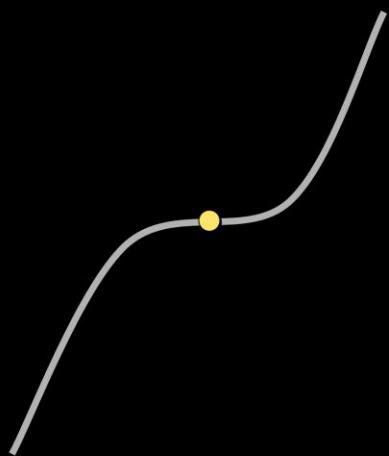
- under perturbations to geometry (i.e., noise)
- under changes in viewpoint

**Observation:** not drawn by artists

**Solution:** prune these curves away

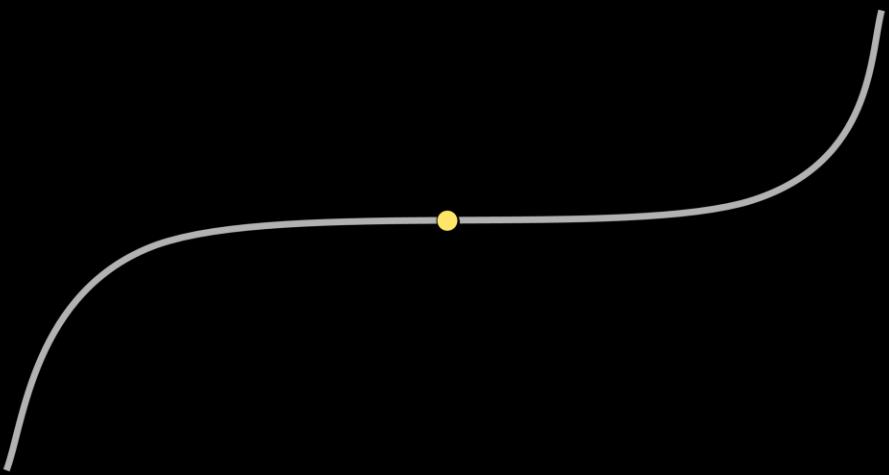
This algorithm can be augmented to throw out some of the unstable lines.

## Unstable Suggestive Contours



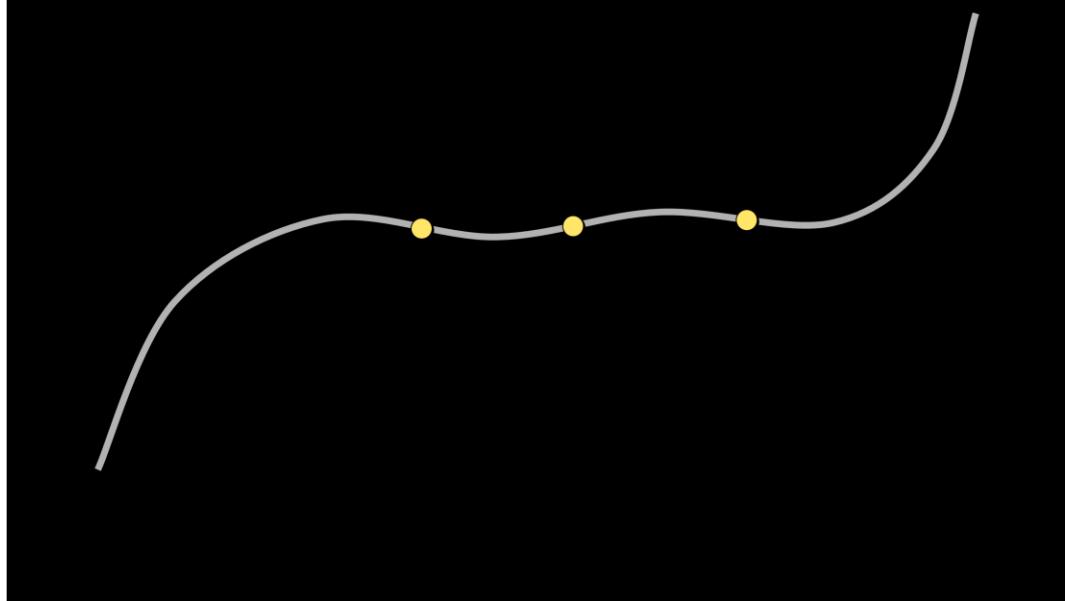
The idea is that if, at an inflection corresponding to a zero of radial curvature, the curvature is varying rapidly, that location is stable.

## Unstable Suggestive Contours



On the other hand, these shallow inflections are rather unstable...

## Unstable Suggestive Contours



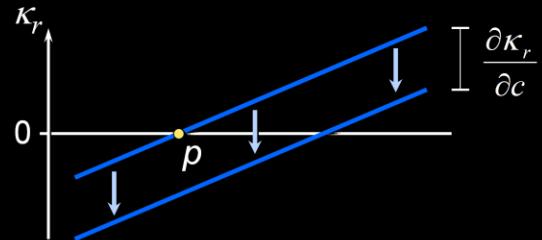
The addition of the slightest bit of noise causes perturbations in the suggestive contours, and might introduce new ones (or delete existing ones). So, one way to prune strokes is to apply some threshold to the magnitude of the curvature derivative, which eliminates these shallow inflections.

# Speed of Suggestive Contours

Definition of “stable”:

lines that move *slowly* as view changes

Derive speed using implicit function theorem



We can derive this speed from the implicit function theorem, which says that we have to look at both how quickly radial curvature is changing with respect to camera motion (numerator), and how quickly radial curvature is varying over the surface (denominator).

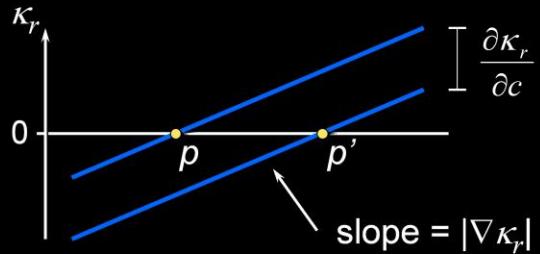
# Speed of Suggestive Contours

Definition of “stable”:

lines that move *slowly* as view changes

Derive speed using implicit function theorem

$$\max |v_{sc}| = \frac{\|\partial \kappa_r / \partial c\|}{\|\nabla \kappa_r\|}$$



We can derive this speed from the implicit function theorem, which says that we have to look at both how quickly radial curvature is changing with respect to camera motion (numerator), and how quickly radial curvature is varying over the surface (denominator).

# Speed of Suggestive Contours

$$\text{Projected speed w.r.t. angular camera motion} = \frac{2 \cos \theta}{\sin \theta} \frac{\sqrt{-K}}{\|\nabla \kappa_r\|}$$

where  $\cos \theta = n \cdot v$

**Stable** suggestive contours when

- approaching contours:  $\theta \rightarrow \pi/2$
- approaching  $K=0$  lines

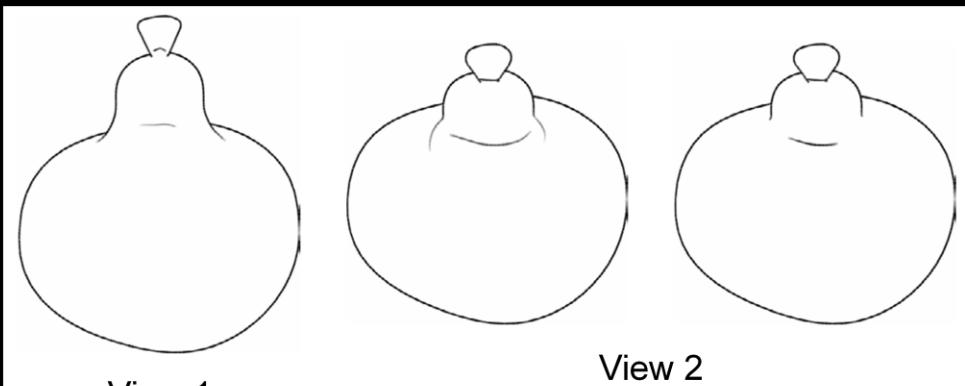
**Unstable** suggestive contours when

- looking straight at a surface:  $\theta \rightarrow 0$
- shallow inflection
- surface is twisting in the view direction

Working out the math, we get this formula. Looking at the individual terms, we can see that velocity will be largest, hence the curves most unstable, when the terms in the denominator are zero. These correspond to  $\sin(\theta)=0$  (looking at the surface) and  $\text{gradient}(kr)=0$  (shallow inflections).

Conversely, when the terms in the numerator are zero we have the maximal stability. This happens when  $\cos(\theta)$  is near zero (i.e., approaching a true contour), or when the Gaussian curvature is small (approaching the parabolic lines). This is a mathematical explanation of why suggestive contours (when considered over all views) tend to hug the parabolic lines.

## Results of Pruning



View 1

View 2

Test  $D_w \kappa_r$

Pruning based on  
s.c. speed

Here are a couple of examples of pruning according to the formula for the speed (right), or according to a simpler formula that just tries to avoid shallow inflections and lines seen head-on (center).

# Non-Exhaustive Extraction

Extending methods originally used for contours

**Hierarchical Algorithm** [Sander 00]

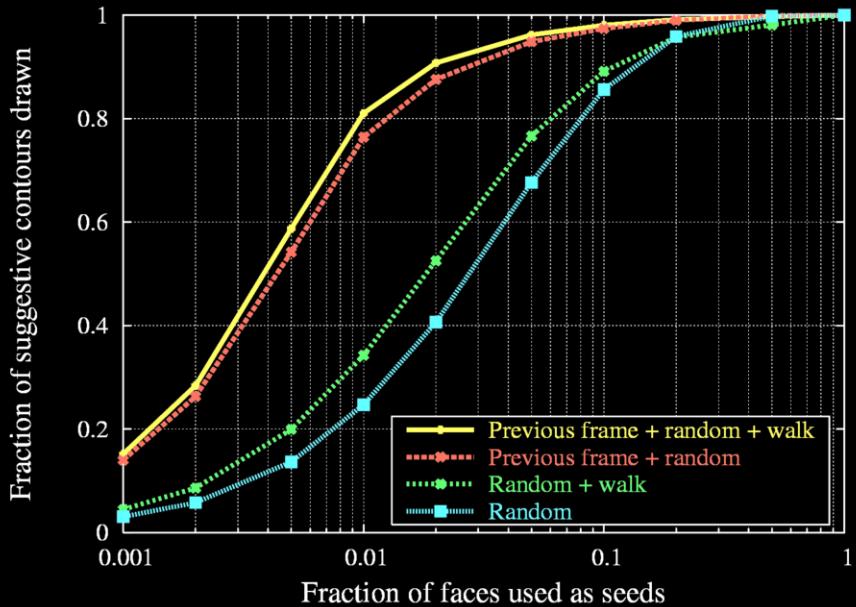
- Sphere/cone hierarchy
- In initial experiments, pruning not as effective as for contours

**Stochastic Algorithm** [Markosian 97]

- Test a subset of faces for zero crossing
- Once found, follow loop
- Result: find most lines using 1-10% of faces as seeds

Very similar acceleration techniques to those used for contours can be used for suggestive contours.

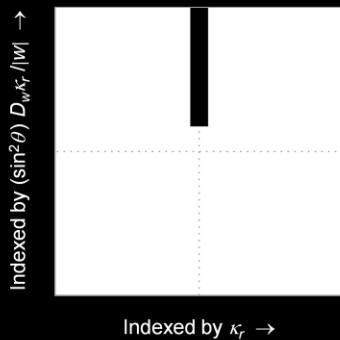
## Performance of Stochastic Algorithm



The performance of the randomized algorithm across a flythrough involving several views is presented here. Using the lines from the previous frame as seeds had a fairly large impact, while another technique (walking “downhill” from the random seeds in search of a zero of radial curvature) shows limited improvement. Overall, very decent results can be obtained by testing 10% of the faces or less.

# Using Graphics Hardware

Alternative algorithm: use texture mapping



Use mipmapping for near-constant stroke width

Finally, there's a way to use the graphics hardware to extract suggestive contours, similar to the use of texture maps indexed by  $(n \cdot v)$  to draw contours. The idea is to use a texture map with a dark line in part of it, with the horizontal texture coordinate indexed by radial curvature and the vertical coordinate indexed by the derivative (possibly with some  $\sin(\theta)$  terms as well). The dark part of the texture map will only be accessed if the radial curvature is near zero and the derivative is greater than zero (or some threshold). Note that in most cases the curvature and derivative will have to be computed "by hand" at each vertex, and the correct texture coordinates passed in.

## Comparison of Effects



Constant-weight  
strokes

Strokes faded  
based on  $D_w \kappa_r$

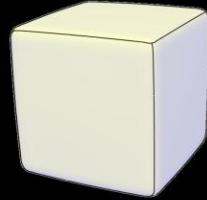
Texture-map  
rendering

Here's a comparison of a few different algorithms. The first two images come from the object-space algorithm, with and without fading of strokes. The rightmost image was done using the texturemap-based algorithm.

# Finding Ridges and Valleys

Definition of ridges:

- Positive local maxima of maximum principal curvature, in corresponding principal direction



Valleys “go the other way”

Some algorithms find extrema directly, while others look for zeros of derivative of curvature

Finally, let's look briefly at algorithms for computing ridge and valley lines. Because these are defined in terms of high-order derivatives, which are often noisy, a big challenge is in getting good, robust estimates of these differential quantities.

## Finding Ridges and Valleys

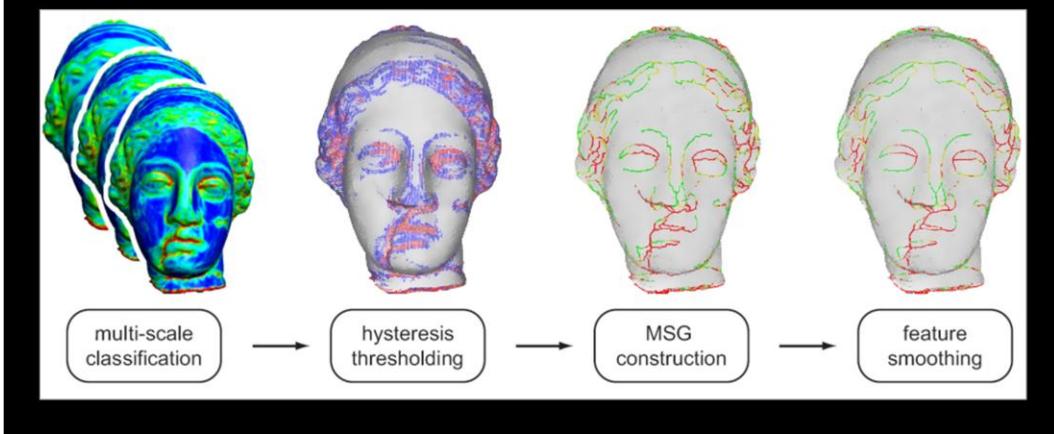
Higher-order derivatives very sensitive to noise  
[Otake 04] uses implicit function fit,  
filters based on curvature integrated over length



A paper from last year achieved good results by computing the derivatives using implicit function fits, then doing some filtering on the resulting strokes.

# Finding Ridges and Valleys

[Pauly 03] looks for stable extrema over different scales of smoothing



Another interesting approach is to look for lines that are stable over different scales of filtering. This algorithm actually operates on unorganized point clouds, and doesn't need a full mesh.

## Summary

- *Image-based vs. object-based algorithms*
- Control over line thickness, stylization
- Filtering and/or smoothing

Software @

<http://www.cs.princeton.edu/gfx/>

# Part VI: Stylization of Line Drawings

Adam Finkelstein

Line Drawings from 3D Models  
SIGGRAPH 2008

Now that “perfect” design is possible with the click of a mouse, the industrialized world has become nostalgic for “imperfect” design.

As computer-aided everything takes over our lives we begin to realize, little by little, what is missing from the high-tech world. We realize that a crooked line sometimes has more soul than a perfectly straight one....

-- David Byrne *When Bad Art is Good*  
Utne, March-April 2003

I'd like to start this section with a quote from David Byrne in an article for Utne. In the article he was mostly talking about 2D design and illustration, but I believe his point translates to renderings from 3D as well.

# Aspects of stylization

Lighting

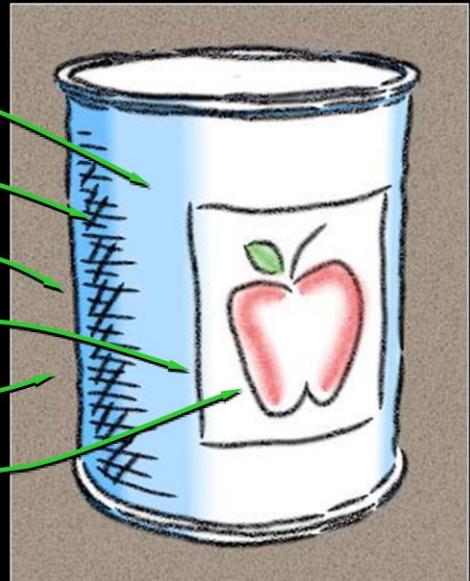
Tonal Marks

Brush Style

Brush Path

Paper Effect

Abstraction



Various aspects of stylization include:

Abstracted shading such as toon shading and tonal marks such as hatching;

Stylized brushes that taper at their ends, have varying width and transparency, wobble along their paths, etc;

Paper or media effects that effect the lines that are drawn as well as the shaded or blank areas of the drawing;

Also tightly coupled with stylization is abstraction, which Doug will address in greater detail in Part 6 of the course.

# Overview of this section

1. Stylized lines
2. Visibility of lines in 3D
3. How to specify stylization
4. Temporal coherence for stylized lines

Here is an overview of this section.

# Overview of this section

1. Stylized lines
2. Visibility of lines in 3D
3. How to specify stylization
4. Temporal coherence for stylized lines

We'll start by talking about stylized lines.

# Stylized strokes



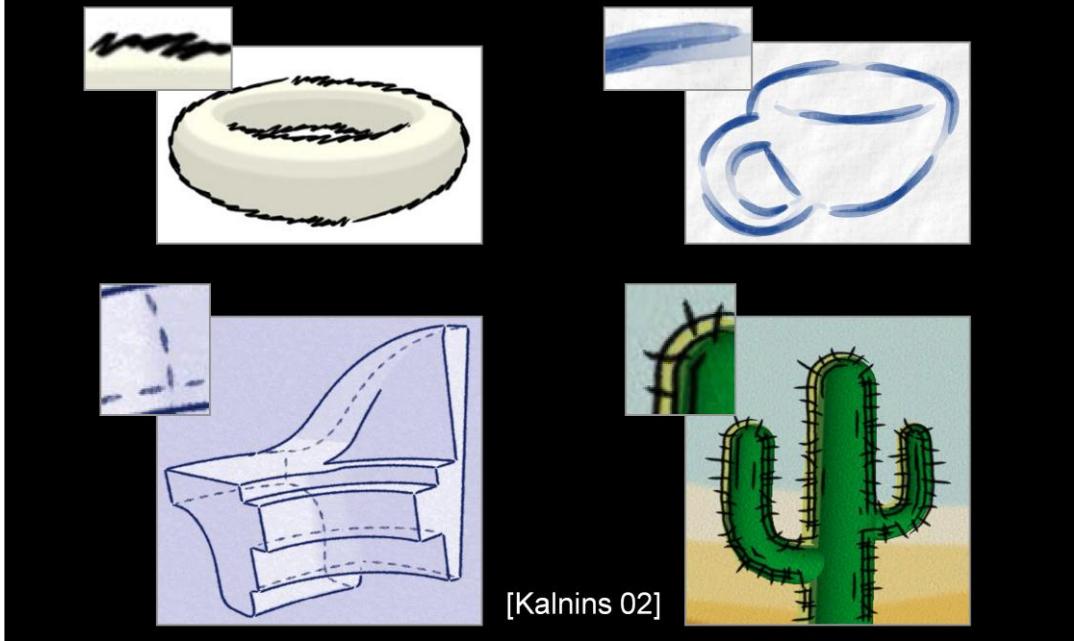
[Hsu 94]



[Curtis 98]

Strokes are the fundamental primitive of a line drawing. Each individual stroke has many qualities in addition to its path -- it can have varying thickness, wiggliness, opacity, and texture, not to mention its time-dependent nature. These qualities give line drawings much of their character or charm, and can convey feeling as well.

# Many Forms of Stylization



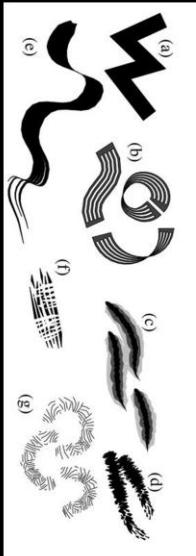
[Kalnins 02]

Here are a few more examples of the range of effects possible, showing up close what the brush might look like.

Lines can be made to wiggled with offsets, textured with watercolor strokes, broken into dashes for mechanical drawings, and even geometric effects can even be suggested, such as the thorny stylization used on silhouettes of this cactus.

# Skeletal strokes

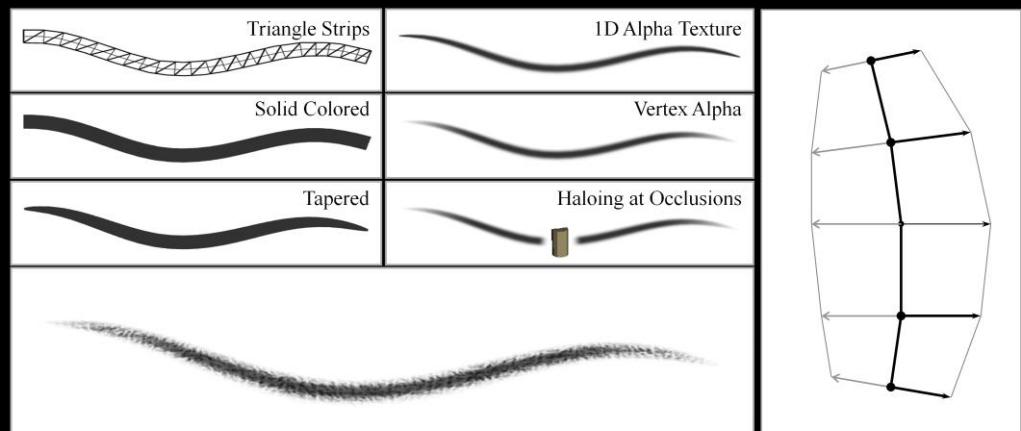
[Hsu 94]



In 1994, Hsu presented a system called Skeletal Strokes that described how such qualities could be applied to CG lines. Indeed many of these features are now standard fare in commercial programs such as Adobe Illustrator.

# Strokes as triangle strips

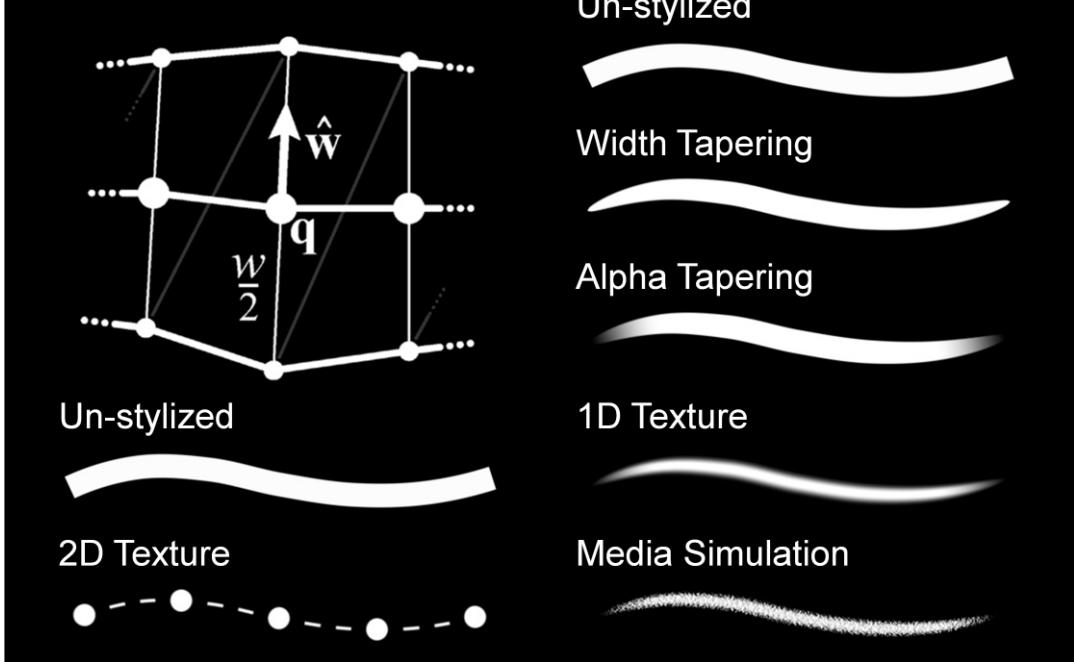
[Northrup 00]



Thinking about the problem of drawing such stylized marks efficiently as part of an interactive rendering system for 3D models, Northrup et al. described how to map such stylization onto textured triangle strips drawn with a standard graphics interface such as OpenGL.

# Stroke texturing

[Kalnins 02]

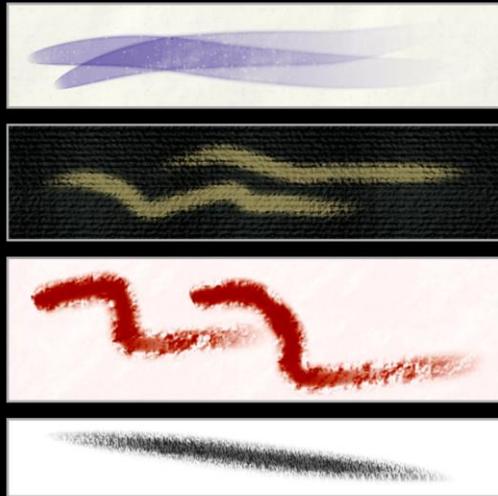


This technique was adapted by Kalnins et al to handle 2D textures along the length of the stroke as well as a little more emphasis on media interaction with paper.

# Brush Style

Per stroke:

- Color
- Width profile
- Alpha profile
- Paper
- etc.



[Kalnins 02]

The lines can thus be drawn with very stylized brushes that simulate the look of real media, at interactive frame rates.

## Overview of this section

1. Stylized lines
2. Visibility of lines in 3D
3. How to specify stylization
4. Temporal coherence for stylized lines

The next question to address is how do we know which lines extracted from a 3D model are visible?

# Visibility challenges



[Kalnins 02]

- Interactivity (ray casting is expensive)
- Thick, wobbly lines (z-buffer won't work)
- Silhouettes are at the visibility threshold

There are a number of challenges for computing visibility. Of course it is possible to fire a ray from many points each line to the viewer and see if it intersects anything along the way. However, that strategy is expensive, meaning it won't support interactive applications for even moderately complex scenes. For interactivity, why not just use the z-buffer? Well it turns out that if you draw the model and also thick, wobbly lines in 3D, then the z-buffer algorithm will chop the lines in ugly ways where they happen to penetrate the model. Finally, a particular challenge for silhouette lines is that by definition they exist right at the boundary between visible and invisible parts of the model, making many visibility algorithms naturally unstable at such points.

# Visibility strategies

Hardware methods: minimal stylistic control  
[Rossignac 92, Gooch 99, Raskar 01]

Ray casting may be accelerated by 3D/2D analysis [Appel 67, Markosian 97]

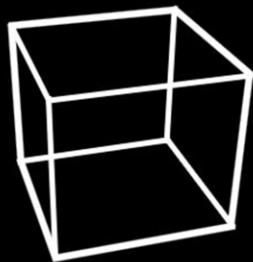
Item buffer gives image-space precision  
[Northrup00, Kalnins03]

Three general strategies have been used for visibility of lines. First, there are hardware-based methods that cause visible -- or invisible -- silhouette lines to appear without explicitly searching for them on the model. Such methods are fast, but admit only minimal control over the stylization of the lines they reveal. Second, while ray casting can be slow, analysis of the shapes involved can reduce the number of rays cast. Third, a hybrid method based on an item buffer can compute visibility with image space precision. I'll go into a little more detail on these last two strategies.

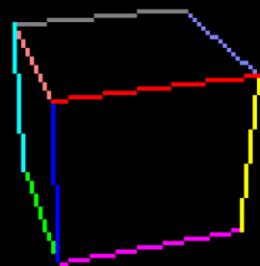
# Item buffer for line visibility

“ID reference image” [Northrup00]

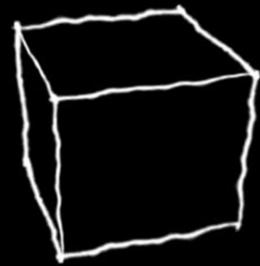
Earlier use in ray tracing [Weghorst84]



model



item buffer

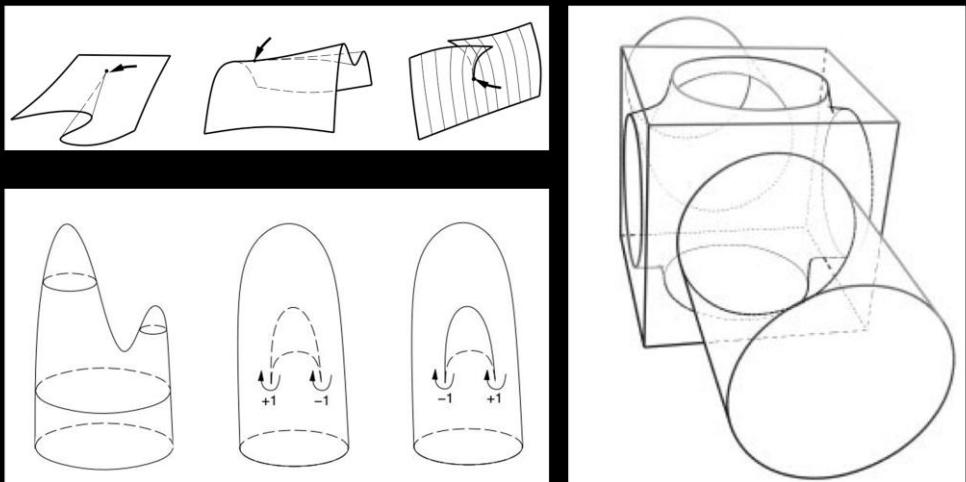


strokes

Here's how the item buffer works. You draw the polygons of the model, using polygon offset, to set the z-buffer. Then you draw the lines of the model, each colored by a unique color that identifies that line. Then to determine which parts of which lines should be drawn as strokes, you walk each line checking for the appropriate ID/color. Then you draw those visible portions as strokes.

# Visibility

[Appel 67]: “quantitative invisibility”

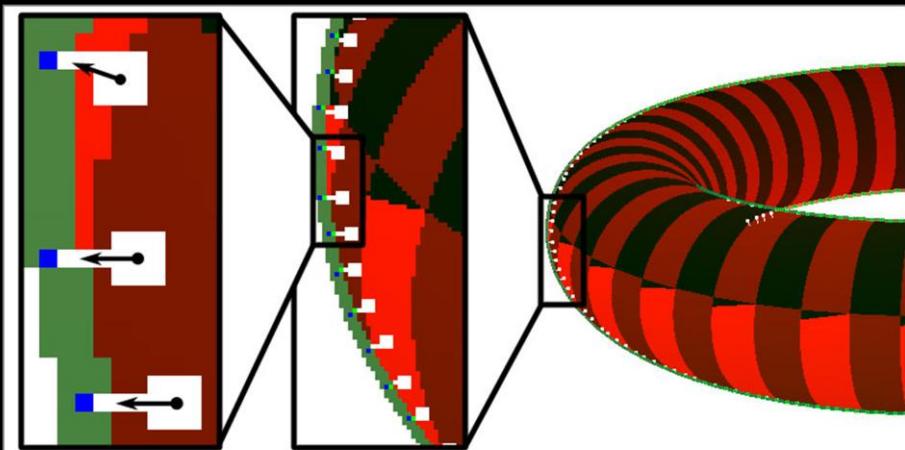


[Markosian 97]

Way back in 1967, Appel introduced the notion of “quantitative invisibility” wherein the set of lines is examined to find explicit places where visibility can change. After such analysis, only a few rays need to be fired to determine the visibility for the whole drawing. In the “Real-time NPR” system, Markosian et al offered some improvements to Appel’s algorithm, requiring even fewer rays to be fired. Based on these algorithms, they were able to render drawings such as the one on the right for very complex models faster than could be done by rendering the full mesh in graphics hardware of the time. However, these algorithms that analyze quantitative invisibility are difficult to implement because they are sensitive to certain kinds of numerical instabilities.

# Visibility

[Northrop 00]: Item buffer gives visibility with image-precision

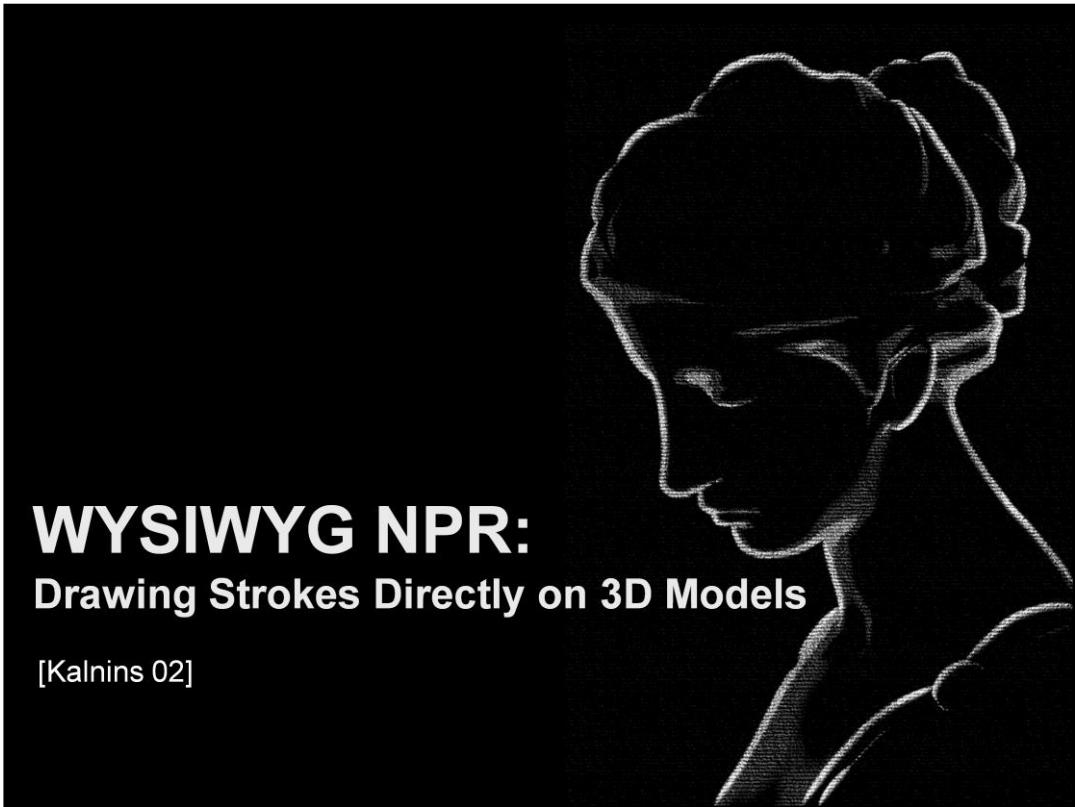


Therefore, Northrup and others developed a hybrid visibility system using an item buffer rendered in graphics hardware. The idea is that you render the entire mesh as well as the lines of your line drawing, each primitive marked with a specific ID. Then you can walk over the lines searching for in the reference image to see if the appropriate ID is in the neighborhood. If so, you consider that part of the line to be visible. Kalnins et al used this method as well. Furthermore they were able to leverage the item buffer for propagating parameterization of the lines from one frame to the next for temporal coherence, as I will describe shortly.

# Overview of this section

1. Stylized lines
2. Visibility of lines in 3D
3. How to specify stylization
4. Temporal coherence for stylized lines

But first, let me discuss the question, how can an artist specify what kind of stylization he would like for a model?



## **WYSIWYG NPR:** **Drawing Strokes Directly on 3D Models**

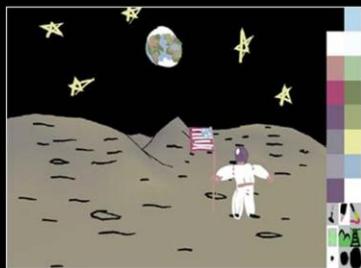
[Kalnins 02]

In 2002 my student Rob Kalnins and others presented a system called WYSIWYG NPR, which stands for “What you see is what you get … non-photorealistic rendering.”

The basic idea is that an artist should be able to draw right onto the 3D scene how he would like it to look.

# Long-Range Goal

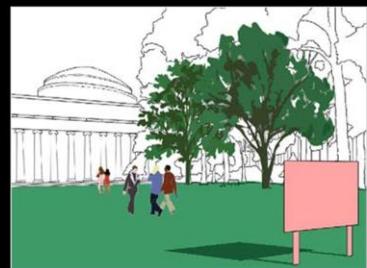
Create full scene by drawing



[Cohen 00]



[Bourguignon 01]



[Tolba 01]

A handful of other researchers have considered tools to creating stylized 3D scenes from scratch via drawing. In these approaches, the artist's input produces a form of stylized NPR geometry. These systems are wonderful because you start with nothing, and produce both a 3D model and its stylization at the same time. But they make assumptions about what you might mean when you draw a stroke and are therefore limited in the kinds of models that can be created.

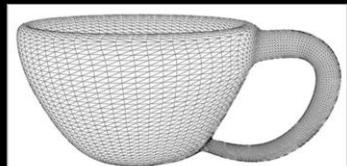
# Annotating geometry



WYSIWYG Painting [Hanrahan 90]

In this sense the WYSIWYG NPR system more like that of Hanrahan and Haeberli in which you start with a model and draw directly on it. However, in Hanrahan's system they were making textures for the model, rather than thinking about annotating a model with an NPR rendering style.

# WYSIWYG NPR



- Draw into 3D scene
- Retain style in new views
- Ensure coherent animation



Here's how the WYSIWYG NPR system works. The designer begins by importing some existing 3D geometry like this tea cup mesh. He then describes its stylized appearance annotating the objects with stylized strokes drawn directly into the scene. When the user moves to new camera positions, the system automatically retains these stylizations. Furthermore, under animation, the system also ensures that the stylizations *\*evolve\** in a coherent fashion.

# Aesthetic Flexibility



We have found that the payoff in aesthetic flexibility is immediate.

Even with a relatively small toolbox of effects that describe shortly by simply permitting the designer to express the stylization via hands-on means...

# Aesthetic Flexibility



...a wide range of diverse aesthetics can be achieved. All of these results were produced by annotating the same 3D tea cup mesh from the previous slide.

WYSIWYG NPR (video)

WYSIWYG NPR

Introduction

The following video clip shows the kind of interface presented to the artist.

# Overview of Components

Base Coat

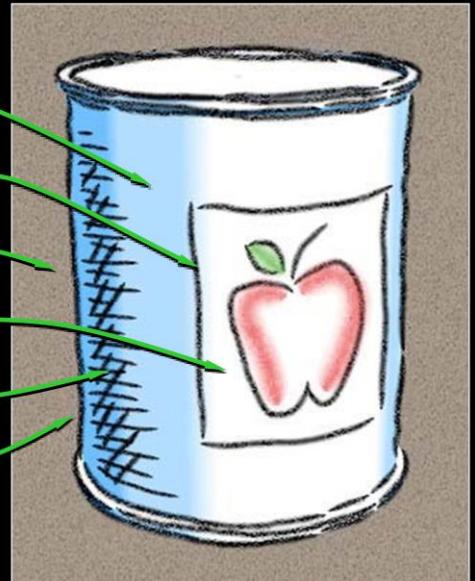
Brush Style

Paper Effect

Detail Marks

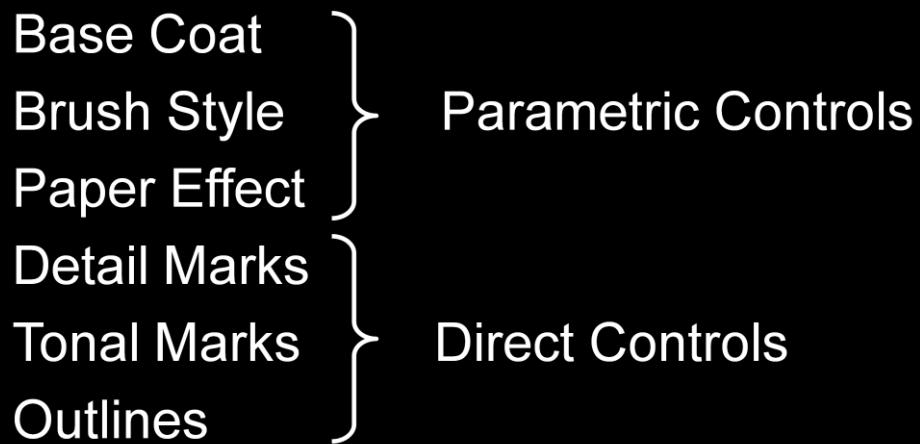
Tonal Marks

Outlines



The WYSIWYG NPR system provides a bunch of different tools to produce a combined effect.

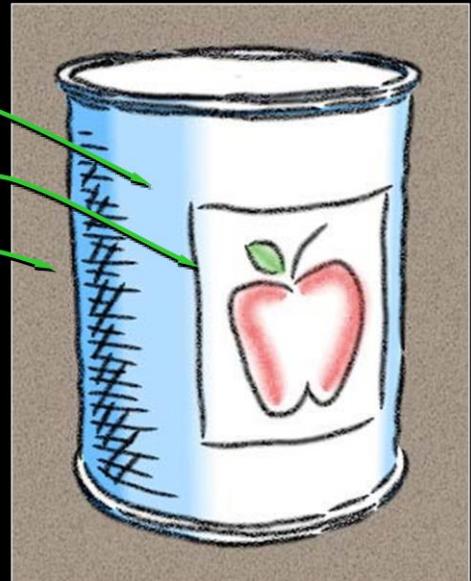
## Two Categories of Control



From an organizational perspective, we can group these components into two broad categories: parametric and direct controls. Let me begin with parametric category...

# Parametric Controls

Base Coat  
Brush Style  
Paper Effect  
Detail Marks  
Tonal Marks  
Outlines



Parametric controls are rendering options tuned by sliders, or checkboxes, for instance to adjust parameters like colors for shaders, widths for brush styles, and textures for paper and media simulations. Controls like these are common to much of the previous work in NPR, and indeed our system also employed these sorts of components.

Though necessary, I won't really focus on these tools today.

# Direct Controls

Base Coat

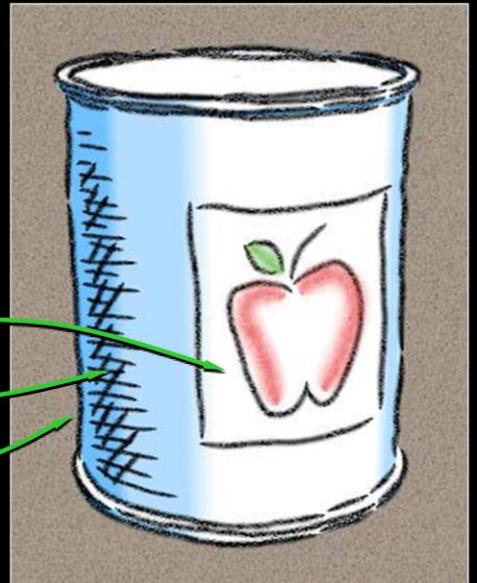
Brush Style

Paper Effect

Detail Marks

Tonal Marks

Outlines



Instead, I'll focus on the second category -- direct controls – which take their input via a more natural means, such as a tablet. These controls allow the artist to influence the look of the scene by sketching strokes directly into it.

# Overview of Components

Base Coat

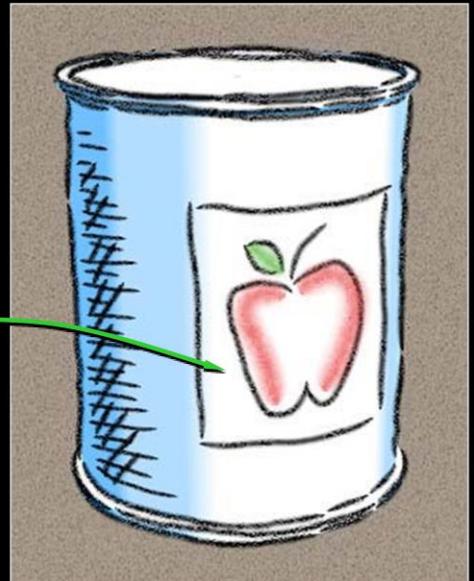
Brush Style

Paper Effect

Detail Marks

Tonal Marks

Outlines



The first of the direct controls are concerned with creating detail marks on object surfaces, such as the strokes depicting a label on this apple sauce can.

# Detail Marks

Direct control:

- Draw on surface
- Generate strokes



For magnified or oblique views:

- No blurring or aliasing
- Explicit control of stroke width

... unlike texture maps

To create detail marks, the user simply selects a brush style, and sketches features onto the mesh. This has the effect of defining stroke paths embedded in the 3D surface. As described earlier, each path is rendered using a stroke primitive based on triangle strips. Note that under magnified or oblique views, this approach differs from texture maps into two ways. First, there are no blurring or aliasing artifacts, because we render these paths using a stroke primitive, and second, we have explicit control over stroke width.

# Detail Marks: Stroke Behavior

## Width Control: *Foreshortening*



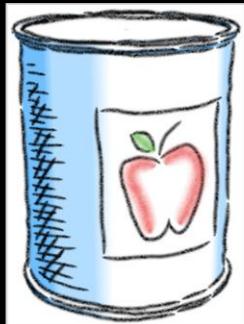
## Width Control: *Magnification*



For instance, we may desire to keep stroke widths close to their original extent in all views, suggestive of a real brush that cannot change size. The designer can control this behavior independently under foreshortening and magnification. Here strokes maintain their width in foreshortening, even extending beyond the surface in the limit, while under minification, widths are allowed to shrink to avoid a piling up of detail.

In this case, the artist employs a subtle effect: the purple flower strokes shrink more quickly than the green ones, so the large central leaf dominates in distant views.

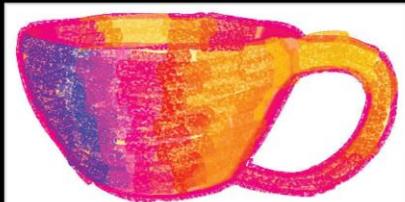
## Detail Marks: Flexibility



Labels



Decals



Painterly Effects



Line Drawings

The designer can achieve a wide range of effect with this straight forward tool. A few strokes can depict labels and decals as in the upper images, while many wide stroke can create rich painterly effects ... and many narrow strokes can lead to detailed line drawings. I'll show several of these scenes in animation in a little while.

# Overview of Components

Base Coat

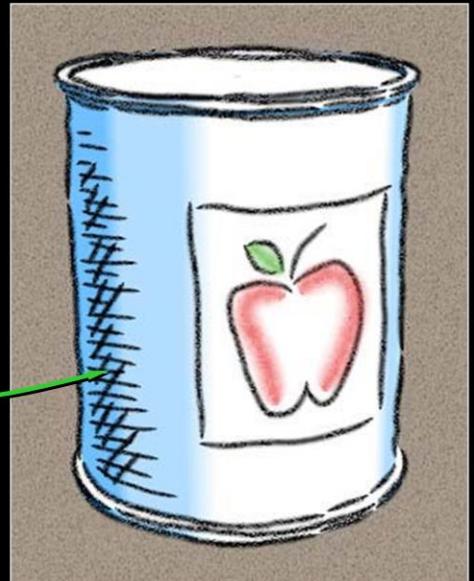
Brush Style

Paper Effect

Detail Marks

Tonal Marks

Outlines



The next direct effect is hatching. Much like decals, hatching is drawn directly on the surface in the location and style desired by the artist. Because these elements are not necessary for sparse line drawings and because of time limitations...

# Overview of Components

Base Coat

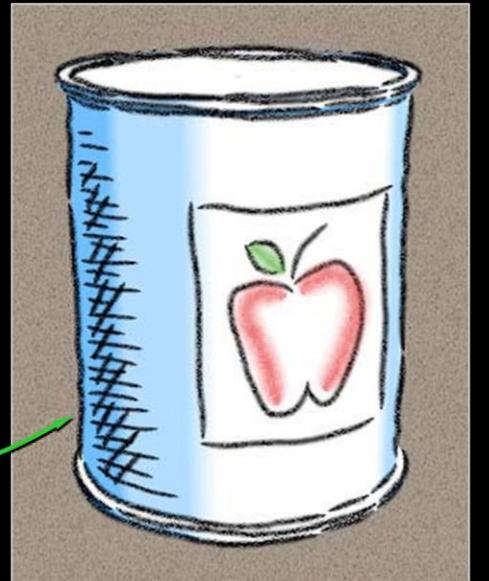
Brush Style

Paper Effect

Detail Marks

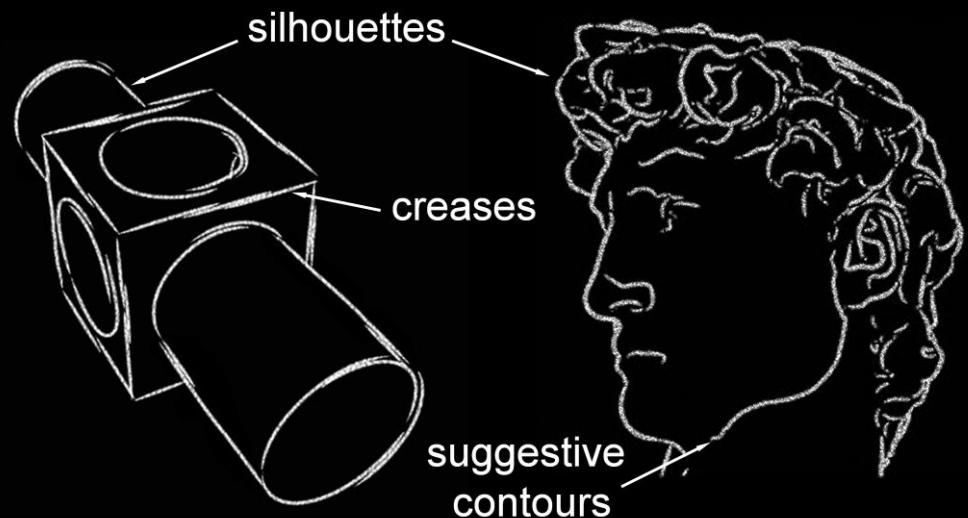
Tonal Marks

Outlines



...I'm going to push on to talk about outlines.

# Outlines



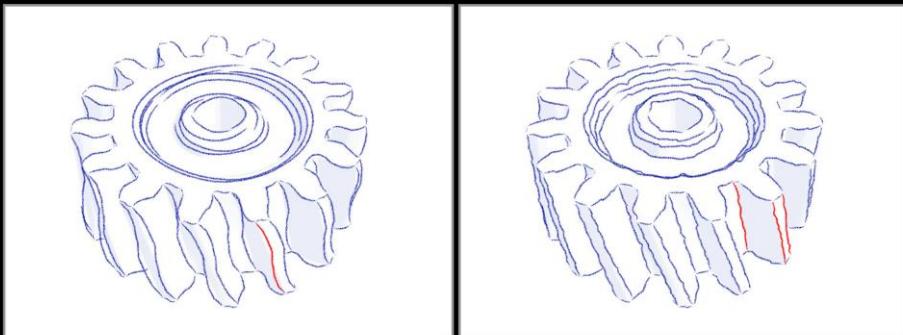
In earlier sessions we talked about silhouettes, creases and suggestive contours. These three elements, though not exhaustive, can produce a wide range of line drawings (and are often components of more complex drawings).

# Outlines

(VIDEO)

Let me show you a video of outline stylization in action.

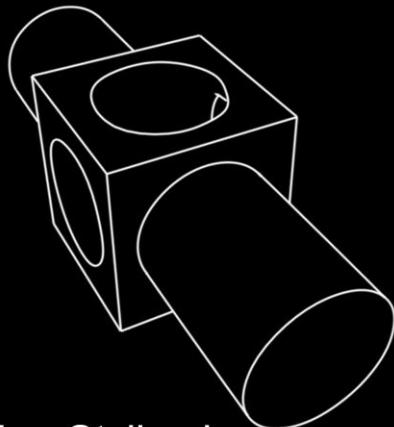
# Stylization Synthesis



Synthesis uses Markov model.  
Similar to “video textures” [Schödl 00]

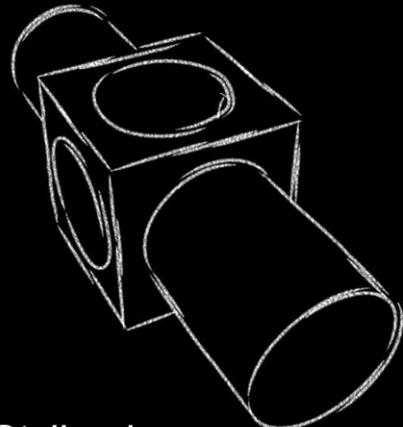
We alleviate the tedium of over-sketching all creases of a model, by providing two schemes for generating stylization from example input. The first method rubber-stamps a single stylization prototype, while the second method synthesizes novel stylization from a small set of examples. The latter method is achieved by feeding the examples through a Markov process model using a method similar to that of Schodl’s Video Textures paper.

# Outlines: Stylization



Non-Stylized

- Simple (`GL_LINE`)
- Easy to animate



Stylized

- Rich, flexible styles
- Challenge to animate

Stylizing the outlines of a model presents a particular challenge for temporal coherence.

On the left, we show the mechanical part with no stylization yet applied to its lines; this rendering scheme can be easily implemented in graphics hardware, but it also has the added advantage of being easy to animate because there are no stylized features to keep track of over time.

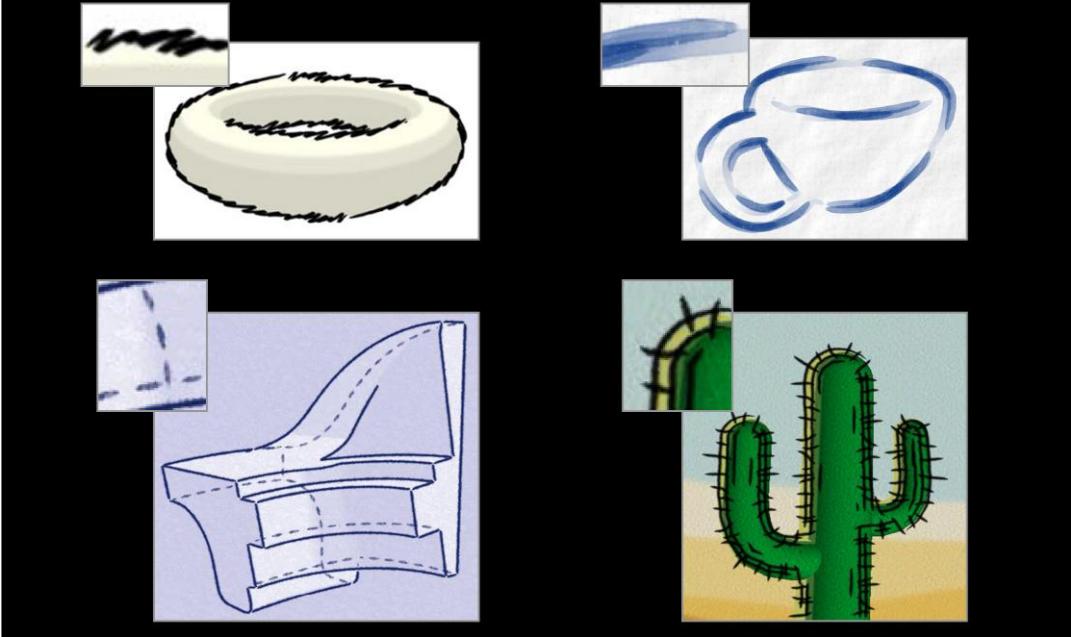
On the right, we show the same mesh rendered with stylized lines. Such stylization can lead to a wide range of rich aesthetics, but this comes at the cost of additional challenges during animation, since we'll have to keep track of the salient features of the stylization over time.

# Overview of this section

1. Stylized lines
2. Visibility of lines in 3D
3. How to specify stylization
4. Temporal coherence for stylized lines

So the last part of this section addresses the challenge of providing temporal coherence for such lines.

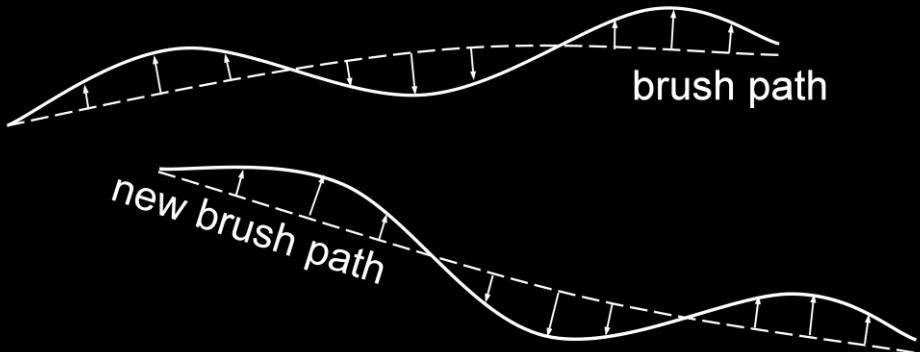
# Many Forms of Stylization



Just as a reminder, we're dealing with a variety of styles of lines, and we'd like their fine-scale detail to be coherent from one frame to the next.

# Stylization as Offsets

- Artist over-sketches outline
- Stylization recorded as 2D offsets
- Applied to new brush path

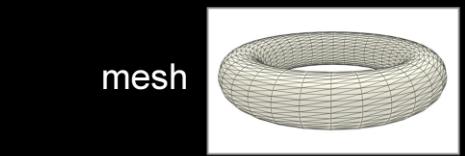


Imagine that the dashed line is some crease we want to stylize. We call its screen-space projection a brush path. If the user then over-sketches the crease to stylize it. The system records it as a sequence of screen space offsets perpendicular to the brush path. To apply the stylization to this crease in another view, the 3D crease is once again projected to yield the new brush path and the offsets are used to generate the stylized path.

For creases, the relationship between the original base path and the new base path is obvious because the crease remains at a fixed location on the mesh in all views. However, for silhouettes this problem is more complicated.

# Stylized Silhouette Pipeline

1. Extraction



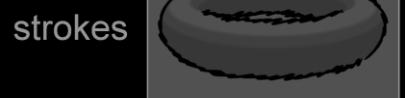
2. Visibility



visible paths



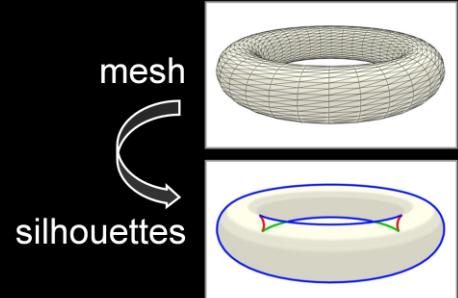
3. Rendering



To see why, we must consider the silhouette rendering pipeline.  
Beginning with the triangle mesh...

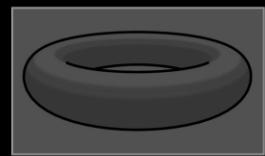
# Stylized Silhouette Pipeline

1. Extraction



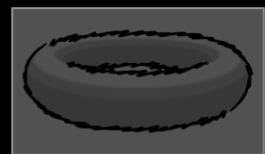
2. Visibility

visible paths



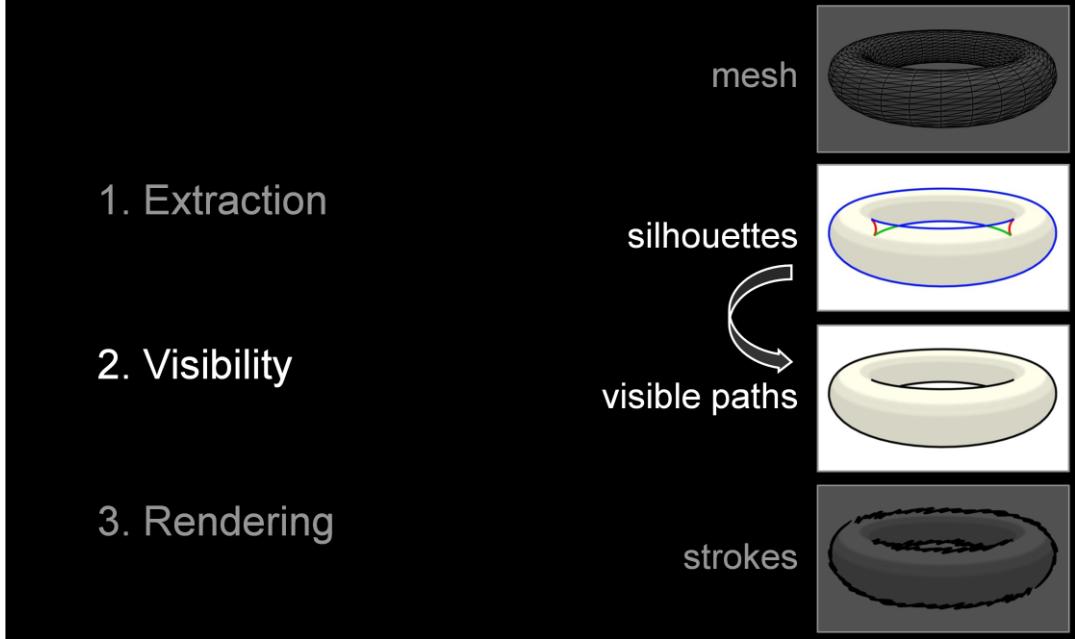
3. Rendering

strokes



The first step is to extract the silhouette contours from the underlying geometry. Szymon discussed this problem in an earlier session.

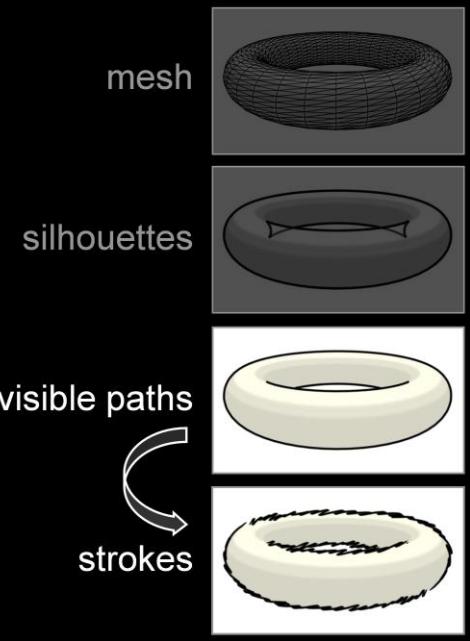
# Stylized Silhouette Pipeline



The second stage is to compute visibility, as I discussed moments ago.

# Stylized Silhouette Pipeline

## 1. Extraction



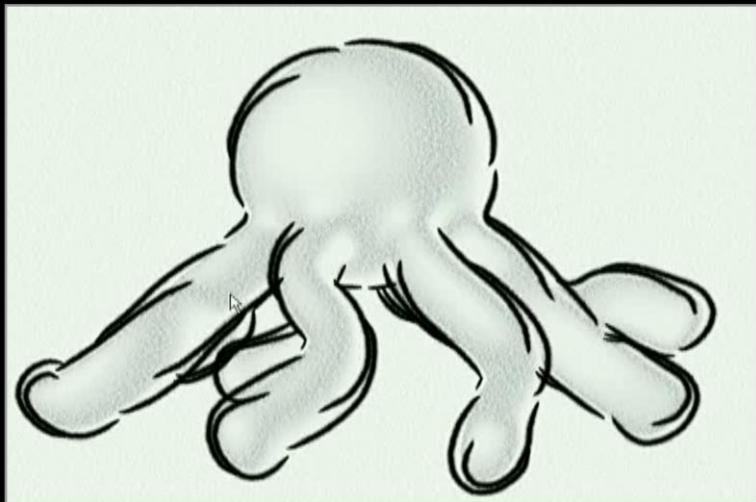
## 2. Visibility

## 3. Rendering

- Re-mapping artist's stylization

And the final stage is to render these visible paths using the stylization provided by the artist.

# Coherence?



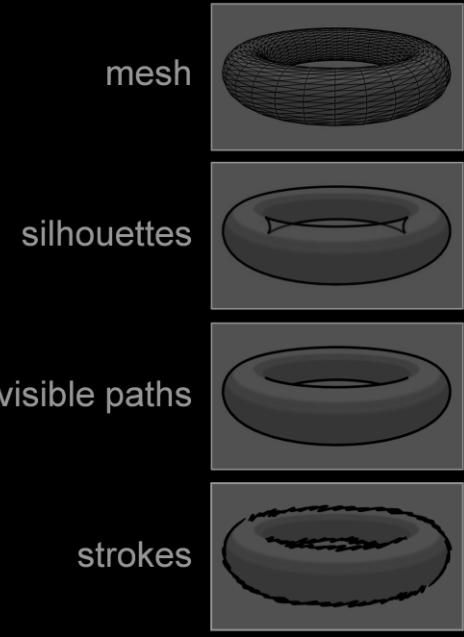
Maintaining temporal coherence of these effect is challenging. To illustrate this I'll show you an animation of this octopus -- first without and then with -- explicit attention to the coherence of the stylization from frame to frame

-Initially, we assign the stylization using the natural arc-length parameterization of the silhouettes. This intrinsic parameterization leads to coherence artifacts, such as 'popping' and 'swimming'.

-To solve this problem we propagate parameterization information from frame to frame. This allows us to explicitly assign stylization with the goal of coherence. Notice how the artifacts have been vastly reduced.

# Coherence?

1. Extraction



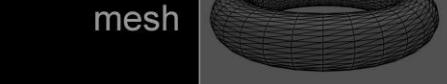
2. Visibility

3. Rendering

So, which part of the silhouette stylization pipeline, described earlier, is responsible for producing coherence?

# Coherence?

1. Extraction



2. Visibility



2.5. Parameterization



parameterized paths



3. Rendering



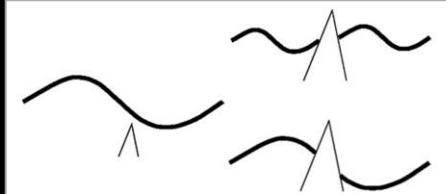
strokes

Well, it turns out we skipped a step. Before the visible paths from stage 2 can be rendered with strokes in stage 3, it is necessary to parameterize those paths. This is the missing step.

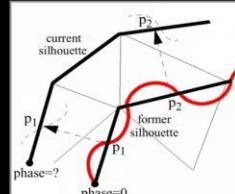
As demonstrated with the octopus example, the coherence of the underlying parameterization is revealed by the stylization. If we do not explicitly assign parameterization with coherence, the lack of coherence arising from an implicit parameterizations will be revealed.

## 2.5. Parameterization

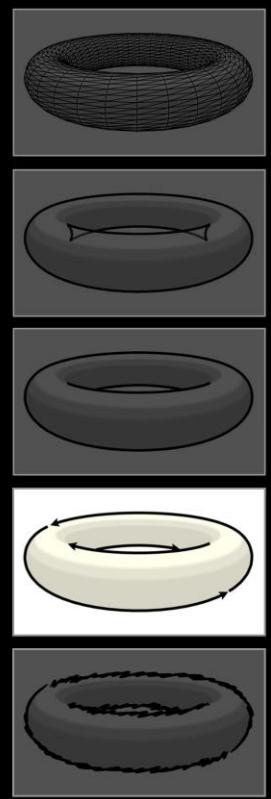
Little previous work on  
parameterization coherence



[Masuch98]



[Bourdev98]

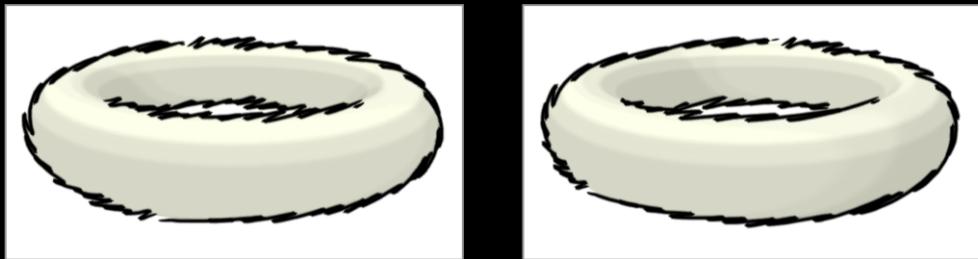


There had in fact been a small amount of previous work in this area of the pipeline

-Masuch in 98 demonstrated how to maintain coherence for the specific case of a single stroke under partial occlusion.

-Bourdev in the same year presented a more general framework. His sample propagation approach inspired the approach I'm about to describe.

# Coherent Stylized Silhouettes



frame  $i$



frame  $i+1$

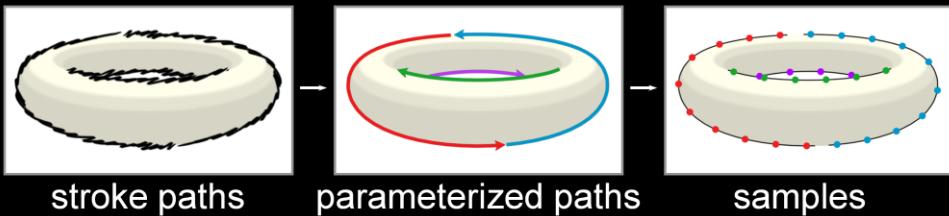
Applications: both offline and interactive animation

Let me concisely state the problem we'll address here.

Given some stylization of the silhouettes of an animated scene at time  $i$ , how can we achieve a coherent stylization in the next frame,  $i+1$ ?

Note that our motivation for considering consecutive frames -- rather than the entire animation as a whole -- is that we want to achieve coherence in interactive in addition to offline settings.

## Propagation: Samples



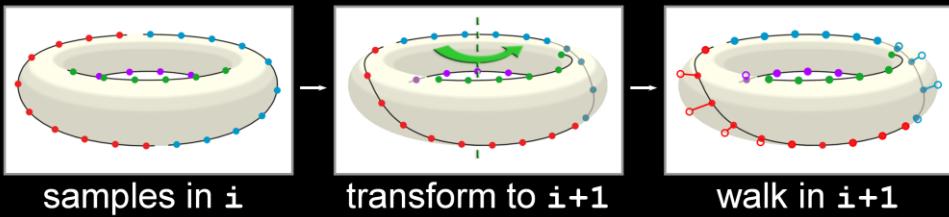
Each sample contains:

- Parameterized path ID
- Parameterization value
- Location on mesh triangle

Recall that the stylized strokes are generated on top of a set of continuously parameterized paths.

To sample this parameterization state for some given frame, we record a set of discrete samples along each parameterized path. Within each sample we store the unique ‘id’ of the path it came from (as denoted by the color coding in the figures), as well as the value of the parameterization at its position along on the path and its location on a triangle of the mesh (so the sample can be located in the next frame, even if the mesh deforms).

# Propagation: 3D and 2D

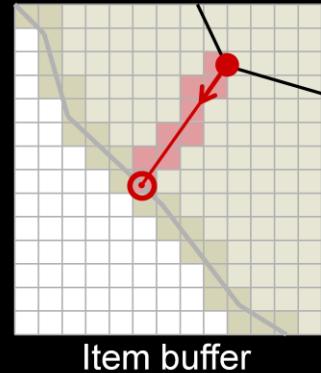


**Transform samples in 3D**

- Animation and camera

**Register samples in 2D**

- Project into ID image
- Search along normal
- Record the sample

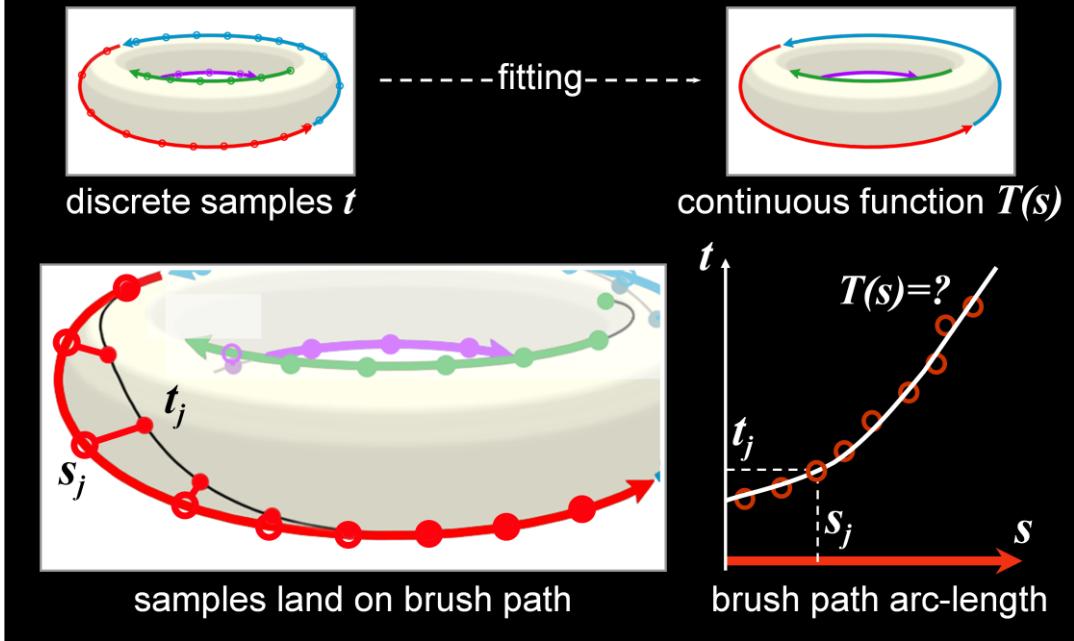


Propagating the samples from frame  $i$  into frame  $i+1$  is a two step process. First we transform the samples to their new 3d location, accounting for animation and camera changes. In this example, the torus has been rotated about a vertical axis.

The samples from frame  $i$  will still lie on the surface of the mesh, but in general they will not coincide with the silhouettes of frame  $i+1$ . So, the second step registers these samples with the new silhouettes by walking in 2D using the item buffer I mentioned earlier, which allows us to efficiently search for the silhouettes in real-time, as follows.

We begin by projecting the transformed sample into the item buffer. We then search along the projected normal direction, since near silhouettes, this will point toward the boundary. If a silhouette pixel is found, we register the sample at the intersection point.

# Optimization: Fitting

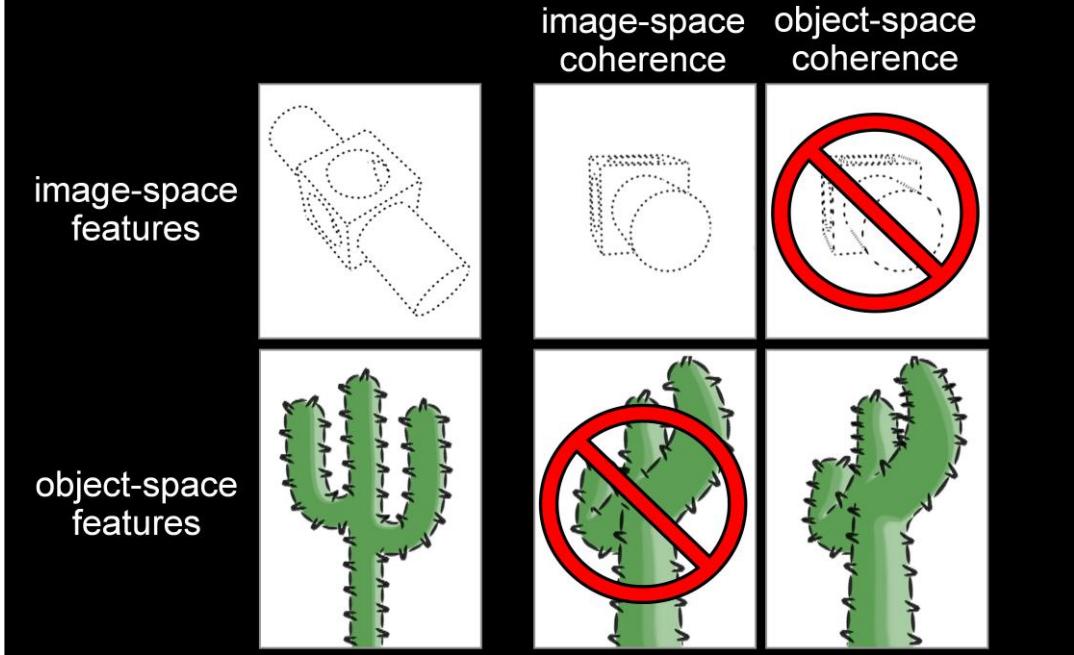


The final step is to parameterize the brush paths with a continuous function, based on the set of samples that were propagated from the previous frame.

To see how this works, consider the example of the red brush path. Each parameter sample,  $t_j$ , that arrives on this brush path can be recorded in terms of its position,  $s$ , measured in arc-length along the brush path. This sample contributes one parameterization data-point to the graph on the right.

Indeed we can plot all of the parameter samples vs. their arrival position along the brush path. Next, to assign a distinct parameterization to all point along the brush path, we need only fit some continuous function to this data.

# Optimization: Competing Goals



Fitting requires that we identify an objective function for coherence. However, one of the results of this work was the observation that there is no single coherence goal suitable for all situations.

First, consider the dotted lines used to stylize this mechanic part. Dotted lines are image-space features. We want the dot spacing to remain constant in image-space. Notice how the dots maintain their spacing even when the object is foreshortened.

On the other end hand, consider thorny features on this cactus. These are object-spaced features. We expect them to appear stuck to the object like geometry. However, if we employ the image-space coherence goal. Notice that in order to maintain image-space density the thorns slide over the surface as the cactus foreshortens. So image space coherence is not suitable for object-space features.

Instead, object-space coherence would have features appear stuck on the silhouette. Notice how the thorns remain in place under foreshortening, allowing density to change. Finally, if we apply the object-space coherence goal to image-space features the density of the dots distorts in an inappropriate way.

In short, our goal for how to fit those samples depends on the nature of the stylization, and typically, one would choose to strike some balance between the two forms of coherence.

# Outline Coherence Results

(VIDEO)

Let me show some video that reveals these effects in action.

# Summary

1. Stylized lines
2. Visibility of lines in 3D
3. How to specify stylization
4. Temporal coherence for stylized lines

Summary

# Part VII: Abstraction and Evaluation

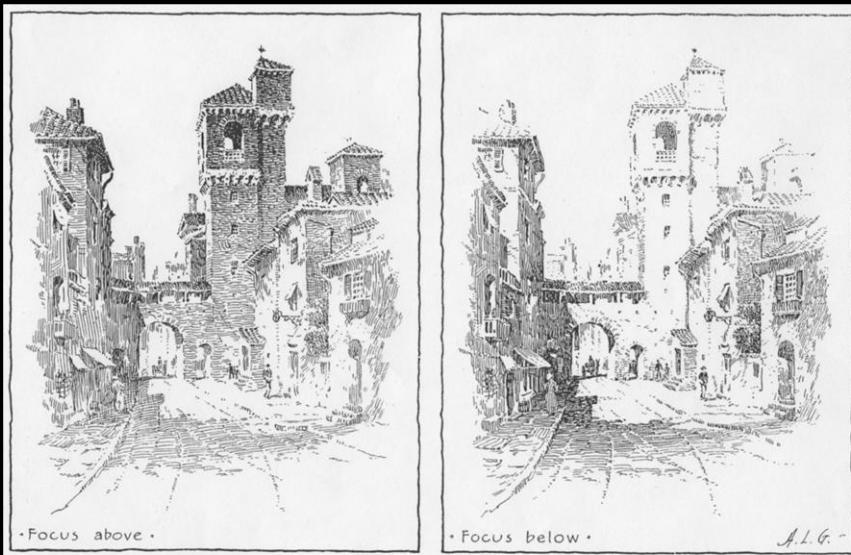
Doug DeCarlo

Line Drawings from 3D Models  
SIGGRAPH 2008

Now that you have an idea of how an artist can make a drawing,  
we can talk about how artists control what content you get out of their drawings.

I'll now talk about methods for achieving meaningful abstraction, and ways to assess how effective a drawing actually is.

# Abstraction



from "Drawing with Pen and Ink", Arthur Leighton Guptill

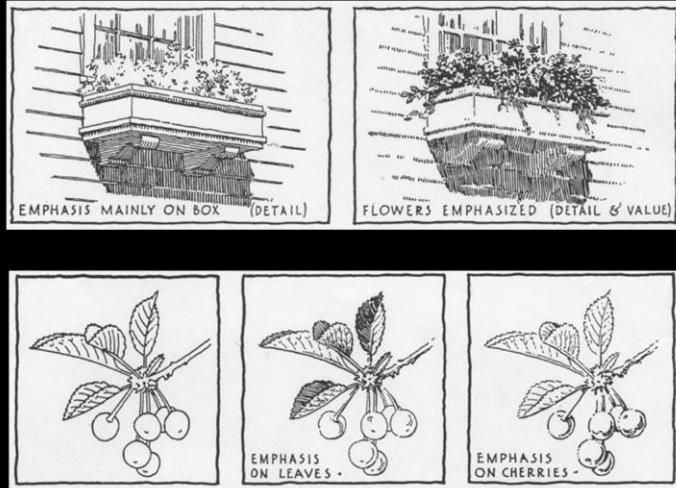
Artists can design effective imagery  
by changing or leaving out specific visual content.

The result of this process  
encourages particular interpretations for the viewer  
and enhances the viewer's understanding of the scene or  
situation.

This is the process of abstraction.  
It's a tool for effective visual communication.

Here, Guptill adapts the shading in this drawing  
to guide your attention to different parts of the scene.

# Abstraction



from "Drawing with Pen and Ink", Arthur Leighton Guptill

Artists often omit content,  
such as the detail on the flowers on the top left.

Contrast this with the detail Guptill included on the right  
in order to stress the flowers.

Same for the cherries here,  
where the focus can be on the leaves,  
on the cherries,  
or split between both.

There are a variety of means to do this.  
The particular visual style and medium  
determine the kinds of omissions and distortions that are  
possible.

# Abstraction



from "Line and Form", Walter Crane

In this example by Crane, for instance, the form of the elements of a scene are very different between these two different renderings.

Apart from selectively including various scene elements such as the clouds, we see how Crane adapts the details in the shape of the distant trees and buildings.

However, their rough shape remains so that we can easily identify these objects.

# Abstraction in NPR

## Automatic approaches

- Models of image salience can make predictions about what content is important



[Colломосе 2002]

How does this work in NPR?

There are a range of possibilities.

The most important difference comes from how the important content is selected.

But techniques also differ in how they go about retaining or removing content, given a particular visual style and medium.

Let's start with NPR approaches that work from photographs.

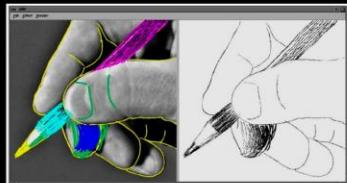
For painterly rendering, a fully automatic approach might attempt to preserve important content as determined by a computational model of image salience, which predicts how noticeable particular parts of the image are.

When such predictions select the important content, this is quite effective.

# Abstraction in NPR

## User guided approaches

- the user explicitly marks the important content



[Durand et al. 2001]



[Hertzmann 2001]

Another approach to producing effective artistic renderings is to have a user explicitly mark important regions of a picture.

The understanding is that these regions of the picture will be rendered with finer detail.

# Abstraction in NPR

Rendering specific content: trees

- automatically leave out lines in the center of the tree



[Kowalski et al. 1999]



[Deussen 2000]

In 3D scenes, the important content is located based on where it is in the world.

However, in restricted domains where the content is known, such as the rendering of 3D models of trees, heuristics can be applied that create effective omissions.

Here, detail in the center is left out as the tree is drawn smaller.

# Abstraction in NPR

Indication in pen and ink illustration

- the user specified what content was important



[Winkenbach and Salesin 1994]

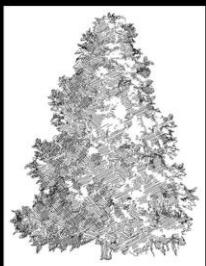
In more general domains,  
a user must specify the important content, just as they  
do in images.

This was the case in Winkenbach's system  
for pen and ink illustration,  
where the process of texture indication  
was guided by a set of marks drawn by the user.

# Abstraction in NPR

Select elements based on density and clutter

- drop strokes in areas of high density



[Winson and Ma 2004]



[Grabli et al. 2004]

Working with 3D scenes presents a new problem since the same objects can be viewed at various distances by simply moving the camera.

Sometimes important information just can't be included without causing clutter.

Thus, one approach is to explicitly measure the density of lines in a potential rendering.

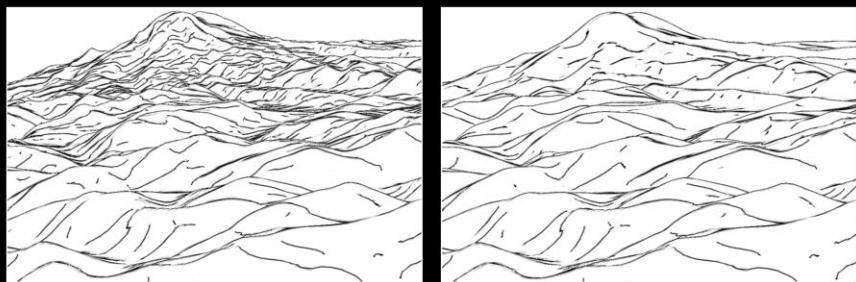
Some approaches work by using models of clutter effectively, alongside methods for prioritizing content.

However, it's difficult to get this working in animation without introducing temporal artifacts.

# Abstraction in NPR

Build a multi-resolution representation

- select mesh based on viewing distance



[Jeong et al. 2006]

Another possible approach echoes work on level-of-detail.

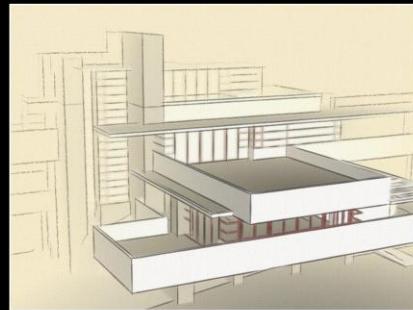
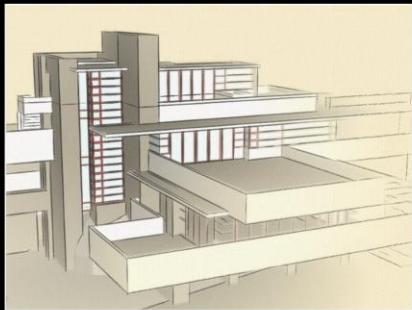
Build a multi-resolution representation of a scene,  
and select the scene elements based on viewing distance.

This simple approach **avoids** clutter,  
although without any real guarantee.

# Abstraction in NPR

Provide control over point of emphasis

Control clutter in the rendered image



[Cole et al. 2006]

It's possible to combine both of these ideas,  
which controls emphasis and manages clutter.

Forrester will be going into the details of such a system.

# Abstraction in NPR

## User guided approaches

- infer important content from a user's eye movements
- evaluate using eye tracking [Santella and DeCarlo 2004]



[DeCarlo and Santella 2002]

Here is an example from my work with Anthony Santella.

A photograph is transformed into a stylized version  
which consists of black lines  
and uniformly colored regions.

The interaction with the user is minimal:  
they simply look at the photograph for a short period of time.

A recording of the user's eye movements  
provides the information required  
to perform meaningful abstraction.

For the rest of this talk,  
I'll be explaining why this is a reasonable approach,  
and how we evaluated the effectiveness of this system.

It's all about how our attention shifts and how our eyes move.

# Eye movements

Eyes dwell on particular locations during *fixations* ○

- Quick motions between these locations are made via *saccades*
- Longer fixations indicate viewer interest



Our eyes are constantly moving.

\*\*\*\*

Here is an example recording.

Several times each second our eyes undergo rapid motions known as *saccades*.

These are punctuated with stabilizing motions known as *fixations*, where our eyes are held fixed over a particular location.

It has been demonstrated through a range of psychological studies that longer fixations indicate **INTEREST** on the part of the viewer.

# Eye movements

Recorded using commercial eye-trackers

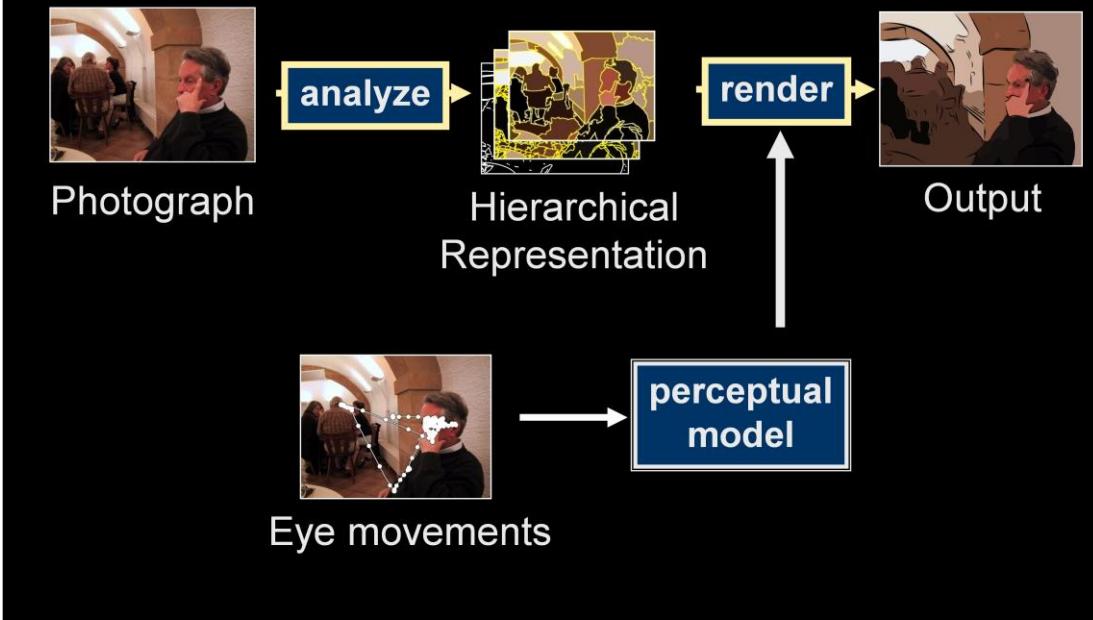


We can record eye movements using commercial eye trackers, such as this one.

So we can indirectly record  
the content of interest for a particular viewer.

# Abstraction and Stylization

[DeCarlo 2002]



Our system starts with photograph,

\*\*\*\*

decomposes it into a hierarchy of visual elements,

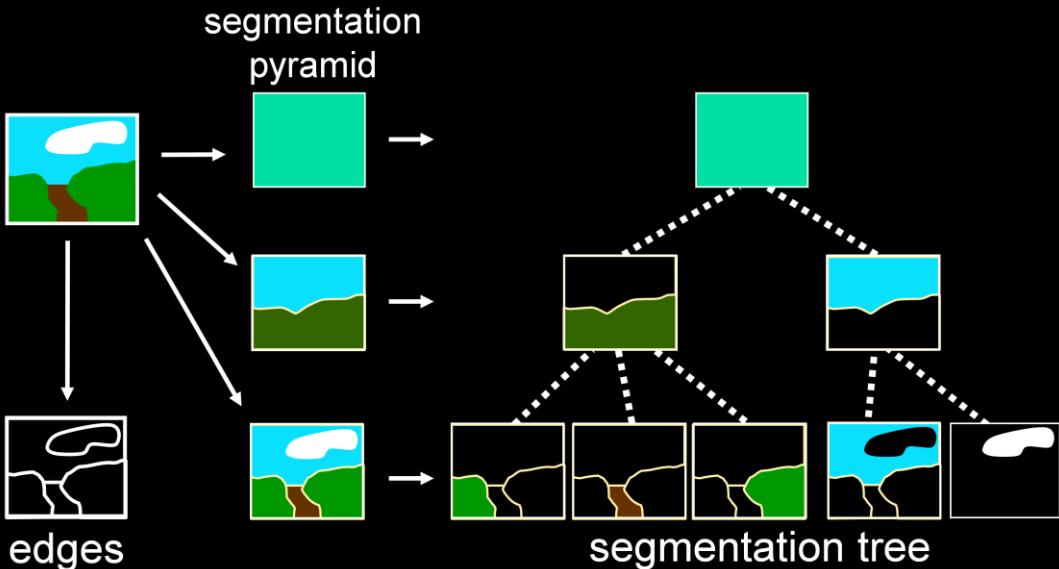
\*\*\*\*

and renders a subset of these elements into an output image.

\*\*\*\*

The features to render are selected by a perceptual model  
that draws upon a recording of a viewer's eye movements.

# Image analysis



Our image analysis starts with edge detection,

\*\*\*\*

and a set of image segmentations performed at a range of resolutions.

\*\*\*\*

Finer scale segmentations contain more detail.

\*\*\*\*

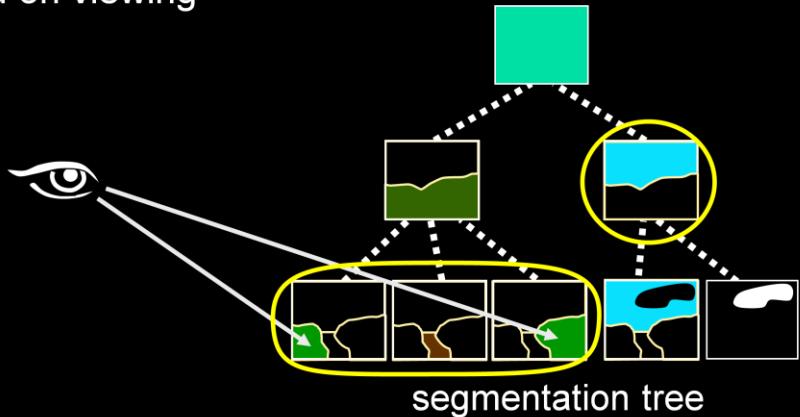
We build a hierarchical segmentation by inferring containment relationships between regions across resolutions.

This is our image representation.

# Rendering: regions

## Prune segmentation tree

- perceptual model decides what to remove based on viewing



We then can prune the segmentation tree \*\*\*  
based on predictions made by the perceptual model.

The perceptual model decides which of these visual elements will be included in the result.

It does this based on the size and local contrast of each visual element, and for how long it was fixated.

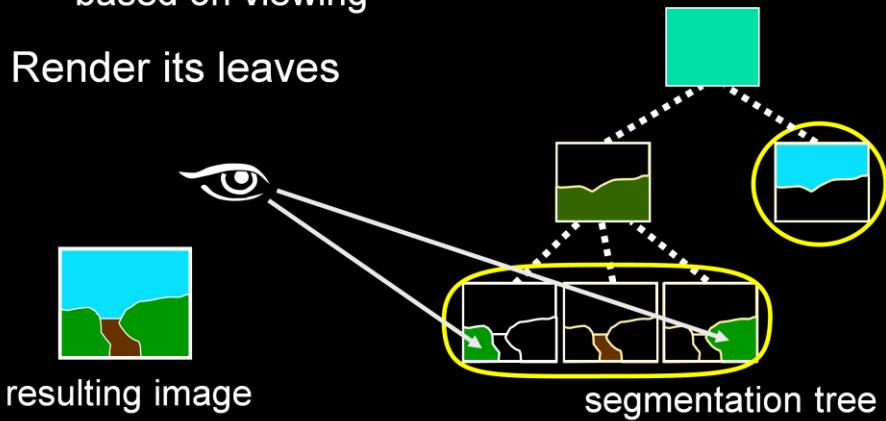
In short, noticeable regions are more likely to be preserved, particularly when the viewer examined it closely.

# Rendering: regions

Prune segmentation tree

- perceptual model decides what to remove based on viewing

Render its leaves

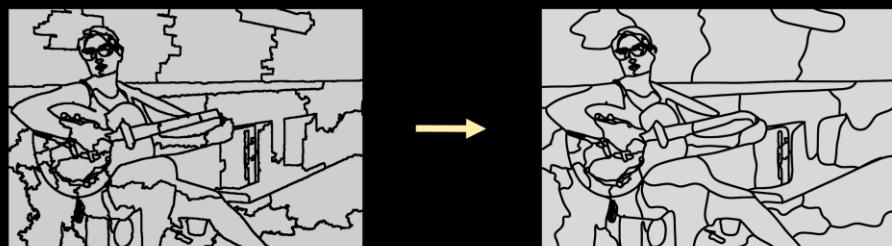


So anything that the viewer probably didn't notice will be removed.

We render the leaves of the pruned tree into the output image.

# Rendering: regions

Smooth boundaries based on their size



We also smooth these regions  
so that the detail in their boundaries reflect the appropriate  
scale.

Larger regions are smoothed more.

# Rendering: lines

Select edges using perceptual model

Smooth lines by a fixed amount

- lines and regions don't line up exactly
- seems to convey "sketchiness"

Line thickness

- increases with length
- tapers over first and last third



The lines are also selected using a perceptual model.

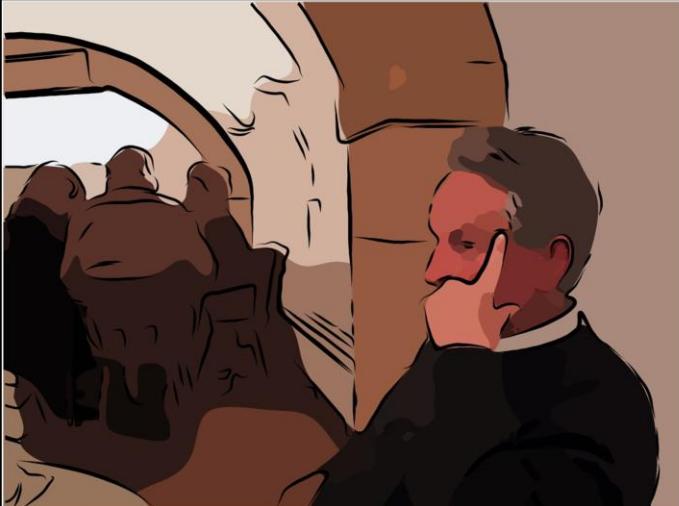
They are smoothed by a fixed amount.

This means that they won't always line up with the regions.

In areas where the regions are heavily simplified,  
this misalignment will be larger.

The result of this is a sketchy look where detail was  
removed.

## Results...



The final picture is made  
by overlaying the lines on top of the regions.

Here, the small photograph on top is the original,  
below are the users fixations, each white circle is a fixation,  
where its size indicates fixation length.

The scale at the lower left is one second.

The foreground figure is clear,  
as the viewer examined that location.

Figures in the background have had most of their detail  
removed.

# Without eye movements: No meaningful abstraction

One knob to control detail...



more detail



less detail

You can compare this to results  
where we don't use eye movements,  
but instead a global control for detail.

With high detail, the background looks distractingly fussy.  
With less detail, important features such as the face are lost.

# Evaluation

How do we measure success?

- Possibility 1:
  - measure performance using images in a specific task  
i.e. [Gooch 2004, ...]
- Possibility 2:
  - measure cognitive activity required to process display  
[Santella 2004]

We can assess whether these images achieve meaningful abstraction.

One approach to evaluating imagery is to measure performance in a particular task.

Another is to measure the activity or effort required of the user.

A commonly used approach in evaluating an interface measures how much the user must move the mouse while performing some task.

It's not a performance measure, but rather an indirect measure of effort.

We take this approach, and a natural activity to measure is eye movements.

As briefly mentioned before, we know eye movements reflect viewers interests and goals.

Because of this link to cognition, they've been used in the past to evaluate complicated visual displays that must provide efficient access to information.

# Evaluation via eye tracking

Expect viewers to concentrate interest on  
emphasized regions in our renderings

Measure and compare fixations on viewing:

- original image
- what else?

We hope to find that, for our images,  
viewers concentrate more on the areas  
that were highlighted with increased detail.

To test this we can compare fixations  
over these images to those on the original photograph.

But there are some other interesting possibilities.

# Uniform detail

One knob to control detail...



more detail



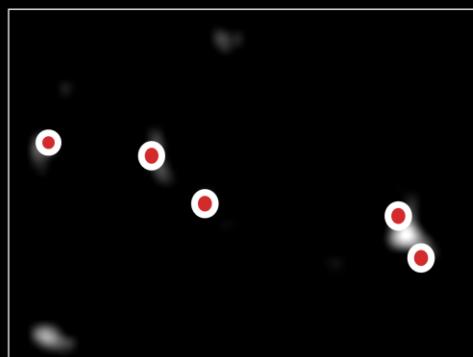
less detail

One possibility is uniform detail control, which uses a global threshold in place of eye movement recordings.

# Fully automatic abstraction

- Salience maps pick locations of potential interest based on low level qualities

[Itti 2000]



Another approach is to choose locations for increased detail automatically.

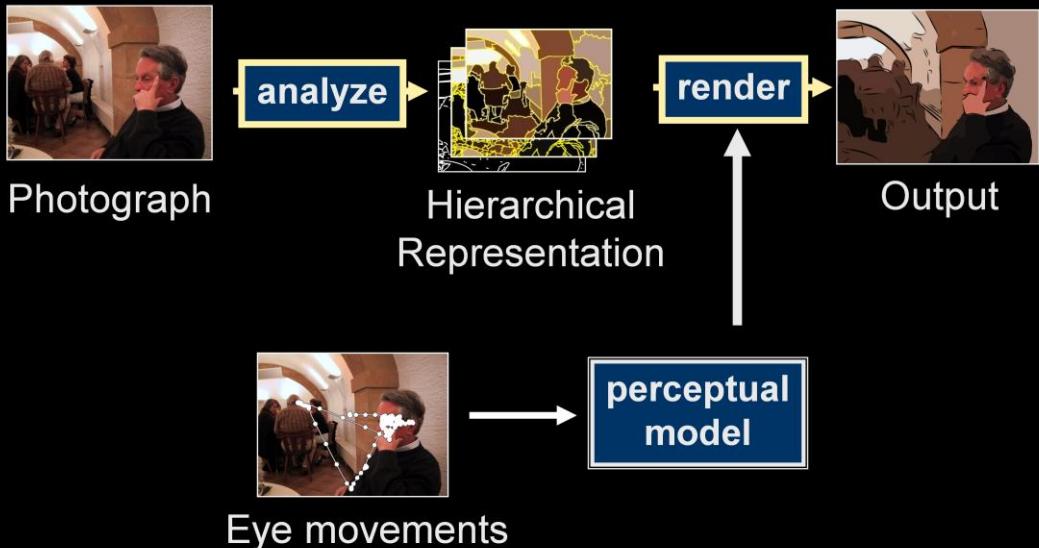
Methods for predicting salience combine a number of filters to create \*\*\*\* a map of feature contrast for an image.

Bright areas are potentially interesting, and algorithms can use them to pick a set of locations completely automatically. \*\*\*\* So like eye tracking, the output of the salience method is a series of points to be rendered with increased detail.

In the upcoming discussion, I'll refer to both fixations and salience points used to control detail as DETAIL POINTS.

# Abstraction and Stylization

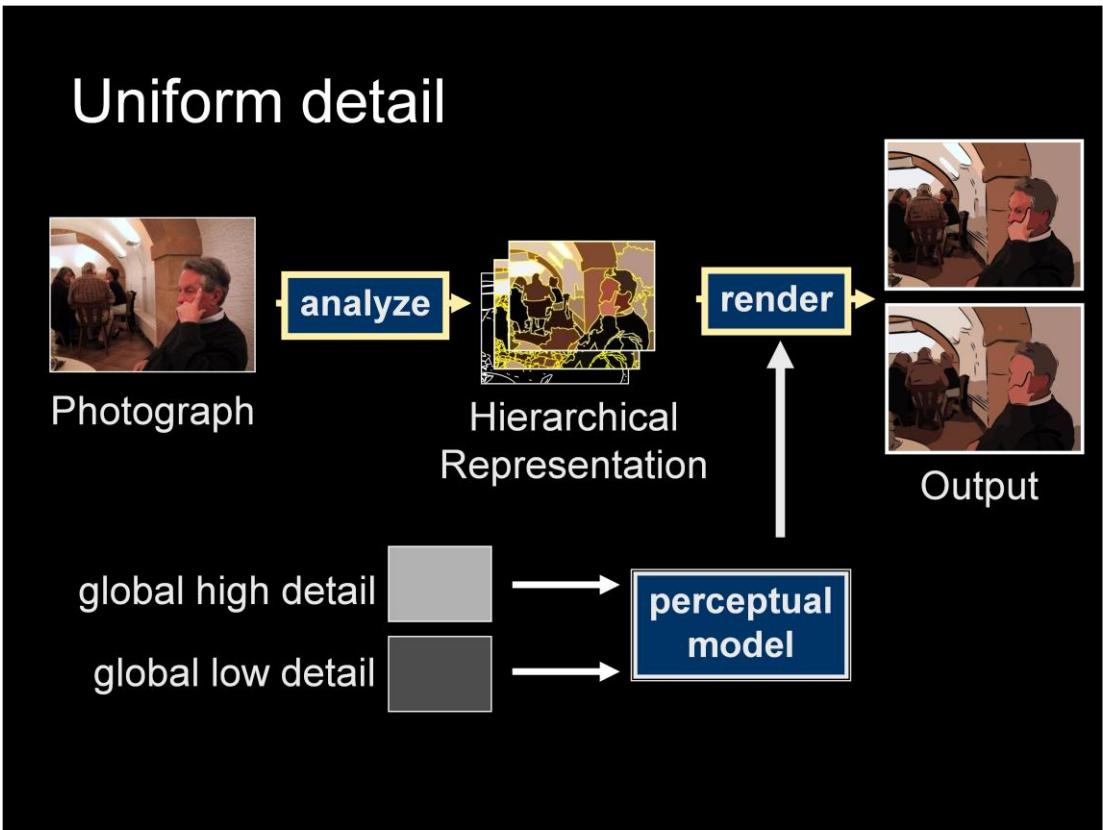
[DeCarlo 2002]



Here again is the design of our system.

Eye movements are input to the perceptual model.

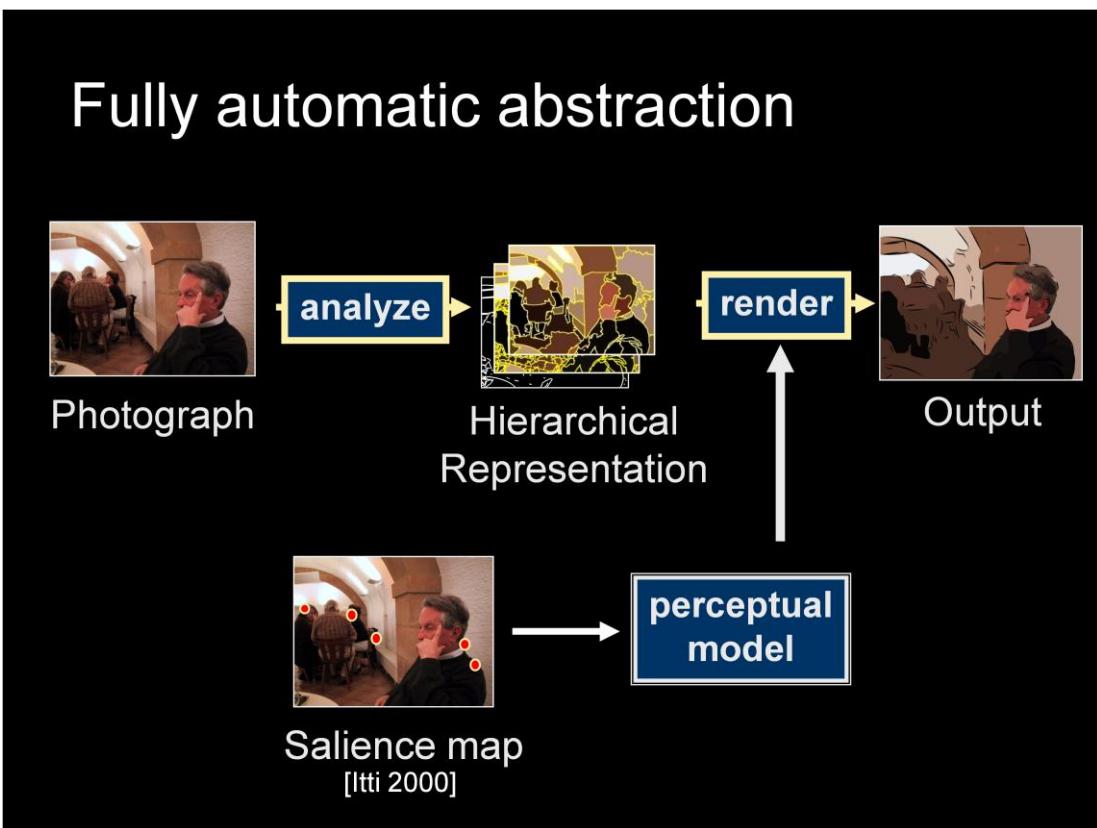
# Uniform detail



Uniform detail uses a global weight to control detail across the image.

We do this for two levels of detail: low and high.

# Fully automatic abstraction



Finally, a model of salience developed by Itti and Koch determines the visually distinctive content.

This model selects a set of low-level image features that might catch your attention.

# Variations of images



Here are the five conditions.

We show each subject in our experiment one of these pictures.

We do this for 50 different photographs.

# Variations of images

This set of images separates:

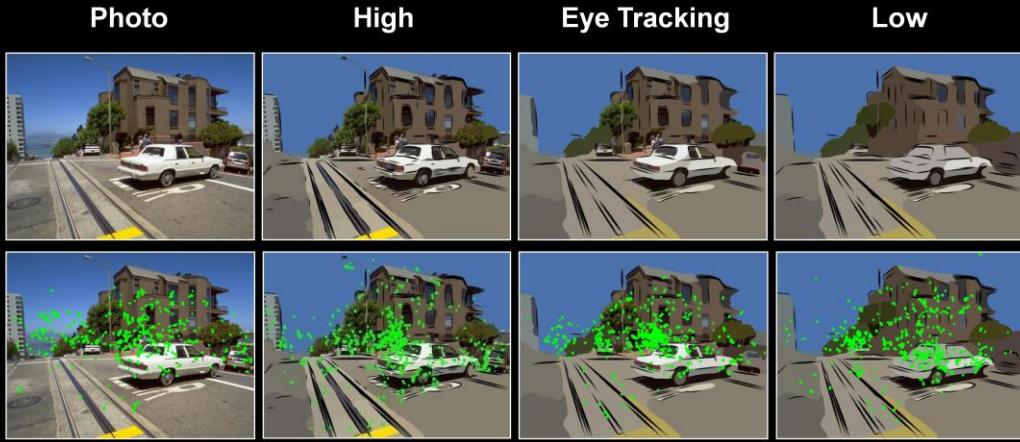
- the visual style
- global level of detail
- locally increased detail
- the locations of increased detail

These five images will let us distinguish between different hypothesis regarding how people examine images with modulated style and detail.

Specifically, we can analyze the effects of style, and of controlling detail both globally and locally.

# Example data

(collected together per image)



How to measure these differences?

The result is data like this.

This combines 10 viewers of each image together  
for 4 of the conditions.

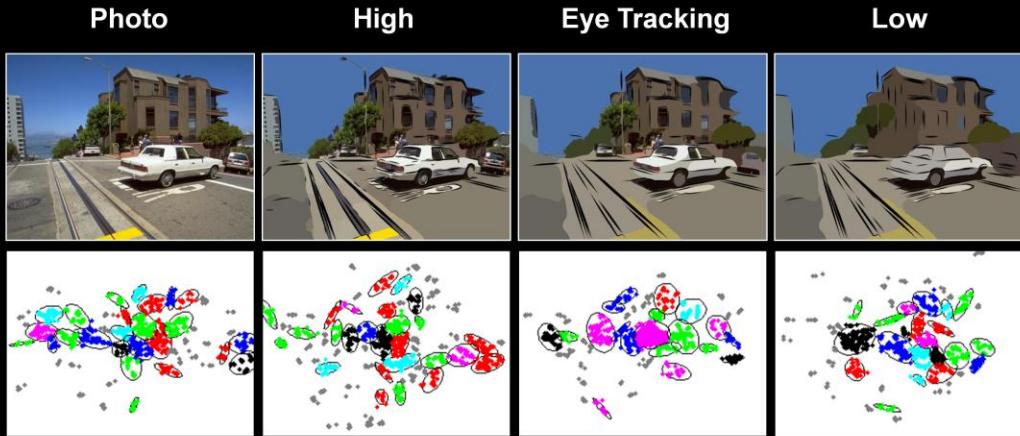
We also analyzed the data individually for each viewer,  
but here I'll only talk about results of analysis  
collapsed over viewers.

We can see differences in distribution of data across  
conditions.

But how can we quantify them?

# Example data

(collected together per image)



How to measure these differences?

- clustering

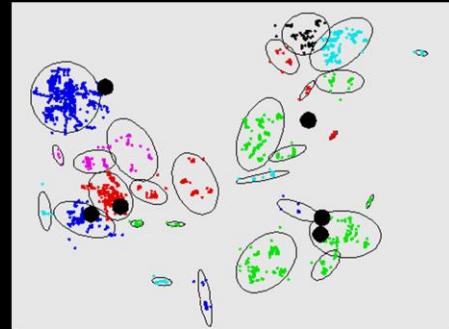
Our approach is to cluster the data.

To divide it into limited regions of the image  
that were viewed coherently.

# Quantitative analysis

Look across conditions at:

- number of clusters
- distance to locations where detail was preserved



Once we've cut the data up into clusters,  
we can compare the number of clusters,  
and the distance from cluster centers to the detail points.

One set of detail points is marked here  
with black circles.

## Results: Number of clusters

Both *eye tracking* and *salience* have significantly fewer clusters than *photo*, *high* and *low detail* (10-20% fewer)

–  $p < 0.001 \dots 0.05$

*Eye tracking* has significantly fewer clusters than *salience* (about 10% fewer)

–  $p < 0.001$

When examining the number of clusters, we found that modulating detail holds viewer interest moreso than uniform detail.

We found that viewers examine 10 to 20 percent fewer locations when the detail is modulated.

This effect is larger when the detail is meaningful.

## Results: Distance to detail points

Clusters of interest are closer to the detail  
points when using *eye tracking or salience*  
–  $p < 0.0001$

When measuring the distance from cluster centers to detail points, we can verify that people were in fact looking in the right places.

This might seem just like what you'd expect given what we know artists do.

But it wasn't clear that this is what we'd find.

It might have been that these techniques simply cannot capture how artists can guide our attention, as there are certainly many other tools that artists use to do this same thing.

## Implications for NPR

Meaningful abstraction is important

- style alone is not enough
- global detail control is not enough
- low level salience is not enough

Use eye tracking to evaluate and understand NPR

In summary,

we find that in achieving meaningful abstraction,  
altering the style is not enough,  
globally changing the detail is not enough,  
and using heuristic measures of importance is not enough.

We need to locally adapt the detail in a meaningful way.

We're also encouraged by the use of eye-tracking  
to evaluate the effectiveness of NPR displays.

# Summary

## Abstraction

- omitting and adapting content
- get importance information from a user/artist

## Evaluation

- eye movements provide one way of assessing abstract displays

# Part VIII: Controlling Detail and Attention

Forrester Cole

Line Drawings from 3D Models  
SIGGRAPH 2008

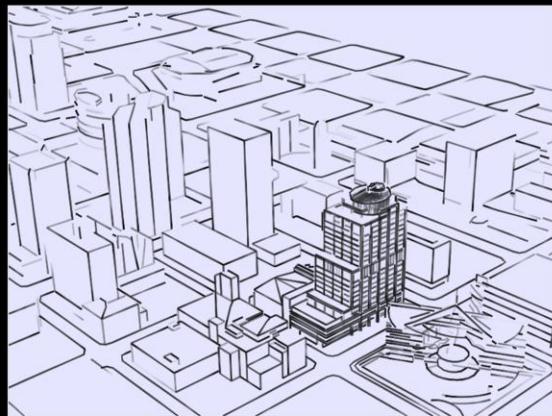
## Abstraction in 3D

- Want everything from 2D, and more
- Temporal coherence
- Performance
- Flexibility

In three dimensions, we would like the same control over abstraction as in two dimensions, but there are several additional requirements. We'll focus on animated and interactive applications, since if we only care about still images, we can just use the techniques Doug described. For animation, we need temporal, or frame-to-frame coherence between consecutive frames. For interactive applications, we need both temporal coherence and good performance. A desirable, though not strictly necessary, quality is flexibility in the types of lines that the method can handle.

# Line Density

- Line density important cue for abstraction
- Basic pipeline lacks control of line density



Line density is one of the most important cues for abstraction in line drawings. In the image shown, the central building is obviously emphasized while the surrounding buildings are not, and the only variation between the two is the line density. However, the basic pipeline described previously has no natural control over screen space line density - lines are drawn wherever the extraction and visibility algorithm determine they should be.

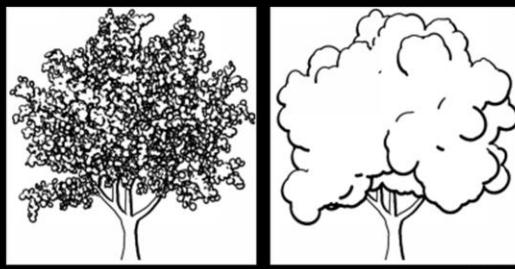
# Strategies for Density Control

- Model based
  1. Make abstraction of 3D model
  2. Extract and draw lines from abstract model

The simplest way to deal with inordinate line density is to nip it in the bud by creating an abstraction of the model itself, such as by smoothing. The lines are then extracted from this smooth model, and rendered as normal.

# Abstraction of Trees

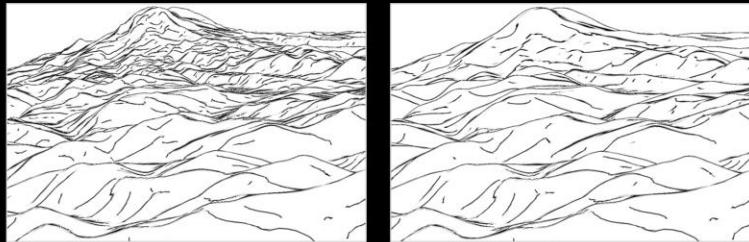
- Represent leaves with disks
- Vary disk radius to control abstraction
- Good temporal coherence



So, how do you make an abstraction of a model? One way is to create a specialized method tailored to a specific type of model, such as bushy trees. In this example, we represent each leaf with a disk, and vary the radius of these disks to control the level of abstraction. This method has the nice effect of not only controlling density, but also varying the shape of the lines themselves.

# Multi-Resolution Meshes

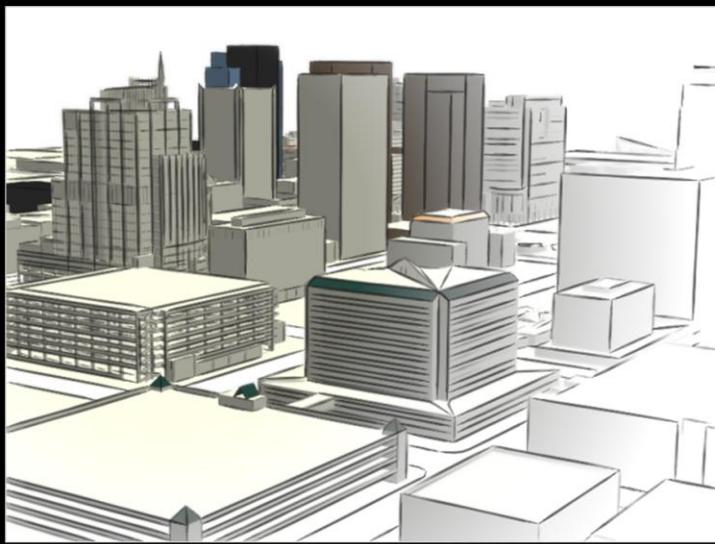
- Create a set of filtered meshes
- Blend meshes based on viewing distance
- Extract lines from blended mesh
- Only works for smooth shapes



[Jeong 2006]

More generally, we can create a set of simplified meshes using some low-pass filtering method. We then interpolate between these meshes at runtime, giving precedence to the smooth mesh in areas where low density are desired, and the detail mesh for areas of high density. The major issue is that this method is restricted to smooth models that can be effectively filtered without losing their character.

## General Shapes



If you have shapes that cannot be filtered easily, such as these buildings, or worse, have a set of lines built in to the model itself (such as the lines on the building in the foreground), then its difficult to use a model based strategy.

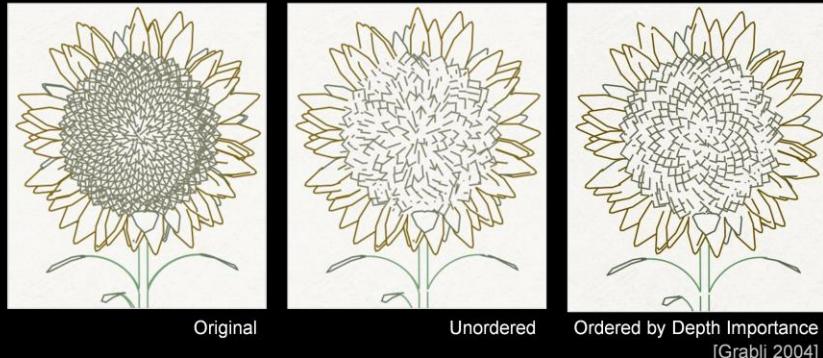
# Strategies for Density Control

- Model based
  - 1. Make abstraction of 3D model
  - 2. Extract and draw lines from abstract model
- Stroke based
  - 1. Extract lines from full 3D model
  - 2. Manipulate density in image space
  - 3. Draw reduced set of strokes

A stroke based strategy can be more general because it doesn't depend on changing the model. The density control happens in image space, after the lines are extracted and turned into 2D strokes. The source of the lines is not important, be it extracted silhouettes and suggestive contours or manually added decorative line, because they all get turned into 2D strokes.

# Stroke Based Control

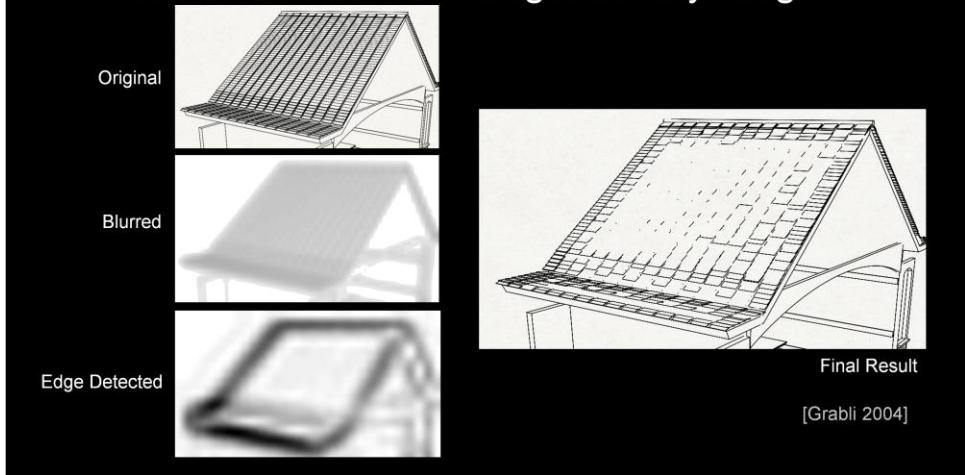
- Compute line density as strokes are drawn
- When density reaches threshold, stop drawing
- Order of strokes is important



The simplest way to control line density in image space is to keep track of the density as each new stroke is drawn. The density function is usually defined as a smoothed version of the drawing in progress. When the density reaches a user-specified threshold, the remaining strokes in that area are dropped. The strokes must be ordered somehow by importance, as a random ordering will cause undesired results such as in the middle image. Ordering the strokes by an importance metric can improve the results by making sure the more important strokes (here, the strokes that lie along large depth discontinuities) get drawn first.

# Image Space Tricks

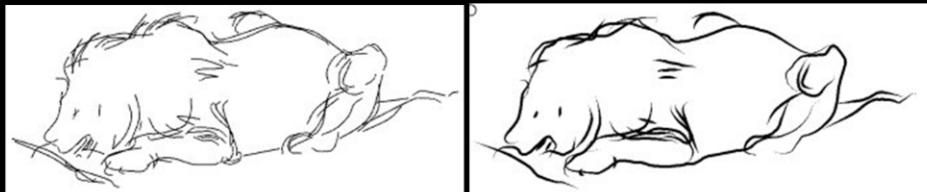
- Render normally, filter result
- Use filtered version as target density image



There are a number of tricks that can be performed in the image domain. One is to simulate indication by drawing strokes more densely around the edges of the object. This effect can be achieved by first drawing the entire scene and blurring it to get a smooth version. The smooth version is then run through an edge detector, which produces an image with smooth dark areas around the edges of the shape. This image can then be used as a target density image to locally control the stroke density across the image. Unfortunately, dropping strokes at an arbitrary threshold generally leads to terrible temporal coherence. Superior results can be achieved by smoothly merging strokes instead of dropping them.

# Stroke Simplification

- Simplify strokes instead of dropping
  - Cluster strokes into sets
  - Replace with “representative” stroke



[Barla 2005]

Techniques to merge dense strokes are generally called stroke simplification techniques. The general idea is to identify a set of similar, nearby strokes, merge them into a cluster, and then replace the cluster with a single representative stroke. The subtlety arises from the problem of choosing a representative stroke. Sometimes long, curly strokes (such as the stroke near the lion's shoulder) need to be broken into a series of strokes.

# Stroke Simplification

- Naively, at least an  $O(n^2)$  problem
- Can be improved with proper datastructure
  - “ $(1+\epsilon)$ -deformable Spanner”
  - Roughly  $O(n)$  for reasonable drawings



[Shesh 2008]

A naive algorithm for clustering strokes is at least  $O(n^2)$ , since every stroke needs to be compared to every other stroke to find the closest matches. Shesh 2008 proposes to use an advanced datastructure called a  $1+\epsilon$  deformable spanner, which is essentially a graph structure that allows finding all pairs of nearest points in roughly linear time for reasonable graphs. Shesh demonstrates the system at interactive framerates with good temporal coherence. However, the implementation of the system is somewhat complex due to the use of the spanner structure.

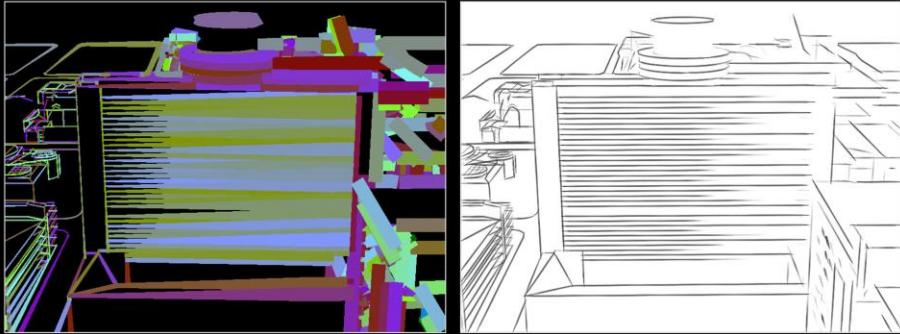
## Priority Buffer

- Yet another density control algorithm
  - Inspired by item buffer
- Good temporal coherence
- Stroke based
- Relatively simple
- Fast

The final line density control scheme that I will mention is the priority buffer, which is inspired by the item buffer Adam mentioned before. The priority buffer test has several important benefits. It has temporal coherence suitable for animation. It is stroke based so it handles all types of lines, with the caveat that the priority ordering must be well defined. It is also fast and relatively easy to implement, especially if you are already using an item buffer to compute line visibility.

# Priority Buffer

- Lines ordered by priority instead of depth
- Wide, high priority lines cover low priority lines
- Final weight proportional to visibility in buffer

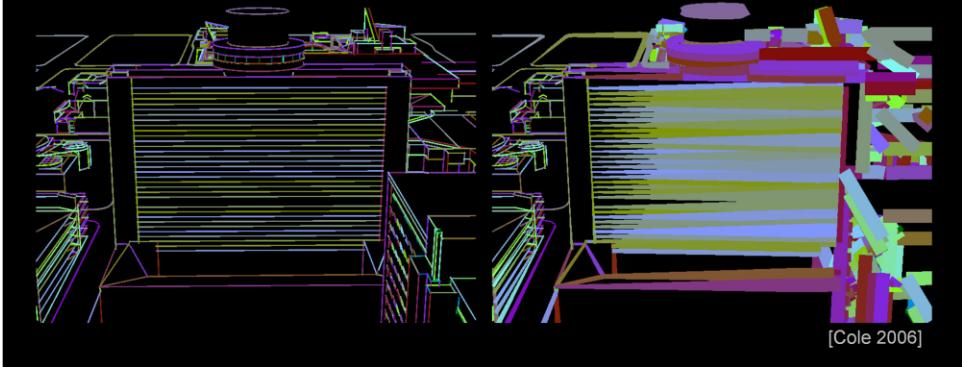


[Cole 2006]

The priority buffer is a second offscreen buffer, where lines are ordered by a priority value rather than depth. We locally vary the width of the lines so that the lines are thin where line density should be high, and wide where line density should be low. The effect is that in areas of low density, wide, high priority lines cover low priority lines and obscure them. We vary the final weight of each line according to its visibility in the priority buffer. So for example, the lines on the left here are thin, corresponding to high density in the final rendering, and the lines on the right are wide, corresponding to low density.

# Item Buffer vs. Priority Buffer

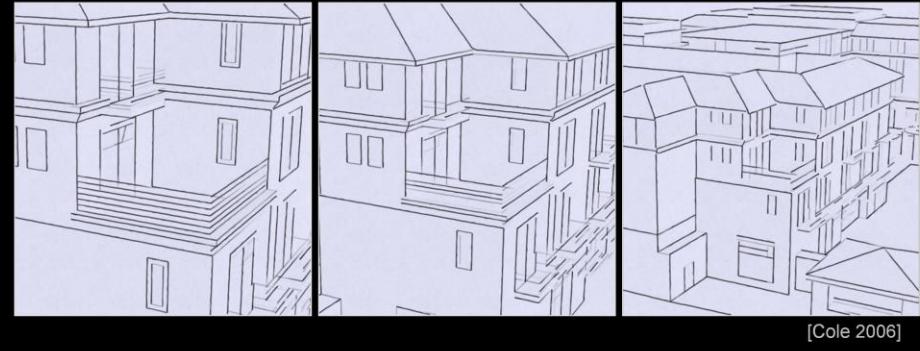
- Visibility and priority tests are similar but distinct
- Ordering: depth vs. priority
- Line thickness: thin (1-3 pixels) vs. very wide



The item buffer and priority buffers are similar, but they are necessarily separate. Of course, the ordering of the lines is by depth in the item buffer and by priority in the priority buffer. Somewhat more subtly, however, lines in the item buffer should be as thin as possible while avoiding rasterization errors. If lines are drawn wide in the item buffer, the visibility test will be inaccurate and we will get haloing and other artifacts. The priority buffer buffer needs wide lines, but it also needs accurate visibility to function, so we have to perform the visibility test first followed by the priority test.

# Priority Buffer Drawbacks

- May limit artistic style
  - Varies line weight continuously
- Dependent on good priority function



[Cole 2006]

This method also has some drawbacks. To achieve the temporal coherence, we vary the line weight continuously, but this can look odd for styles such as pen and ink.

Second, we need to have a good priority ordering. Currently, we use 3D line length for priority: that is, a line's priority is proportional to its length in world space, prior to any visibility clipping. This function tends to bring out long lines like the rooflines in the right picture. Our function can have problems where a large number of nearby lines have exactly the same length. You can see the problem in dense bars in the middle picture. In this case the priority is effectively random.

Of course, our priority function can be arbitrary, and we could choose other functions to, for example, prioritize silhouettes or lines with high depth discontinuity. For really high quality results the user could actually go in and manually set the priority in these kinds of situations.

## Beyond Line Density

- Line style
- Line texture
- Color saturation
- Color intensity



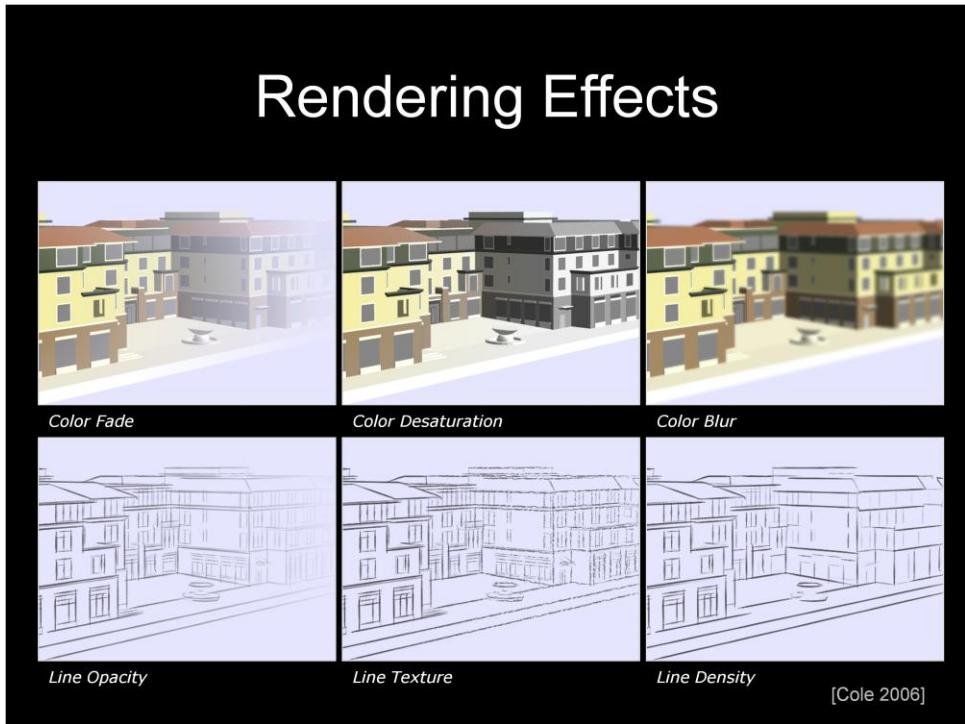
*Rembrandt*



*Winslow Homer*

Of course, density is not the only way to control abstraction in line drawings. Line style and texture, such as in the Rembrandt sketch, are important. Color saturation and intensity, while not strictly line drawing techniques, can also be very effective. In the Homer watercolor the region to the right is faded and desaturated, and therefore deemphasized, while the boat right next to it is emphasized with bold colors.

# Rendering Effects



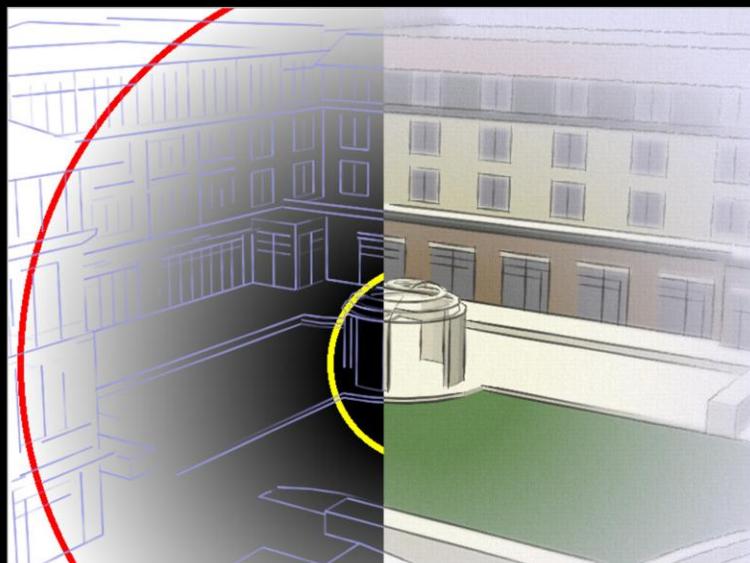
These are the main rendering effects that we use to control abstraction. They can be broken down broadly into color effects and line effects. For color, we can fade out the color away from the point of emphasis, desaturate the color, and blur the color. We can similarly fade out the lines, change the line texture, and locally adjust the line density using the priority buffer.

## Stylized Focus

- Analogous to photorealistic defocus
- Scalar focus value over 3D scene
- Focus based on distance from
  - 2D focal point
  - 3D focal point
  - Focal plane
  - Segmented object

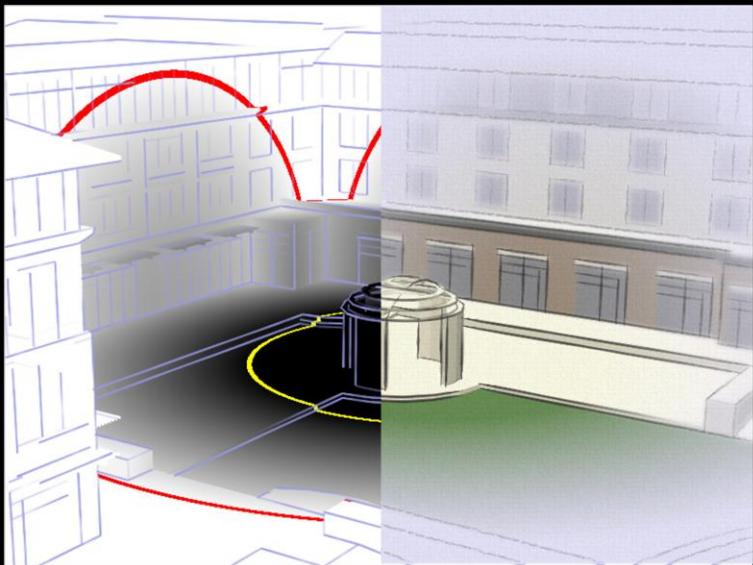
To control all these effects it is helpful to define a single scalar value, which we call stylized focus. We can define the focus value pretty much arbitrarily, so long as it is relatively smooth. In the 2006 paper, we define four methods: distance from a point in image space (2D), distance from a point in world space (3D), distance from a focal plane (meant to simulate camera defocus) and segmentation, where each model section gets a constant focus value.

## 2D Focal Point



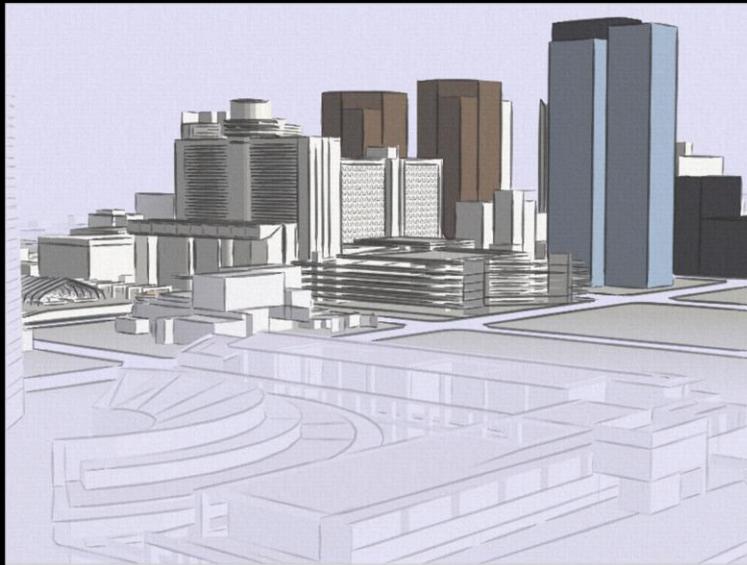
This is an example of the 2D mode, where the focus value falls off linearly from 1 inside the yellow ring to 0 outside the red ring.

## 3D Focal Point



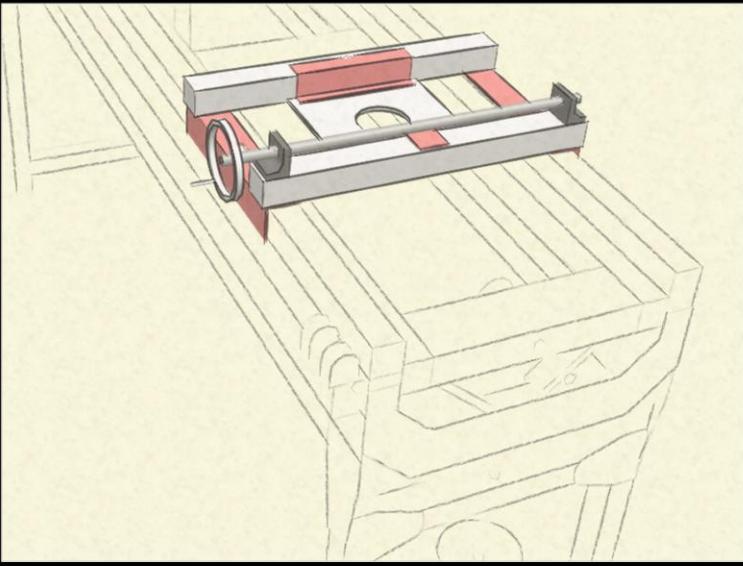
The same visualization for the 3D, world space method

## Focal Plane



The camera focal place method, where the focal plane is placed in the distance.

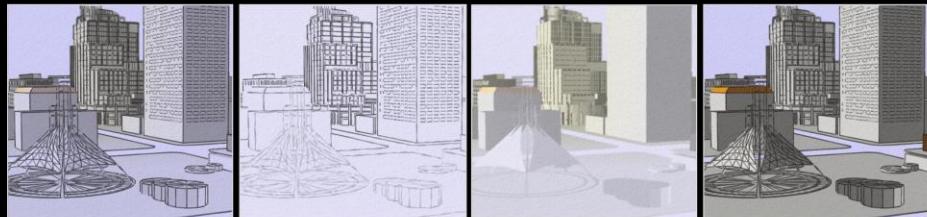
# Segmentation



Finally, a simple example of segmentation.

# Evaluating Effectiveness

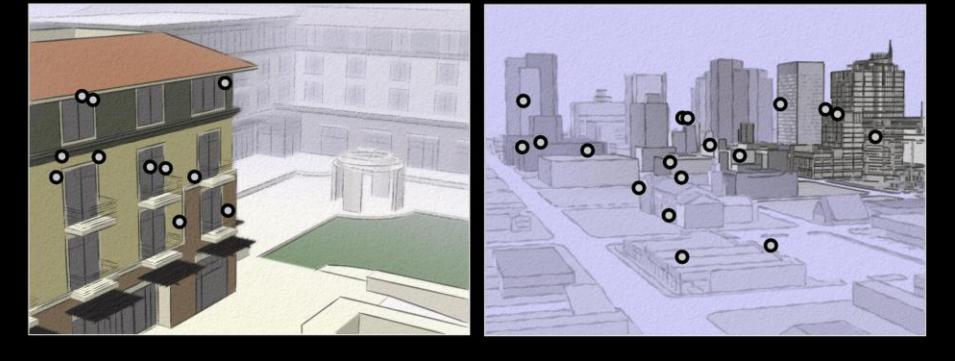
- Eyetracker study similar to [DeCarlo 2002]
- Scenes created to isolate effects
  - Color only, lines emphasized, color constant, etc.
- Compared with and without emphasis



Once we have created the stylized focus effect, a natural question to ask is how well it actually works. We ran a study using the same eye tracking hardware that Doug described earlier. The subjects were shown a range of scenes and effects. The scenes were created to isolate individual rendering effects, so for instance only colors or only lines, or both color and lines, but where only the lines are emphasized. We then compared the eye tracking data for the emphasized versions with unemphasized control images.

# Study Results

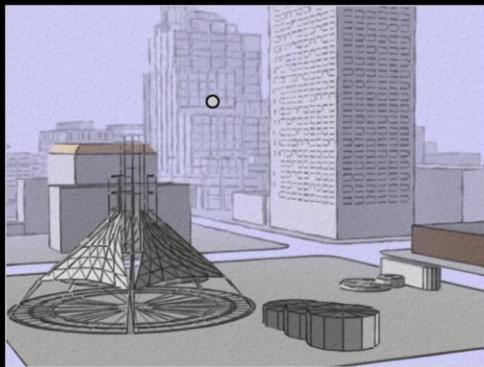
- Emphasized areas draw the viewer's gaze
- Effect exists with either color or lines
- Strongest with both color and lines
- Aids, but does not replace, composition



The results show that our emphasis effect does draw the viewers eye in all cases, though the effect can be small. For example, the left image shows a strong case, while the right image shows a weak one. The circles indicate eye fixations. The effect exists when either lines or color are emphasized alone, though it is strongest when both are used together. Composition of the scene is still important – the stylized focus effect does not prevent the viewer from looking at deemphasized areas, as shown by the right image.

# Limitations

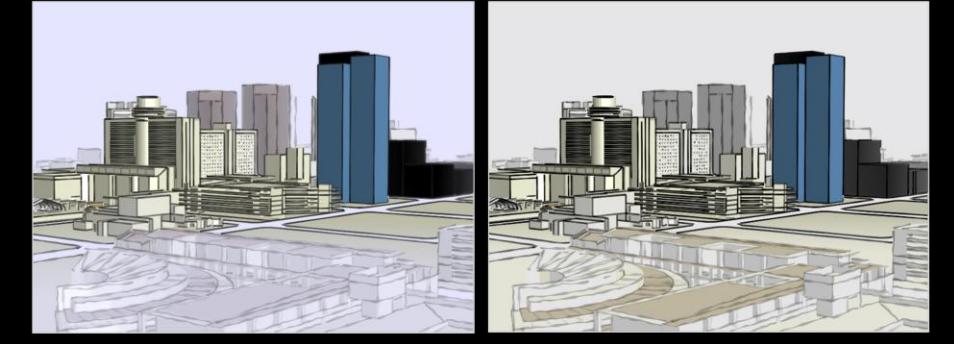
- No Feedback
  - Some areas require more deemphasis than others
  - No way to know if deemphasis is effective



This image shows an eye fixation on a heavily deemphasized object. We are never going to eliminate fixations in deemphasized areas. However, some areas (such as this skyscraper, with lots of line detail) need more deemphasis than others. We currently have no way to tell when this is the case. There are various algorithms for automatically calculating some kind of saliency map for the image. You could imagine running our algorithm, calculating a saliency map from the result, and using that saliency map to determine where further deemphasis was required, and then iterating until the emphasized areas had the desired saliency.

# Limitations

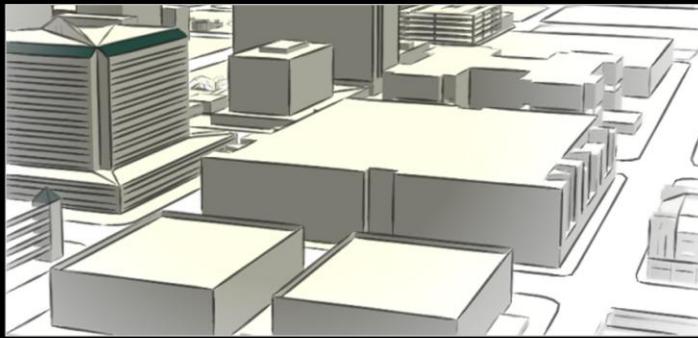
- Color effect often looks like fog
- Increasing prominence of lines helps
- Can try simplifying color (similar to [Barla 2006])



The color fading effect can resemble fog, and look bad in some cases. A better effect can be gained by simplifying the color without fading it out. Basically, we quantize the colors into a couple of desaturated buckets. This effect was also proposed by Barla in relation to toon shading.

# Limitations

- Only works if detail already exists
  - Can only “defocus”
  - No way to invent line detail



A big limitation of this, and most other abstraction approaches we examined, is that if there is a big area of the image without much detail cannot be effectively emphasized. Possible solutions to this include using some kind of hatching scheme to add line detail even where there is none originally, though hatching of course affects the tone of the image. This situation may also not arise very often, because it is rare that we want to draw attention to a big, blank area of the scene.

Thank You

Questions?