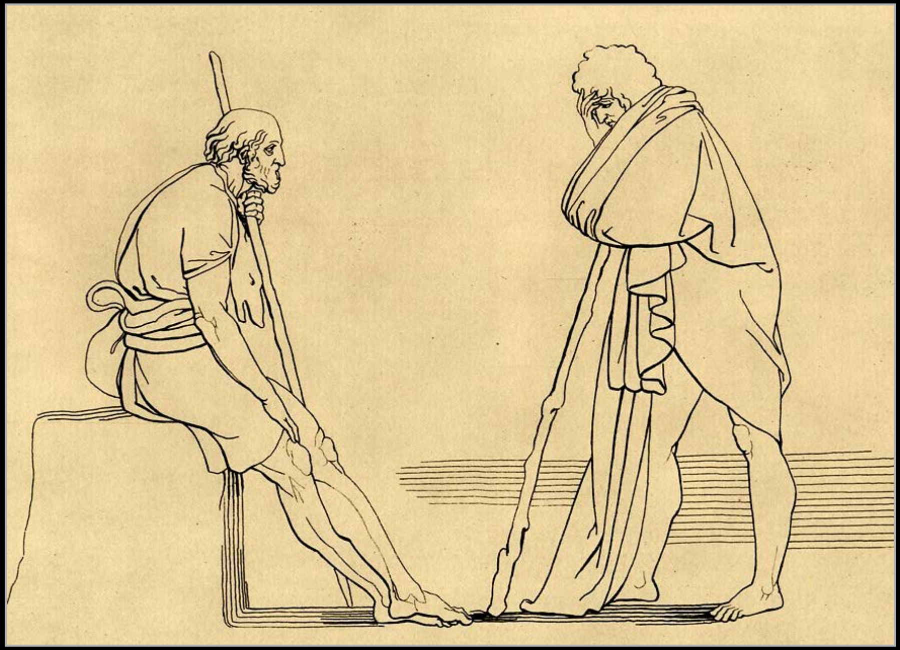
Differential Geometry and Line Drawing

COS 526: Advanced Computer Graphics

PRINCETON UNIVERSITY

How to Describe Shape-Conveying Lines?

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



[Flaxman 1805]

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, …]

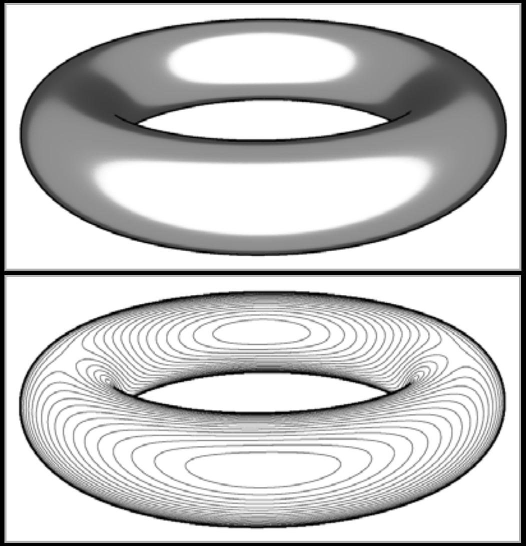
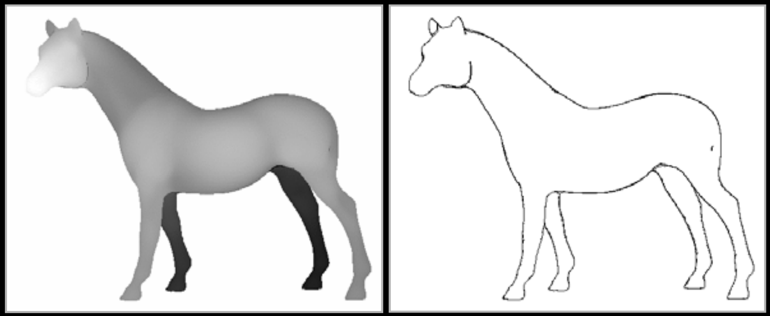


Image Edges and Extremal Lines

Edges:

Local maxima of gradient magnitude, in gradient direction



Ridges/valleys:

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

View-Independent Object-Space Lines

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

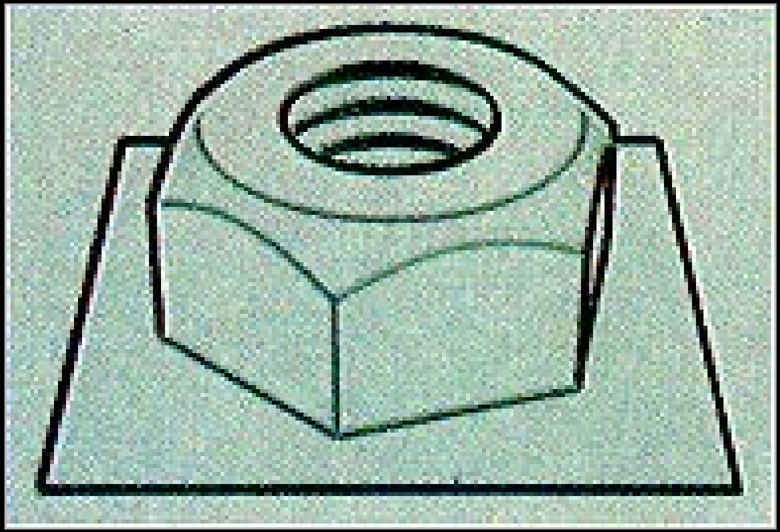
View-Independent Object-Space Lines

Topo lines: constant altitude

[USGS]

View-Independent Object-Space Lines

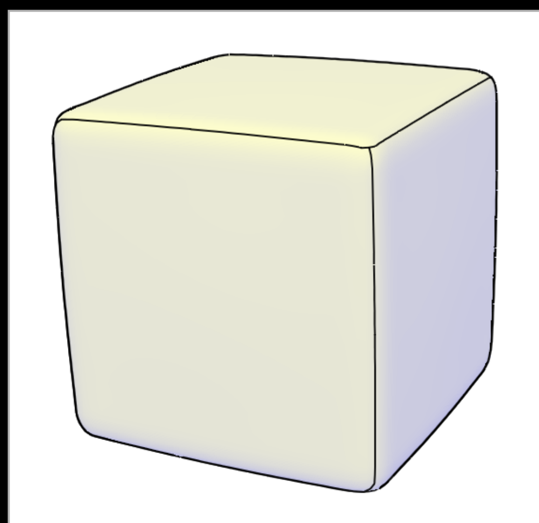
Creases: infinitely sharp folds



[Saito & Takahashi 90]

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, Ohtake 04 …]

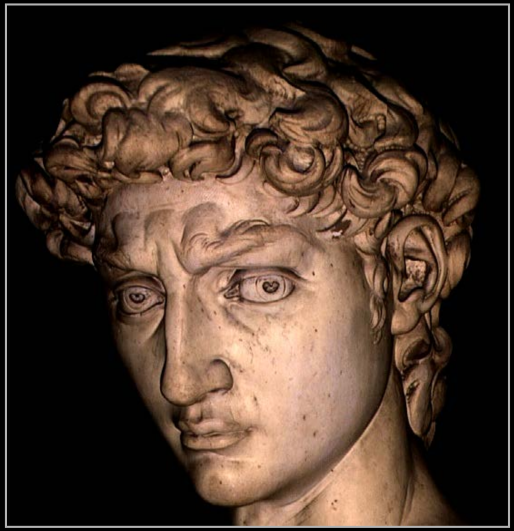
View-Dependent Object-Space Lines

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

What Lines to Draw?

Silhouettes:

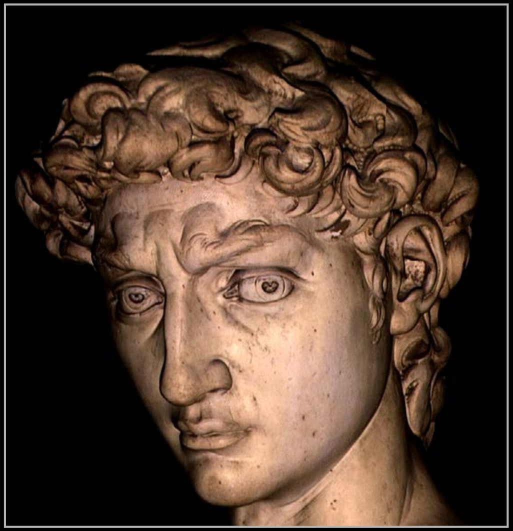
- Boundaries between object and background

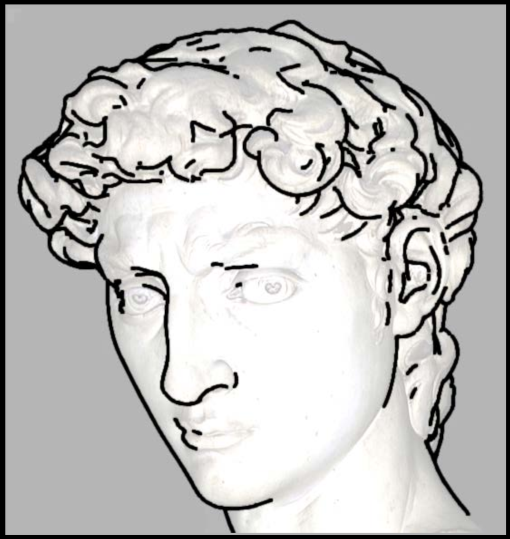


What Lines to Draw?

Occluding contours:

- Depth discontinuities
- Surface normal perpendicular to view direction

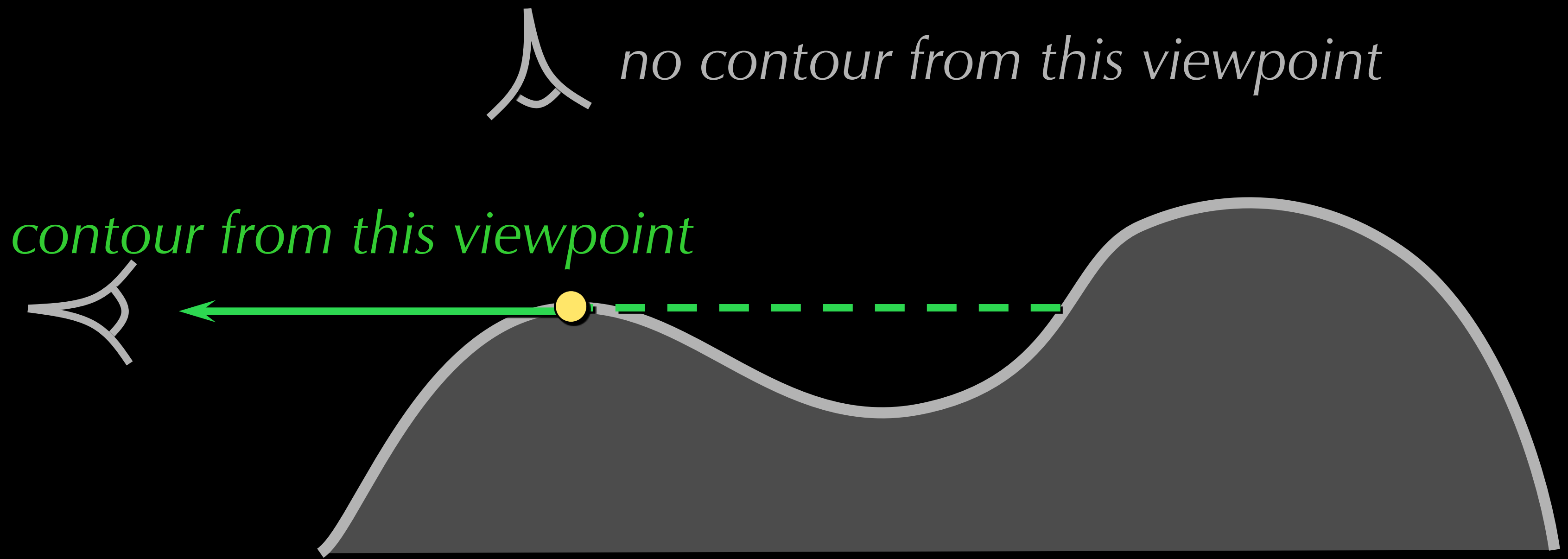




[Saito & Takahashi 90, Winkenbach & Salesin 94, Markosian et al 97, …]

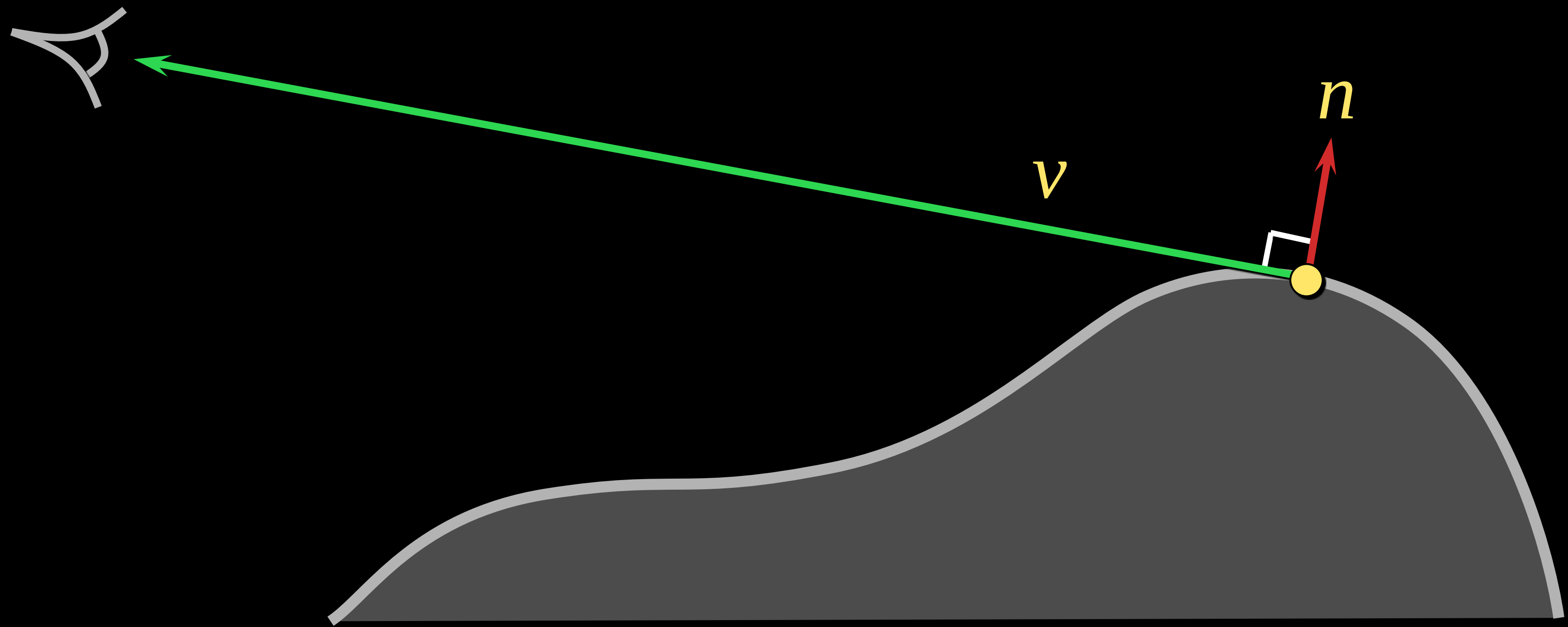
Occluding Contours

For any shape: locations of depth discontinuities
- View dependent
- Also called "interior and exterior silhouettes"



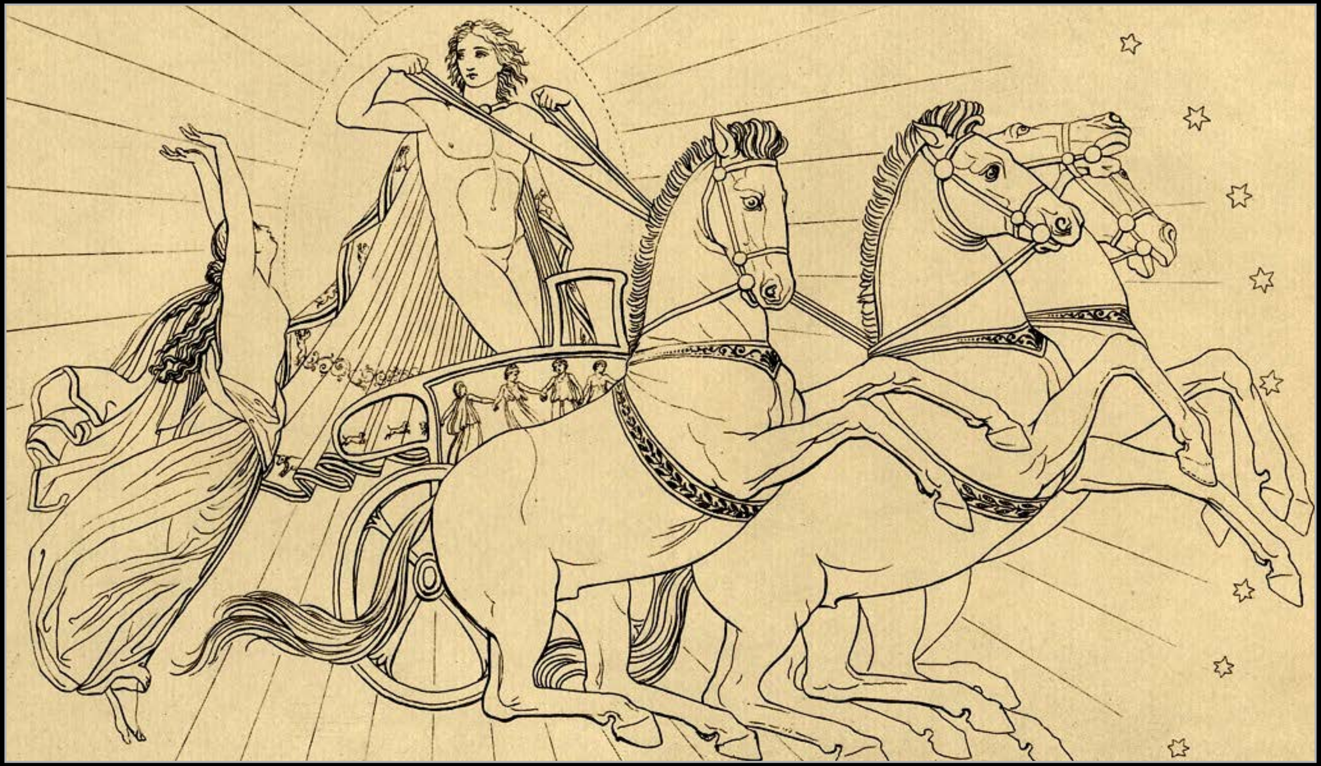
Occluding Contours

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



What Lines to Draw?

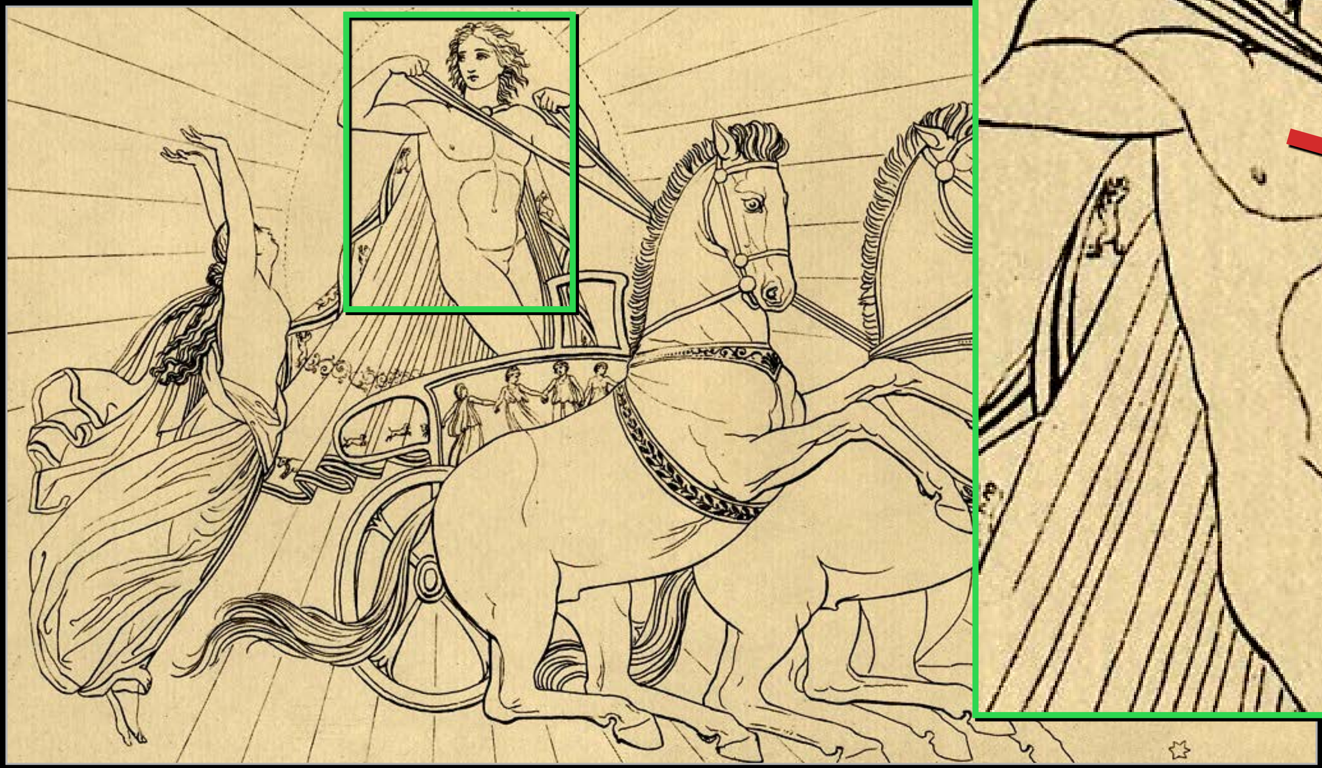
There are other lines…

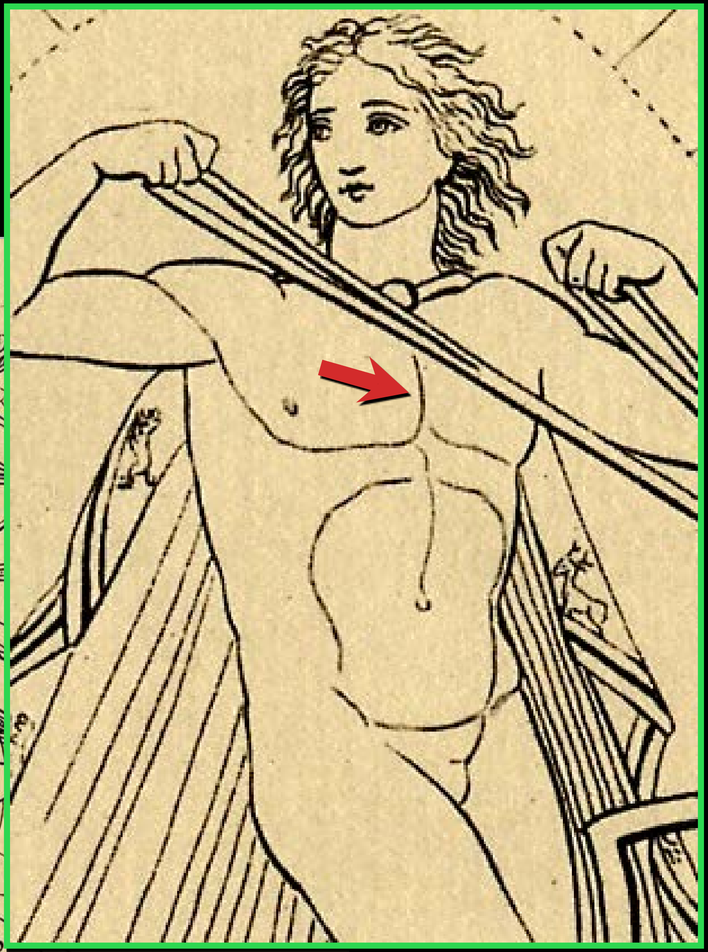


[Flaxman 1805]

What Lines to Draw?

There are other lines…

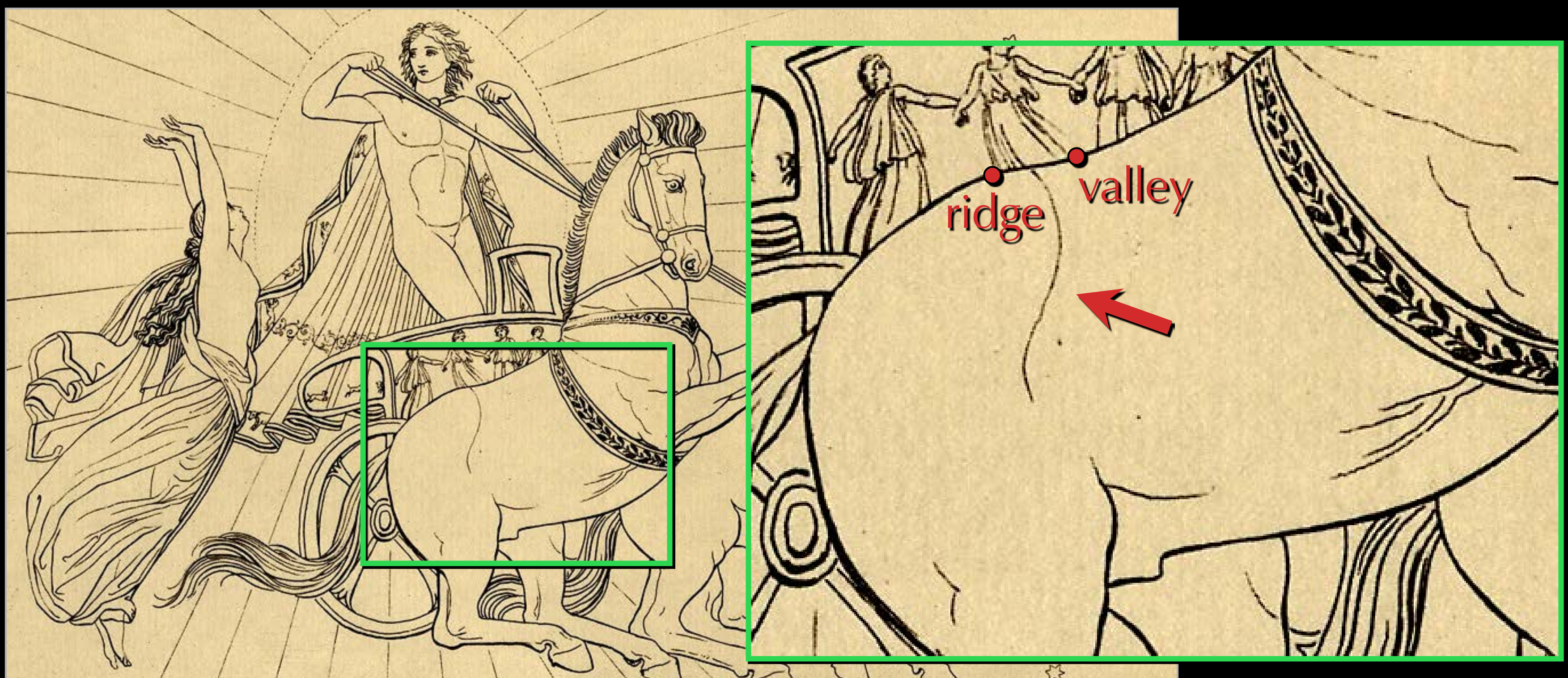




[Flaxman 1805]

What Lines to Draw?

There are other lines…



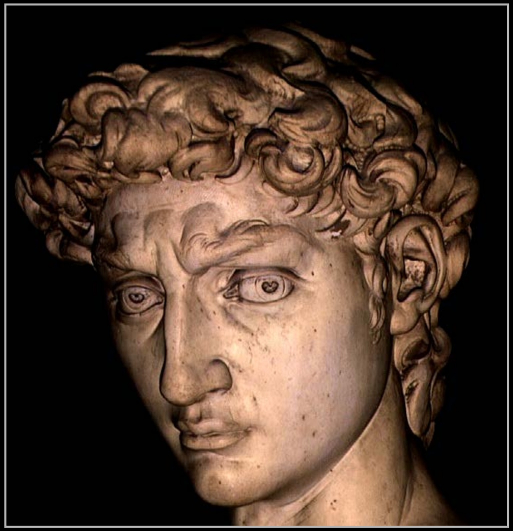
Hypothesis: some are "almost contours"

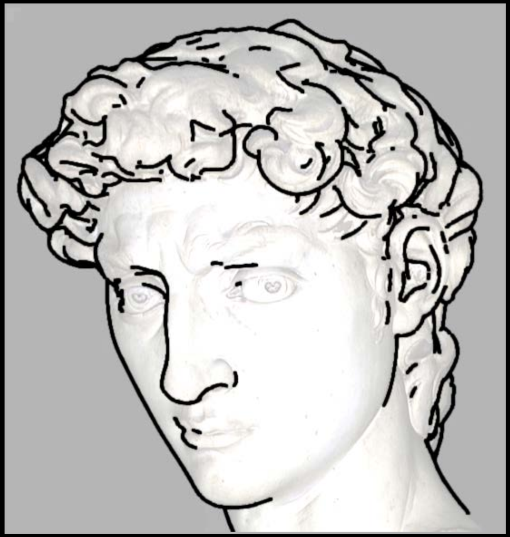
[Flaxman 1805]

Suggestive Contours

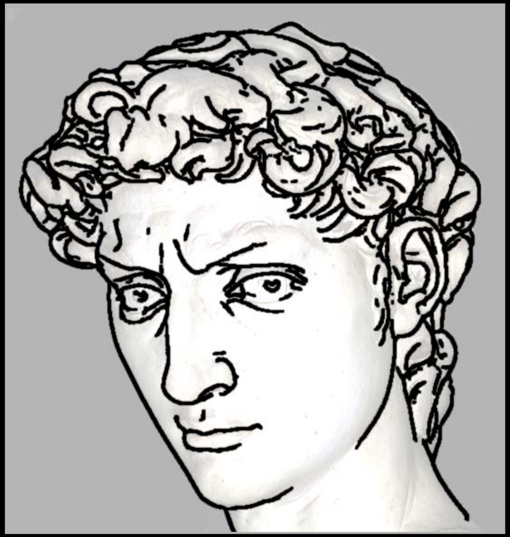
"Almost contours":

– Points that become contours in nearby views





contours

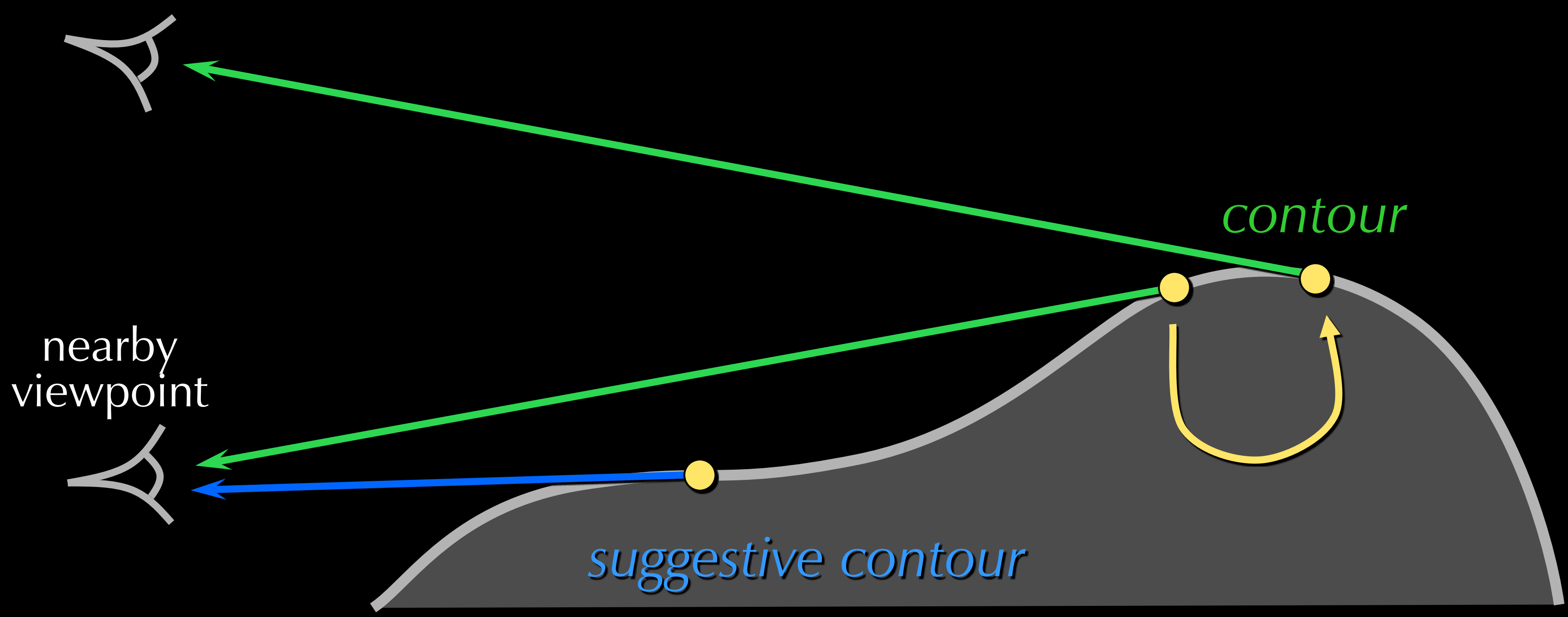


contours +
suggestive contours

Suggestive Contours: Definition 1

Contours in nearby viewpoints

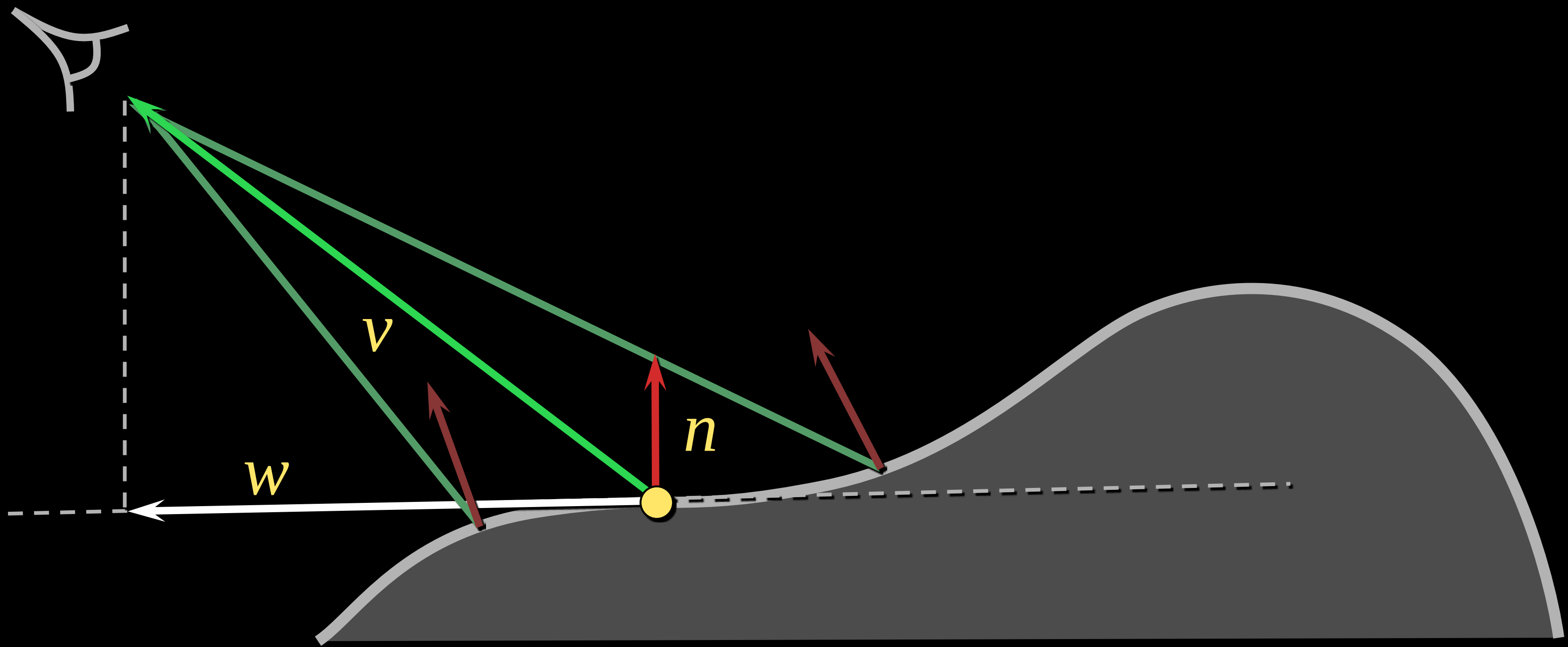
(not corresponding to contours in closer views)



Suggestive Contours: Definition 2

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

(in the projected view direction w)



Minima vs. Zero Crossings

Definition 2: Minima of $n \cdot v$

Finding minima is equivalent to:

finding zeros of the derivative

checking that 2^{nd} derivative is positive

This leads to definition 3.

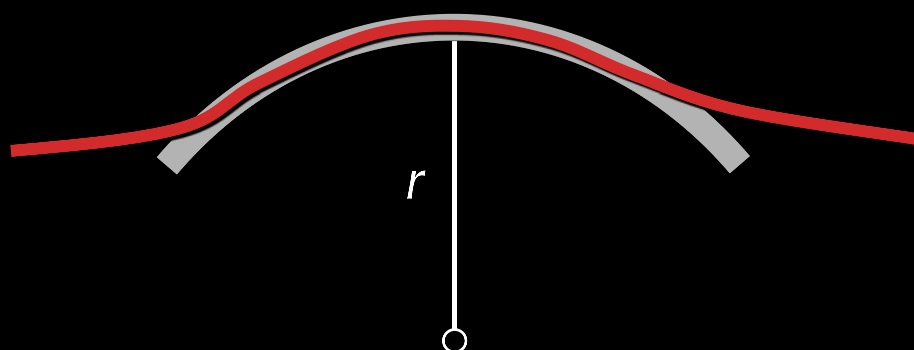
Derivative of $n \cdot v$ is a form of curvature…

Differential Geometry

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

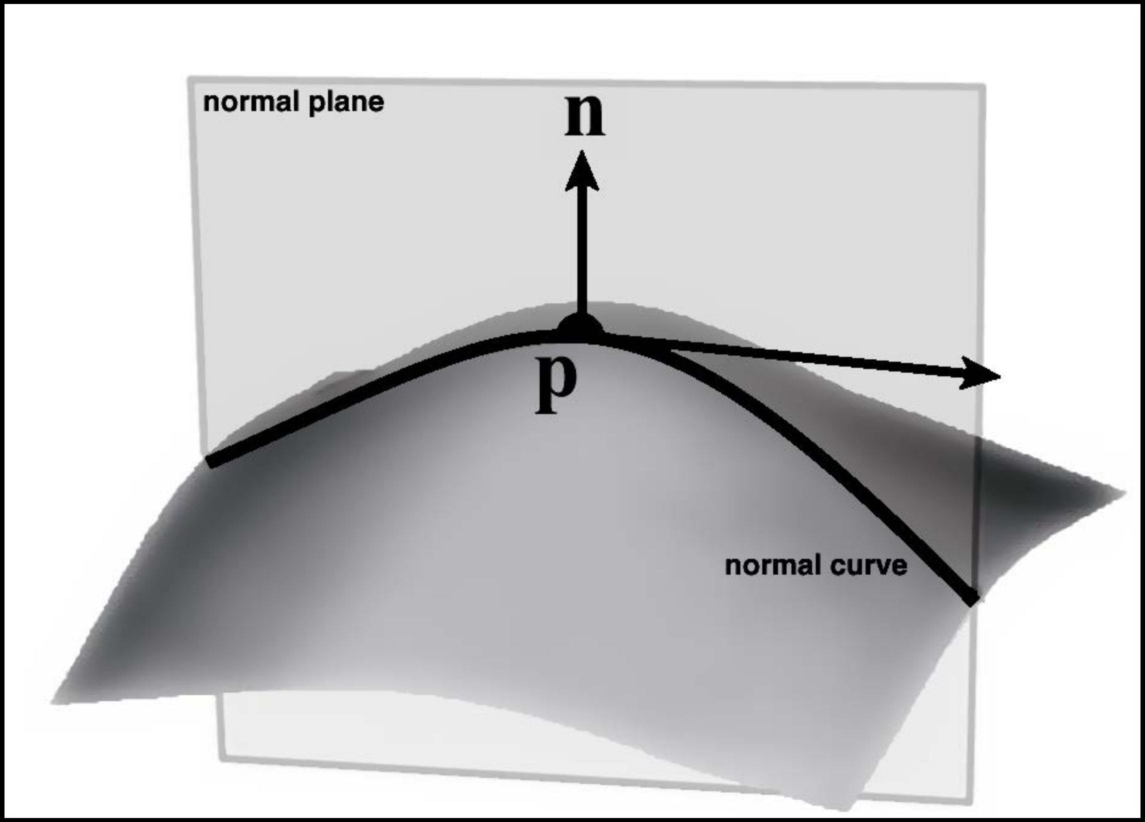


$$\kappa = {}^{1}\!/_{r}$$

- For a surface: ?

Normal Curvature

Curvature of a normal curve

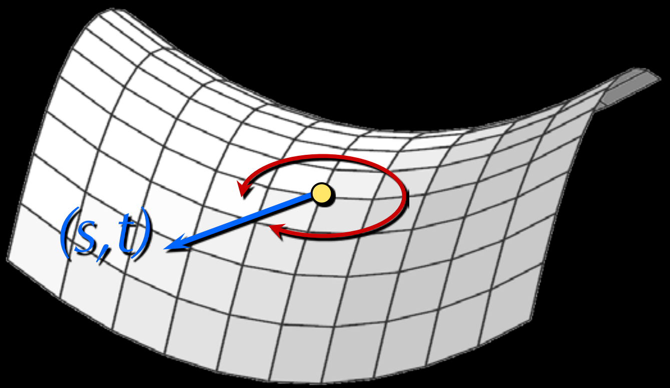


Curvature on a Surface

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

$$\kappa_n = \begin{pmatrix} s & t \end{pmatrix} \begin{pmatrix} e & f \\ f & g \end{pmatrix} \begin{pmatrix} s \\ t \end{pmatrix}$$

$$= \begin{pmatrix} s & t \end{pmatrix} \mathbf{II} \begin{pmatrix} s \\ t \end{pmatrix}$$



for a direction (s,t) in the tangent plane and a symmetric matrix **II**

Interpretation of **II**

Second-order Taylor-series expansion:

$$z(x, y) = \tfrac{1}{2} e x^2 + f x y + \tfrac{1}{2} g y^2$$

"Hessian": second partial derivatives

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

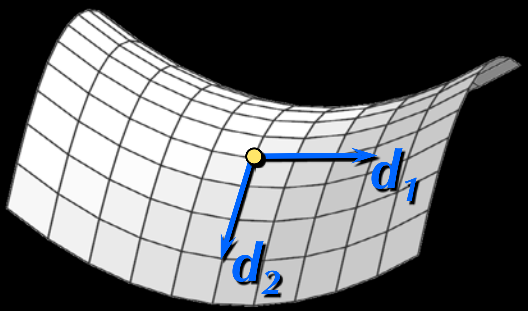
Derivatives of normal

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

Principal Curvatures and Directions

Can always rotate coordinate system
so that **II** is diagonal:

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



κ_1 and κ_2 are *principal curvatures*, and are minimum and maximum of normal curvature

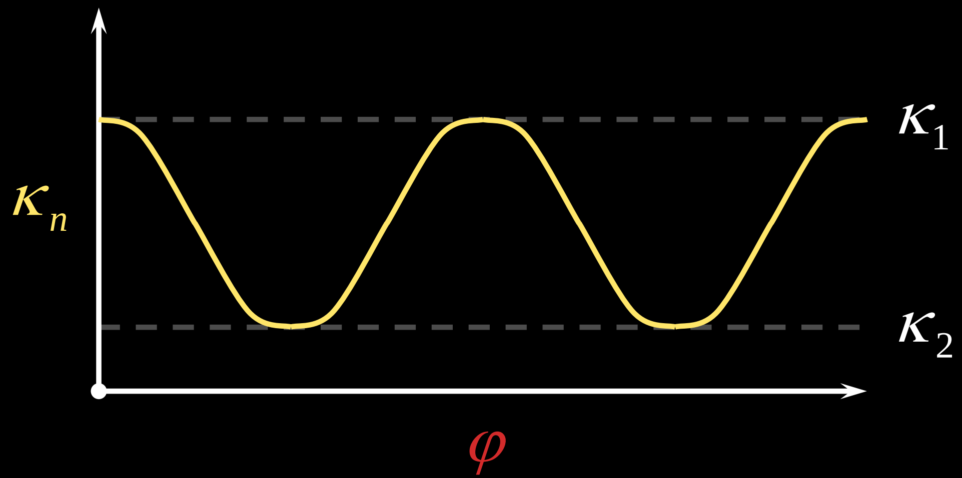
Associated directions are *principal directions*

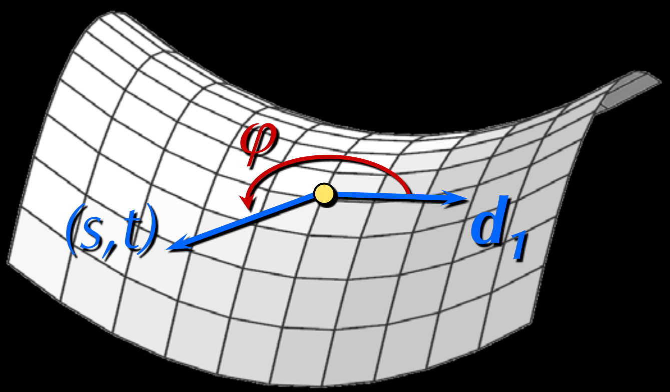
Eigenvalues and eigenvectors of **II**

Euler's Formula

This lets us write normal curvature differently:

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





Gaussian and Mean Curvature

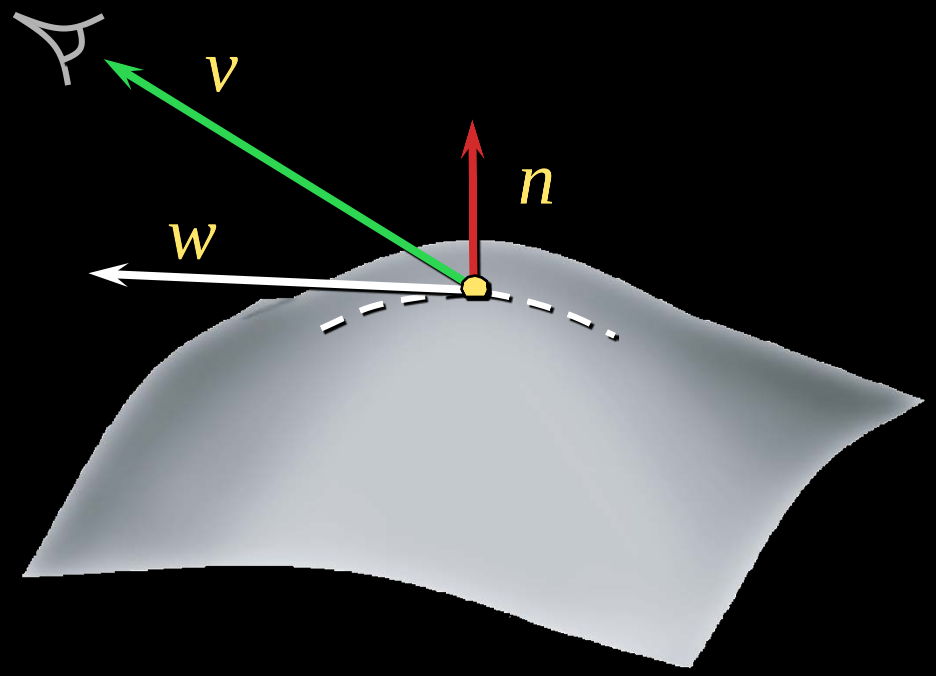
The Gaussian curvature $K = \kappa_1 \kappa_2$

The mean curvature $H = ½ (\kappa_1 + \kappa_2)$

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

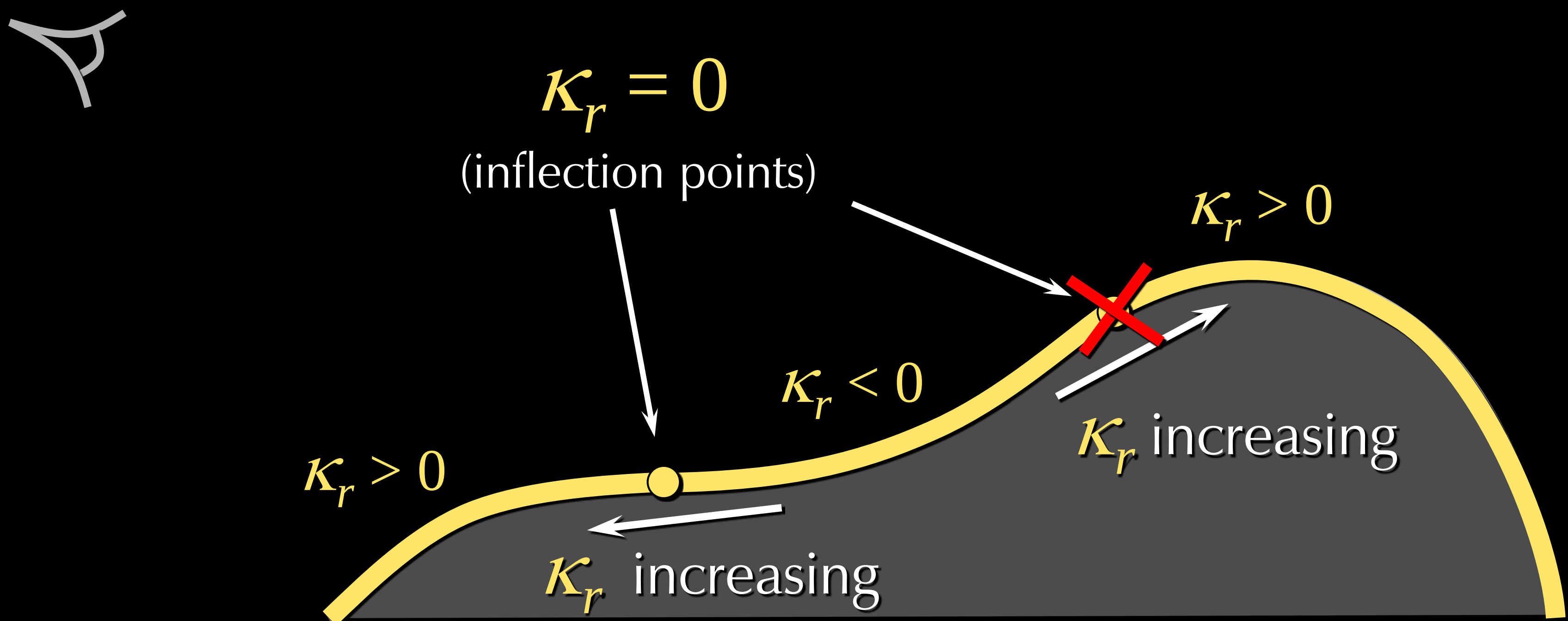
Radial Curvature κ_r

Curvature in projected view direction, *w*:



Suggestive Contours: Definition 3

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



Finding Suggestive Contours

Finding κ_r:

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

Finding $D_w \kappa_r$:

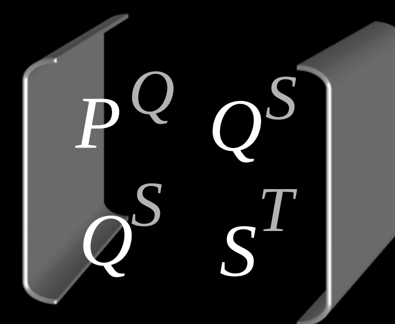
?

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: $\mathbf{C} =$



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

Finding Suggestive Contours

Finding κ_r:

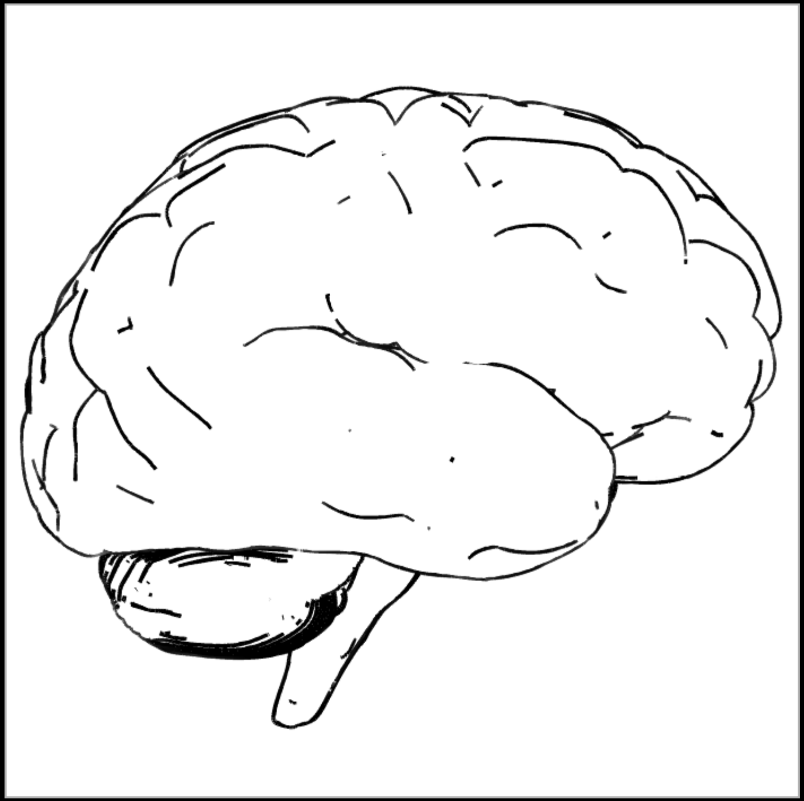
$$\kappa_r = \mathbf{II}(\hat{w}, \hat{w})$$

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

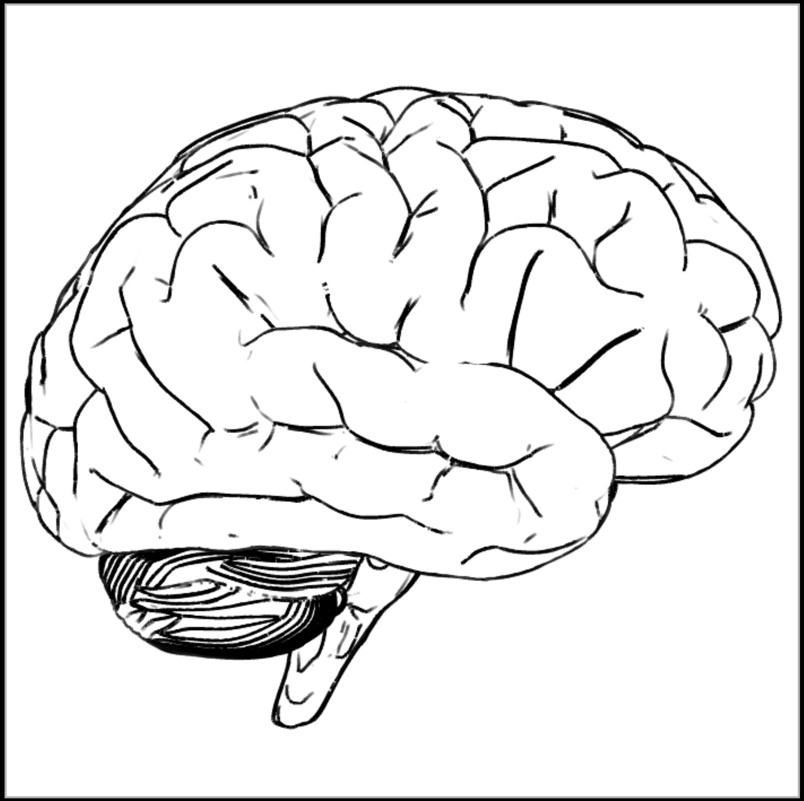
$$D_{\hat{w}}\kappa_r = \mathbf{C}(\hat{w}, \hat{w}, \hat{w}) + 2K\cot\theta,$$

where $\kappa_r = 0$

Results…

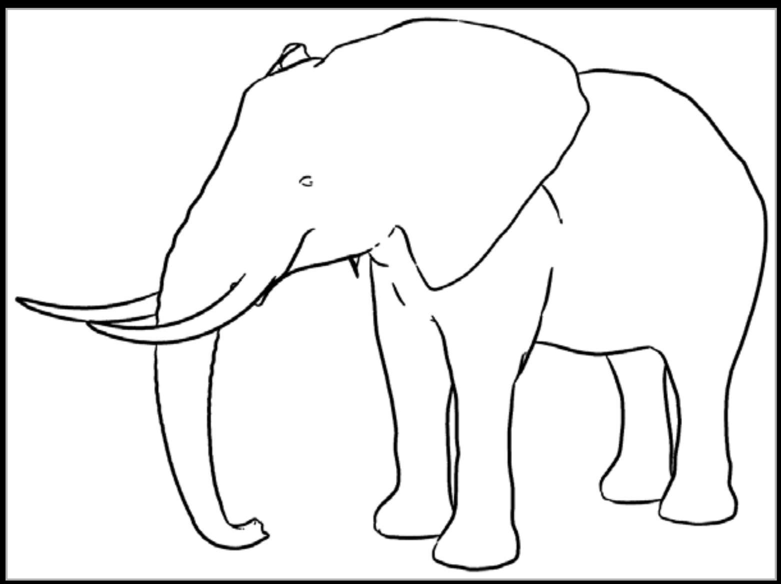


contours

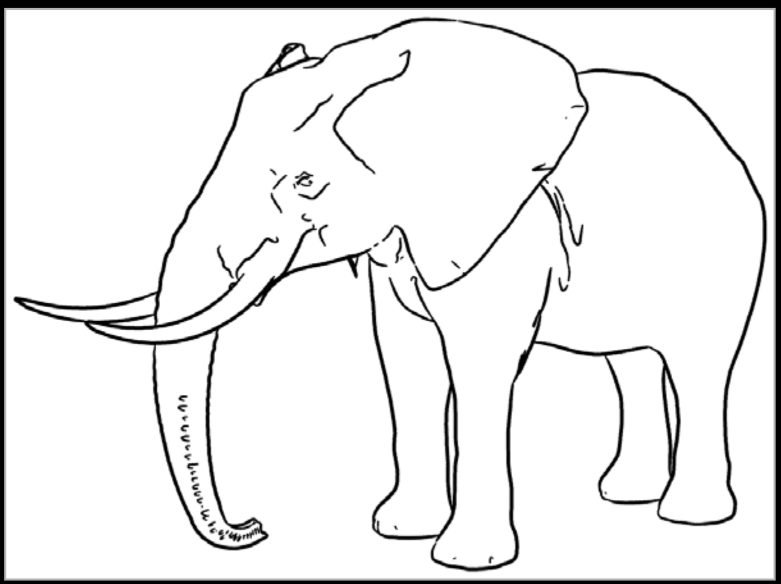


contours +
suggestive contours

Results…

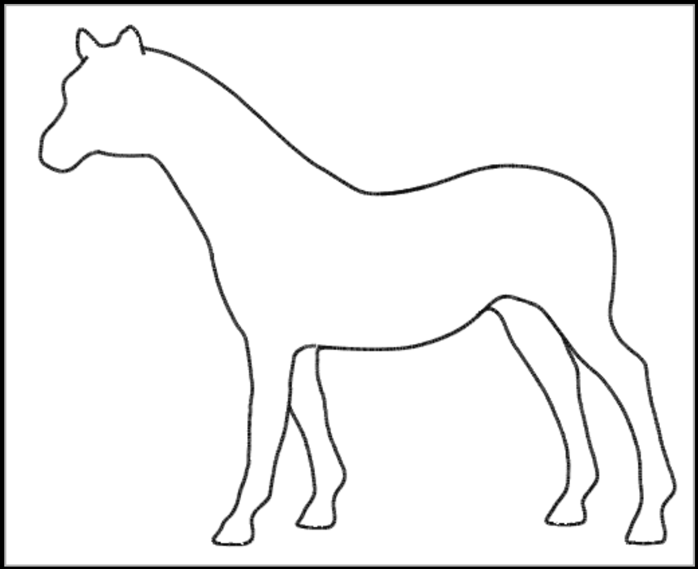


contours

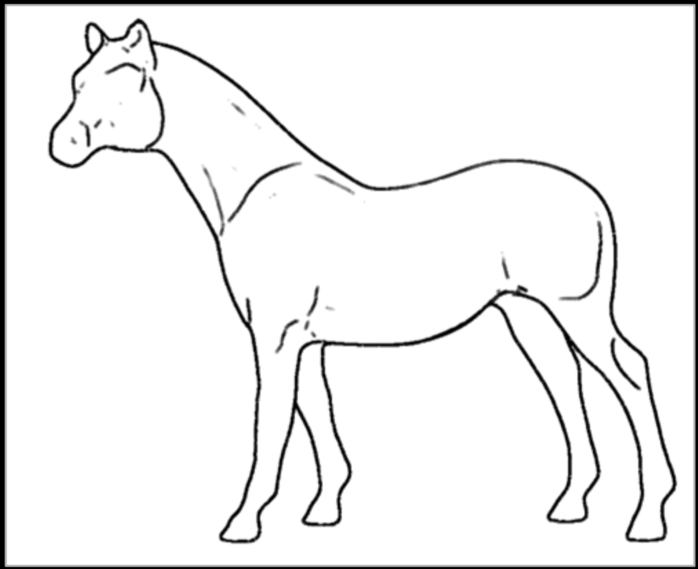


contours +
suggestive contours

Results…



contours



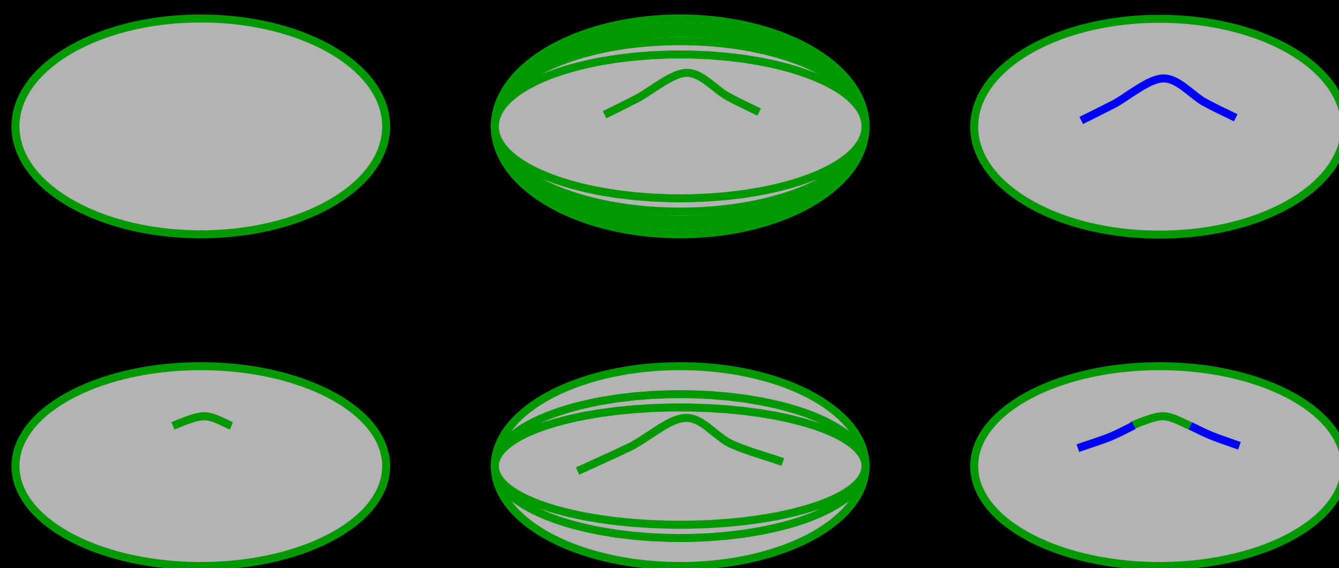
contours +
suggestive contours

Qualitative Structure

Suggestive contours have two behaviors:

anticipation

extension



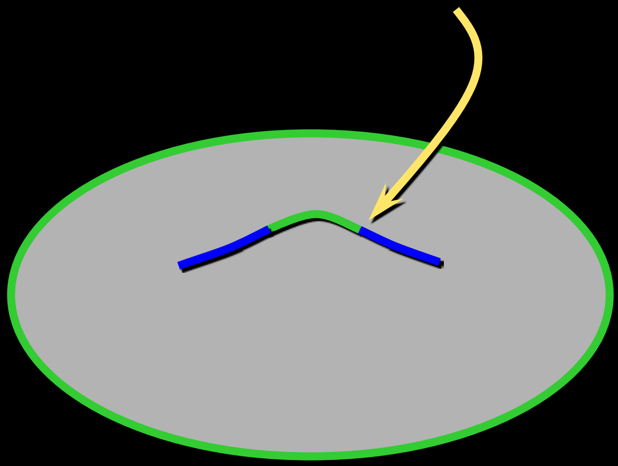
original viewpoint, contours only

nearby viewpoint, contours only

original viewpoint, contours + suggestive contours

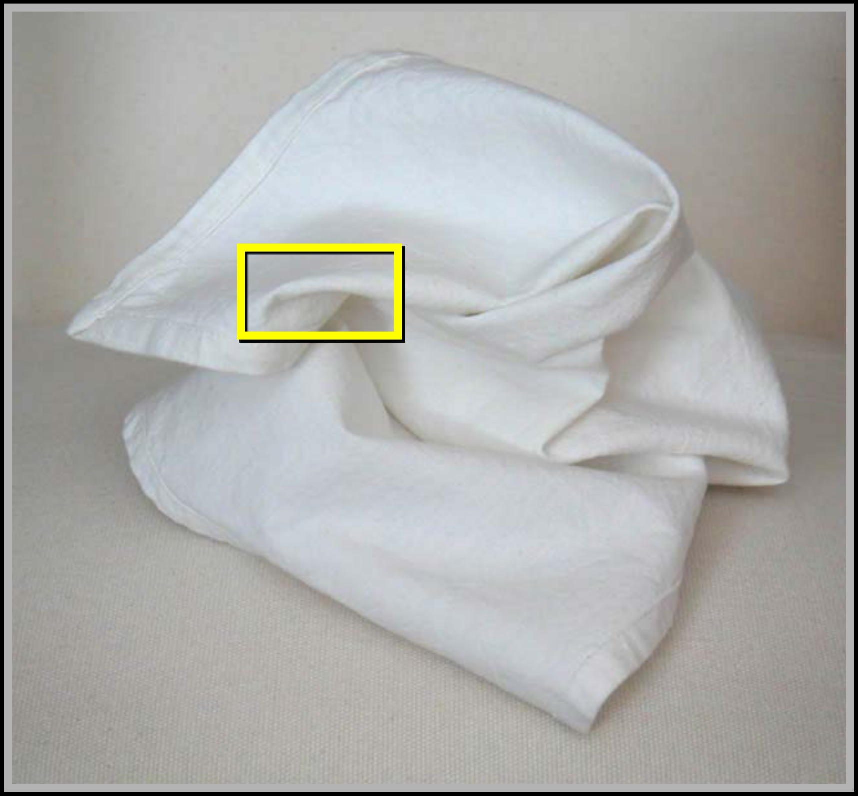
Continuity of Extensions

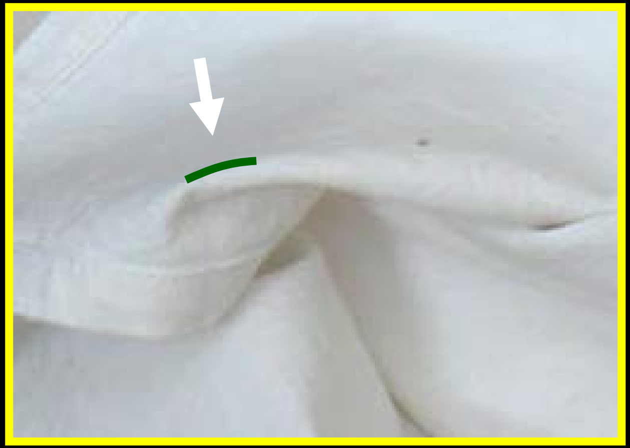
Suggestive contours line up with contours in the image



Ending contours

Difficult to localize in real images





Algorithms for Extracting Lines

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

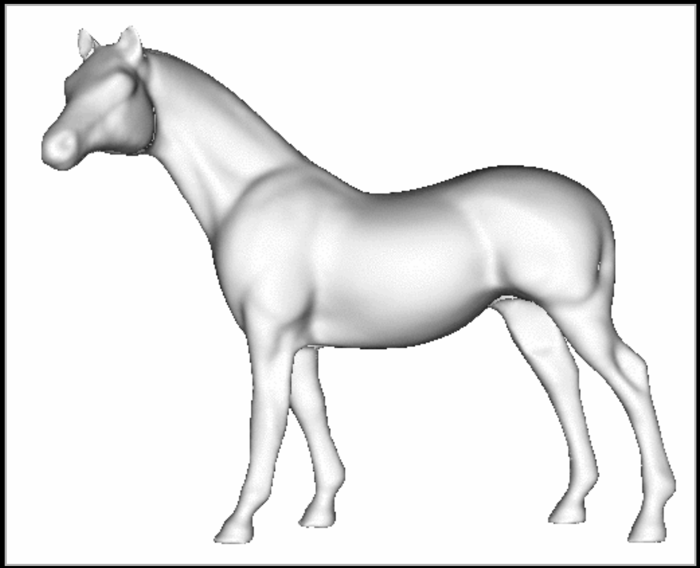
Based on [Isenberg 2003]

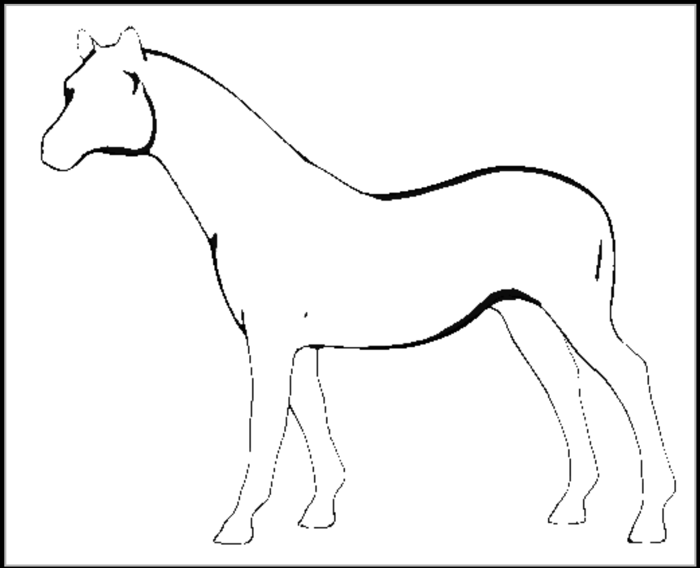
Contours: Image-Space Algorithm

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

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

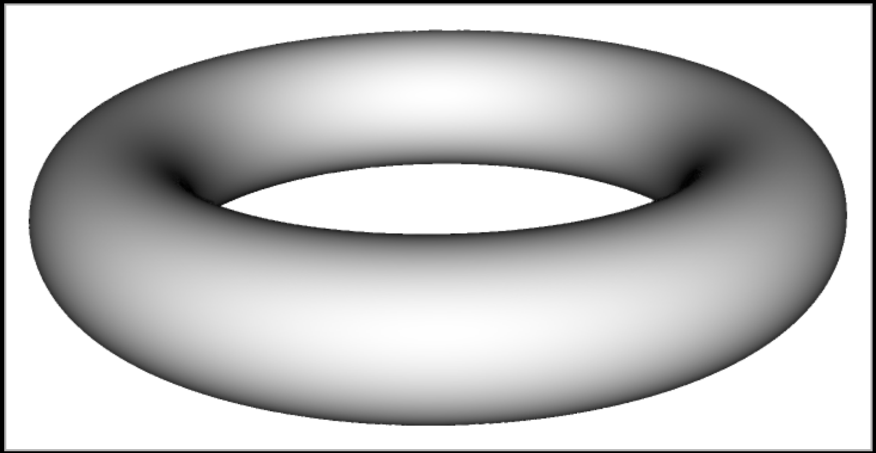
- Variant: index into texture based on $n \cdot v$
- More variants: environment map, pixel shader

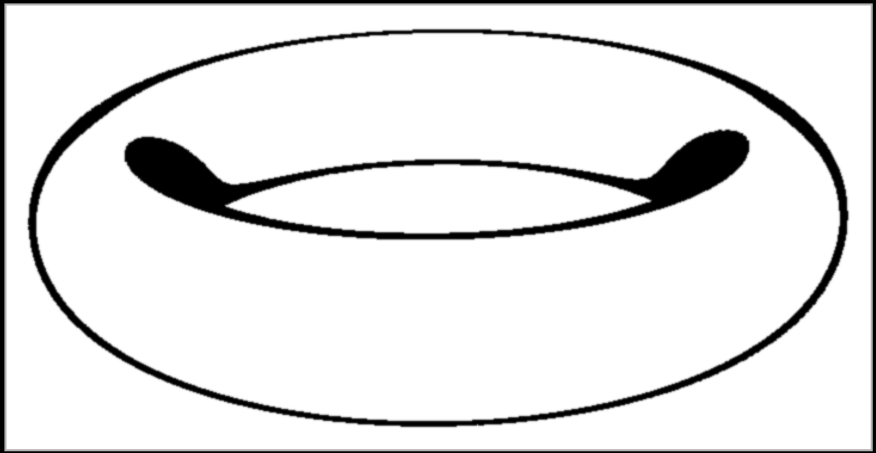




Line Thickness

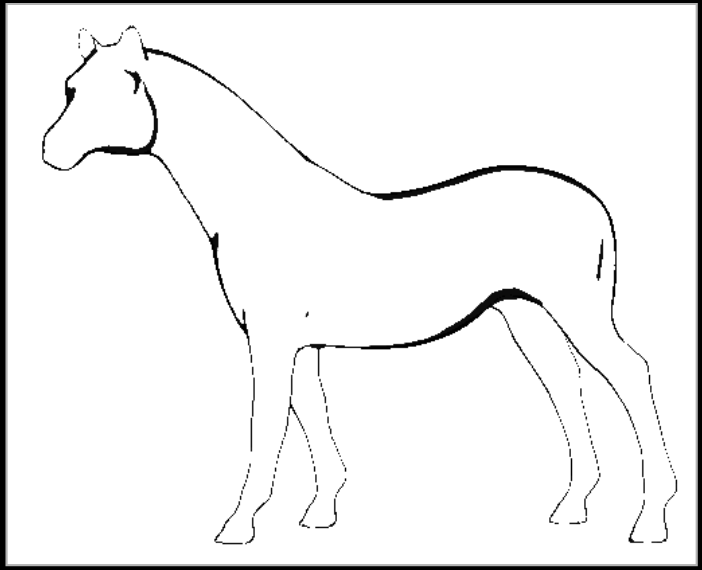
- Drawback: line thickness varies
  - Thicker lines in low-curvature regions



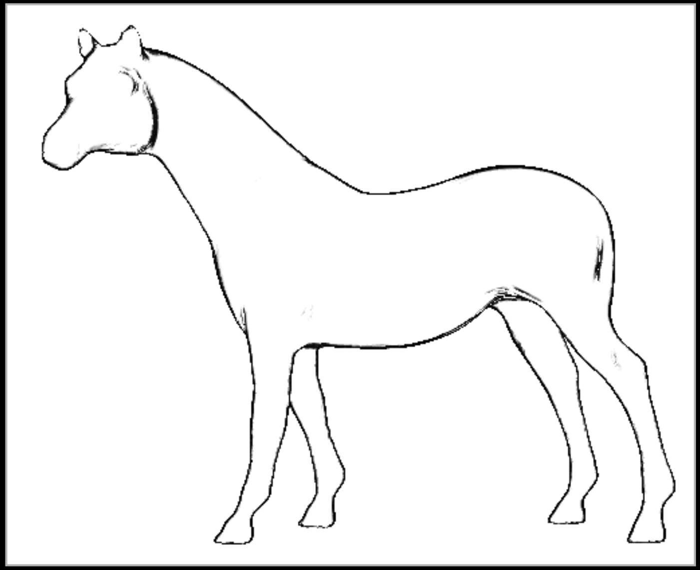


Line Thickness Control

- Solution #1: mipmap trick
  - Load same-width line into each mipmap level



Original

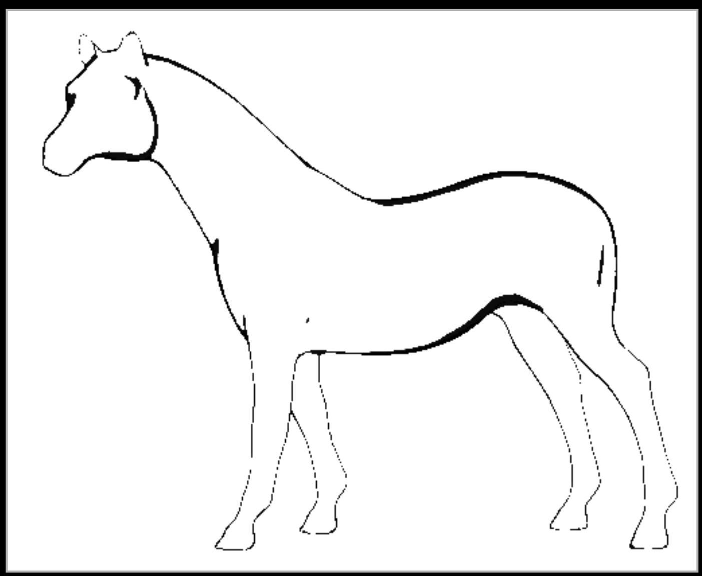


New

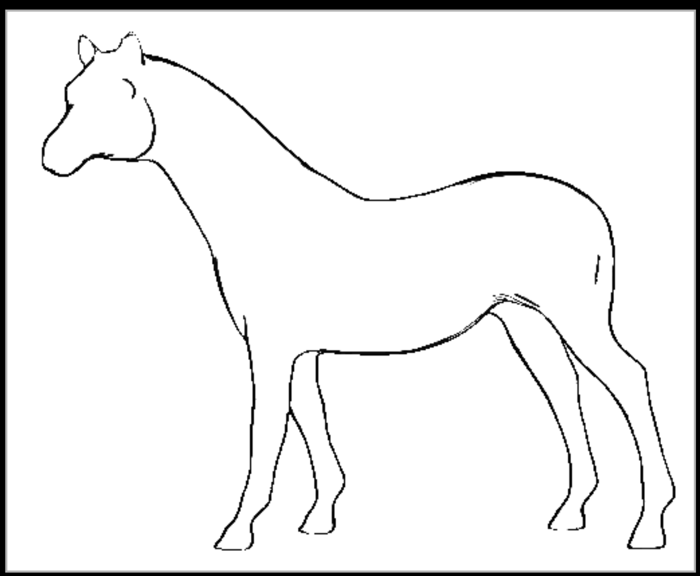
Line Thickness Control

Solution #2: curvature-dependent threshold

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



Original

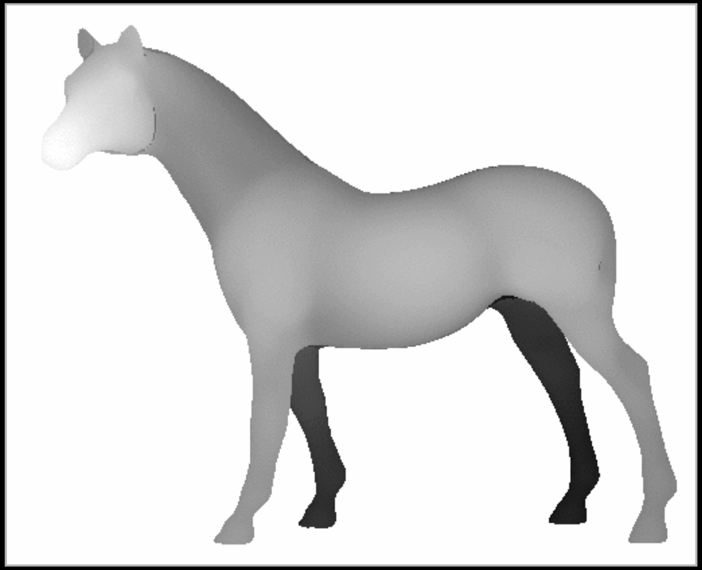


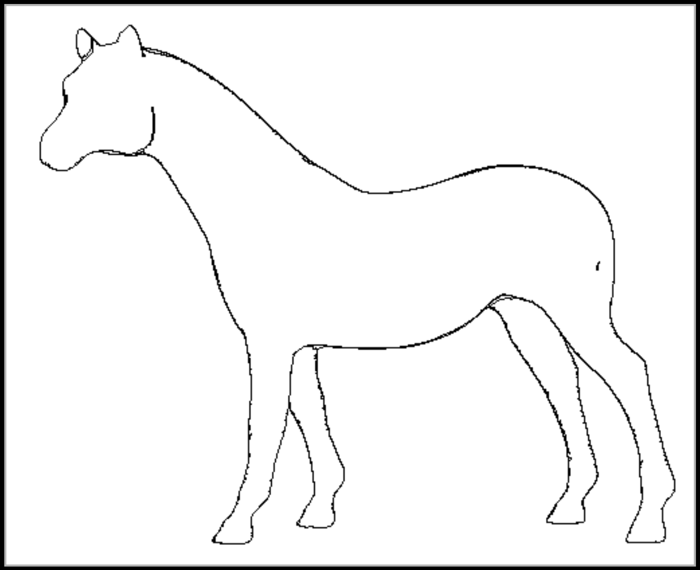
New

Contours: 2nd Image-Space Algorithm

Render depth image, find edges

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





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 $n \cdot v$

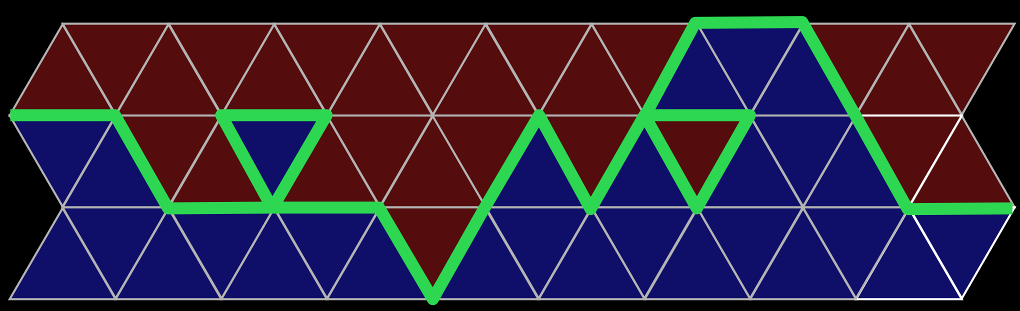
Contours: Object-Space Algorithm

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



Frontfacing

Backfacing

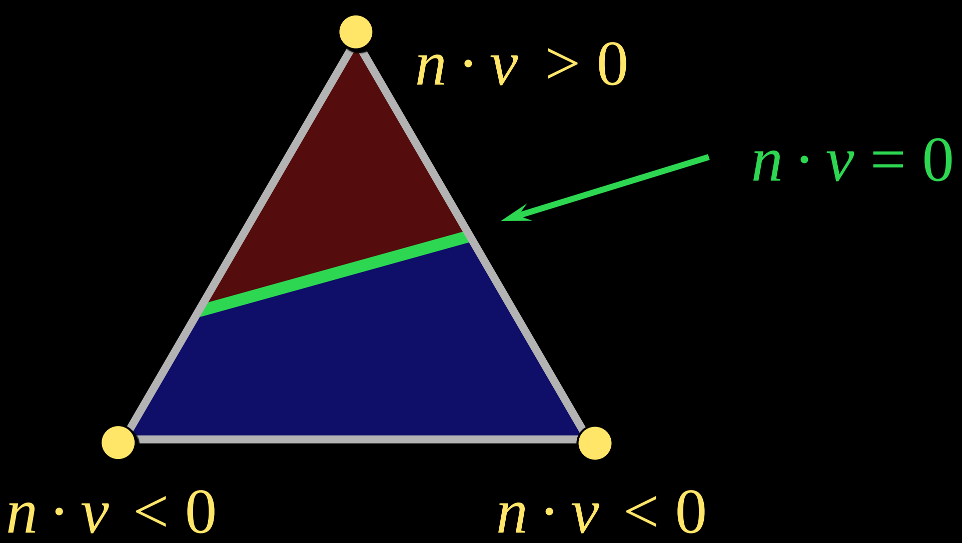
Contour

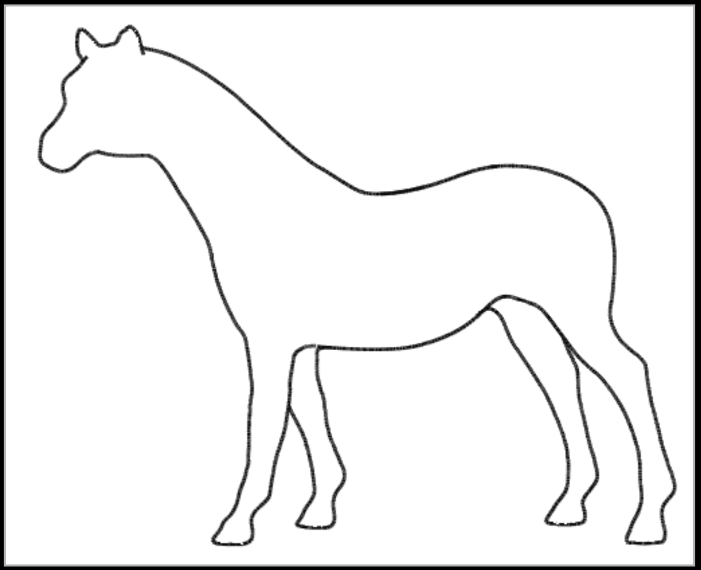
Contours: Object-Space Algorithm

[Hertzmann 2000]

Second definition: within-face lines

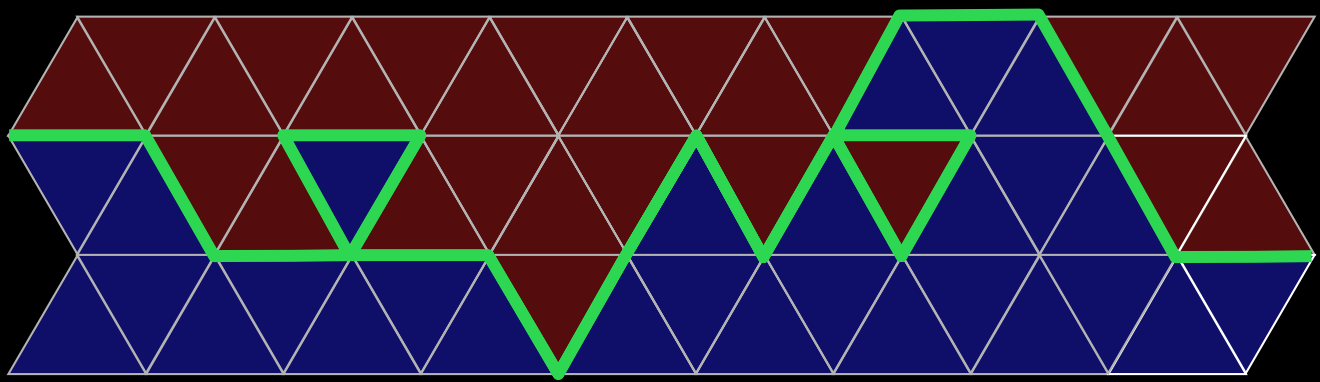
- For each vertex: compute $n \cdot v$
- For each face: if signs not the same, interpolate to find zero crossing within face

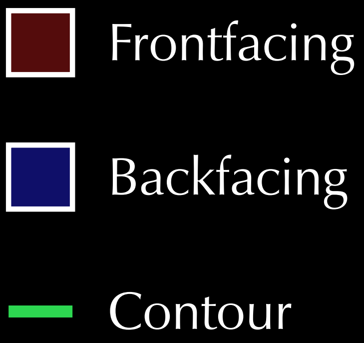




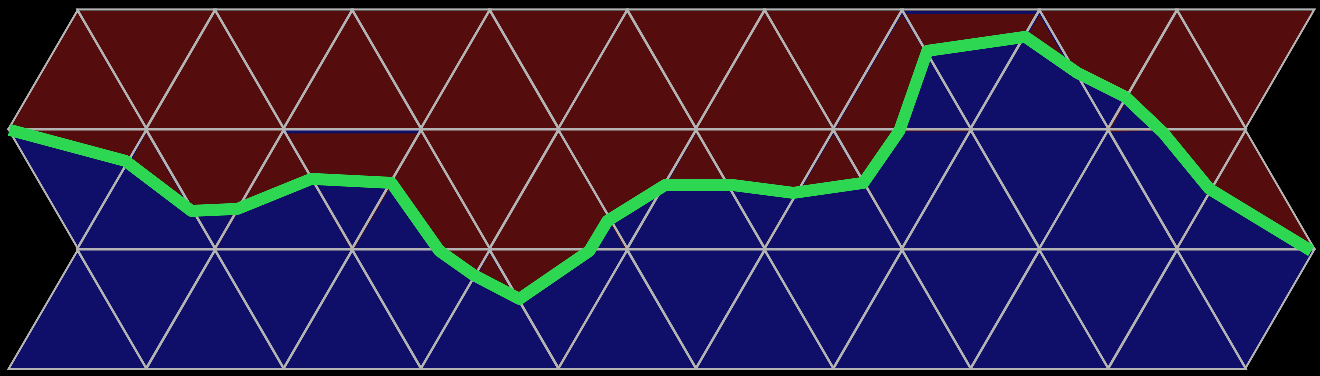
Occluding Contours on Meshes

Contours along edges

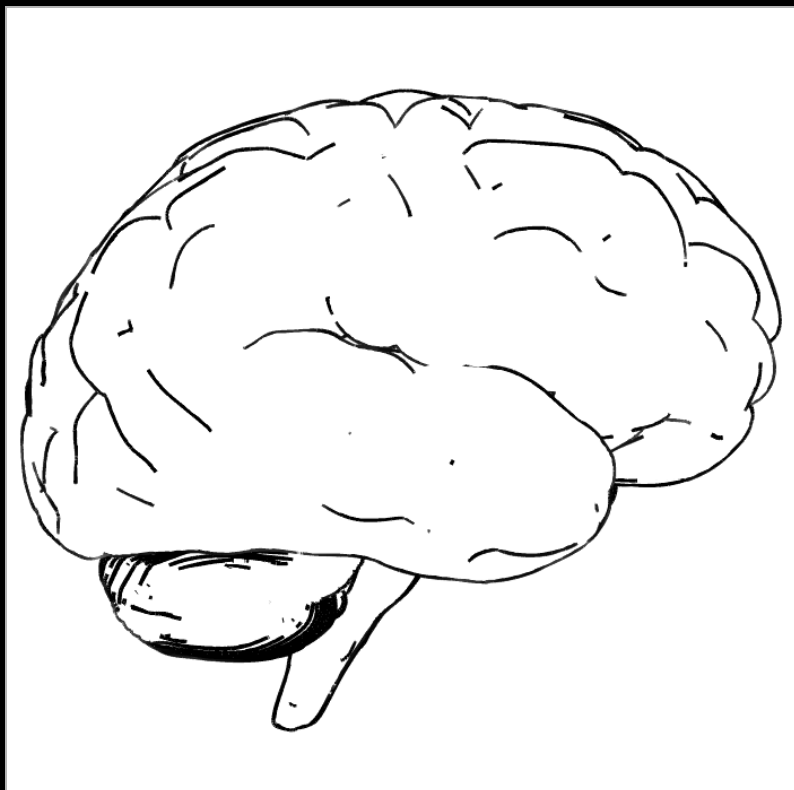




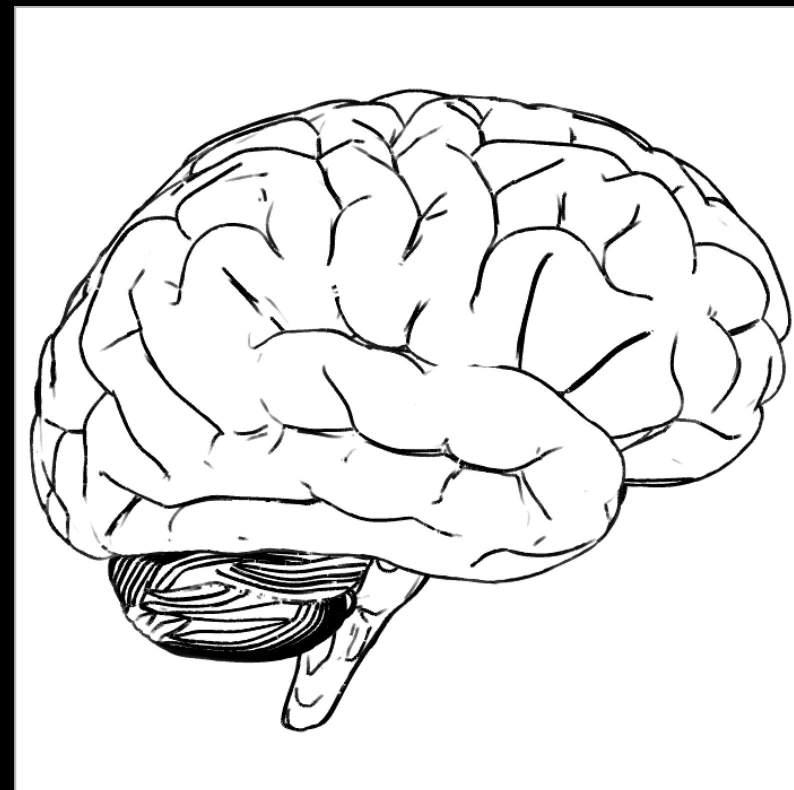
Contours within faces



Moving on to Suggestive Contours…



contours



contours +
suggestive contours

Algorithms for Suggestive Contours

Definition 1: contours in nearby views

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

Definition 3: zeros of radial curvature

Algorithms for Suggestive Contours

Definition 1: contours in nearby views

→ search over viewpoints

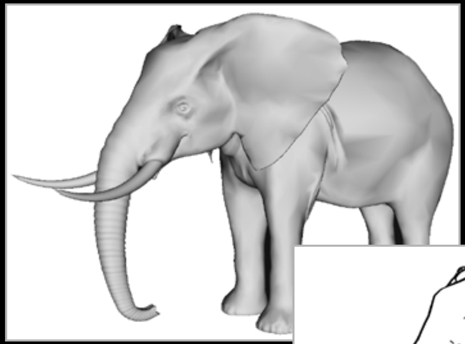
Algorithms for Suggestive Contours

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

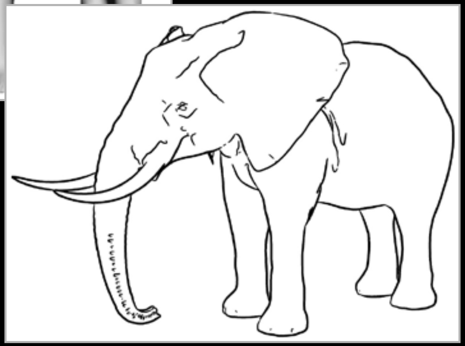
$\rightarrow$ image processing to detect minima

[DeCarlo 03, Lee 07]

Render diffuse-shaded image



Filter to detect valleys in intensity



Algorithms for Suggestive Contours

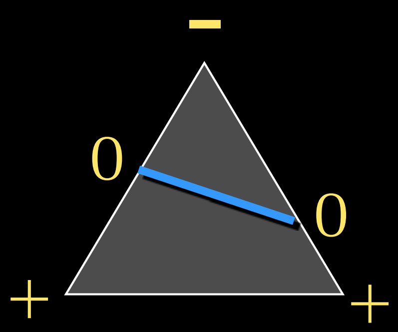
Definition 3: zeros of radial curvature

$\rightarrow$ object-space curve extraction

Extraction Algorithm

For each view:

- Compute radial curvature κ_r per vertex
- Find zero crossings per-face
- Keep lines with $D_w \kappa_r > 0$



$\Rightarrow$ Need κ_r and $D_w \kappa_r$ for each vertex

Curvature Estimation

Desirable Properties of a Curvature Estimation Algorithm

No degenerate configurations

Works on arbitrary meshes:
no special cases for holes, etc.

Handles tessellations with varying triangle sizes

Efficient in time and space

- No auxiliary data structures for connectivity

Estimating Normals

Common algorithm for per-vertex normals:
(weighted) average of face normals

- No degeneracies or special cases
- Works on arbitrary meshes
- No extra data structures

Simple to extend to curvatures!

Algorithm

For each face f:

– Estimate $\mathbf{II}_f$

– For each of the three vertices:

$$\mathbf{II}_v \mathrel{+}= w_{f,v} * \mathbf{II}_f$$

For each vertex:

– Divide $\mathbf{II}$ by sum of weights

– If desired, compute eigenstuff

Algorithm

For each face f:

- Estimate $\mathbf{II}_f$ How?
- For each of the three vertices:

$$\mathbf{II}_v \mathrel{+}= w_{f,v} * \mathbf{II}_f$$

For each vertex:

- Divide $\mathbf{II}$ by sum of weights
- If desired, compute eigenstuff

Computing Per-Face Curvature

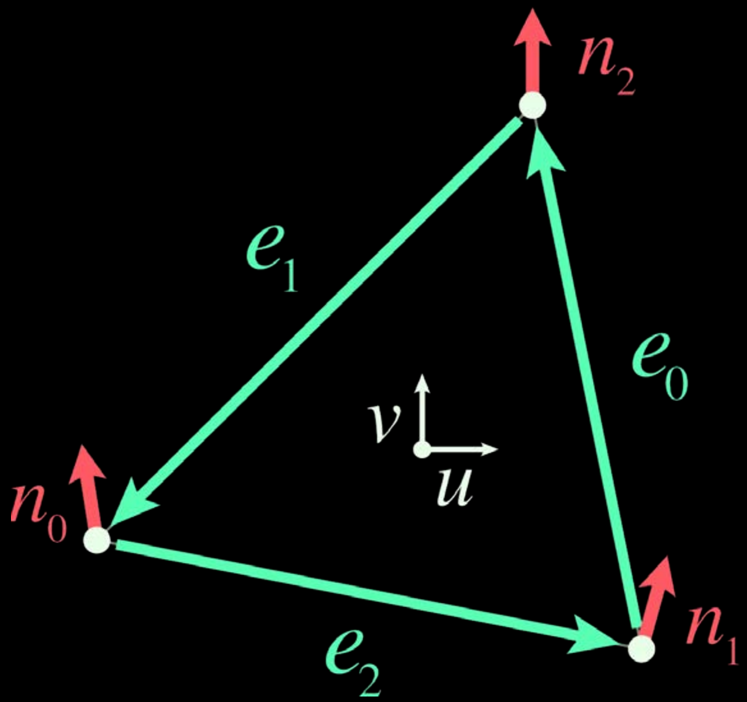
Key: curvature as derivative of normals

$$\mathbf{II}\begin{pmatrix} s \\ t \end{pmatrix} = D_{(s,t)}\,\mathbf{n}$$

Finite differences approximation:

- Know differences of normals along edges
- Solve for elements of $\mathbf{II}$

Computing Per-Face Curvature



$$\mathbf{II}\begin{pmatrix} e_0\cdot\hat{\mathbf{u}} \\ e_0\cdot\hat{\mathbf{v}} \end{pmatrix} = \begin{pmatrix} (n_2-n_1)\cdot\hat{\mathbf{u}} \\ (n_2-n_1)\cdot\hat{\mathbf{v}} \end{pmatrix}$$

$$\mathbf{II}\begin{pmatrix} e_1\cdot\hat{\mathbf{u}} \\ e_1\cdot\hat{\mathbf{v}} \end{pmatrix} = \begin{pmatrix} (n_0-n_2)\cdot\hat{\mathbf{u}} \\ (n_0-n_2)\cdot\hat{\mathbf{v}} \end{pmatrix}$$

$$\mathbf{II}\begin{pmatrix} e_2\cdot\hat{\mathbf{u}} \\ e_2\cdot\hat{\mathbf{v}} \end{pmatrix} = \begin{pmatrix} (n_1-n_0)\cdot\hat{\mathbf{u}} \\ (n_1-n_0)\cdot\hat{\mathbf{v}} \end{pmatrix}$$

6 equations (2 redundant), 3 unknowns:
solve using least squares

Algorithm

For each face f:

- Estimate $\mathbf{II}_f$
- For each of the three vertices:

$$\mathbf{II}_v \mathrel{+}= w_{f,v} * \mathbf{II}_f$$

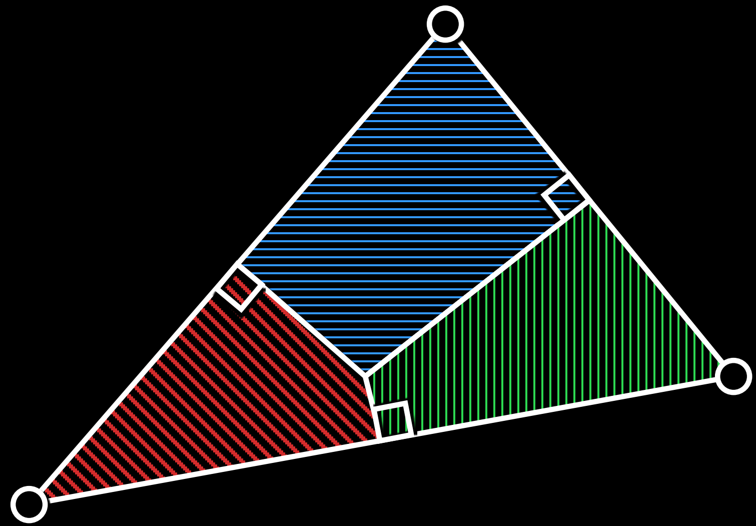
What weights?

For each vertex:

- Divide $\mathbf{II}$ by sum of weights
- If desired, compute eigenstuff

Voronoi Weighting

$w_{f,v}$ = area of the part of f closest to v



Algorithm

For each face f:

- Estimate $\mathbf{II}_f$
- For each of the three vertices:

$$\mathbf{II}_v \mathrel{+}= w_{f,v} * \mathbf{II}_f$$

Change of coordinates

For each vertex:

- Divide $\mathbf{II}$ by sum of weights
- If desired, compute eigenstuff

Change of Coordinates

If old and new coordinates are coplanar:

$$\mathbf{II}_{new} = R^{\mathrm{T}} \mathbf{II}_{old} R$$

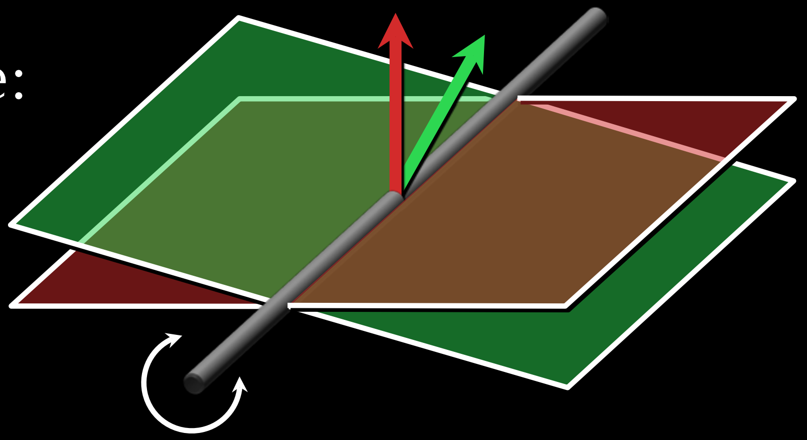
Change of Coordinates

If old and new coordinates are coplanar:

$$\mathbf{II}_{new} = R^{\mathrm{T}} \mathbf{II}_{old} R$$

If not coplanar:

- Can't just project onto plane: would "lose" curvature
- Instead: rotate around cross product of normals



Derivative of Curvature

Just as

$$\mathbf{II}\begin{pmatrix} s \\ t \end{pmatrix} = D_{(s,t)}\,\mathbf{n}$$

gives derivative of the normal,

$$\mathbf{C}\begin{pmatrix} s \\ t \end{pmatrix} = D_{(s,t)}\,\mathbf{II}$$

gives derivative of curvature

Generalized Algorithm

Compute $\mathbf{II}$ per vertex

For each face f:

- Estimate $\mathbf{C}_f$ from $\mathbf{II}$ at vertices
- For each of the three vertices:

$$\mathbf{C}_v \mathrel{+}= w_{f,v} * \mathbf{C}_f$$

For each vertex:

- Divide $\mathbf{C}$ by sum of weights

Suggestive Contour Implementation

Precompute **II** and C per vertex

Finding κ_r:

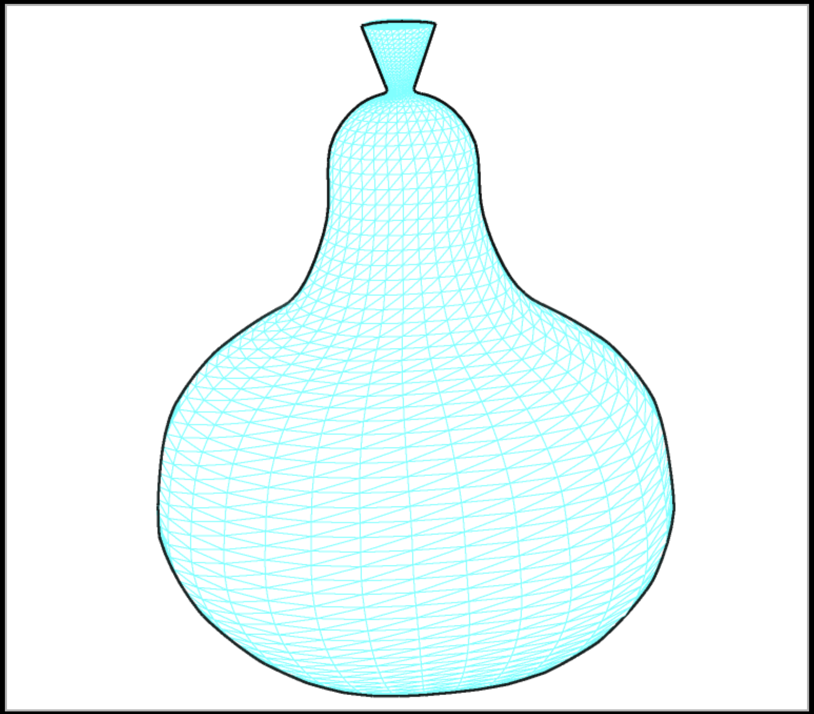
$$\kappa_r = w^{\mathrm{T}}\,\mathbf{II}\,w$$

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

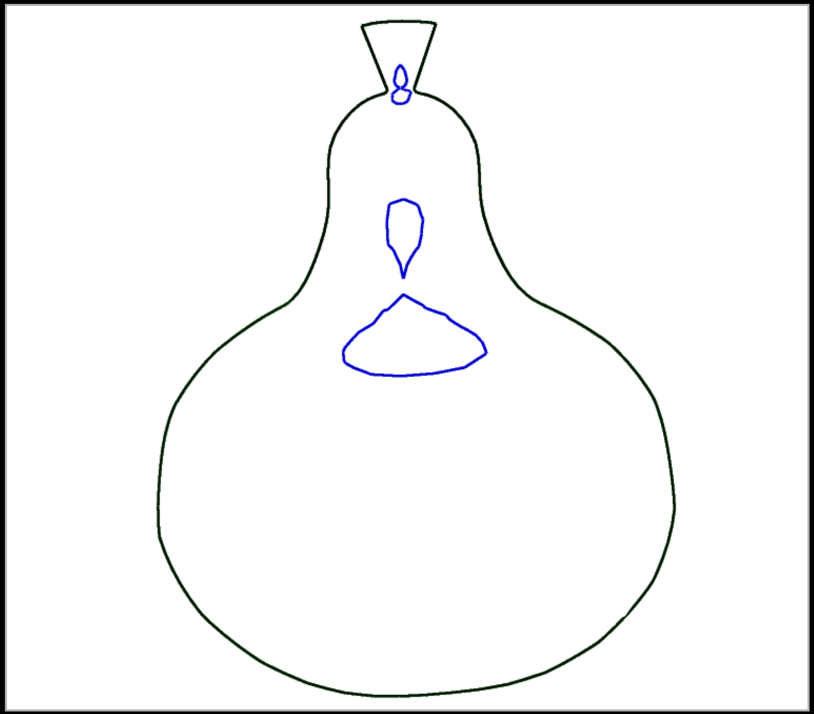
$$D_w\kappa_r = \mathbf{C}(w,w,w) + 2K\cot\theta,$$

$$\text{where } \kappa_r = 0,\ \theta = \cos^{-1}(n \cdot v)$$

Suggestive Contours as Zeros of κ_r

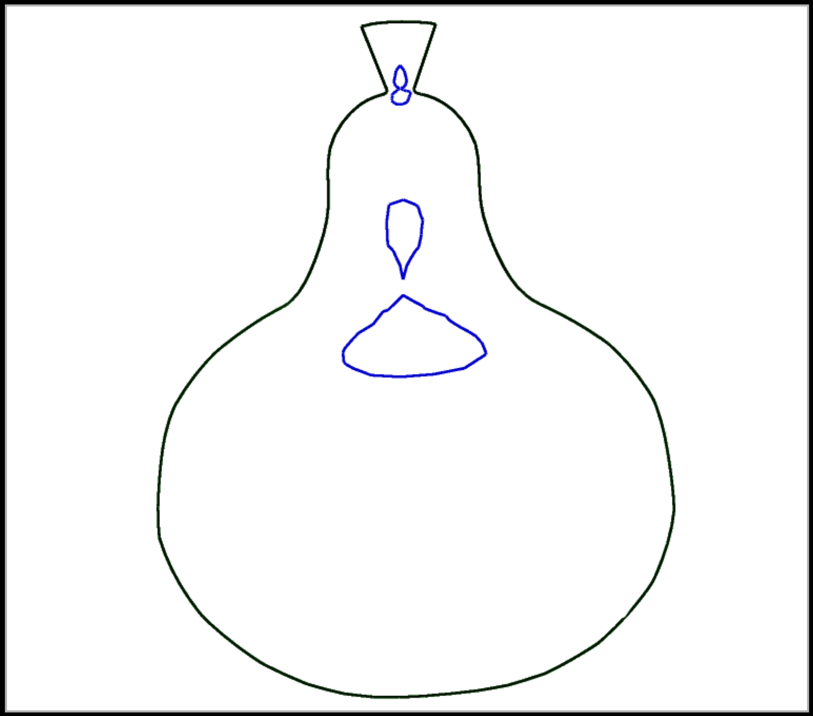


Mesh

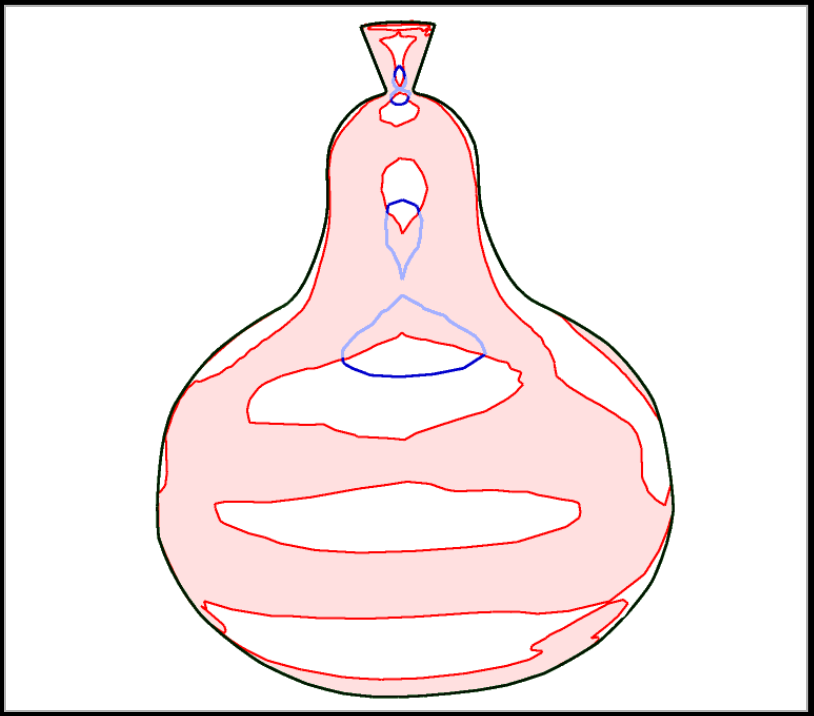


$\kappa_r = 0$

Suggestive Contours as Zeros of κ_r

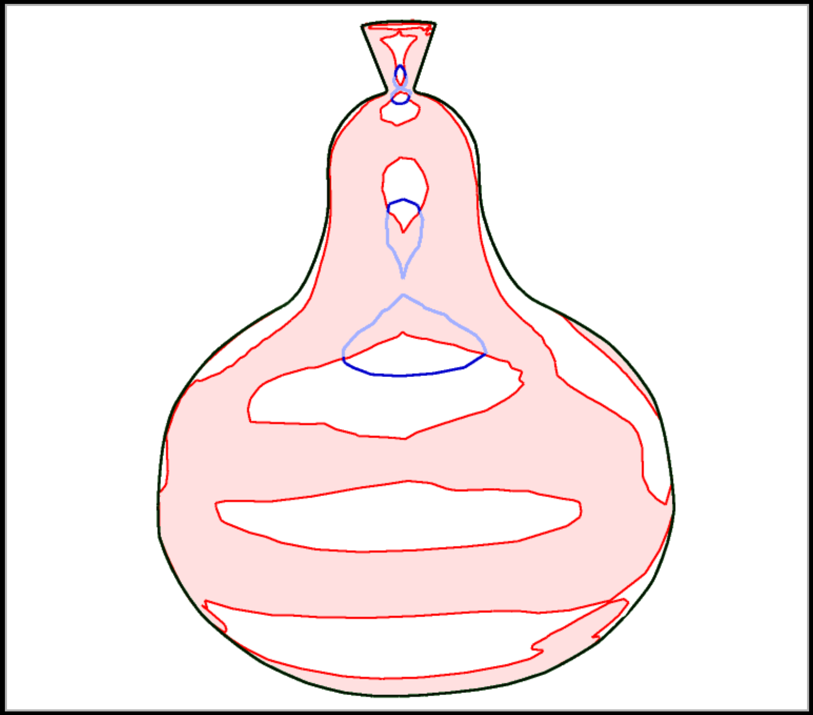


$\kappa_r = 0$

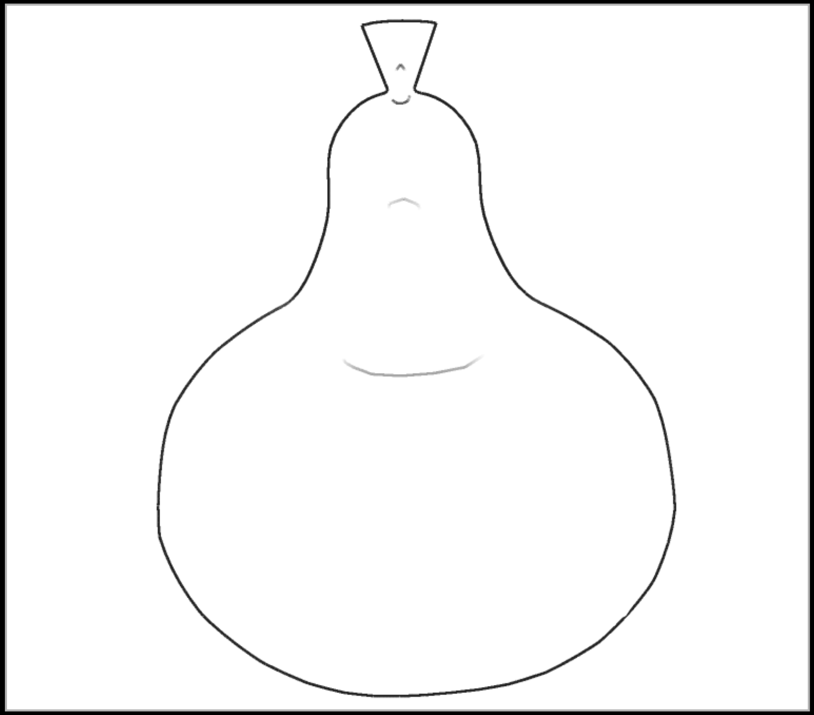


Reject if $D_w \kappa_r < 0$

Suggestive Contours as Zeros of κ_r



Reject if $D_w \kappa_r < 0$



Suggestive contours