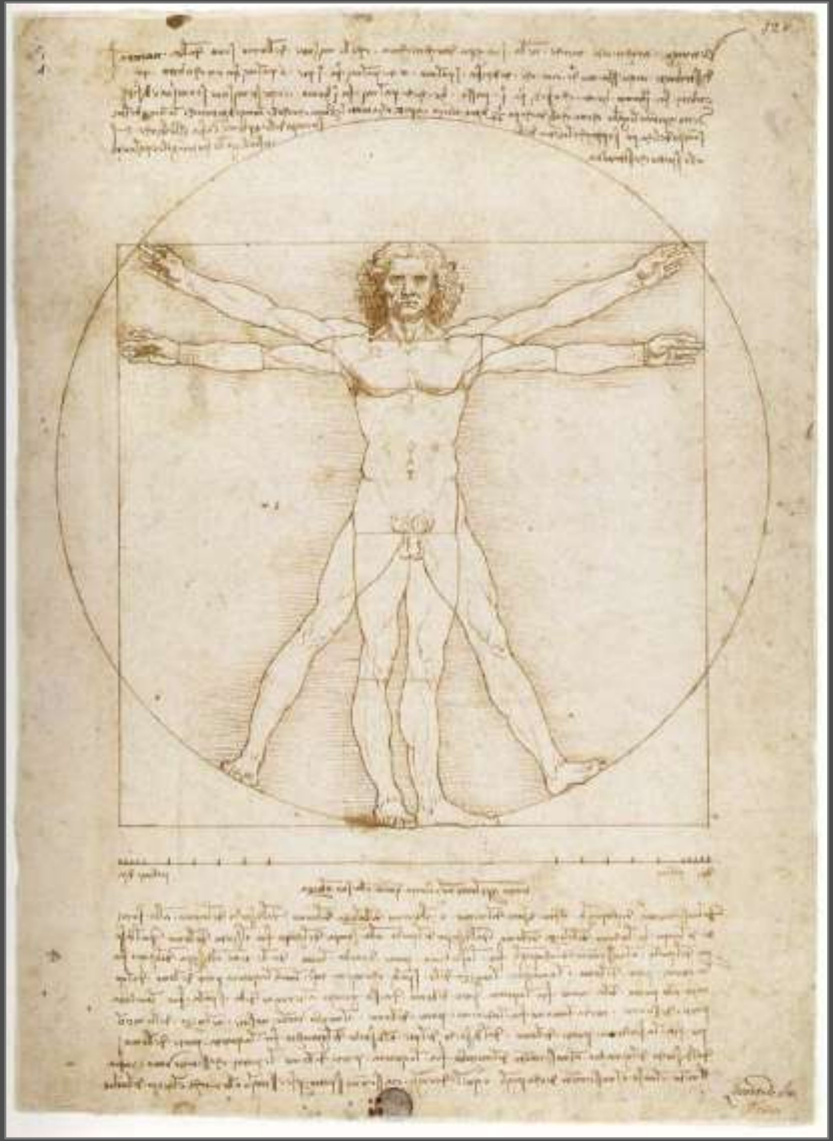
Non-Photorealistic Rendering

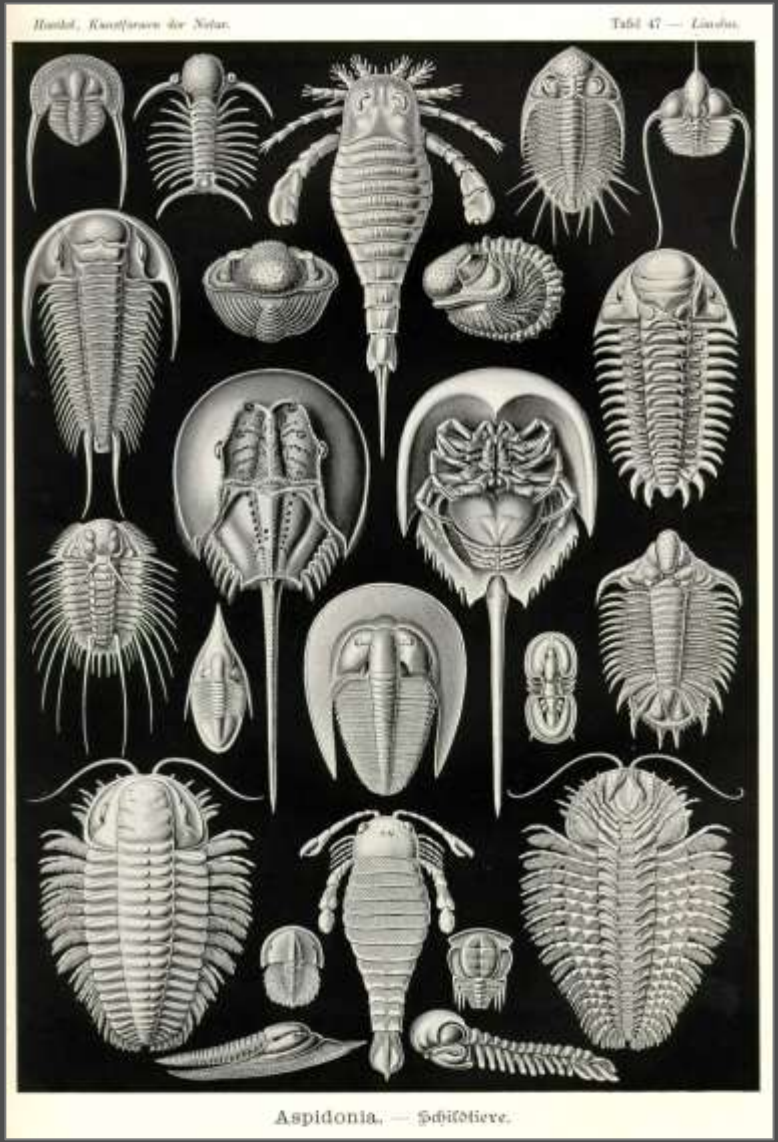
Sparse Line Rendering:
Silhouettes and Feature Lines

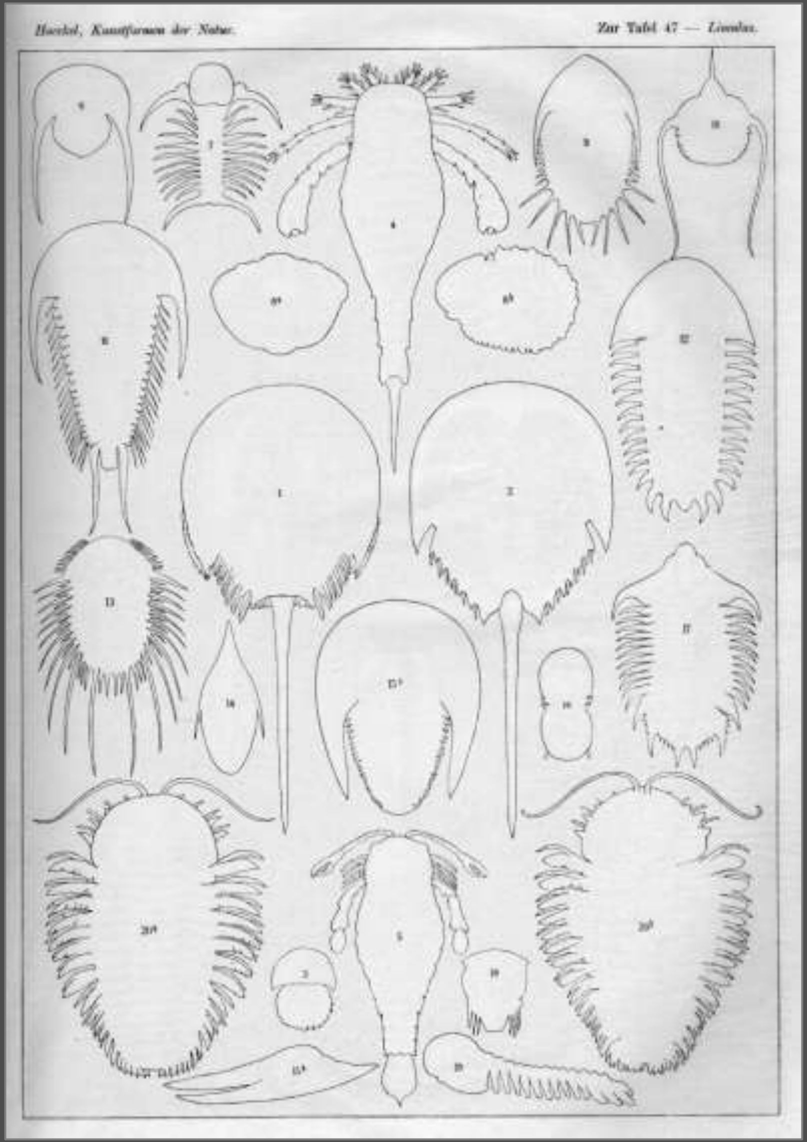
Tobias Isenberg

Sparse Lines vs. Hatching

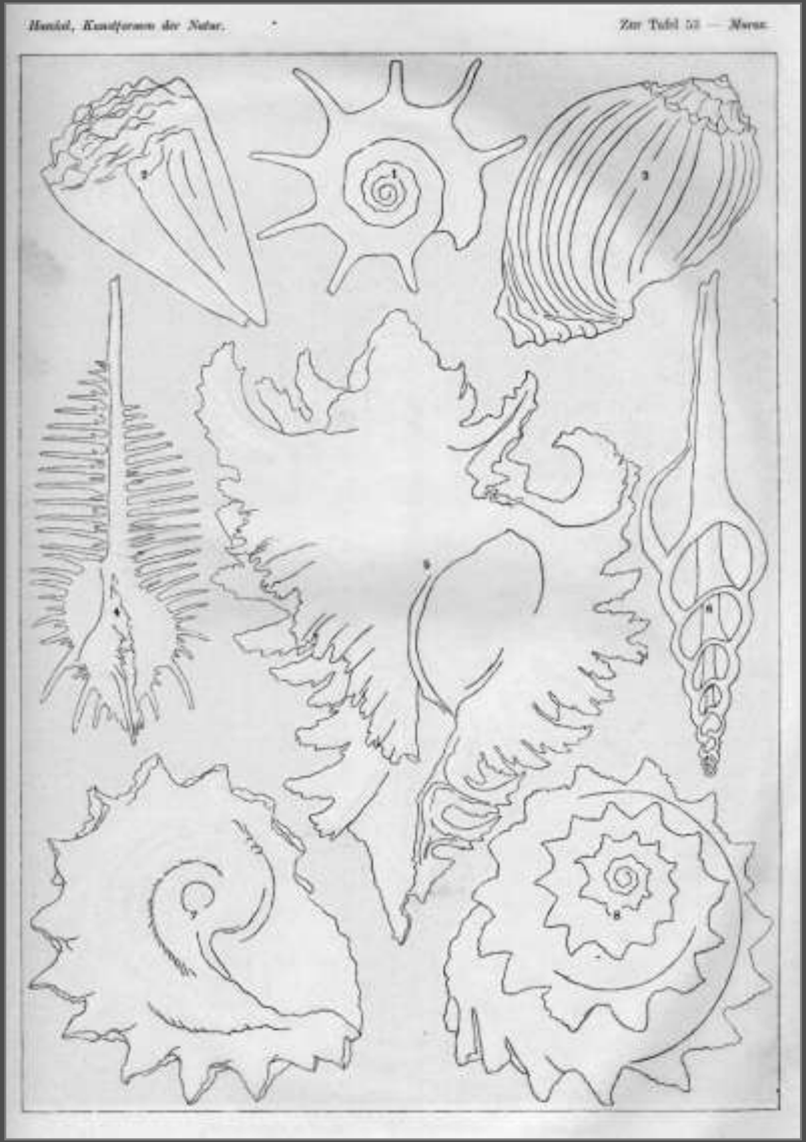


Sparse Lines vs. Rendering with Shading





Sparse Lines vs. Rendering with Shading

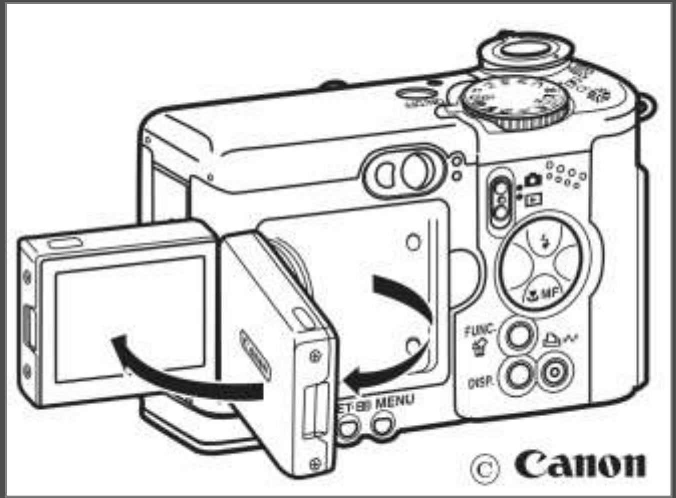


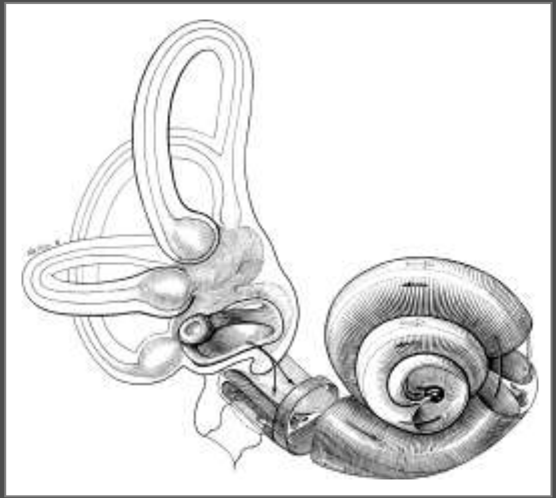
Structure of the Lecture

1. Introduction
2. Silhouettes for Polygonal Meshes
3. Hidden Line Removal for Object Space Silhouettes
4. Static and View-Dependent Feature Strokes
5. Stroke Processing
6. Miscellaneous Techniques
7. Summary

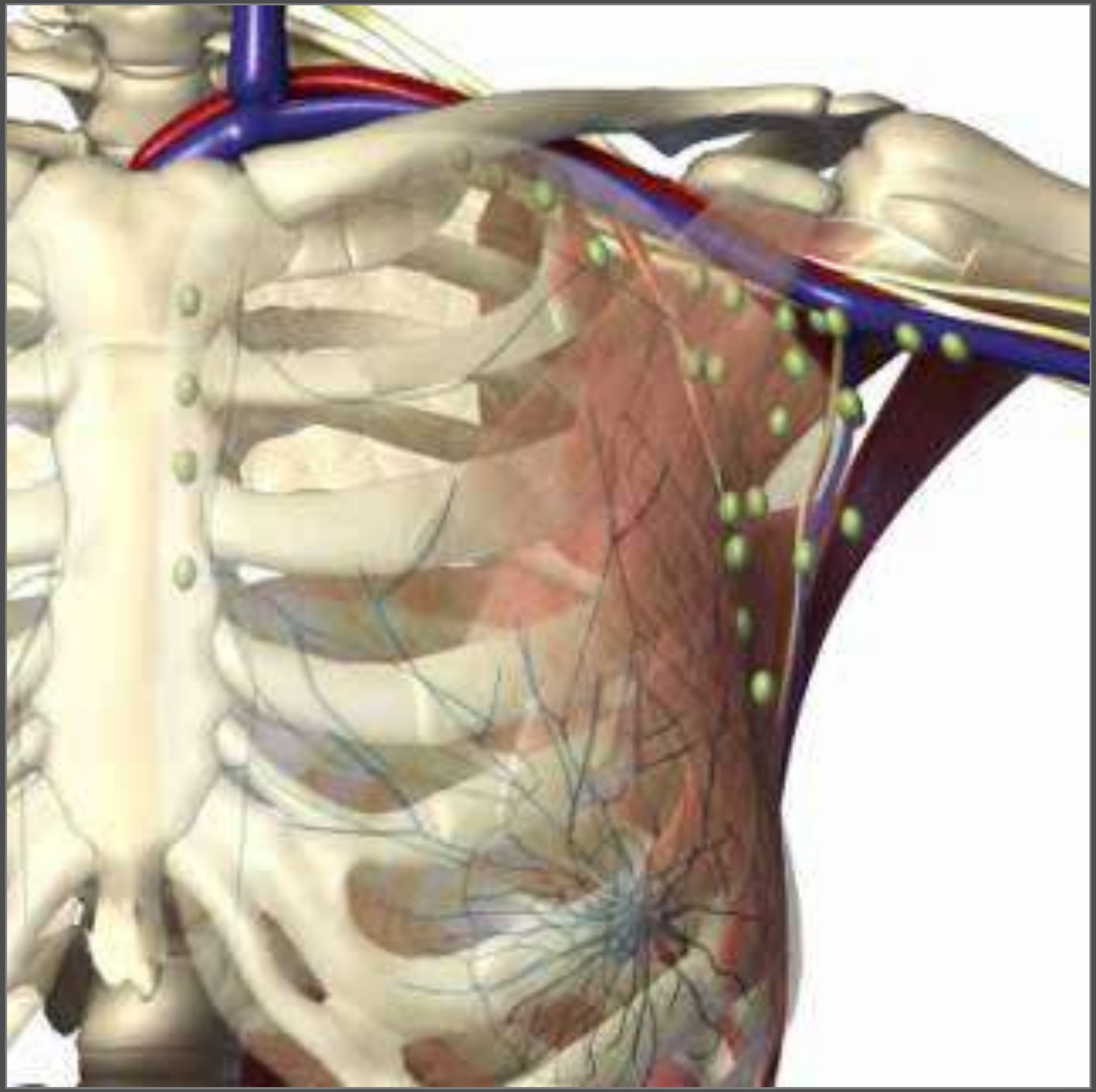
Sparse Line Drawings: Concept

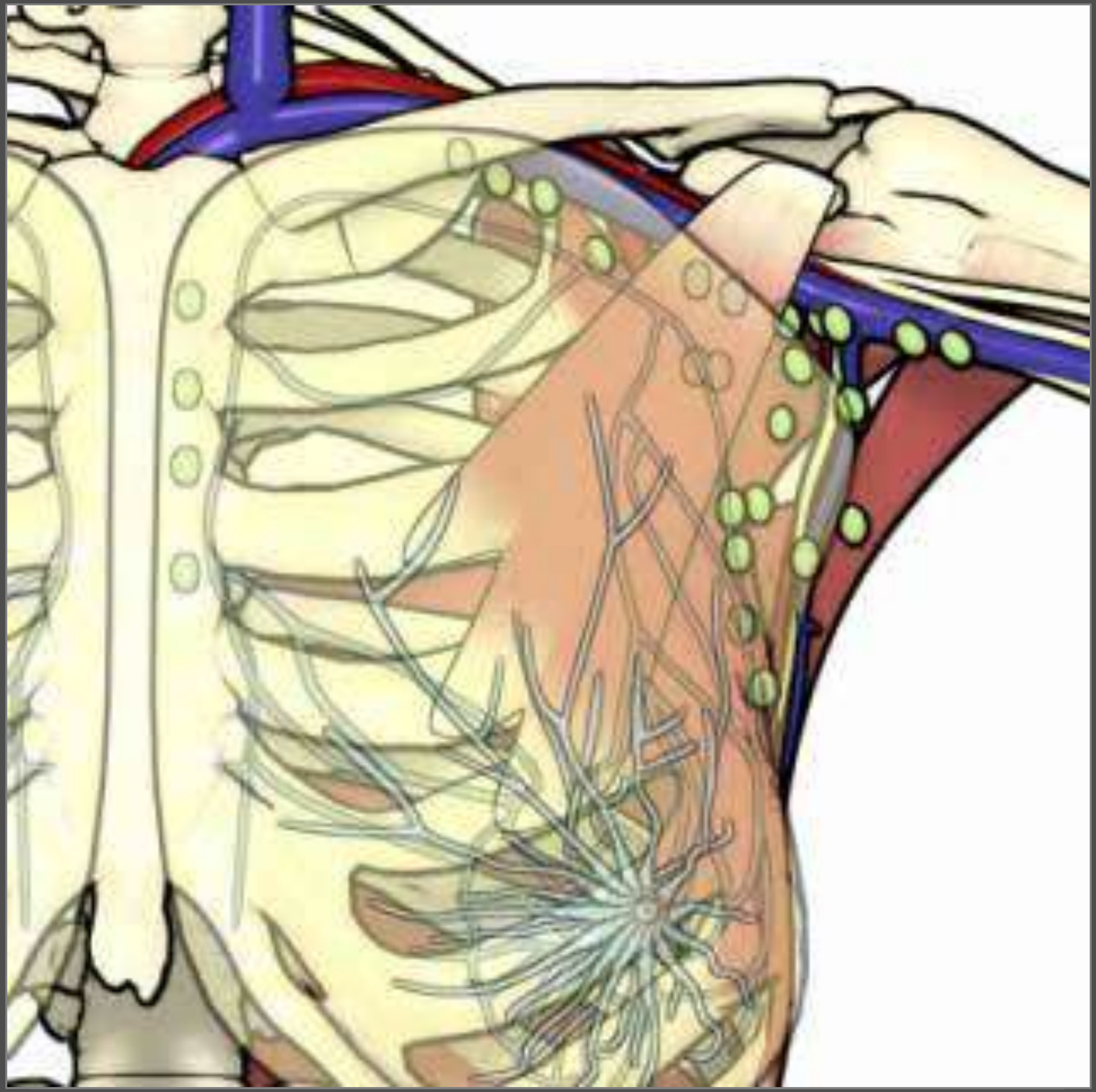
- renditions that are restricted to linear features
- silhouette/contour lines & feature lines only
- abstract: usually no shading or texture depicted
- application domains:
  - many traditional graphic depictions
  - e.g., in comics, technical and medical illustrations, assembly instructions, etc.
  - as additional layer for clarification in illustrations



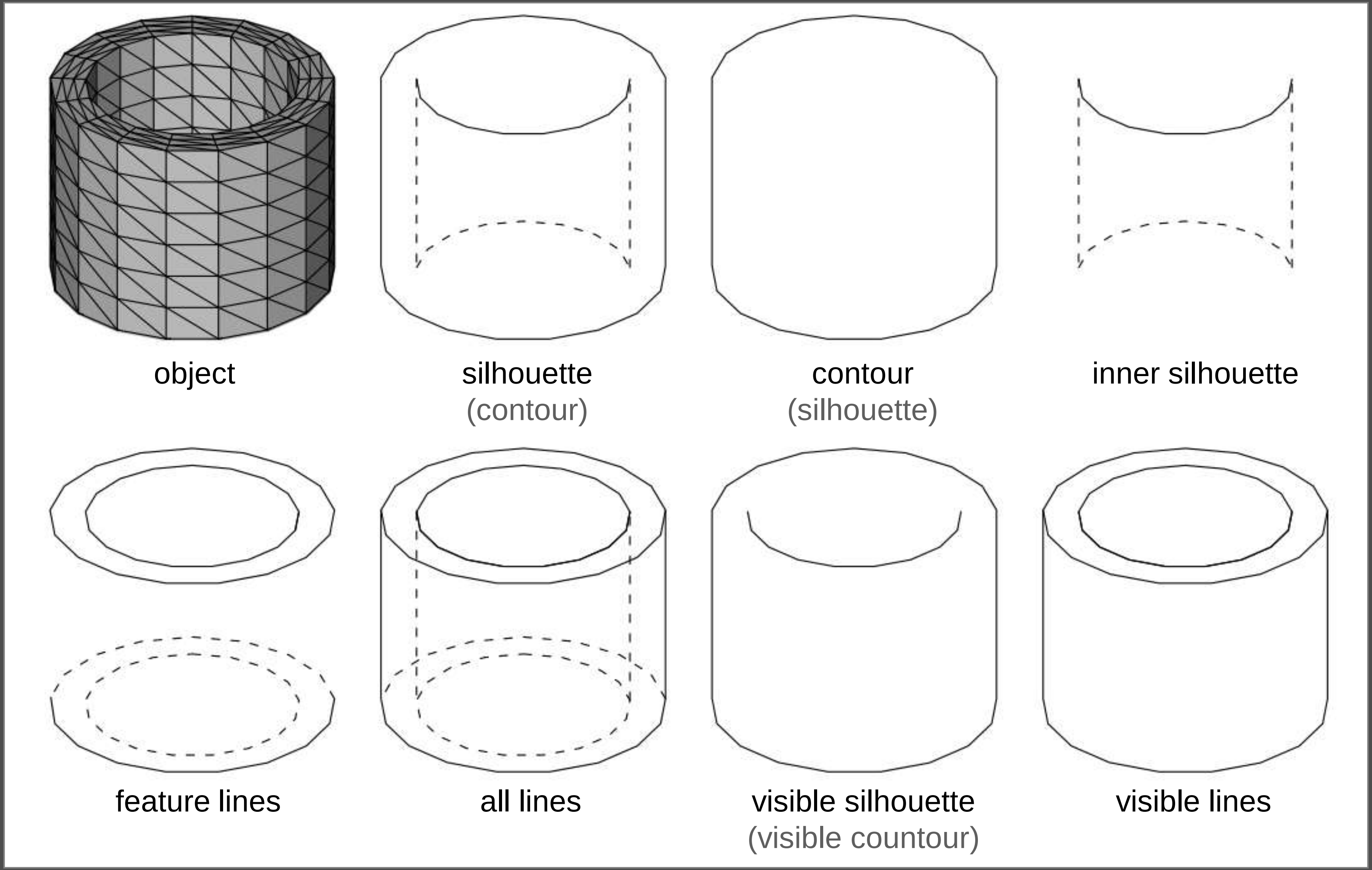


Sparse Lines as an Additional Layer for Clarification



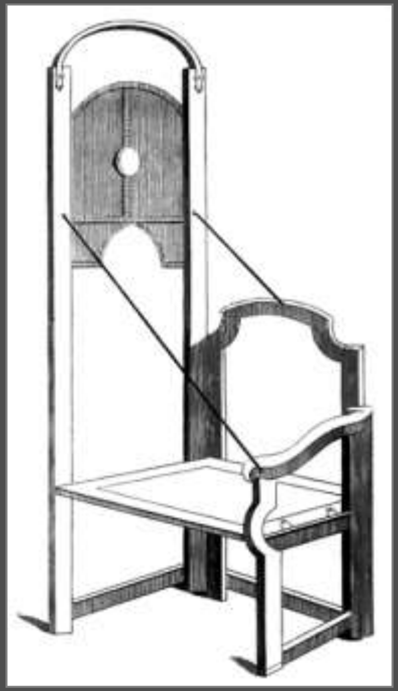


Sparse Line Drawings: Terminology



The Origin of the Term Silhouette

- papercut images in the 17th century in Europe, mostly profile images
- originating from Chinese art
- Étienne de Silhouette (1709–1767): Louis XV’s finance minister
- several well-known silhouette artists; e.g., August Edouart, John Meirs

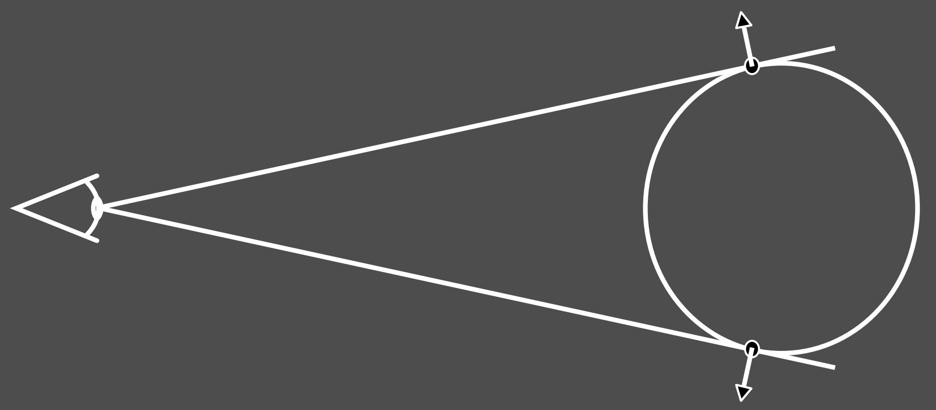



Silhouette: Definitions

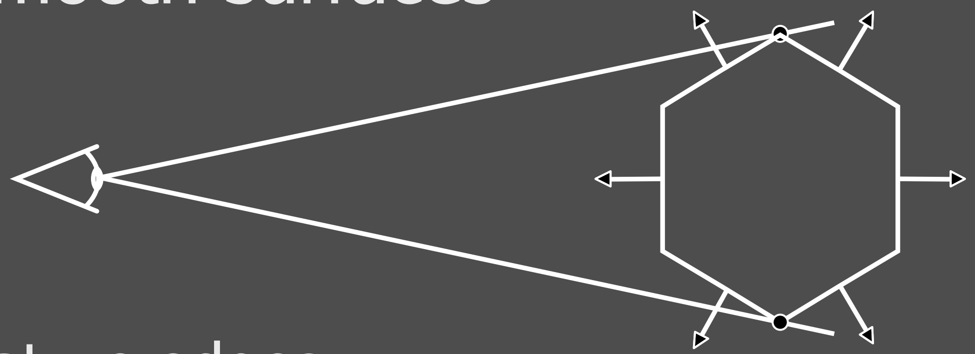
- silhouette (in general):
  set S of all points that are
  tangential as seen by a viewer

  $S = \{P : 0 = \vec{n_i} \cdot \vec{v}\}$ (orth.proj.)

  $S = \{P : 0 = \vec{n_i} \cdot (\overrightarrow{p_i - c})\}$ (persp.proj.)



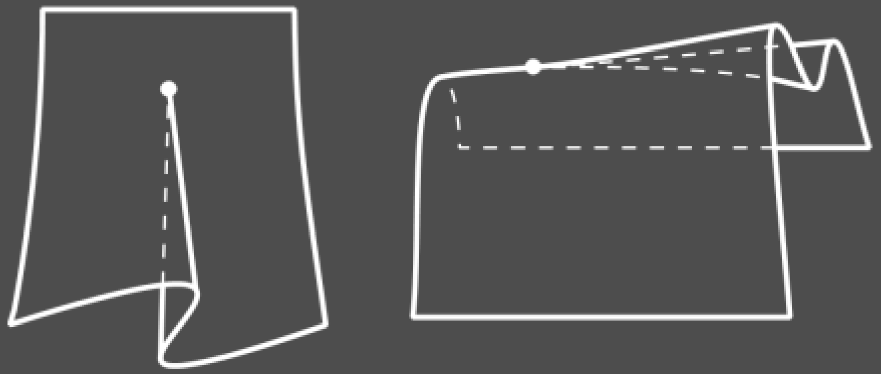
- polygonal shapes & discontinuities on smooth surfaces
  - one front-facing polygon/surface and
  - one back-facing polygon/surface

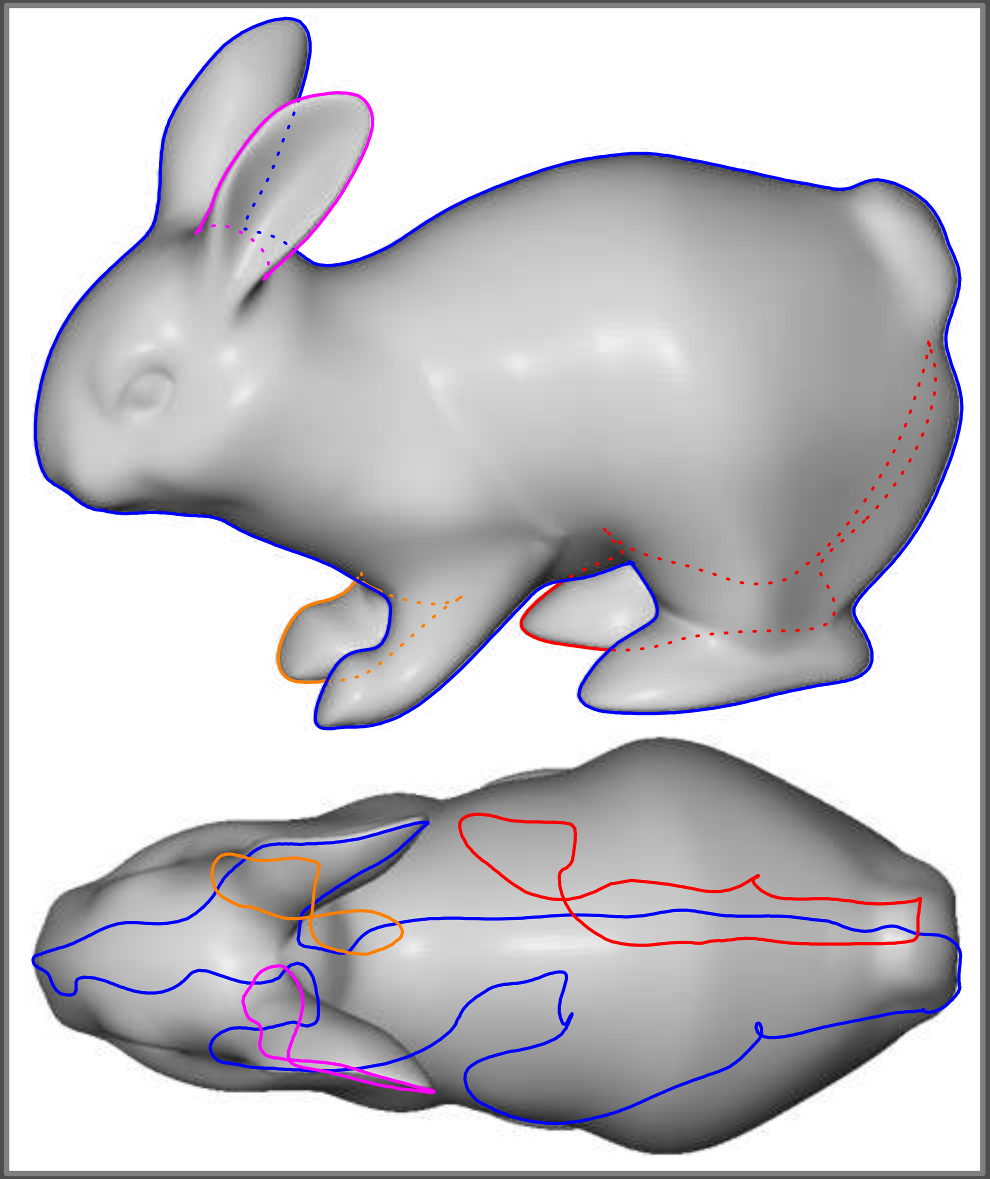


- silhouette/feature strokes
  - set of a number of connected silhouette/feature edges
  - at most two edges connected to the same vertex

Silhouette: Properties

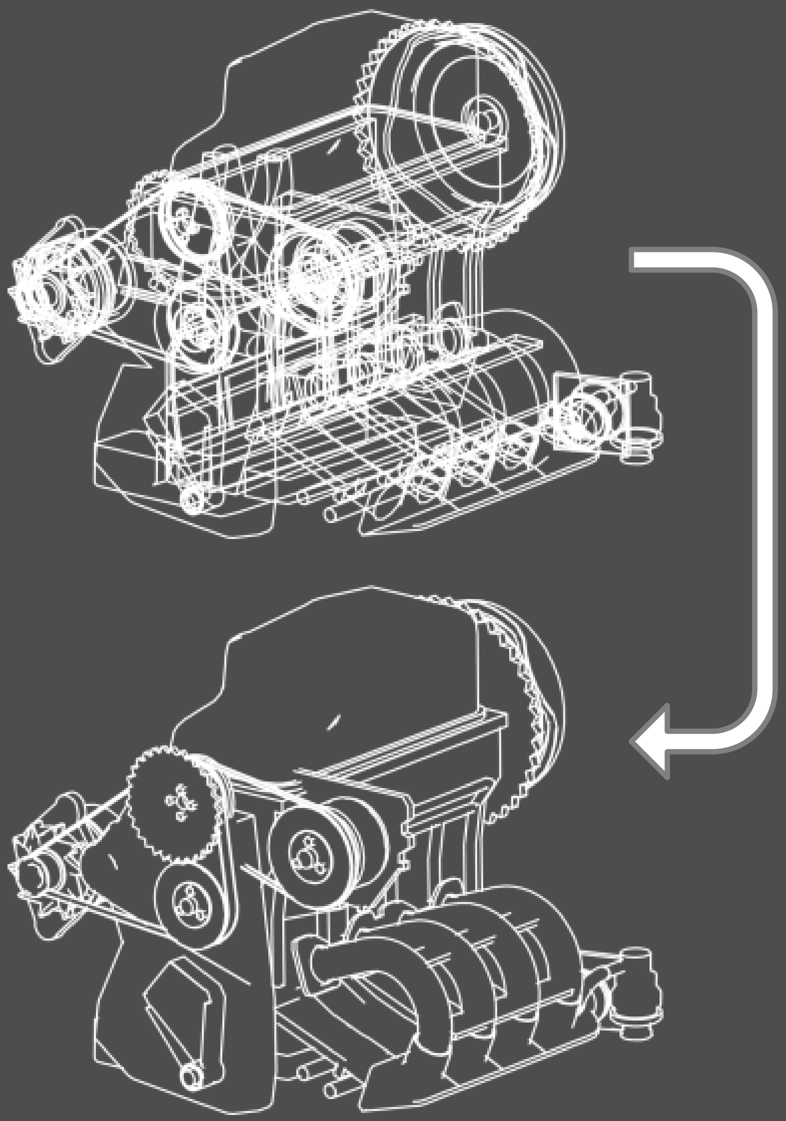
- closed silhouette loops on the surface (for closed surfaces)
  - loops may be entirely visible, partially visible, or entirely hidden
- cusps: visibility changes where silhouette is co-linear to the viewing direction
  - even number of silhouette curves emerging from cusp



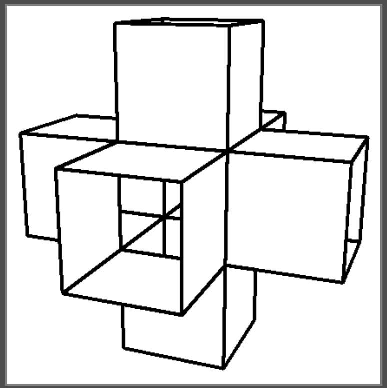


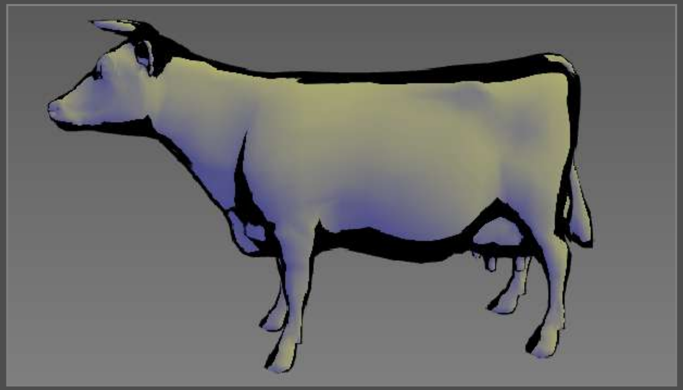
Silhouette Algorithm

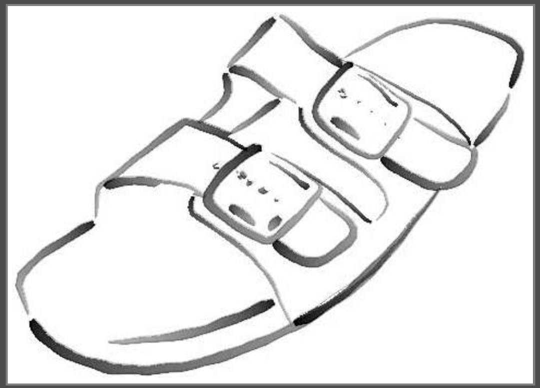
- two main tasks
  - detection of potential silhouette edges
  - hidden line removal (visibility culling)
- some algorithms perform both tasks in one step
- computation per frame necessary (both tasks)
  $\rightarrow$ often high run-time complexity
- need for algorithms that speed this up
  - allow small inaccuracies
  - pre-processing, smart data structures
  - stochastic techniques



Silhouettes for Polygonal Meshes

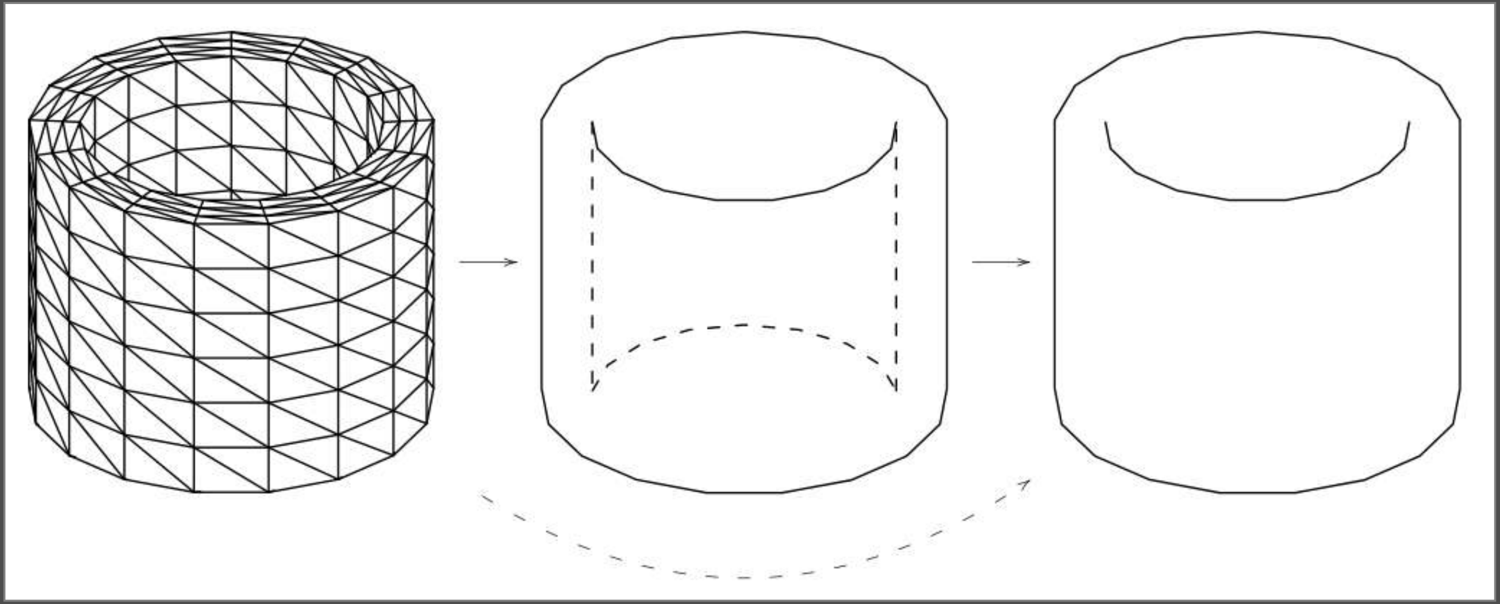






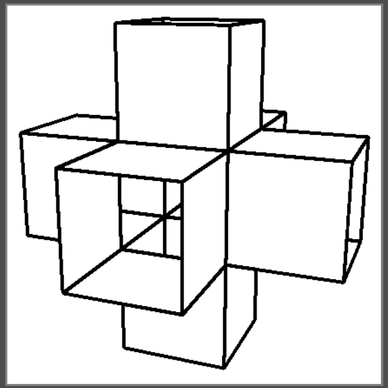
Silhouettes of Polygonal Meshes

- input: polygonal mesh
- output: (visible) silhouette edges
  - as pixel image
  - as set of edges
- intermediate step of hidden line removal not always necessary

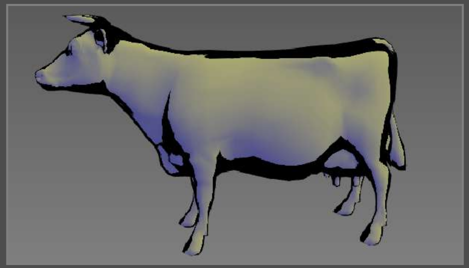


Algorithm Classification

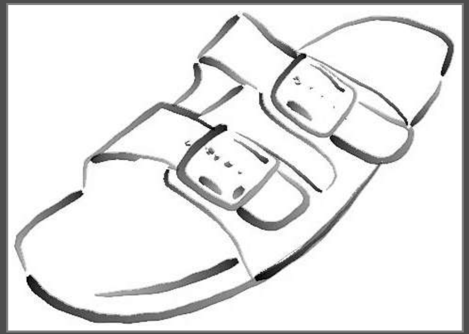
- algorithms in image-space
  - work with rendered pixel buffers
  - silhouette detection and visibility culling simultaneously
- hybrid algorithms
  - manipulations in object-space
  - normal rendering to get silhouettes in image-space
  - silhouette detection and visibility culling simultaneously
- algorithms in object-space
  - silhouette detection without rendering
  - silhouette detection and visibility culling separately
- visibility culling for object-space silhouettes
  - in image-space, in object-space, hybrid algorithms



Hertzmann (1999)



Gooch et al. (1999)



Northrup & Markosian (2000)

Image-Space Algorithms

- goal: extract significant edges from a model
- utilization of traditional rendering techniques
- usage and manipulation of rendered G-buffers

→ application of algorithms from image processing

→ in particular, image filtering for edge detection

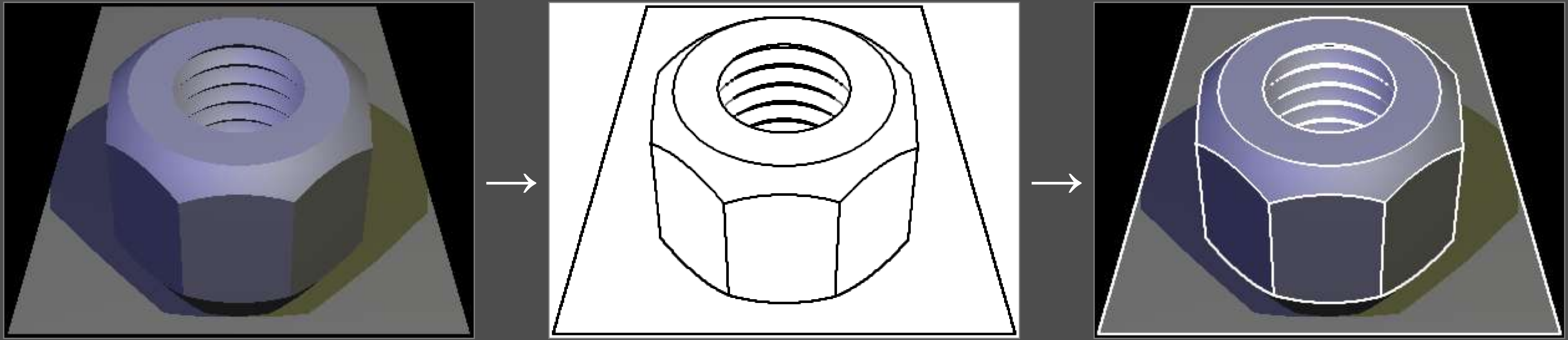
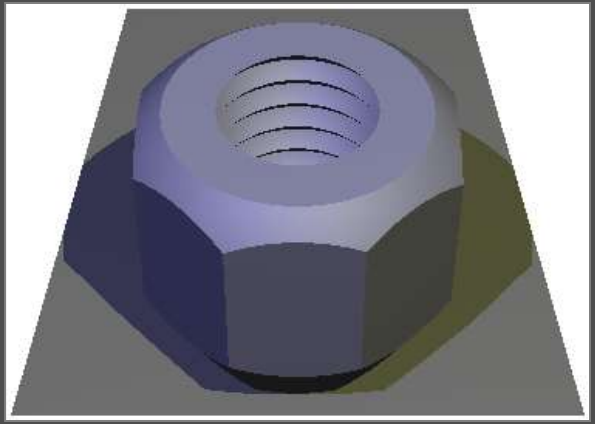
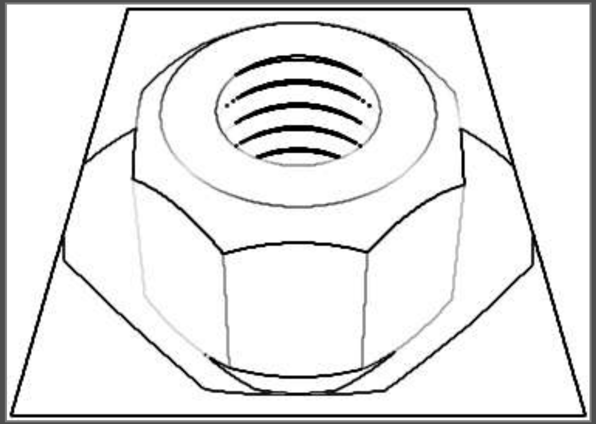
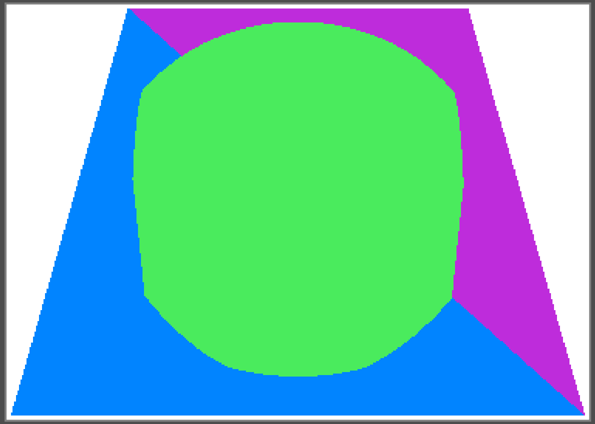


Image-Space Algorithms

- discontinuities in the frame buffer
  - not well suited for silhouette detection
  - unwanted edges detected at discontinuities; e.g., due to textures, shadow, or shading
  - some essential edges not detected, e.g., due to shadows
- discontinuities in object ID buffer
  - yields only contour/outline of objects
  - potential artifacts if model erroneous e.g., one object modeled in two parts with two different object IDs


frame buffer


edges


object ID buffer

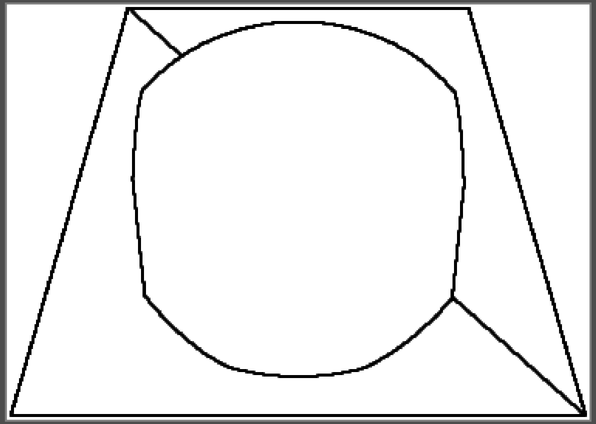
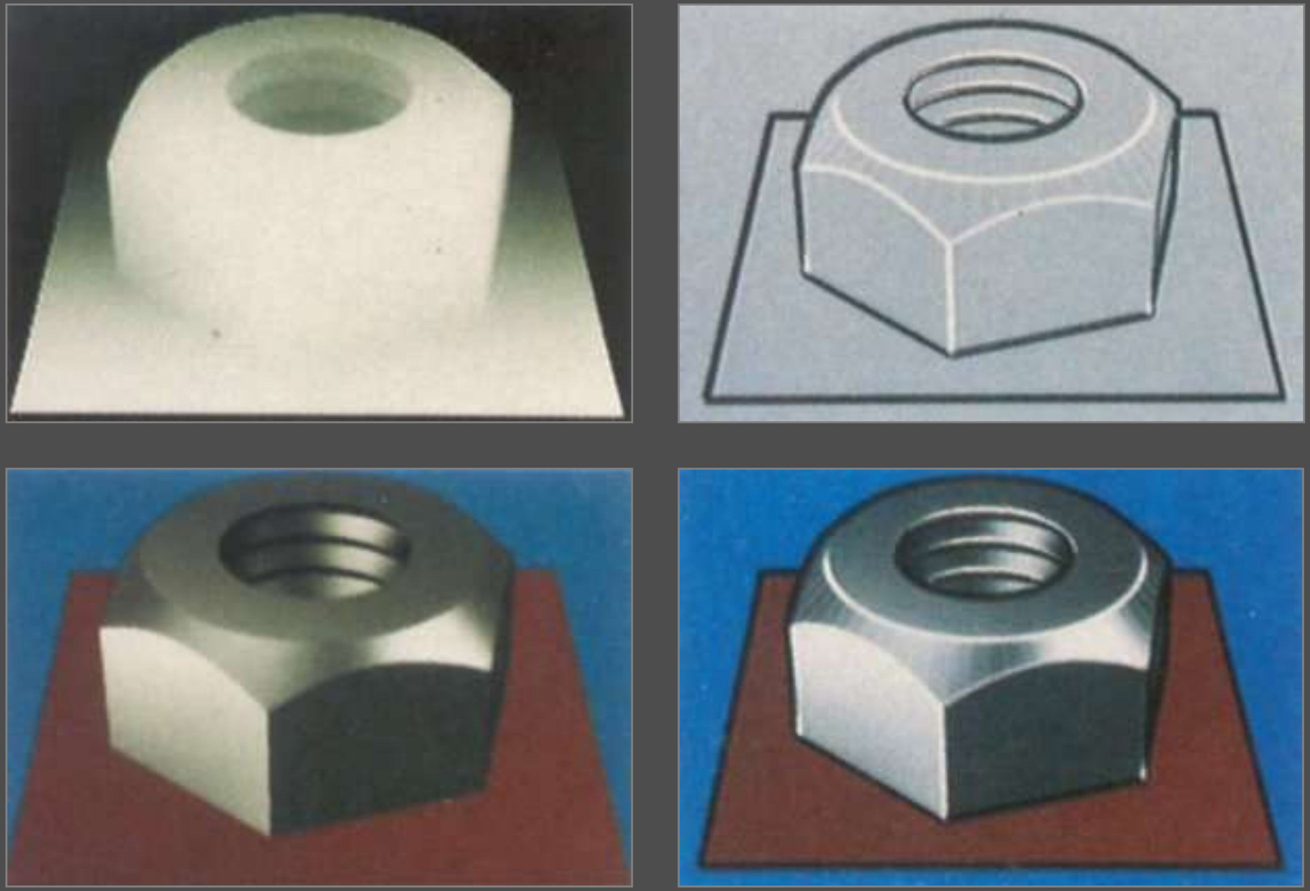

edges

Image-Space Algorithms: z-Buffer

- Saito & Takahashi (1990): edge detection in the z-buffer
  ⇒ only looking at discontinuities of depth values

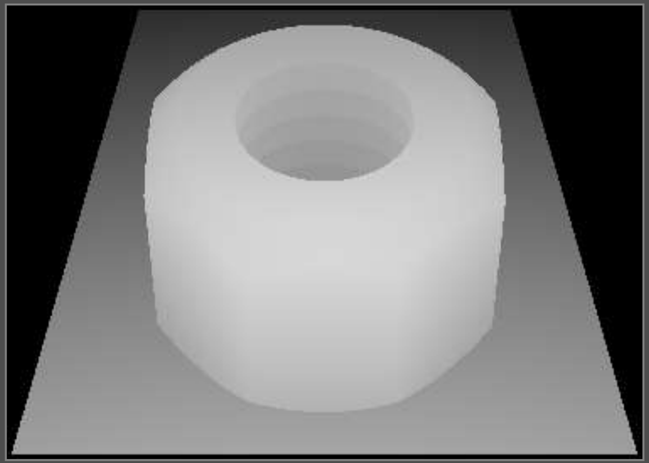


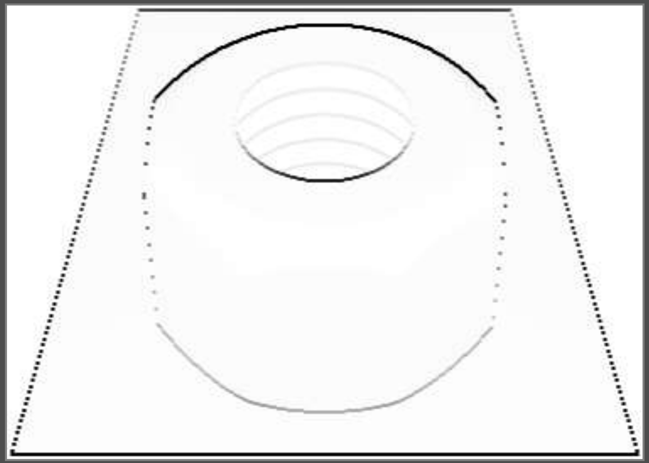
Saito & Takahashi (1990)

Image-Space Algorithms: z-Buffer

- edge detection operators from image processing
  - Sobel operator (1^{st} derivative), e.g.,

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$





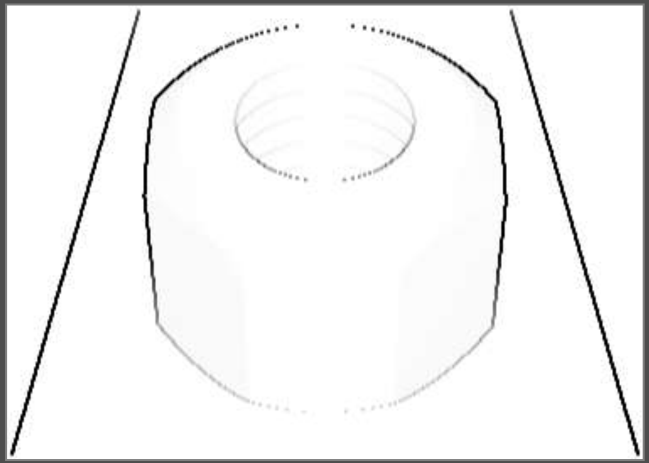


Image-Space Algorithms: z-Buffer

- edge detection operators from image processing
  - Laplace operator (2nd derivative), e.g.,

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

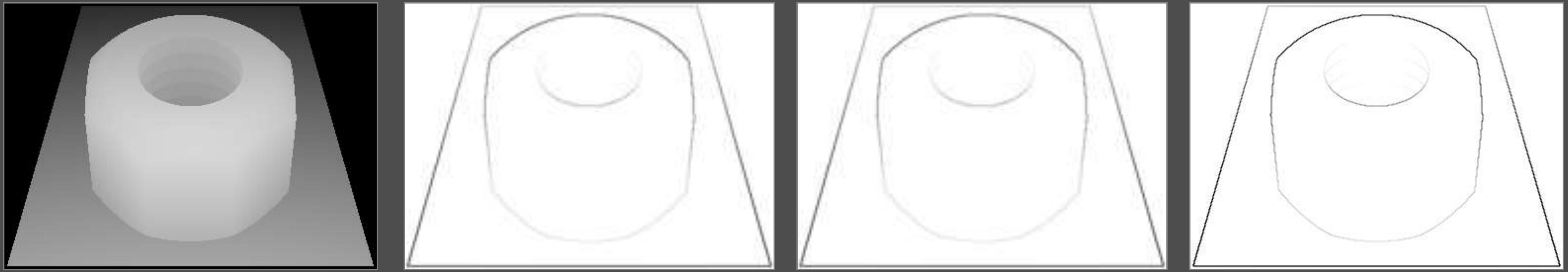


Image-Space Algorithms: Normal Buffer

- extension of the principle to normal buffer
  - Decaudin (1996) & Hertzmann (1999)
- normal buffer contains normal direction
  - x-, y-, and z-components as RGB values
- edge detection operator on normal buffer (e.g., Laplace)
- application of edge detection operator
  - individually per RGB channel or gray value derived from maximum RGB color

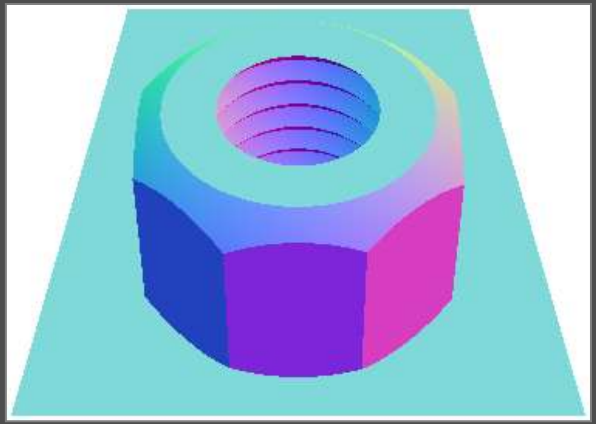
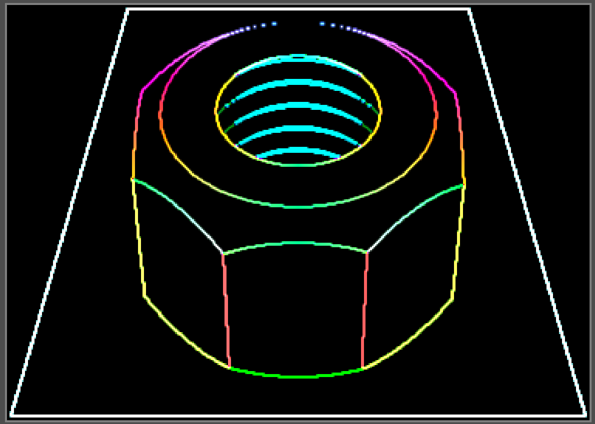
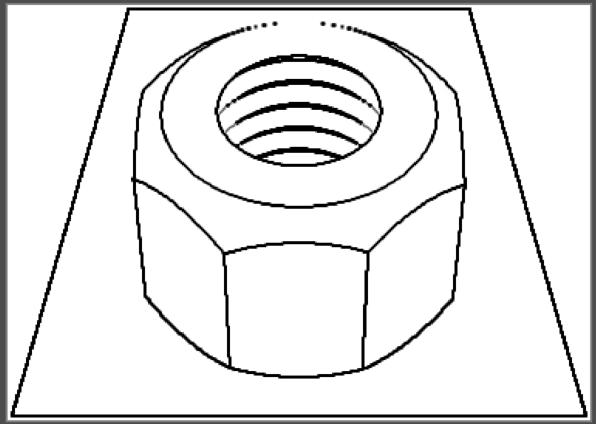




Image-Space Algorithms: Combination

- all edges neither through z-buffer nor normal buffer algorithm
  ⇒ combination both z-buffer and normal buffer edge detection
  → combination of the buffers with edges into one image
- all desired edges are present in the combined buffer

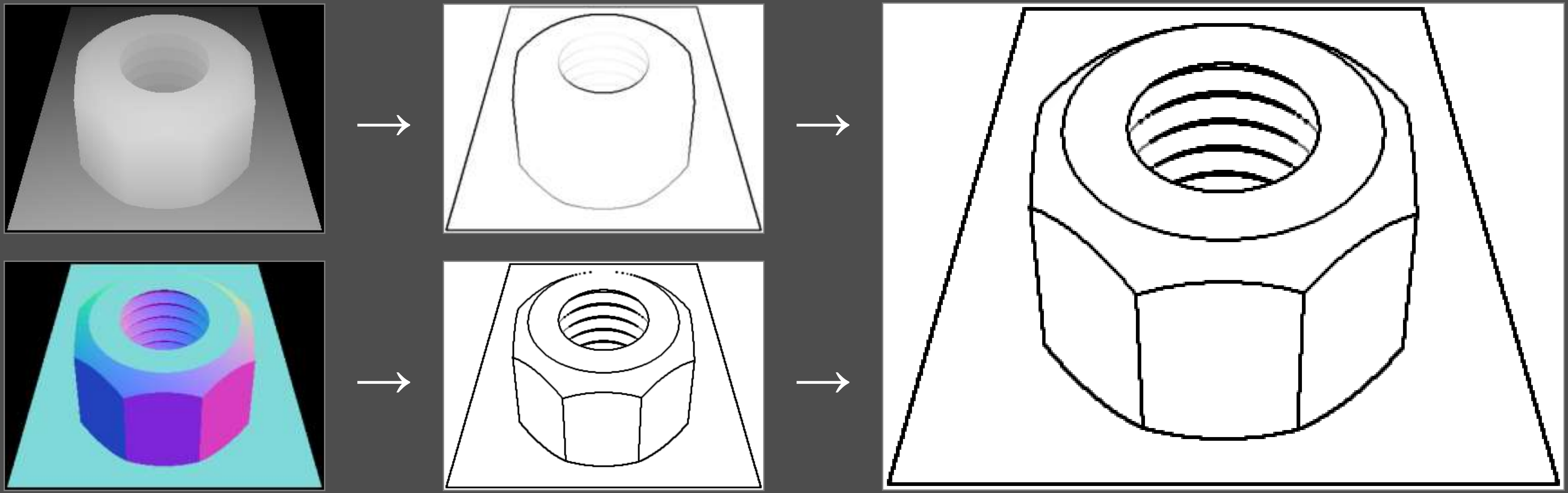


Image-Space Algorithms: Example

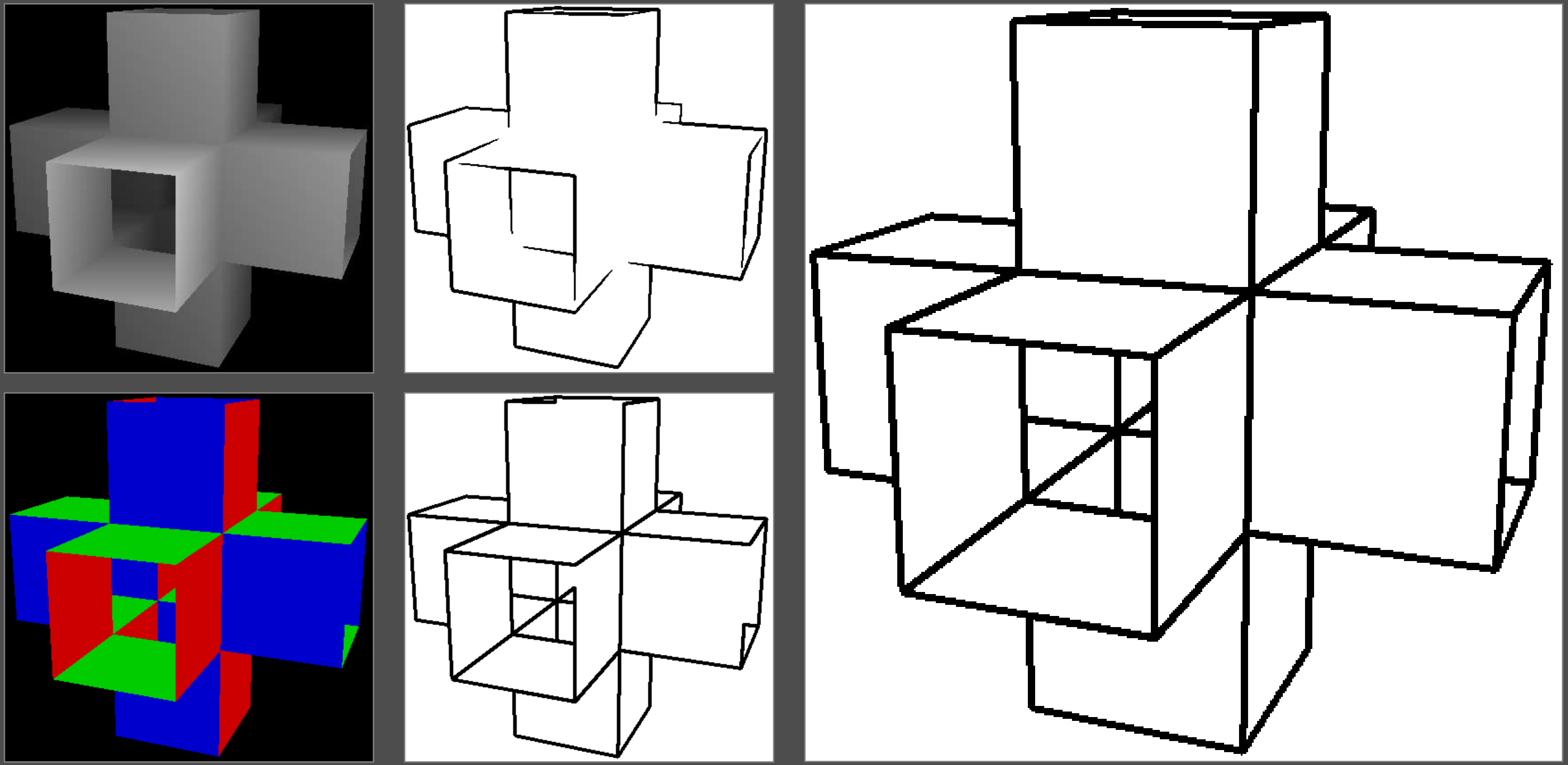
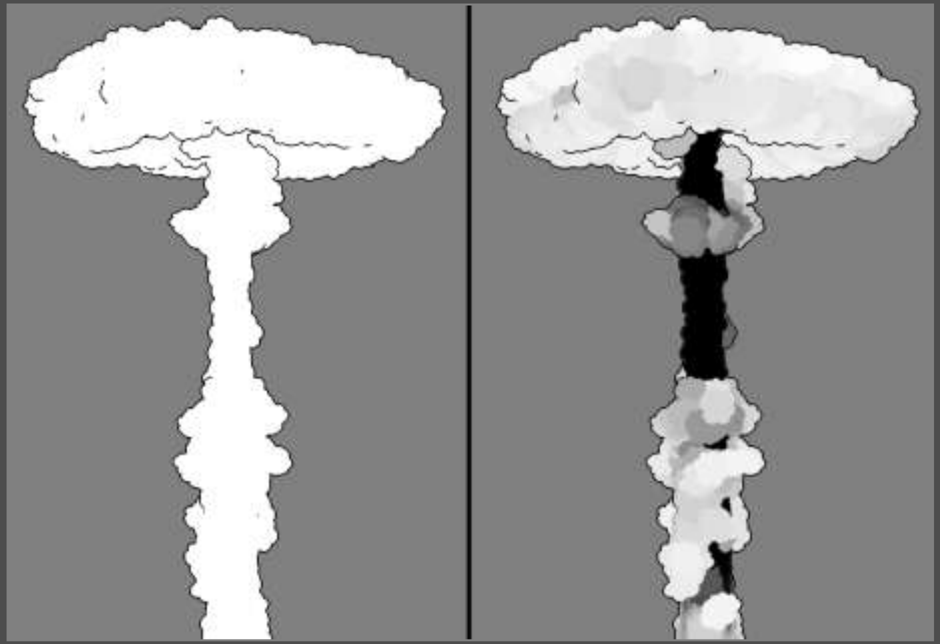


Image-Space Algorithms: Special Techniques

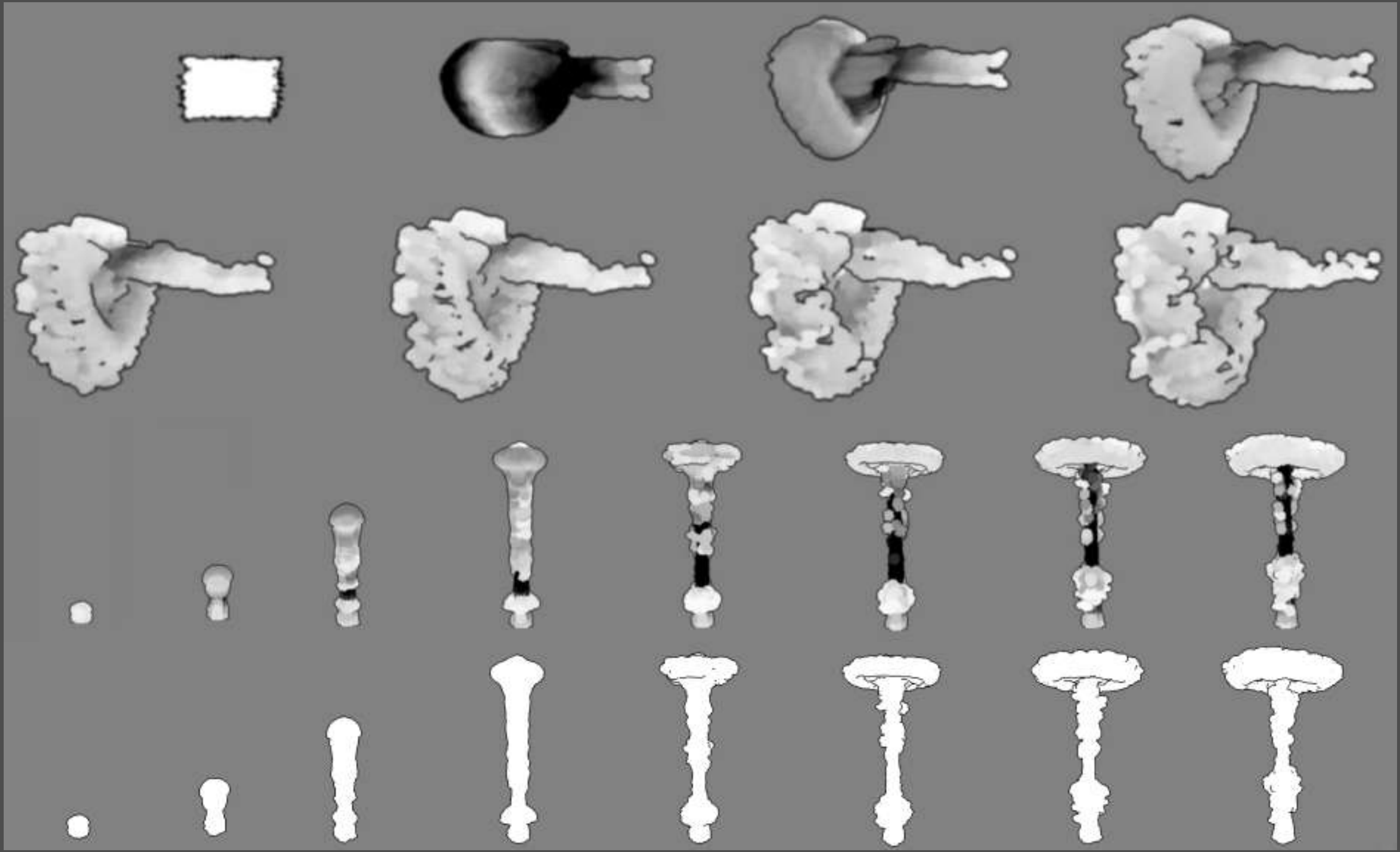
- tree rendering through z-buffer discontinuities of primitives
  - view-aligned disks represent leaves or groups of leaves
  - stylizable through disk size & number
- cartoon rendering of smoke animations
  - data from physical simulations
  - technique similar to previous one
  - z-distance compared to threshold
  - differently shaped smoke primitives → sketchy primitives yield sketchy shapes
  - indication of velocity by primitive color

Deussen & Strothotte (2000)



Selle et al. (2004)

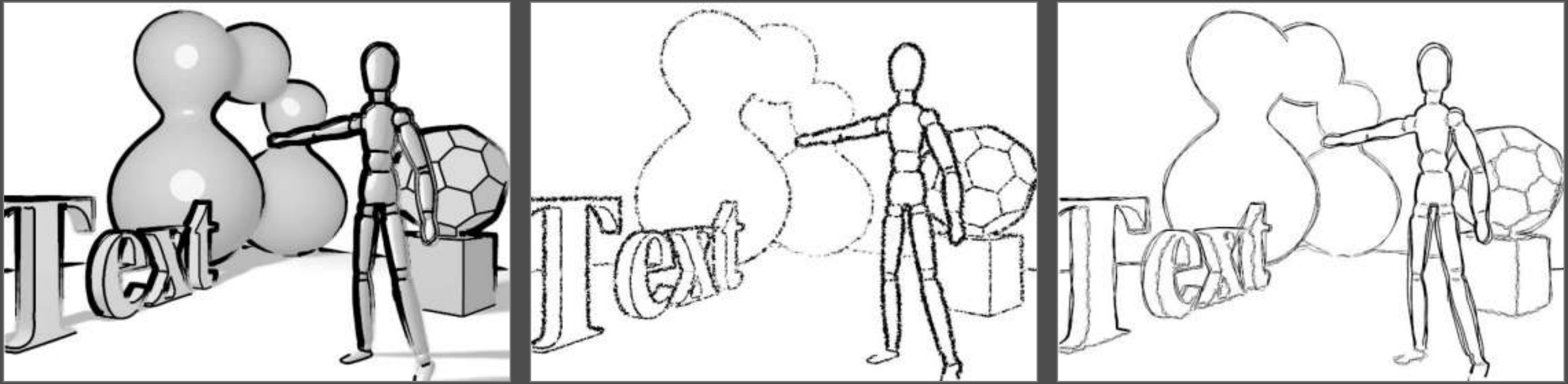
Image-Space Algorithms: Special Techniques



Selle et al. (2004)

Image-Space Algorithms: Special Techniques

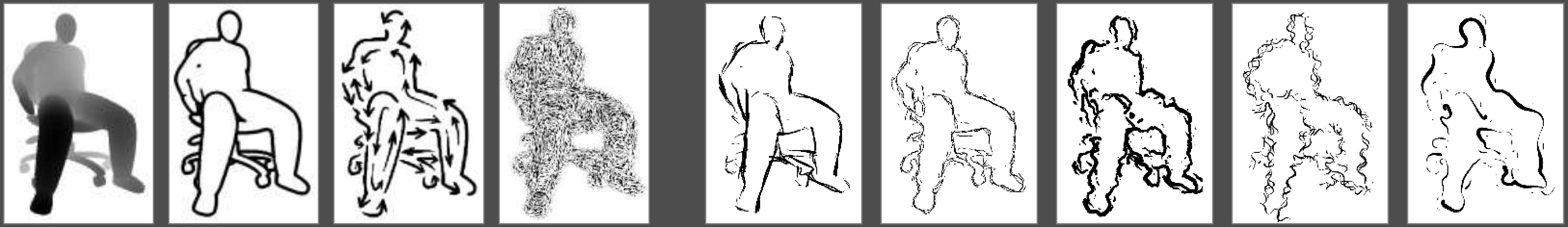
- hardware acceleration (Mitchell, 2002)
  - implementation of z-buffer and normal technique as pixel shader ⇒ real-time frame-rates (at the time!)
- vector graphic extraction from edge image (Loviscach, 2002)
  - iteratively connecting segments to create curves to match the pixel silhouettes
  - fairly slow, not for interactive or real-time purposes, but allows stylization



Loviscach, (2002)

Image-Space Algorithms: Special Techniques

- loose and sketchy animation (Curtis (1998), Ho and Komiya (2004))
  - stochastic, physically-based particle system based on z-buffer image of object
    → buffer with edges (“template”), thickened and blurred
    → force field buffer (vector field perpendicular to gradient of z-buffer)
  - generation of n particles at random positions where ink needs to go
  - animation according to force field, random movement for sketchy appearance
  - if particle leaves ink area it dies and another one is born elsewhere

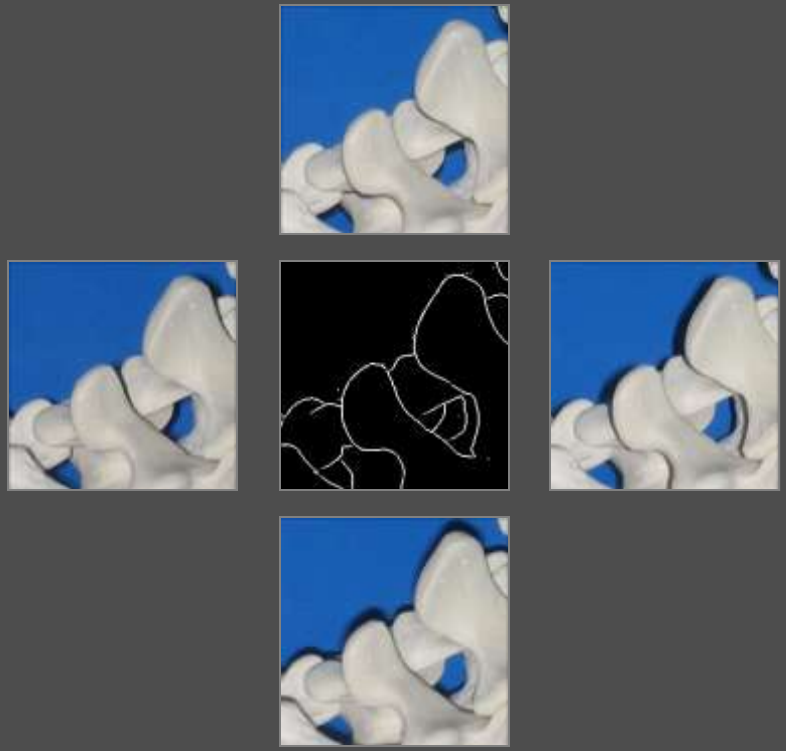


Curtis (1998)

Image-Space Algorithms: Special Techniques

Image-Space Algorithms: Special Techniques

- digital NPR-Camera (Raskar et al., 2003, 2004)
  - 4 flashes on a digital camera, 4 individual images
  - derives edges from image parallax
  - shadow is attached to each z-edge
  - edges detected in each individual image
  - result composited into edge image
  - extra NPR effects added to real image, composited
  - animation possible (synchronized flashes)



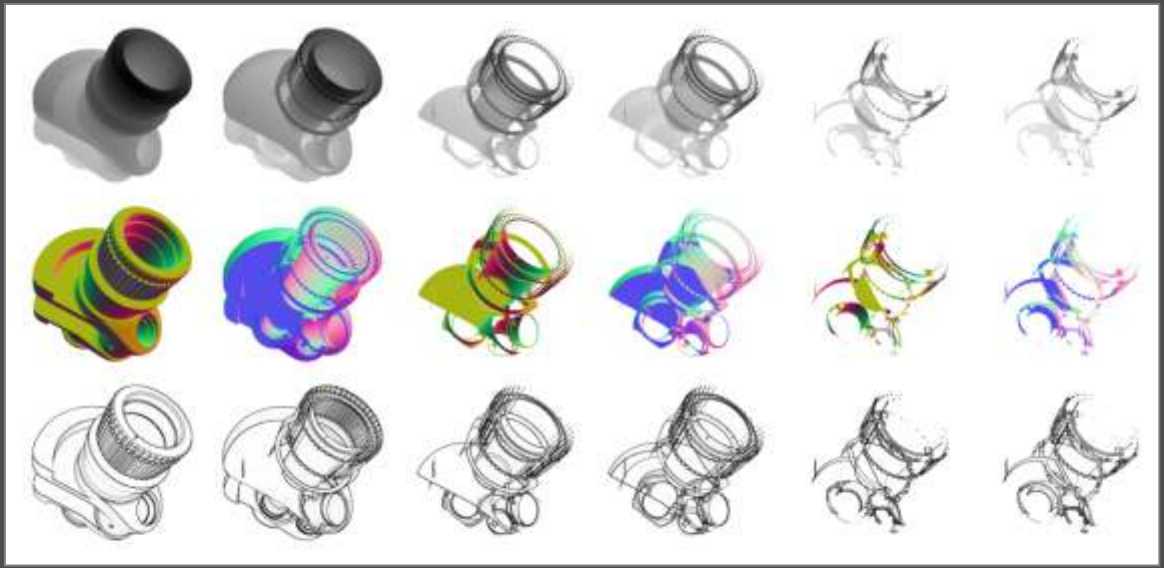
Raskar et al. (2004)

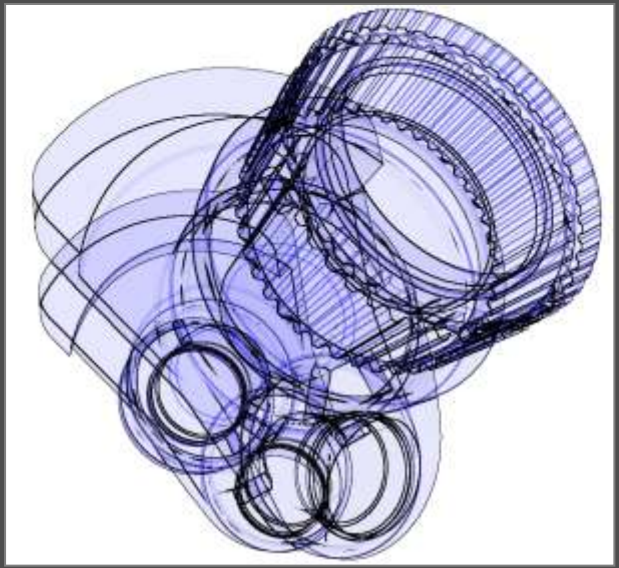
Image-Space Algorithms: Special Techniques

Raskar et al. (2004)

Image-Space Algorithms: Special Techniques

- Nienhaus & Döllner (2004): image-space blueprints
  - showing hidden lines with image-space techniques
  - silhouette and feature lines through z-buffer and normal buffer
  - iterative depth peeling by comparing the current z-depth to the previous one
  - ⇒ n layers of z-buffer and normal buffer → regular silhouette technique
  - → combination of the individual edge buffers to yield the result

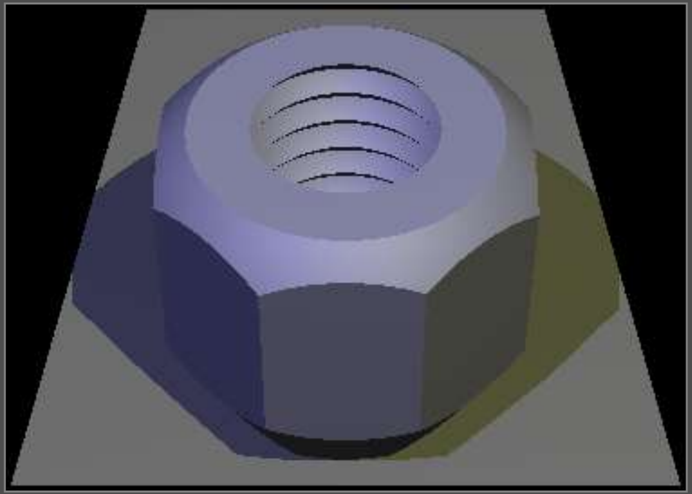




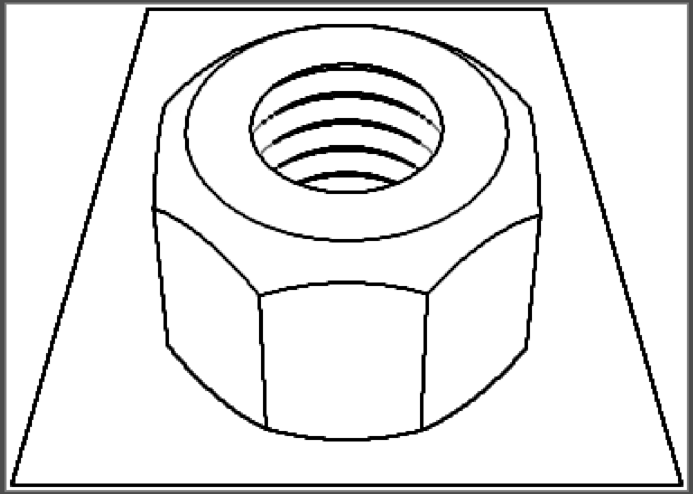
Nienhaus & Döllner (2004)

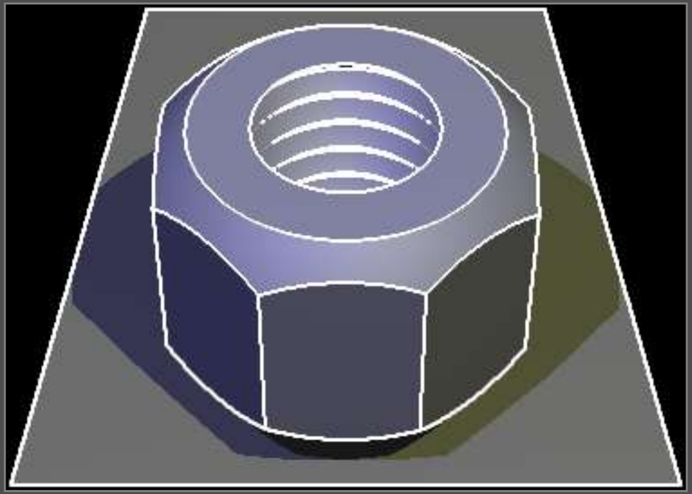
Image-Space Algorithms: Summary

- positive:
  - fast, "constant" run-time complexity (depends on pixel count, not on triangles)
  - easy implementation, hardware acceleration possible
  - hidden line removal automatically
  - feature lines detected automatically



- negative:
  - result only as pixel matrix ⇒ only pixel accuracy
  - only little control of result
  - silhouettes difficult to stylize, further processing difficult





Hybrid Algorithms

- two-step process:

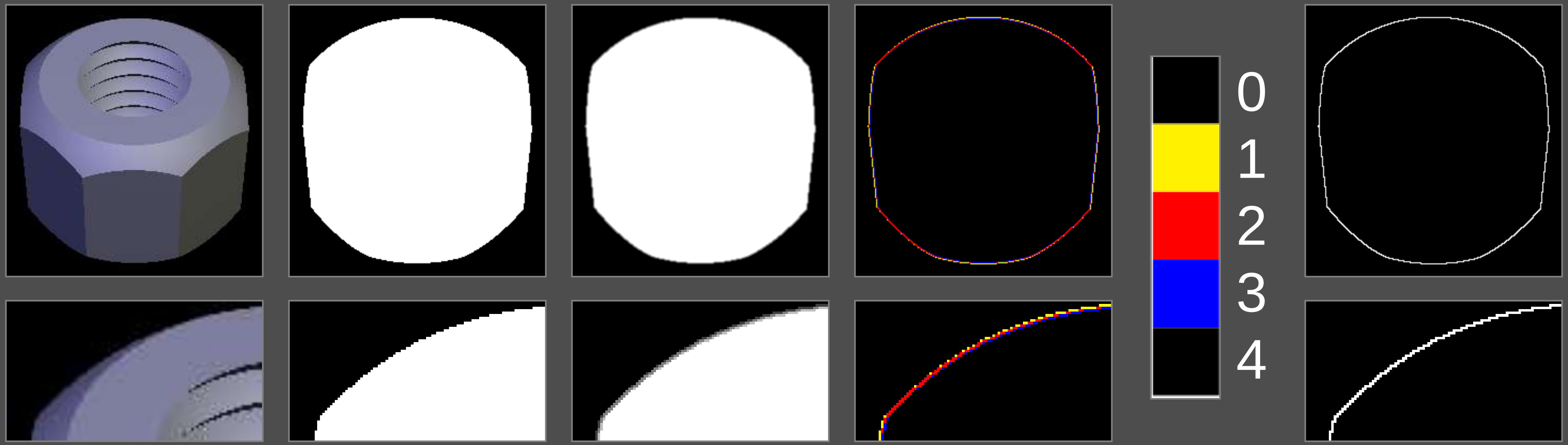
1. manipulation of the model data structures in object-space, e.g., through
   - translation of triangles or objects
   - scaling of triangles or objects
   - smart sorting of primitives that are to be rendered
   - manipulation or creation of special textures
2. traditional rendering (e.g., using OpenGL)
   $\rightarrow$ automatic generation of silhouettes

$\Rightarrow$ silhouettes are generated in image-space

Contour Detection (Rustagi, 1989)

- stencil buffer method, 4 rendering passes instead of the regular one
  - each translated by one pixel in x & y: (x−1, y), (x+1, y), (x, y−1), (x, y+1)
  - in each pass increment stencil buffer by one where object was rendered
  - where stencil buffer contains values of 2 or 3 fill with silhouette color
    ⇒ only contour of the object
- also possible with one stencil rendering pass and image filter

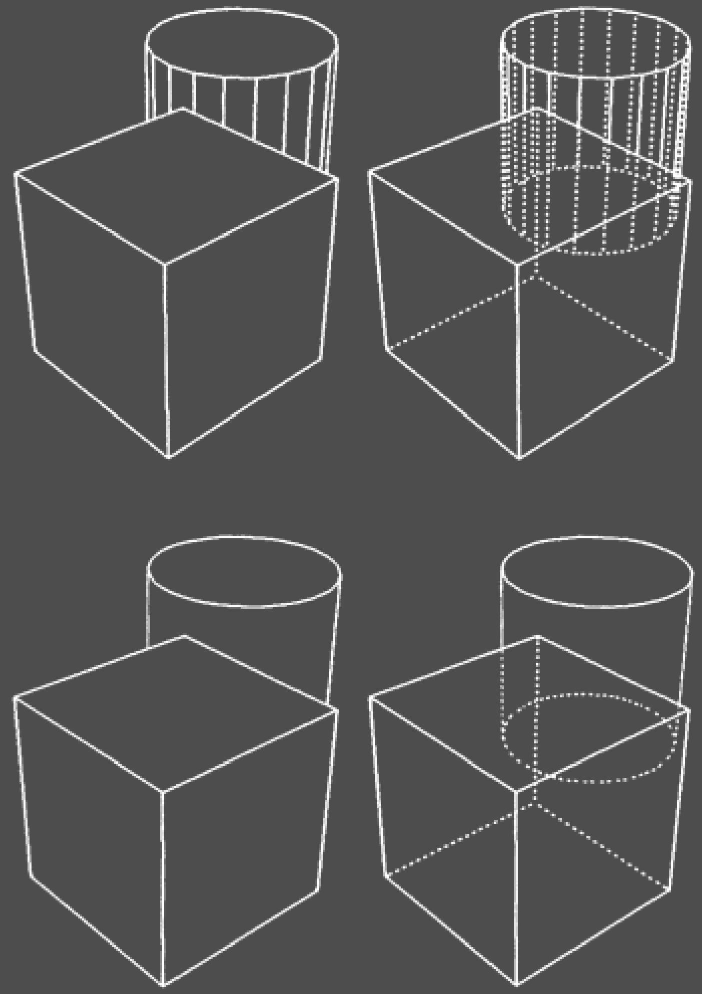
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



Wireframe Technique (Rossignac & Emmerik, 1992)

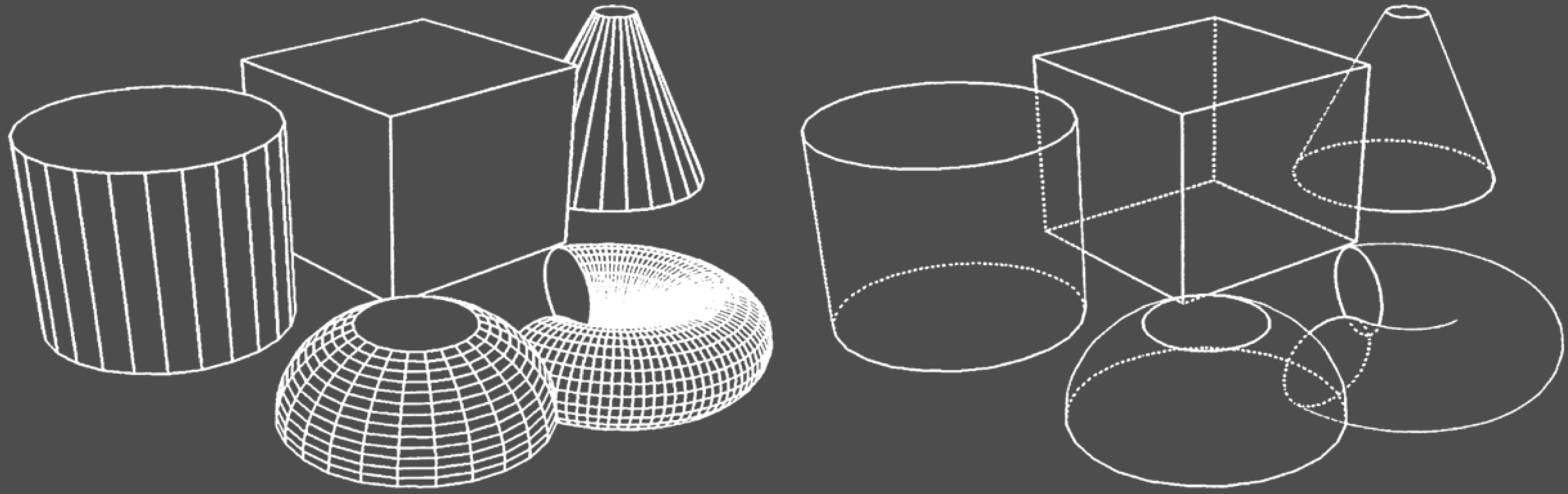
- produces wireframe/silhouettes with hidden lines removed or dashed
- to render a regular silhouette:
  1. only render regular z-buffer of the object
  2. translate the object away from viewer by small offset
  3. render object in wireframe mode with thick lines
  4. translate the object toward the viewer by 2× the offset
  5. render of feature lines only with regular line thickness

  ⇒ silhouette rendering of objects with feature lines
- dashed hidden lines by initial line rendering pass for individual (sub-)objects one-by-one with all wireframe lines drawn in dashed style



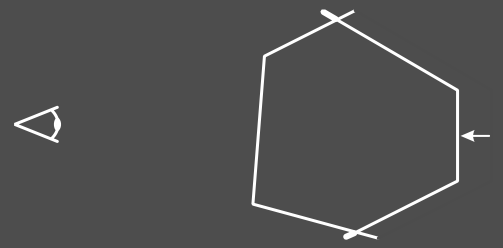
Rossignac & Emmerik (1992)

Wireframe Technique: Example

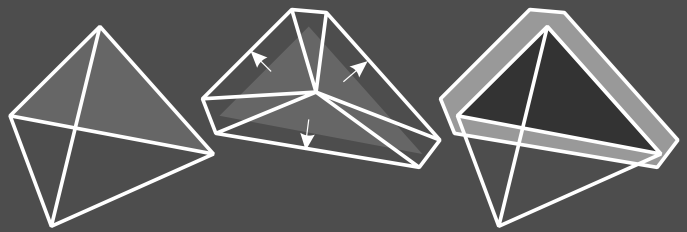


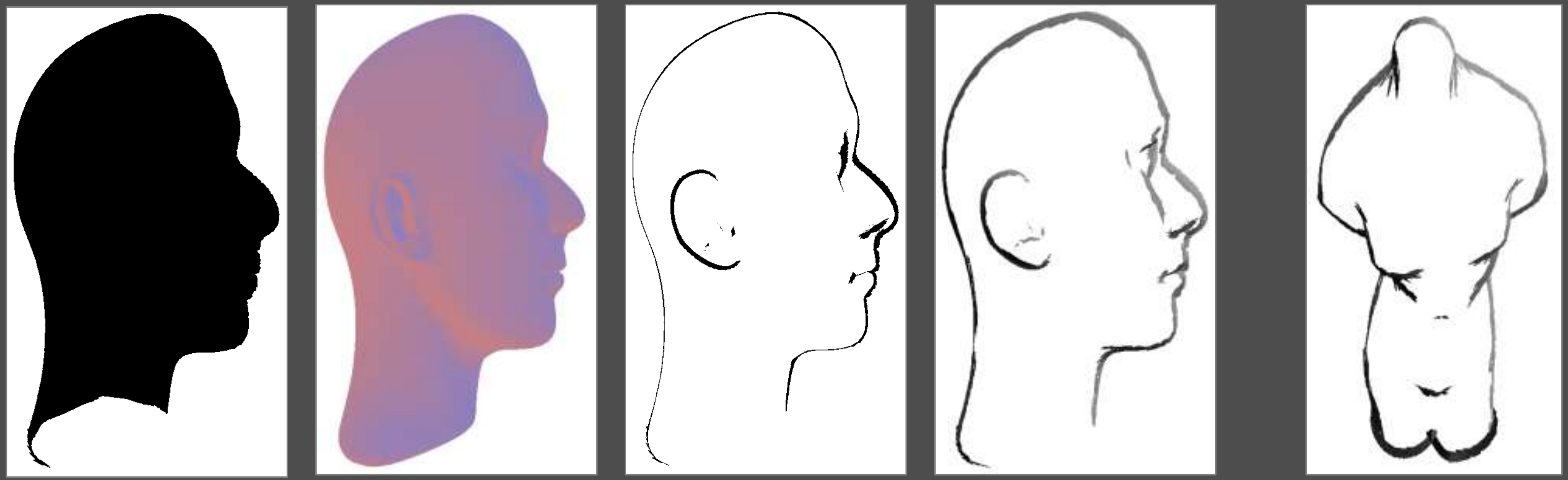
Rossignac & Emmerik (1992)

Back-Face Technique: Raskar & Cohen (1999)



- translation of back-facing polygons by a certain distance toward the viewer and render them in silhouette color
- enlarging of back-facing polygons by a factor and render them in silhouette color

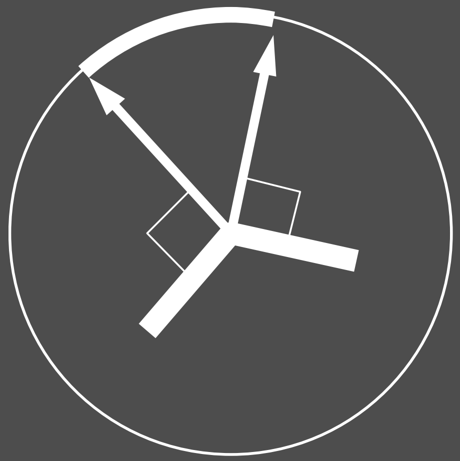


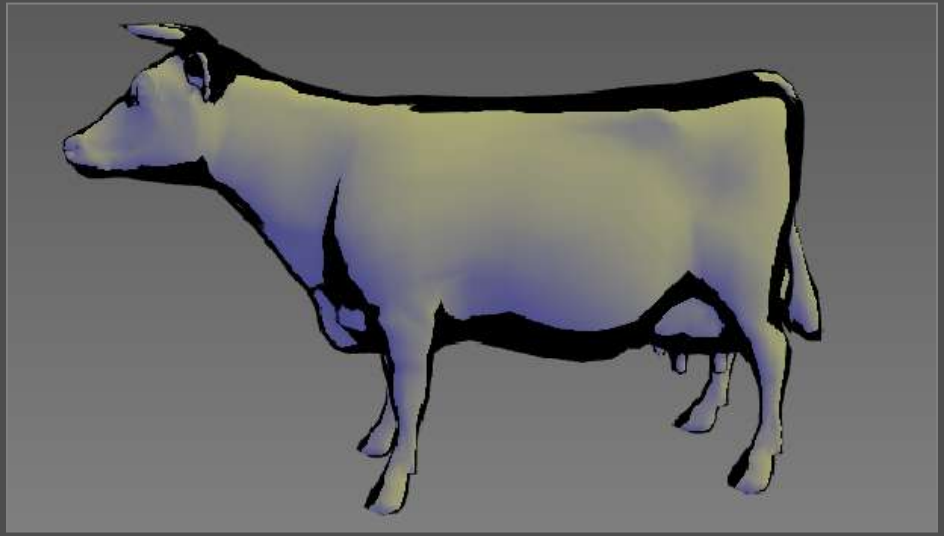


Raskar & Cohen (1999)

Texture Mapping Technique: Gooch et al. (1999)

1. polygon manipulations similar to Raskar & Cohen (1999)
2. application of special texture mapping
   - environment mapping with spherical mapping function
   - peripheral part of texture black, remainder transparent ⇒ silhouettes appear
   - stylized appearance, but not stylizable beyond thickness control
   - no silhouette at hard edges where no normal interpolation occurs

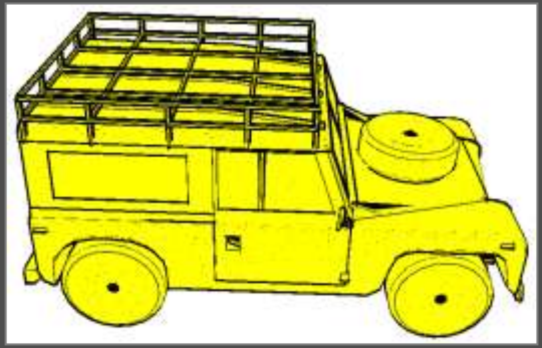


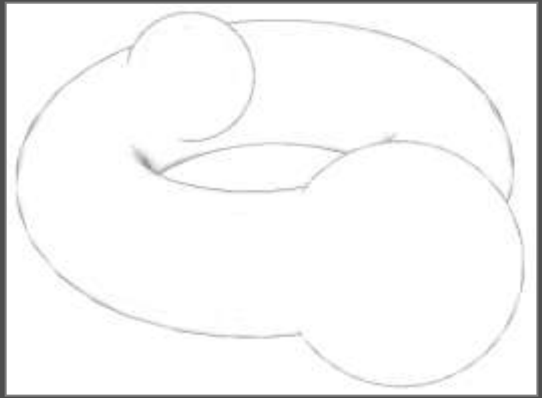


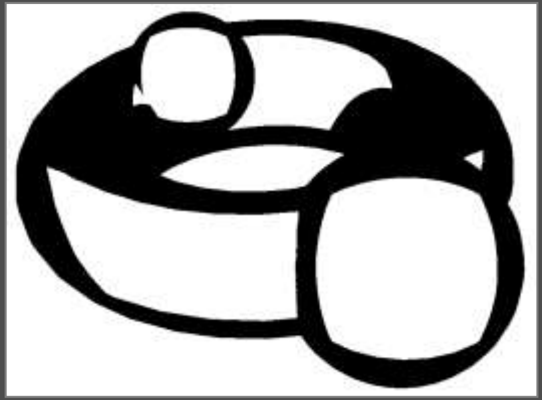
Gooch et al. (1999)

Hybrid Techniques with Early Hardware Acceleration

- Raskar (2001): one-pass on-chip processing
  - silhouettes: back-facing polygons in black & larger
  - ridges: to each edge of front-facing polygon add small polygon with angle greater than threshold
  - ruts similar; intersections through double z-test
  - → polygon soups usable and animation easily possible
- Everitt (2002):
  - one-pass technique using register combiners and hardware normal interpolation
  - using that dot product of normal with view vector is close to zero at silhouette
  - yields gray-scale silhouette
  - fast but not very flexible, no feature edges detected

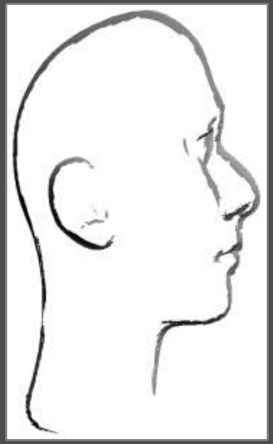


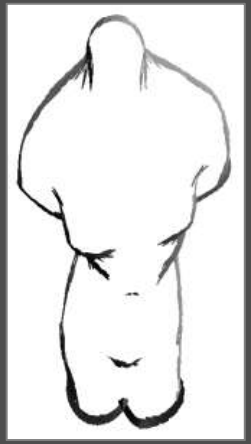


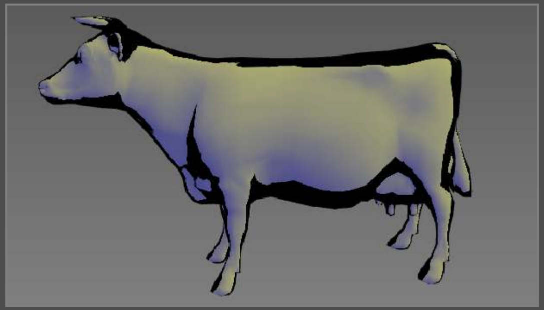


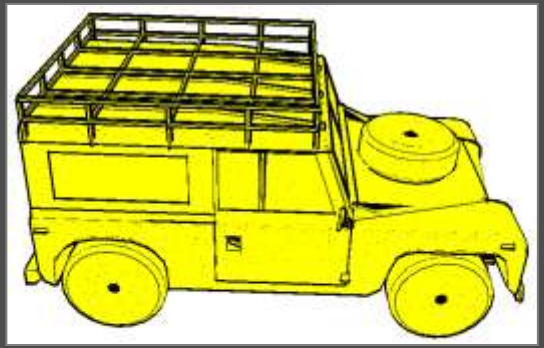
Hybrid Silhouettes: Summary

- positive:
  - existing hardware & rendering systems usable
  - relatively fast computation
    - regular rendering (e.g., OpenGL), partially several rendering passes necessary
    - run-time complexity depends on type of manipulation of primitives
    - hardware acceleration possible
  - more influence on result than with image-space techniques
    - by choice of specific method
    - by parameterization
- negative:
  - result only as pixel matrix ⇒ only pixel accuracy
  - further processing and application of line styles difficult



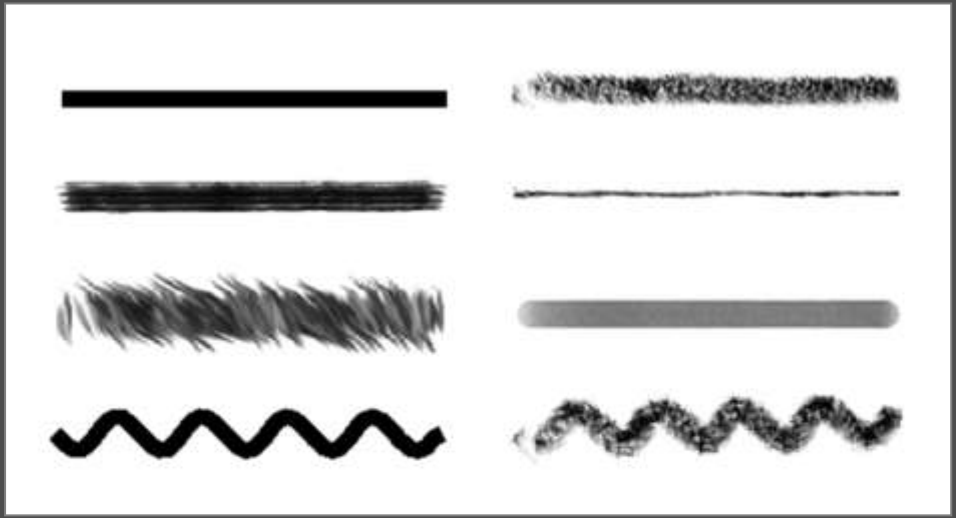






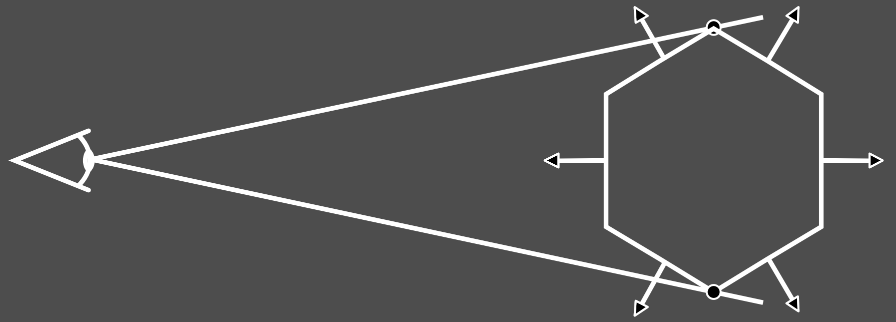
Object-Space Silhouettes

- work entirely in object-space ⇒ no rendering required for detection
- usually require elaborate geometric data structure
  - local connectivity information necessary with fast access → typically O(1)
- major advantage: silhouette strokes as analytic data (vector image)
  - ⇒ silhouettes can be processed further
  - ⇒ silhouettes can be stylized
- line styles: modifications of a stroke
  - path alterations
  - line attribute changes (e.g., width, color, transparency)
  - line texture
  - → simulation of drawing and painting tools possible



Trivial Method

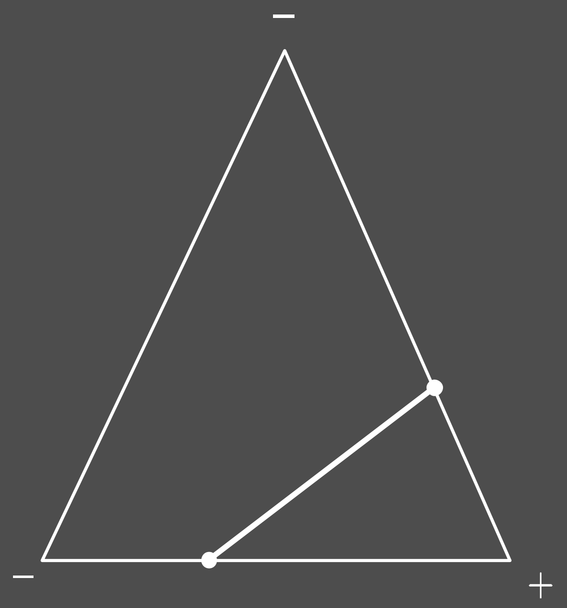
- driven by definition
- determination of visibility of all polygons
- detection of all edges that share a front-facing and a back-facing polygon → brute-force approach

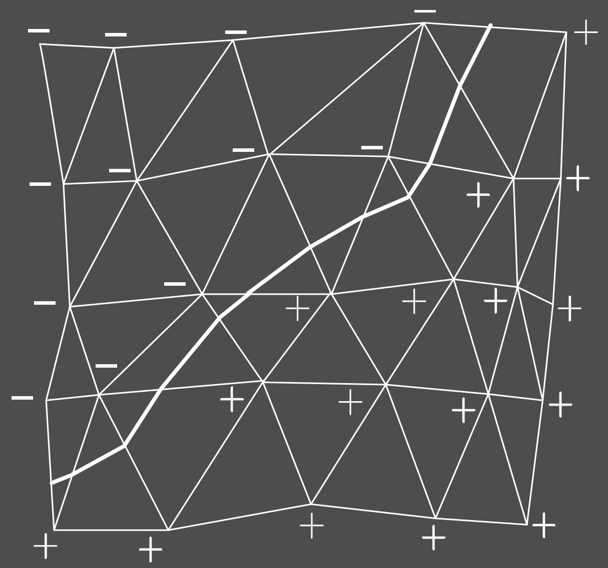


```
EdgeList silhouette_extraction(Mesh m) {
  EdgeList silhouetteEdges;
  determineFaceVisibility(m);
  for (Edge e = m.edges.f(); e != m.edges.l(); e++){
    if ((e.face1.visible && !(e.face2.visible)) ||
      (e.face2.visible && !(e.face1.visible)))
     silhouetteEdges.add(e);
  return silhouetteEdges;
}
```

Sub-Polygon Silhouettes

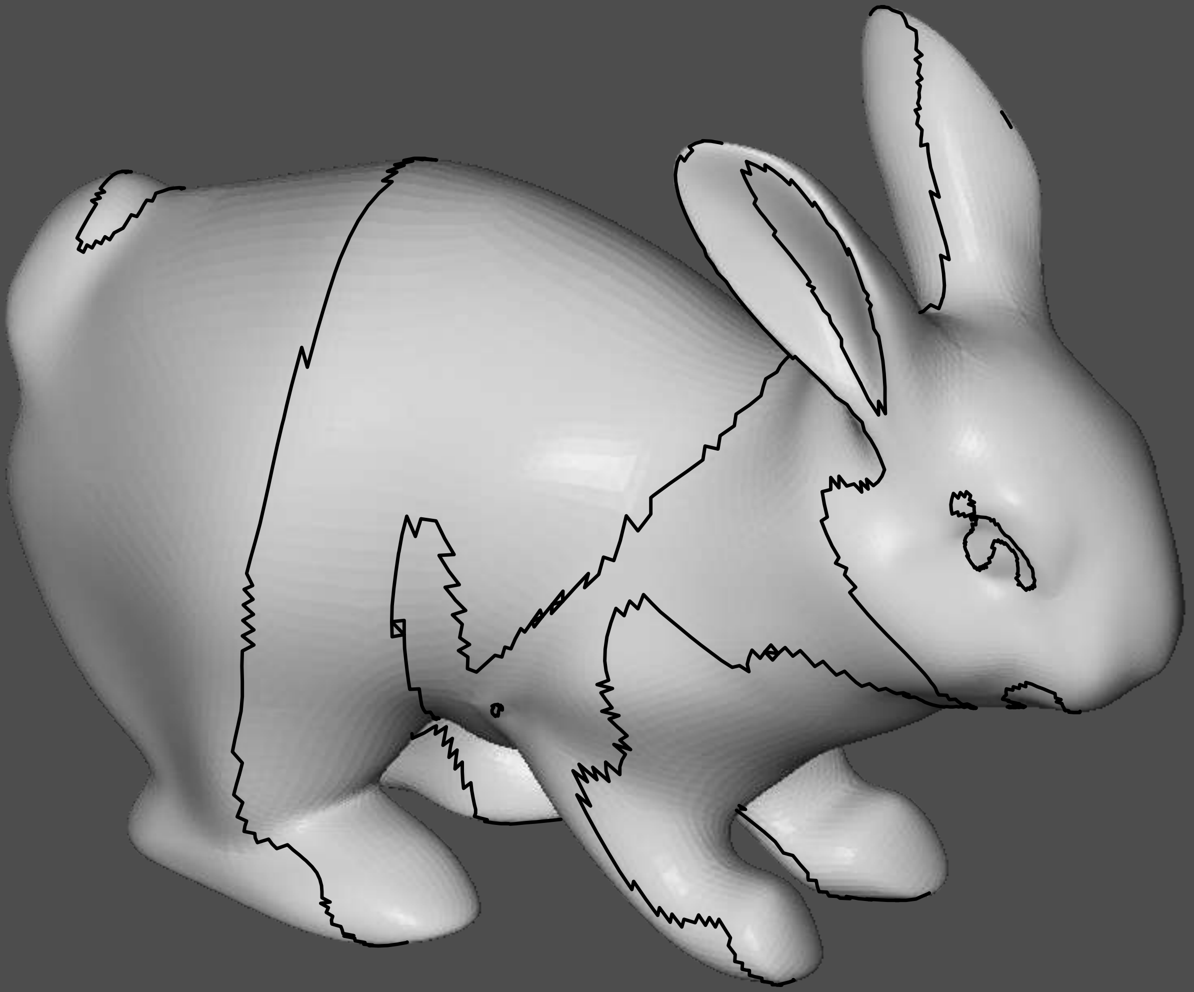
- Hertzmann & Zorin (2000)
  - interpolation of silhouettes over triangles
  - higher accuracy of silhouette strokes
  - less triangulation artifacts (zig-zags, sharp bends, short edges, etc.)
- computation of the dot product between vertex normal and viewing direction for each vertex
- linear interpolation over edges to locate zero-crossings
- connecting these points yields smoother silhouette stroke

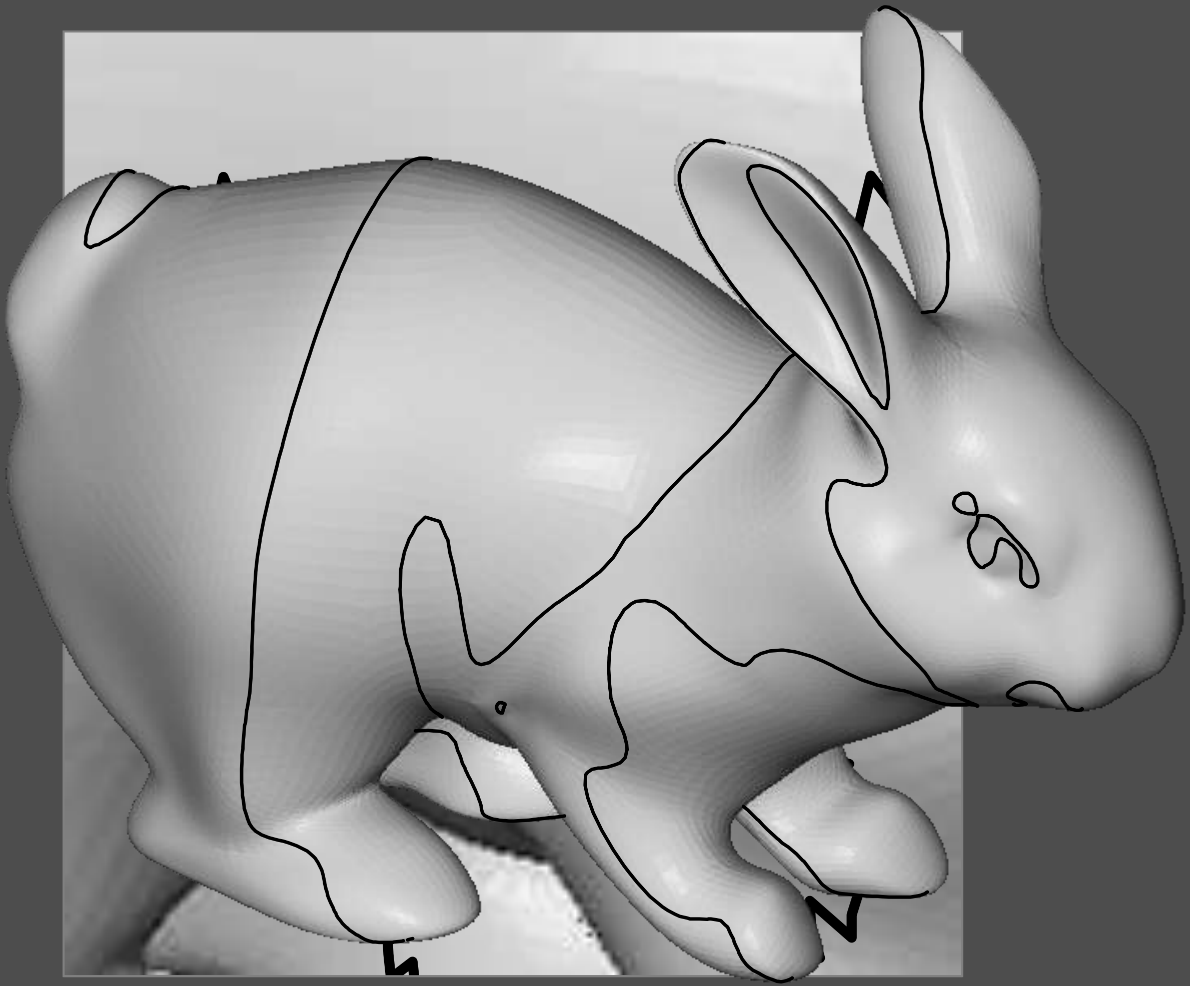




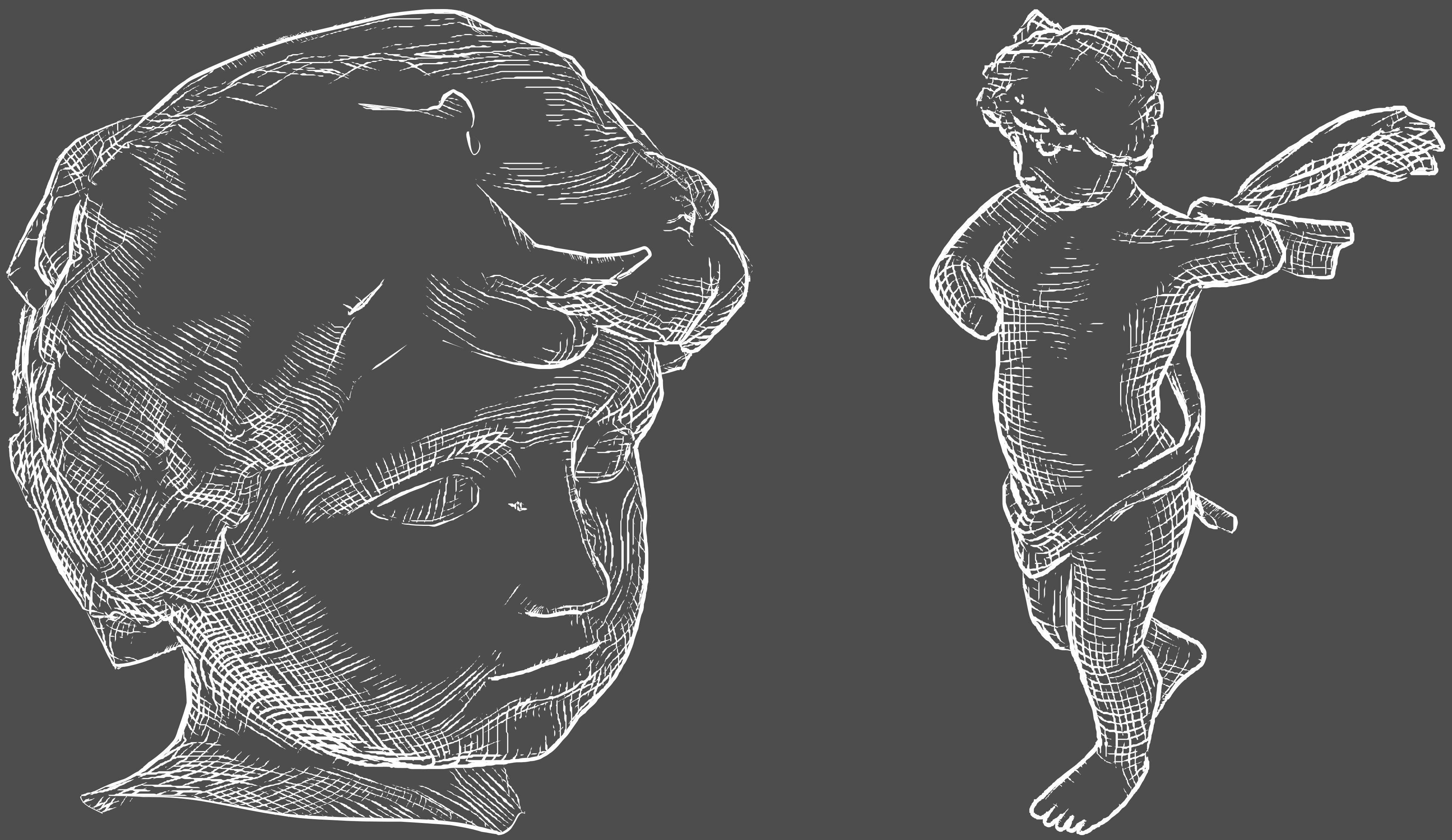
Hertzmann & Zorin (2000)

Comparison Trivial & Sub-Polygon Silhouettes





Sub-Polygon Silhouettes: Example



Hertzmann & Zorin (2000)

Silhouette Computation Pre-Processing

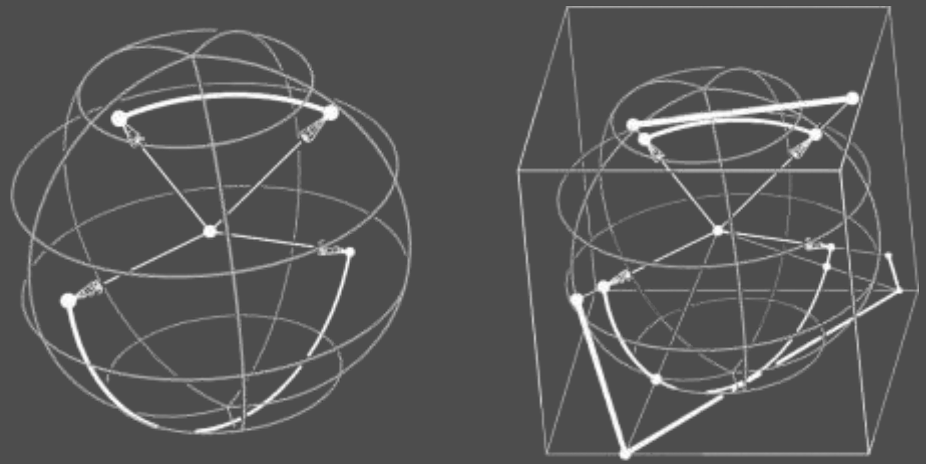
- goal: faster detection of silhouette edges
- reduction of search domain by use of dual spaces
  - translation of the problem into equivalent problem into a dual space
  - dual problem is easier to solve
  - solution to dual problem leads to solution in primal space
- our problem: run-time complexity of $O(N_{faces} + N_{edges})$

$\rightarrow$ silhouette detection typically implemented in software

$\Rightarrow$ O(N) still too slow for real-time rendering

Gaussian Sphere Pre-Processing: Benichou & Elber (´99)

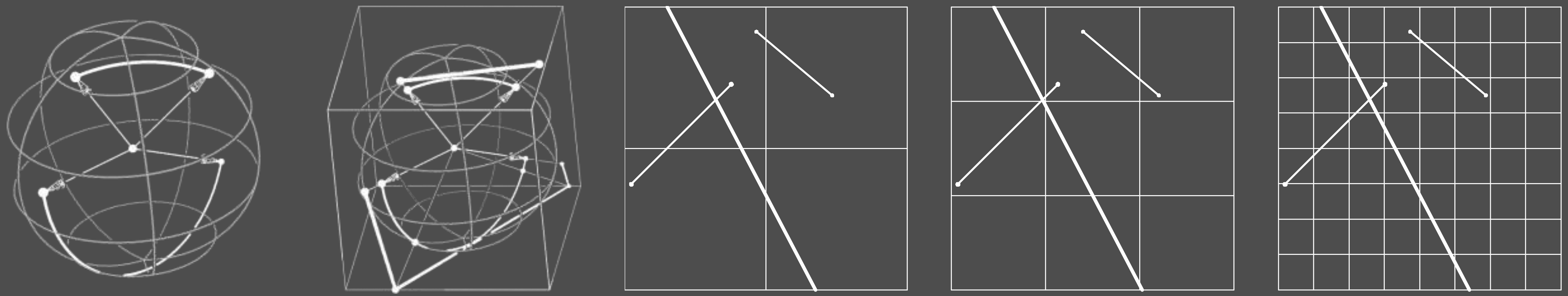
- project face normals onto Gaussian sphere (visibility space)
- for each mesh edge: connect 2 corresponding projections by unique segment of great circle → represents the visibility of the edge
- view direction equivalent to its perpendicular plane
- visibility culling: cut Gaussian sphere with this plane → orth. proj. only



Benichou & Elber (1999)

Gaussian Sphere Pre-Processing: Benichou & Elber (´99)

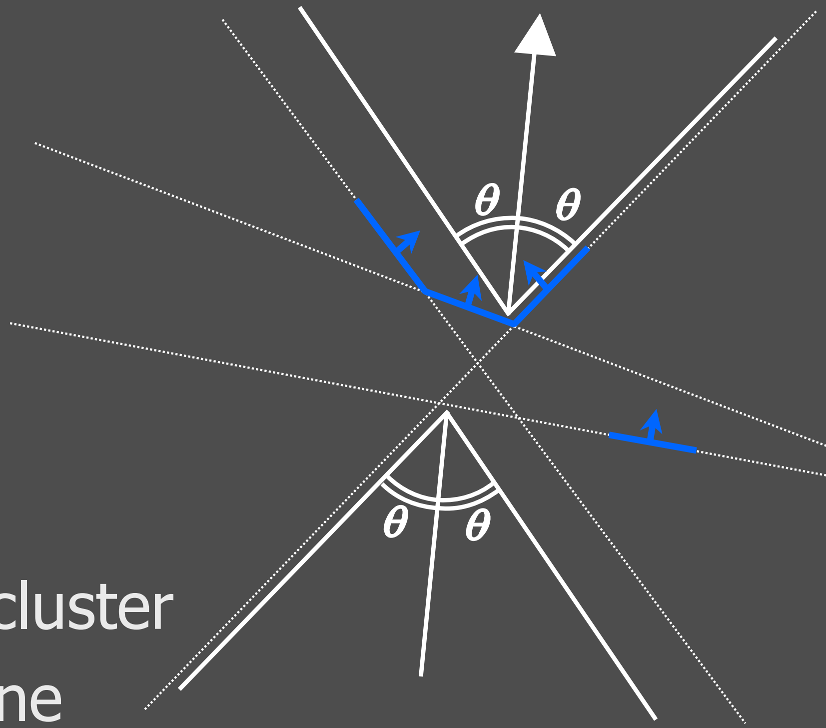
- each circle segment intersected by view plane corresponds to change from visible to invisible: the silhouette
  → reduction of time complexity to $O(N_{edges})$ at runtime
- optimization 1: project onto a surrounding cube → easier intersection
- optimization 2: grid data structure for each cube side
  → reduction of **expected** run-time complexity to $O(N_{silhouette\ edges})$



Benichou & Elber (1999)

Pre-Processing: Other (Using Dual Spaces)

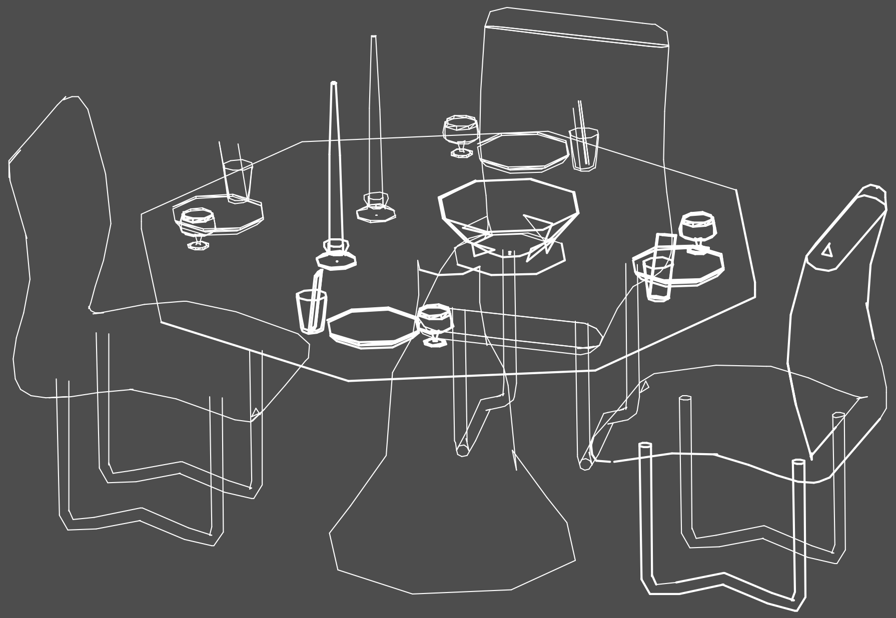
- Gooch et al. (1999): similar method using Gaussian sphere
- Hertzmann & Zorin (2000): 4D unit hypercube as dual space
  - triangle-plane intersection in eight 3D unit cubes (the hypercube's sides)
  - also applicable for perspective projection
- Pop et al. (2001): 3D dual space
  - points → planes and vice versa; viewpoint's dual plane cut with edges' duals
  - only changes between frames tracked; perspective projection possible
- Sander et al. (2000): anchored cones
  - hierarchical search tree; each node stores face cluster
  - face normals per cluster are within anchored cone



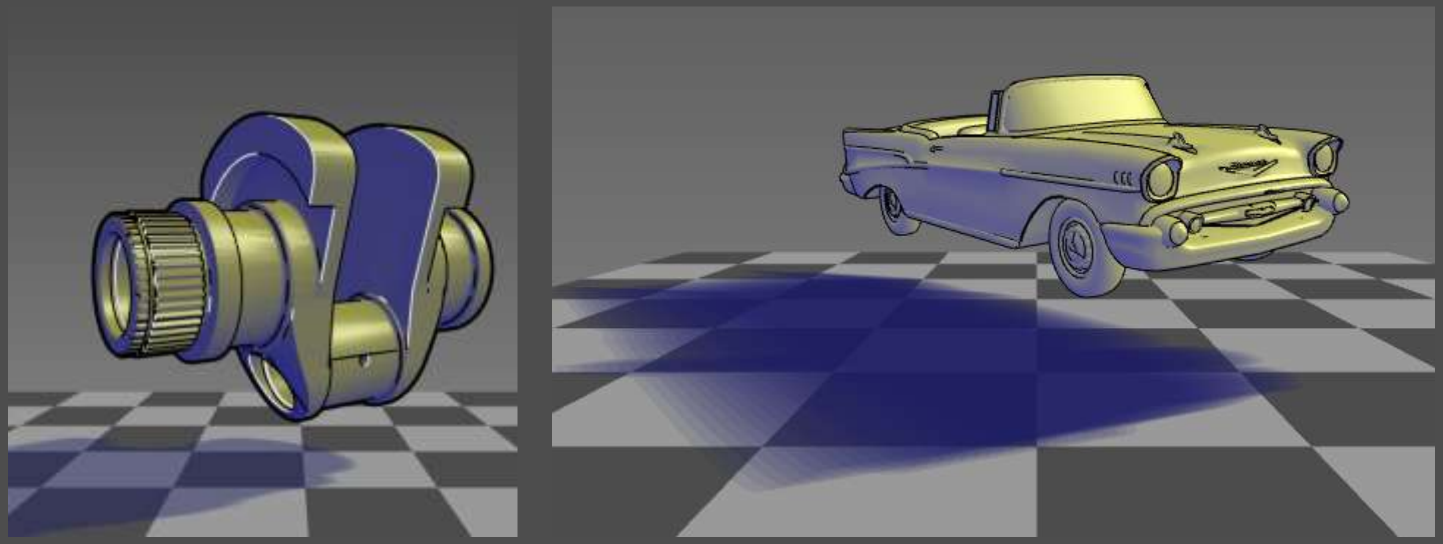
Sander et al. (2000)

Dual-space Pre-processing: Properties

- correct result, i.e., same as with brute-force method
- fast "flight through the scene" possible
- no animation of the scene itself due to elaborate data structures



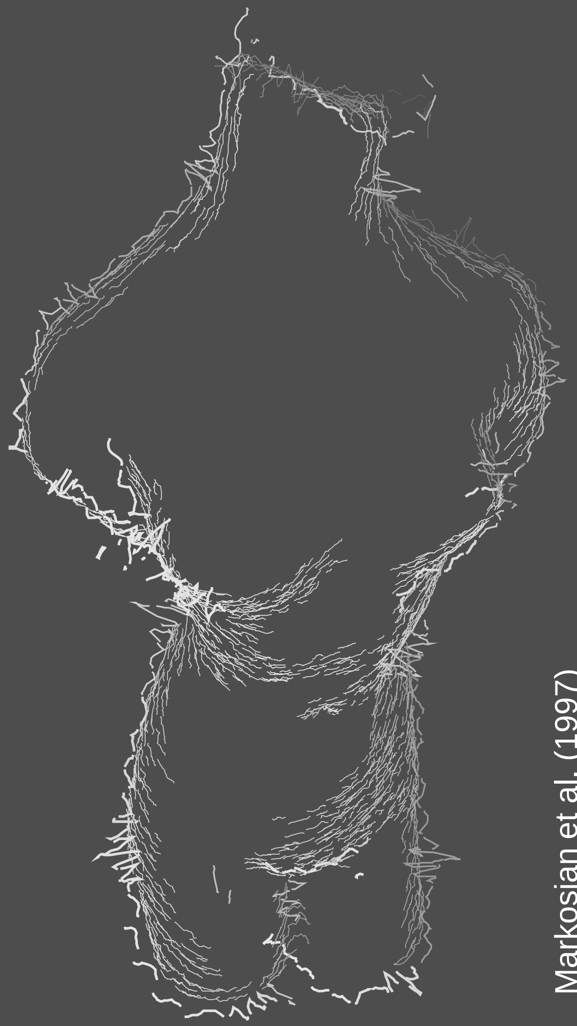
Benichou & Elber (1999)



Gooch et al. (1999)

Speed-Up by Stochastic Method: Markosian et al. (1997)

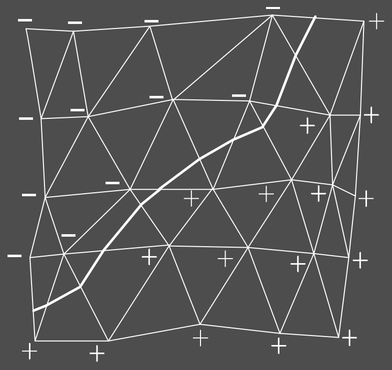
- test randomly selected subset of edges (≈10%) for silhouette property
- continue to locally track detected silhouette edges
  → taking advantage of spatial coherence
- search in the neighborhood of silhouette edges of the last frame
  → taking advantage of temporal coherence
  → fast detection of silhouette edges
  → not all silhouette edges are necessarily found
  ⇒ result not guaranteed to be exact
- in most cases artifacts from missing edges not noticeable

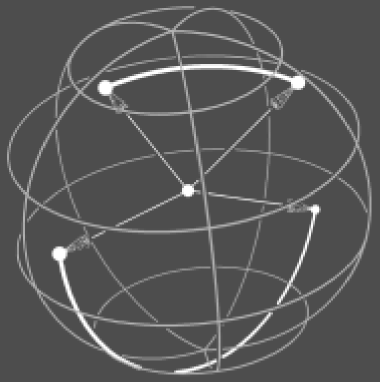


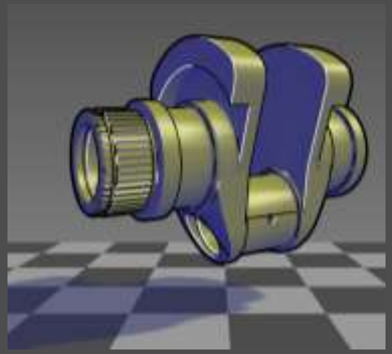
Markosian et al. (1997)

Object-Space: Summary

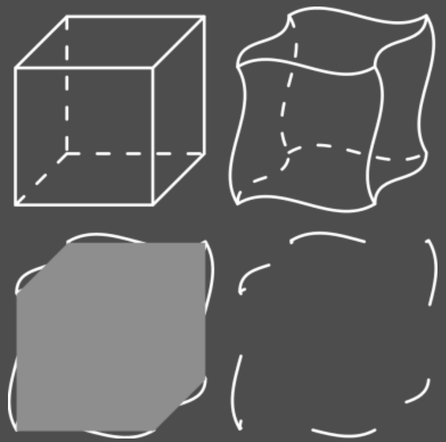
- positive
  - better accuracy or exact result, even sub-polygon accuracy
  - acceleration through pre-processing and stochastic methods
  - silhouettes generated as vector data
  - ⇒ silhouette rendering independently from silhouette detection
  - ⇒ easy stylization and further processing possible
- negative
  - often higher rendering times
  - more difficult to implement
  - acceleration may limit animation and/or accuracy
  - hardware acceleration hardly possible
  - additional hidden line removal step necessary

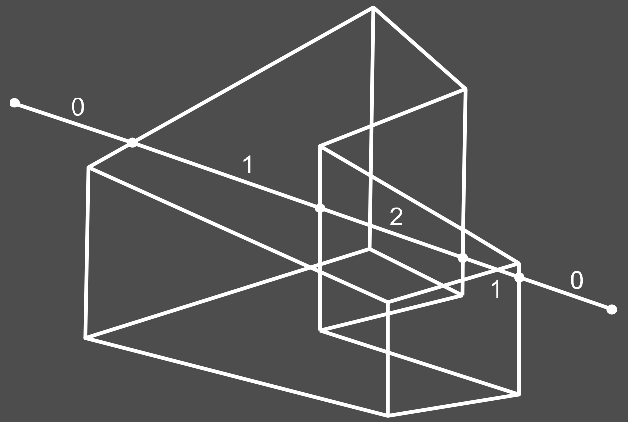


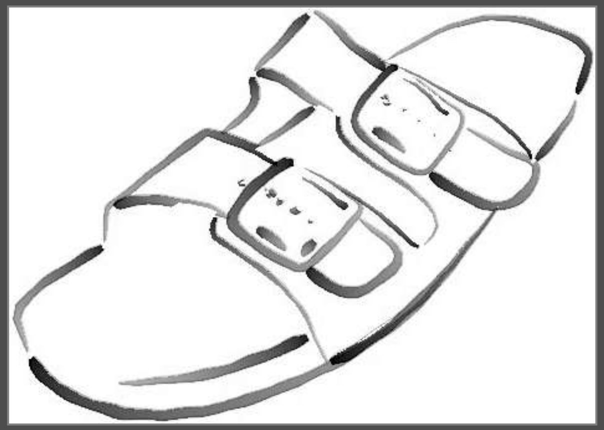




Hidden Line Removal for Object-Space Silhouettes



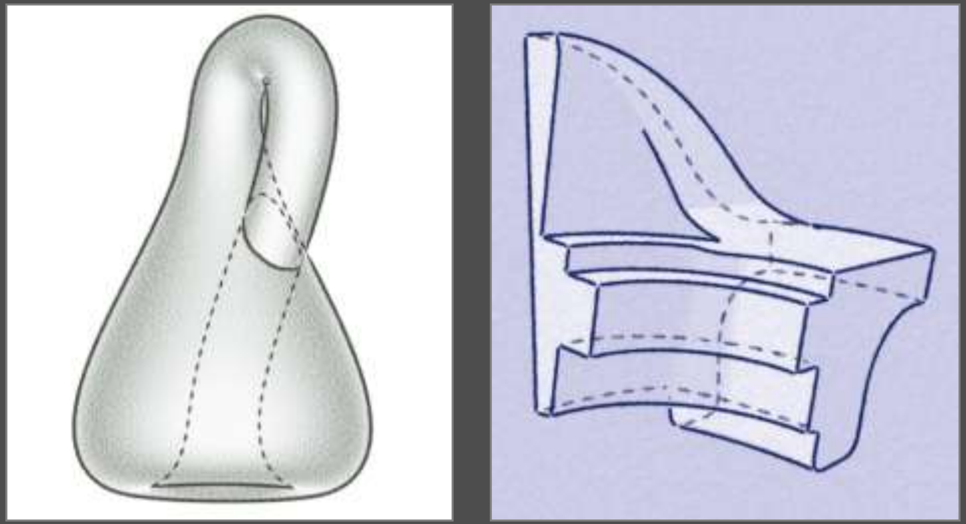




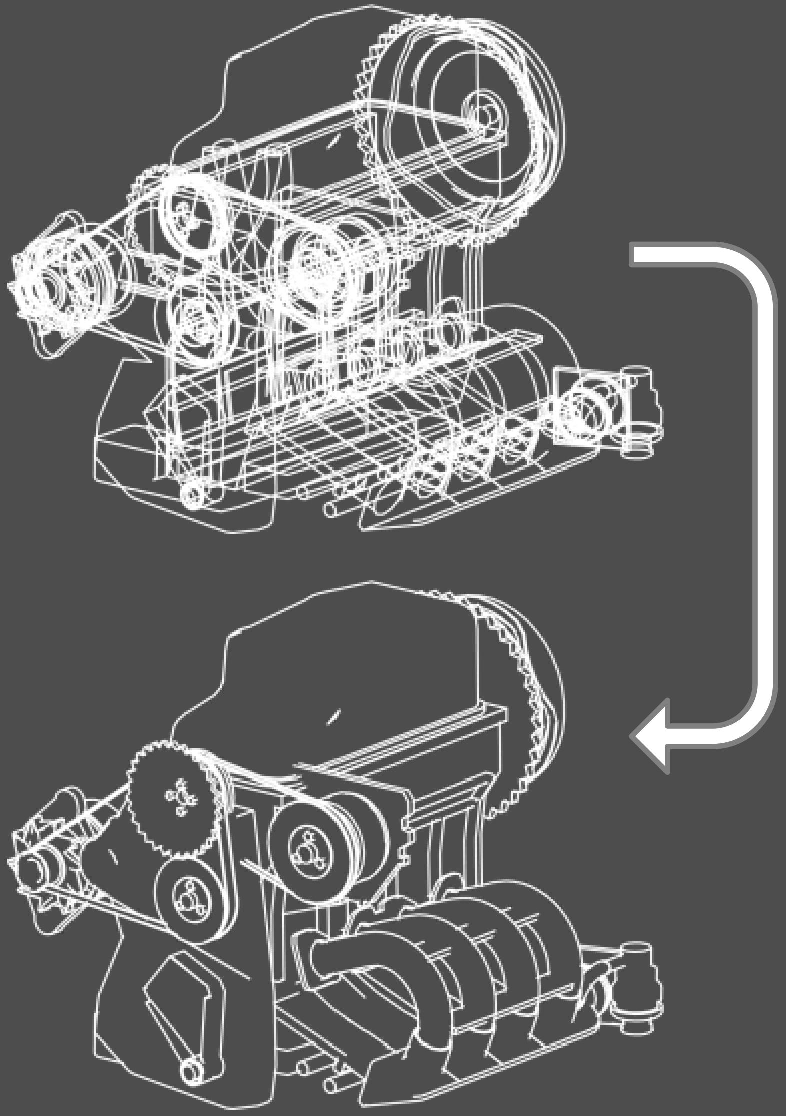
HLR for Silhouettes (and Feature Strokes)

- up to now: all possible silhouette (and feature) edges detected
- in most cases edges partially occluded
- occluded subset has to be omitted or rendered in special line style

⇒ goal: determine the visible subset



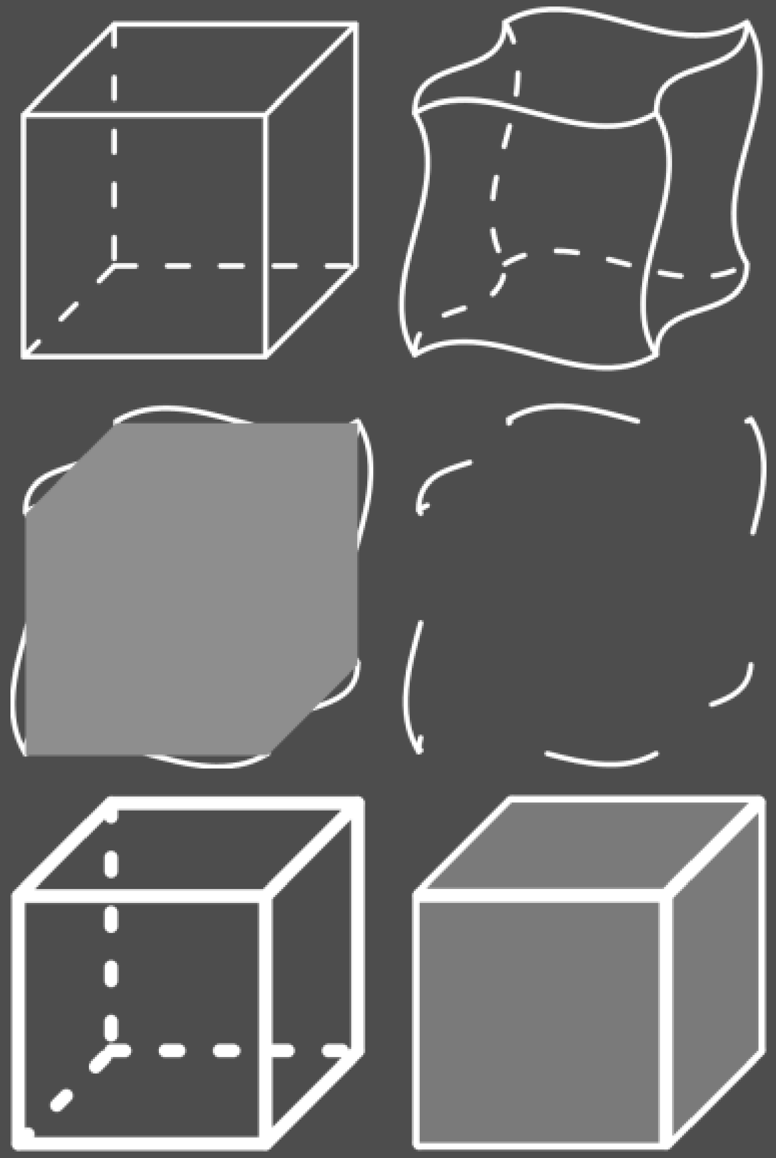
Kalnins et al. (2003)



Naïve Method: Image-Space HLR

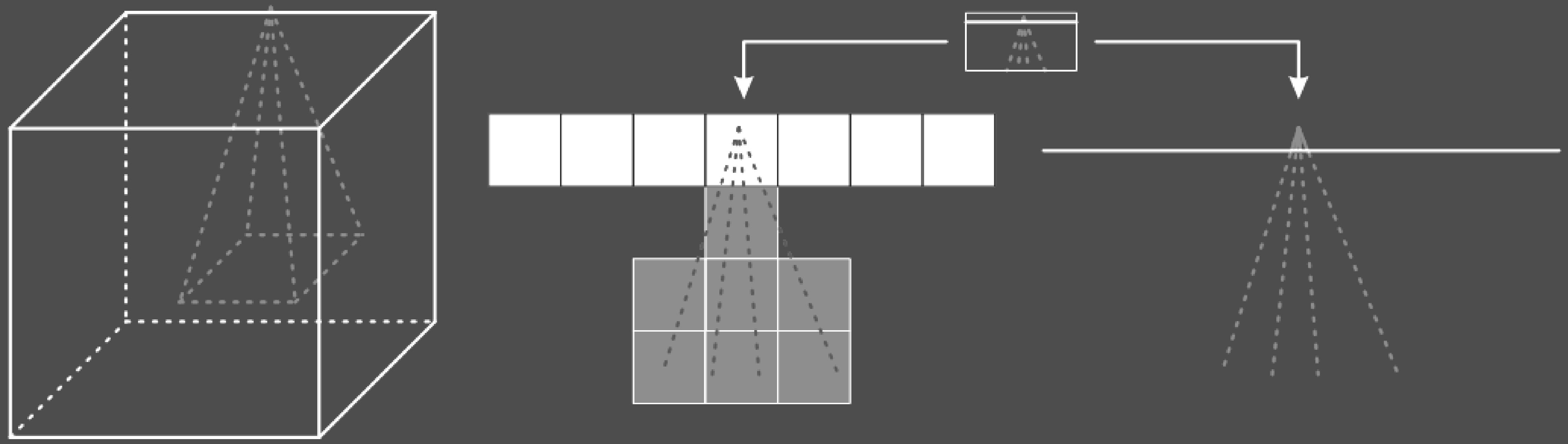
- first: render objects normally
  or only the z-buffer
- then: render (potentially stylized) silhouette
  strokes using z-buffering
- problems:
  - numerical problems with z-buffer rendering (“z-fighting” etc.)
  - occlusion of style elements
  - different line widths of silhouettes and feature strokes

⇒ better: first visibility culling (HLR),
then render without z-buffer



Object-Space HLR

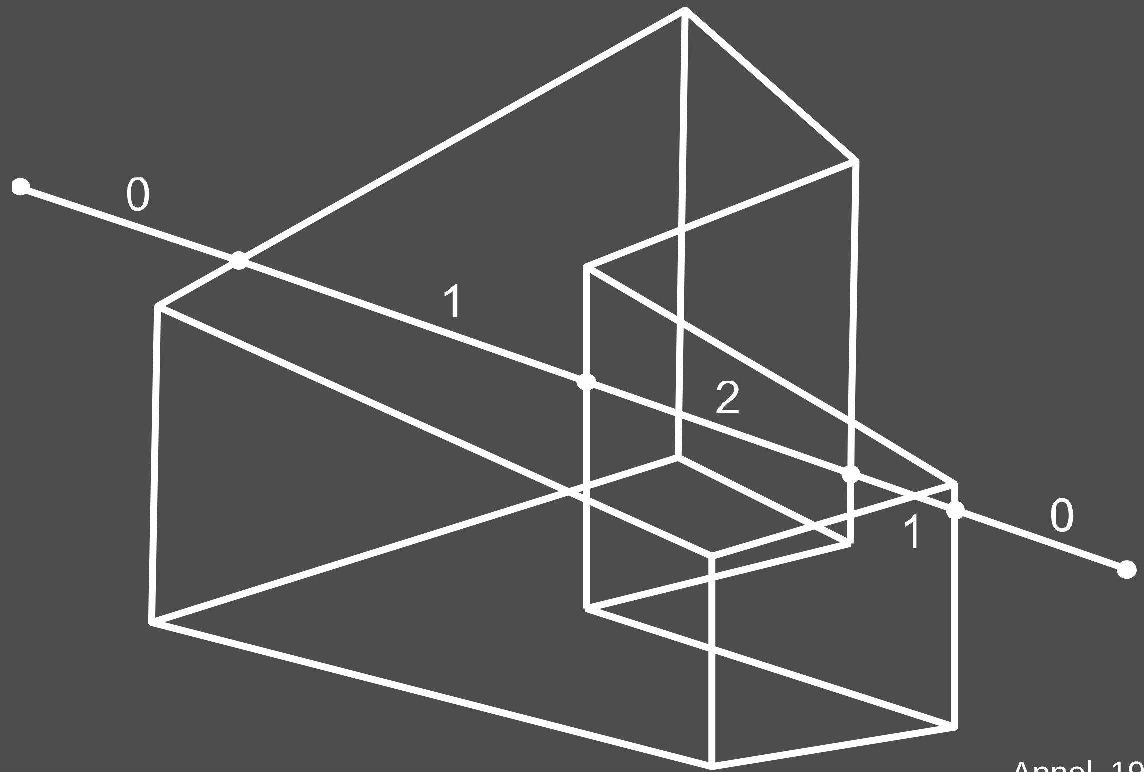
- a variety of algorithms for object-space hidden **surface** removal
- many work for silhouette HLR; e.g., Sechrest & Greenberg (1981)
- positive: algorithms are analytic and therefore yield exact results



- negative: high computational complexity ⇒ typically very slow

Object-Space HLR Using Appel's Algorithm

- visibility culling of lines using quantitative invisibility QI (Appel, 1967)
- QI changes only at silhouette edges
- visible: line segments with QI = 0; invisible: line segments with QI ≥ 1

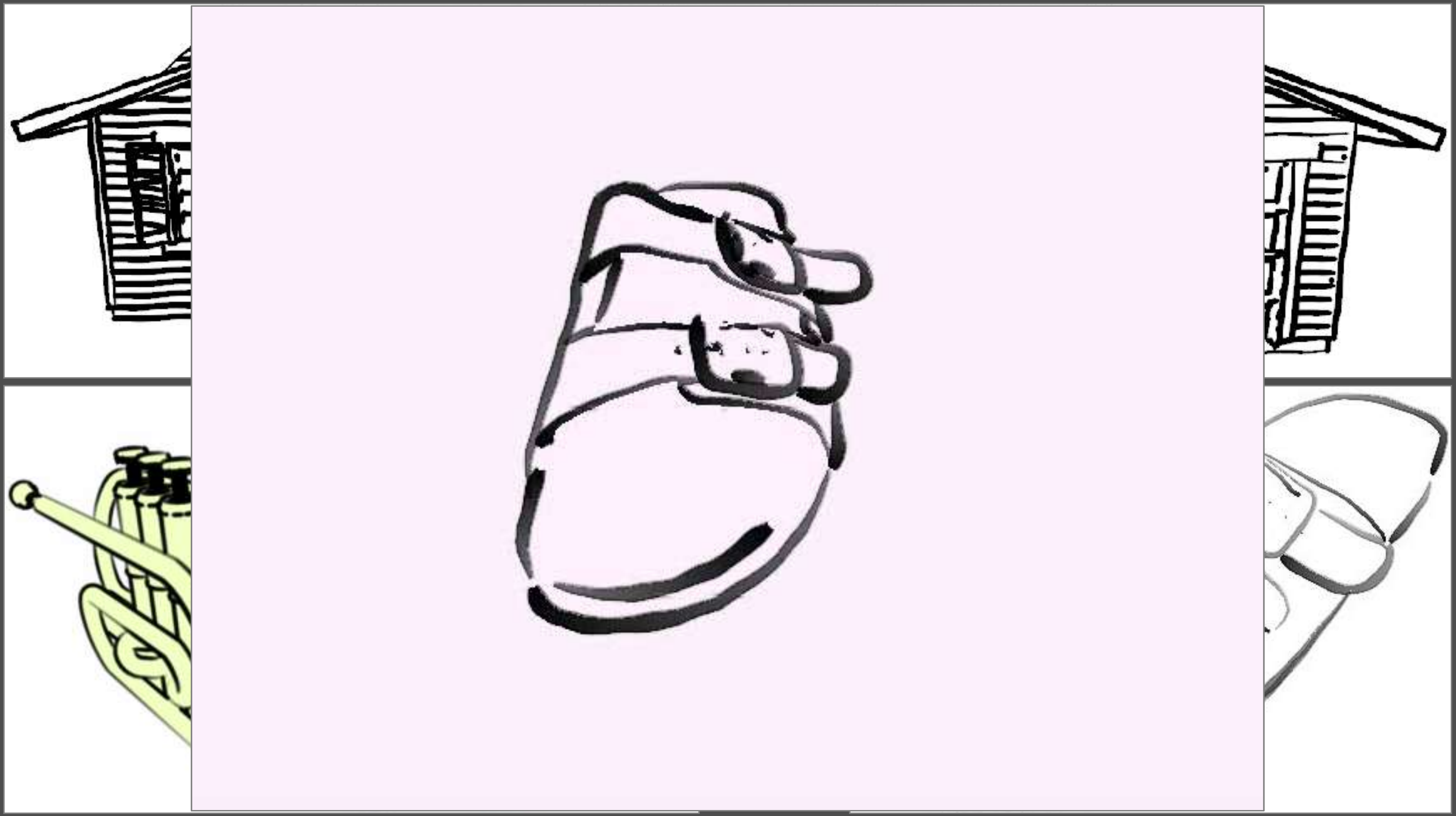


Appel, 1967

Hybrid HLR with ID Buffer (Northrup & Markosian, 2000)

- two rendering passes
  - regular rendering (frame buffer)
  - ID buffer (every triangle and every edge is assigned a unique color)
- scan conversion of each silhouette edge
- test if edge contributes to ID buffer ⇒ visible silhouette segments

Hybrid HLR with ID Buffer: Examples



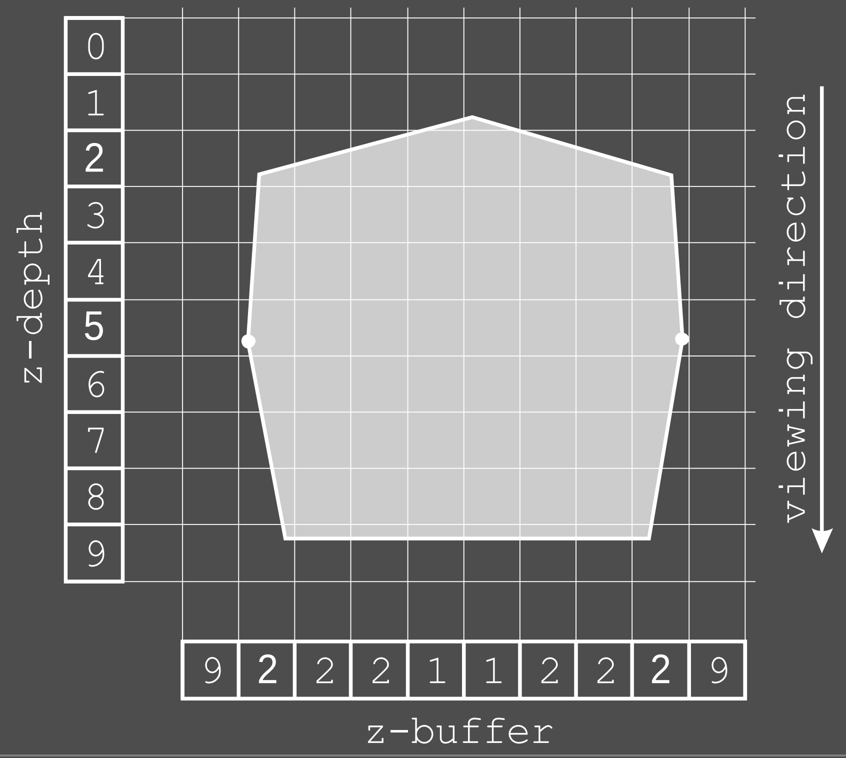
Northrup & Markosian, (2000)

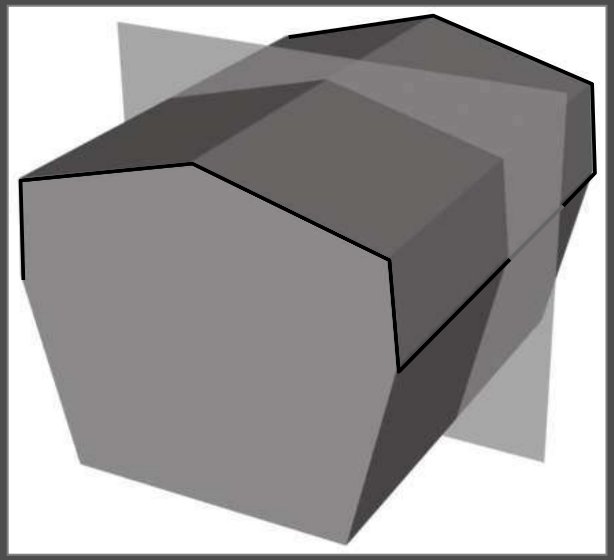
Hybrid Hidden Line Removal

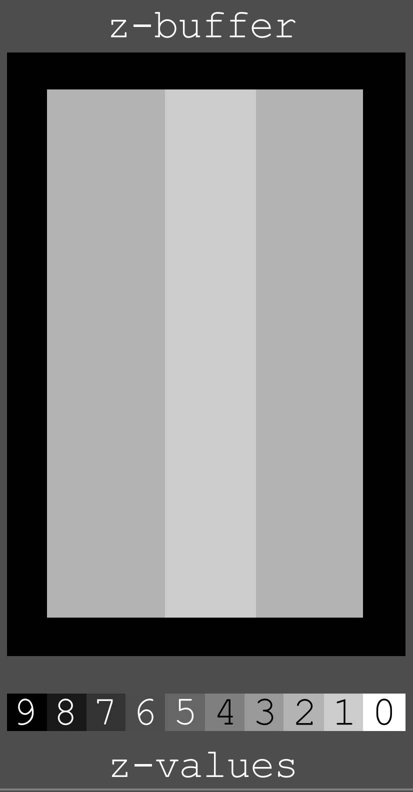
- problem with ID buffer method:
  extra buffer needs to be generated → two rendering passes
- idea: using occlusion information available in z-buffer
- fast because z-buffer generated in most cases anyway

Hybrid HLR with z-Buffer

- observations
  - silhouette edges always at discontinuities of z-buffer
  - numerical problem:
    z-buffer value at silhouette pixel often closer to viewer than silhouette



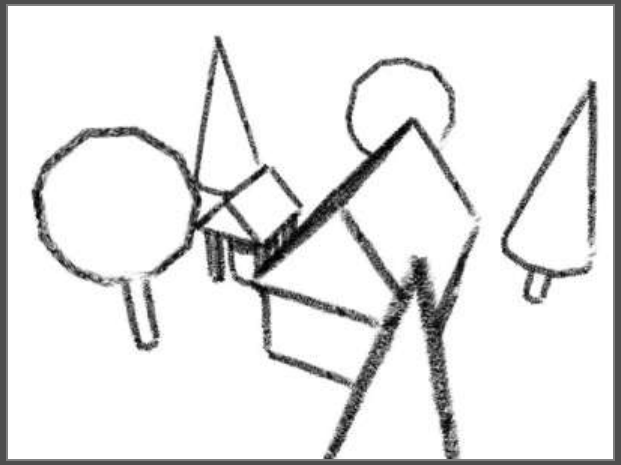


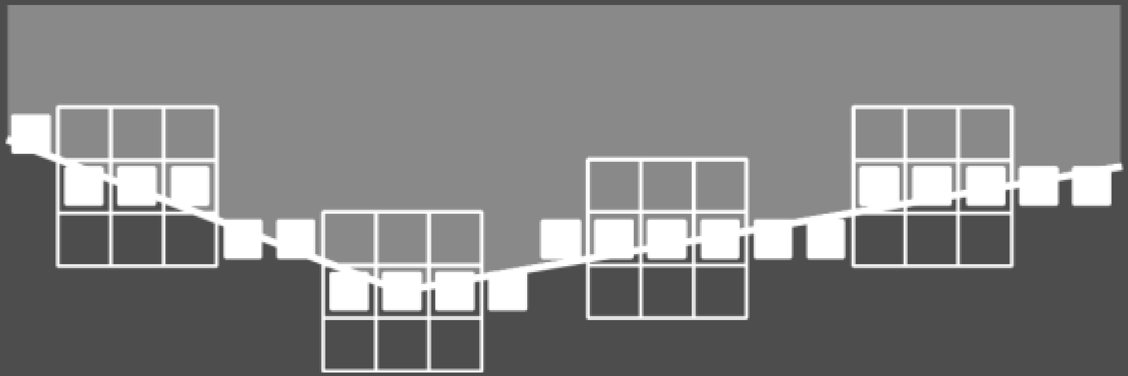


Hybrid HLR Using z-Buffer

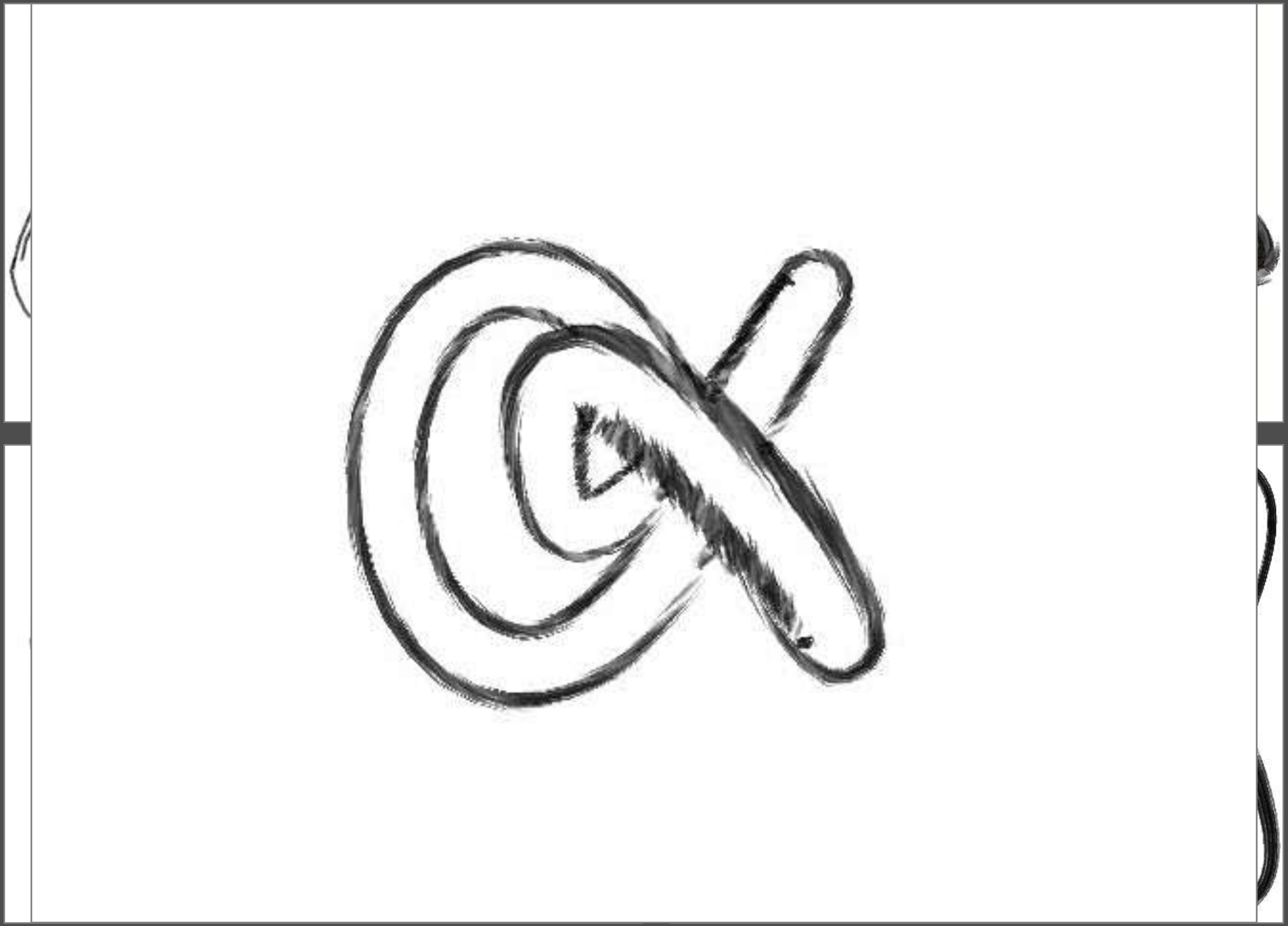
- 8-neighborhood search
  - z-buffer lookup of 8 neighboring pixels
  - silhouette visible if any pixel farther than silhouette
- enhancements
  - skipping of n pixels & binary search
  - reading back of z-buffer in chunks
  - caching of z-buffer in main memory
- properties
  - at most pixel accuracy
  - rendering of additional buffers not necessary
  - artifacts negligible (skipping of thin objects, ends of some invisible strokes)

  → trade of accuracy for speed



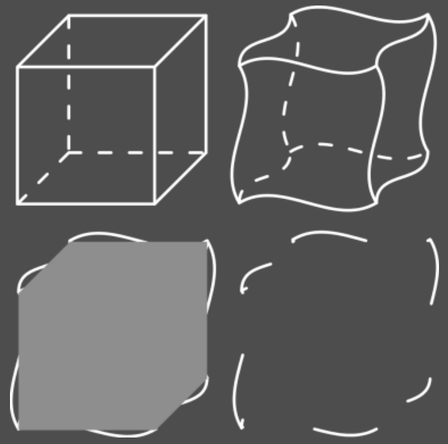


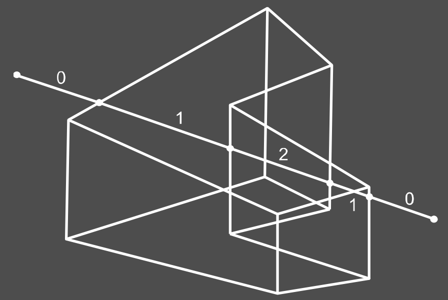
Hybrid HLR with z-Buffer: Examples

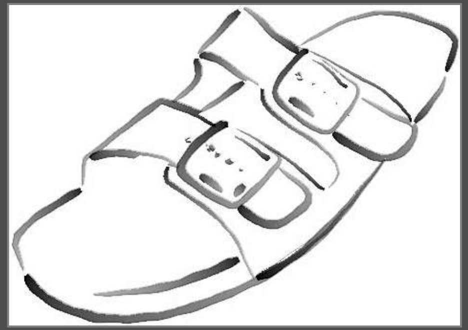


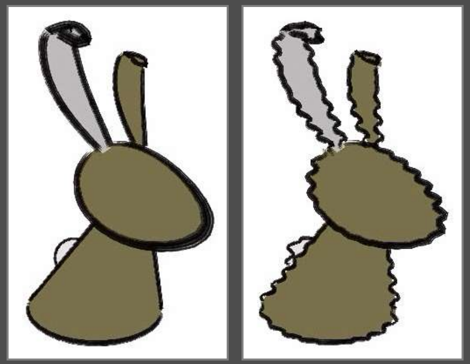
Hidden Line Removal: Summary

- image-space HLR usually not possible
  - insufficient for extracting occlusion information
  - numerical problems
  - artifacts when using stylized silhouettes
- object-space (analytic) HLR
  - exact but typically to slow
- hybrid HLR
  - occlusion information from G-buffers
  - ID buffer or z-buffer
  - only pixel accuracy → sufficient in most cases
  - interactive frame-rates possible
  - trading accuracy for speed

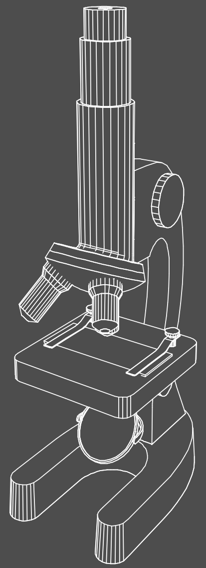


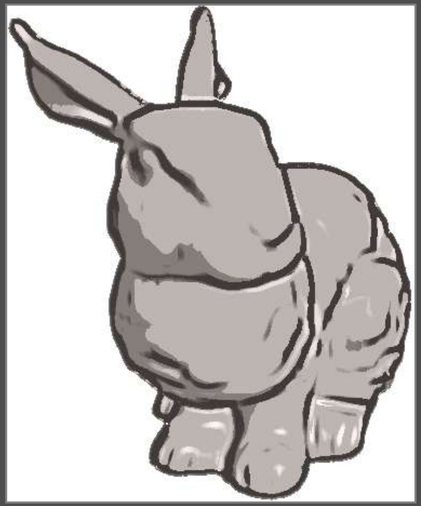






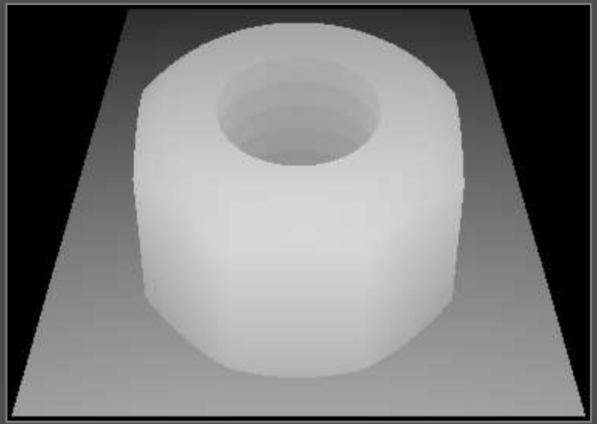
Feature Lines



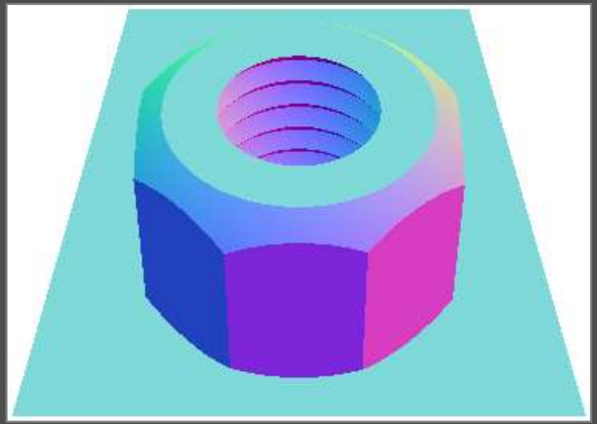



Feature Edge/Stroke Detection

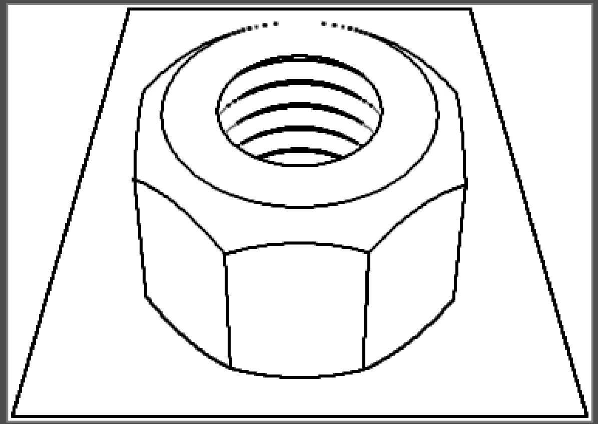
- significant lines/curves that are not silhouettes
  - geometric discontinuities or properties of the surface: creases & crest lines, lines that indicate high curvature, etc.
  - intersections of separate objects or surfaces of the same object
  - other curves significant for a scene (potentially off the surface)
- usually not necessary in image-space algorithms (normal buffer)



→

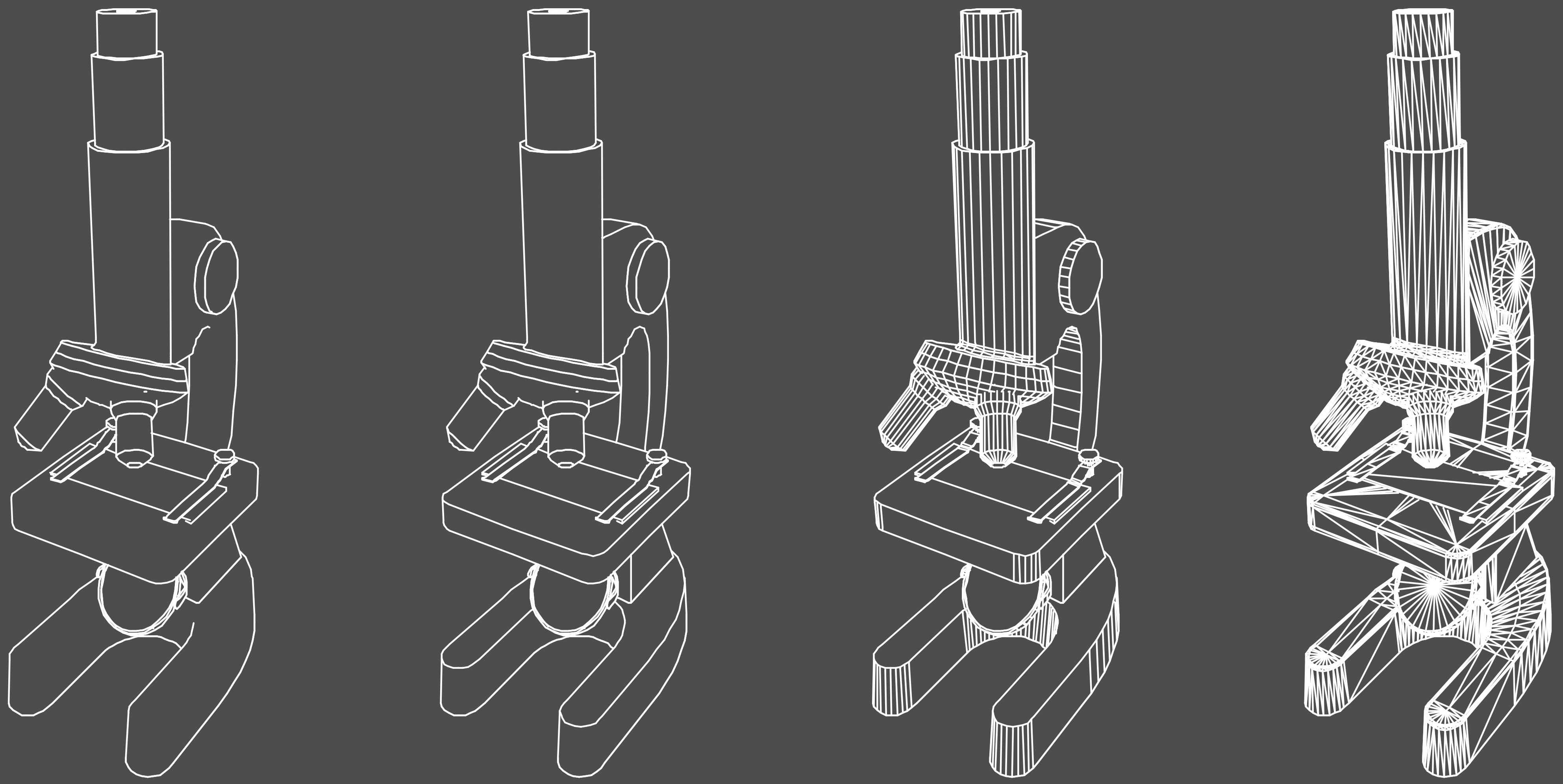


→



- necessary in some hybrid and all object-space algorithms

Surface Properties: Edge Classification

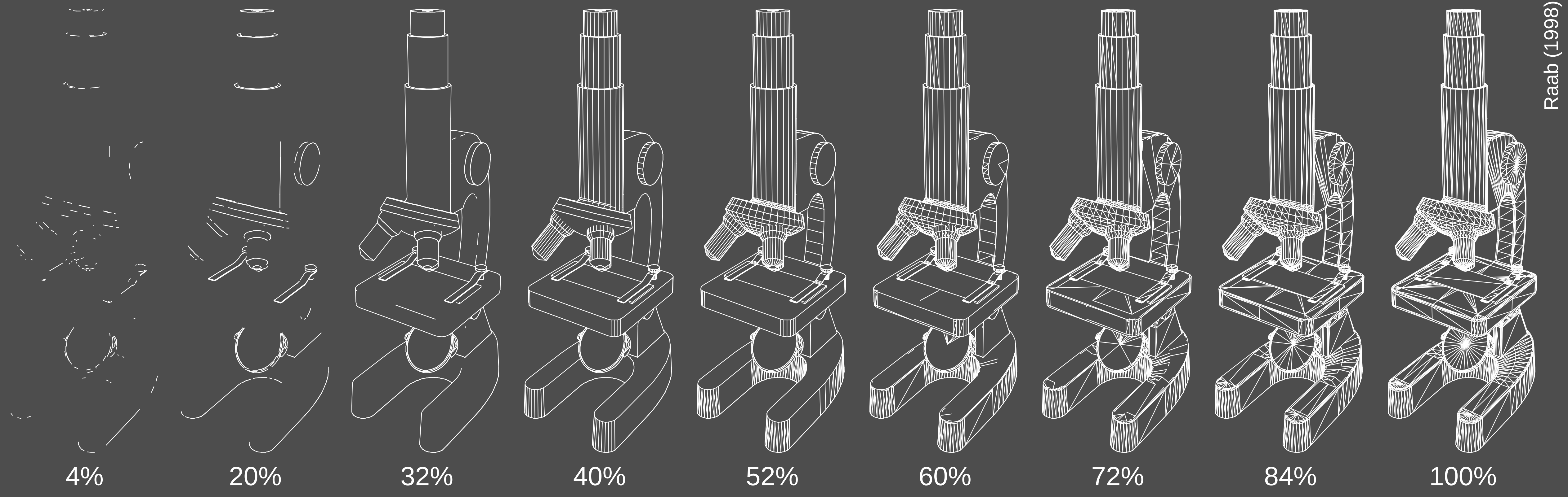


silhouette edges | + sharp edges | + smooth edges | + triangulation edges

Raab (1998)

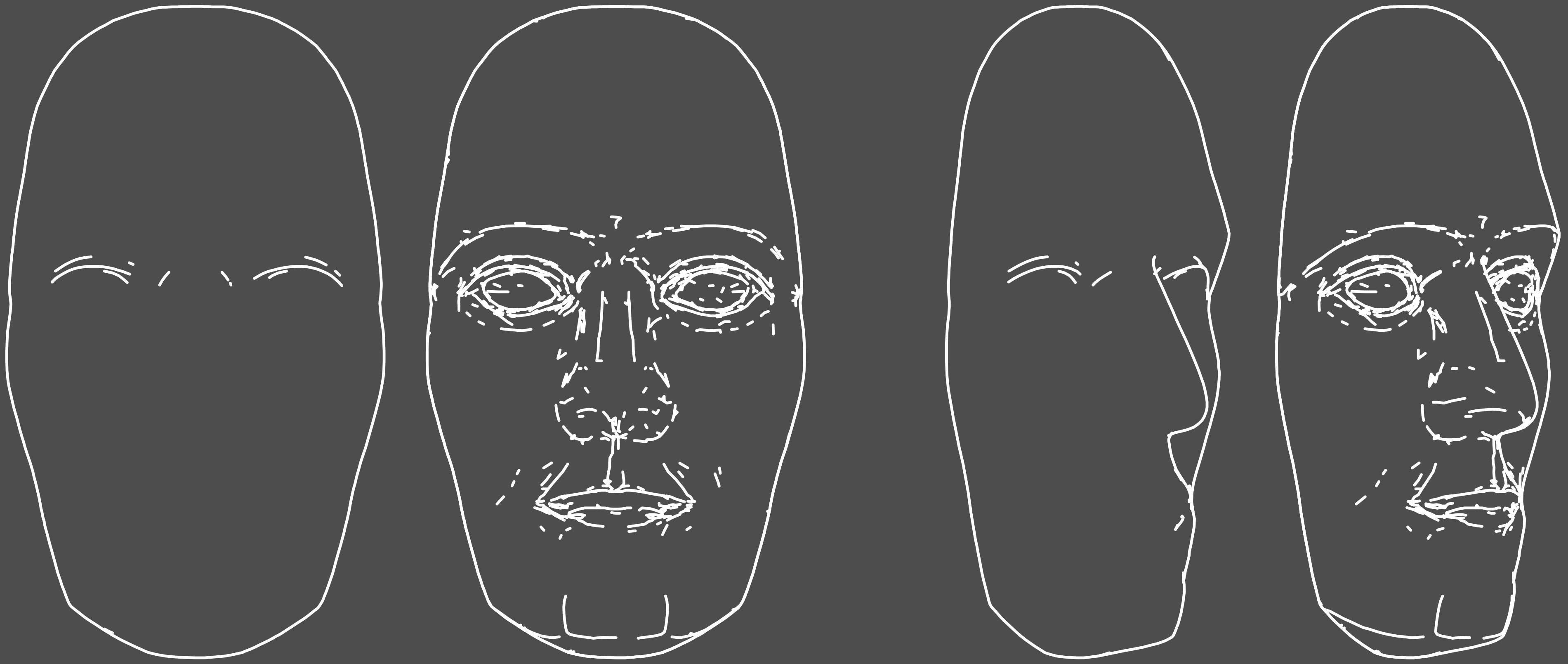
Features based on Surface Properties: Sharp Edges

- trivial approach: threshold angle between two adjacent polygons
- threshold is model-dependent
- **no perfect threshold for most models!**



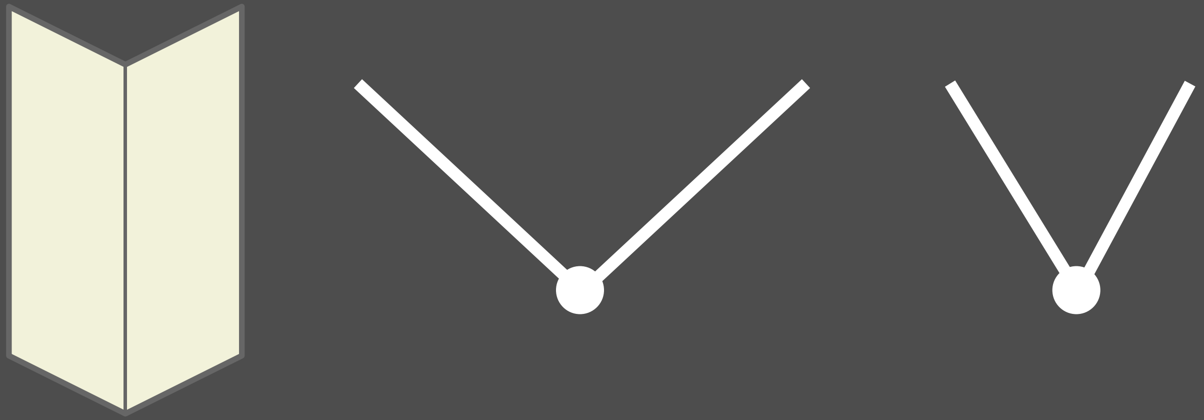
Features based on Surface Properties: Feature Extraction

- using methods from feature extraction
- usually based on measures computed from principal curvatures
- many specific techniques and measures possible



The Problem with Static Feature Lines

- in practice: there are no ideal creases
- degree of a crease changes depending on view

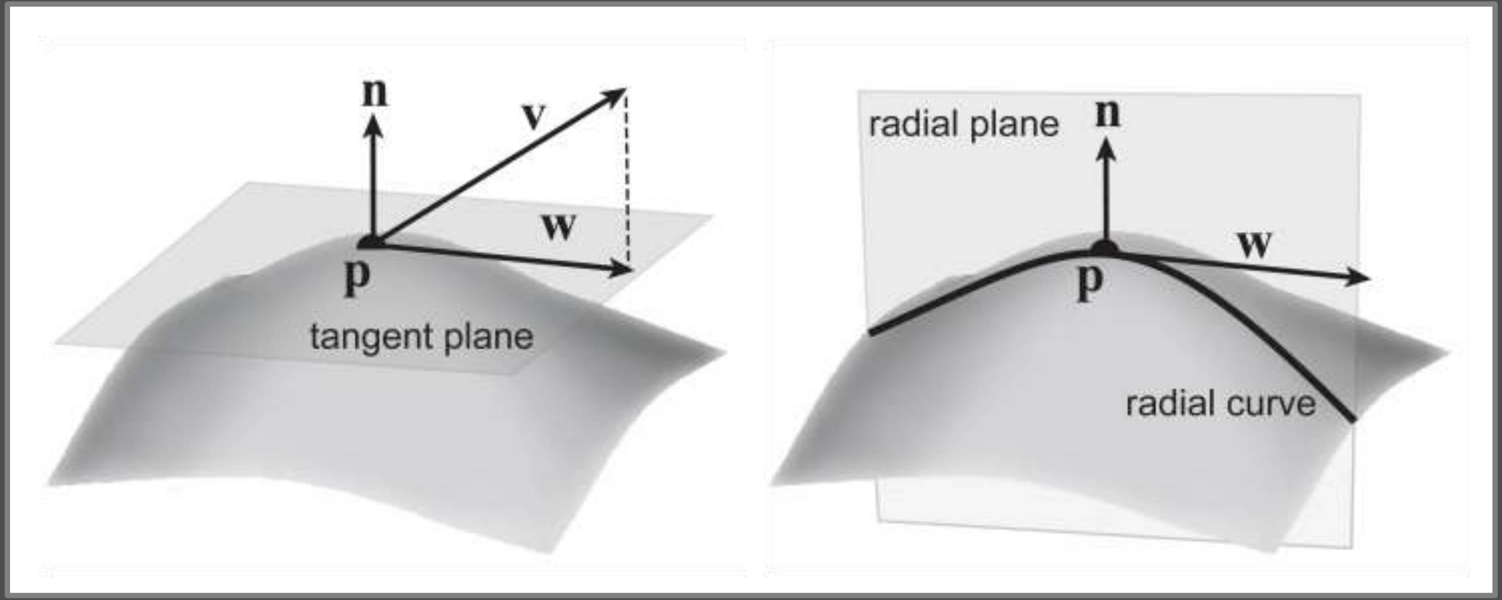


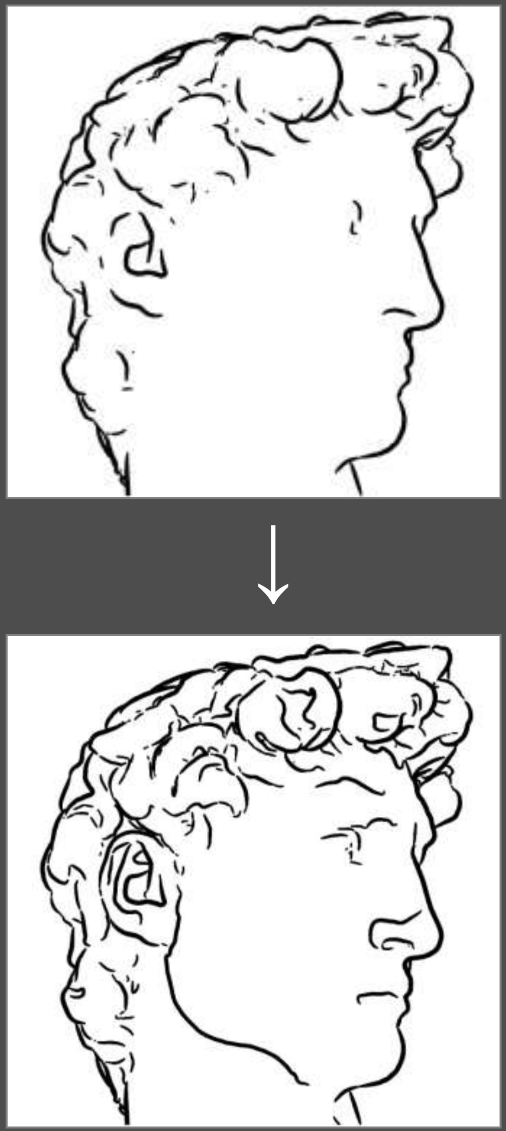
- thus: the way we perceive a surface **is not absolute**, it depends on our view of it
- static creases not beautiful on natural objects:

Judd et al. (2007)

Suggestive Contours: DeCarlo et al. (2003, 2004)

- intuitive notion: curves where silhouettes first appear when slightly turning the camera
- points of inflection of the radial curvature κ_r where surface bends away from viewer
- radial curvature κ_r:

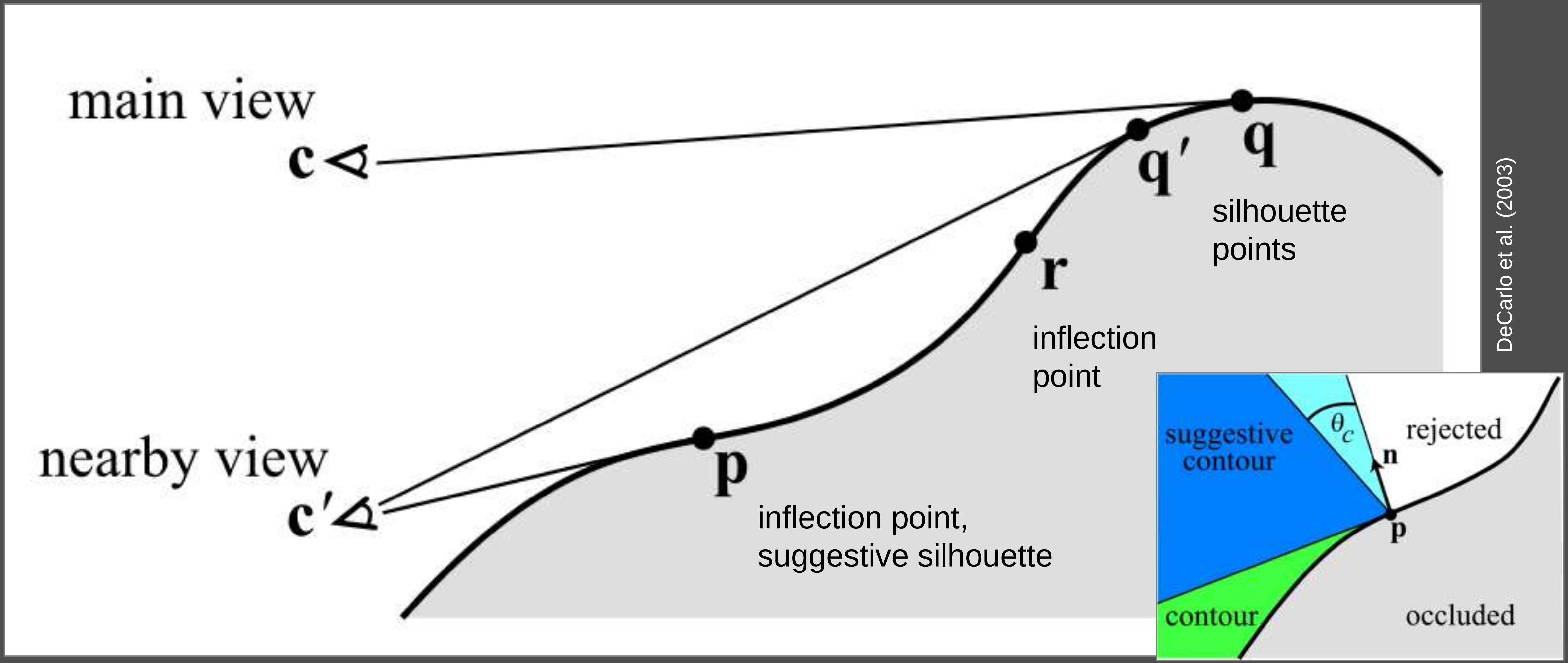




DeCarlo et al. (2003)

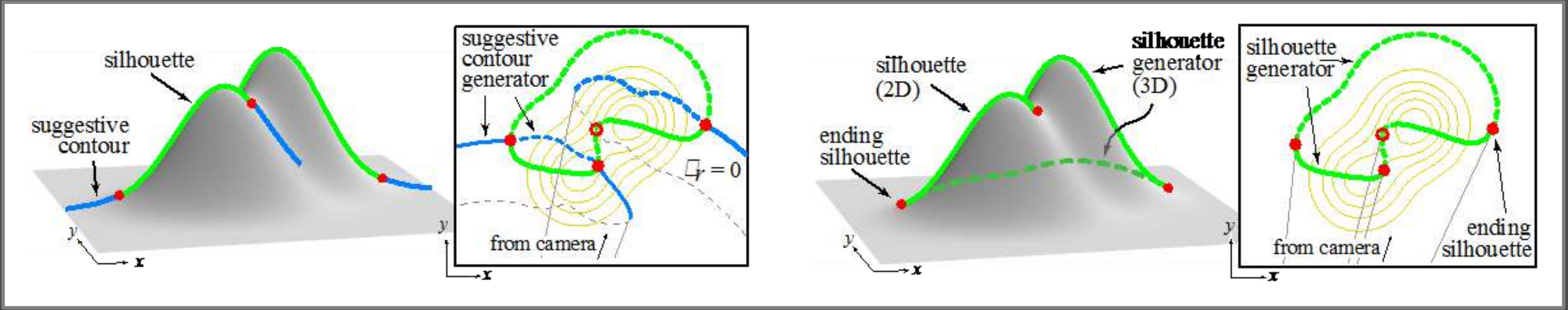
Suggestive Contours: DeCarlo et al. (2003, 2004)

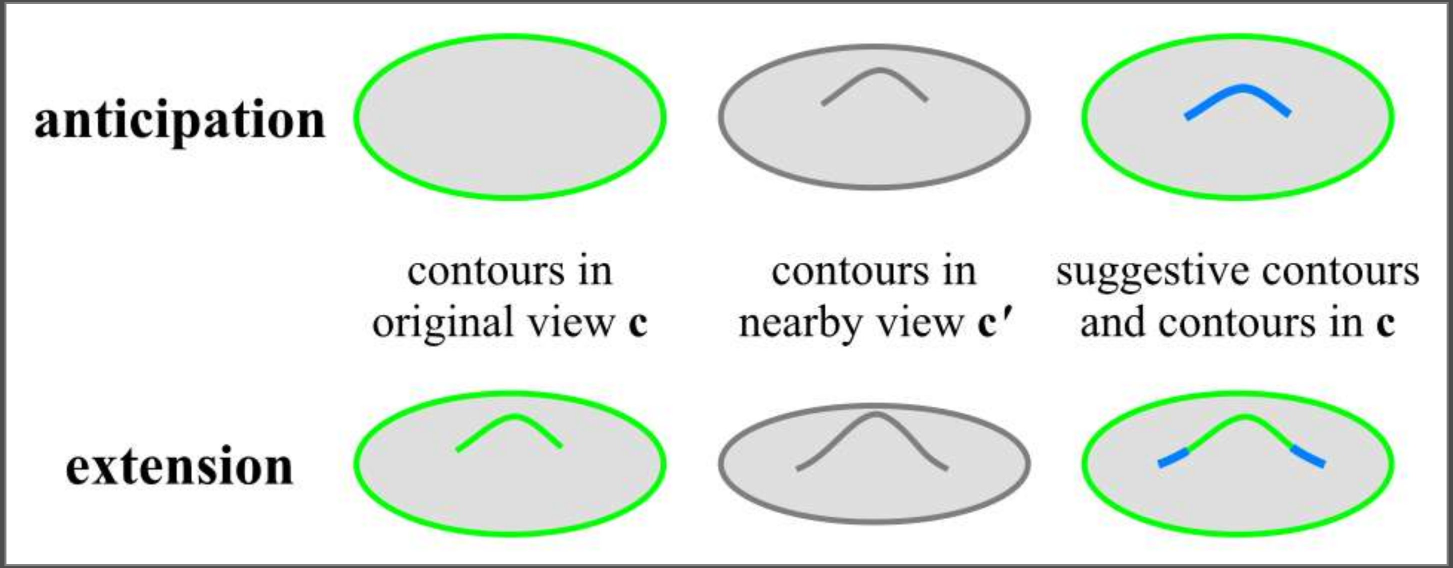
- inflection points of radial curvature, surface bends away from viewer:



DeCarlo et al. (2003)

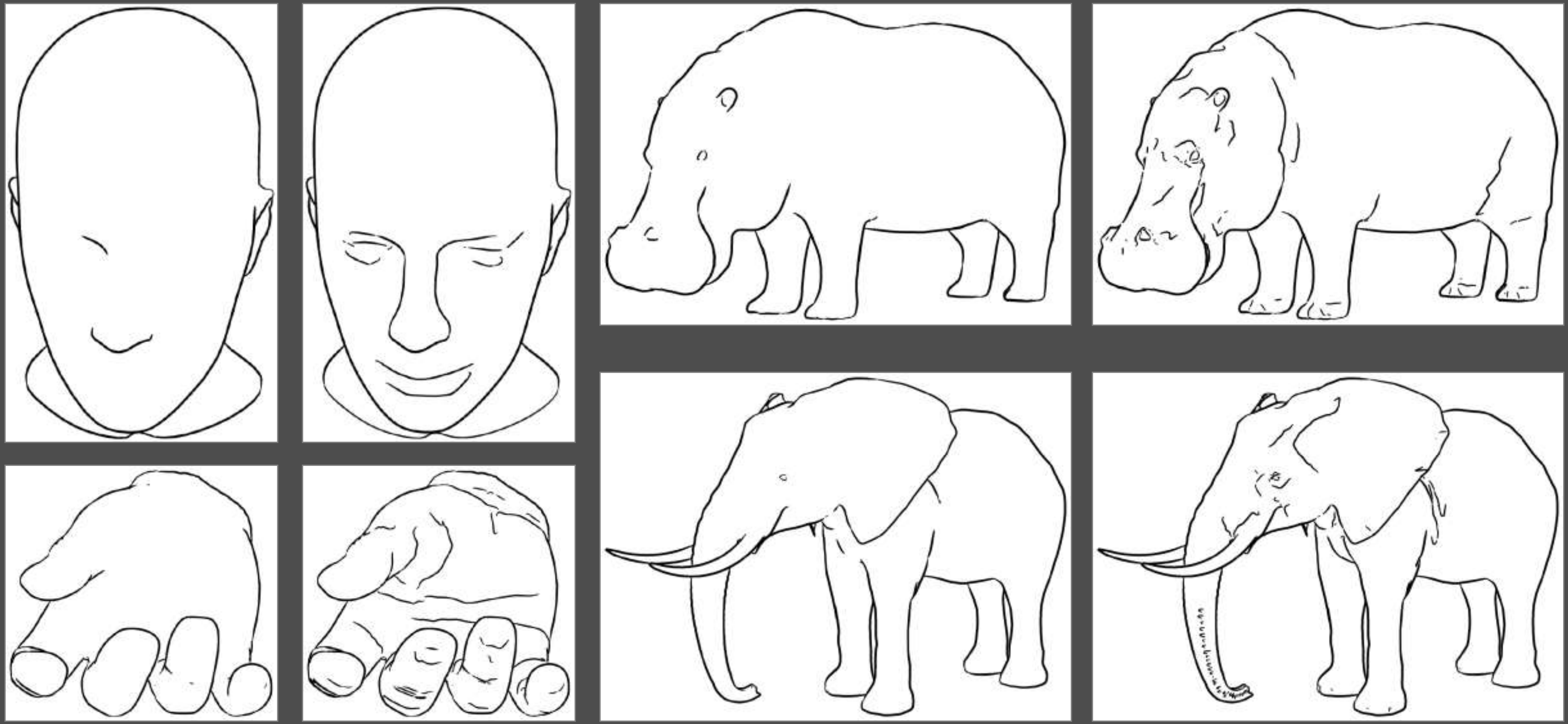
Suggestive contours extend silhouettes (screen-space C¹)





DeCarlo et al. (2003)

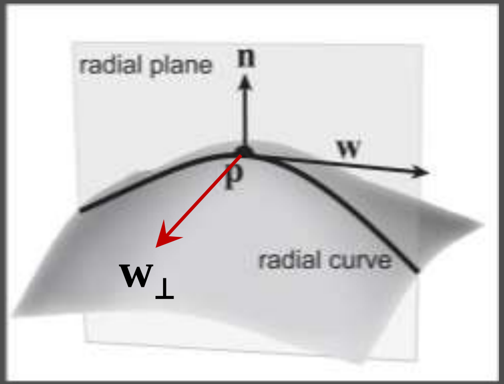
Suggestive Contours: Examples

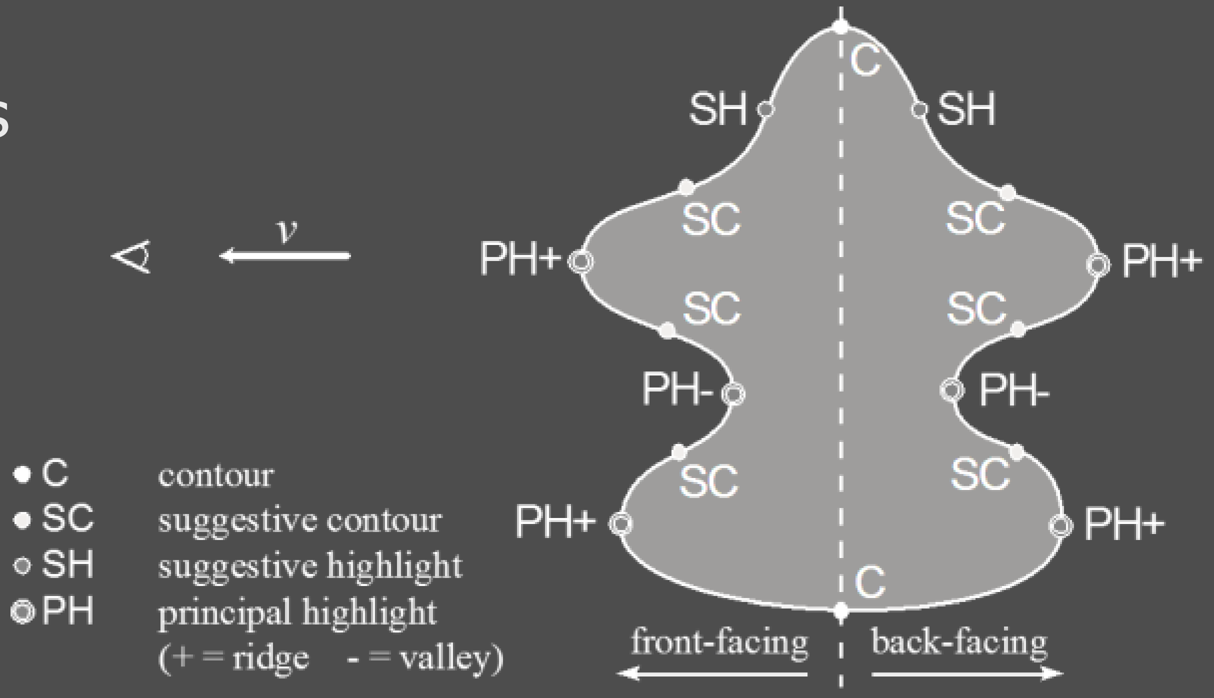


DeCarlo et al. (2003)

Highlight Lines: DeCarlo & Rusinkiewicz (2007)

- suggestive highlights
  - complementary to suggestive contours
  - positive maxima (or negative minima) of radial curvature
- principal highlights
  - strong positive maxima (or negative minima) of curvature along $w_\perp$





DeCarlo & Rusinkiewicz (2007)

- typically shown as black or white lines on a gray object

Highlight Lines: Examples

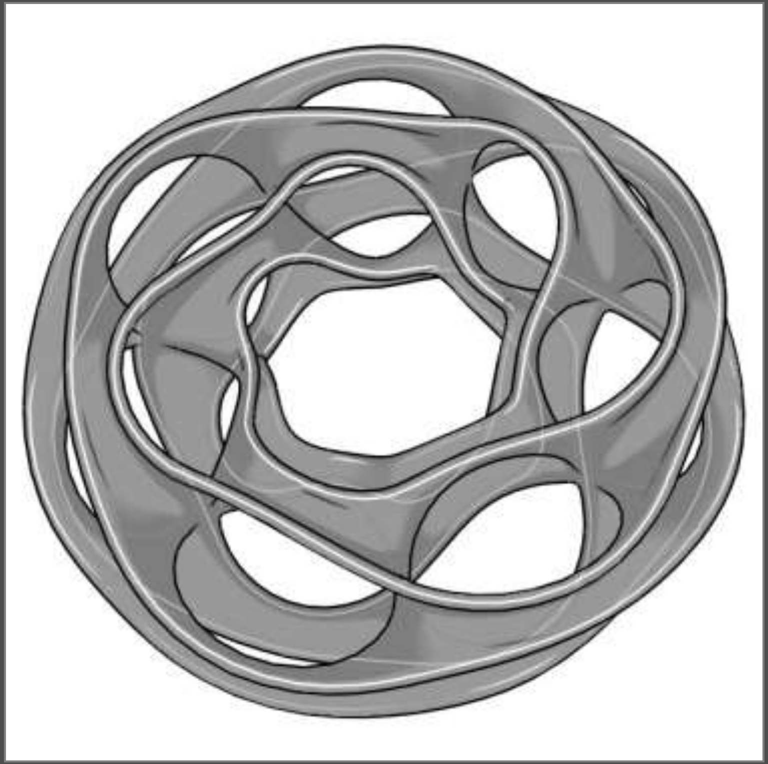
C + SC + PH + SH

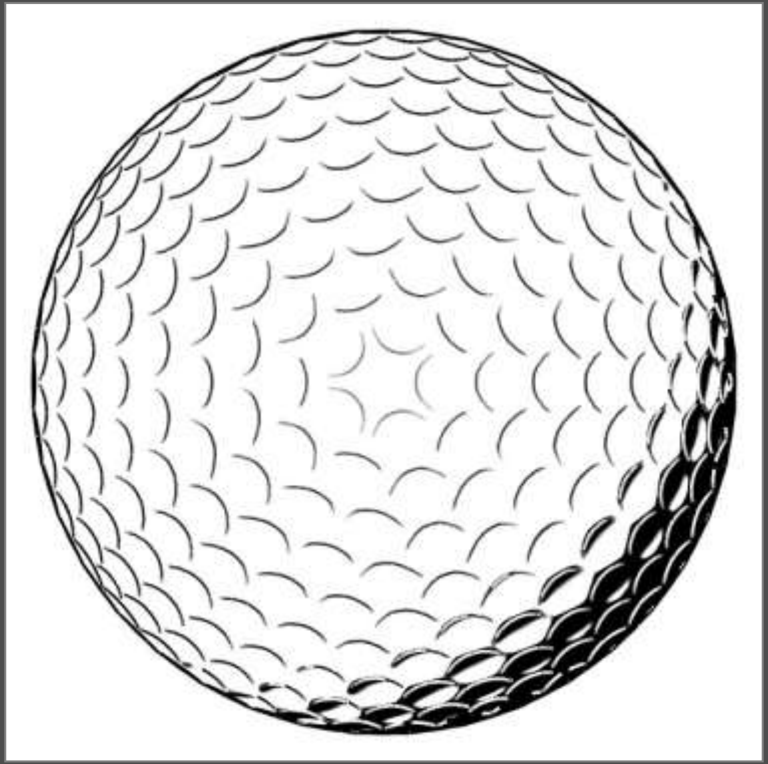
C + SC + PH

C + SC + PH ridges

DeCarlo & Rusinkiewicz (2007)

Highlight Lines: Examples



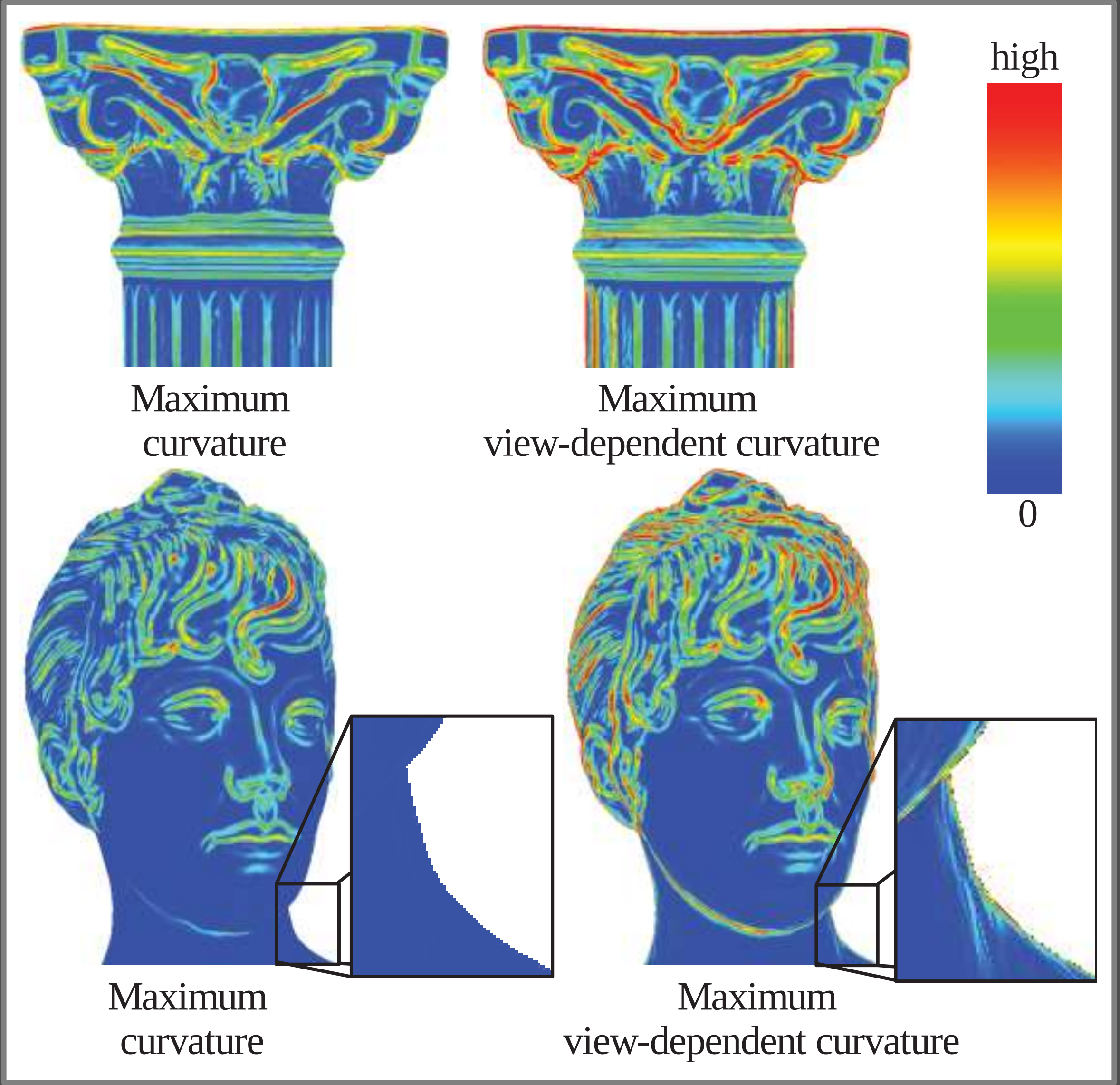


DeCarlo & Rusinkiewicz (2007)

Apparent Ridges: Judd et al. (2007)

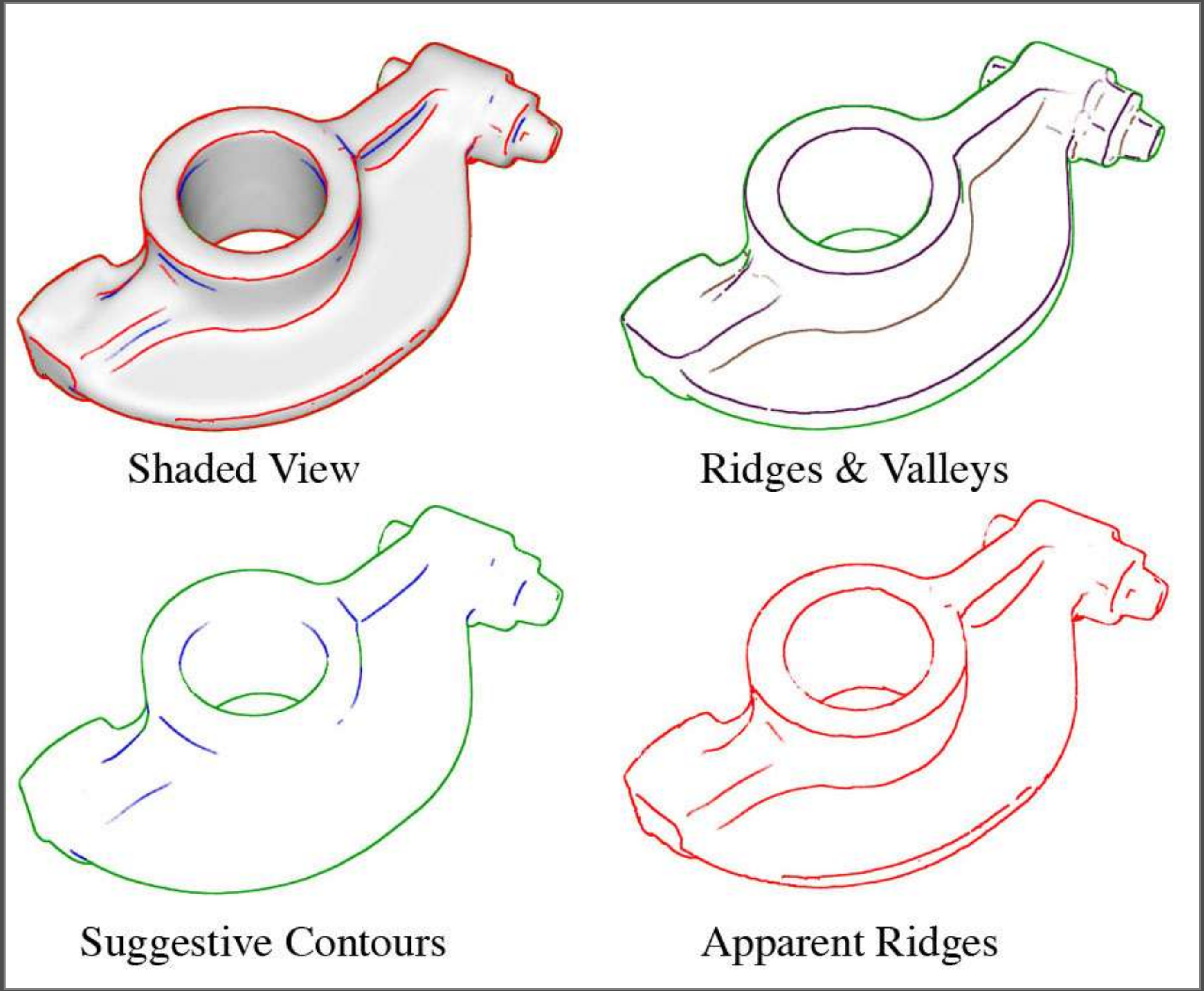
- need to define a view-dependent ridge-like feature
- curvature alone is not enough (is view-independent)
- apparent ridges: points at which the maximum view-dependent curvature assumes a local maximum in the principal view-dependent curvature direction

→ where high shading contrast is most likely under arbitrary lighting



Judd et al. (2007)

Apparent Ridges: Extend Silhouettes along Object Edges

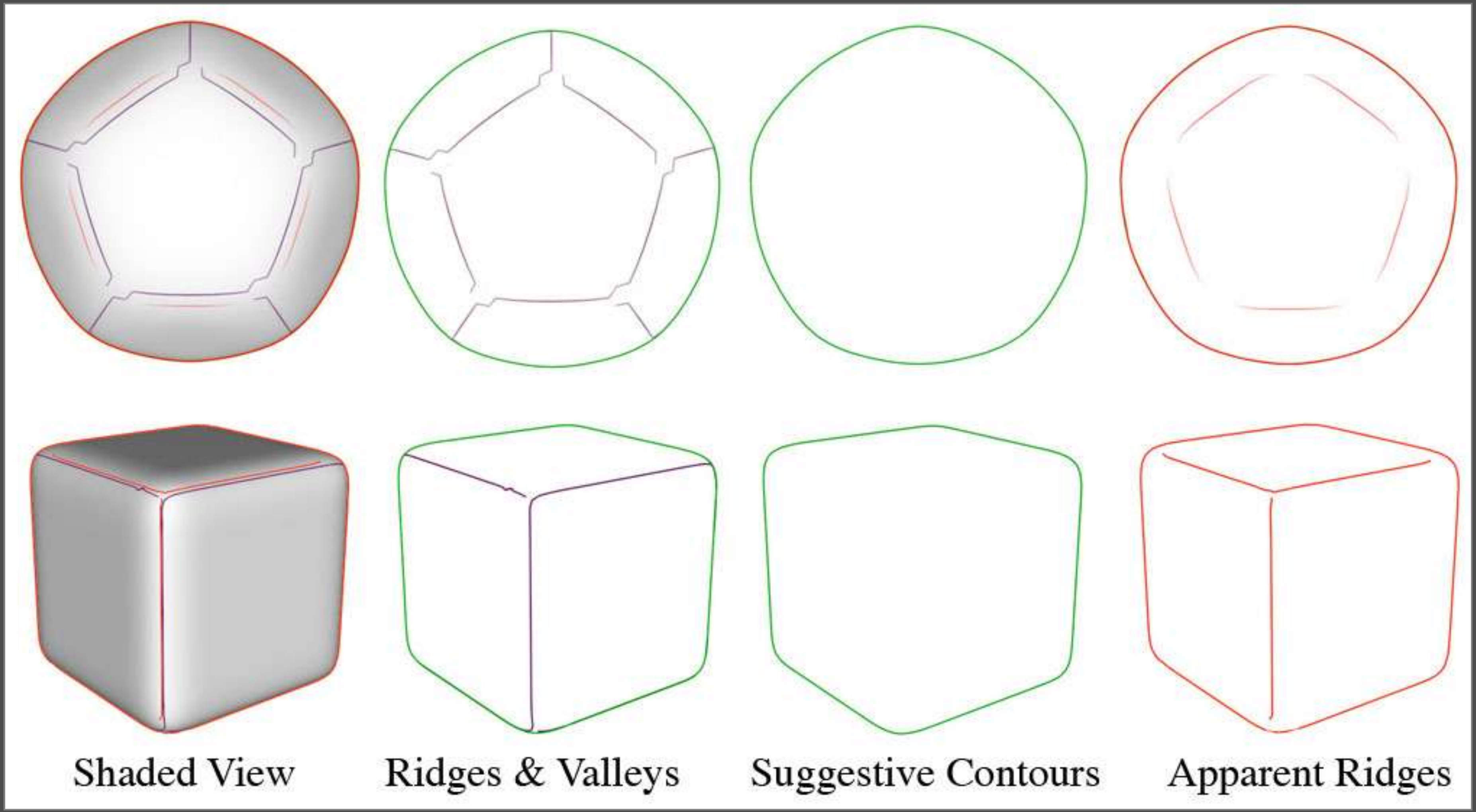


- silhouette/contour
- ridge (static)
- valley (static)
- suggestive contour
- apparent ridge

Judd et al. (2007)

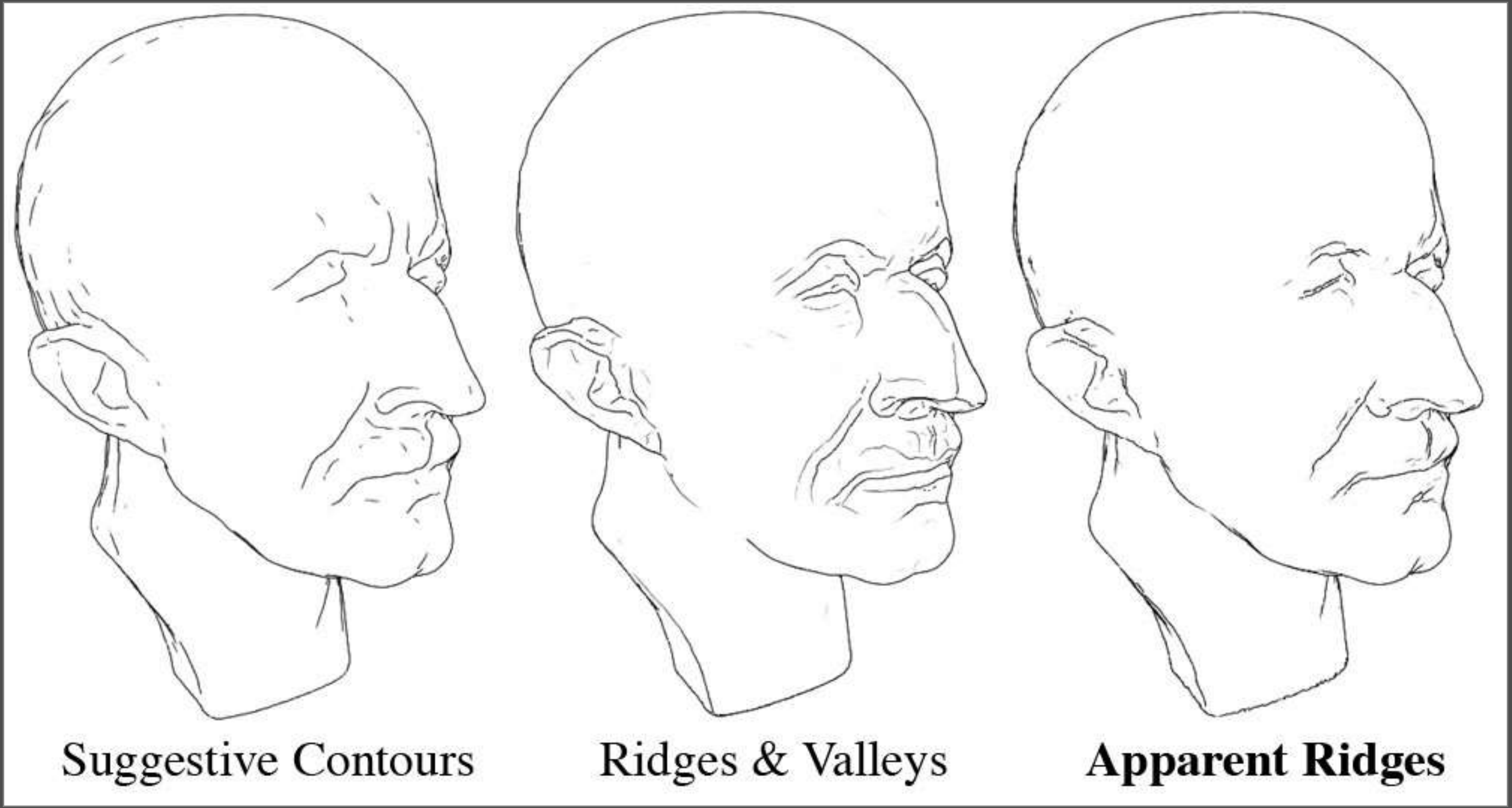
Apparent Ridges: Line Comparison (2)

silhouette ridge valley suggestive contour apparent ridge



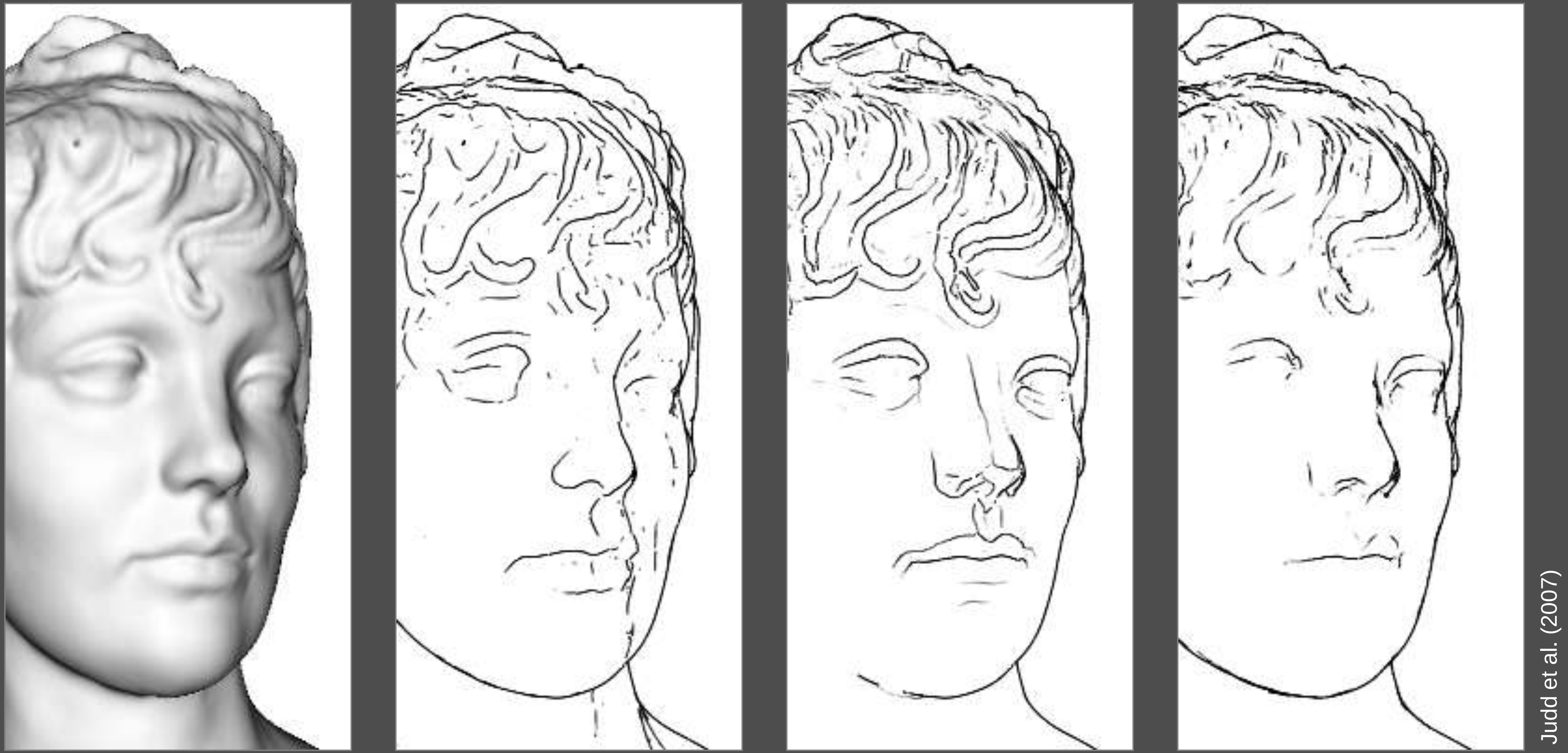
Judd et al. (2007)

Apparent Ridges: Examples



Judd et al. (2007)

Apparent Ridges: Examples



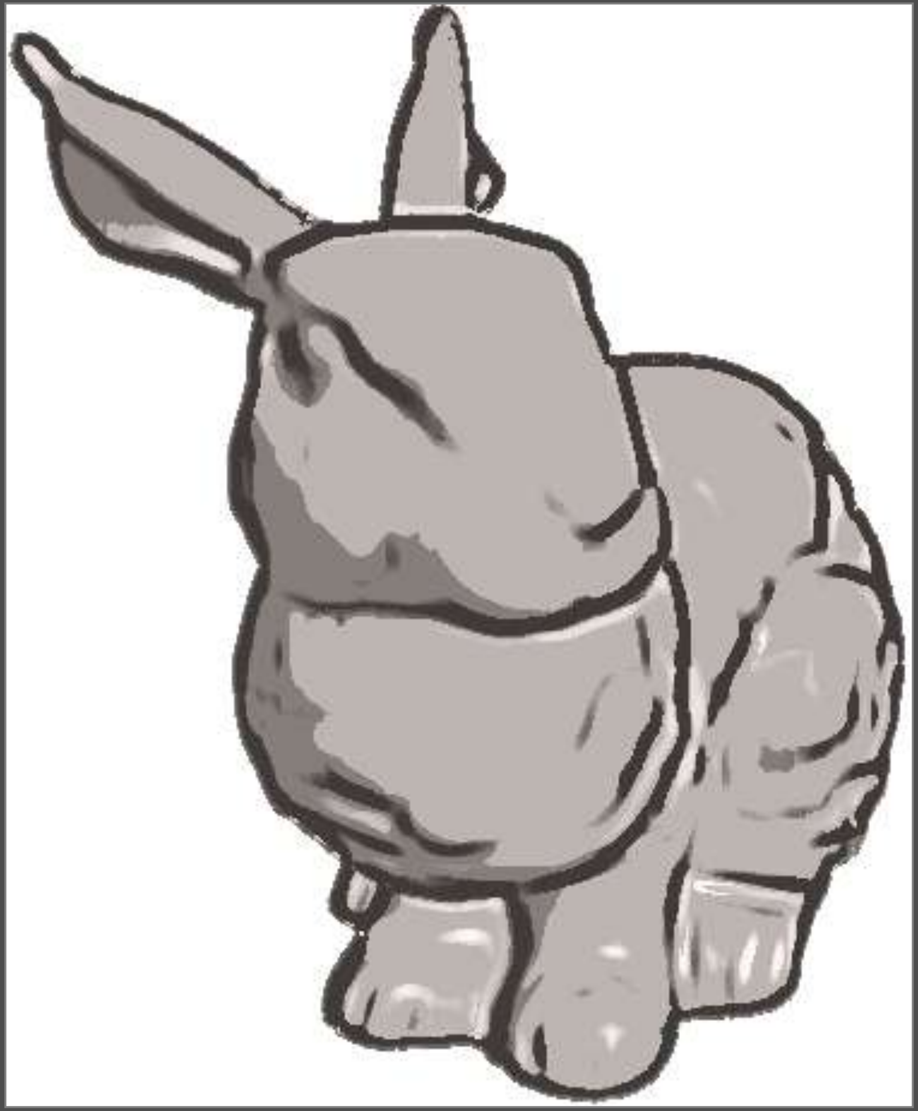
shaded | sug. contours | ridges & valleys | apparent ridges

Abstract Shading: Lee et al. (2007)

- new line concept that generalizes the suggestive contour concept
- based on ridge detection in image-space (thus view-dependent)
- inspired by traditional drawing
- new highlight features
  - similar to highlight lines
  - correspond to diffuse & specular highlights under arbitrary illumination
- lines work best when drawn on toon-shaded background rendering

Lee et al. (2007)

Abstract Shading: Control of Level of Abstraction

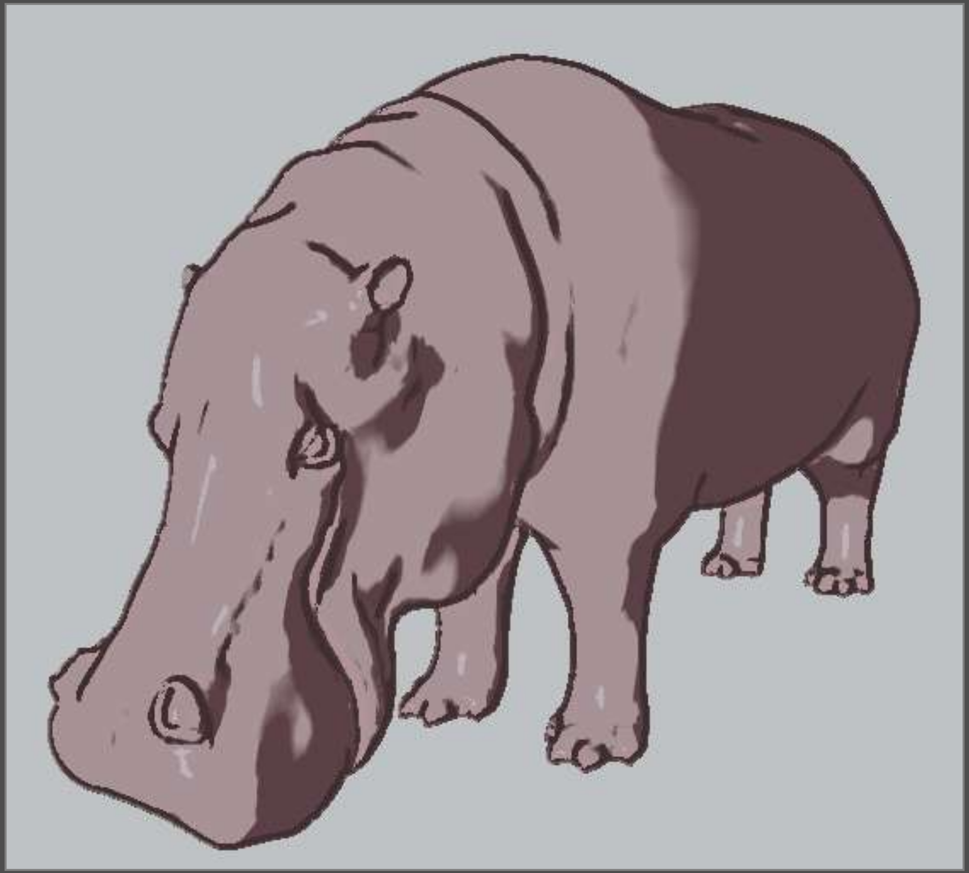


Lee et al. (2007)

automatic control

control by depth

Abstract Shading: Examples



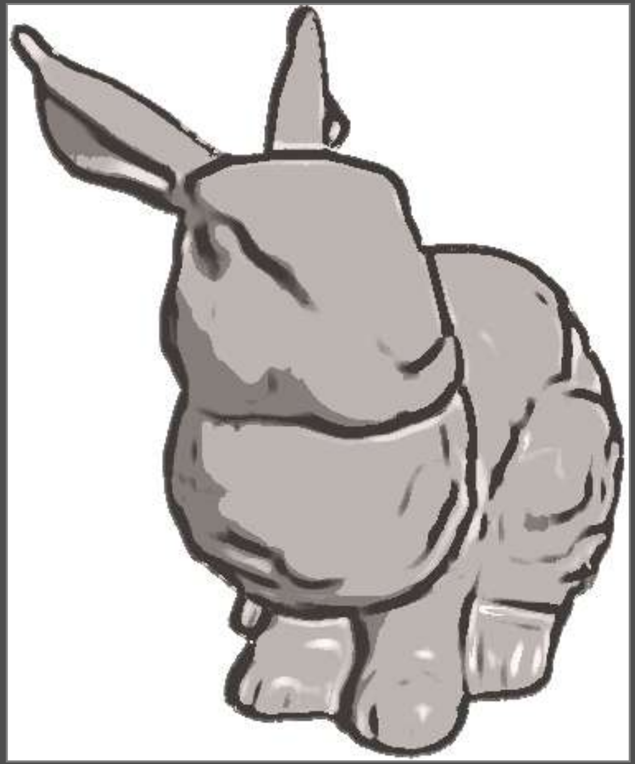
Lee et al. (2007)

A Zoo of View-Dependent Feature Lines

suggestive contours

principle highlights &
suggestive highlights

apparent ridges



abstract shading

Surface Properties: Manual Feature Edges

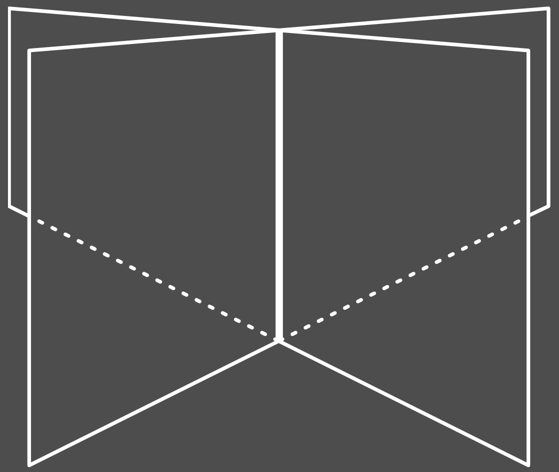
- automatic detection of some strokes types not possible or not desired ⇒ human interaction required
- WYSIWYG-NPR (Kalnins et al., 2002)
  - manual strokes drawn on objects
  - using tools such as, e.g., brushes or pens
  - convey different properties
    - texture
    - surface images
    - hatching
    - etc.
  - strokes represented in 3D

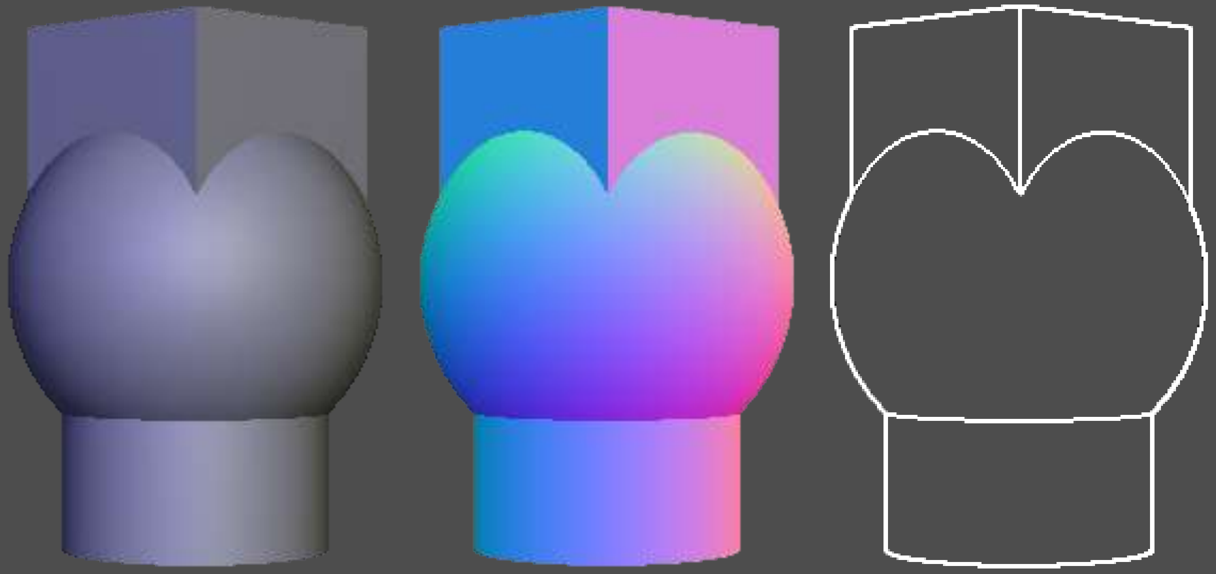



Kalnins et al. (2002)

Feature Lines: Intersections

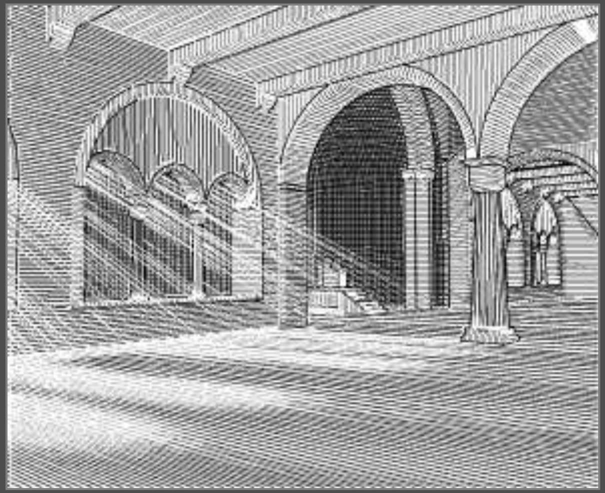
- lines that result from intersections of surfaces
- not denoting properties of surfaces
- not necessary for correctly modeled objects
- image-space: detection using normal buffer
- object-space: analytic computation
  - intersection of each triangle with all other triangles
  - speed-up by using space partitioning, bounding boxes, and polygon connectivity
  - high computational complexity ⇒ fairly slow ⇒ typically not used



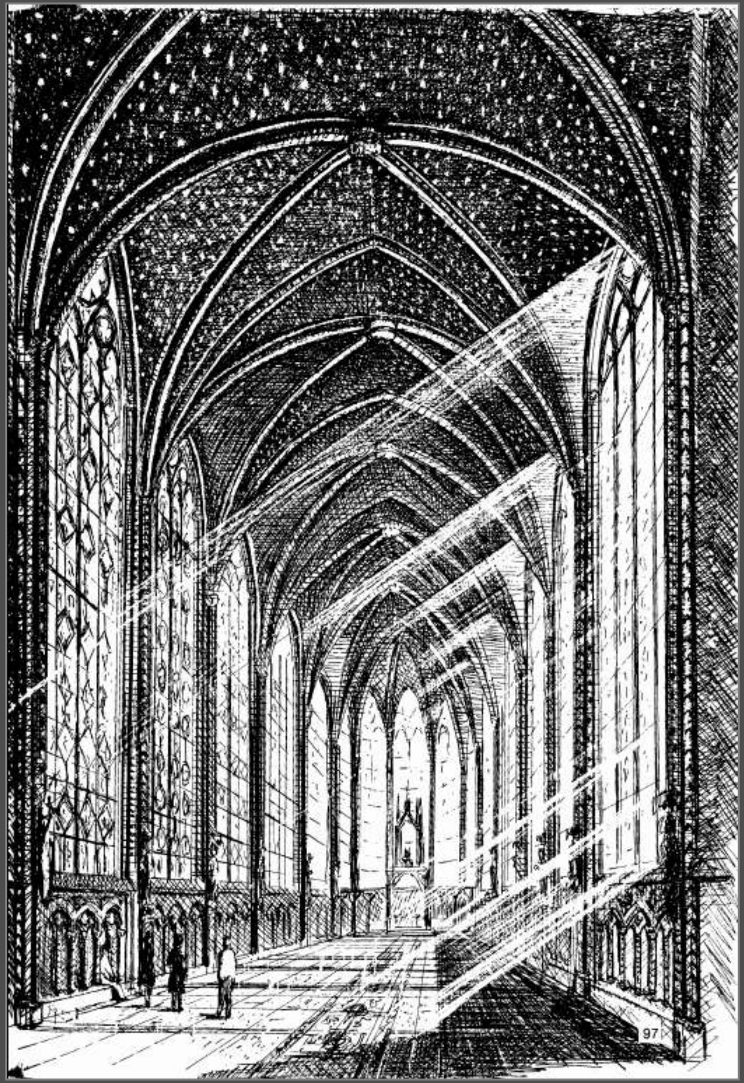


Off-Surface Strokes

- linear structures directly represented as strokes
  – as part of the 3D model
  – automatically generated or manually created
- structures emerging from surfaces
  – e.g., hair, fur, etc.
- botanical primitives
  – e.g., grass, tree branches, etc.
- atmospheric effects
  – e.g., light, shadow, rain, etc.



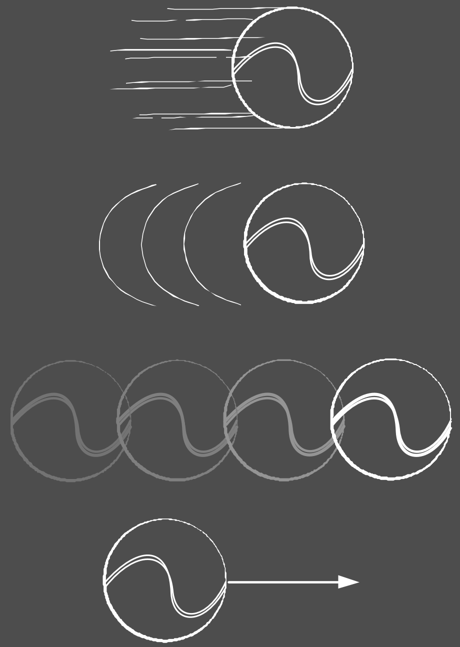
Hamel (2000)

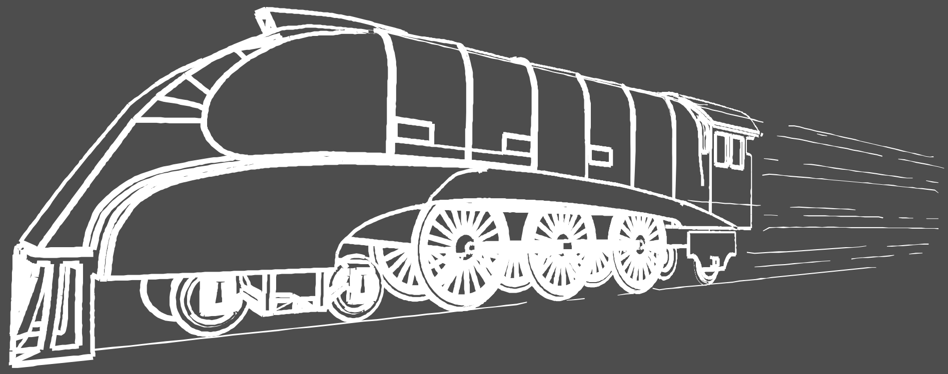


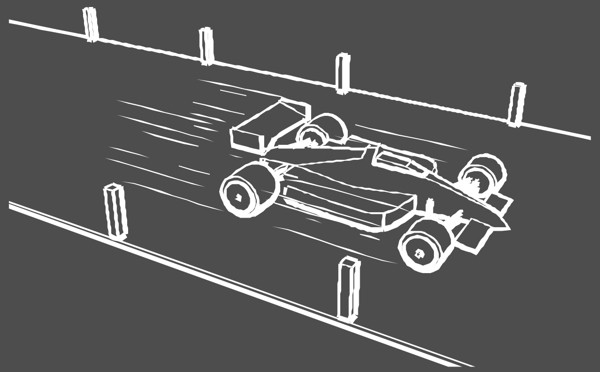
→ generated by specific algorithms and represented as 3D strokes

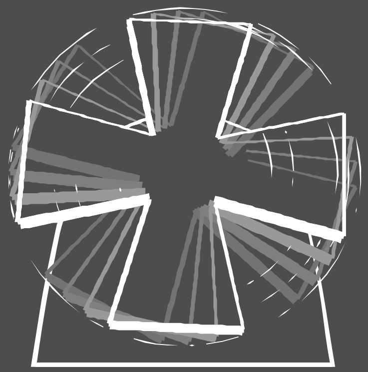
Masuch et al. (1999)

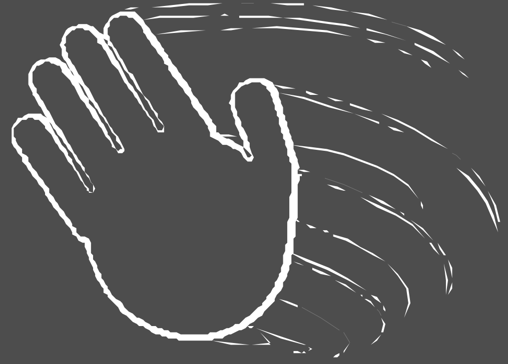
- depiction of motion in still images
- derived from techniques in comics
- four general techniques
  - speedlines
  - repeated parts of the contour
  - repeated silhouettes with changed shading
  - arrows



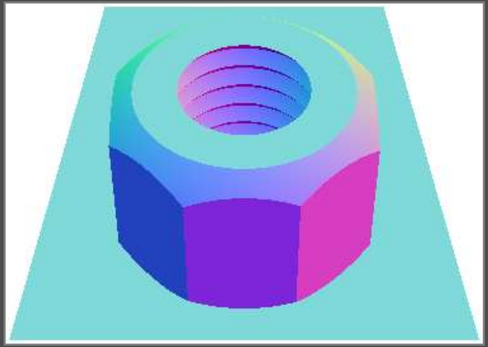
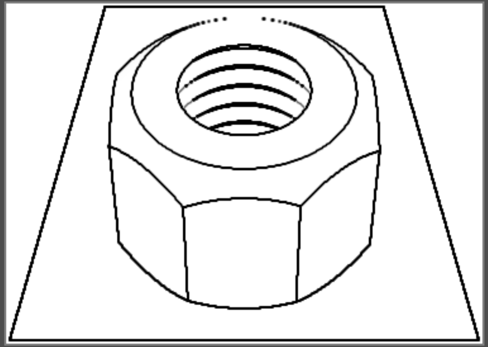
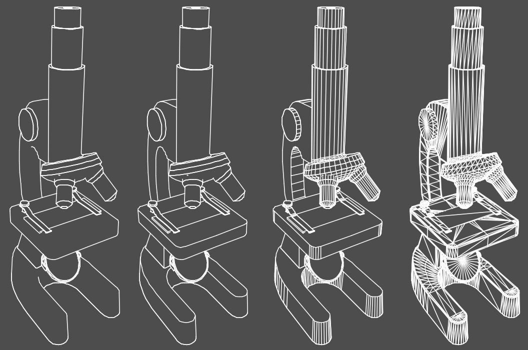


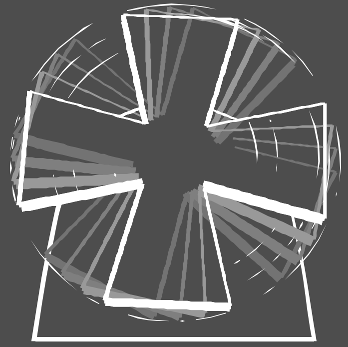






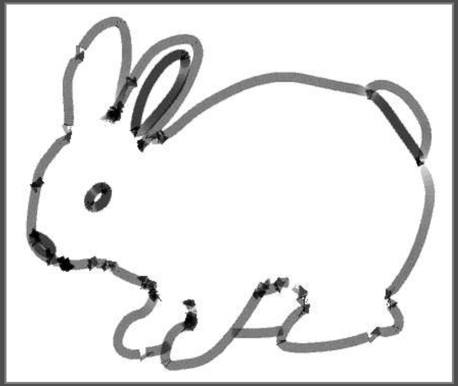
Feature Stroke Detection: Summary

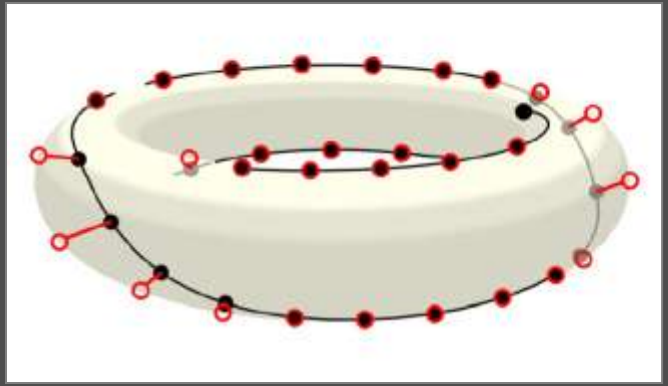




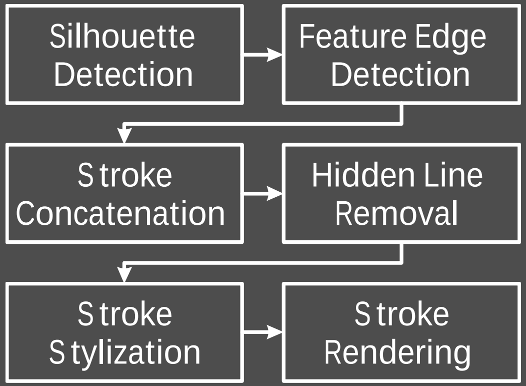


- done automatically in image-space (normal buffer)
- treatment only needed in object-space and for some hybrid methods
- different classes of feature strokes
  - surface properties using sharp edges, feature extraction, or suggestive contours (automatic vs. manual methods; static vs. view-dependent strokes)
  - intersections (not necessary for good models, too expensive in object-space)
  - off-surface strokes (e.g., speedlines)
- features in object-space: stored in 3D & treated similar to silhouettes

Strokes and Stroke Processing







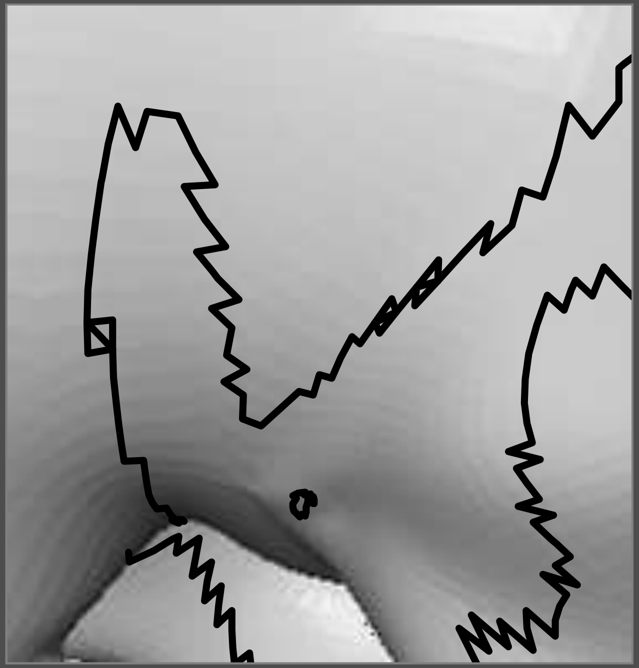
Triangular Meshes & Line Stylization: Artifacts

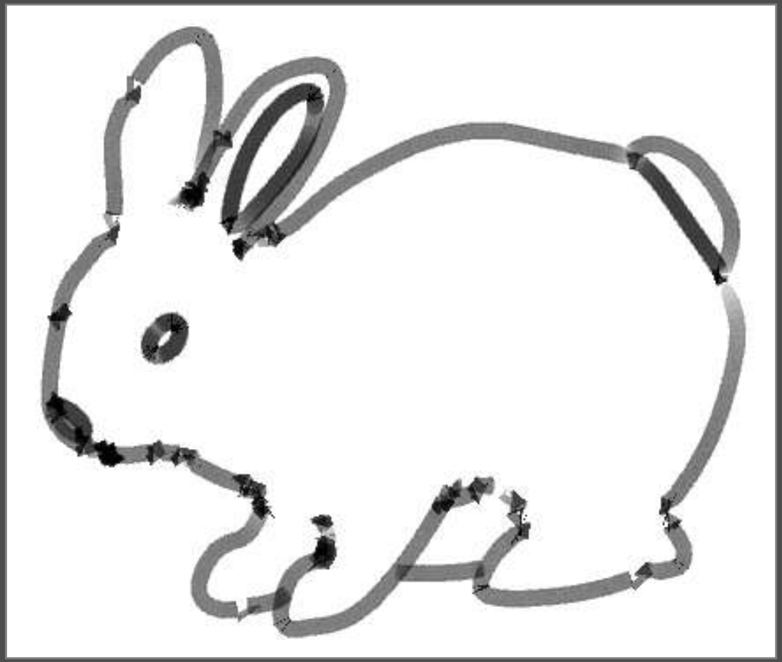
- line rendering typically computed as part of triangular mesh

⇒ restricted to edges of the mesh

⇒ triangulation artifacts in line styles ⇒ stroke texture folding

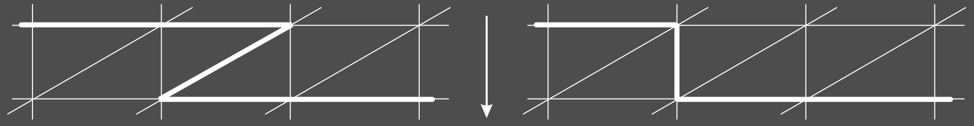
→ sharp bends, zig-zags, short edges, triangle chunks

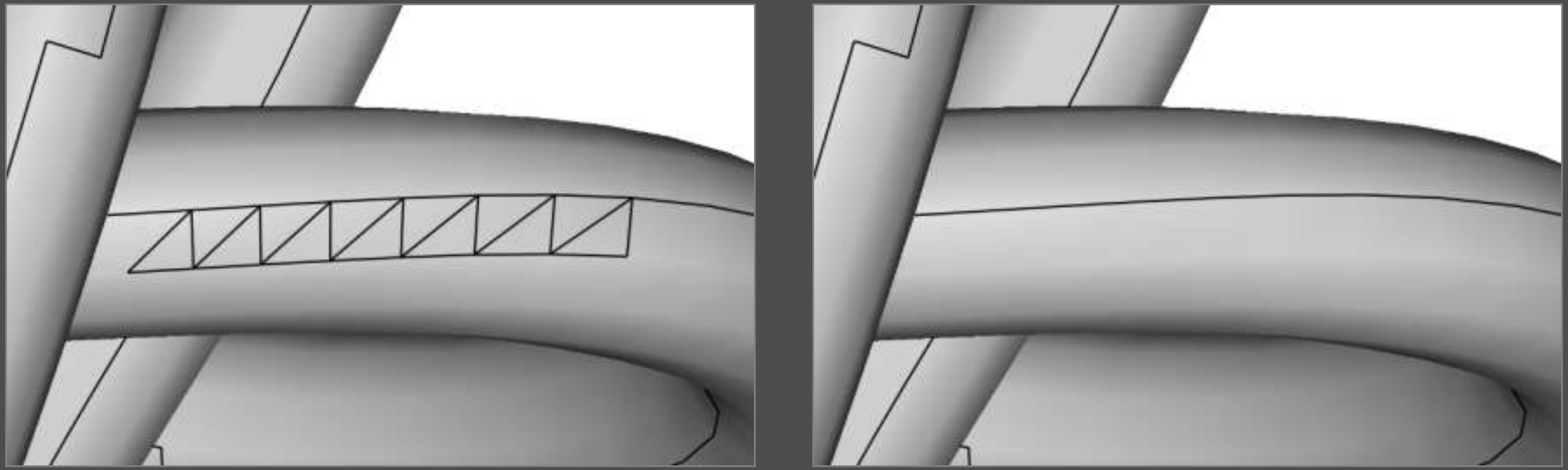




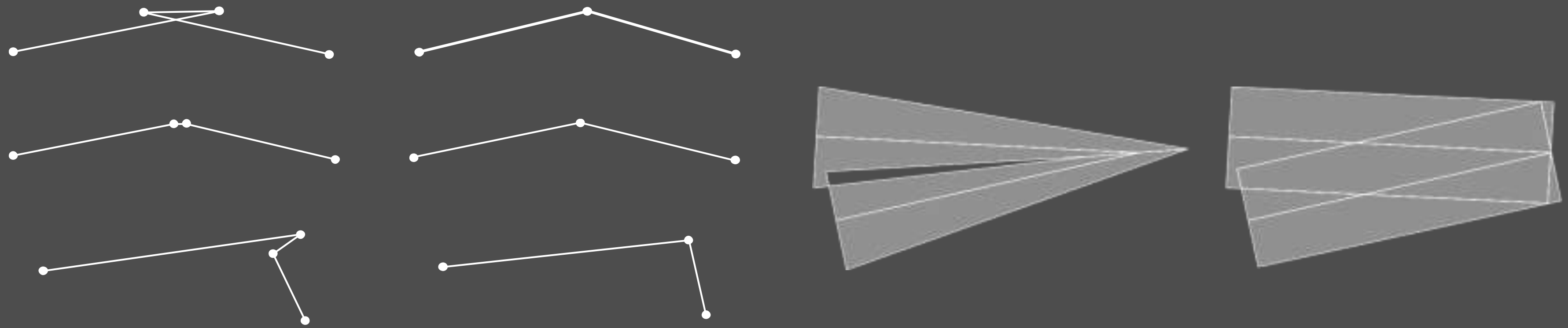
Stylization Artifact Removal

- two stages for artifact removal:
  1. on the mesh





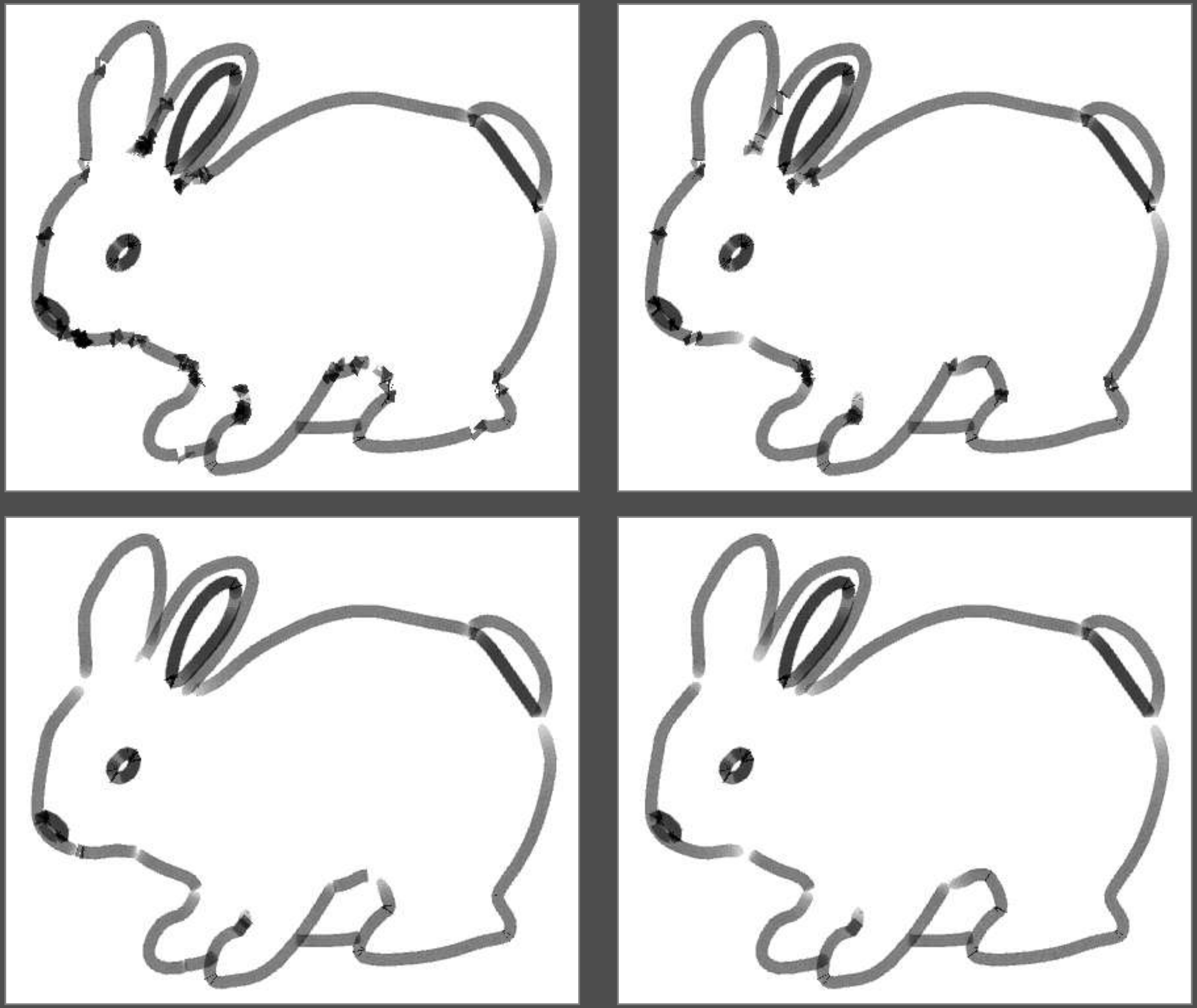
  2. stroke analysis



Stylization Artifact Removal: Example

none

on the mesh

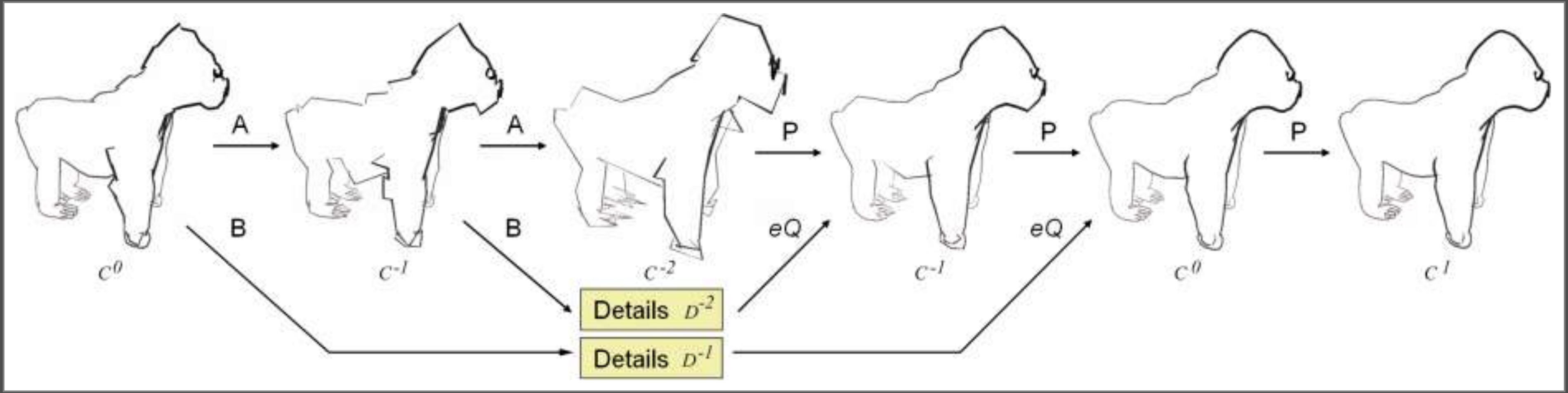


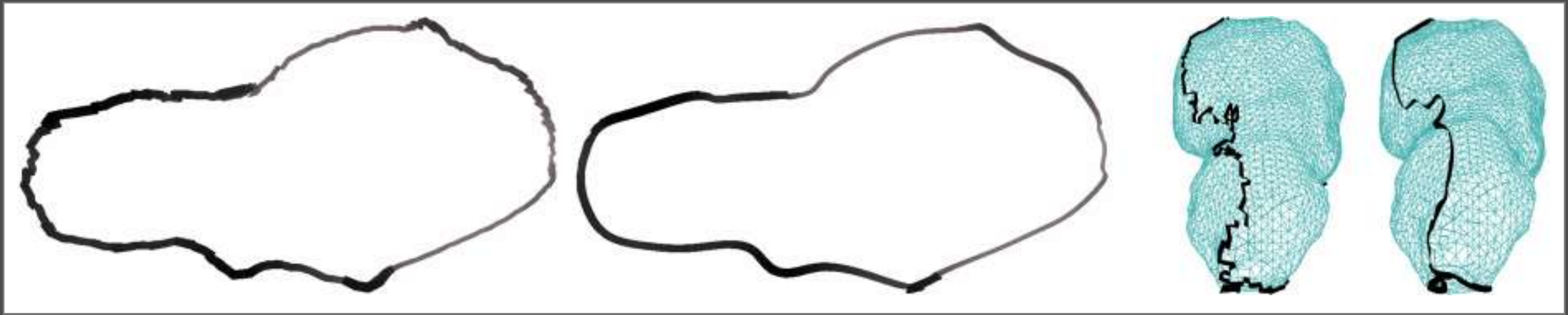
stroke analysis

all

Artifact Removal by Multiresolution Curve Analysis

- errors in high frequency of stroke, low pass filter to remove
- several steps of multiresolution analysis for improved results

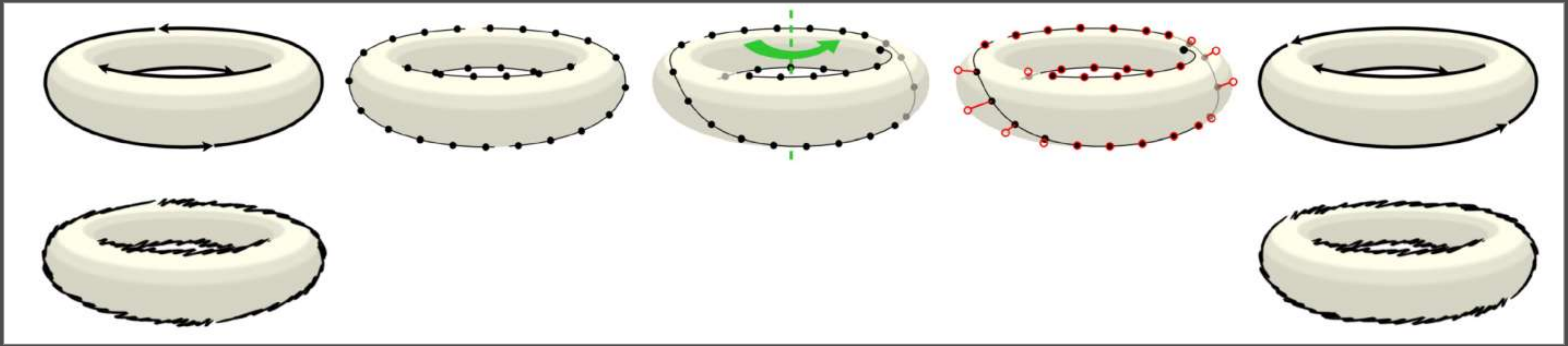




Foster et al. (2004)

Stroke Parameterization for Stylized Animation

- temporal coherence of (silhouette) stroke stylization necessary
- coherent stylized silhouettes: Kalnins et al. (2003)
- propagation of parameterization from one frame to the next frame



Kalnins et al. (2003)

Coherent Stylized Silhouettes: Examples

- two schemes: 2D coherence or 3D coherence
  - 2D coherence for stroke attributes (e.g., line width) using arc-lengths
  - → may lead to swimming artifacts
  - 3D coherence for surface details in stroke textures by stroke scaling
  - → may awkwardly scale strokes
- without coherence:
  - swimming effects
  - sudden texture changes
- typically a mix of 2D and 3D coherence necessary

Kalnins et al. (2003)

Stroke Detection and Processing Pipelines

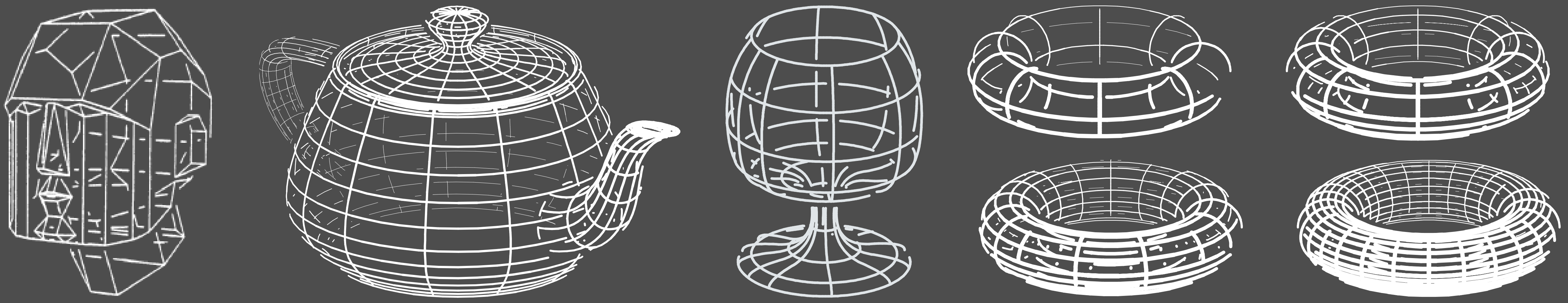
- stroke processing usually in form of stroke pipelines
- each node in pipeline adds, removes, or modifies edges/strokes
- steps in a typical pipeline:

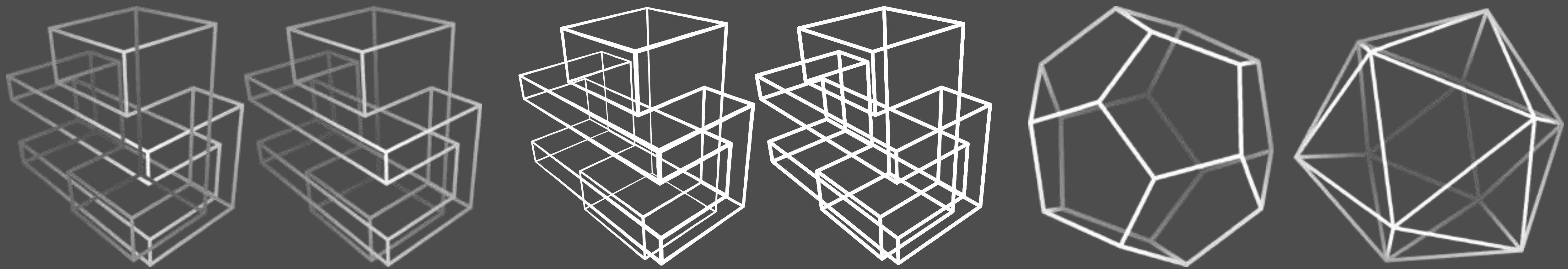
Silhouette Detection → Feature Edge Detection → Stroke Concatenation → Hidden Line Removal → Stroke Stylization → Stroke Rendering

- stroke pipeline captures and manipulates stroke properties
- other steps in line rendering pipelines possible, for example
  - extraction of other line types (e.g., hatching)
  - artifact removal for strokes from polygonal meshes
  - line shading (e.g., illumination model or depth cueing)

Additional Stroke Processing Steps

- line haloes (Appel (1979), Elber (1995), Loviscach (2004)); depth cues

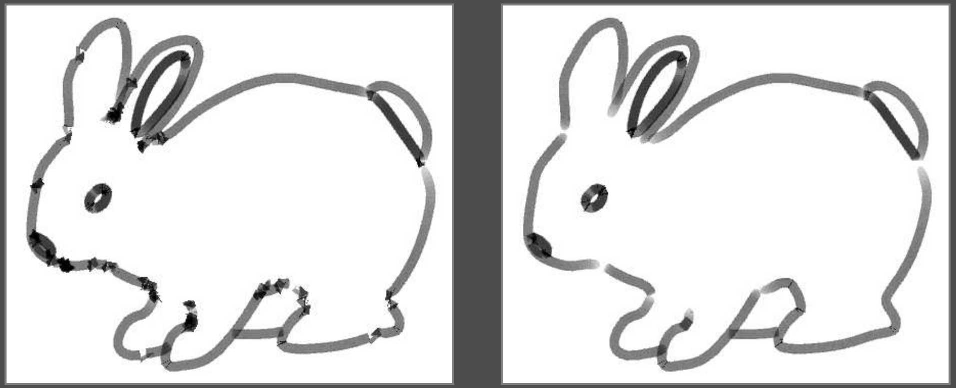


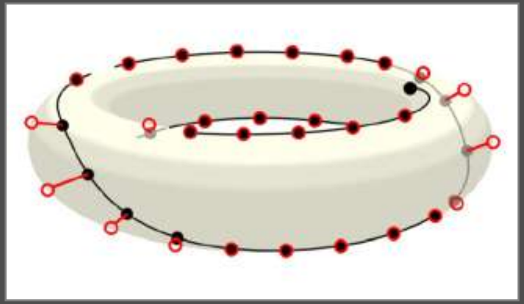


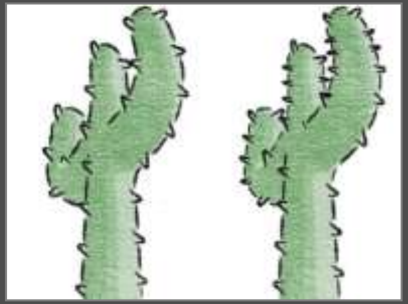
Appel (1979) and Elber (1995)

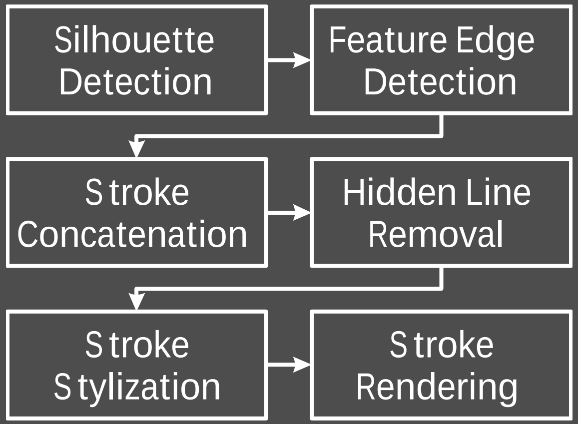
Stroke Processing: Summary

- stroke artifact filtering
  - artifacts contained in strokes computed from polygonal meshes
  - artifact removal by local mesh and stroke processing
  - artifact removal through multiresolution curve analysis
- stroke parametrization
  - necessary for smooth animation
  - two coherence schemes: 2D and 3D
- stroke pipelines

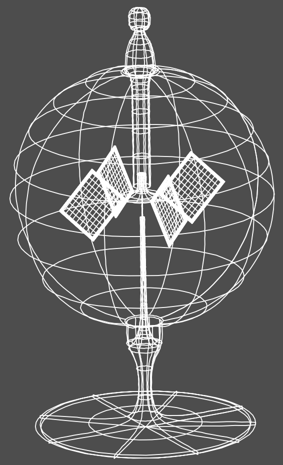
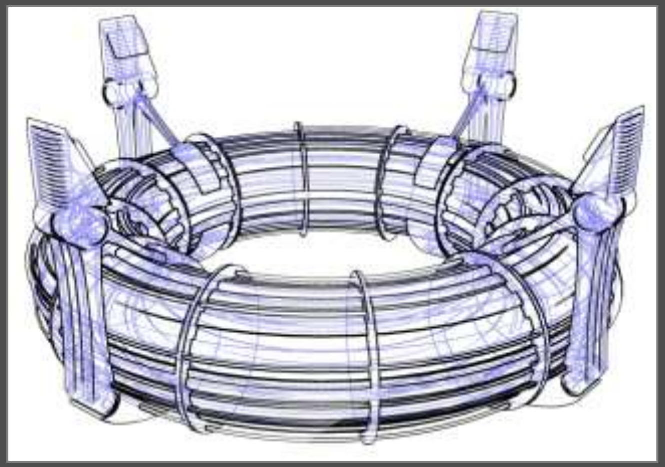
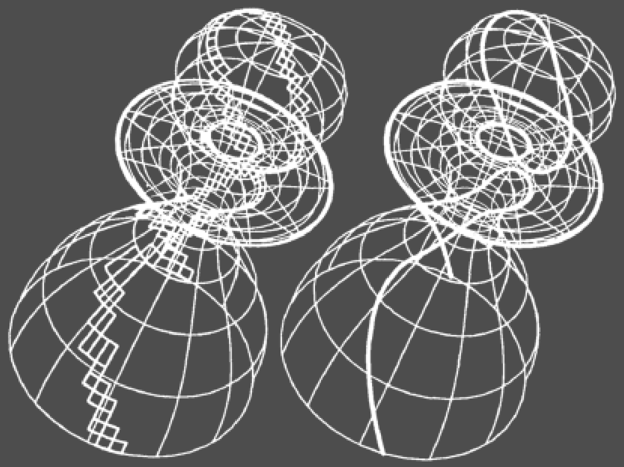






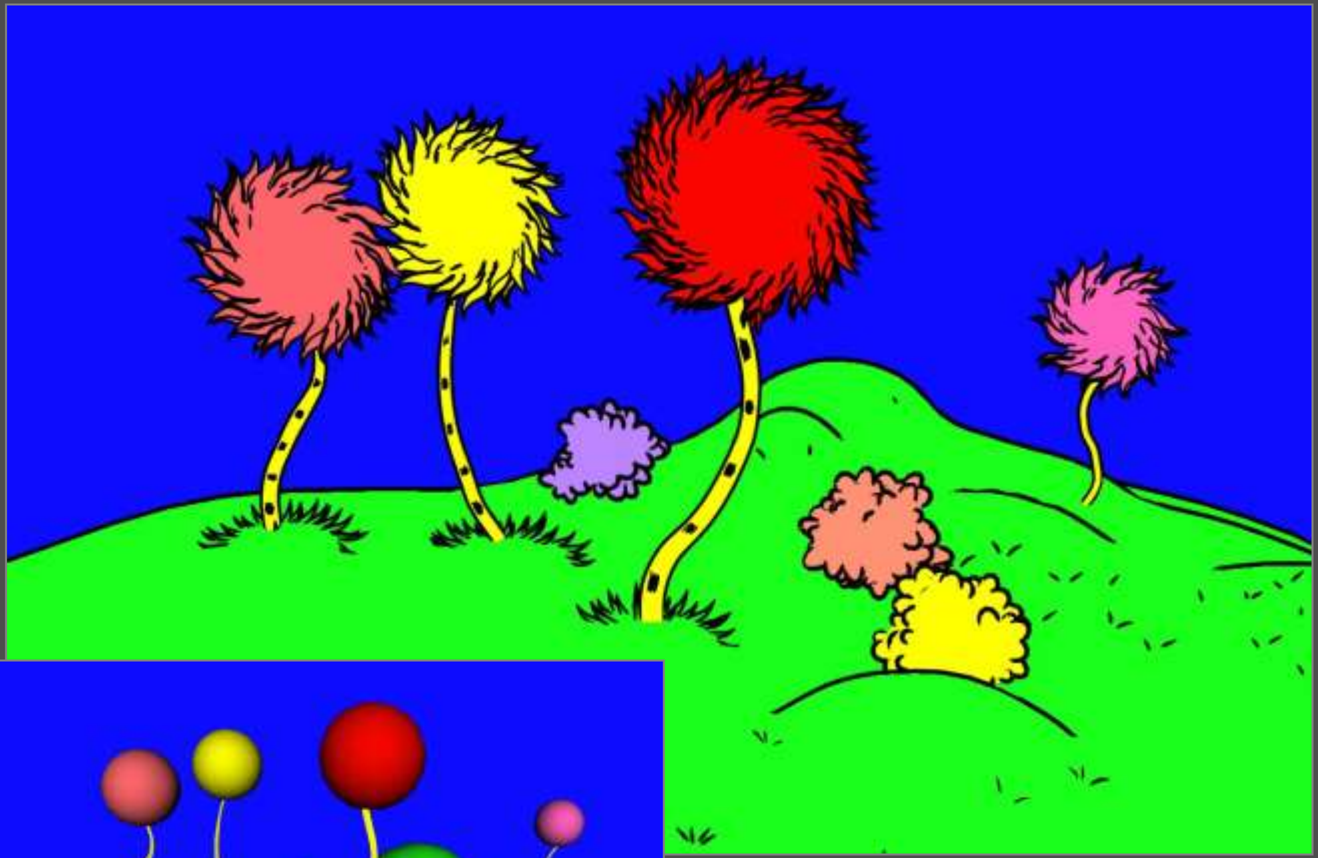


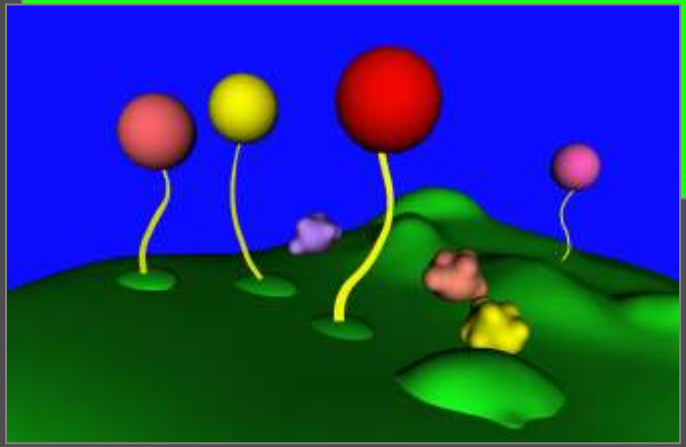
Miscellaneous Techniques

Graftals: Kowalski et al. (1999), Markosian et al. (2000), Kaplan et al. (2000)

- **graftal:** graphic element that sticks on an object's surface to suggest complex geometry
- placed with constant screen-space density based on a difference image
- different rendering styles depending on, e.g., orientation
- frame-coherence: graftals live for several frames and are smoothly blended in and out





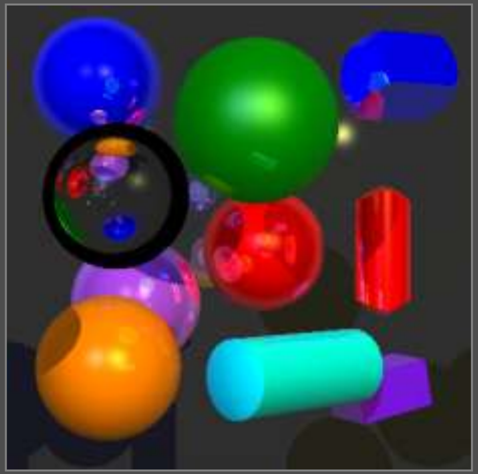
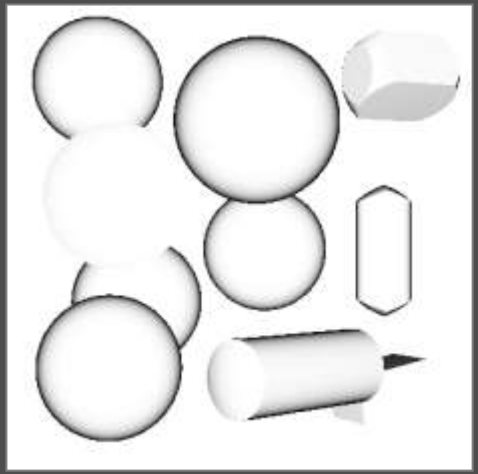
Kowalski et al. (1999)

Graftals: Video Examples

Kowalski et al. (1999)

Silhouette Raytracing

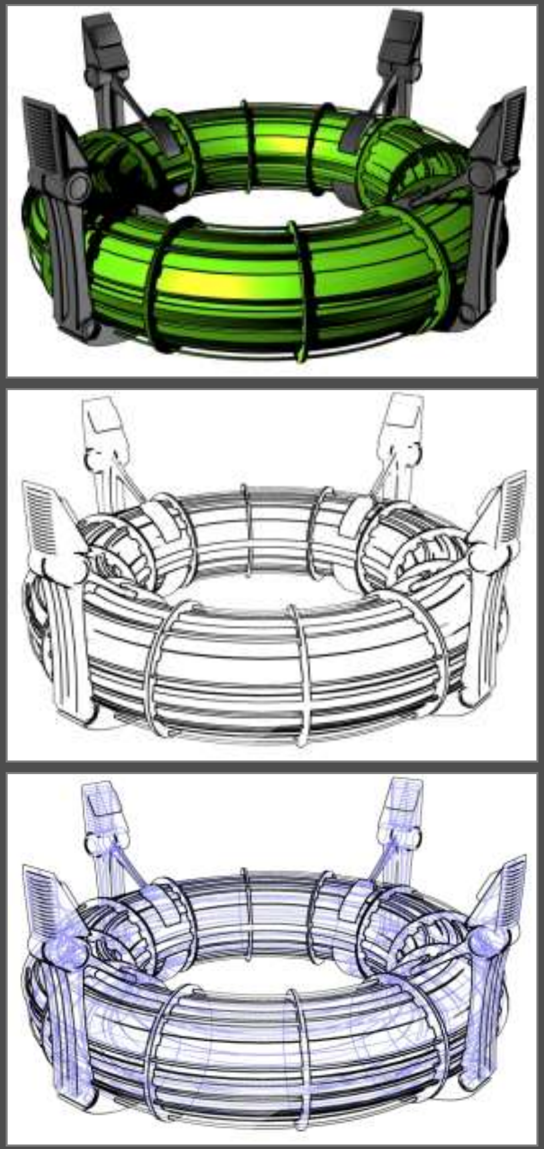
- trivial approach: dot product of normal vector and view direction
  - super-sampling as well as reflections and refractions possible
  - silhouette width is shape-dependent → not limited and not constant
  - problems with detecting silhouettes at discontinuities

 →  → 

- indirect approach
  - computation of normal buffer and z-buffer using regular raytracing
  - application of image-space techniques to the result

Silhouette Raytracing

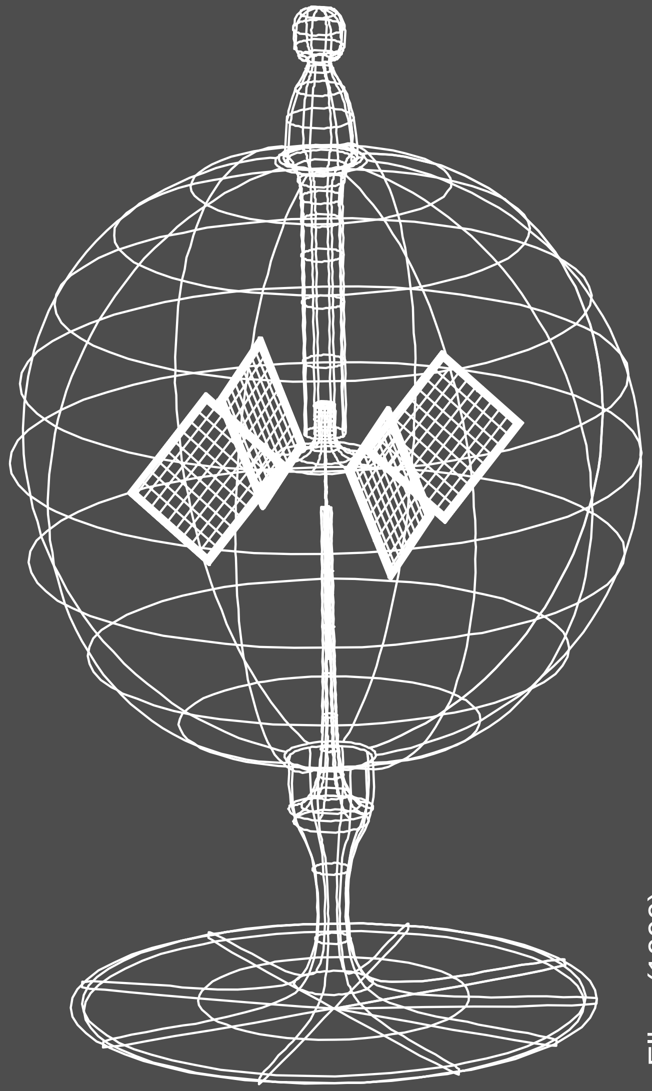
- local image-space approach
  - use super-sampling or distributed raytracing
  - per ray shoot several “silhouette rays” covering a pixel
  - compute various G-buffer data (z, normal, object ID, …)
  - derive silhouette based on thresholds
  - → a lot of double tracing ⇒ very slow
- parallel raytracing
  - distribute small rectangular tiles to individual clients
  - each does a local image-space technique
  - → overlapping tiles necessary
  - avoids unnecessary ray tests
  - some pixels are traced twice or four times



© Oliver Heyme

Sparse Line Drawings from Free-Form Surfaces

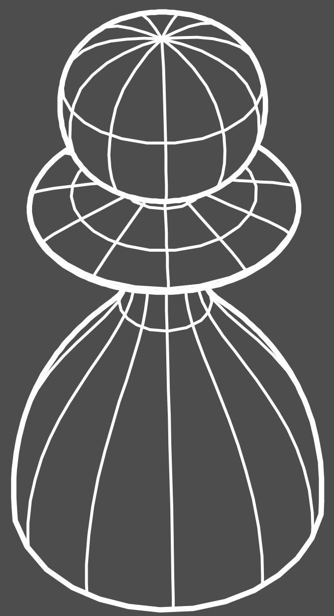
- usually depiction of free-form surfaces as line drawings using iso-parametric curves
- iso-parametric curves can be pre-computed → follow parametric directions
- feature curve detection
  - C^1 discontinuities on the patches
  - borders of non-connected patches
- hidden line removal
  - sometimes not at all
  - analytic techniques

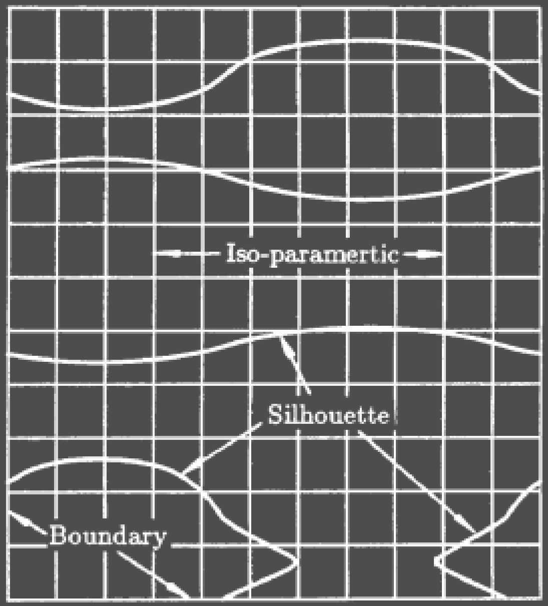


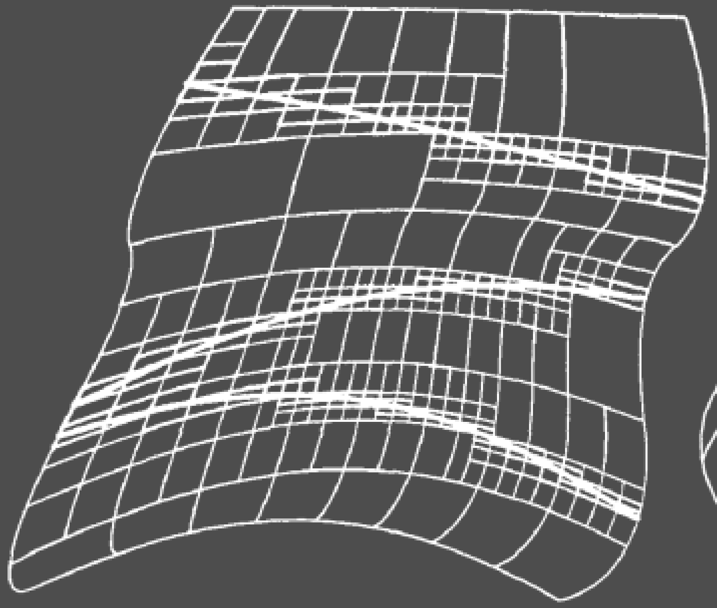
Elber (1999)

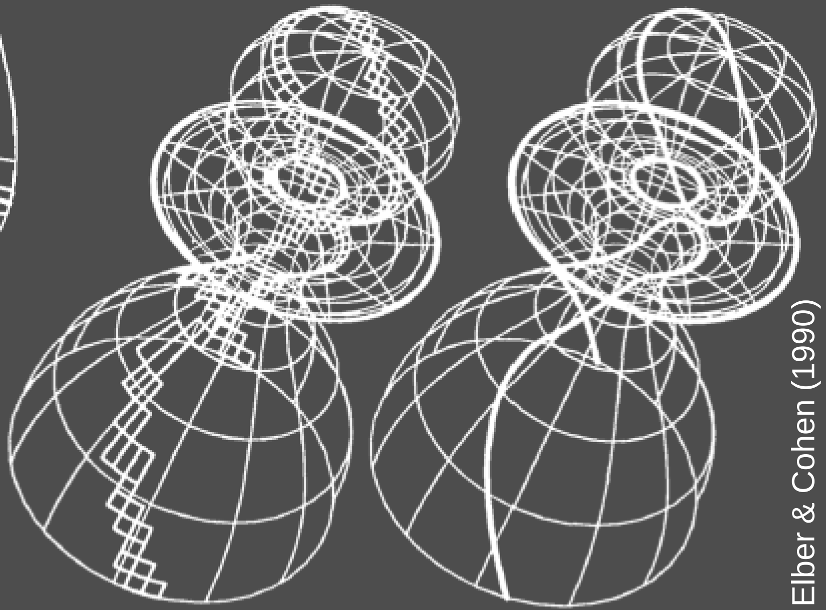
Adaptive Free-Form Surface Silhouette Extraction

- Elber & Cohen (1990): for B-spline surfaces
  - recursive surface subdivision process, up to a given ε tolerance
  - in each step check if patch may contain silhouettes, subdivide if it does
  - connect the resulting small patches to piecewise linear silhouettes
  - (static) feature strokes by detecting C^1 discontinuities





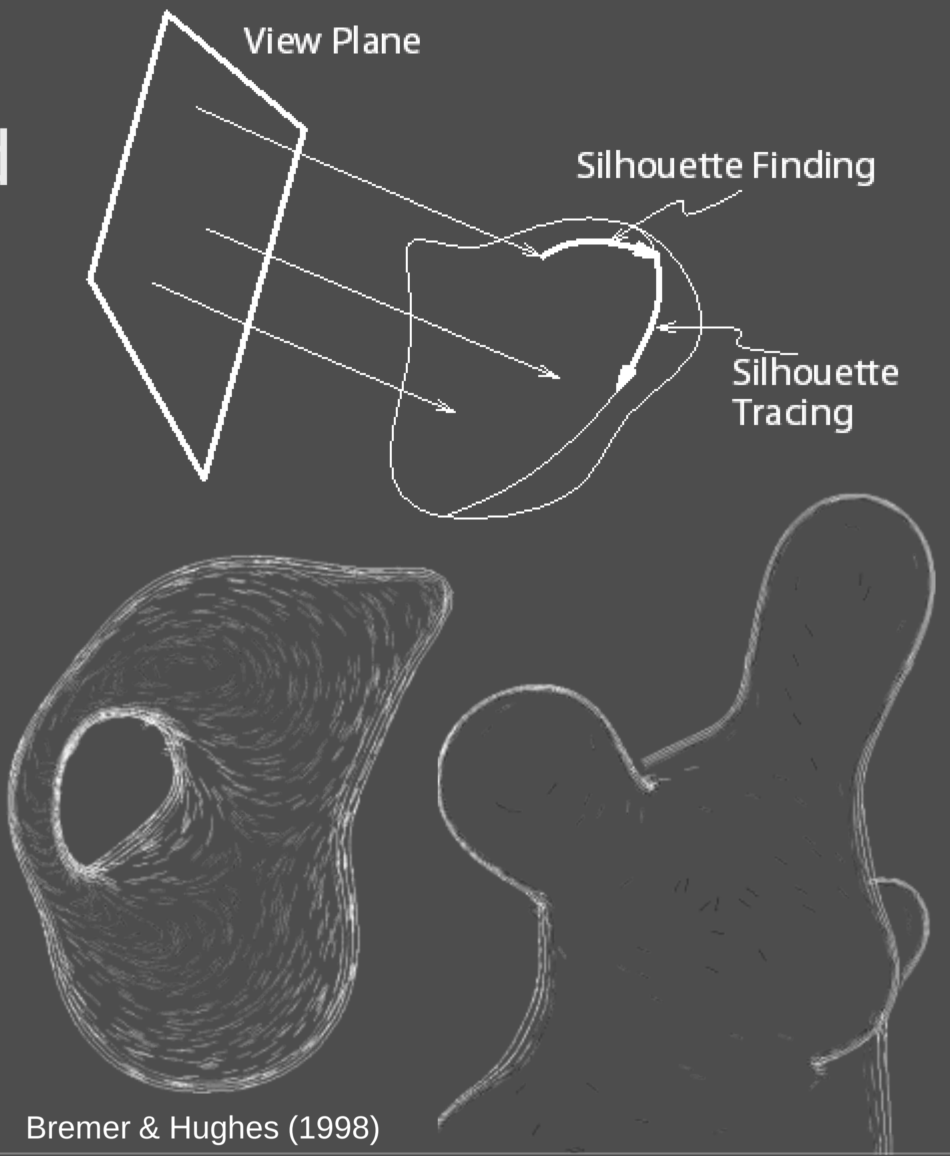




Elber & Cohen (1990)

Silhouettes from Implicit Surfaces: Bremer & Hughes (1998)

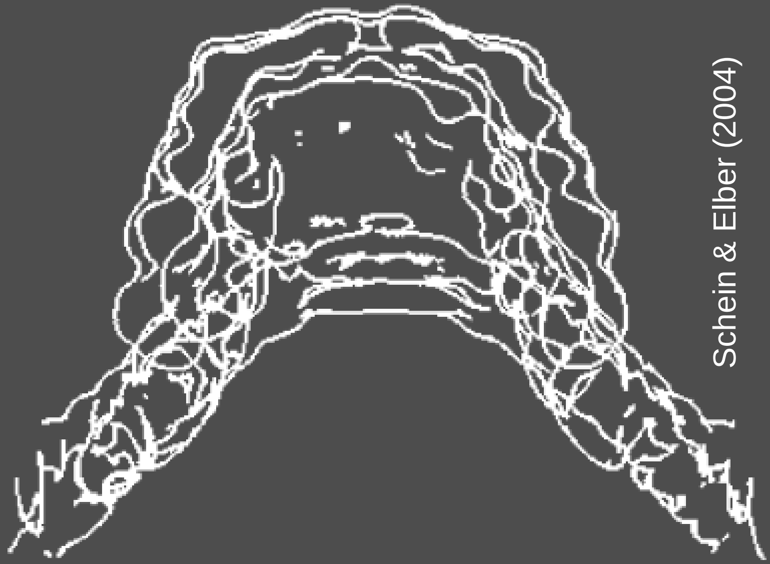
- trivial: indirect rendering using meshes
- direct technique: approximate & randomized
  - basic algorithm:
    1. locate surface point through surface-ray test
    2. trace along surface to locate silhouette point
    3. trace the silhouette iteratively
  - result: piecewise linear silhouette approximation
  - re-use of silhouettes found in previous frames
  - hatching to indicate curvature
  - uses only few parameters of the implicit surface (function value, gradient, and 2nd derivatives)
  - only for orthographic projection, not analytically accurate lines



Bremer & Hughes (1998)

Silhouettes from Volumetric Data

- trivial: indirect rendering using polygonal meshes
- direct technique: Schein & Elber (2004)
  - voxel data set as control points for smooth 3D function
  - ⇒ no discontinuities and no open surfaces
  - ⇒ definition for smooth silhouettes can be applied
  - silhouettes computed for user-defined iso-surface
  - look-up table for viewing directions used for speed-up
- lengthy preprocessing, high memory requirements
- only orthographic projection



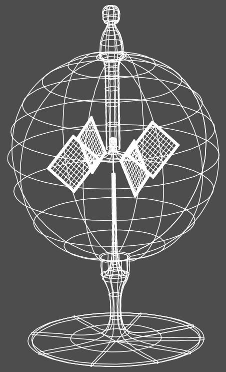
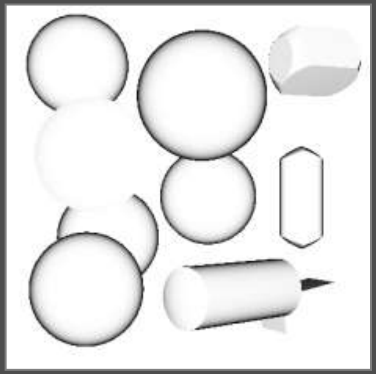
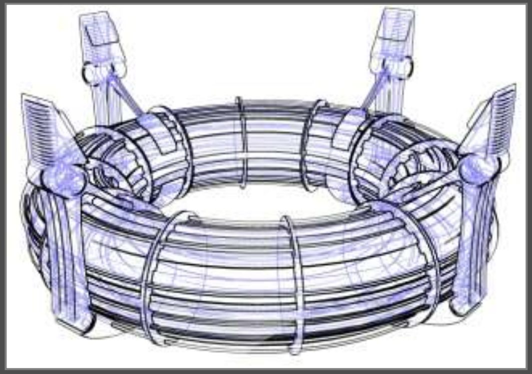
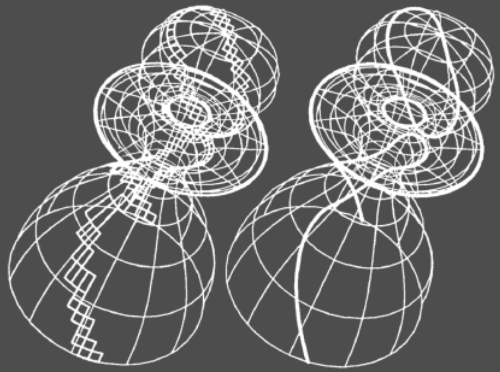
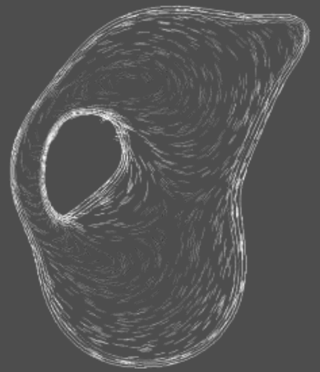
Schein & Elber (2004)

Silhouettes from Volumetric Data

- direct technique: Nagy & Klein (2004)
  - rendering of voxel data slice by slice from front to back
  - potential silhouette pixels determined in each slide using the alpha channel
  - silhouette pixel propagation from slice to slice
- fast and interactive technique
- silhouettes only extracted as pixels

Nagy & Klein (2004)

Miscellaneous Techniques: Summary

- graftals as special form of sparse line primitive
- silhouette and feature line extraction also possible using raytracing
  - best results with methods based on image-space silhouette extraction
- silhouettes from free-form surfaces
  - approximate silhouettes through adaptive subdivision
- silhouettes from non-polygonal model representations
  - indirect method (via mesh extraction) and adapted direct methods

Silhouette and Feature Stroke Detection: Summary

- three main classes of silhouette detection algorithms
  - image-space: image processing (filtering) of generated G-buffers
  - hybrid: manipulations in object-space, silhouettes rendered into image-space
  - object-space: computation for silhouette detection entirely in object-space
- differences of these methods
  - visibility culling integrated in detection process or not
  - type of result (pixel matrix vs. vector graphic)
  - precision of the result (pixel precision to sub-polygon precision)
  - detection of all silhouette edges or only most of them
  - speed of the method and run-time complexity
  - animation easily possible or not

⇒ best algorithm depends on application

Silhouette and Feature Stroke Detection: Summary

- feature strokes
  - surface properties, intersections, and other strokes
  - significant strokes necessary for shape recognition
  - feature stroke detection not necessary for image-space techniques
  - necessary for some hybrid and all object-space methods
- hidden line removal
  - using image-space, object-space, and hybrid techniques
  - image-space: numerically instable and too restricted for stylized strokes
  - object-space: exact solution but typically too slow
  - hybrid HLR: trading accuracy for speed (using ID or z-buffer data)
  - silhouettes and feature strokes are treated similarly
  - stylization easily possible based on visibility information

Silhouette and Feature Stroke Detection: Summary

- stroke processing
  - silhouettes from polygonal meshes: triangulation artifacts
  - artifact removal using local filtering or multiresolution analysis
  - stroke parametrization for coherently animated stylized strokes
  - stroke processing and stylization in pipelines
- miscellaneous techniques & silhouettes from non-polygonal data
  - graftals
  - silhouette raytracing
  - silhouettes from free-form surfaces
  - silhouettes from implicit surfaces
  - silhouettes from volumetric representations

  → indirect by extracting polygonal meshes or using adapted approaches