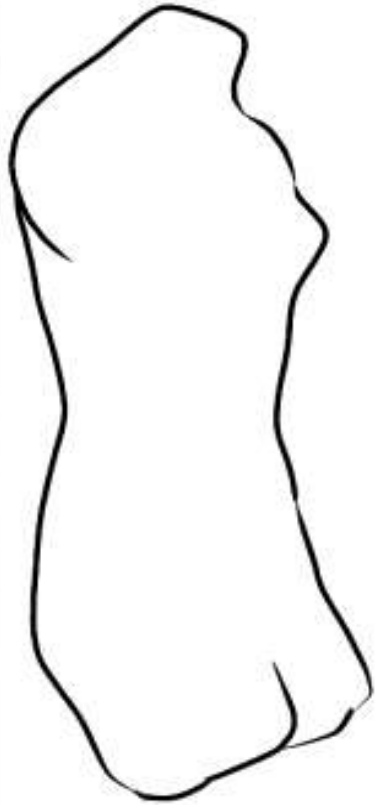
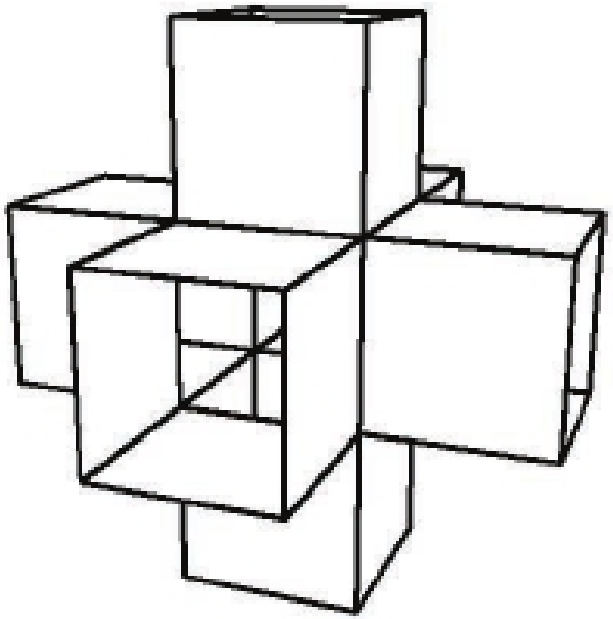
Silhouette extraction



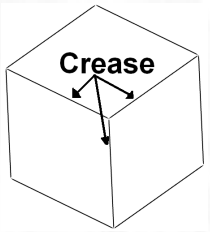


Overview

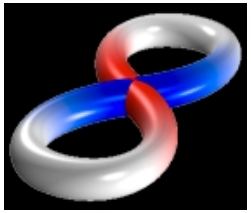
- Introduction
- Image space algorithms
- Hybrid algorithms
- Object space algorithms
- Summary

What are sparse lines drawings?

- Feature lines:



  - Creases
  - Borders

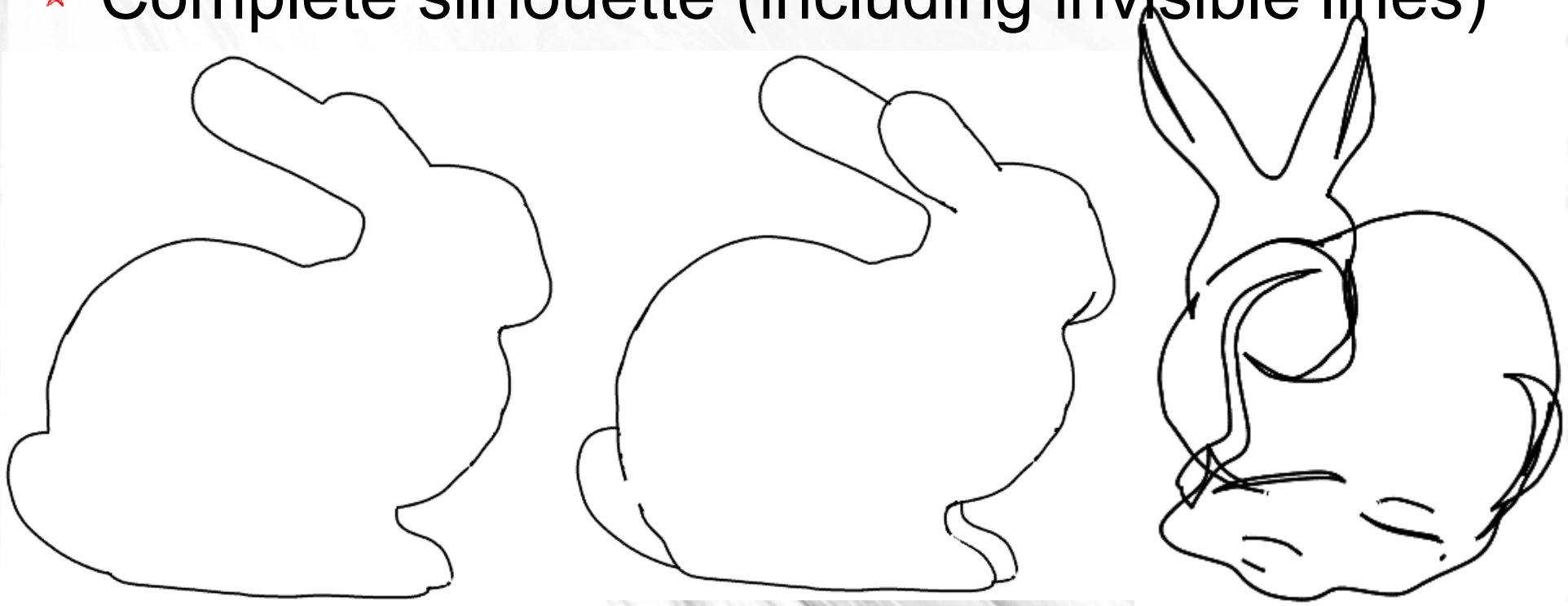


  - Self-intersections

- Silhouette/contour

What is a silhouette?

- Contour
- Internal silhouettes
- Complete silhouette (including invisible lines)



Mathematical definition

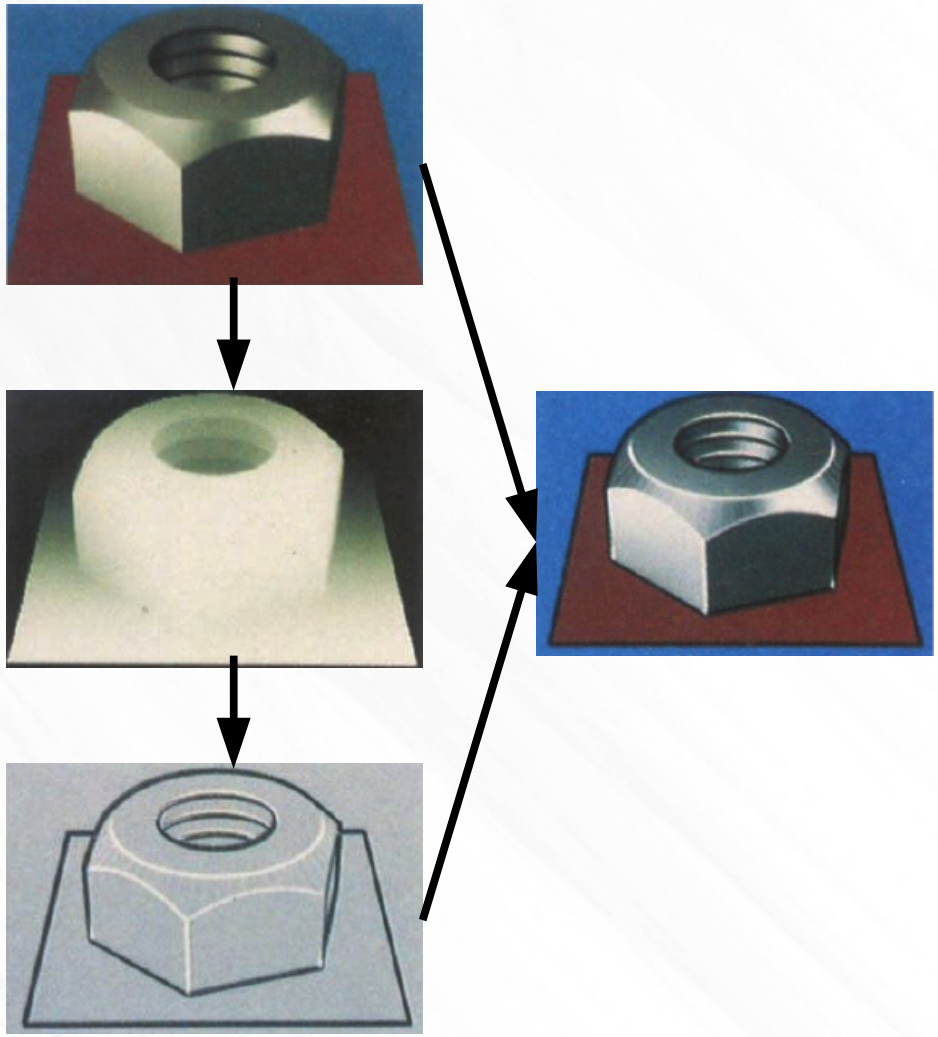
- The silhouette S of a free-form object = points on the object's surface where the surface normal is perpendicular to the vector from the viewpoint
- $S = \{P : 0 = n_i \cdot (p_i - c)\}$, with normal n_i, vertex P, position p_i, center of projection c. In a orthographic projection $(p_i - c)$ is exchanged with the view direction vector v
- Can't be applied to polygonal objects (Normals per polygon)
- Silhouette edges of a polygonal model: Edges that share a front- and a back-facing polygon

How to extract the silhouette?

- A lot of different algorithms
- Three main categories:
  - Image space algorithms
  - Hybrid algorithms
  - Object space algorithms

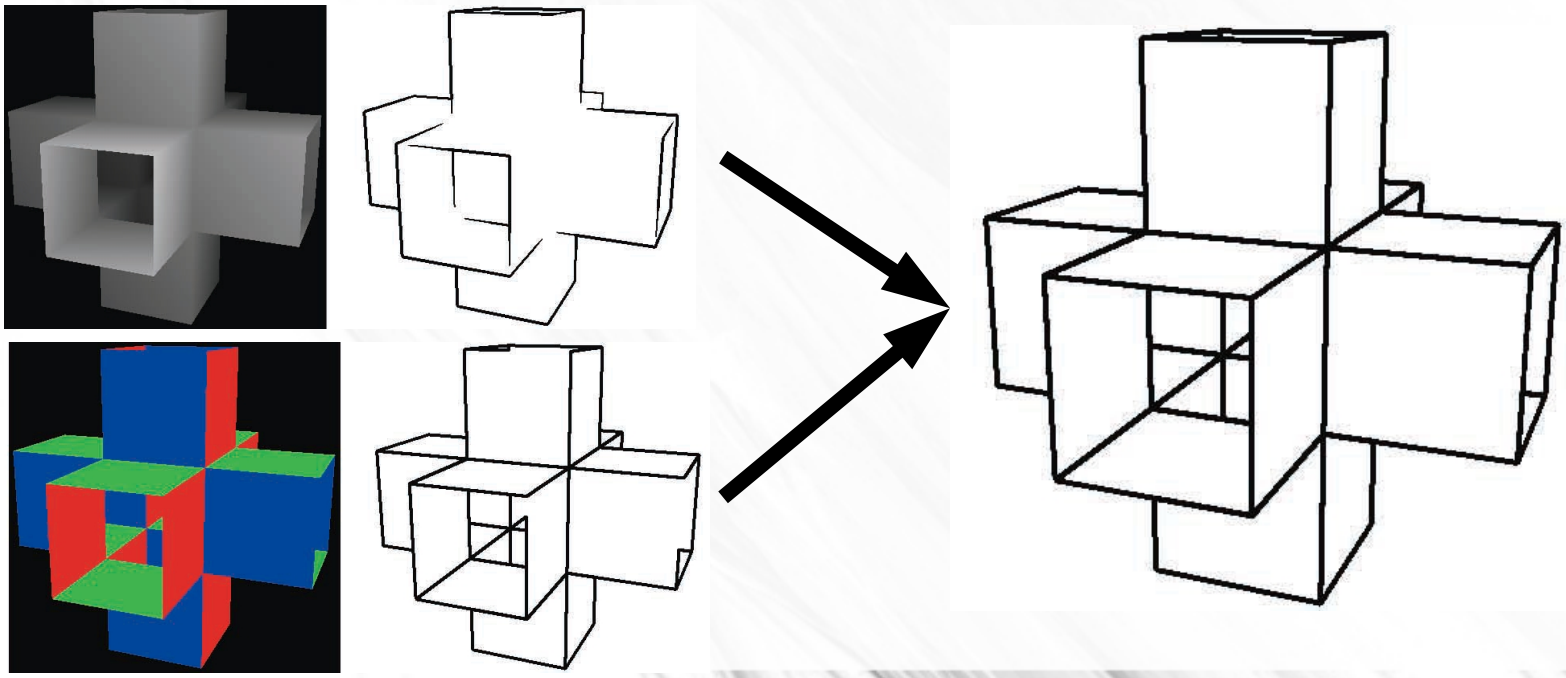
In the beginning...

- Saito and Takahashi 1990
- Z-buffer
- Edge detection



Using z-buffer and normal buffer

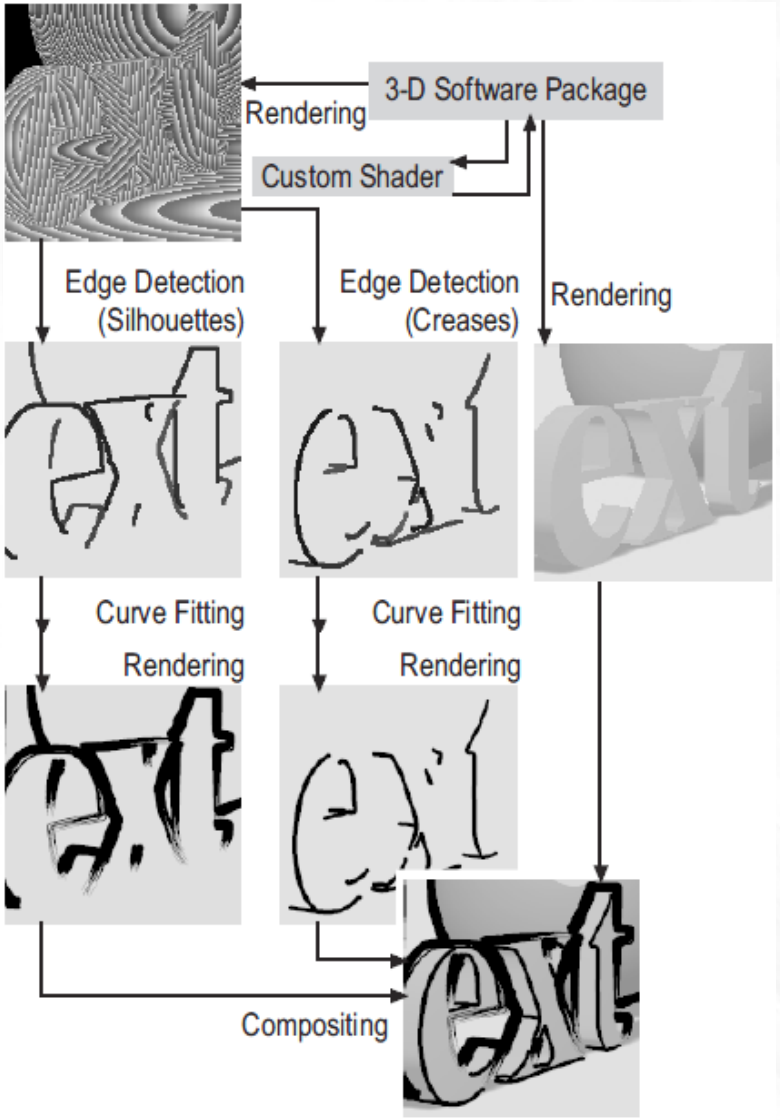
- Extract edges from z-buffer
- Extract edges from normal buffer
- Combine



Hardware acceleration

- RGBA(*nxworld*, *nyworld*, *nzworld*, *deptheye*) texture
- Vertex shader to populate color channels
- Very fast
- No distinction between creases and silhouettes

Using Bezier curves

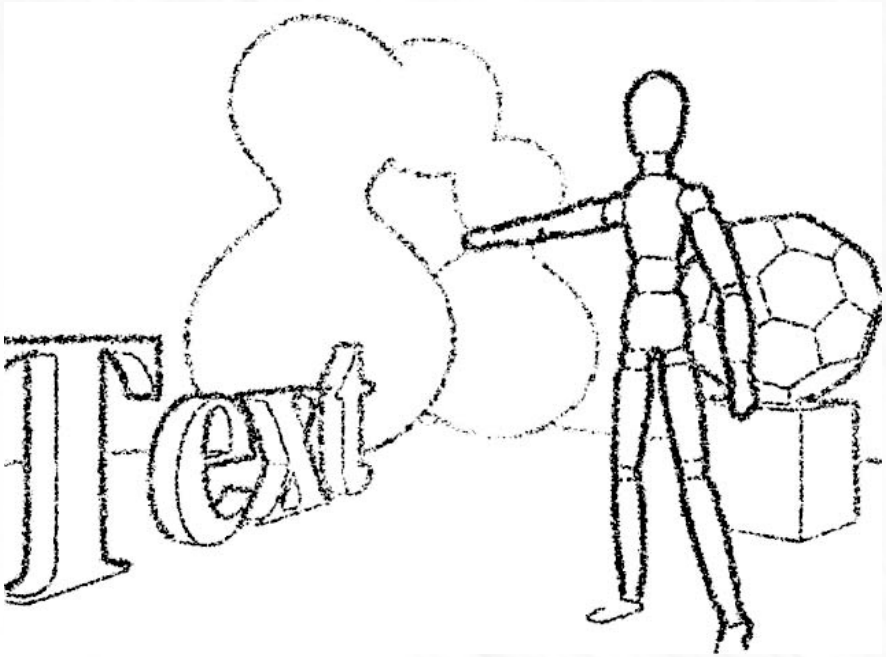


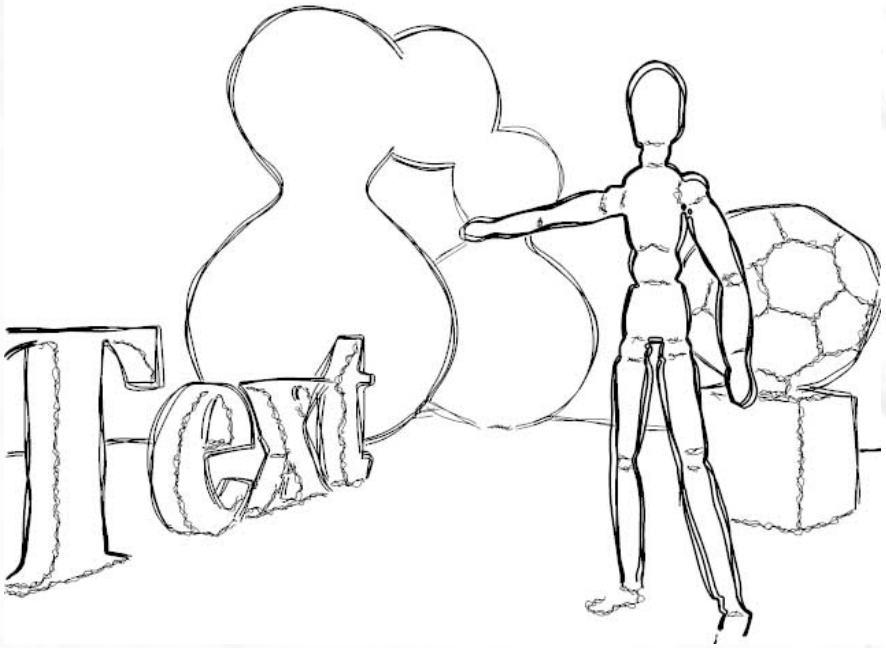
- Extracts silhouettes
- Extracts creases
- Fitting curves to edge points

- Can use stylized lines
- Very slow

Using Bezier curves

- Examples





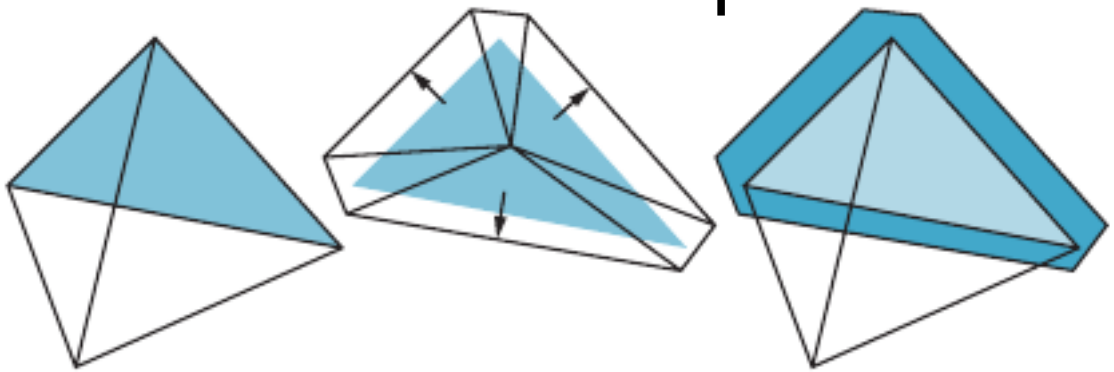
Advantages/disadvantages

- Most image space algorithms are fast (real time)
- No aliasing

- No distinct borders
- Little control over resulting lines
- No good support for stylized lines, because of pixel matrix representation
- Fine details may be hidden

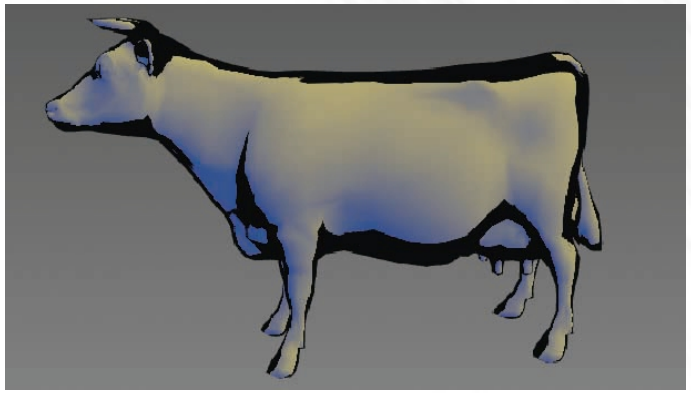
Based on z-buffer rendering

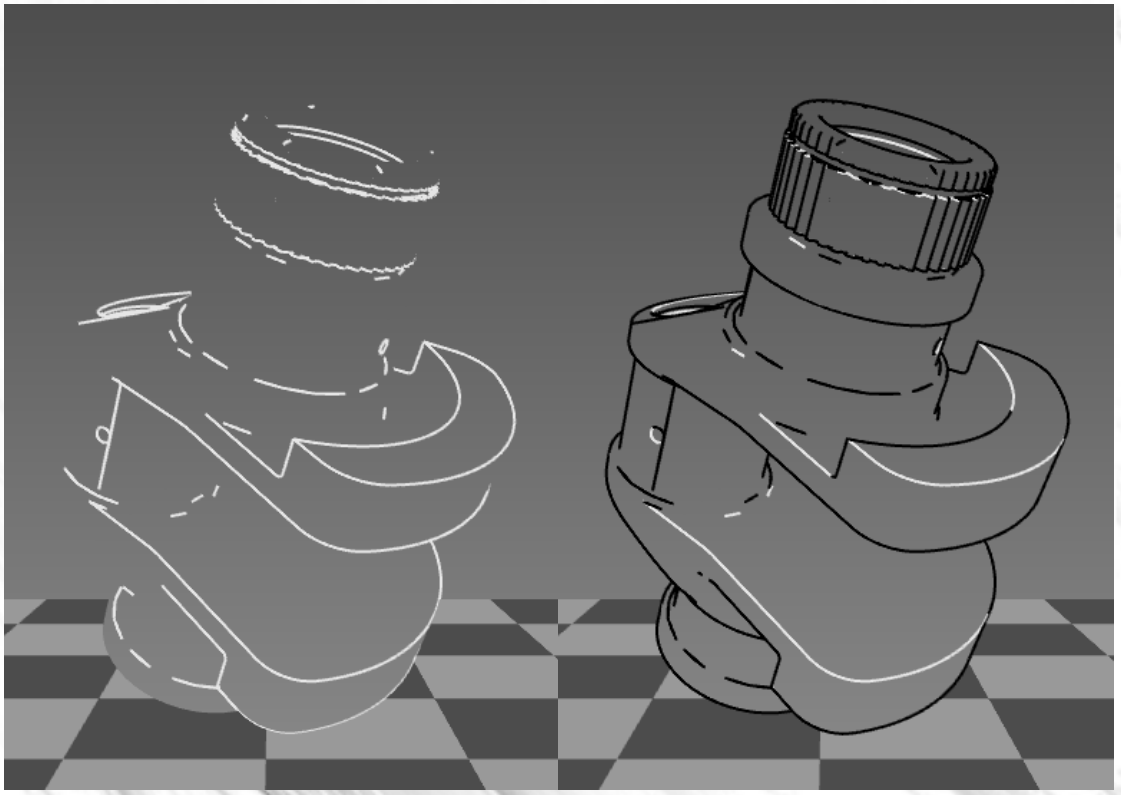
- Separate back and front facing polygons by culling
- Enlarge back-facing polygons to achieve wide silhouette lines
- Silhouette color can be selected
- Line thickness depends on translation distance



Hardware accelerated rendering

- Uses stencil buffer as mask for second pass
- Creases in different color than silhouettes
- Faster



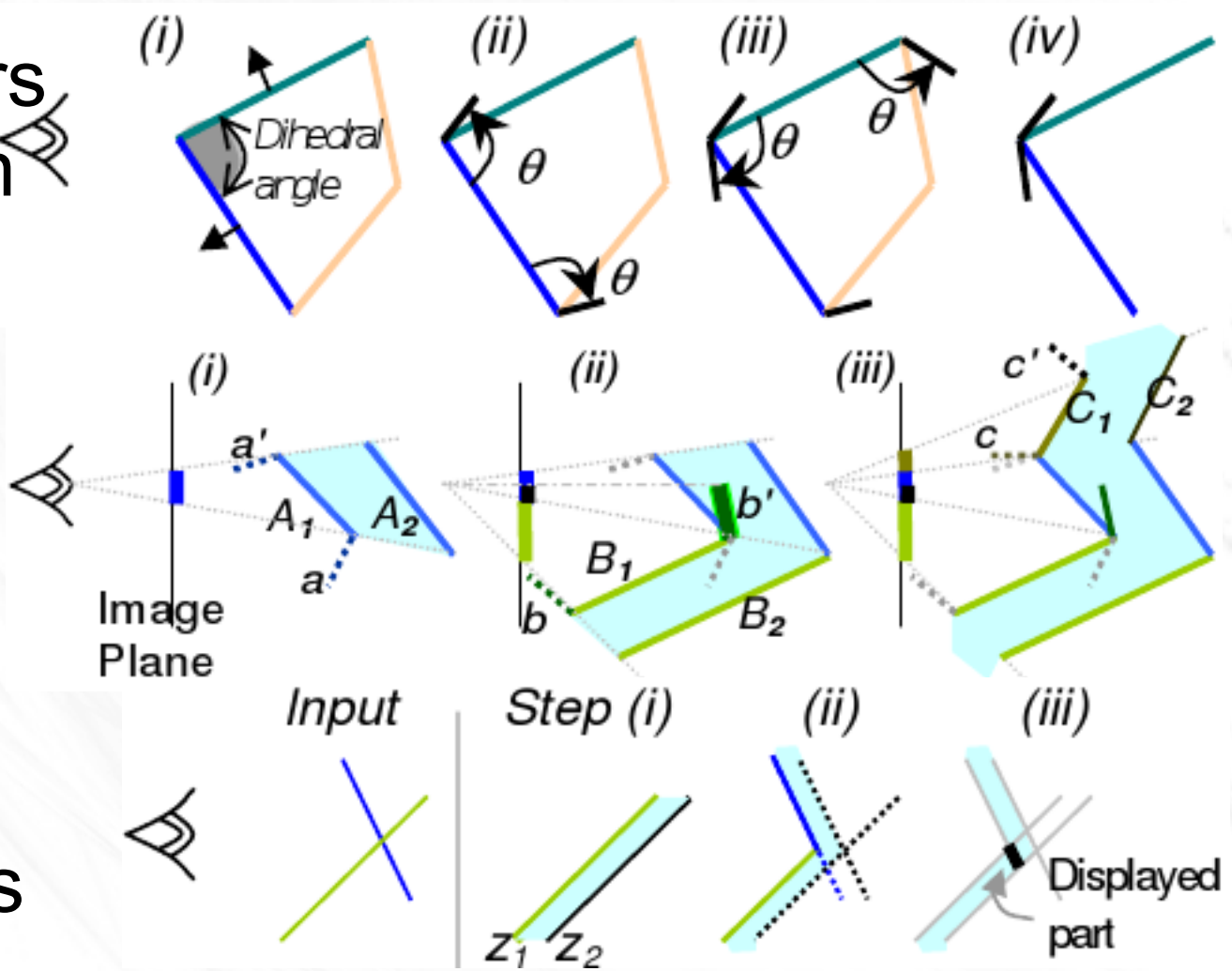


Marcel Koster

Silhouette Extraction

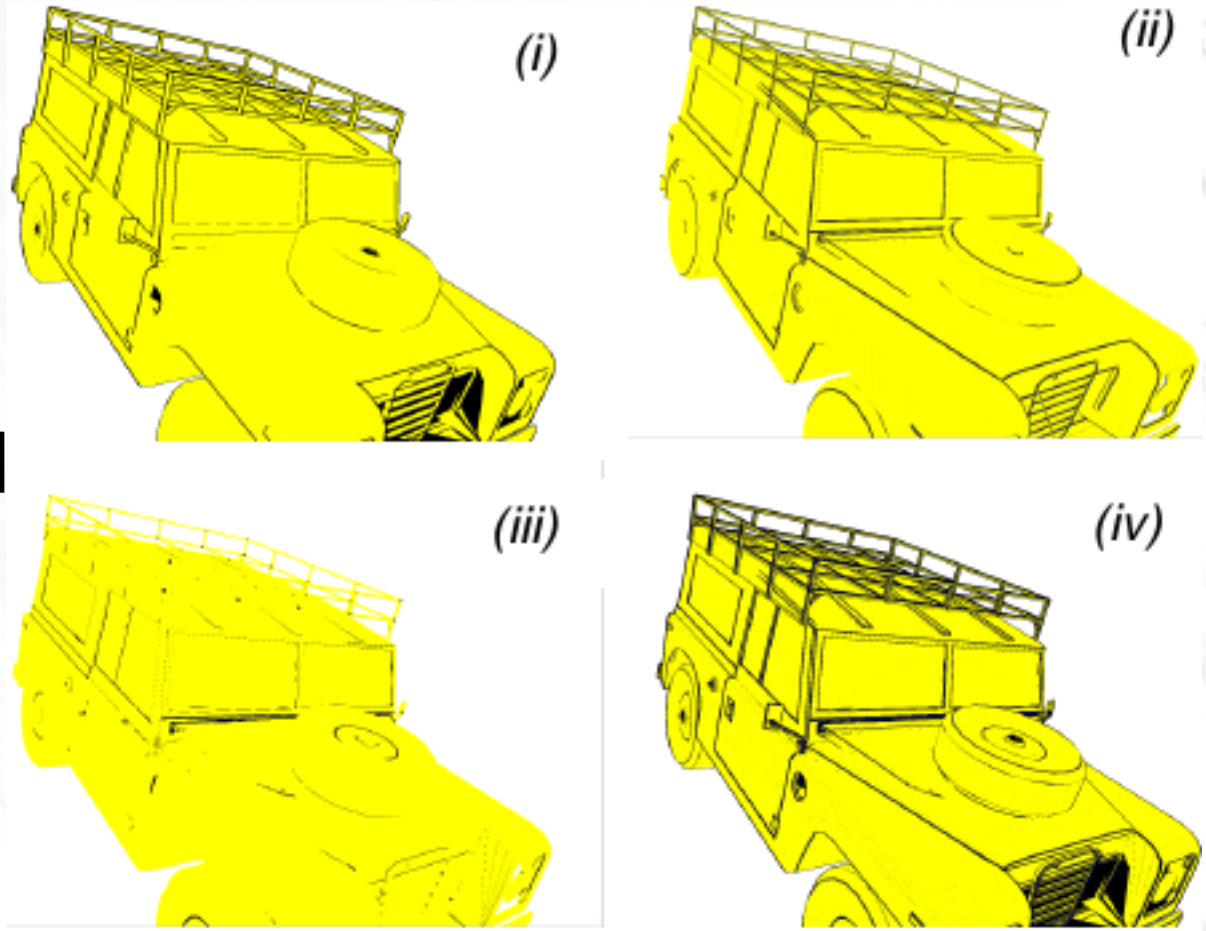
One-pass hardware implementation

- Adds borders around each triangle
- Silhouettes
- Ridges
- Valleys
- Intersections



One-pass hardware implementation

- Example:
- (i) Silhouettes
- (ii) Ridges
- (iii) Valleys
- (iv) Combined



Advantages/disadvantages

- Higher degree of control over outcome
- Images are generally more stylistic
- Distinct borders
- Also quite fast (interactive to real time)

- Aliasing can be present
- No further stylization possible, because of pixel matrix representation

Object space

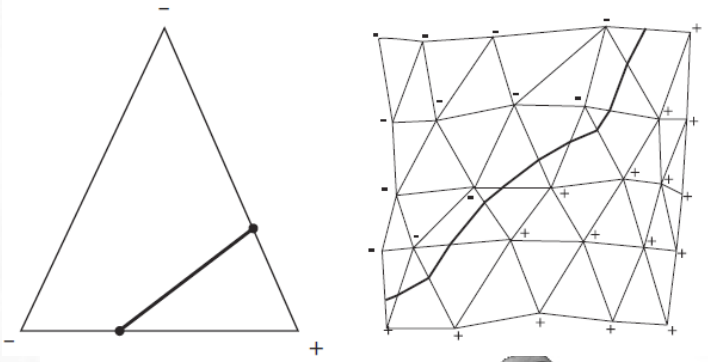
- Silhouette edge detection
- Line visibility determination

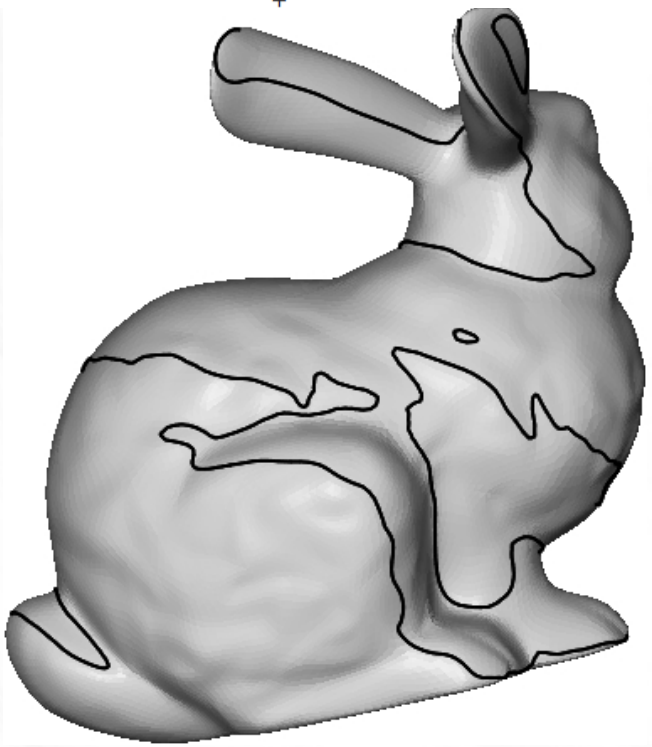
Silhouette edge detection: Trivial method

- Classify all polygons as front or back facing
- Select edges that share 1 front and 1 back facing polygon
- Quite fast (real-time)
- Aliasing problems
- Speed up by edge buffer or vertex processing hardware

Silhouette edge detection: Subpolygon routines

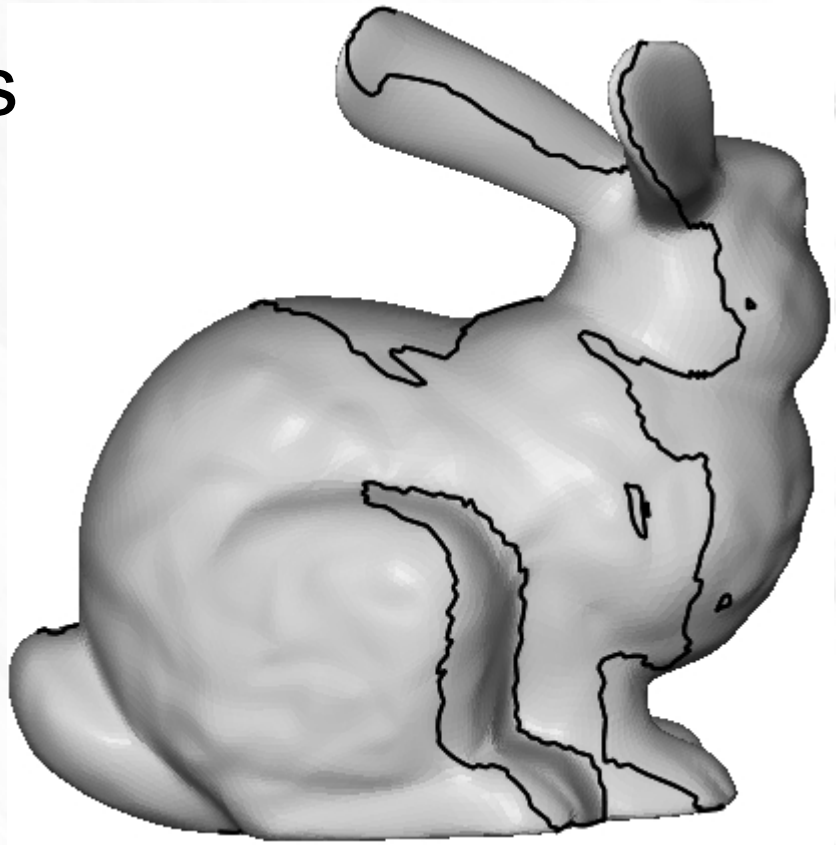
- Recompute normals
- Compute dot product w.r.t. viewing direction
- Linear interpolation on edges where sign is different
- Somewhat slower (at least interactive frame rate)
- No aliasing





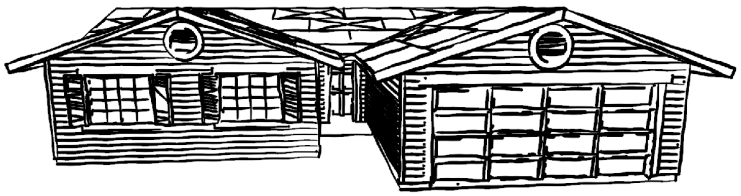
Silhouette edge detection: Precomputation methods

- Many different approaches
- Reduces nr. of triangles or edges checked at runtime
- Efficient data structure
- Interactive frame rates
- Animation inefficient



Silhouette edge detection: Stochastic method

- Only few edges are silhouette edges ($O(\sqrt{n})$)
- Random selection of small fraction of edges
- Spatial coherence



- Recursively test adjacent edges
- Fast run-time execution (real-time)
- No guarantee that all edges are found

Line visibility determination: Image space

- Uses z-buffer
- Render edges, z-buffer removes hidden lines

- Very fast and trivial
- Pixel accuracy is limited
- Lines with style variations might be partially occluded

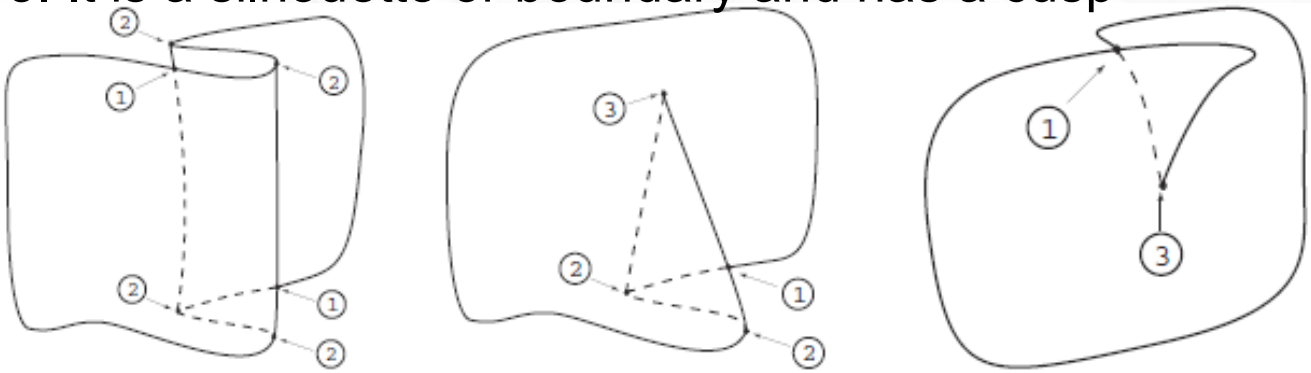
Line visibility determination: Object space

- Gives highly precise visibility information
- Usually slower than image space approaches
- Lines can be stylized

- Many different approaches
- Mainly used: Quantitative invisibility (QI)

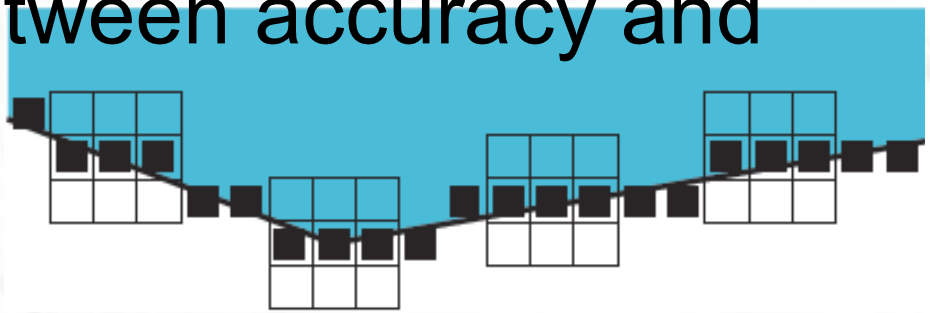
Line visibility determination: Object space

- QI = total number of faces between a point and the viewer
- Three situations where the visibility of a surface curve can change:
  - 1. It passes under a silhouette, boundary or crease in the image plane
  - 2. It intersects a silhouette, crease, or self-intersection curve on the surface
  - 3. It is a silhouette or boundary and has a cusp



Line visibility determination: Hybrid

- Faster and less accurate than object space approaches
- Generates set of visible silhouette lines, which can be stylized
- Method based on z-buffer
- Check 8-pixel neighborhood and scan every *n*th pixel (trade off between accuracy and speed)



Advantages/disadvantages

- Generates analytic representation of the silhouette
- Thus supports more stylization of the lines
- Lines are more precise
- Difficult to implement, because of separate stages
- Slowest method
- Not very well suited for large models and animations

Summary

- All approaches have their own advantages and disadvantages
- Best silhouette detection algorithm depends on application
- Examples:
- Silhouettes in an analytic description → Object space silhouette algorithms with an object space or a hybrid visibility test.
- Least amount of memory → Image space or hybrid algorithm
- Real-time or interactive frame rates → Image space and hybrid algorithms.

Questions?

References

- Tobias Isenberg, Bert Freudenberg, Nick Halper, Stefan Schlechtweg, Thomas Strothotte, "A Developer's Guide to Silhouette Algorithms for Polygonal Models," IEEE Computer Graphics and Applications, vol. 23, no. 4, pp. 28-37, July/Aug. 2003.
- A. Hertzmann, “Introduction to 3D Non-Photorealistic Rendering: Silhouettes and Outlines,” Non-Photorealistic Rendering (Siggraph 99 Course Notes), S. Green, ed., ACM Press, 1999.
- J. Loviscach, “Rendering Artistic Line Drawings Using Off-the-Shelf 3-D Software,” Proc. Eurographics: Short Presentations, I.N. Alvaro and P. Slusallek, eds., Blackwell Publishers, 2002, pp. 125-130.
- J.L. Mitchell, C. Brennan, and D. Card, “Real-Time ImageSpace Outlining for Non-Photorealistic Rendering,” Siggraph 02 Conf. Abstracts and Applications, ACM Press, 2002, p. 239.
- B. Gooch et al., “Interactive Technical Illustration,” Proc. 1999 ACM Symp. Interactive 3D Graphics, ACM Press, 1999, pp. 31-38.
- R. Raskar, “Hardware Support for Non-Photorealistic Rendering,” Proc. 2001 Siggraph/Eurographics Workshop on Graphics Hardware, ACM Press, 2001, pp. 41-46.
- J.D. Northrup and L. Markosian, “Artistic Silhouettes: A Hybrid Approach,” Proc. 1st Int’l Symp. Non-Photorealistic Animation and Rendering, J.-D. Fekete and D.H. Salesin, eds., ACM Press, 2000, pp. 31-37.