

Image Precision Silhouette Edges

by Ramesh Raskar and Michael Cohen
Presented at I3D 1999

Presented by Melanie Coggan

Outline

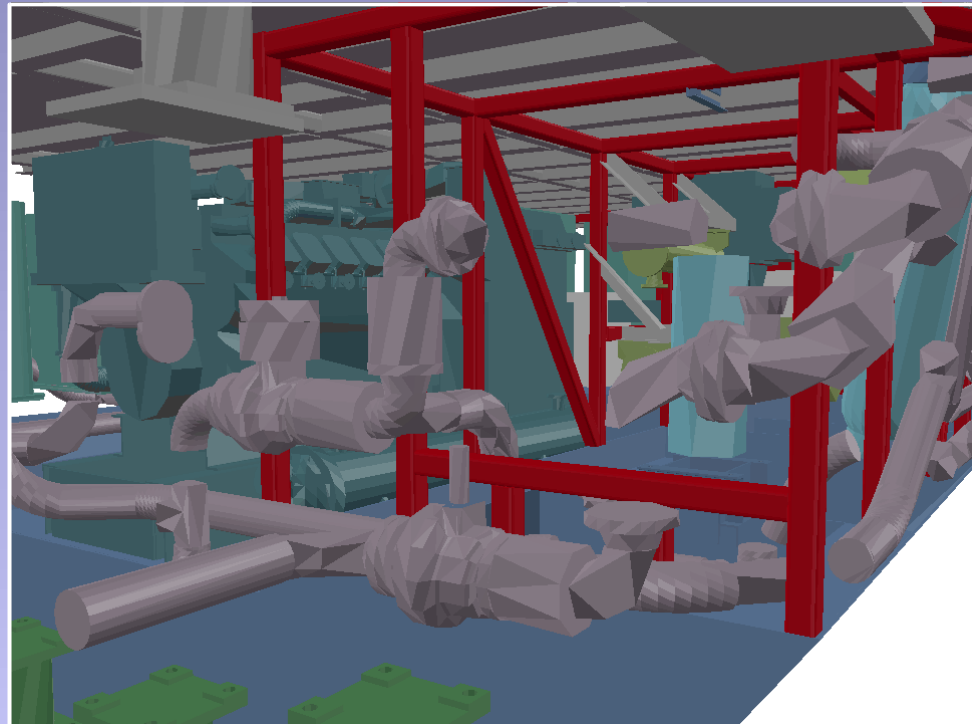
- Motivation
- Previous Work
- Method
- Results
- Conclusions

Outline

- Motivation
- Previous Work
- Method
- Results
- Conclusions

Motivation

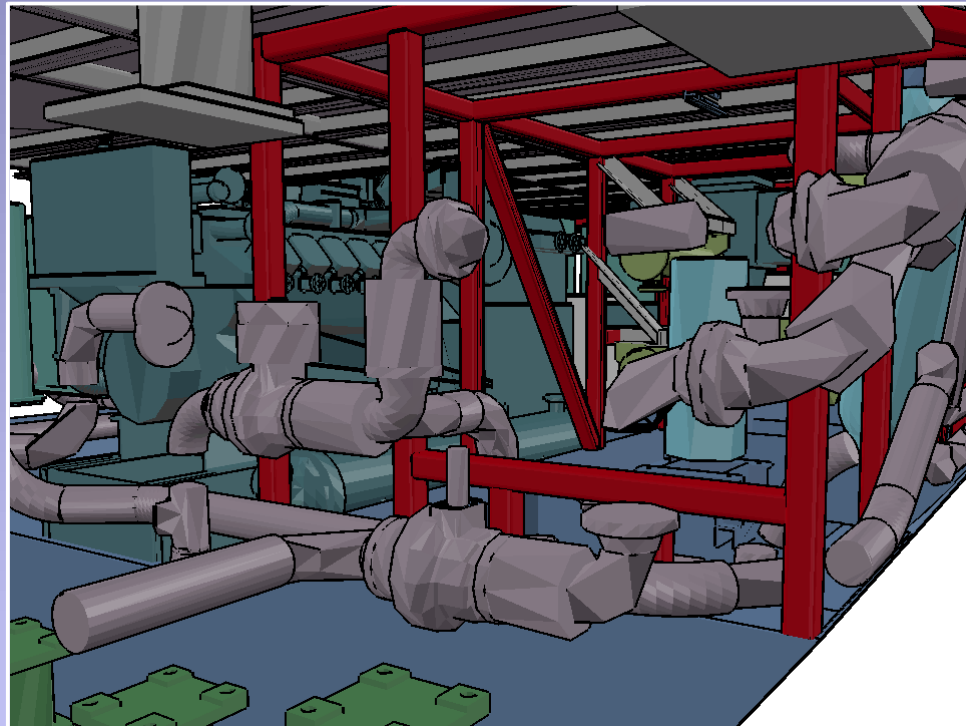
- Conveys more detail
- Useful for technical diagrams



Raskar and Cohen – I3D Slides

Motivation

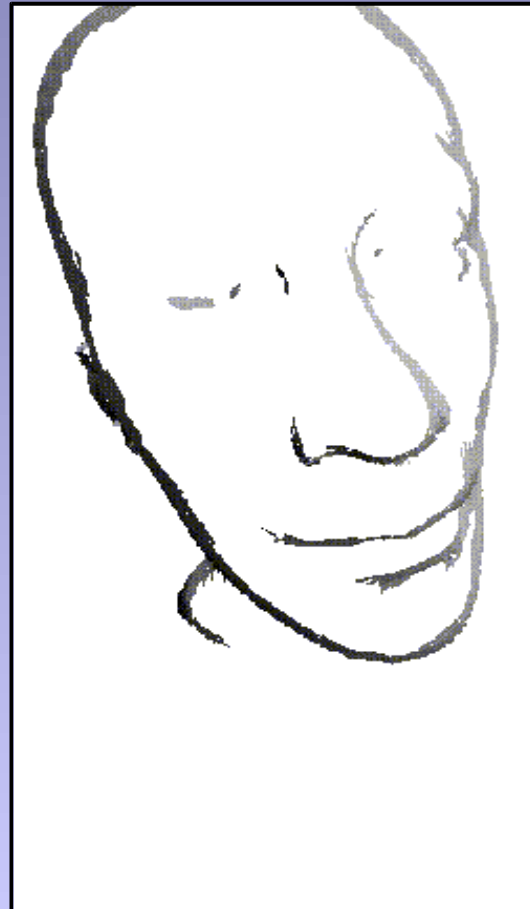
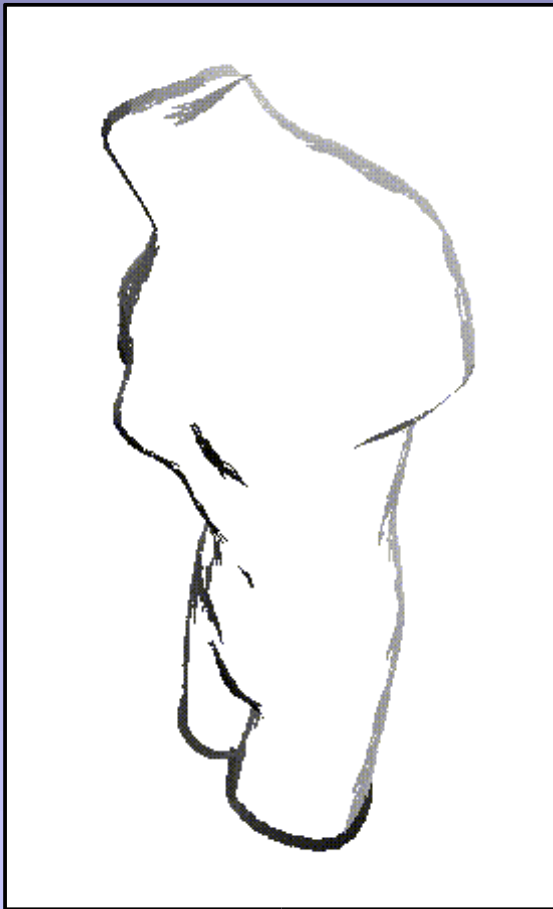
- Conveys more detail
- Useful for technical diagrams



Raskar and Cohen – I3D Slides

Motivation

- Non-Photorealistic Rendering
- Use fewer strokes



Outline

- Motivation
- Previous Work
- Method
- Results
- Conclusions

Previous Work

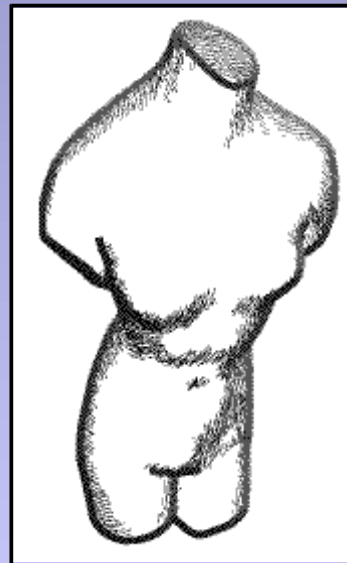
- Sutherland et al. 1974
 - Hidden line removal
- Markosian et al. 1997



Sutherland with Sketchpad - Dmitry Shklyar

Previous Work

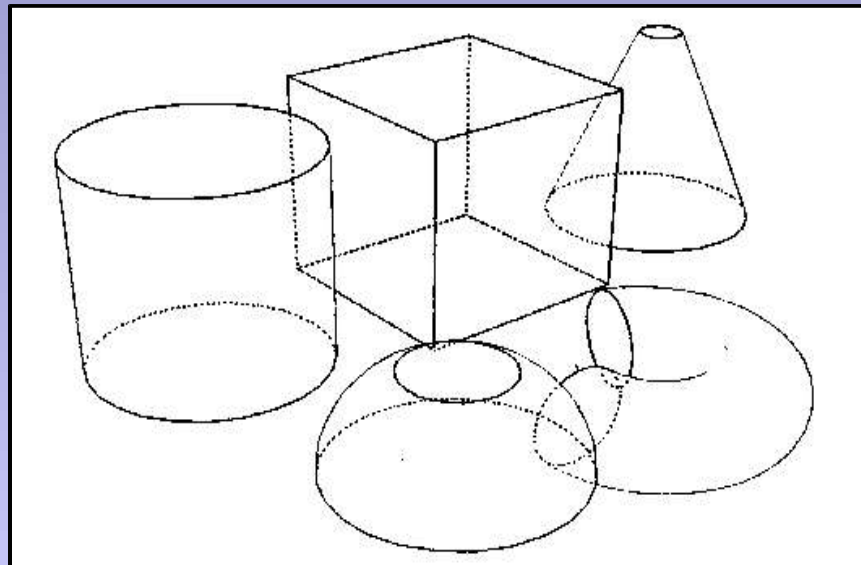
- Sutherland et al. 1974
- Markosian et al. 1997
 - Probabilistic method
 - Scene does not need to be traversed



Shaded Venus – Lee Markosian

Previous Work

- Zeleznik et al. 1996
 - Interface to highlight edges and boundaries
- Rossignac et al. 1992
 - Image precision / Wireframe



Previous Work - Problems

- Adjacency information required
- Pre-processing or batch processes
- Static scenes
- Two types of primitives involved, polygons and edges

Outline

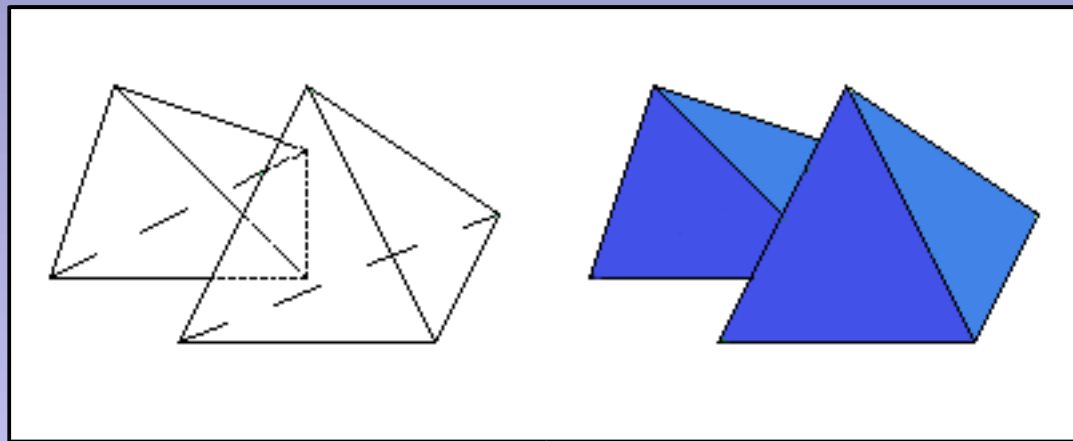
- Motivation
- Previous Work
- Method
- Results
- Conclusions

Goals

- Work with changing scenes
 - no adjacency information
- Dynamic
 - no pre-processing
- Real-time
- Single primitive - polygons

Algorithm

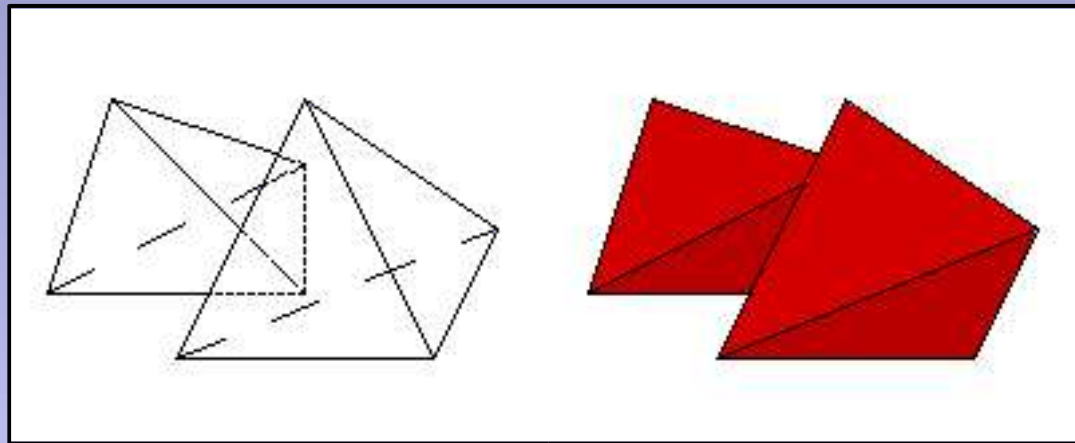
- Use only two sets of polygons
 - P1: Completely visible from viewpoint; front-facing
 - P2: Second layer from viewpoint; back-facing



Original polygons and layer P1

Algorithm

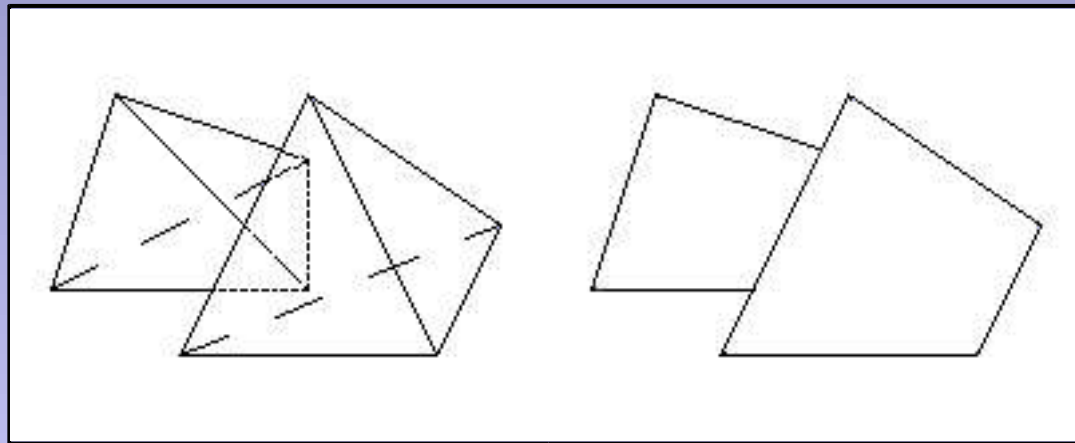
- Use only two sets of polygons
 - P1: Completely visible from viewpoint; front-facing
 - P2: Second layer from viewpoint; back-facing



Original polygons and layer P2

Algorithm

- Use only two sets of polygons
- Compute intersection of P_1 and P_2



Original polygons and silhouette

Implementation - basic

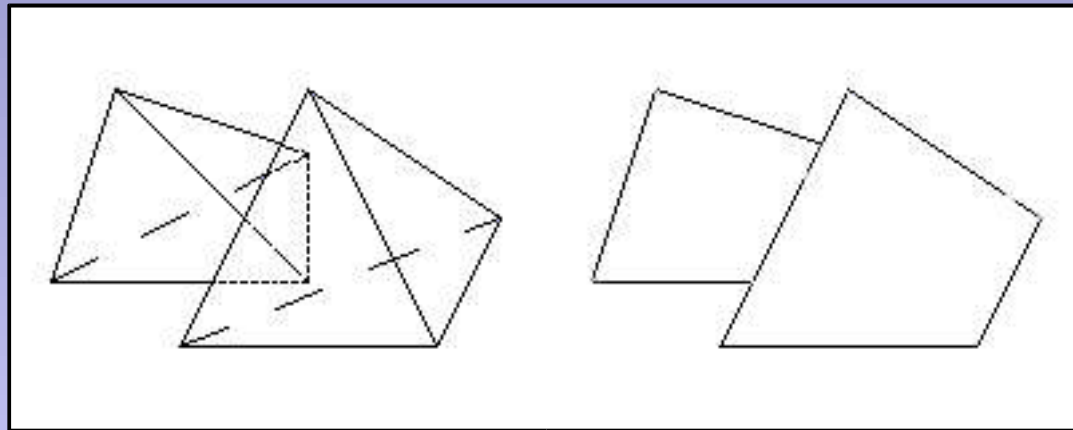
- Use a depth buffer
- Depth function $f(d_1, d_2)$
 - Pixel with depth d_1 overwrites pixel with depth d_2 iff $f(d_1, d_2)$ is true
- Very simple!

Pseudo-code

- Draw background with white colour
- Enable back face culling, set depth function to “Less Than”
- Render front facing polygons in white
- Enable front face culling, set depth function to “Equal To”
- Draw back facing polygons in black
- Repeat for a new viewpoint

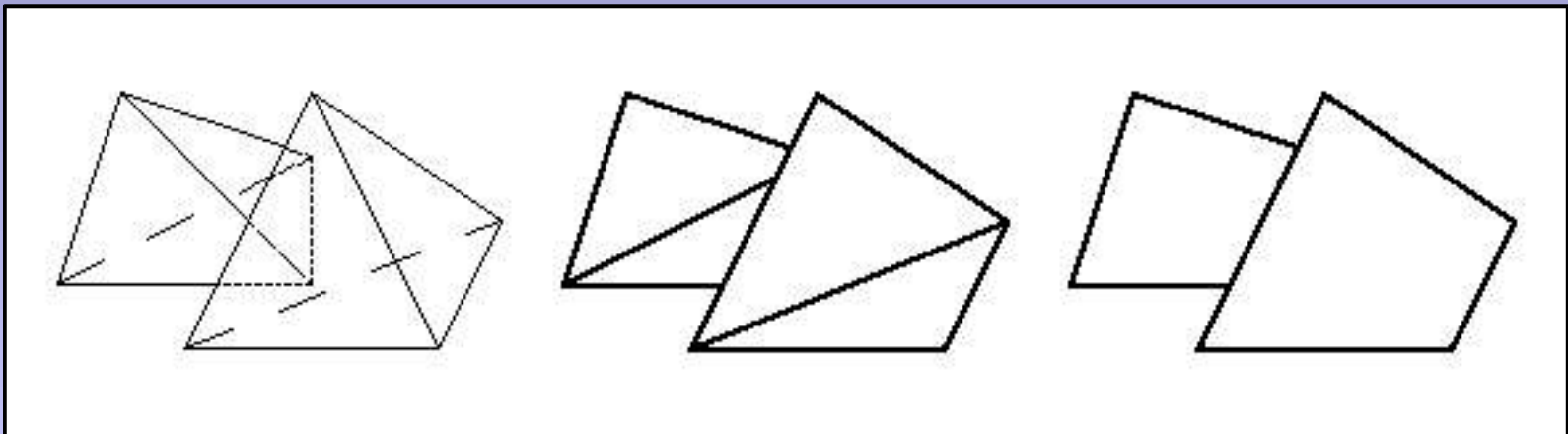
Limitations

- Quantization and sampling
 - Some pixels may be missed
- Silhouette edges at most one pixel wide



Modification 1 - Wireframe

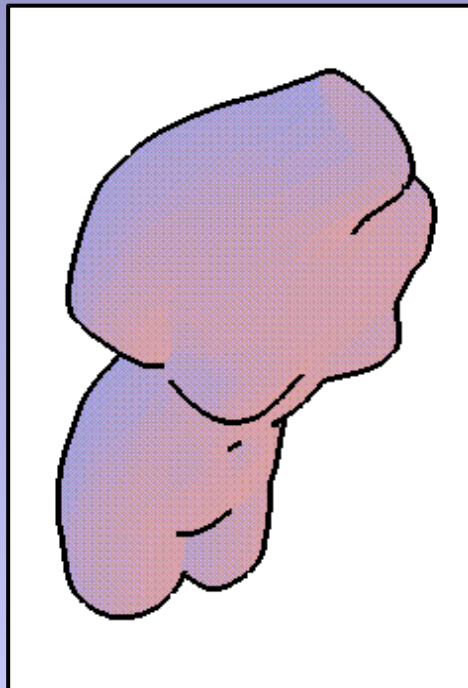
- Render front facing polygons as before
- Back facing polygons
 - Render in wireframe
 - Set depth function to “Less Than Or Equal”



Original polygons | Back facing rendered in wireframe | Silhouette
Allows thicker silhouettes, no missing pixels

Advantages

- Visible edges with constant thickness
- Width and colour of wireframe may be modified according to distance/orientation from camera



Raskar and Cohen

Venus - Wireframe Technique

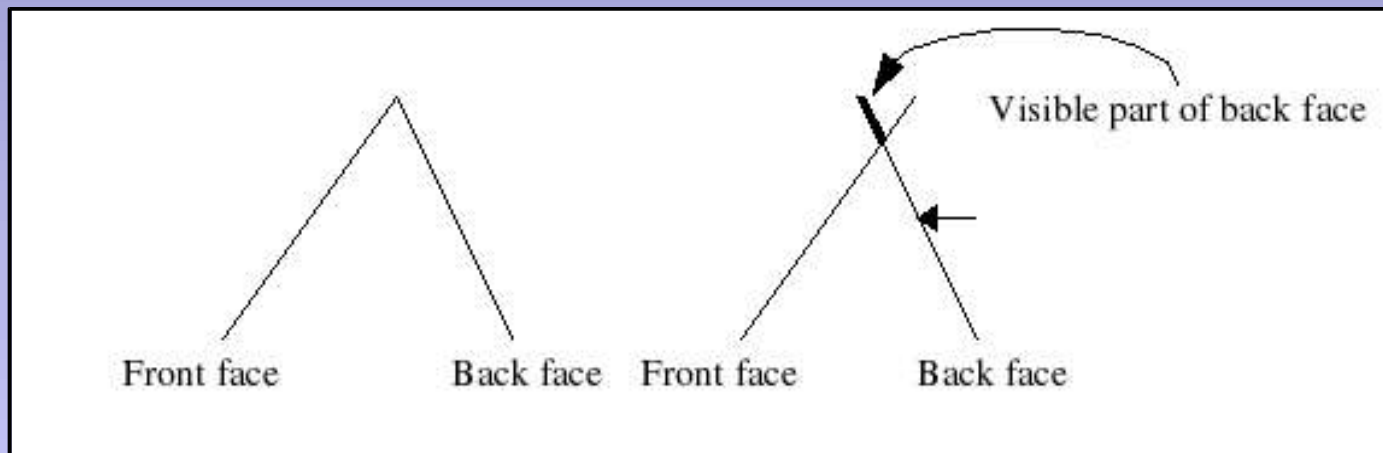
<http://www.cs.unc.edu/~raskar/Sil/Images/venusWireframe.gif>

Disadvantages

- Two types of primitives, polygons and lines
 - Restrictions in rendering pipeline
- Gaps between silhouette edges of neighbouring polygon as line width increases

Modification 2 - Translation

- Render front facing polygons as before
- Back facing polygons
 - Translate slightly towards camera
 - Set depth function to “Less Than Or Equal”



Methods of translation

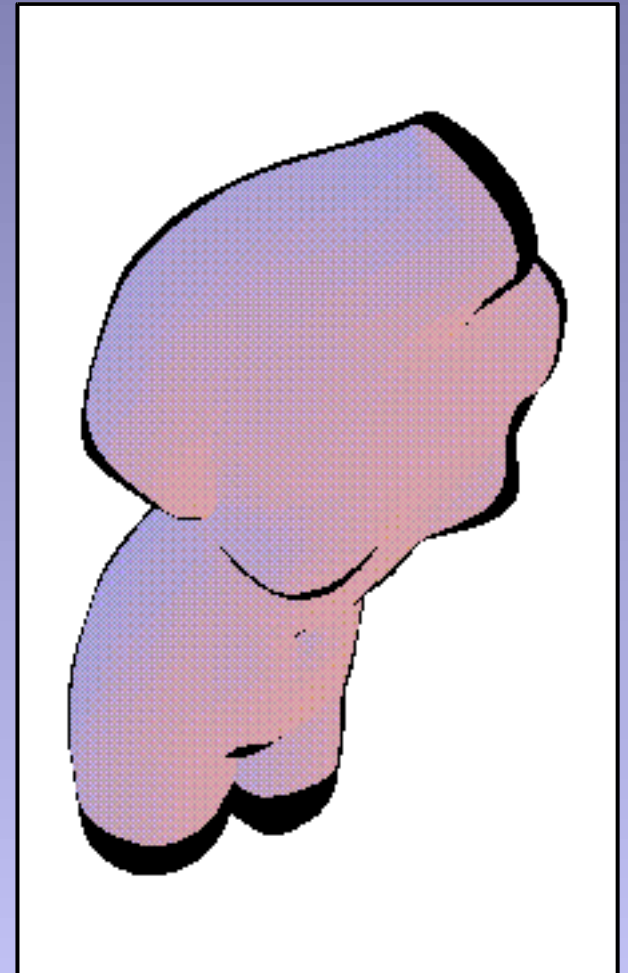
- Translate back-facing polygon by a fixed amount t
- Translate back-facing polygon by $k*z$ (z -scaled)
 - z : average distance from polygon to camera
 - k : scaling factor
- API like *glPolygonOffset()*
 - depends on z as well as orientation of polygon

z -scaled translation

- Can be achieved two ways
- Scale down model with respect to camera
 - The closer the back-facing polygon, the shorter the translation distance
- Change the depth range of the view frustum
 - Move the near z -plane further back or vice-versa

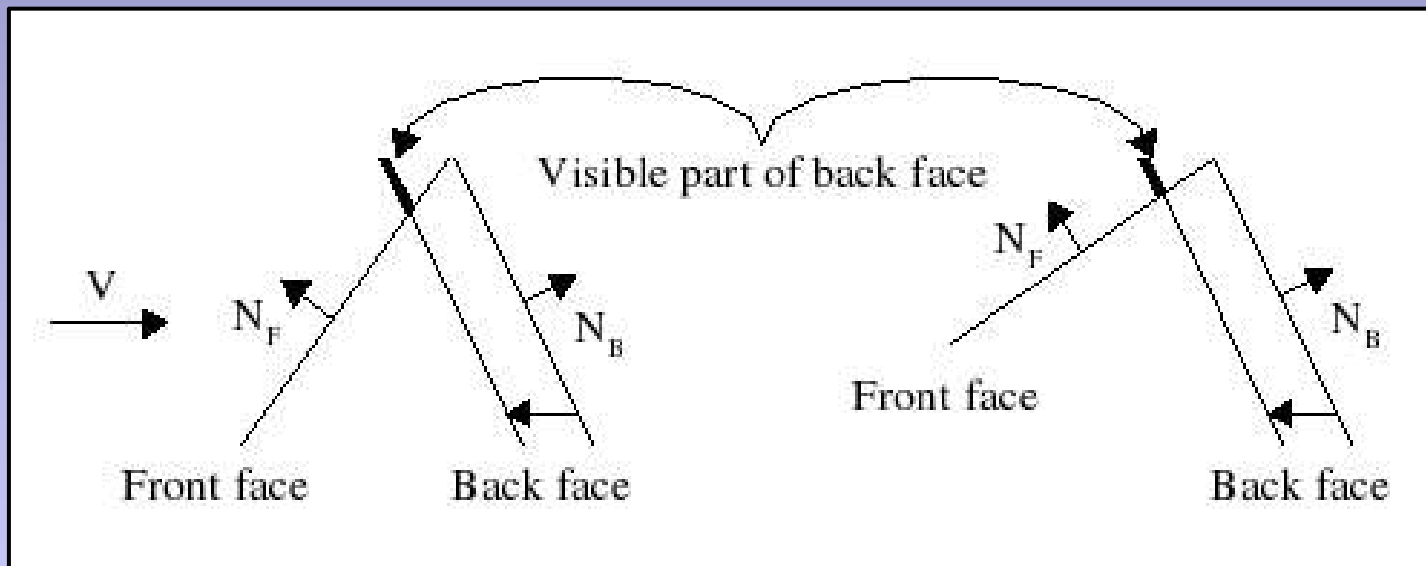
Characteristics

- Large scaling factor
 - smooth continuous silhouette edges
- Varying line widths: artistic?
 - Looks good for small angles
 - Depends on taste, context



Varying Line Width

- Silhouette depends on
 - Viewer orientation
 - Orientation of back face
 - Orientation of front face



Visible part of back-facing polygon depends both on $V \cdot N_F$ and $V \cdot N_B$

Varying Line Width

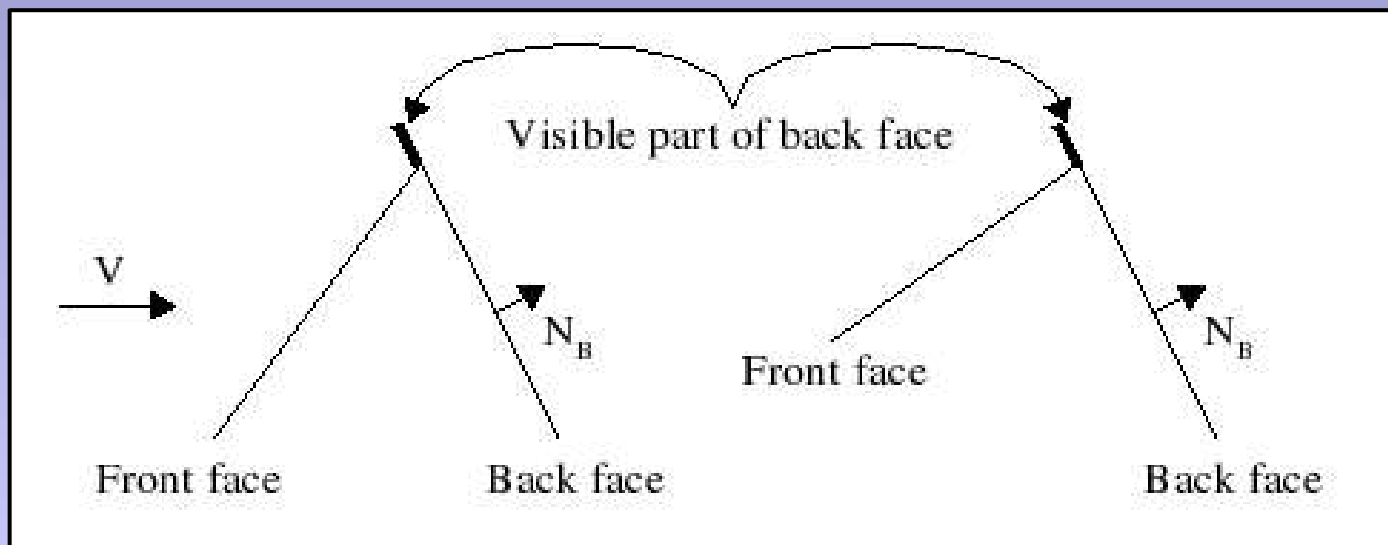
- Can be either a feature or a problem
- Feature: simple translation is good!
- Problem: want uniform lines
 - Need adjacency information to use translation
 - Recall goals: we want to avoid this!

Modification 3 - Fattening

- “View-dependant modification of back-facing polygons”
- Draws constant width silhouette edges
- Assumes convex polygons

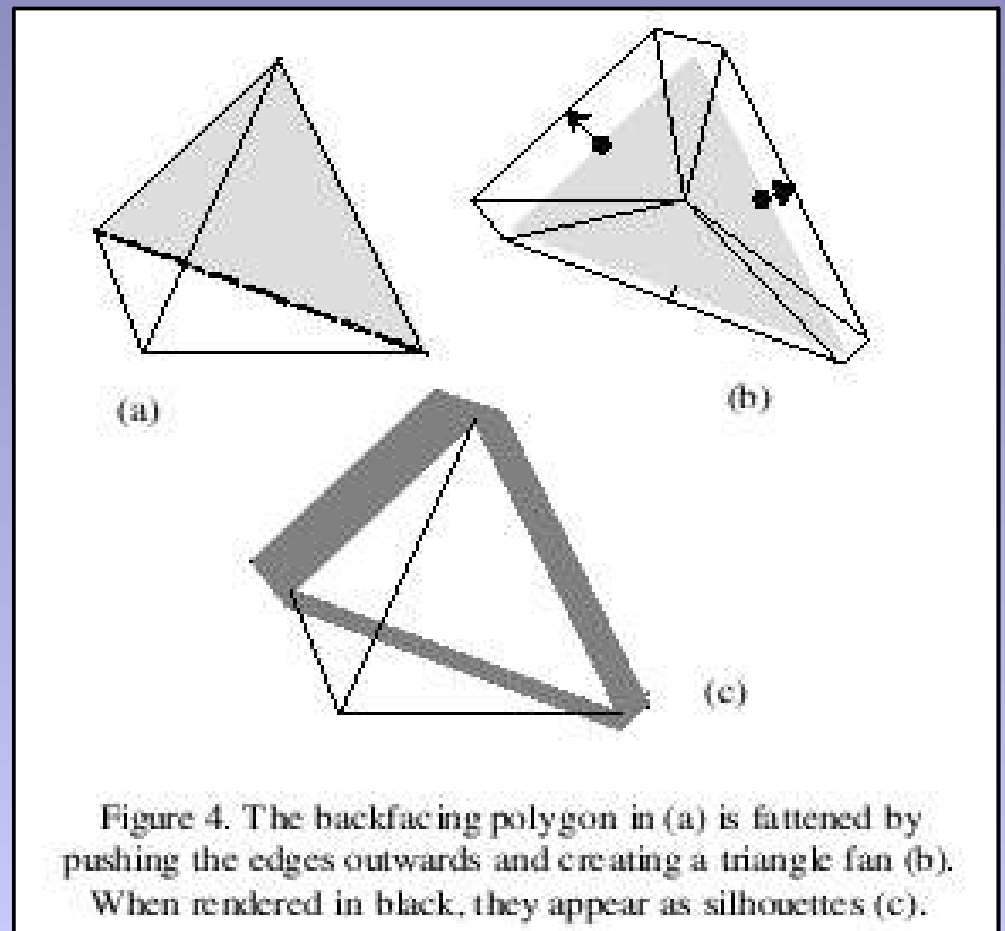
Fattening Method

- Edges of back-facing polygons pushed outwards
 - Proportional to $z / (V \cdot N_B)$
 - z : distance from midpoint of the edge to the camera
- Depends only on orientation of back face



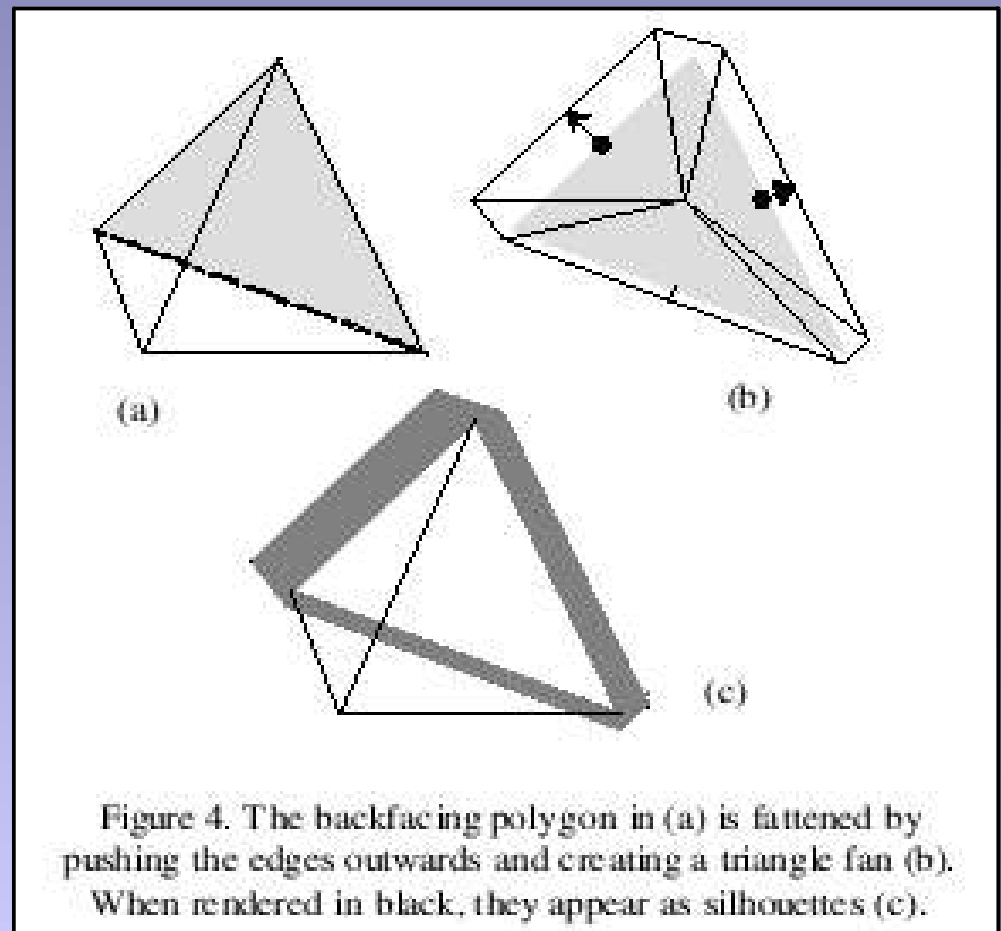
Fattening Method

- Actually also depends on orientation of the *edges* of the back face



Fattening Method

- E is an edge vector and $\cos(\alpha) = V \cdot E$



Fattening Method

- E is an edge vector and $\cos(\alpha) = V \cdot E$
- Fattening for E is proportional to $z * \sin(\alpha) / (V \cdot N_B)$

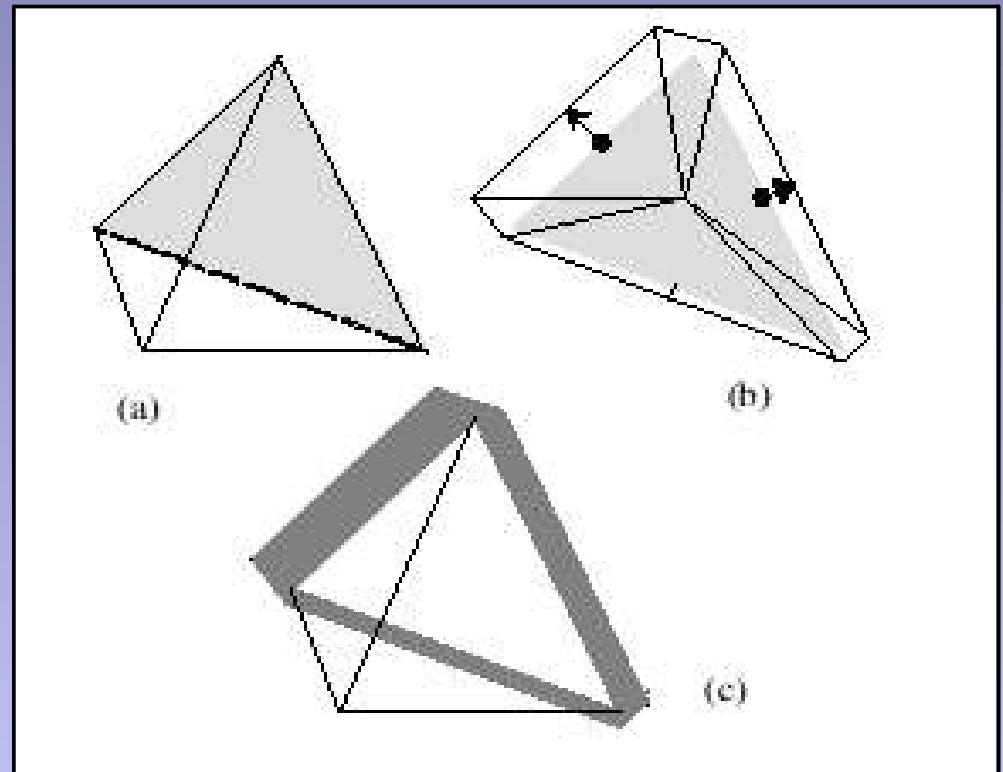


Figure 4. The backfacing polygon in (a) is fattened by pushing the edges outwards and creating a triangle fan (b). When rendered in black, they appear as silhouettes (c).

Fattening Method

- E is an edge vector and $\cos(\alpha) = V \cdot E$
- Fattening for E is proportional to $z * \sin(\alpha) / (V \cdot N_B)$
- Direction is $E \times N_B$

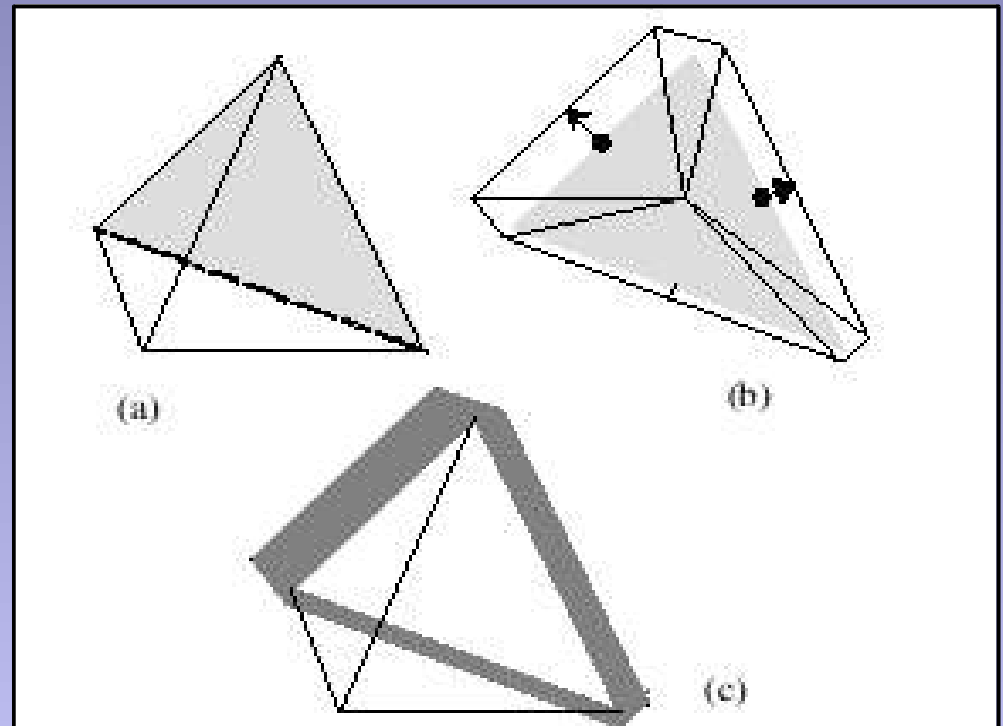


Figure 4. The backfacing polygon in (a) is fattened by pushing the edges outwards and creating a triangle fan (b). When rendered in black, they appear as silhouettes (c).

Fattening Method

- E is an edge vector and $\cos(\alpha) = V \cdot E$
- Fattening for E is proportional to $z * \sin(\alpha) / (V \cdot N_B)$
- Direction is $E \times N_B$
- z can be approximated once per polygon

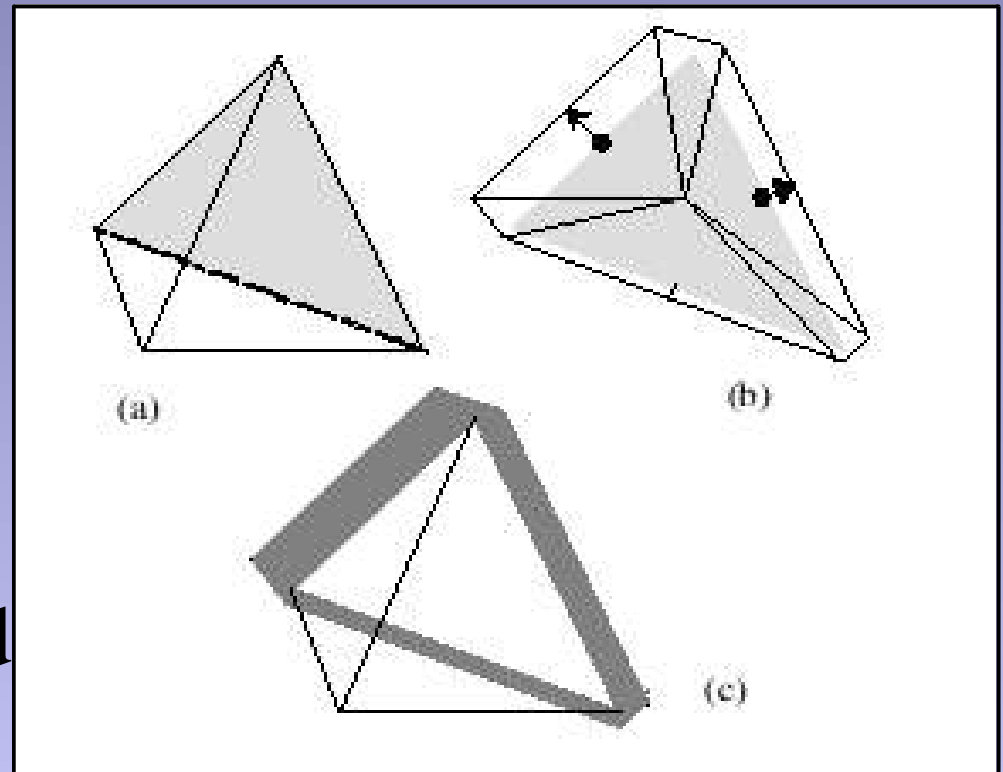
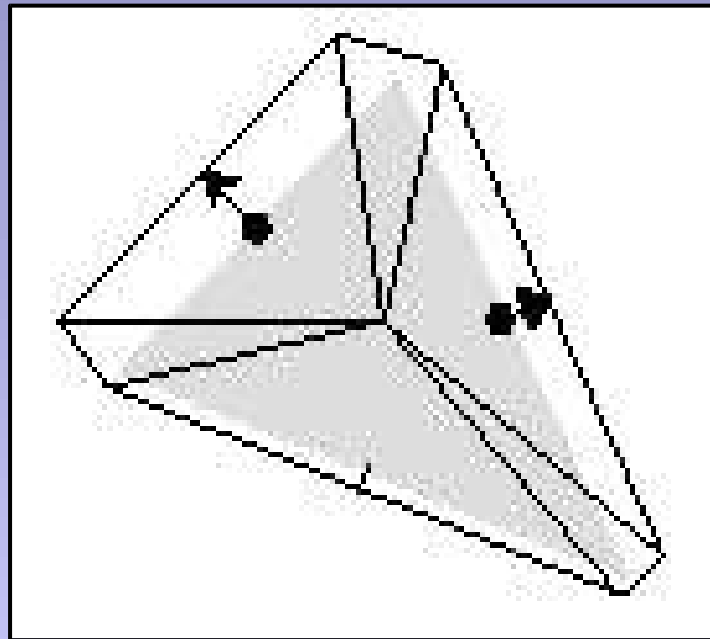


Figure 4. The backfacing polygon in (a) is fattened by pushing the edges outwards and creating a triangle fan (b). When rendered in black, they appear as silhouettes (c).

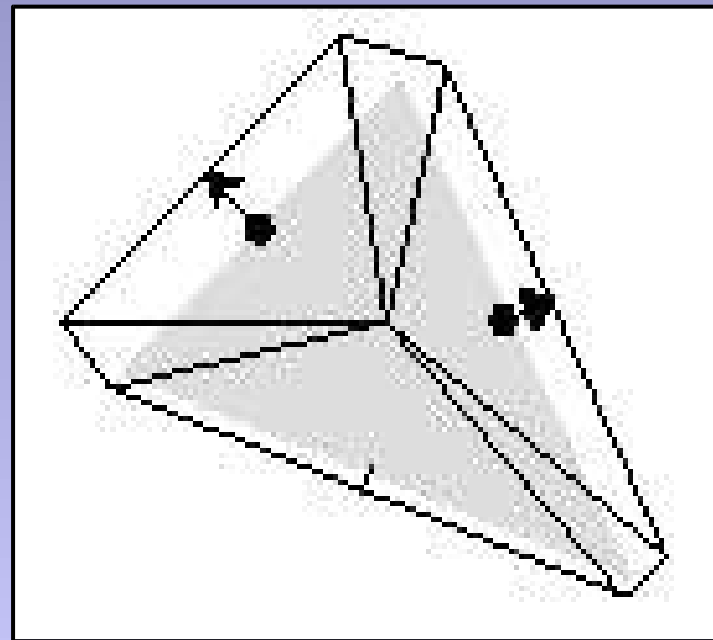
Fattening Method

- n -sided polygon gets $2n$ extra vertices
- Vertices are connected: triangulation
 - Triangle fan



Fattening Method

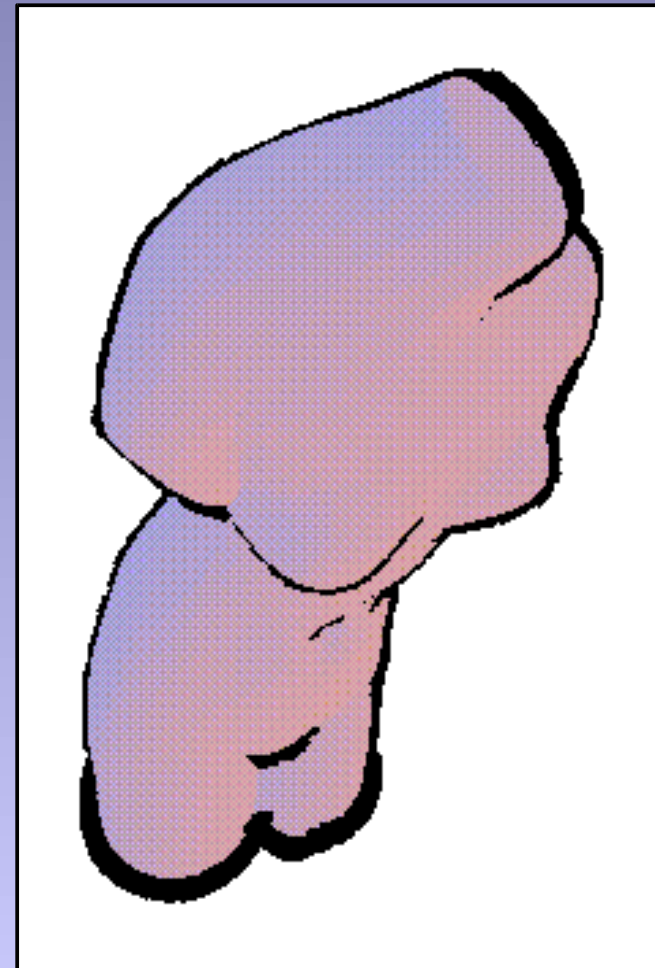
- Fattening at run-time involves shifting the vertices



Raskar and Cohen "Image Precision Silhouette Edges"

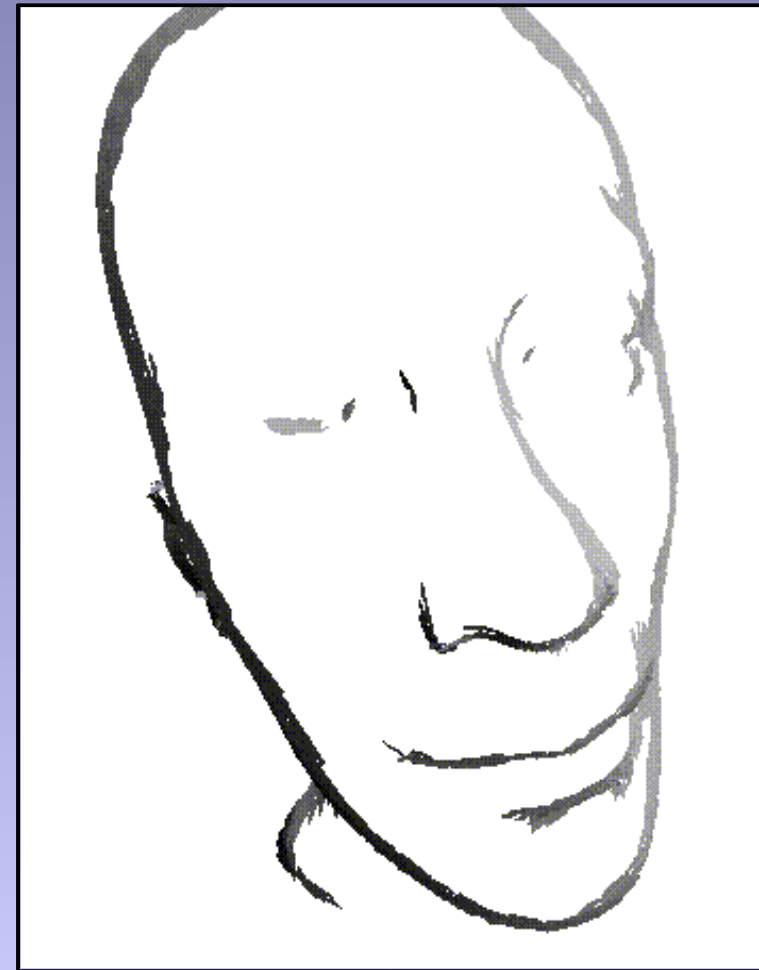
Results

- Better results than wireframe
 - Only polygons involved
 - No gaps between edges of neighbouring polygons
- No adjacency information needed



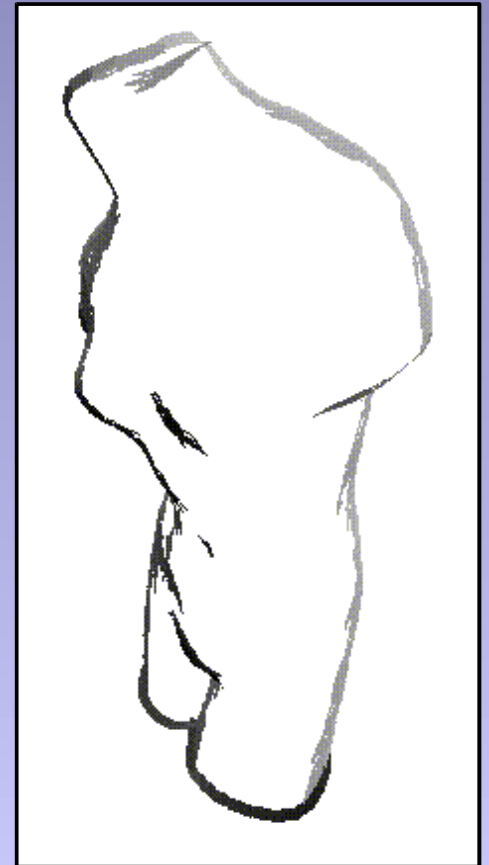
Results

- Use anti-aliasing, transparency, etc...
- Render different widths of silhouette edges with fattening parameter
- Use texture-mapping to get cooler silhouette edges



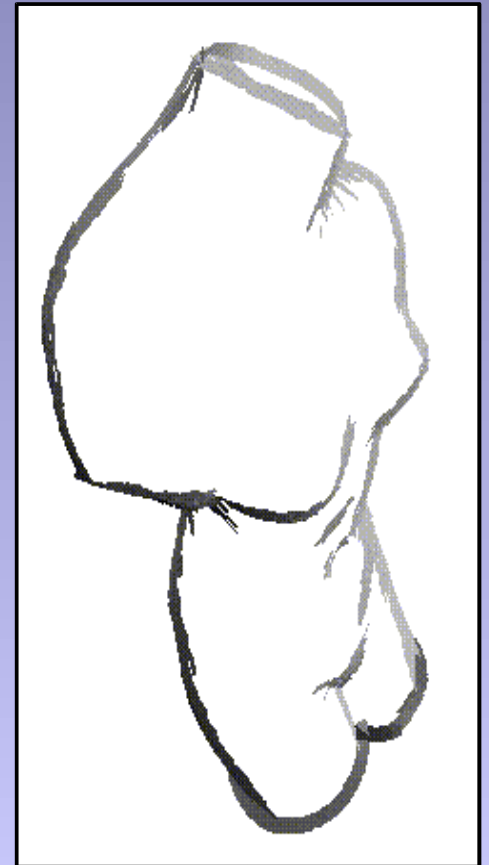
Extra Modifications

- Fatten front facing polygons with normal perpendicular to view direction ($0 < V \cdot N < 0.1$)
 - Creates charcoal like strokes



Extra Modifications

- Fatten front facing polygons with normal perpendicular to view direction ($0 < V \cdot N < 0.1$)
 - Creates charcoal like strokes
- Simple lighting technique for colour
 - Vertex with normal N
 - $I = (I + V \cdot N) / 3$



Extra Modifications

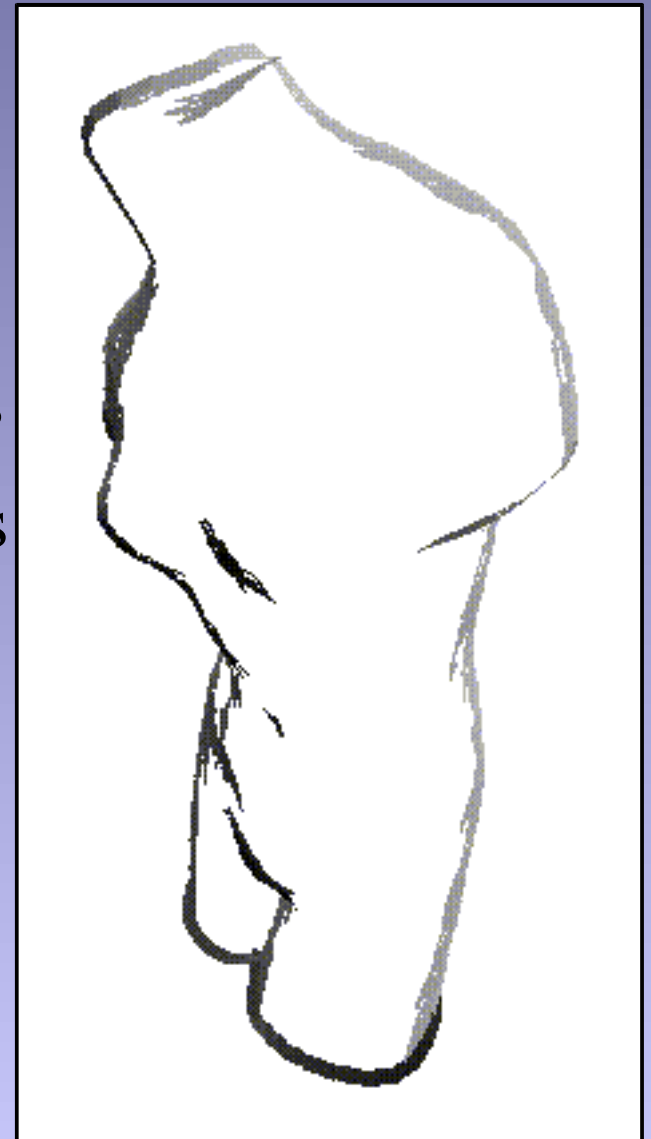
- Inconsistent order of vertices
 - No front- or back-facing polygons
- Find first layer by rendering all polygons with colour ID and read back framebuffer
- Set pixels to white, keep depth buffer
- Render remaining polygons in black for black silhouette

Outline

- Motivation
- Previous Work
- Method
- Results
- Conclusions

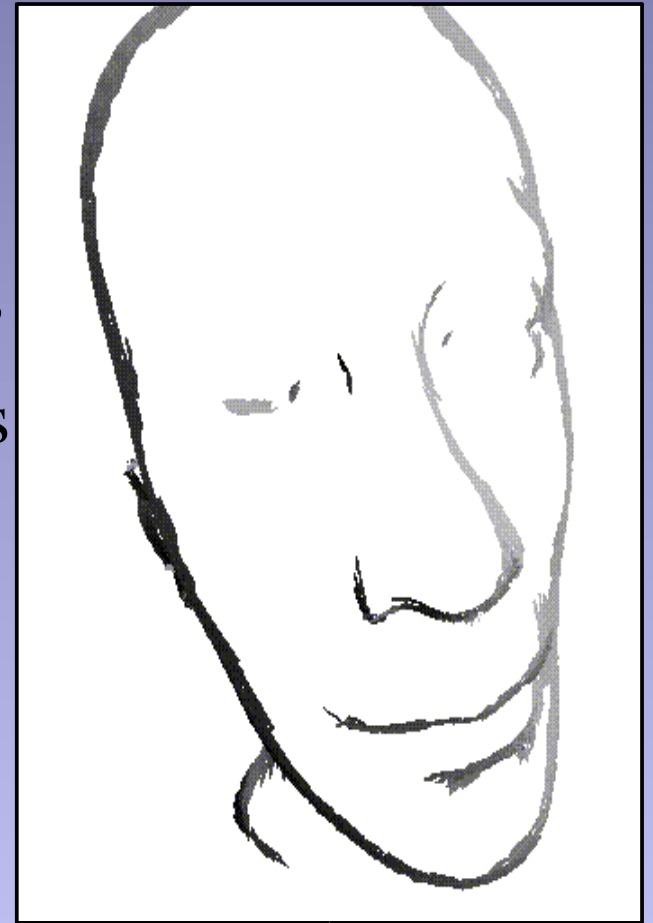
Performance

- Venus model: 5672 triangles
- Frame rates
 - W/o silhouettes.....66 fps
 - Wireframe.....40 fps
 - z-scaled.....50 fps
 - View-dependant fattening..11.5 fps



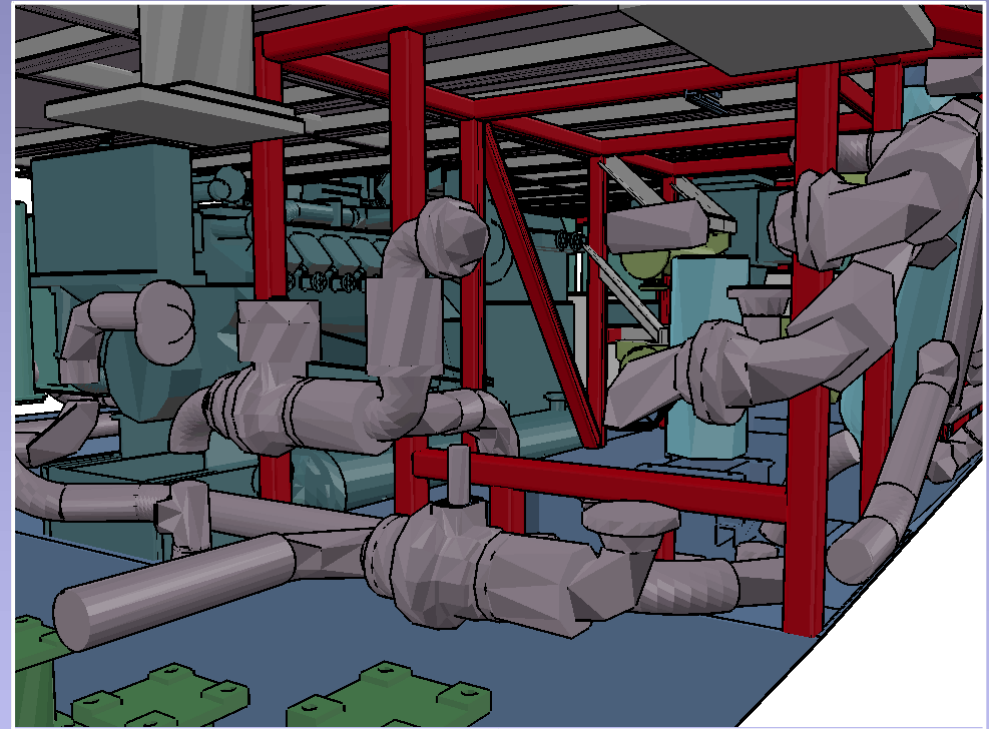
Performance

- Face model: 13408 triangles
- Frame rates
 - W/o silhouettes.....30 fps
 - Charcoal.....5 fps
- Performance drop with view dependant fattening due to extra triangles
 - 6x more back-facing



Performance

- Auxiliary machine room: 252000 triangles
- Frame rates
 - W/o silhouettes.....6 fps
 - Wireframe.....4.2 fps



Raskar and Cohen – I3D Slides

Outline

- Motivation
- Previous Work
- Method
- Results
- Conclusions

Conclusions

- Robust, real-time technique
- Traditional polygon rendering setup
- Simple method with sophisticated enhancements
- No pre-processing
- No adjacency information

Conclusions

- Complete scene traversal required
- Could render fewer back-facing polygons
 - Find potential silhouette edges more efficiently using techniques described in Markosian et al. 1997
- Could eliminate pairs of polygons both front- or both back-facing
 - Described in Zhang et al. 1997

Conclusions

- Far-away scene results in cluttered silhouettes
 - Improve technique by fattening according to average distance of scene from camera