

Shading



Pixel-level processes

- For each polygon
 - Find those pixels that are inside the polygon (scan conversion or rasterization)
 - For each pixel determine hidden surface removal
 - For each pixel determine intensity (shading and illumination models)

Scan conversion (Rasterization)

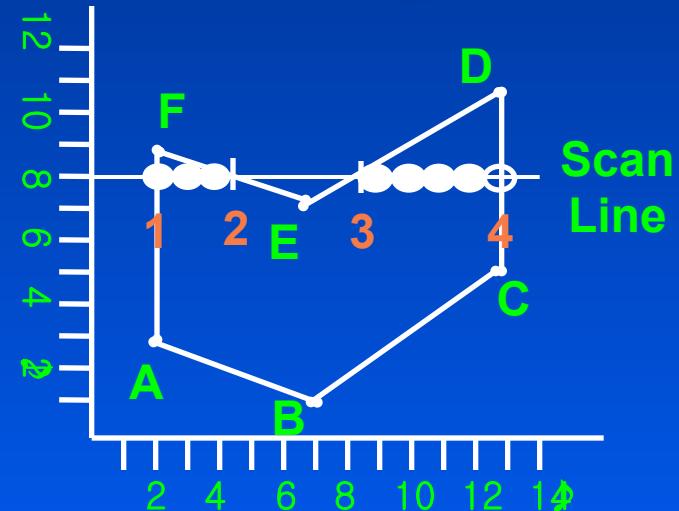
- Process of converting polygons into set of pixels
- Pixels are at integer values of x and y
 - How to convert real-number coordinates into integers?
 - Round up, round down, does not matter: be consistent
 - For example: for start of scan line, round up for end of scan line, round down (avoid overlap)

Scan conversion

- For each scanline, determine edges of polygons that intersect
- Find the start and end of the span
- Rely on scanline and pixel coherence to linearly interpolate (between scanlines and between pixels)
 - Add slope value to determine next start or end of span
 - Same technique used to linearly interpolate other values (z value for z-buffer, color for Gouraud shading, normal for Phong shading)

Scanline polygon fill

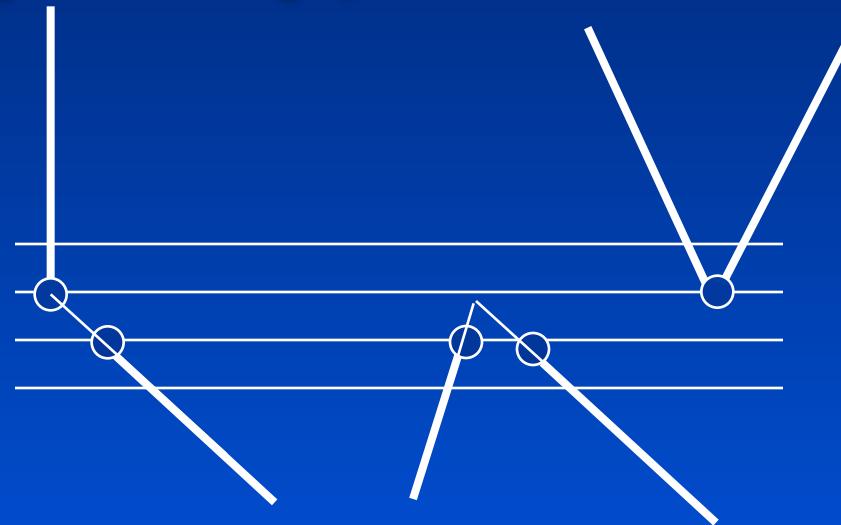
- Determine intersections of an edge with scanline
- Use odd-even rule to determine inside spans (normally, no concave polygon)
- Problems at the vertices (assume no special end-handling)
 - For scanline 9, at point F
2 intersections so everything is fine
 - For scanline 3, at point A
2 intersections:
no fill and then fill after the next intersection...problem



Alternative handling of vertex-scanline intersection

- Shorten one of the edges (e.g. lower edge)

- Y_{\min} counted
 - Y_{\max} not counted
(shortened)



- One other special case to consider: horizontal edges
 - Using above solution, bottom horizontal edge drawn, top horizontal edge not drawn

Using coherence

- Brute force method for determining intersections of edges with scanlines is expensive
- Coherence is the similarity of one part of the image to another part that can be used to speed up computation
- Note that the x intercept of adjacent scanlines with an edge differ by a constant
- If we know the x intercept of one scanline, the next one can be gotten by one addition

- Slope of an edge with respect to pixel coordinates

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

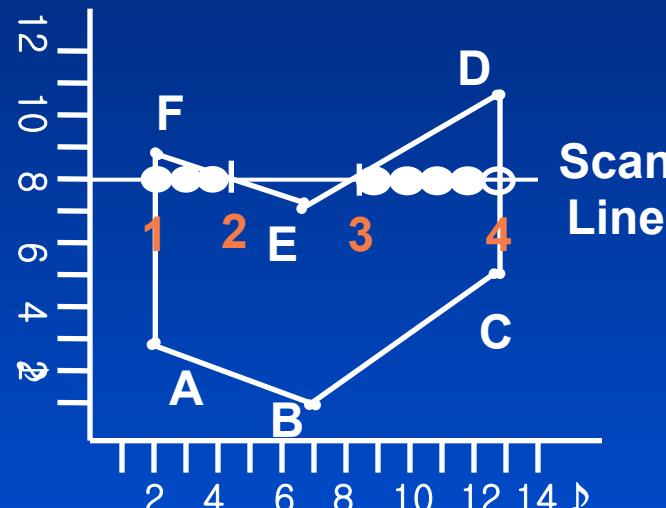
- Difference in y between two successive scanlines is 1

$$y_{k+1} - y_k = 1$$

- Therefore

$$x_{k+1} = x_k + 1/m$$

Edge Table



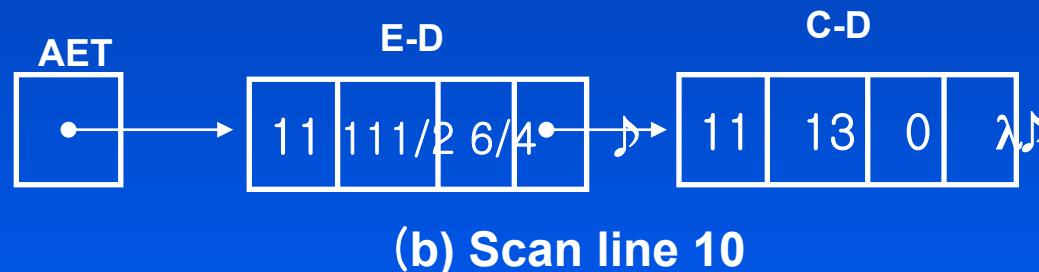
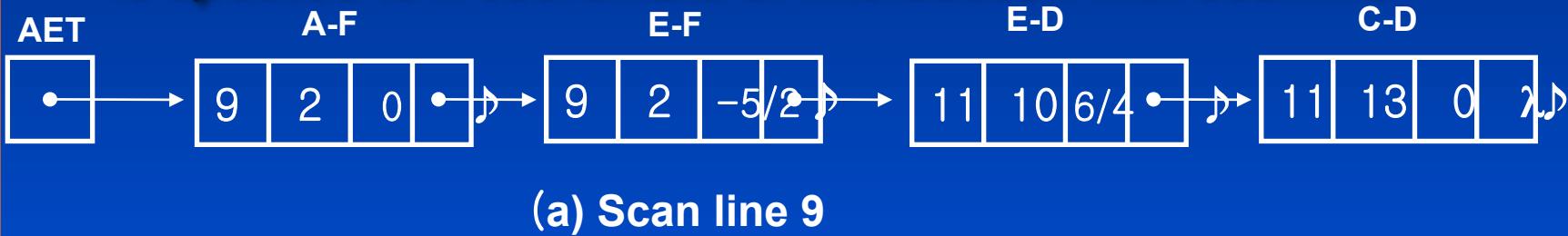
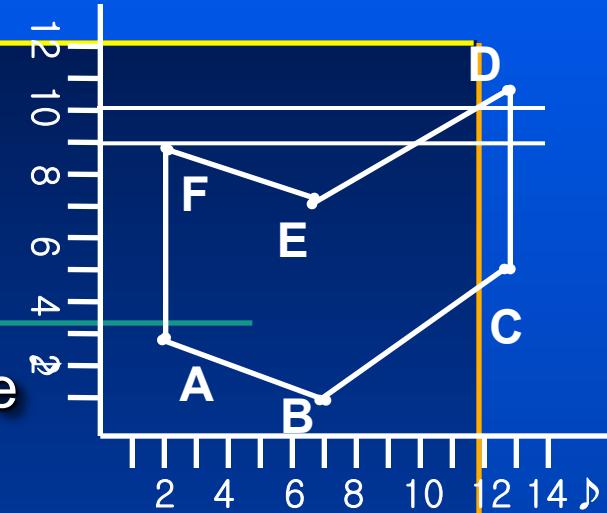
Y_{max}	$X_{min}: x \text{ coord of bottom end pt}$	$\Delta X \text{ over } \Delta y$
-----------	---	-----------------------------------

Edge Table

Edge Table			
12			
10			
8			
6			
4			
2			
0			
	Ymin		
		9 7 $\frac{3}{2}$ ●	11 7 $\frac{6}{4}$ ↗
		11 13 0 λ ↗	E-D
		9 2 0 λ ↗	C-F
		3 7 -5 $\frac{1}{2}$ ↗	A-F
		5 7 $\frac{6}{4}$ λ ↗	B-C

Active Edge Table

- Contains all edges that intersect current scanline
- Second entry (x-coordinate of bottom endpoint) is updated to x coordinate of intersection with scanline



Polygon Scan Conversion Algorithm

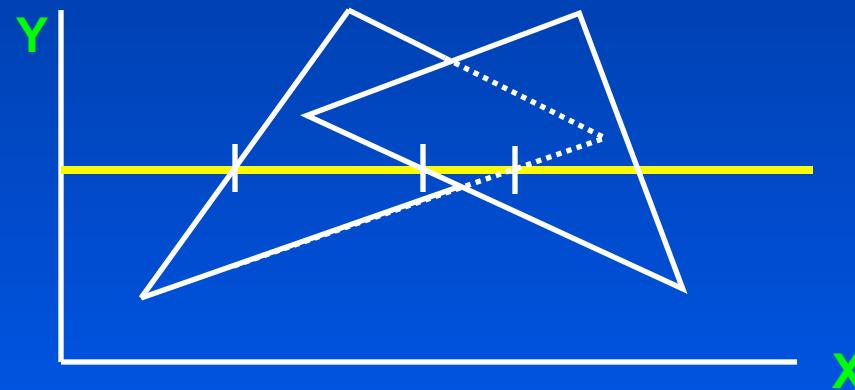
- Convex or concave polygon in general
 - Can be simplified for convex and even more for triangles only
- Once the ET has been formed, processing steps for scan-line algorithm are:
 1. Set y to smallest y -coordinate which has an entry in ET (first non-empty bucket)
 2. Initialize the AET to be empty

3. Repeat until the AET and ET are empty :

- 3.1 move edge(s) from ET bucket y whose $y_{\min} = y$ (entering edges) to ATE, maintaining ATE sort order on x
- 3.2 Fill in desired pixel values on scan line y by using pairs of x -coordinates from the AET
- 3.3 Remove from AET entries for which $y = y_{\max}$ (leave edges)
- 3.4 For each entry remaining in AET, replace x by $x + \text{increment}$
This places next scan-line intersection into each entry in AET
(algorithm from previous slide)
- 3.5 Because previous step may have caused the the AET to become out of order on x , re-sort AET (not a problem with single polygons)
- 3.6 Increment y by 1 (to the coordinate of the next scan line)

More than one polygon

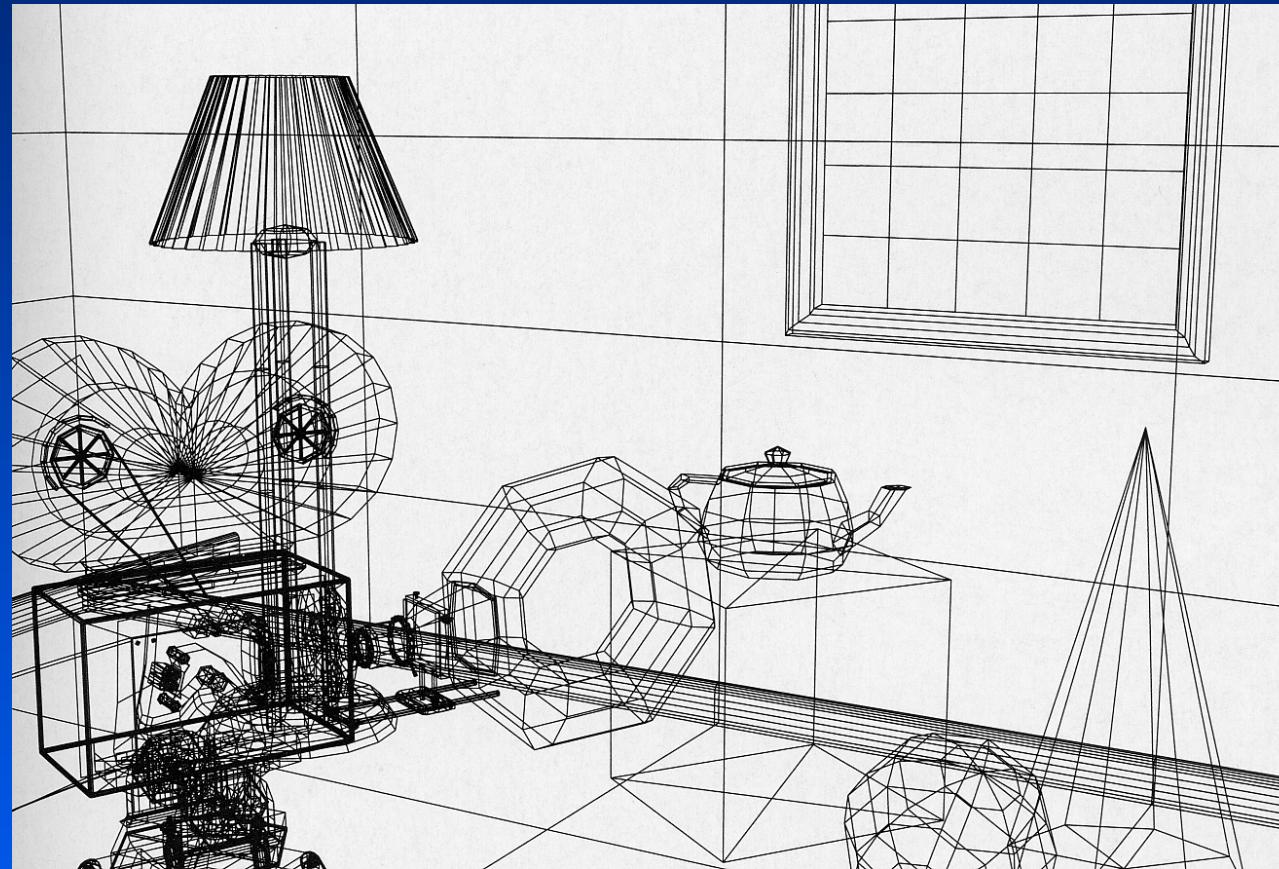
- Scan convert each polygon
- For z-buffer, overwrite if polygon in front of previous polygon



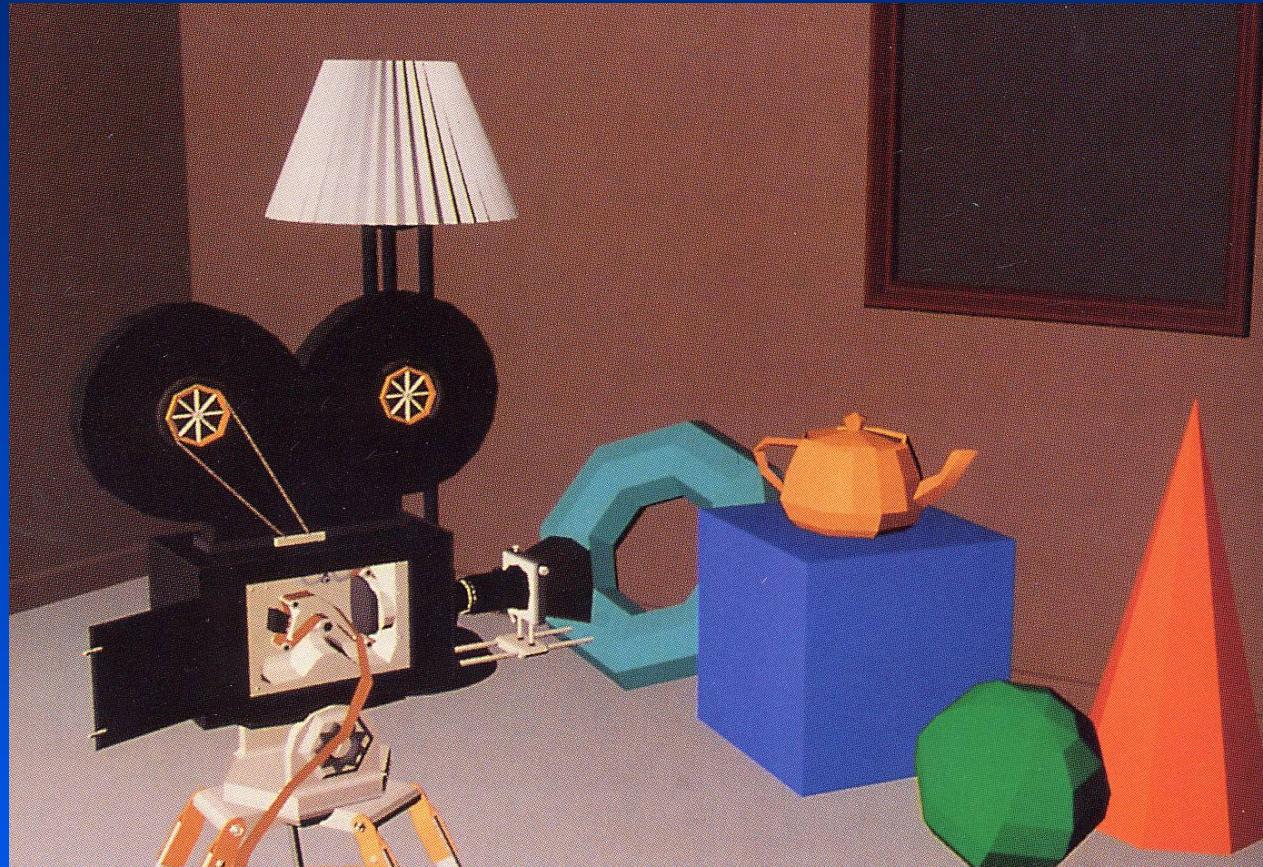
Shading

- Determination of pixel values (color)
- Difference between Shading and Illumination model
 - Illumination model tells you how to calculate light intensity
 - Shading model tells you when to invoke the illumination model
- Invoking illumination model at each pixel expensive
- Three methods: each treats a single polygon independent of all others
 - Constant
 - Gouraud (intensity interpolation)
 - Phong (normal -vector interpolation)

Wireframe



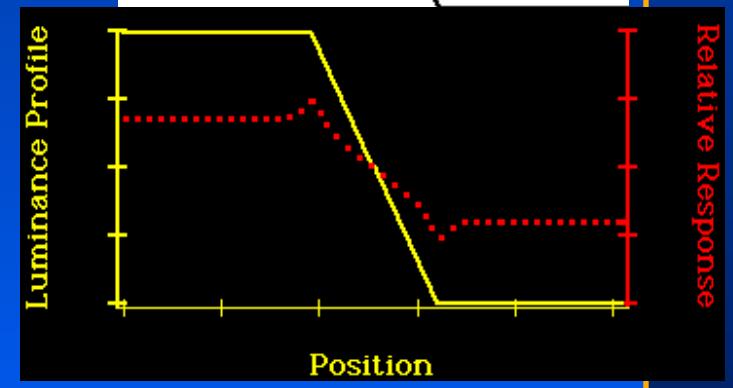
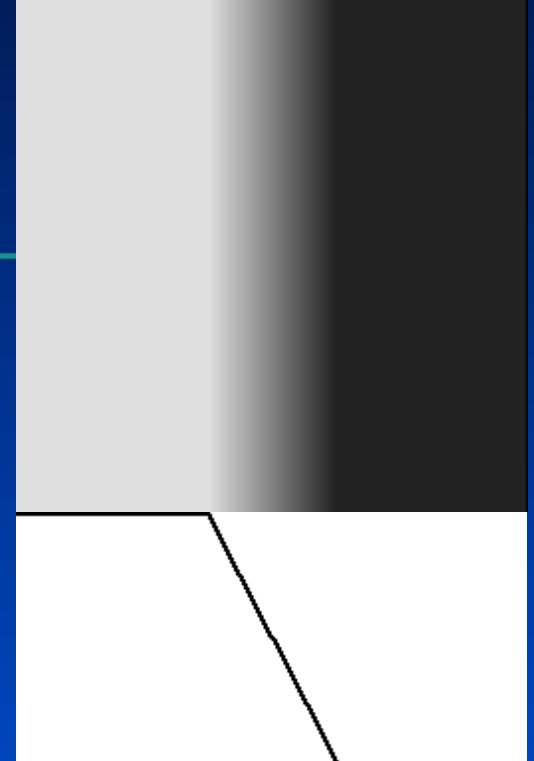
Constant Shading



- Single intensity value per polygon
- Gives reasonable results only if
 - Object is a polyhedron: polygon mesh is an approximation to curved surface, faceted look is a problem
 - Light source is far away from the surface (or is parallel) so that $\mathbf{N} \cdot \mathbf{L}$ is constant over each polygon
 - Viewing position is far away from the surface so that $\mathbf{V} \cdot \mathbf{R}$ is constant over each polygon

Mach Band

- Facets exaggerated by mach band effect
- Image shows white band on left and dark band on right of the ramp
- Graph shows the actual intensity
- Actual and perceived intensities



<http://www.cquest.utoronto.ca/psych/psy280f/ch3/mb/mb.html>

- Constant shading accentuate the facets due to Mach banding



Gouraud Shading

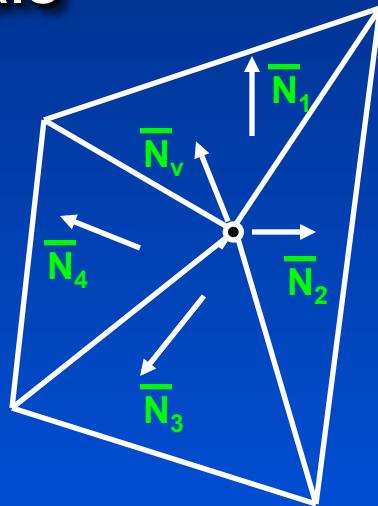


- Used for polygon approximations to curved surfaces
- Linear interpolation of intensity along scan lines
- Eliminates intensity discontinuities at polygon edges
- Mach banding is improved
- Still have gradient discontinuities: Mach band still there

Gouraud Shading

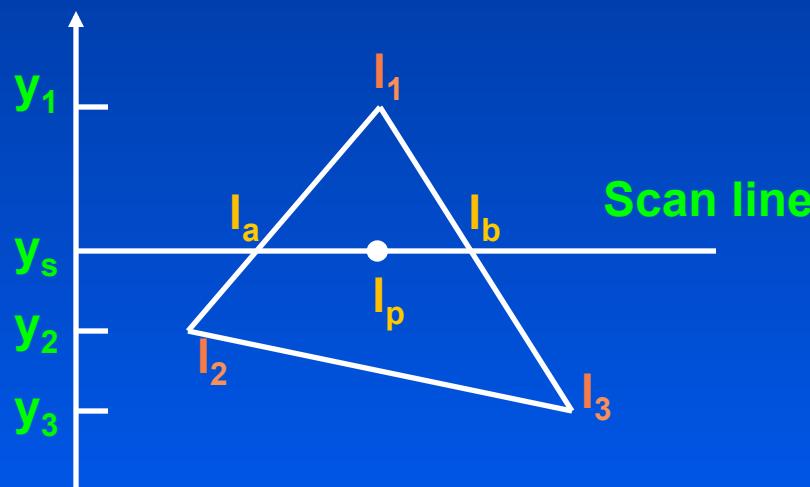
- 1. Calculate vertex normals as average of surrounding polygons' normals
- 2. Calculate the illumination at the vertices
- 3. Interpolate intensity along polygon edges
- 4. Interpolate intensity along scan lines♪

- Vertex normal is average of neighboring polygon normals



$$N_v = \frac{(N_1 + N_2 + N_3 + N_4)}{\|(N_1 + N_2 + N_3 + N_4)\|}$$

- Linearly interpolate intensities at the vertices to get intensities inside the polygon...tri-linear interpolation

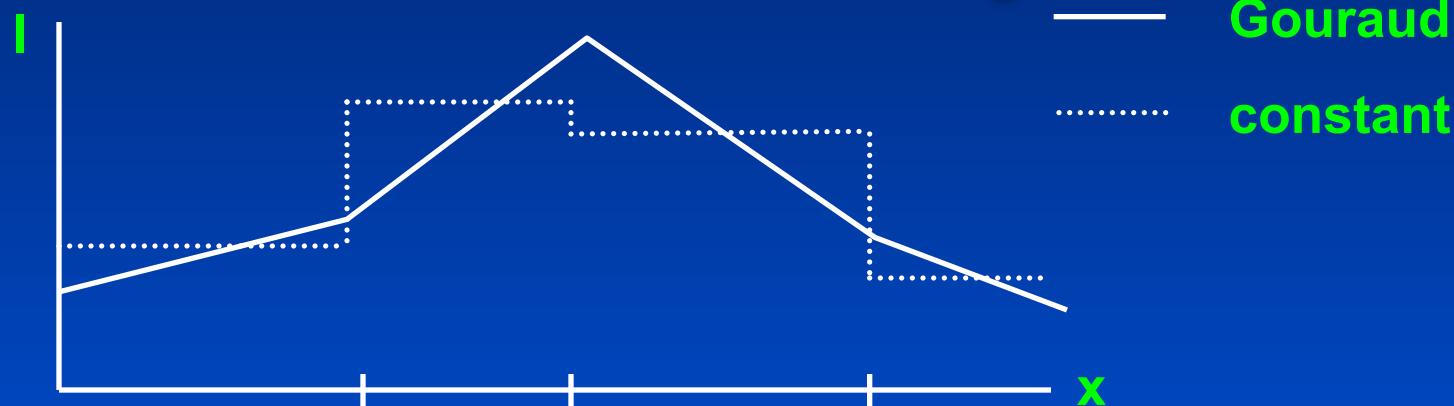


$$l_a = l_1 \frac{y_s - y_2}{y_1 - y_2} + l_2 \frac{y_1 - y_s}{y_1 - y_2}$$

$$l_b = l_1 \frac{y_s - y_3}{y_1 - y_3} + l_3 \frac{y_1 - y_s}{y_1 - y_3}$$

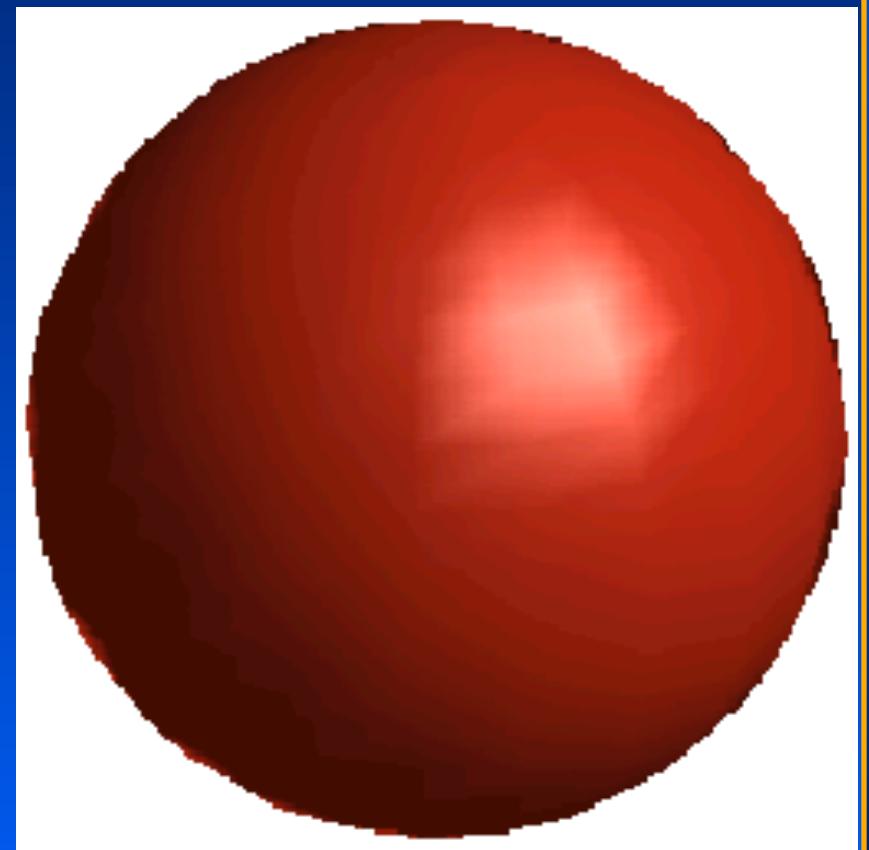
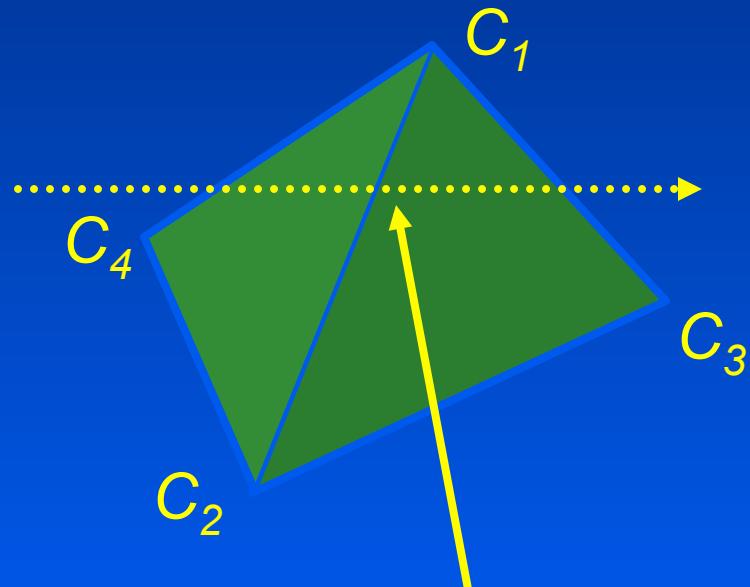
$$l_p = l_a \frac{x_b - x_p}{x_b - x_a} + l_b \frac{x_p - x_a}{x_b - x_a}$$

- Gouraud Versus constant shading

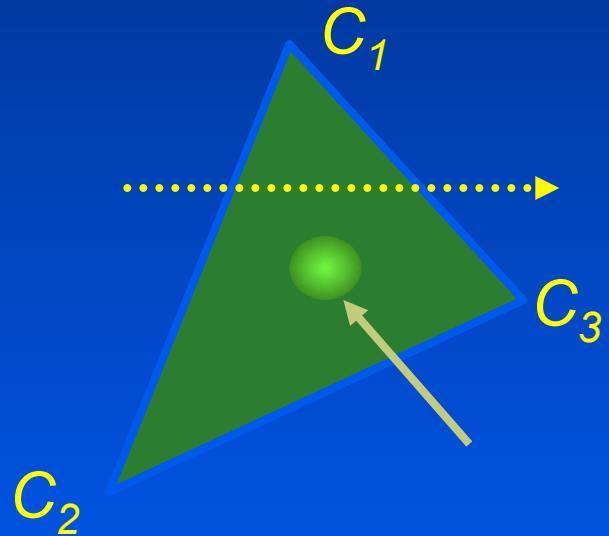


- Integrates nicely with scan line algorithm :
 - $\frac{\Delta I}{\Delta Y}$ is constant along polygon edge

- Notice the Mach banding effects on the highlight



- Highlights that fall inside the polygon (and not on the vertices) are missed



Phong Shading

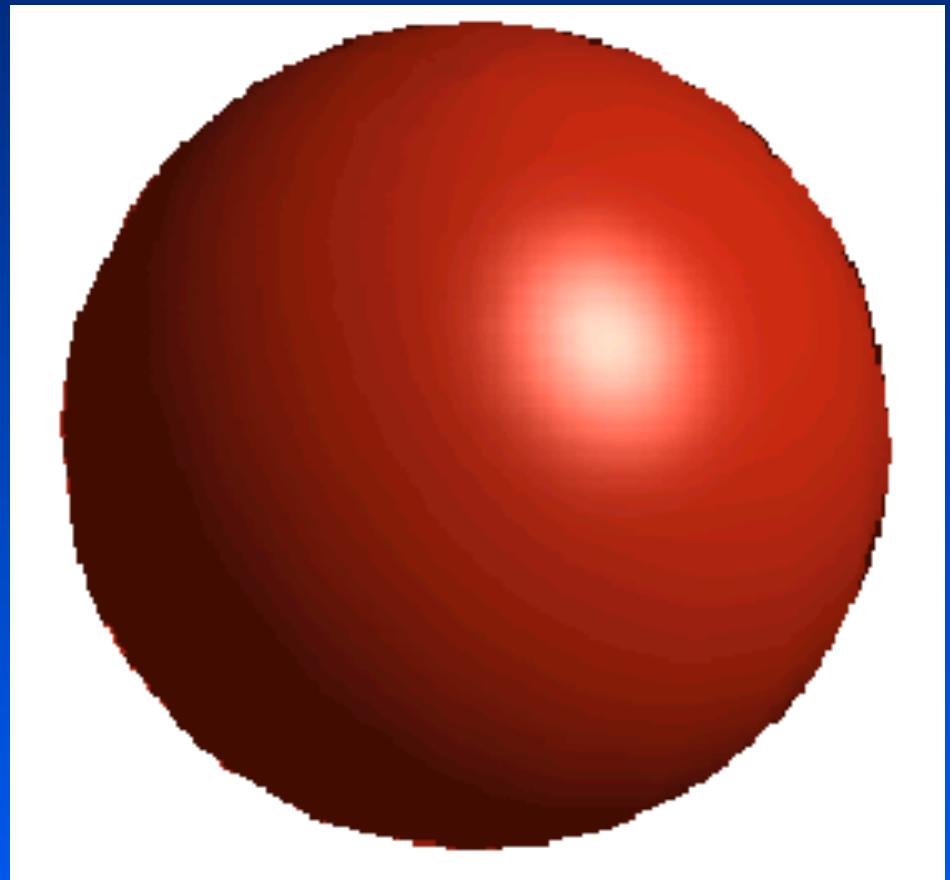


- Interpolate N rather than I
- Especially important with specular reflection
 - Since specular highlight may fall within a polygon
- Computationally expensive at each pixel to
 - Re-compute N : must re-normalize, requiring expensive square root
 - Re-compute I

Phong shading

- 1. Calculate vertex normals as average of surrounding polygons' normals
- 2. Interpolate normals along polygon edges
- 3. Interpolate normals along scan lines
- 4. Calculate the illumination at each pixel

- Interior point indistinguishable from smooth surface
- Silhouette still faceted unless use large number of polygons
- Notice the color quantization error and the resulting Mach banding



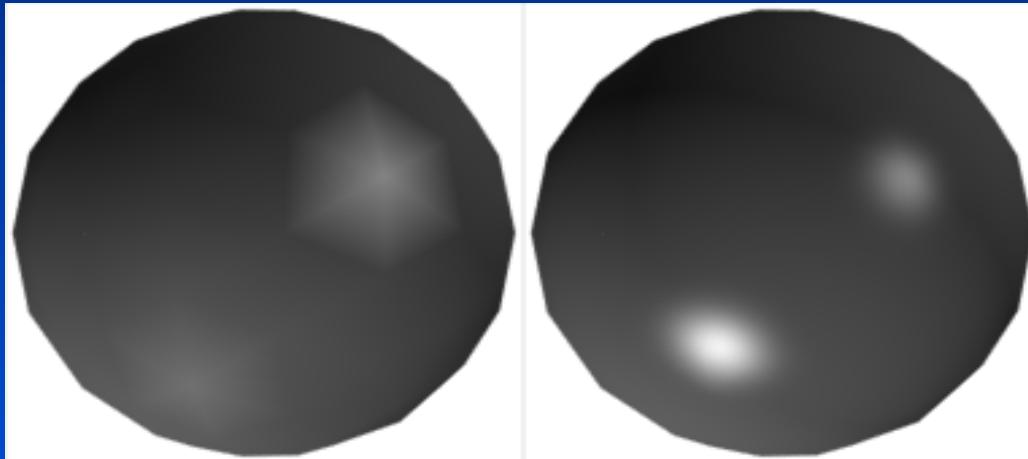
Where is it done?

- Illumination calculation done in the world coordinate system before perspective transformation
 - Perspective transformation will distort the space
- Interpolation (color or normals) done after perspective transformation during scan conversion
 - Take advantage of scanline and pixel coherence
 - Approach similar to the use of coherence in scan conversion

- What about Phong shading?
 - Need to wait until scan conversion to do the illumination model since we don't know what the normal for a pixel is until then
 - But do the calculation in the world coordinate system
 - May need to inverse transform to world coordinate for L and V

Problems at the Silhouette

- improved by increasing polygons

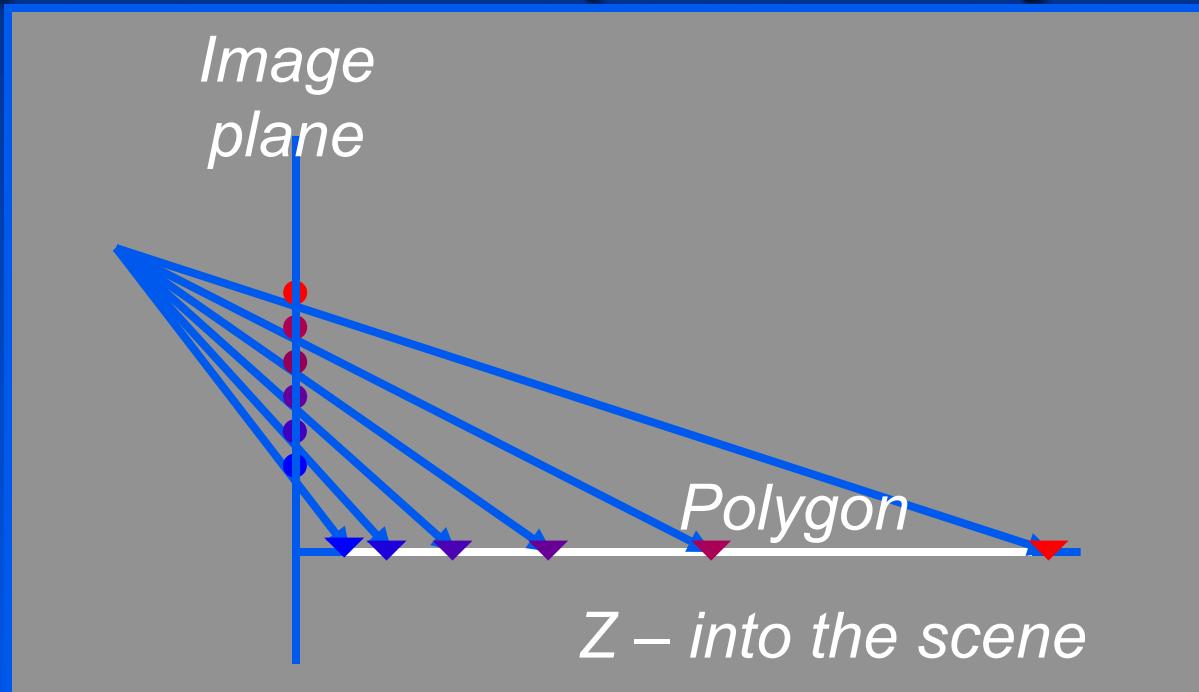


Gouraud

Phong

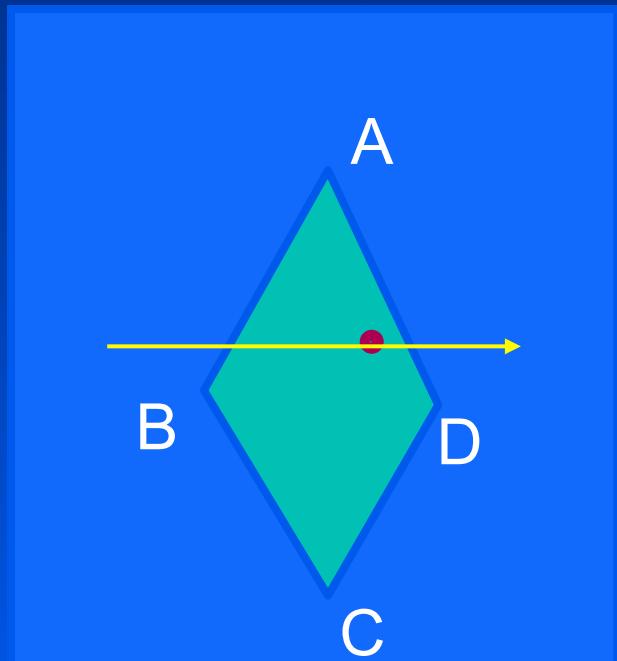
Perspective distortion

- linear interpolation across scan line (in image space) is not linear interpolation in object space

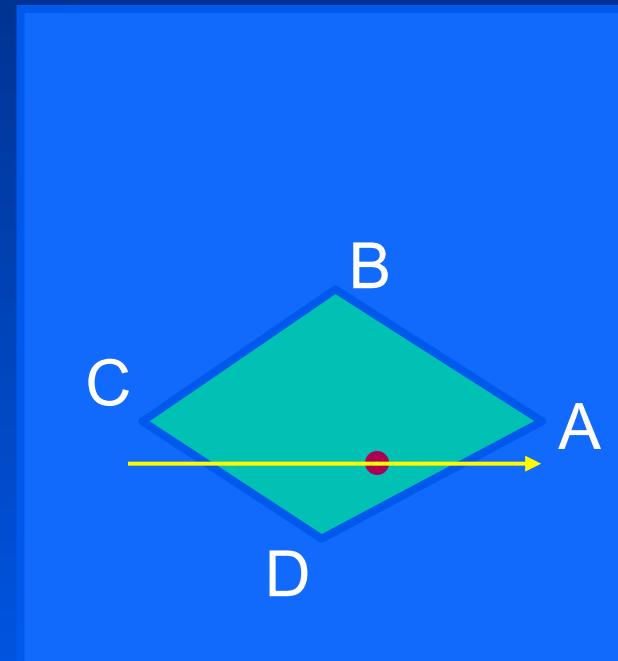


Orientation dependence

- What is interpolated changes as object is rotated



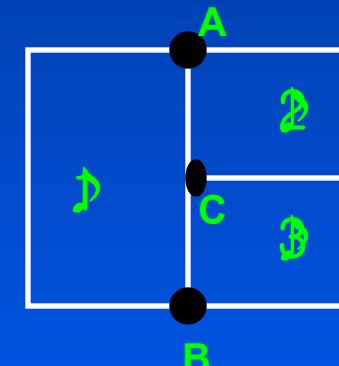
*Interpolate between
AB and AD*



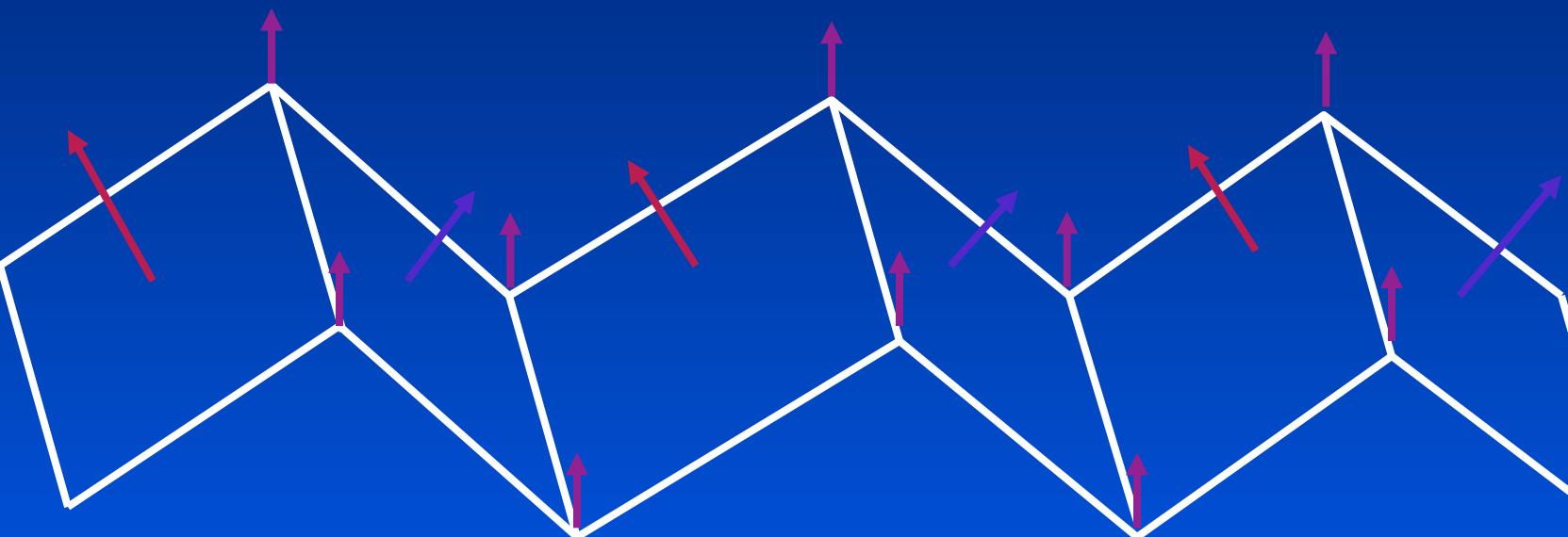
*Interpolate between
CD and AD*

Discontinuities due to non-shared vertices

- Near vertex C
 - On polygon 1, interpolated from A and B
 - On polygon 2 and 3, from vertex C
- This kind of polygon usually not allowed



Error due to averaging normals



Next: Visible surface algorithms

