# 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)


© 2010 James K. Hahn

# 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)


© 2010 James K. Hahn

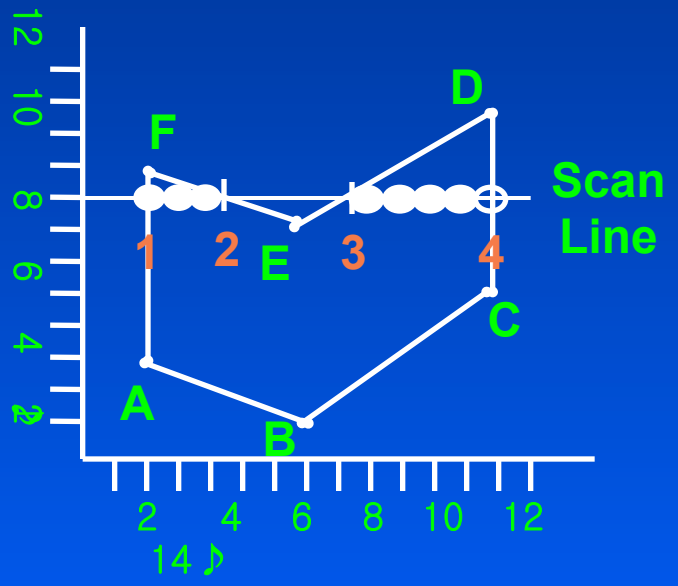
# Scan Conversion

- 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)


© 2010 James K. Hahn

# Scanline Polygon fill

- Determine intersections of an edge with scanline
- Use odd-even rule to determine inside spans
- Problems at the vertices (assume no special end-handling)
  - For scanline 9, at point F
    2 intersections so everything is fine
  - For scanline 3, after point A
    no fill and then fill after the next intersection…problem
  - Can resolve by noting that for the second case, the two edges are on opposite sides of scanline
    (y is monotonically increasing)

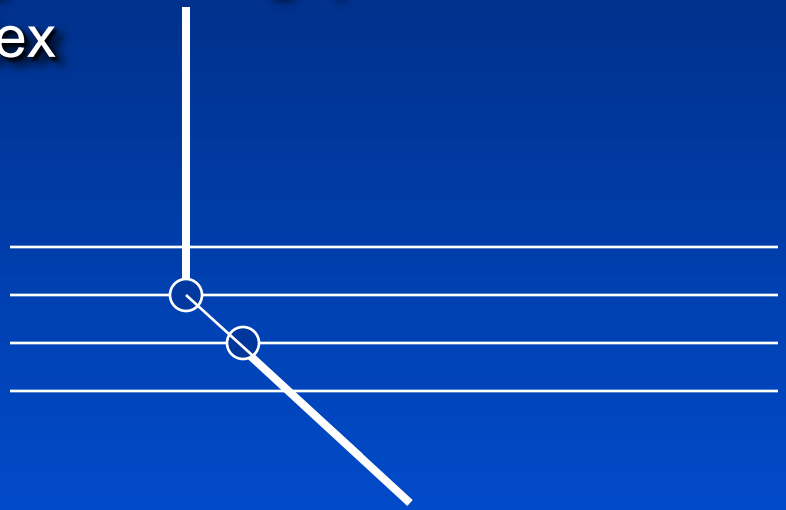




© 2010 James K. Hahn

# Alternative handling of vertex-scanline intersection

- Shorten one of the edges (e.g. lower edge) so that there is only one intersection at a vertex



- One other special case to consider: horizontal edges


© 2010 James K. Hahn

# 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


© 2010 James K. Hahn

- 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

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

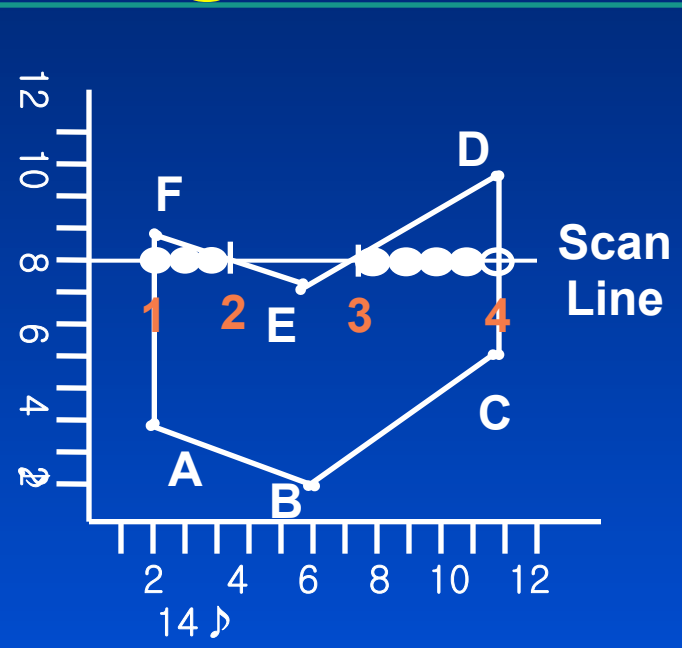
- Therefore

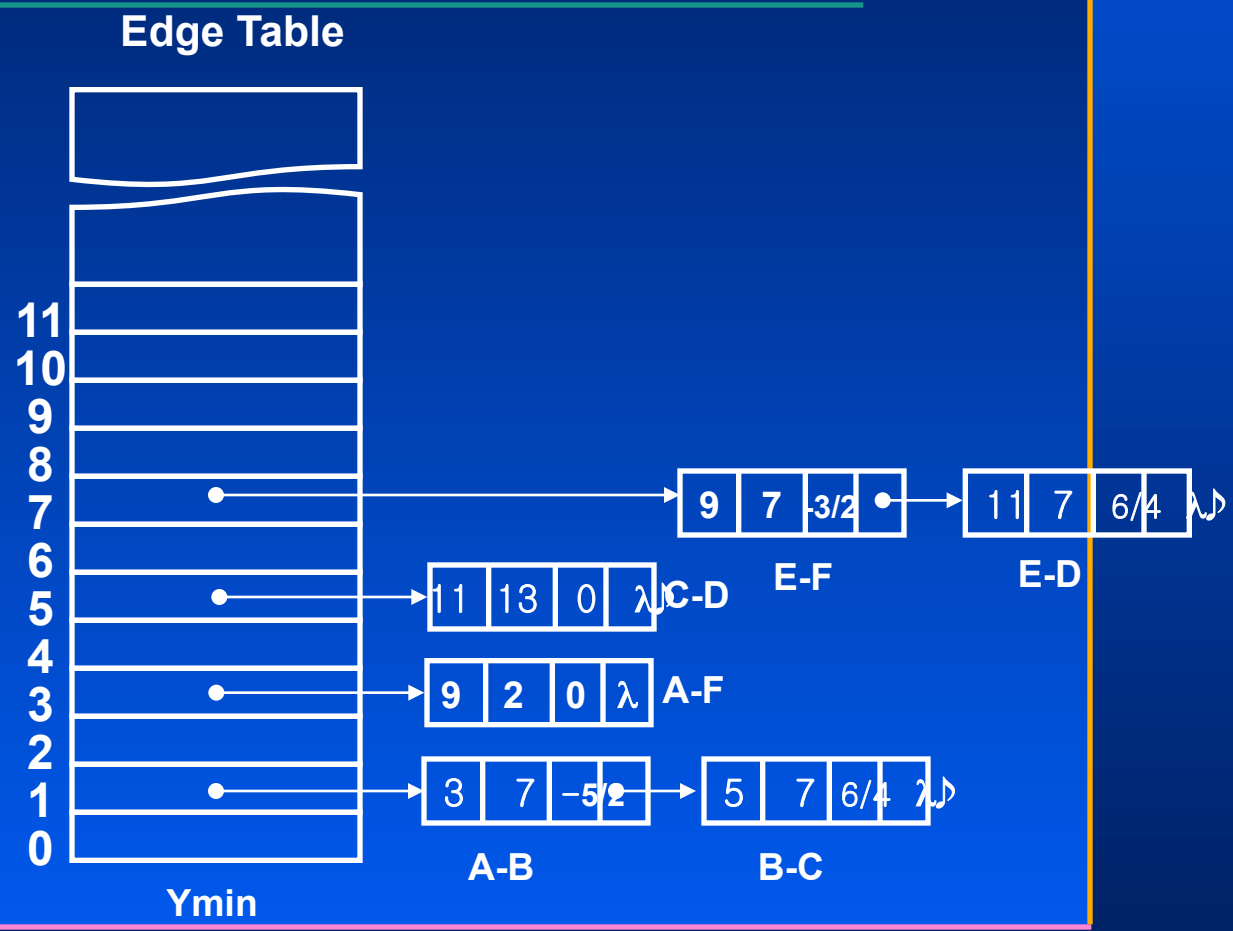
$$x_{k+1} = x_k + 1/m$$


© 2010 James K. Hahn

# Edge Table

| Ymax | Xmin: x coord of bottom end pt | ΔX over Δy |
|---|---|---|

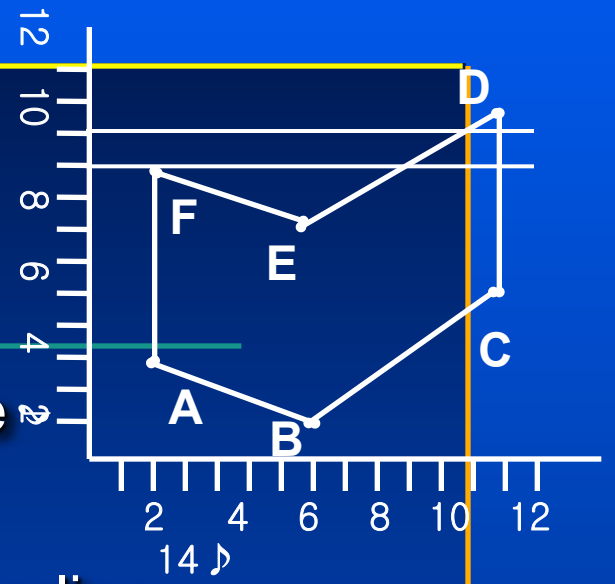


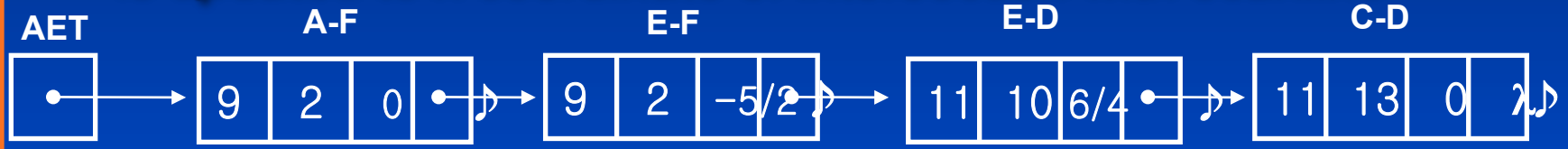



© 2010 James K. Hahn

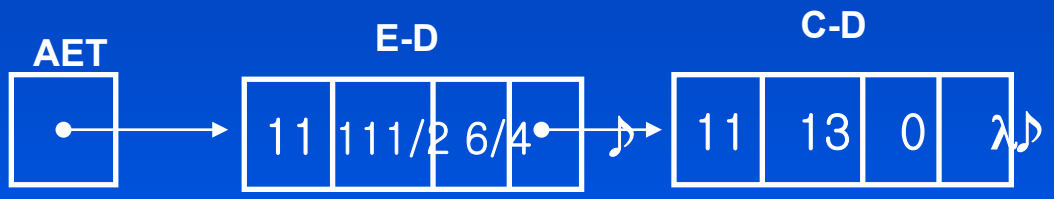
# Active Edge Table




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




(a) Scan line 9




(b) Scan line 10



© 2010 James K. Hahn

# Polygon Scan Conversion Algorithm

- Generally for convex or concave polygon
  - 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


© 2010 James K. Hahn

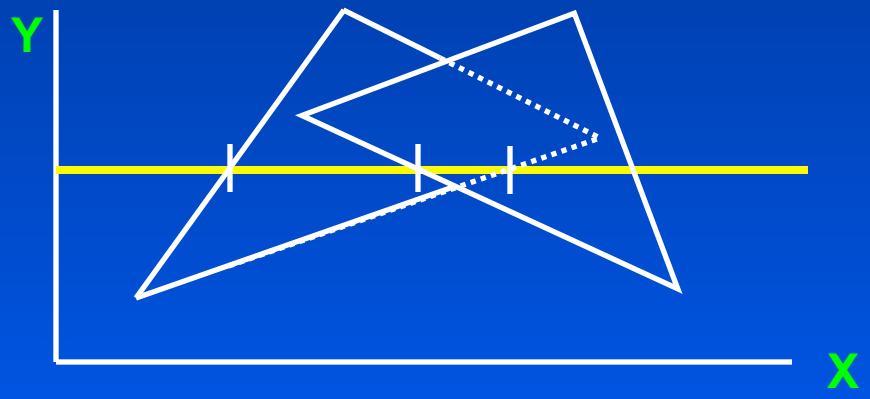
## 3. Repeat until the AET and ET are empty :

- 3.1 move edge(s) from ET bucket y whose ymin = 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 = ymax (leave edges)
- 3.4 For each entry remaining in AET, replace x by x + 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)


© 2010 James K. Hahn

# More than one polygon

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





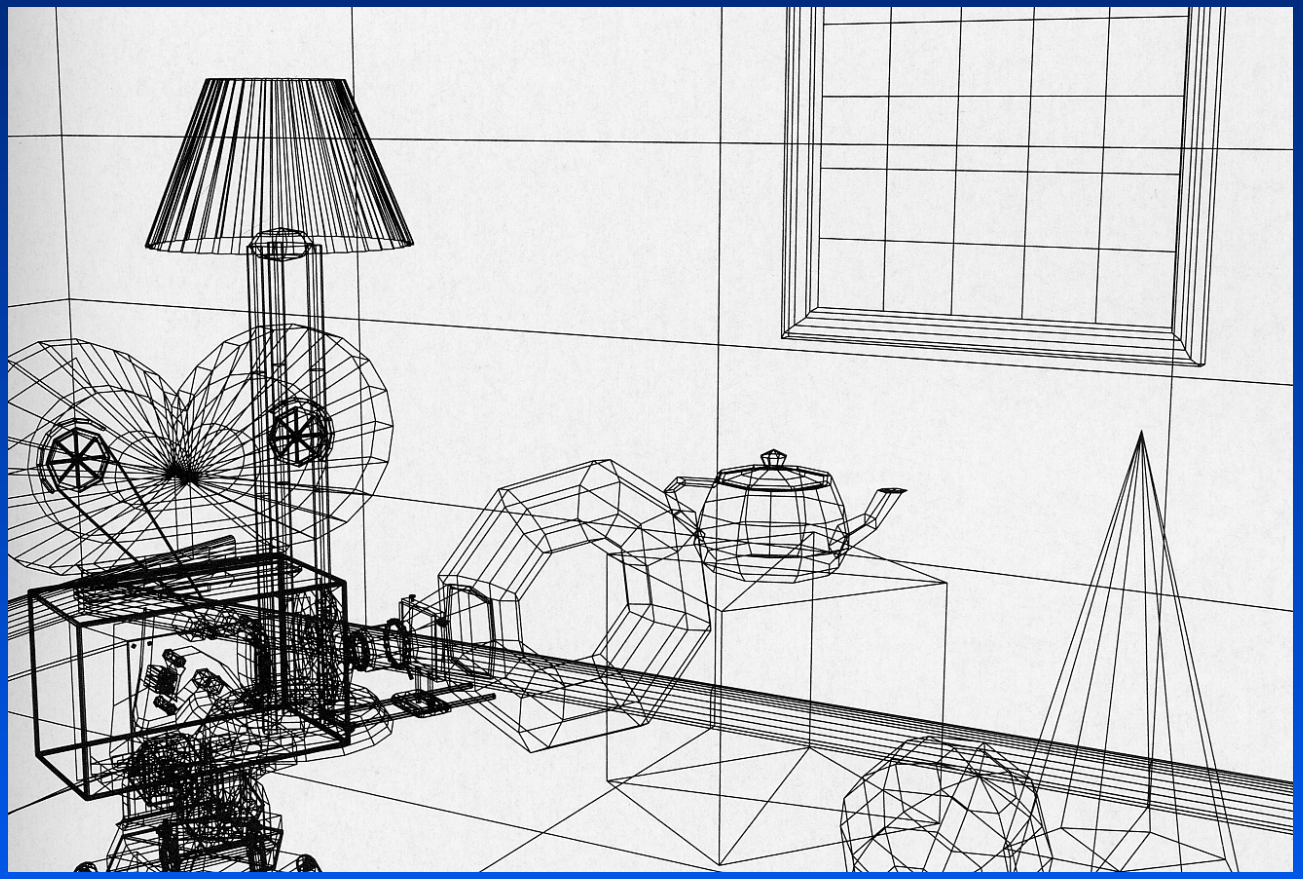
© 2010 James K. Hahn

# 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)


© 2010 James K. Hahn

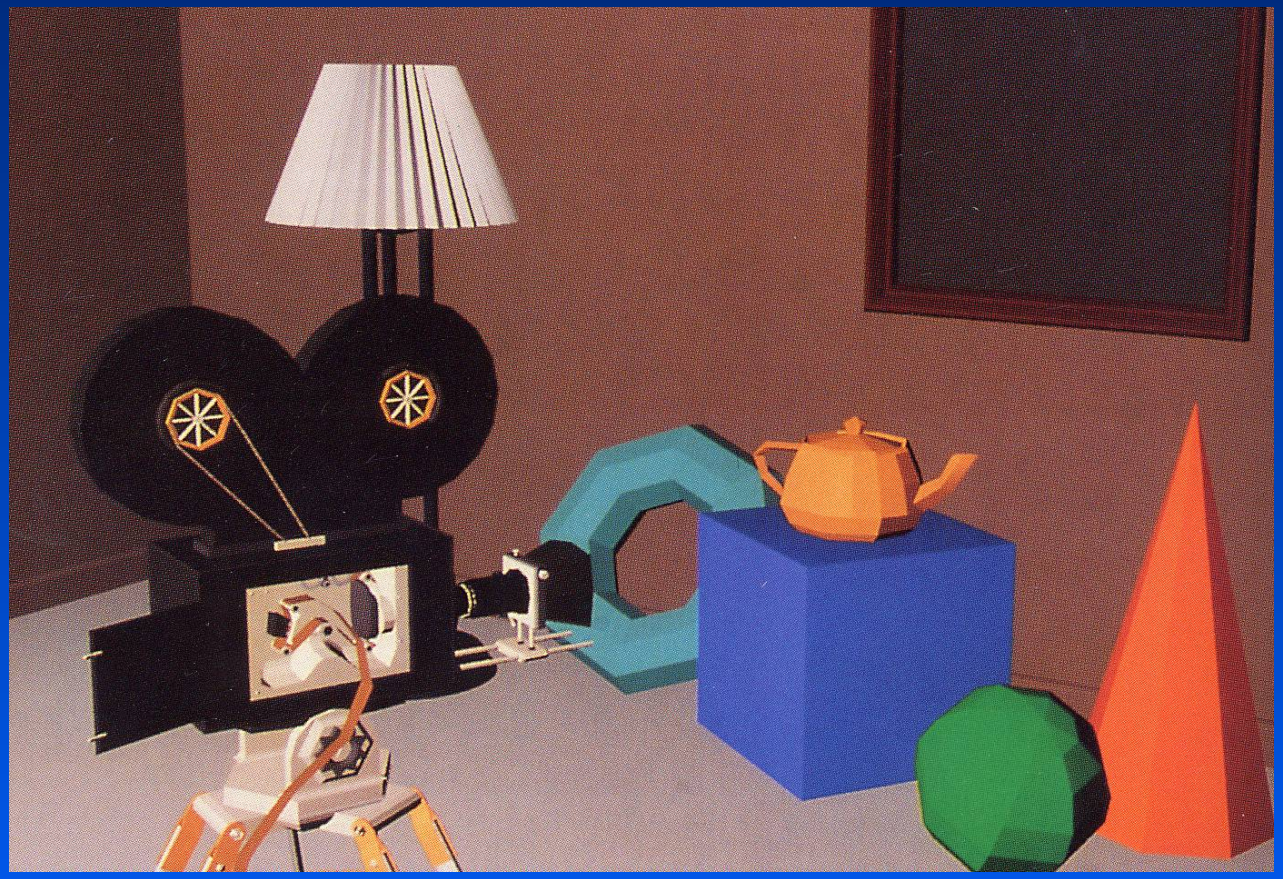
# Wireframe





© 2010 James K. Hahn

# Constant Shading





© 2010 James K. Hahn

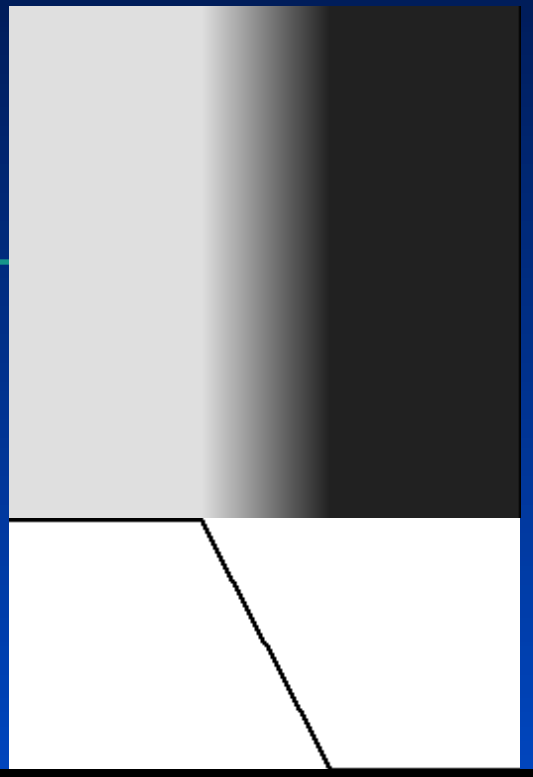
- Single intensity value per polygon
- Gives reasonable results only if
  - Object is a polyhedron: polygon mesh is an approximation to curved surface, faceted look is a problem
  - Light source is far away from the surface (or is parallel) so that N•L is constant over each polygon
  - Viewing position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

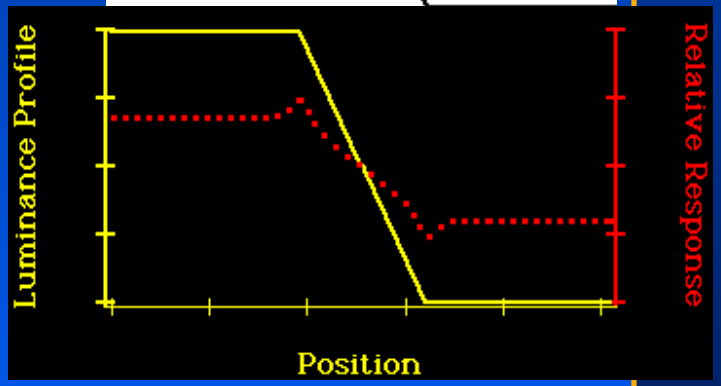

© 2010 James K. Hahn

# 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



© 2010 James K. Hahn

- Constant shading accentuate the facets due to Mach banding



© 2010 James K. Hahn

# Gouraud Shading



© 2010 James K. Hahn

- 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

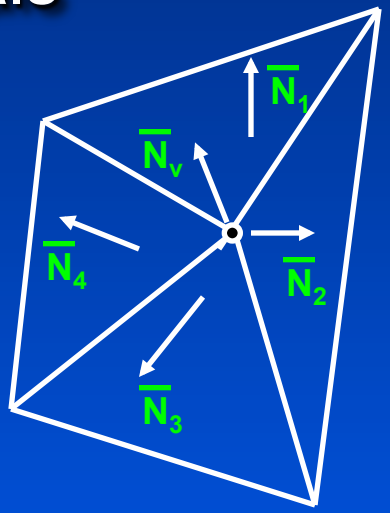

© 2010 James K. Hahn

# 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♪


© 2010 James K. Hahn

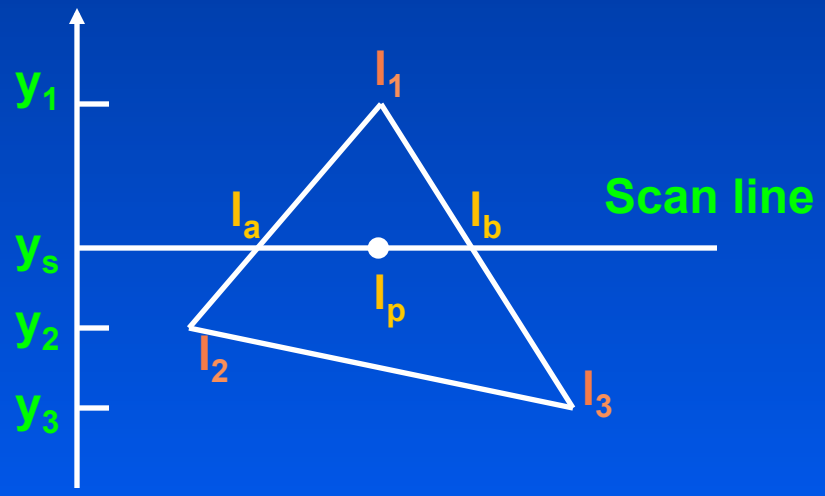
- Vertex normal is average of neighboring polygon normals




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


© 2010 James K. Hahn

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



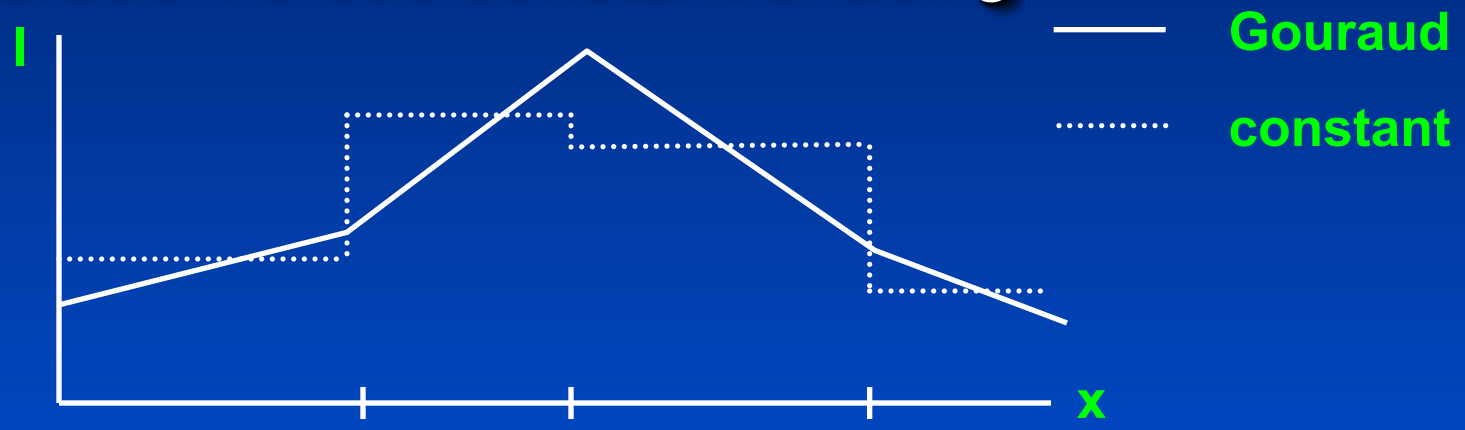

$$l_a = l_1 \frac{y_s - y_2}{y_1 - y_2} + l_2 \frac{y_1 - y_s}{y_1 - y_2}$$

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

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


© 2010 James K. Hahn

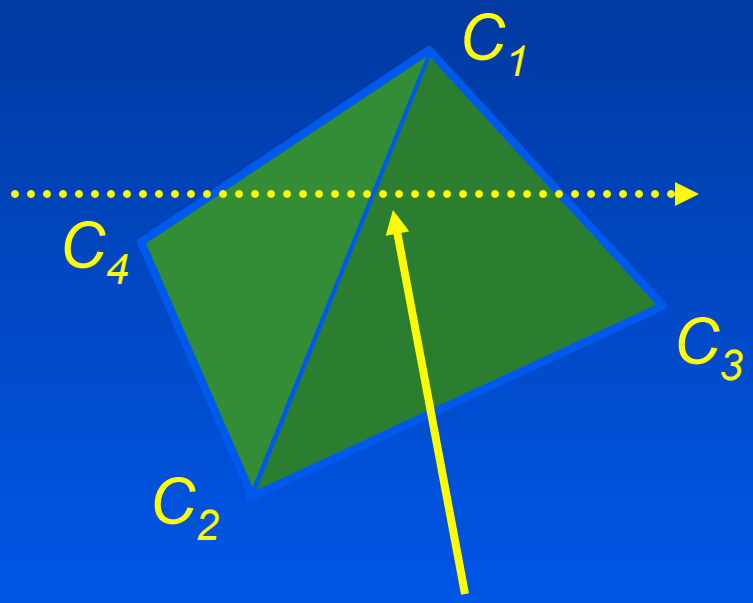
- Gouraud Versus constant shading

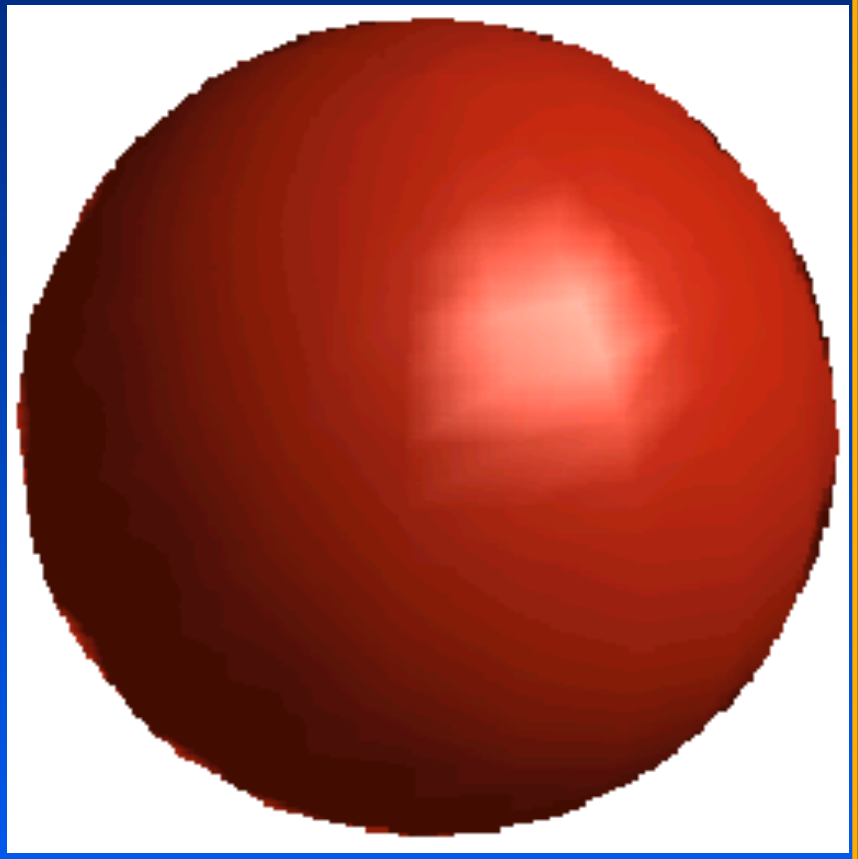



- Integrates nicely with scan line algorithm :
  - $\frac{\Delta I}{\Delta Y}$ is constant along polygon edge


© 2010 James K. Hahn

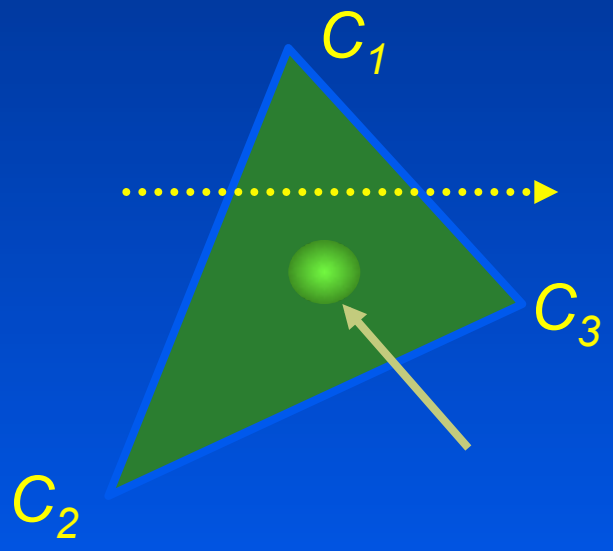
- Notice the Mach banding effects on the highlight




© 2010 James K. Hahn

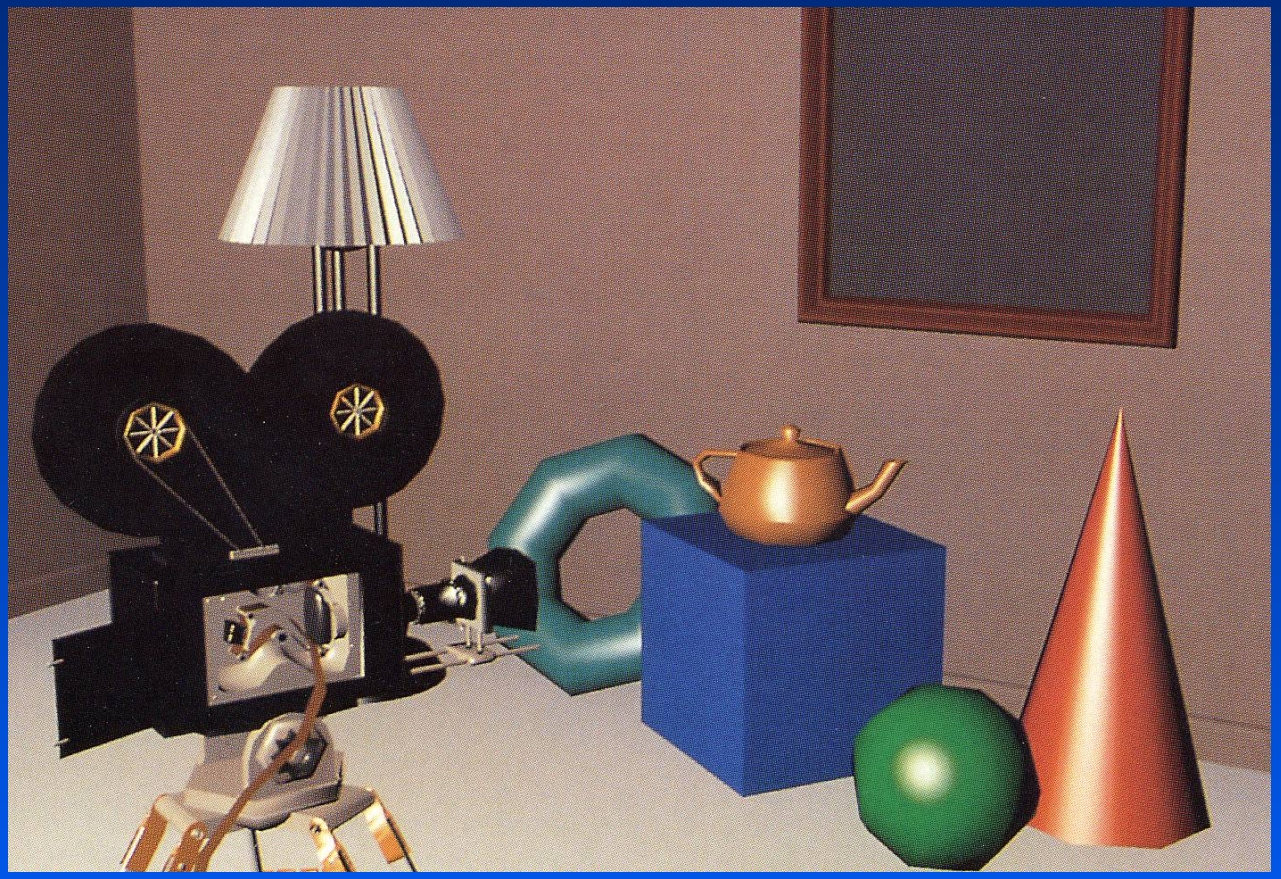
- Highlights that fall inside the polygon (and not on the vertices) are missed





© 2010 James K. Hahn

# Phong Shading





© 2010 James K. Hahn

- 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

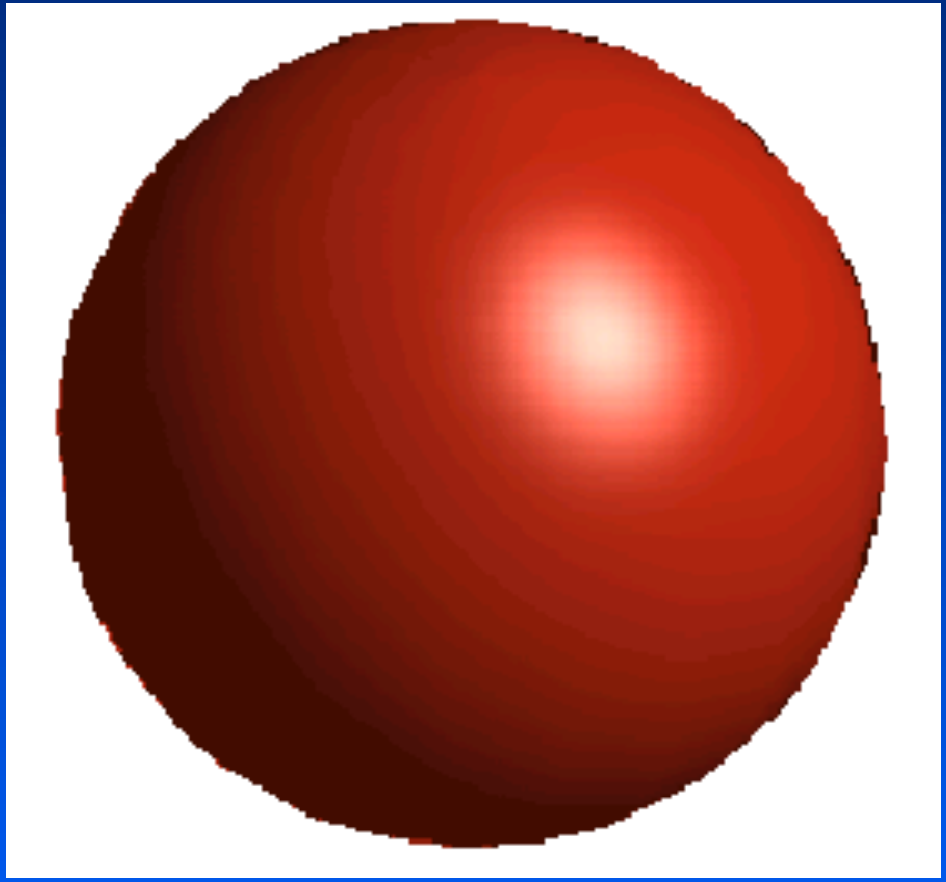

© 2010 James K. Hahn

# 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


© 2010 James K. Hahn

- 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




© 2010 James K. Hahn

# 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

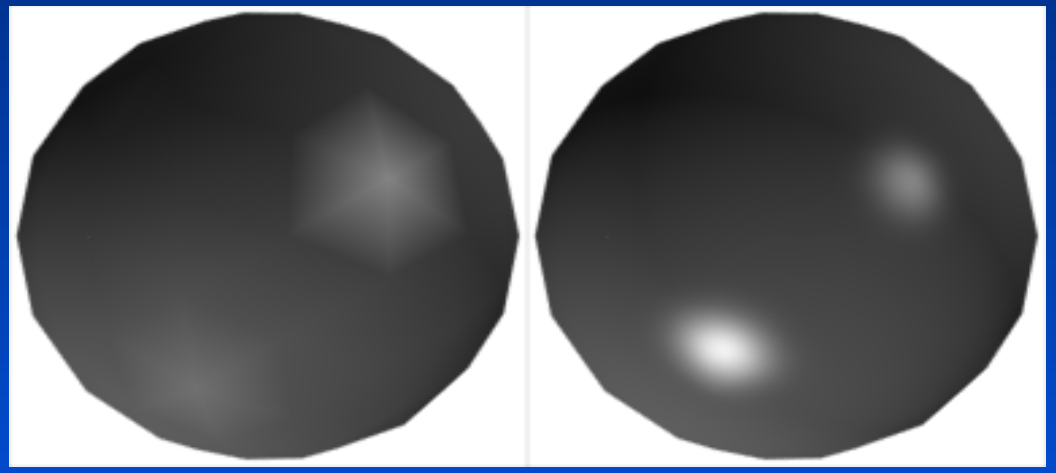

© 2010 James K. Hahn

- 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


© 2010 James K. Hahn

# Problems at the Silhouette

- improved by increasing polygons

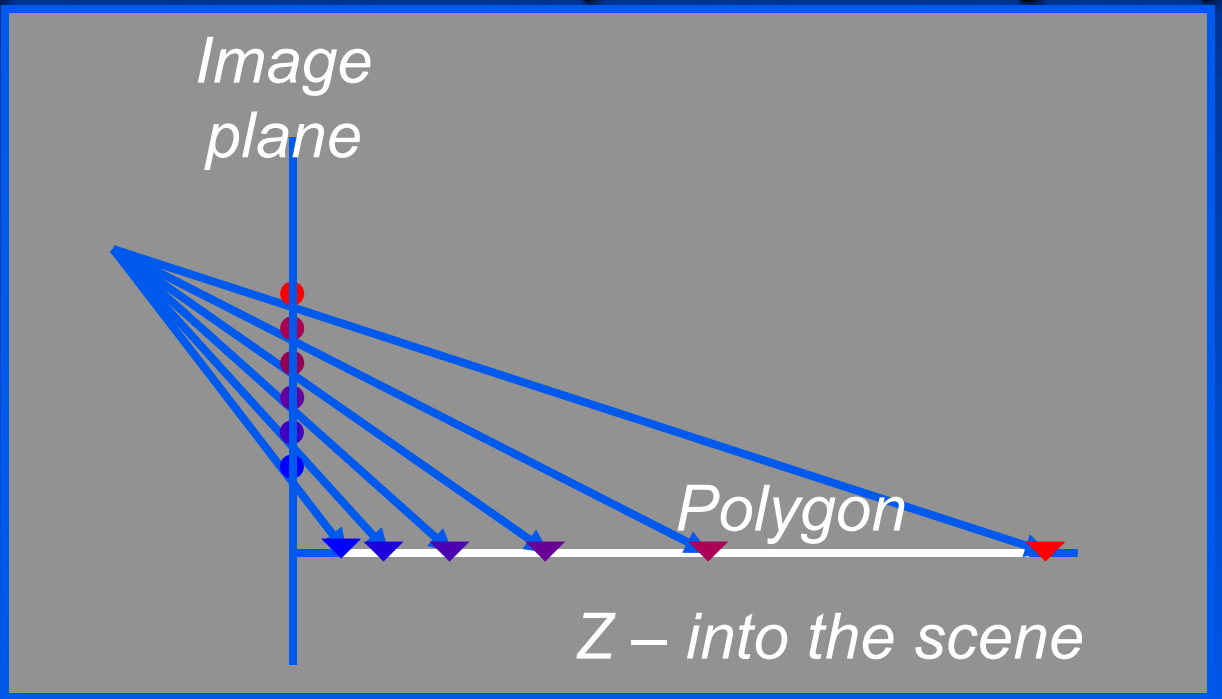


*Gouraud* *Phong*


© 2010 James K. Hahn

# Perspective distortion

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

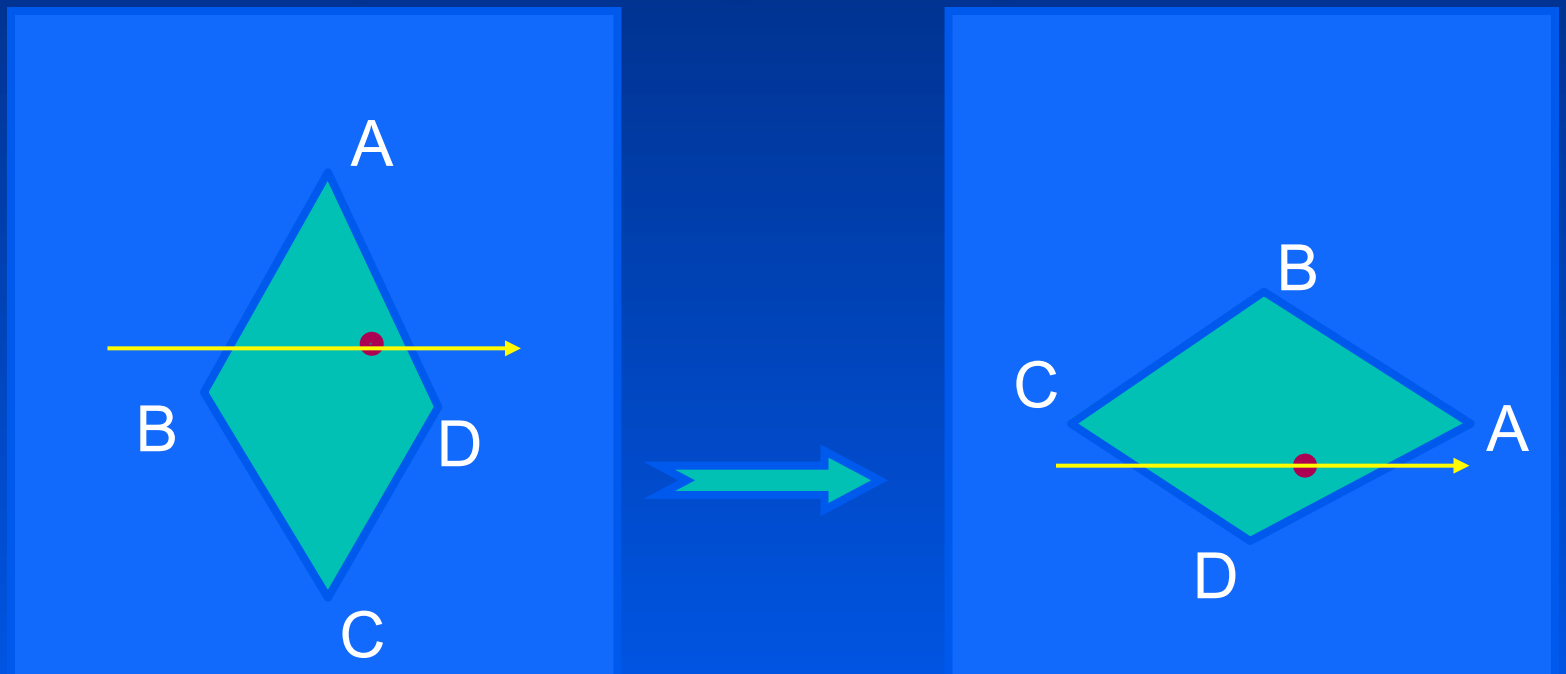




© 2010 James K. Hahn

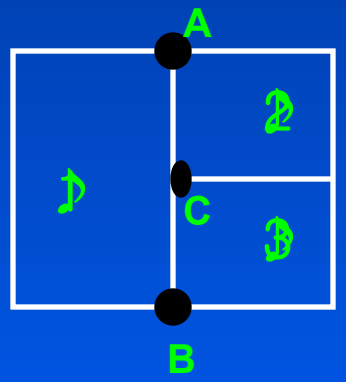
# Orientation dependence

- What is interpolated changes as object is rotated



# Discontinuities due to non-shared vertices

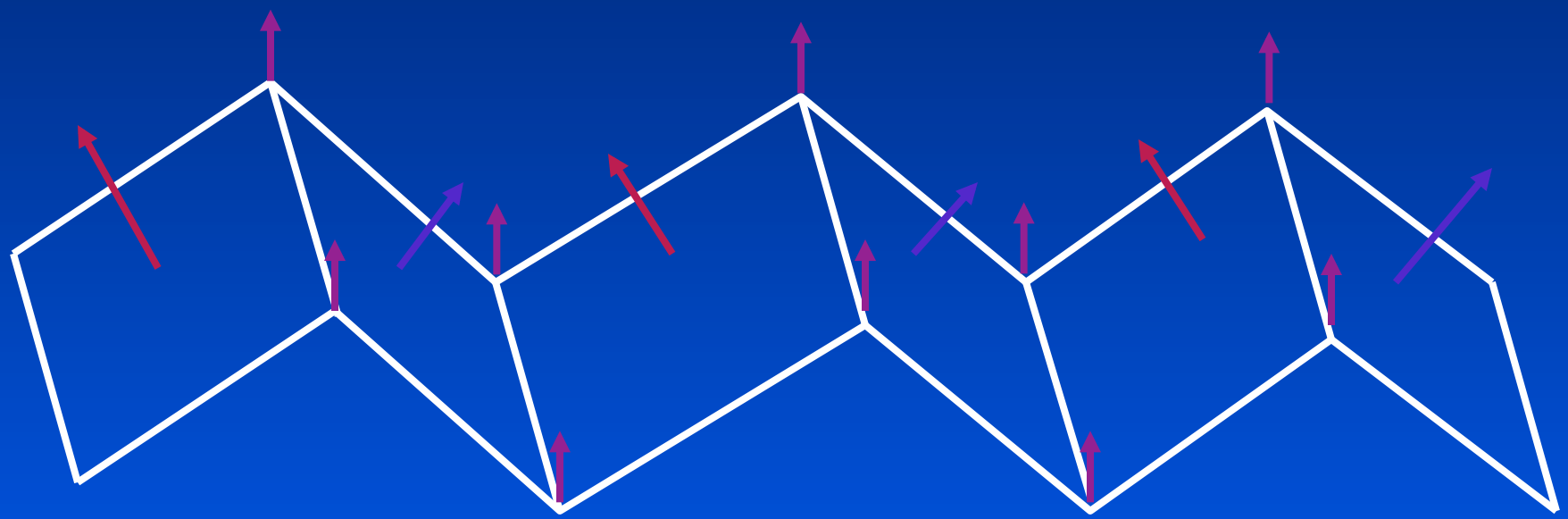
- Near vertex C
  - On polygon 1, interpolated from A and B
  - On polygon 2 and 3, from vertex C
- This kind of polygon usually not allowed





© 2010 James K. Hahn

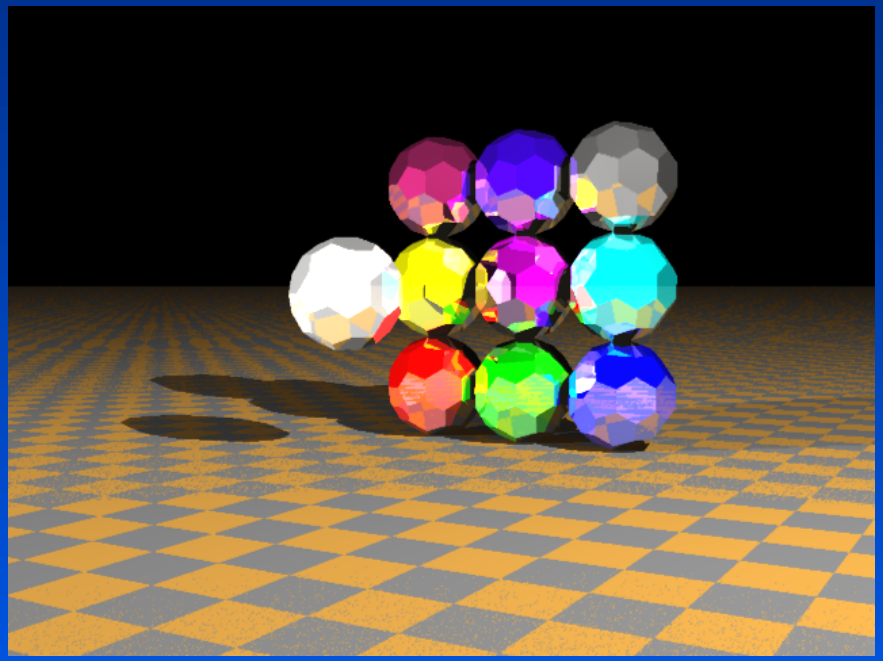
# Error due to averaging normals





© 2010 James K. Hahn

# Next: Visible surface algorithms





© 2010 James K. Hahn