



3D Rasterization II

COS 426



3D Rendering Pipeline

(for direct illumination)

3D Primitives



Modeling Transformation



Lighting



Viewing Transformation



Projection Transformation



Clipping



Viewport Transformation



Scan Conversion

Image



Rasterization

- Scan conversion
 - Determine which pixels to fill
- Shading
 - Determine a color for each filled pixel
- Texture mapping
 - Describe shading variation within polygon interiors
- Visible surface determination
 - Figure out which surface is front-most at every pixel



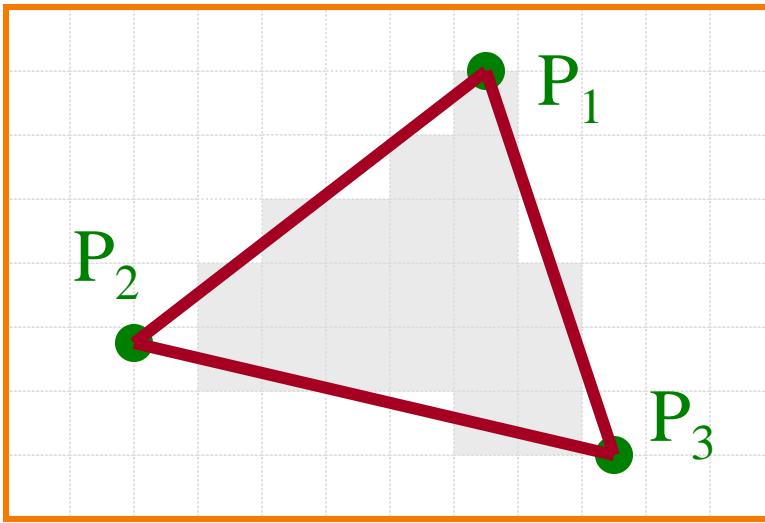
Rasterization

- Scan conversion (last time)
 - Determine which pixels to fill
- Shading
 - Determine a color for each filled pixel
- Texture mapping
 - Describe shading variation within polygon interiors
- Visible surface determination
 - Figure out which surface is front-most at every pixel



Shading

- How do we choose a color for each filled pixel?

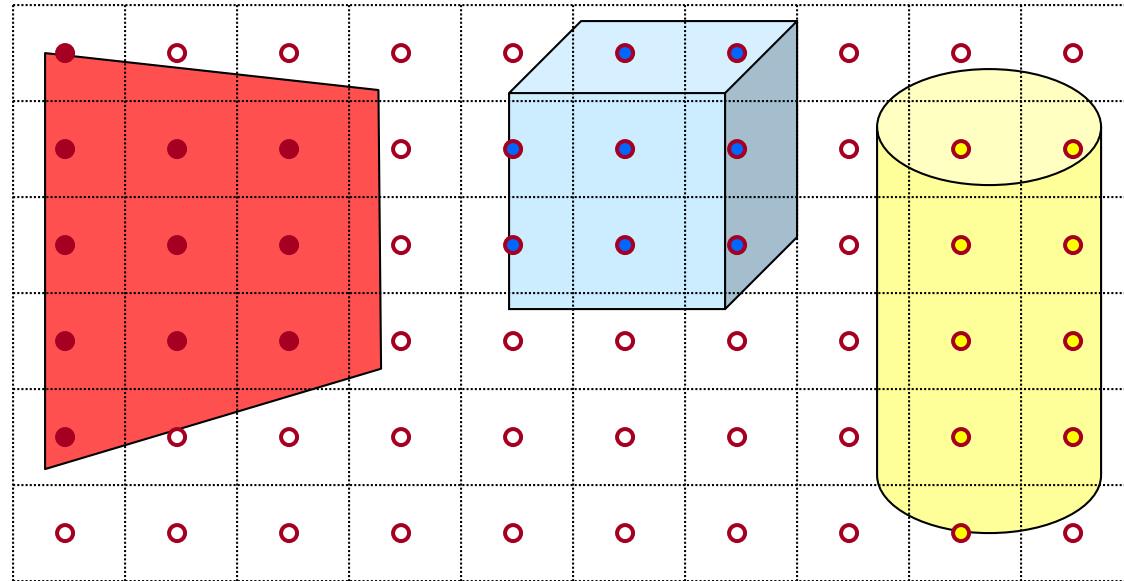


Emphasis on methods that can
be implemented in hardware



Ray Casting

- Simplest shading approach is to perform independent lighting calculation for every pixel

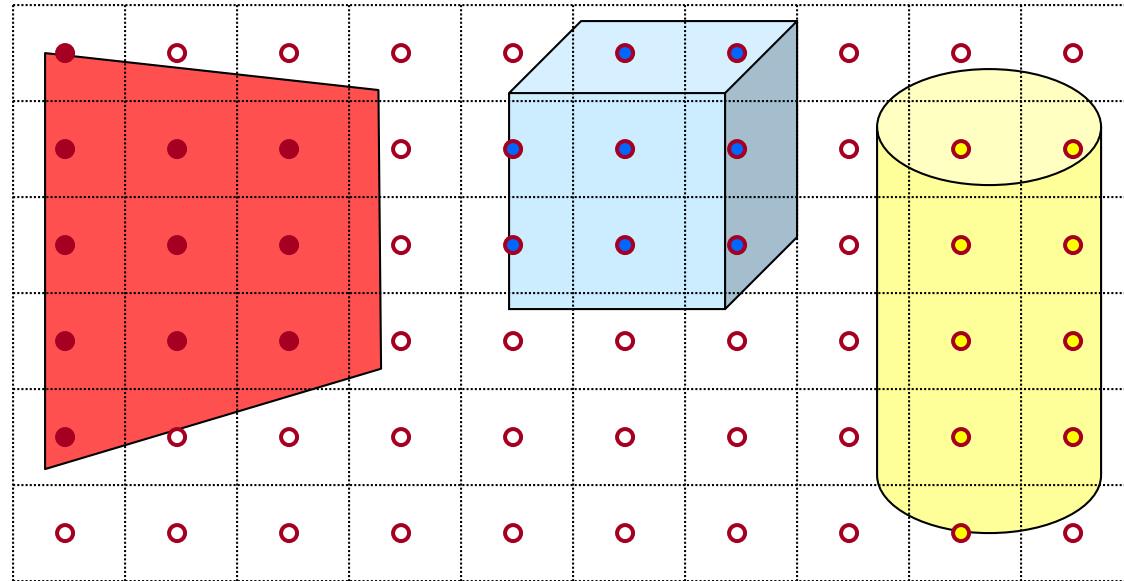


$$I = I_E + K_A I_{AL} + \sum_i \left(K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i \right)$$



Polygon Shading

- Can take advantage of spatial coherence
 - Illumination calculations for pixels covered by same primitive are related to each other



$$I = I_E + K_A I_{AL} + \sum_i \left(K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i \right)$$



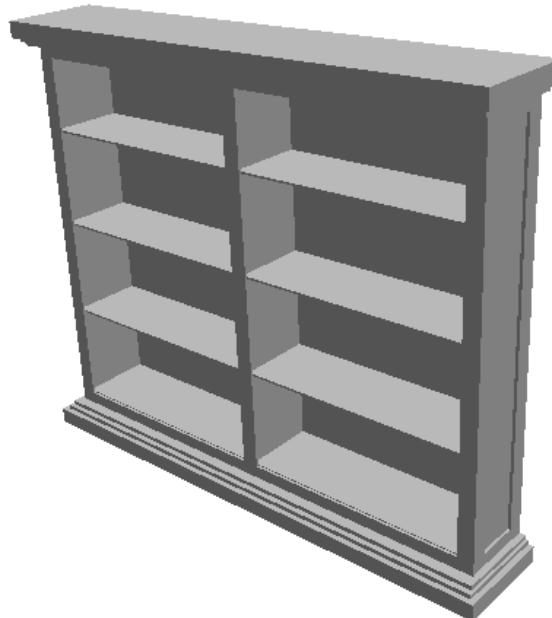
Polygon Shading Algorithms

- **Flat Shading**
- Gouraud Shading
- Phong Shading



Flat Shading

- What if a faceted object is illuminated only by directional light sources and is either diffuse or viewed from infinitely far away

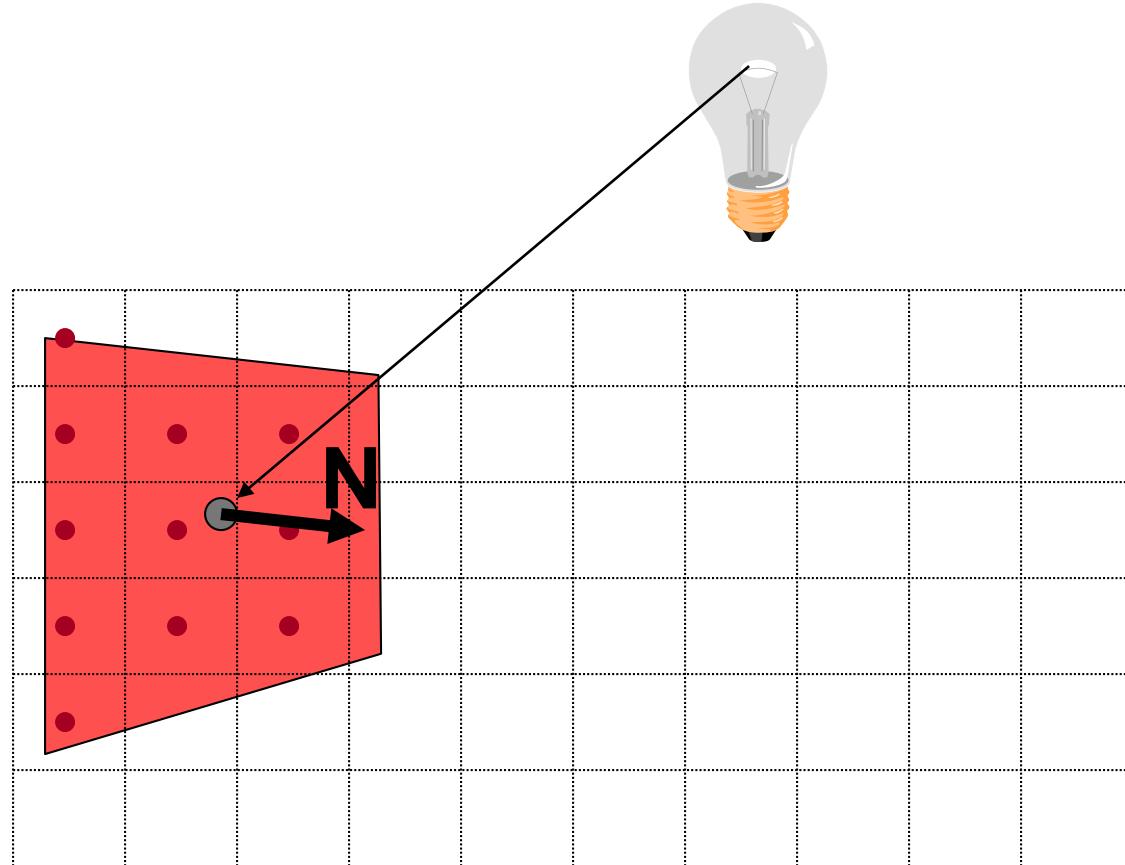


$$I = I_E + K_A I_{AL} + \sum_i \left(K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i \right)$$



Flat Shading

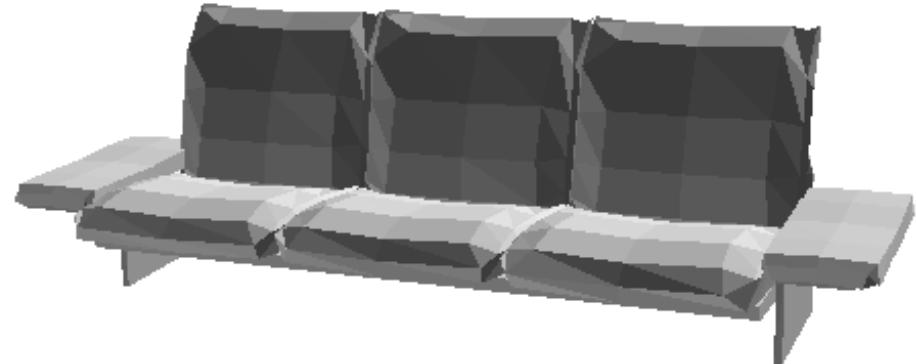
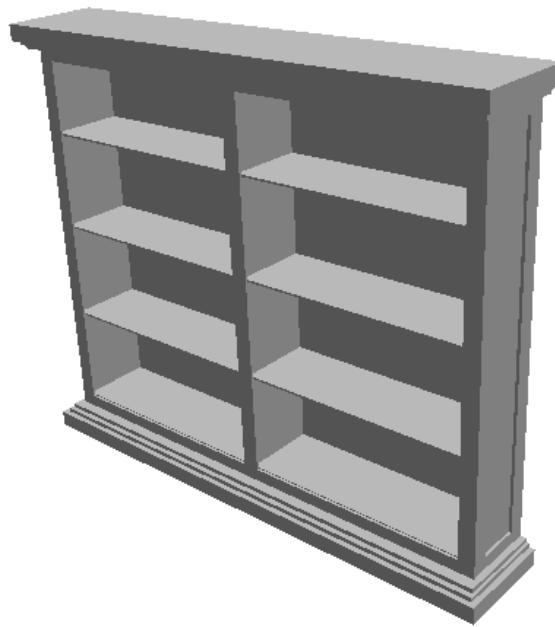
- One illumination calculation per polygon
 - Assign all pixels inside each polygon the same color





Flat Shading

- Objects look like they are composed of polygons
 - OK for polyhedral objects
 - Not so good for smooth surfaces





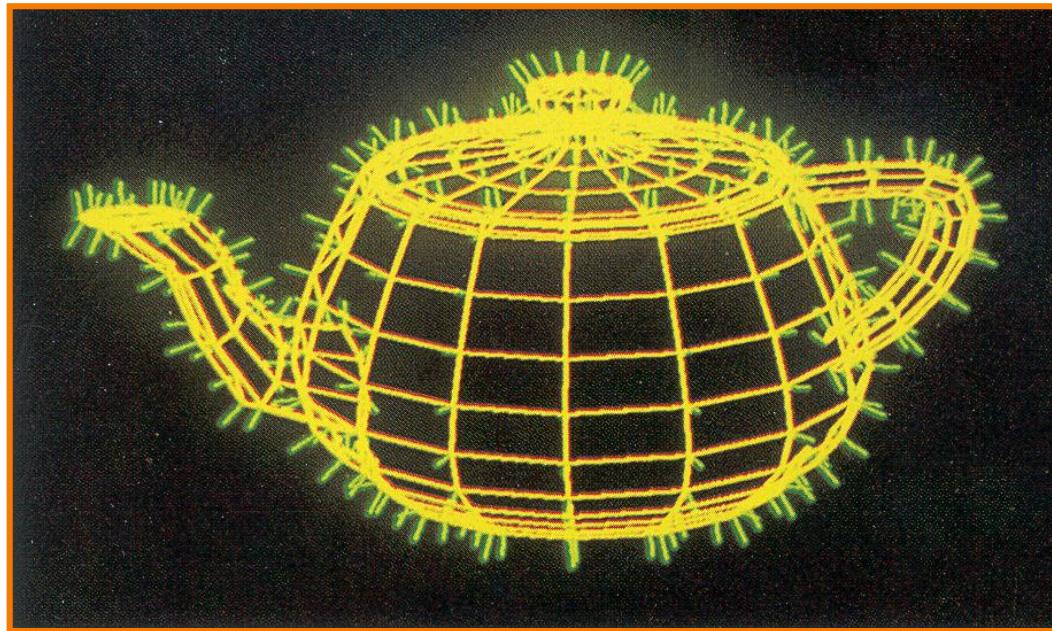
Polygon Shading Algorithms

- Flat Shading
- **Gouraud Shading**
- Phong Shading



Gouraud Shading

- What if smooth surface is represented by polygonal mesh with a normal at each vertex?



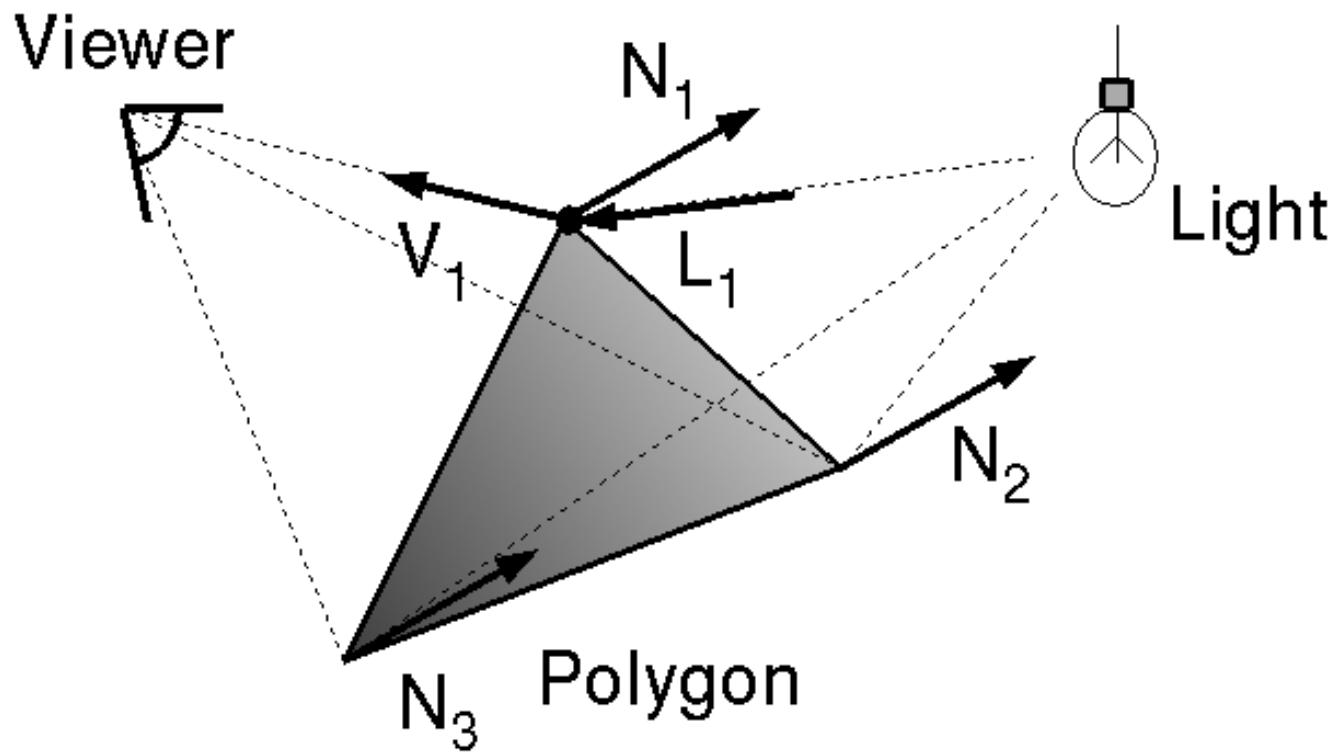
Watt Plate 7

$$I = I_E + K_A I_{AL} + \sum_i \left(K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i \right)$$



Gouraud Shading

- Method 1: One lighting calculation per vertex
 - Assign pixels inside polygon by interpolating colors computed at vertices

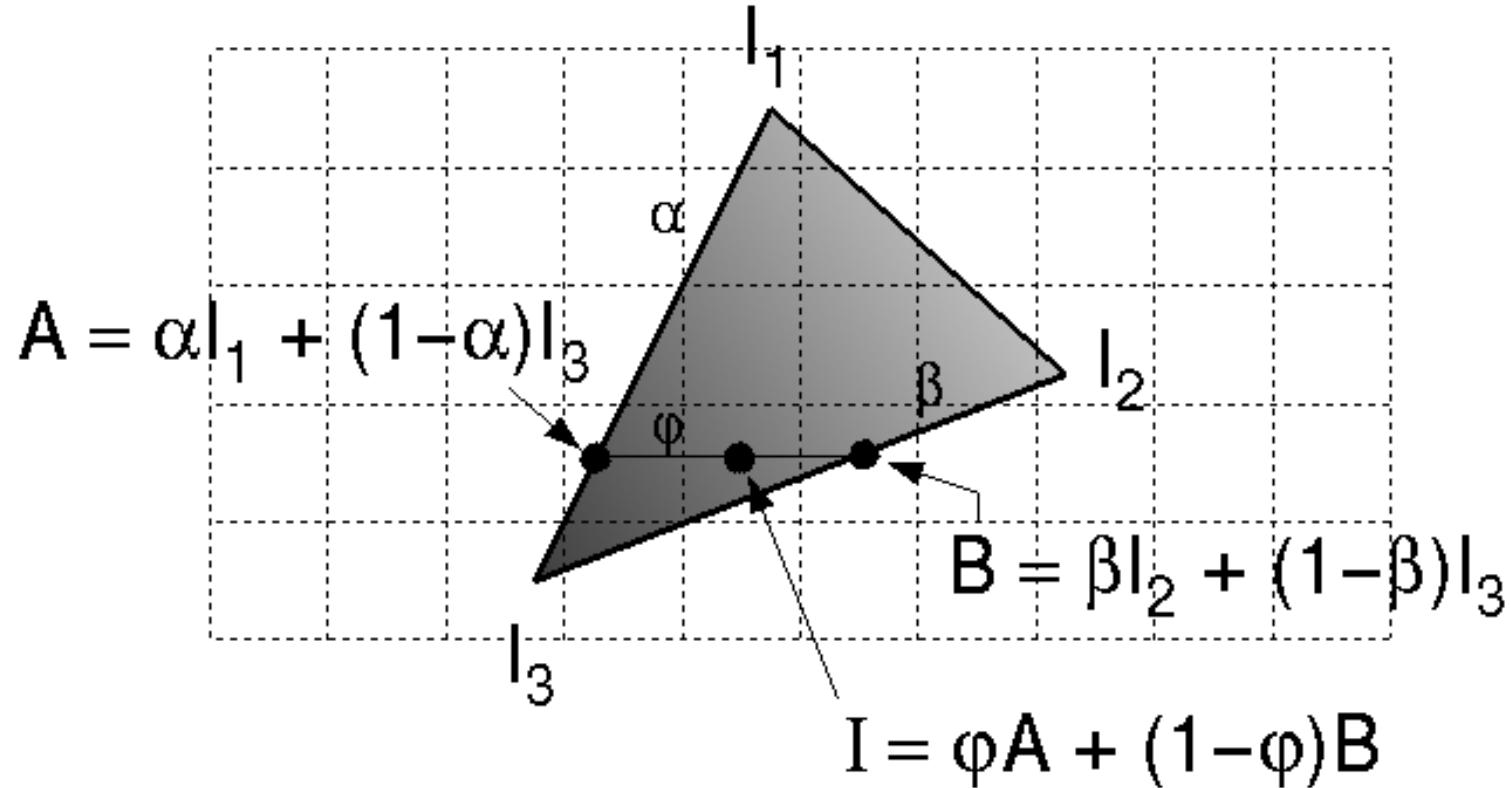




Gouraud Shading

Bilinear interpolation of colors at vertices

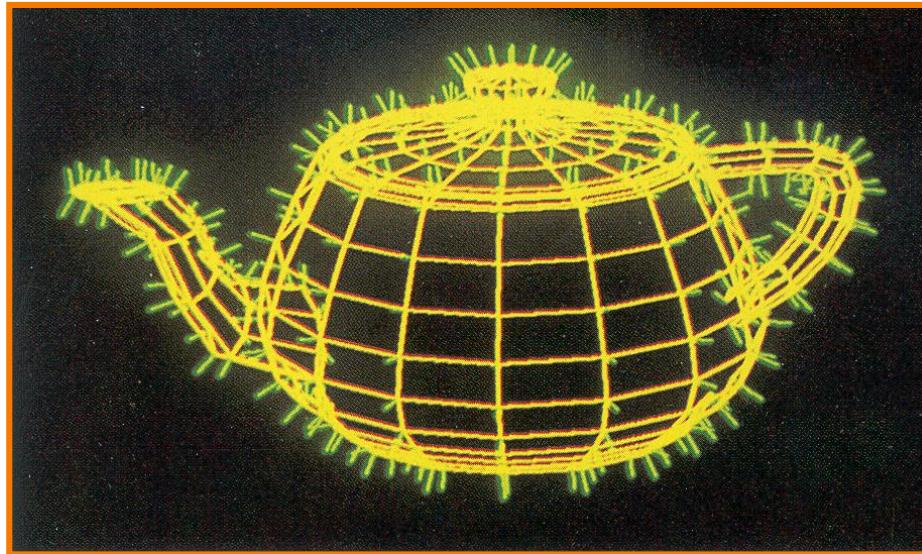
- down and across scan lines = barycentric coords





Gouraud Shading

- Smooth shading over adjacent polygons
 - Curved surfaces
 - Illumination highlights
 - Soft shadows

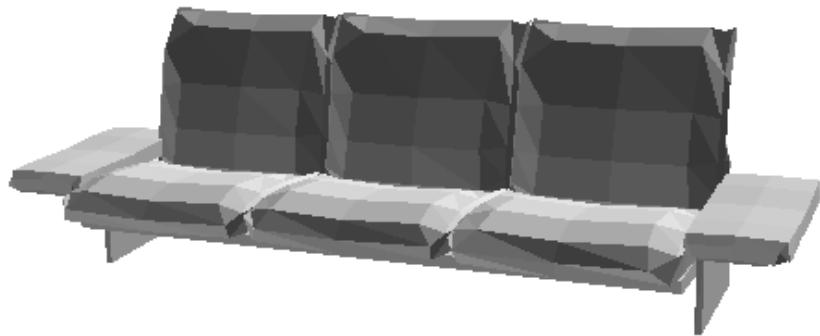


Mesh with shared normals at vertices

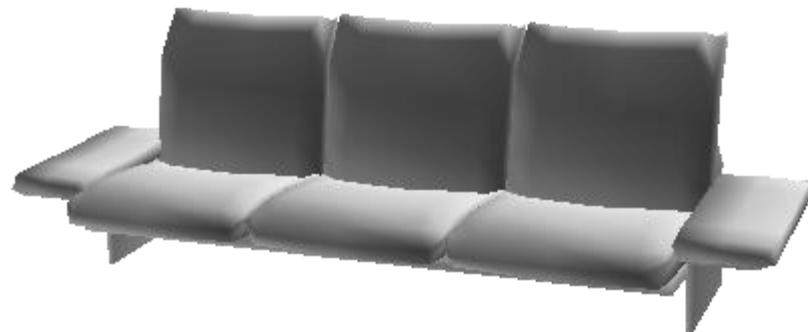


Gouraud Shading

- Produces smoothly shaded polygonal mesh
 - Piecewise linear approximation
 - Need fine mesh to capture subtle lighting effects



Flat Shading



Gouraud Shading



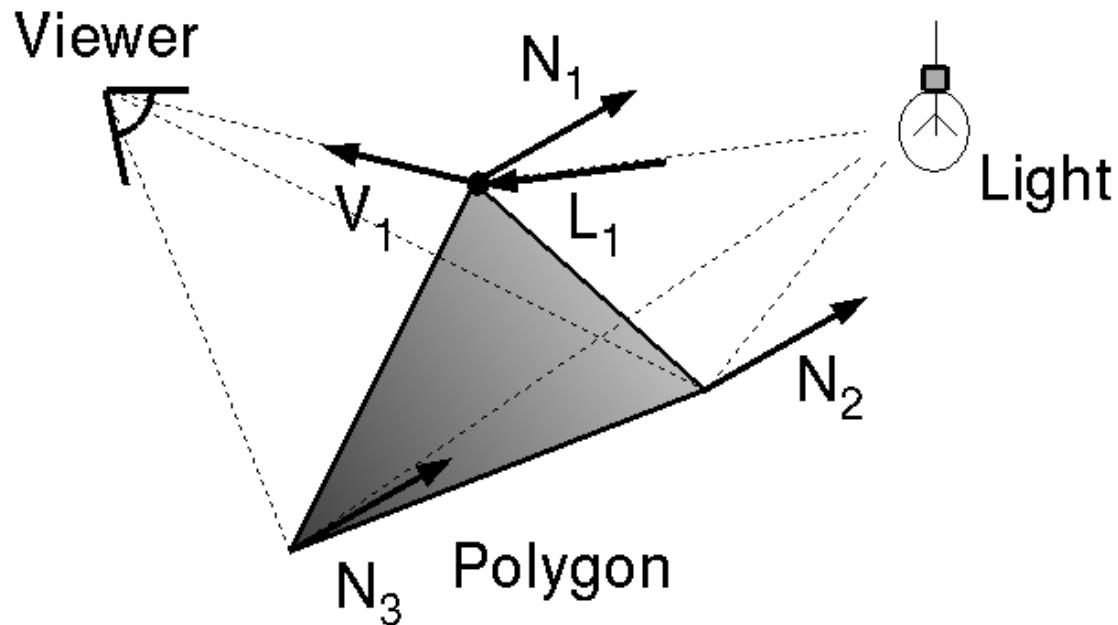
Polygon Shading Algorithms

- Flat Shading
- Gouraud Shading
- **Phong Shading** (\neq Phong reflectance model)



Phong Shading

- What if polygonal mesh is too coarse to capture illumination effects in polygon interiors?

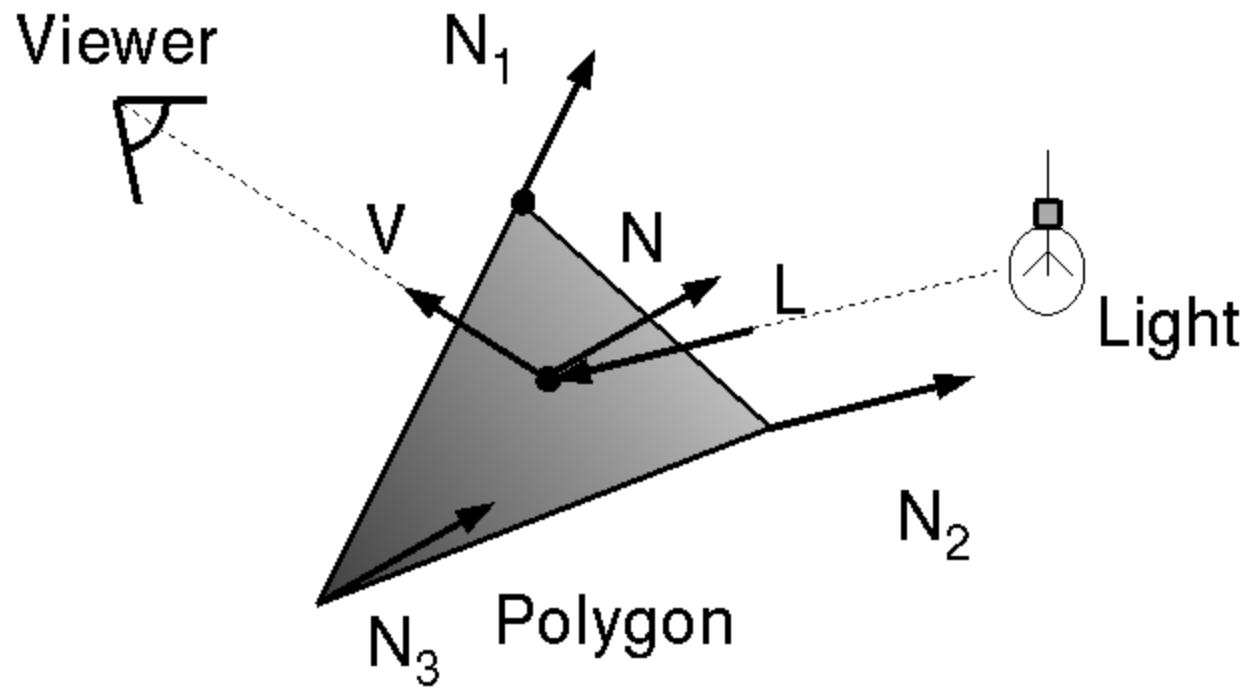


$$I = I_E + K_A I_{AL} + \sum_i \left(K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i \right)$$



Phong Shading

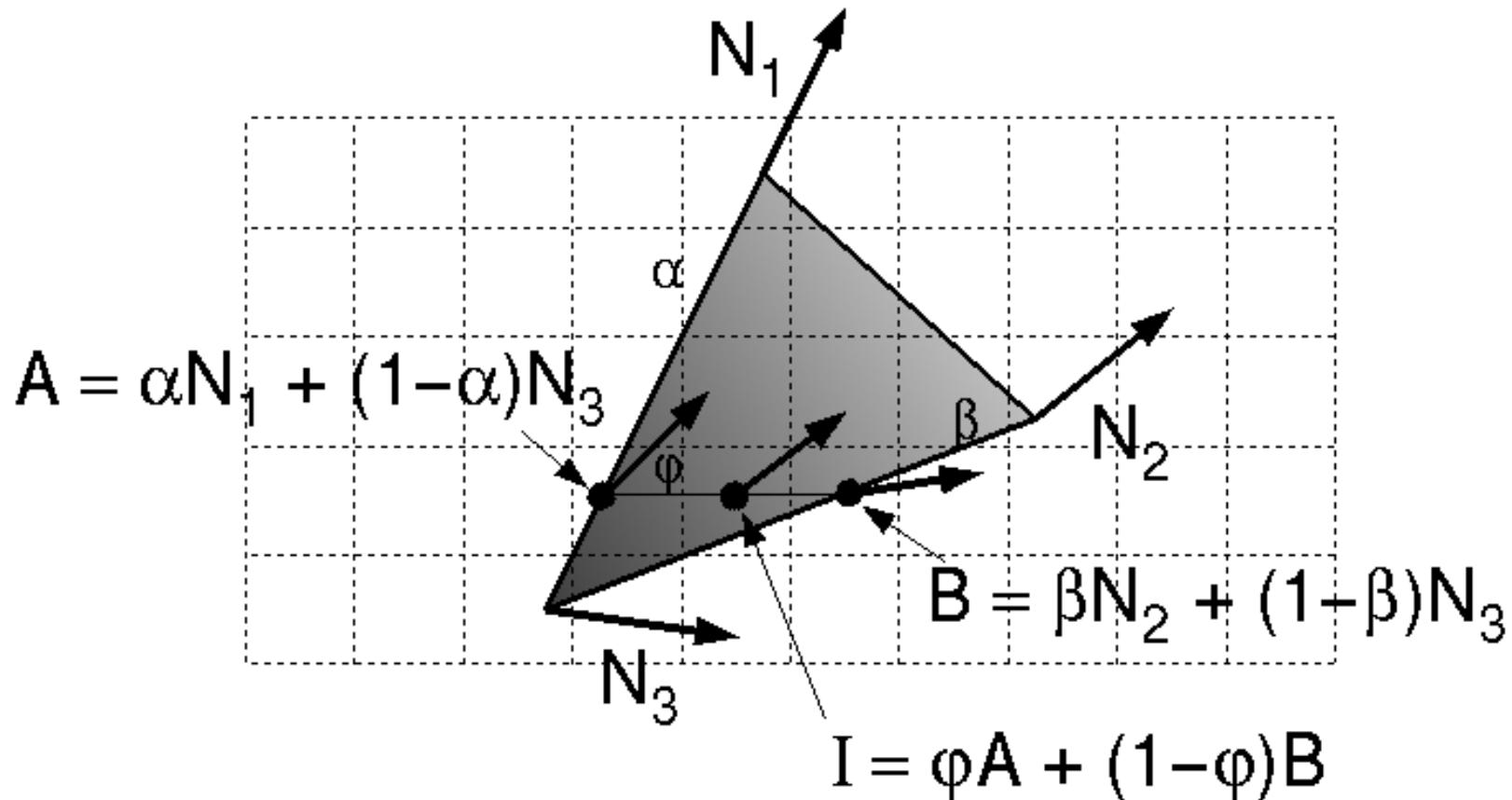
- One lighting calculation per pixel
 - Approximate surface normals for points inside polygons by bilinear interpolation of normals from vertices





Phong Shading

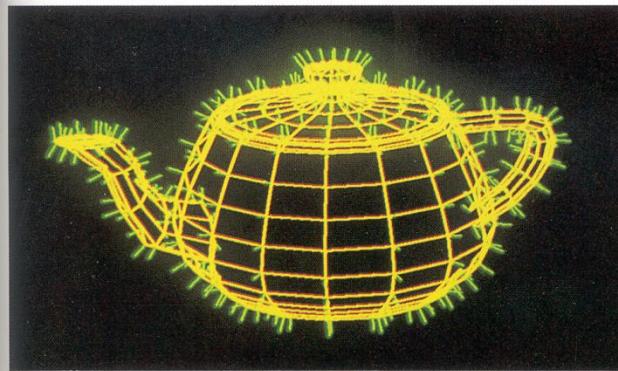
Bilinear interpolation of surface normals at vertices



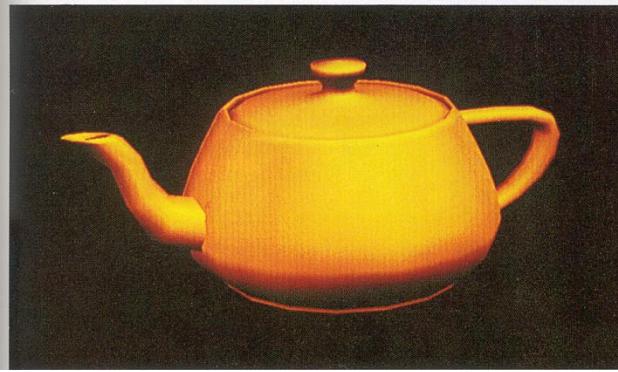


Polygon Shading Algorithms

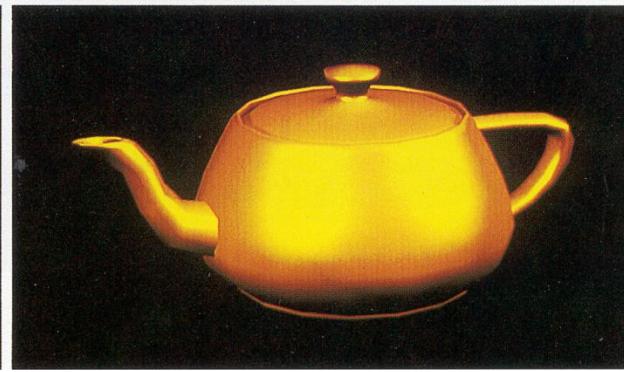
Wireframe



Flat



Gouraud

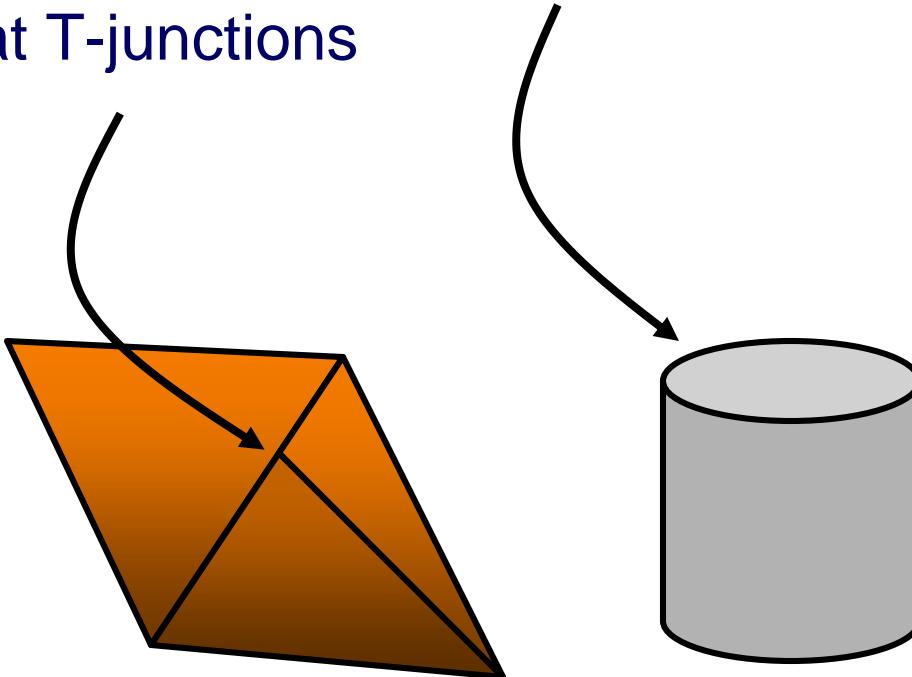


Phong



Shading Issues

- Problems with interpolated shading:
 - Polygonal silhouettes still obvious
 - Perspective distortion (due to screen-space interpolation)
 - Problems computing shared vertex normals
 - Problems at T-junctions





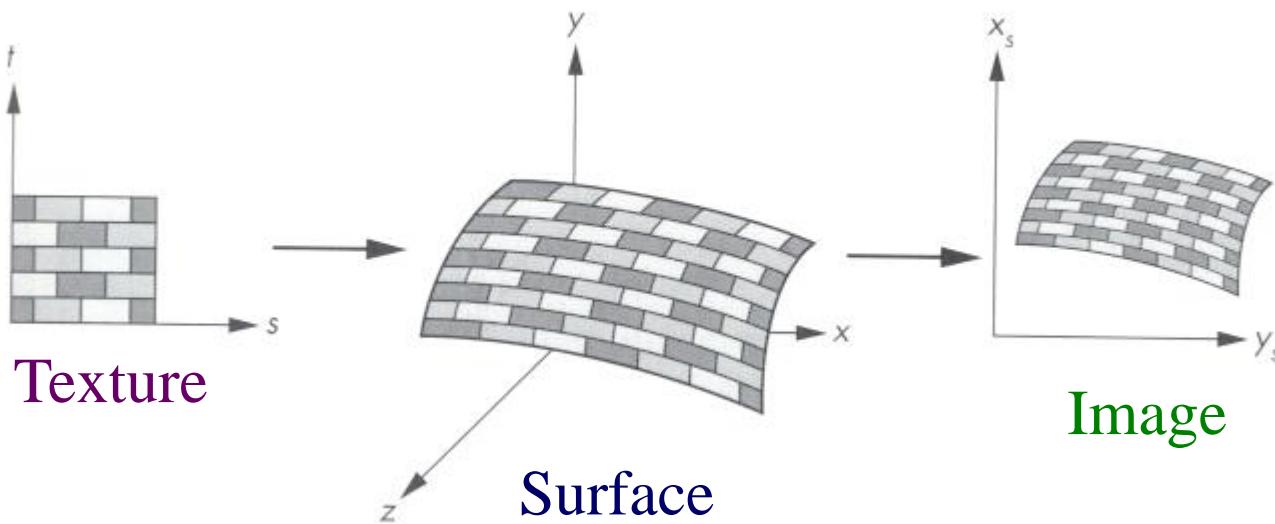
Rasterization

- Scan conversion
 - Determine which pixels to fill
 - Shading
 - Determine a color for each filled pixel
- Texture mapping
- Describe shading variation within polygon interiors
- Visible surface determination
 - Figure out which surface is front-most at every pixel



Textures

- Describe color variation in interior of 3D polygon
 - When scan converting a polygon, vary pixel colors according to values fetched from a texture image



Angel Figure 9.3



Surface Textures

- Add visual detail to surfaces of 3D objects



Polygonal model



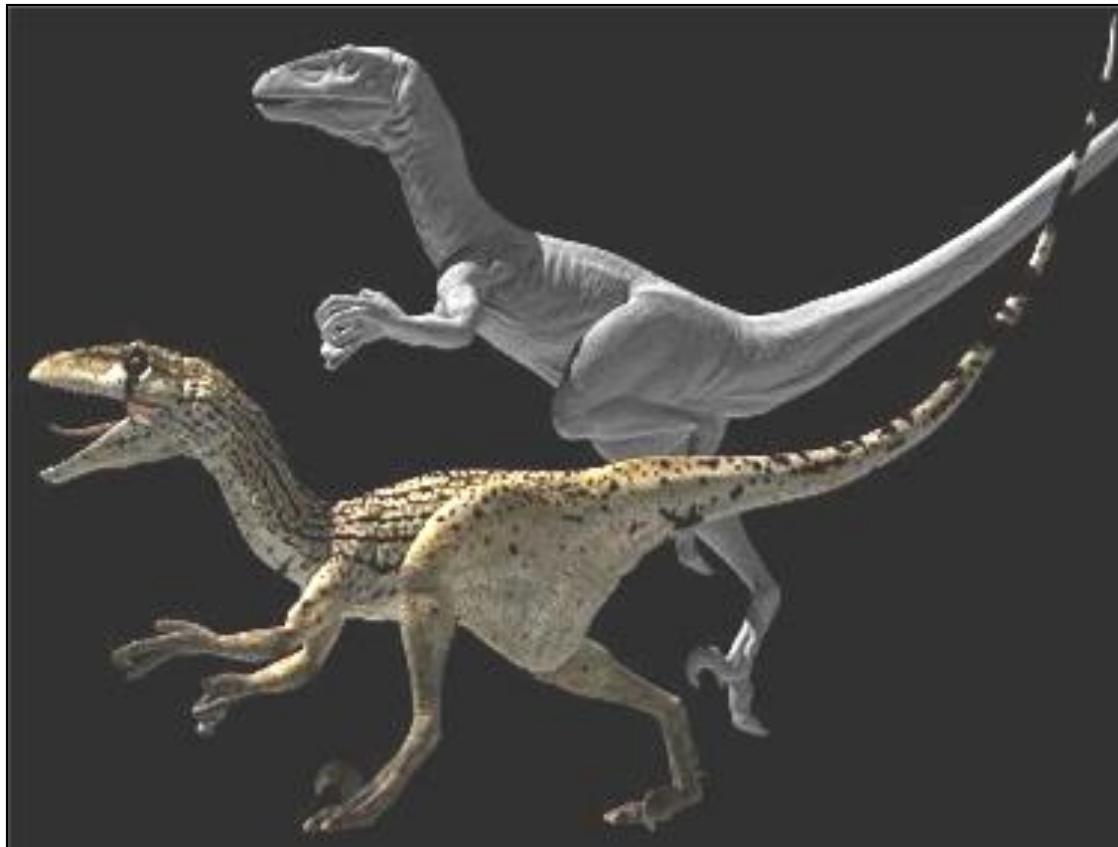
With surface texture





Textures

- Add visual detail to surfaces of 3D objects

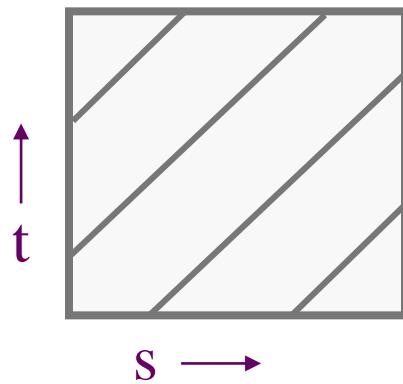


[Daren Horley]

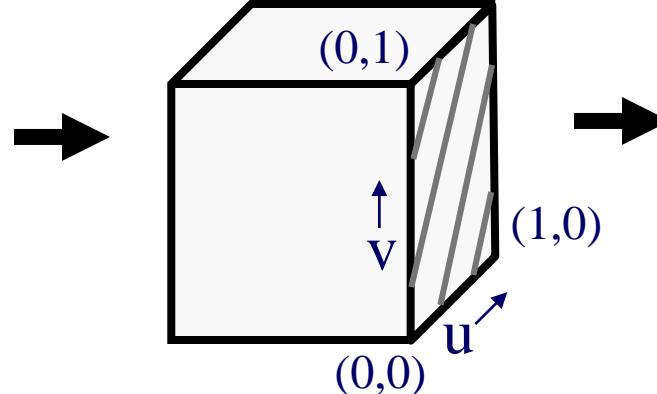


Texture Mapping

- Steps:
 - Define texture
 - Specify mapping from texture to surface
 - Look up texture values during scan conversion



Texture
Coordinate
System



Modeling
Coordinate
System

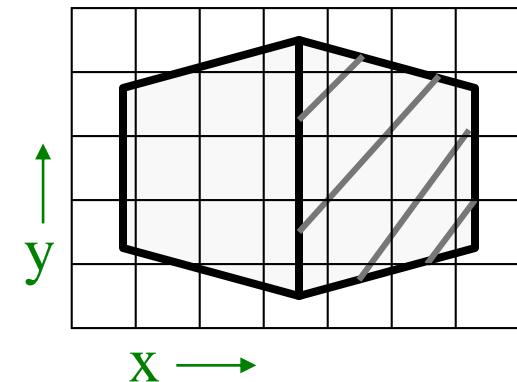
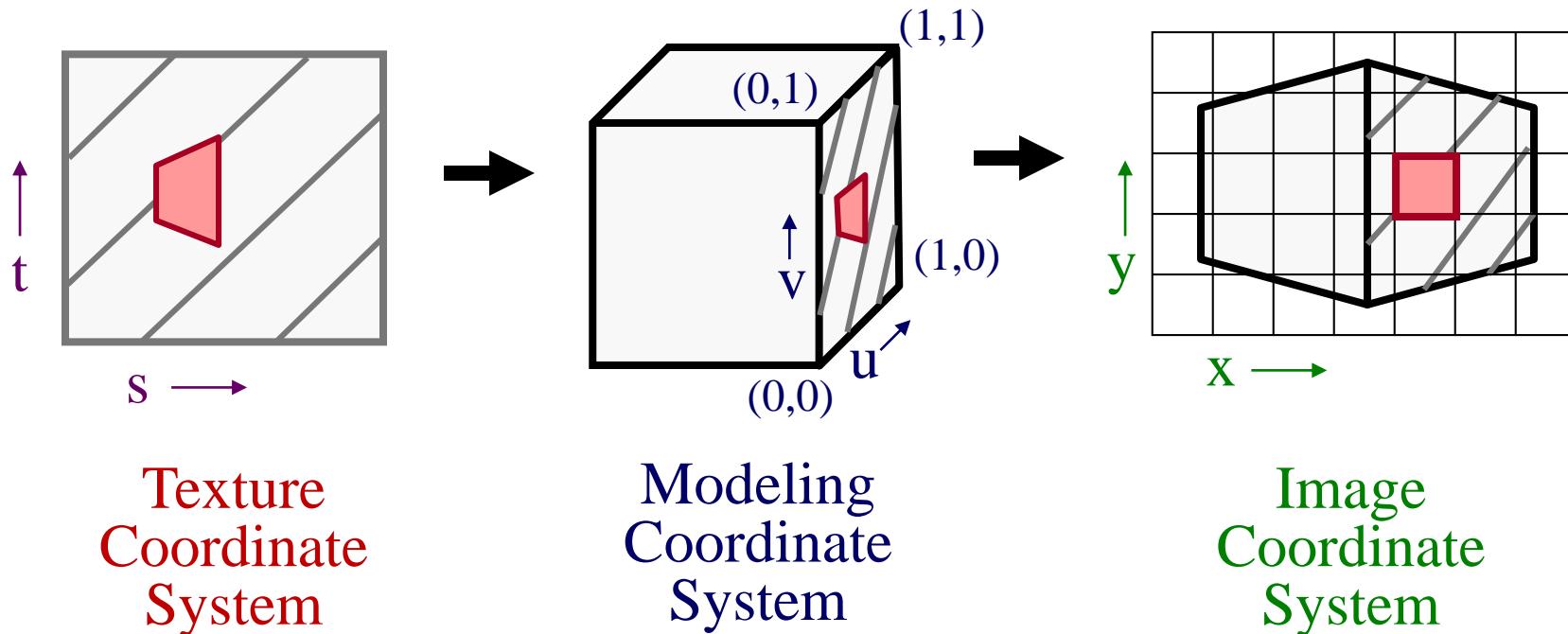


Image
Coordinate
System



Texture Mapping

- When scan converting, map from ...
 - image coordinate system (x,y) to
 - modeling coordinate system (u,v) to
 - texture image (s,t)





Texture Overview

- Texture mapping stages
 - Parameterization
 - Mapping
 - Filtering
- Texture mapping applications
 - Modulation textures
 - Illumination mapping
 - Bump mapping
 - Environment mapping
 - Image-based rendering
 - Non-photorealistic rendering

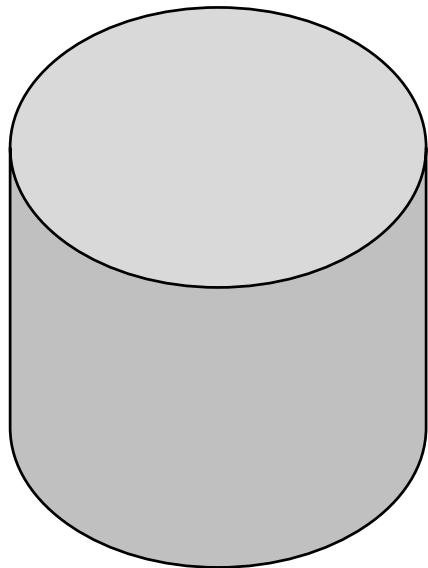


Texture Overview

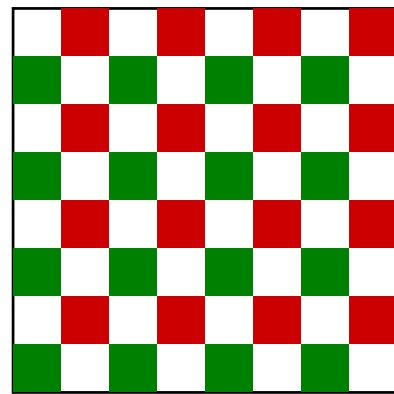
- Texture mapping stages
 - Parameterization
 - Mapping
 - Filtering
- Texture mapping applications
 - Modulation textures
 - Illumination mapping
 - Bump mapping
 - Environment mapping
 - Image-based rendering
 - Non-photorealistic rendering



Texture Parameterization



+



=



geometry

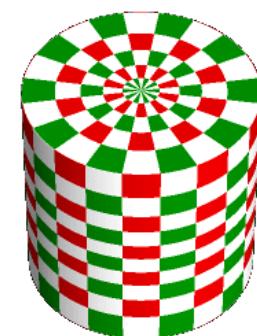
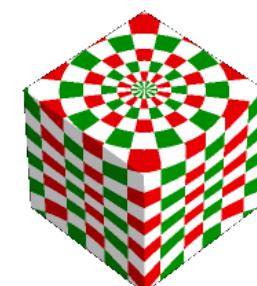
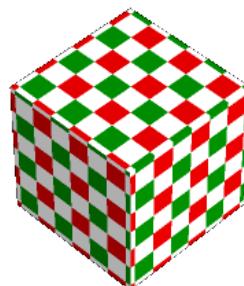
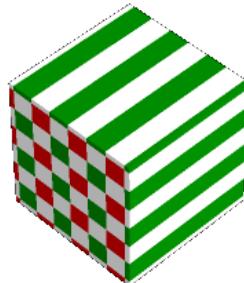
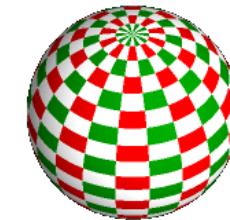
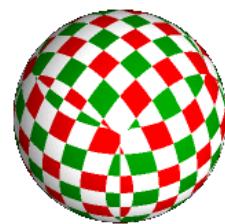
image

texture map

- Q: How do we decide *where* on the geometry each color from the image should go?



Texture Parameterization

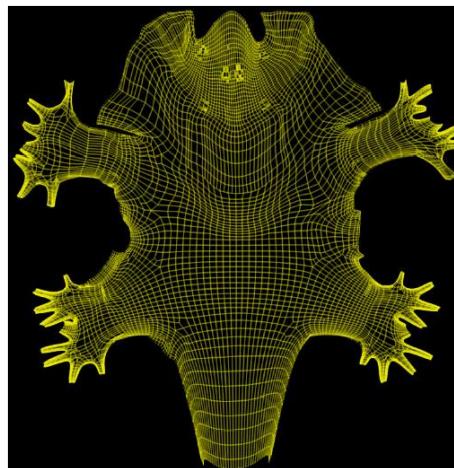
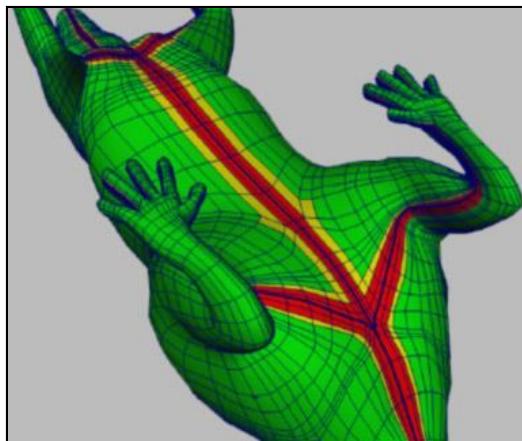


[Paul Bourke]

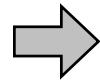
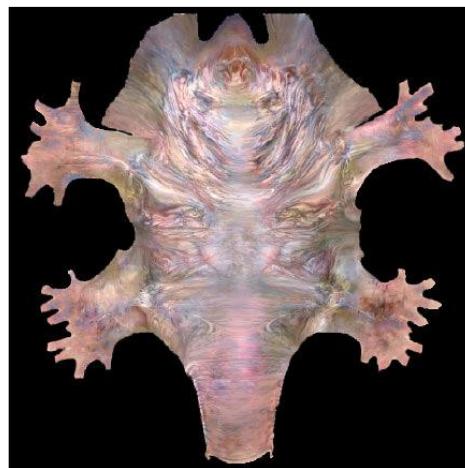


Texture Parameterization

Option1: unfold the surface



[Piponi2000]



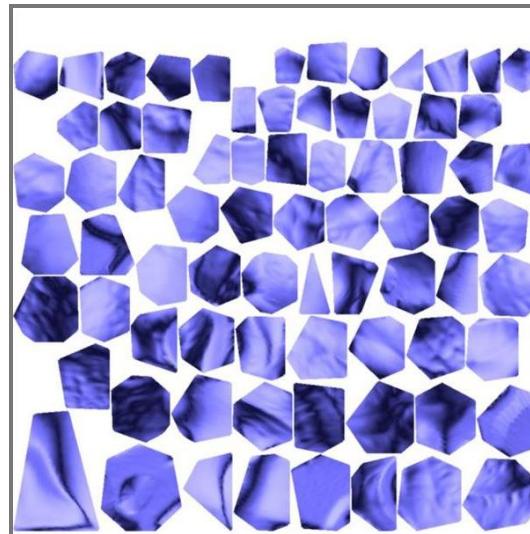


Texture Parameterization

Option2: make an atlas



charts



atlas



surface

[Sander2001]



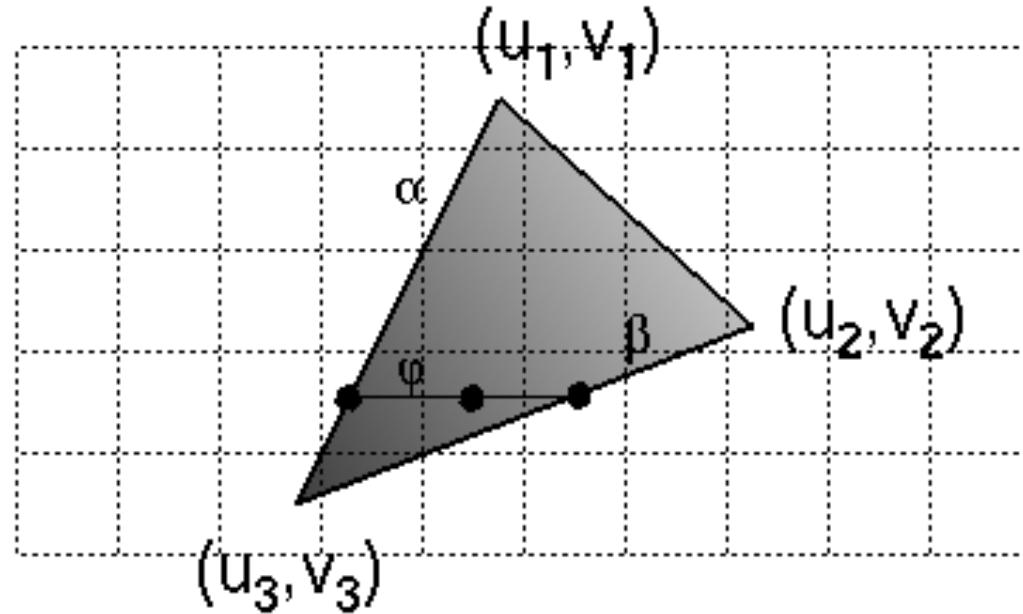
Texture Overview

- Texture mapping stages
 - Parameterization
 - Mapping
 - Filtering
- Texture mapping applications
 - Modulation textures
 - Illumination mapping
 - Bump mapping
 - Environment mapping
 - Image-based rendering
 - Non-photorealistic rendering



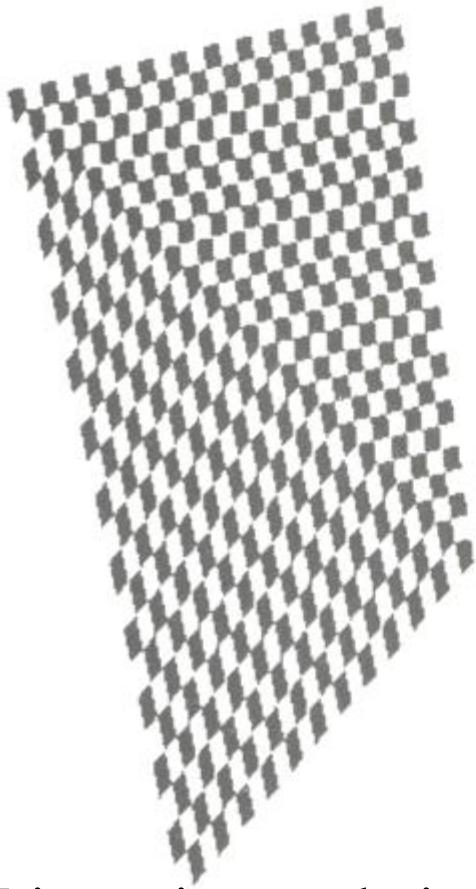
Texture Mapping

- Scan conversion
 - Interpolate texture coordinates down/across scan lines
 - Distortion due to bilinear interpolation approximation
 - » Cut polygons into smaller ones, or
 - » Perspective divide at each pixel

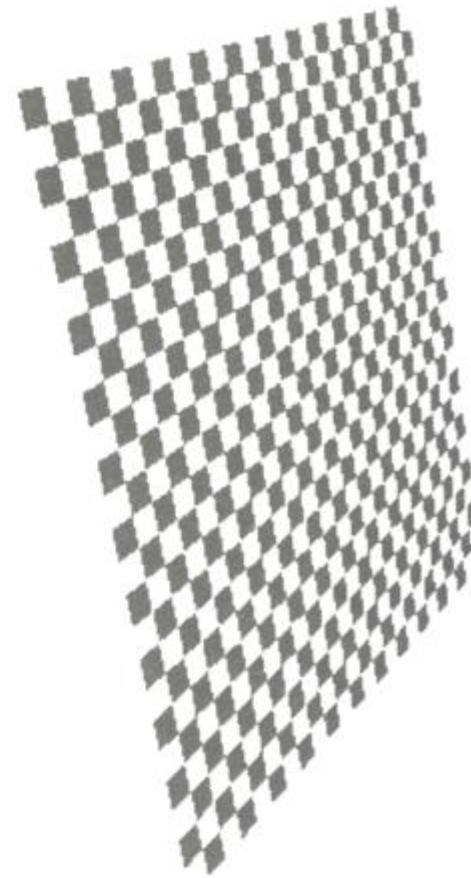




Texture Mapping



Linear interpolation
of texture coordinates



Correct interpolation
with perspective divide

Hill Figure 8.42



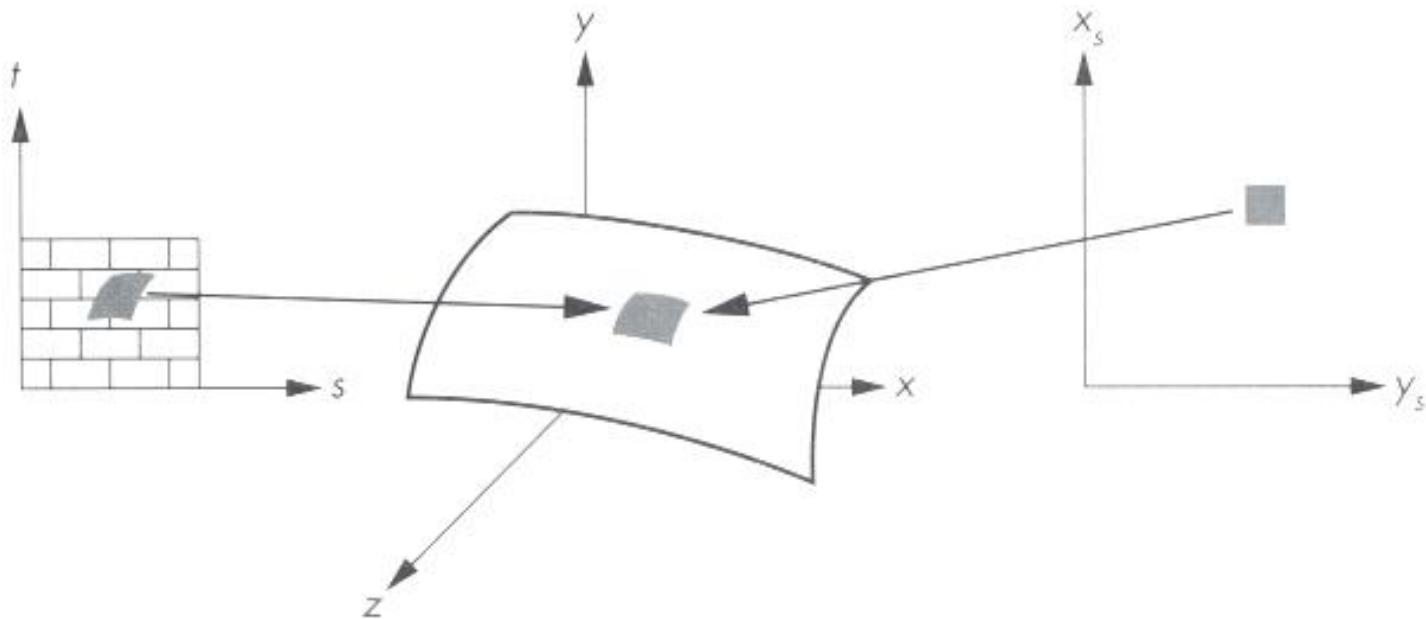
Texture Overview

- Texture mapping stages
 - Parameterization
 - Mapping
 - Filtering
- Texture mapping applications
 - Modulation textures
 - Illumination mapping
 - Bump mapping
 - Environment mapping
 - Image-based rendering
 - Non-photorealistic rendering



Texture Filtering

- Must **sample** texture to determine color at each pixel in image

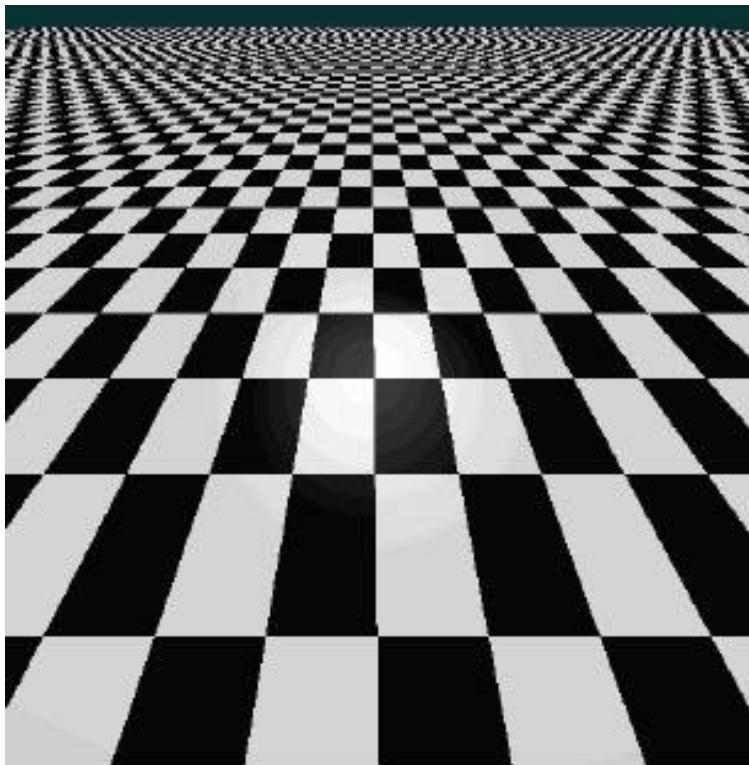


Angel Figure 9.4

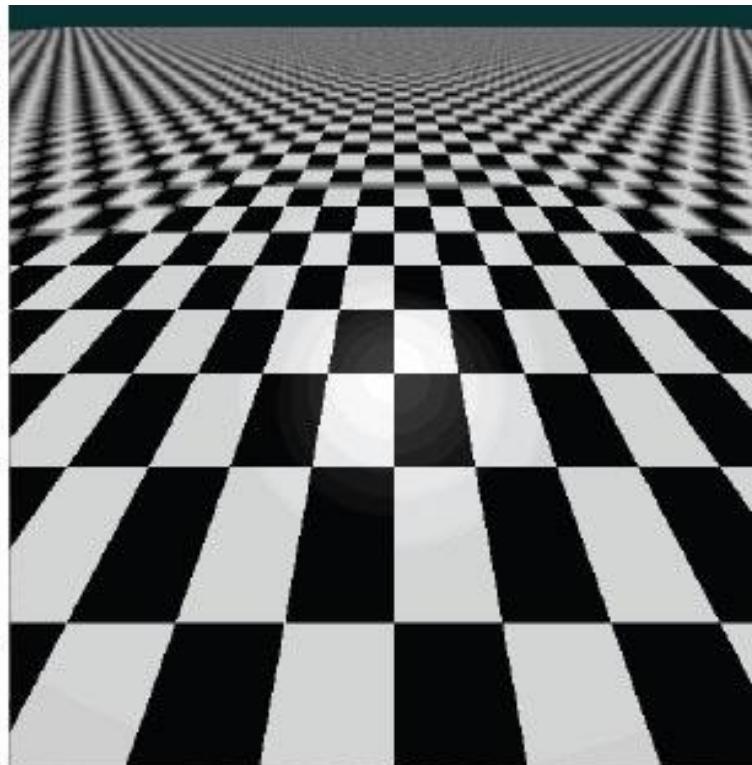


Texture Filtering

- Aliasing is a problem



Point sampling

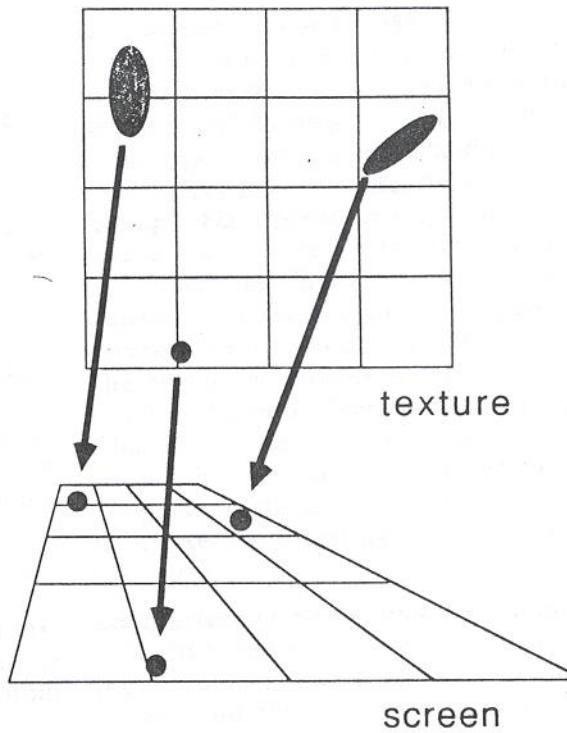


Area filtering



Texture Filtering

- Ideally, use elliptically shaped convolution filters

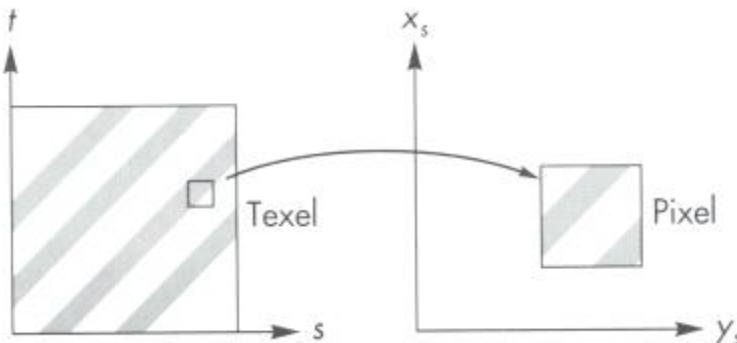


In practice, use rectangles or squares

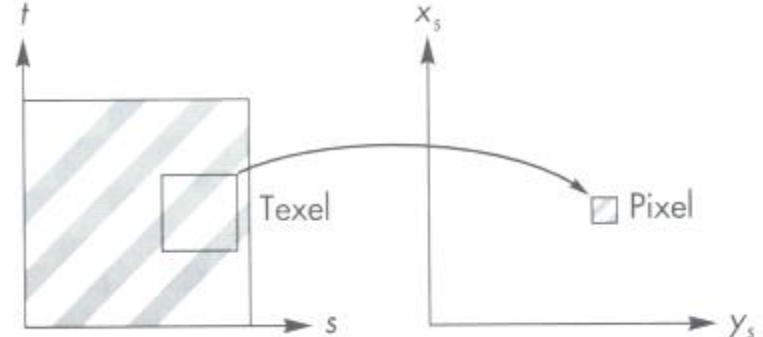


Texture Filtering

- Size of filter depends on projective warp
 - Compute prefiltered images to avoid run-time cost
 - » Mipmaps
 - » Summed area tables



Magnification

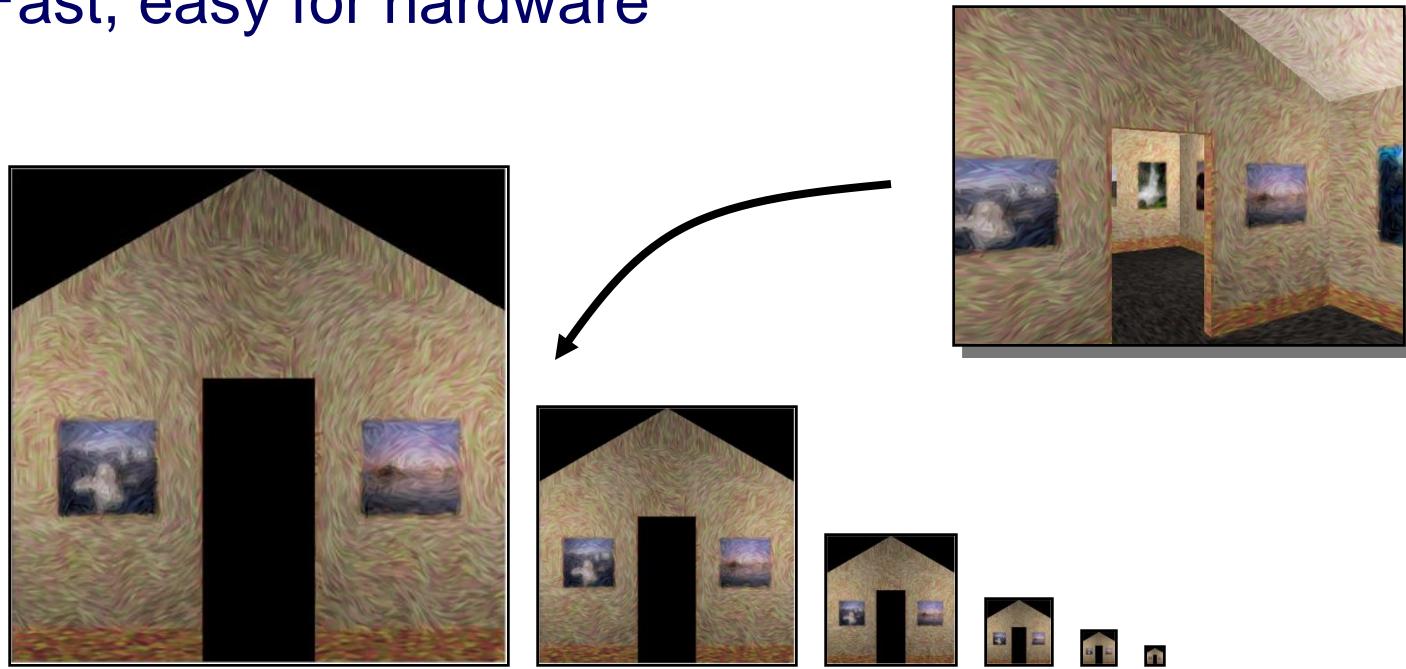


Minification



Mipmaps

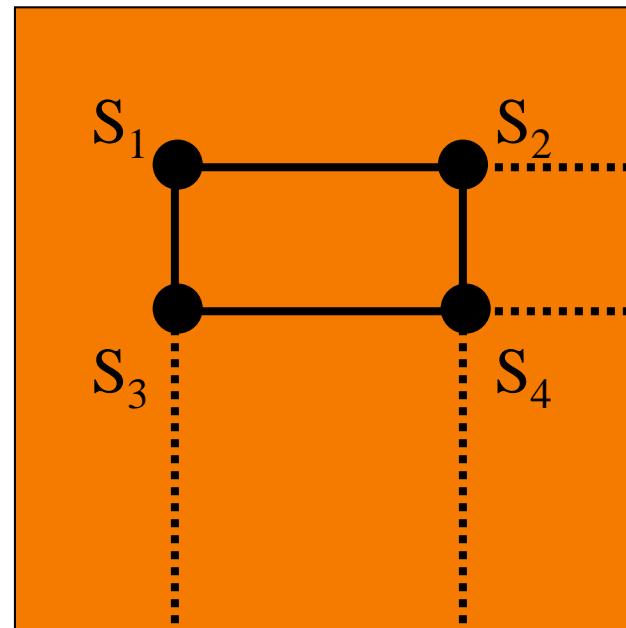
- Keep textures prefiltered at multiple resolutions
 - Usually powers of 2
 - For each pixel, linearly interpolate between two closest levels (i.e., trilinear filtering)
 - Fast, easy for hardware





Summed-area tables

- At each texel keep sum of all values down & right
 - To compute sum of all values within a rectangle, simply combine four entries: $S_1 - S_2 - S_3 + S_4$
 - Better ability to capture oblique projections, but still not perfect



- (Mipmaps are more common.)



Texture Overview

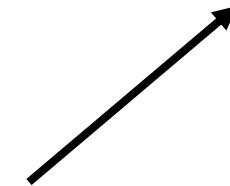
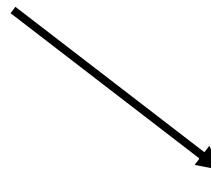
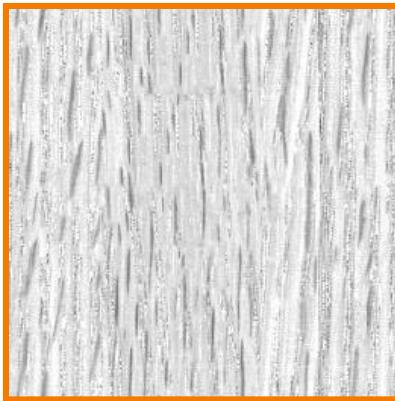
- Texture mapping stages
 - Parameterization
 - Mapping
 - Filtering
- Texture mapping applications
 - Modulation textures
 - Illumination mapping
 - Bump mapping
 - Environment mapping
 - Image-based rendering



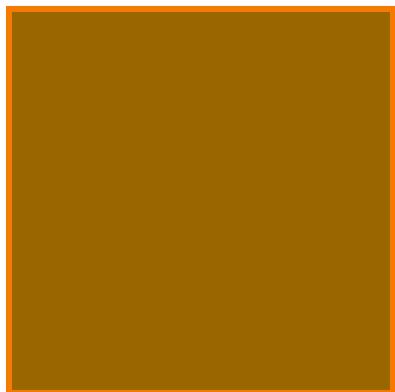
Modulation textures

Texture values scale result of lighting calculation

Wood texture



Texture
value



$$I = T(s, t) \left(I_E + K_A I_A + \sum_L \left(K_D (N \cdot L) + K_S (V \cdot R)^n \right) S_L I_L + K_T I_T + K_S I_S \right)$$



Illumination Mapping

Map texture values to surface material parameter

- K_A
- K_D
- K_S
- K_T
- n



Texture
value



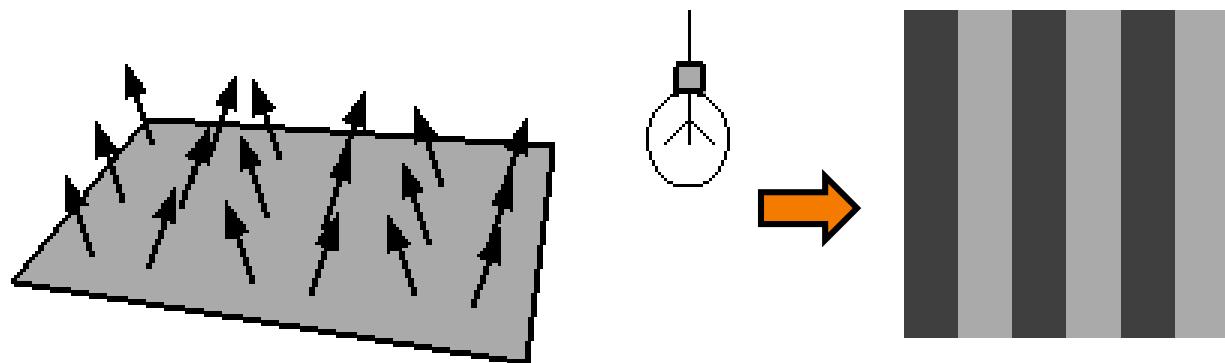
$$I = I_E + K_A I_A + \sum_L \left(K_D(s, t)(N \cdot L) + K_S(V \cdot R)^n \right) S_L I_L + K_T I_T + K_S I_S$$



Bump/Normal Mapping

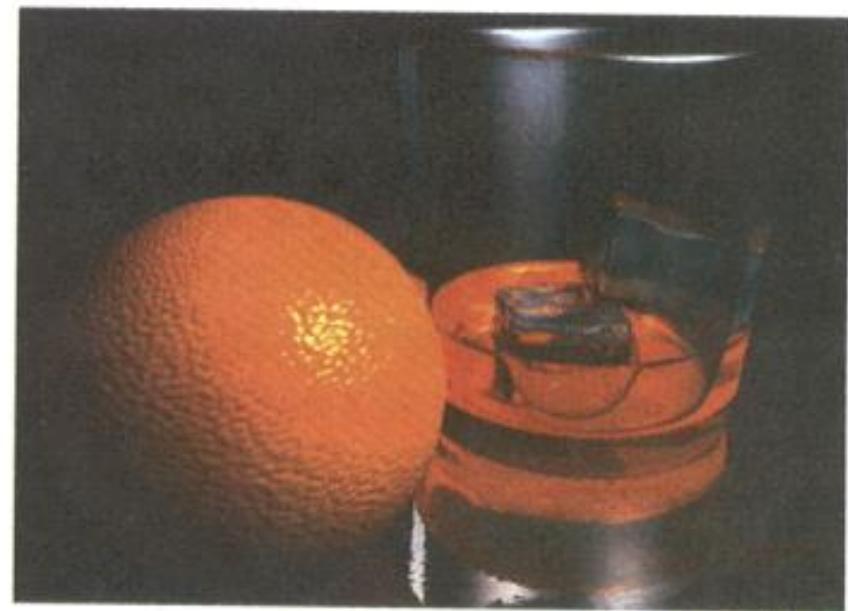
Texture values perturb surface normals:

- Use gradient of grayscale image (“bump”)
- Encode normals (or offsets) in RGB
- Encode normal offsets in tangent space





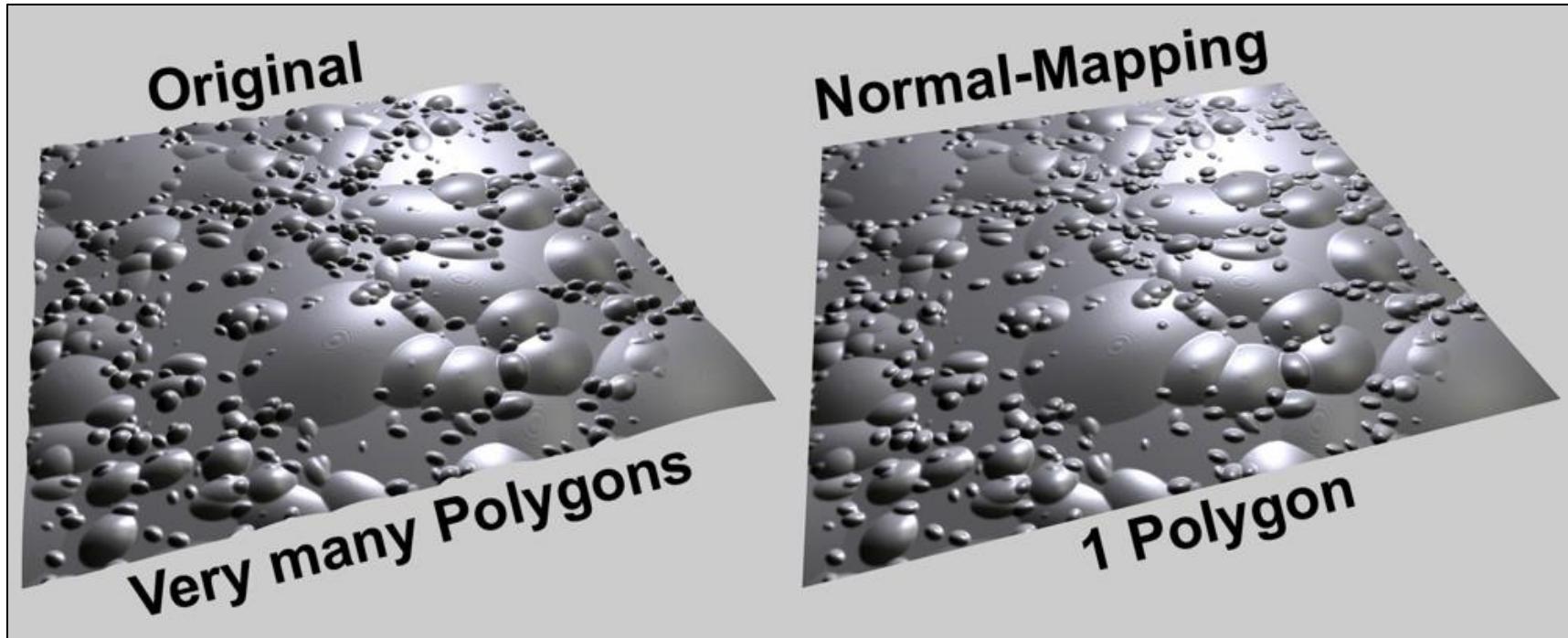
Bump Mapping



H&B Figure 14.100



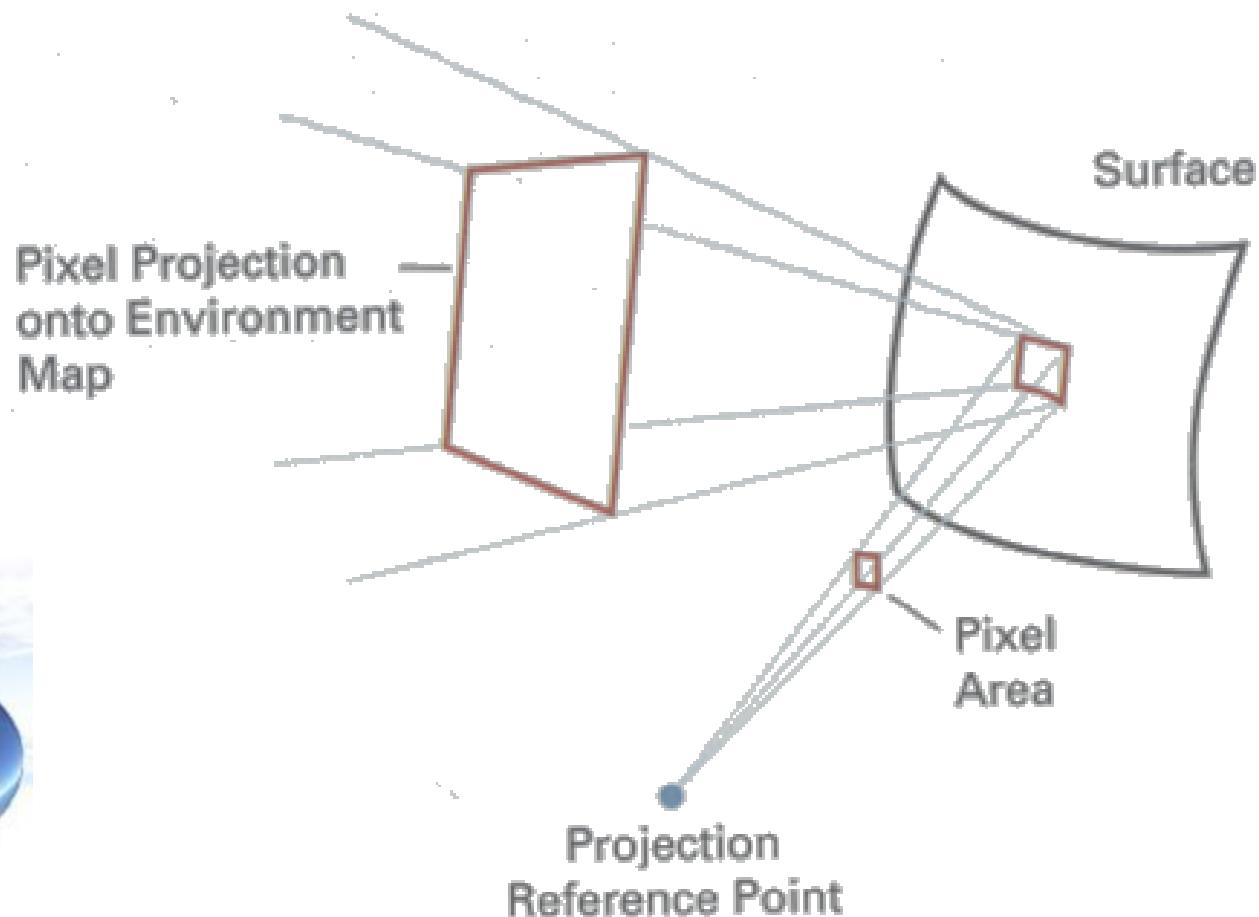
Normal Mapping



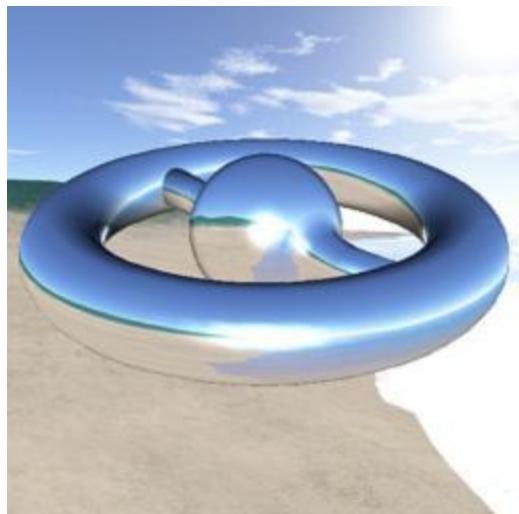


Environment Mapping

Texture values are reflected off surface patch



Gamer3D/Wikipedia

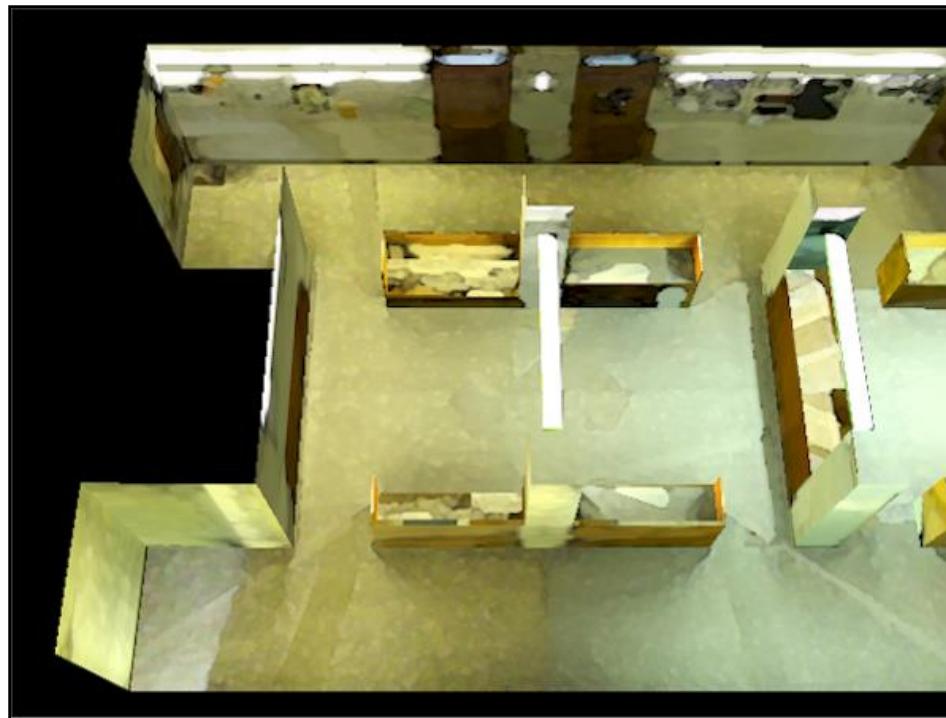


H&B Figure 14.93



Image-Based Rendering

Map photographic textures to provide details for coarsely detailed polygonal model





Solid textures

Texture values indexed
by 3D location (x,y,z)

- Expensive storage, or
- Compute on the fly,
e.g. Perlin noise →





Texture Summary

- Texture mapping stages
 - Parameterization
 - Mapping
 - Filtering
- Texture mapping applications
 - Modulation textures
 - Illumination mapping
 - Bump mapping
 - Environment mapping
 - Image-based rendering
 - Volume textures



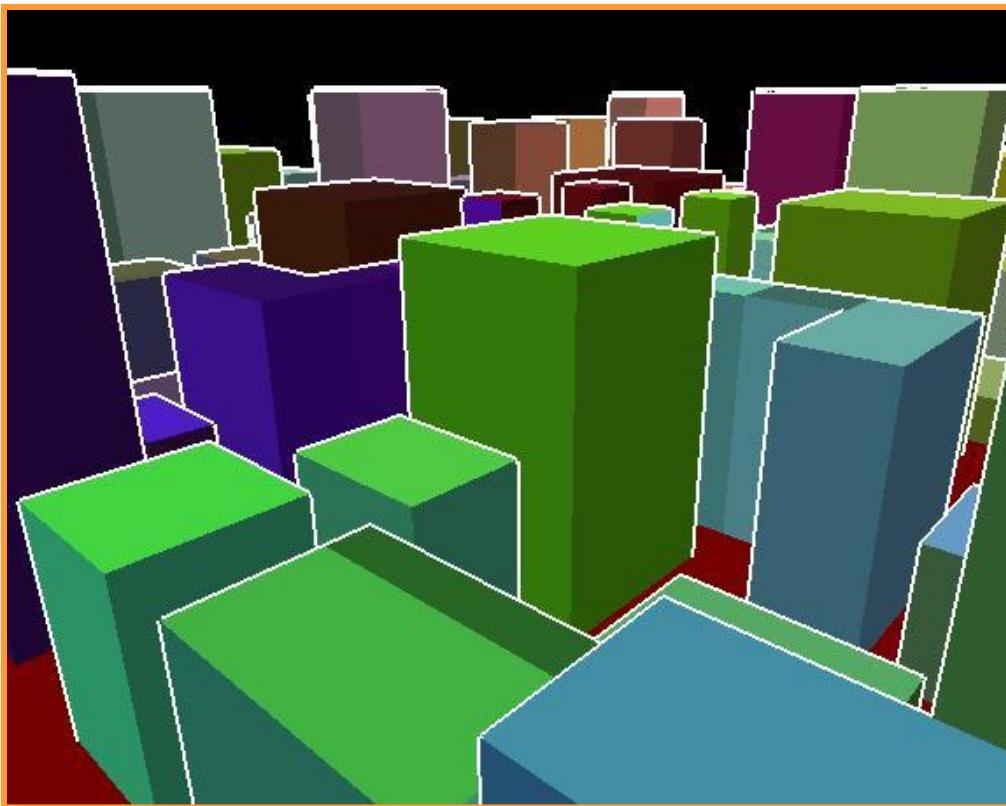
Rasterization

- Scan conversion
 - Determine which pixels to fill
 - Shading
 - Determine a color for each filled pixel
 - Texture mapping
 - Describe shading variation within polygon interiors
- **Visible surface determination**
- Figure out which surface is front-most at every pixel



Visible Surface Determination

Make sure only front-most surface contributes to color at every pixel

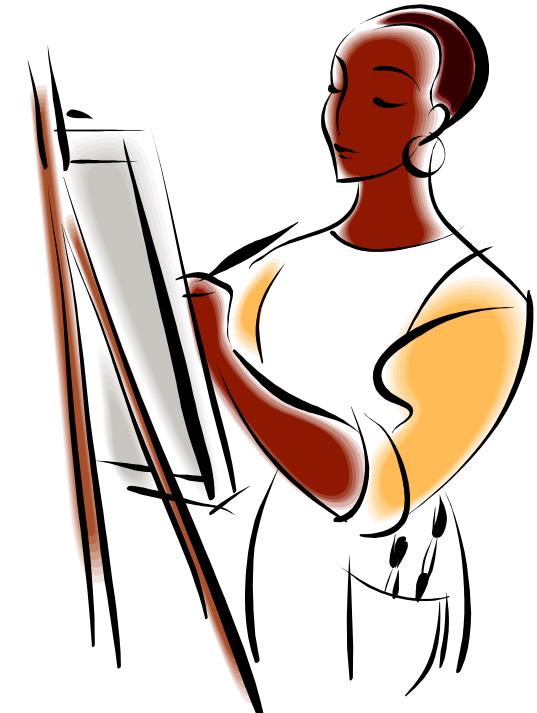
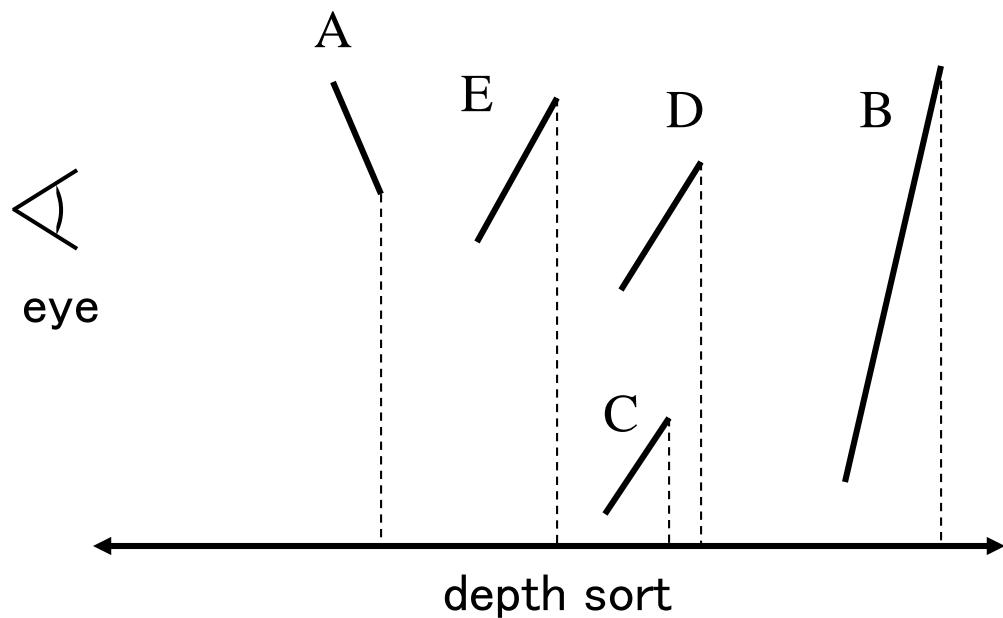




Depth sort

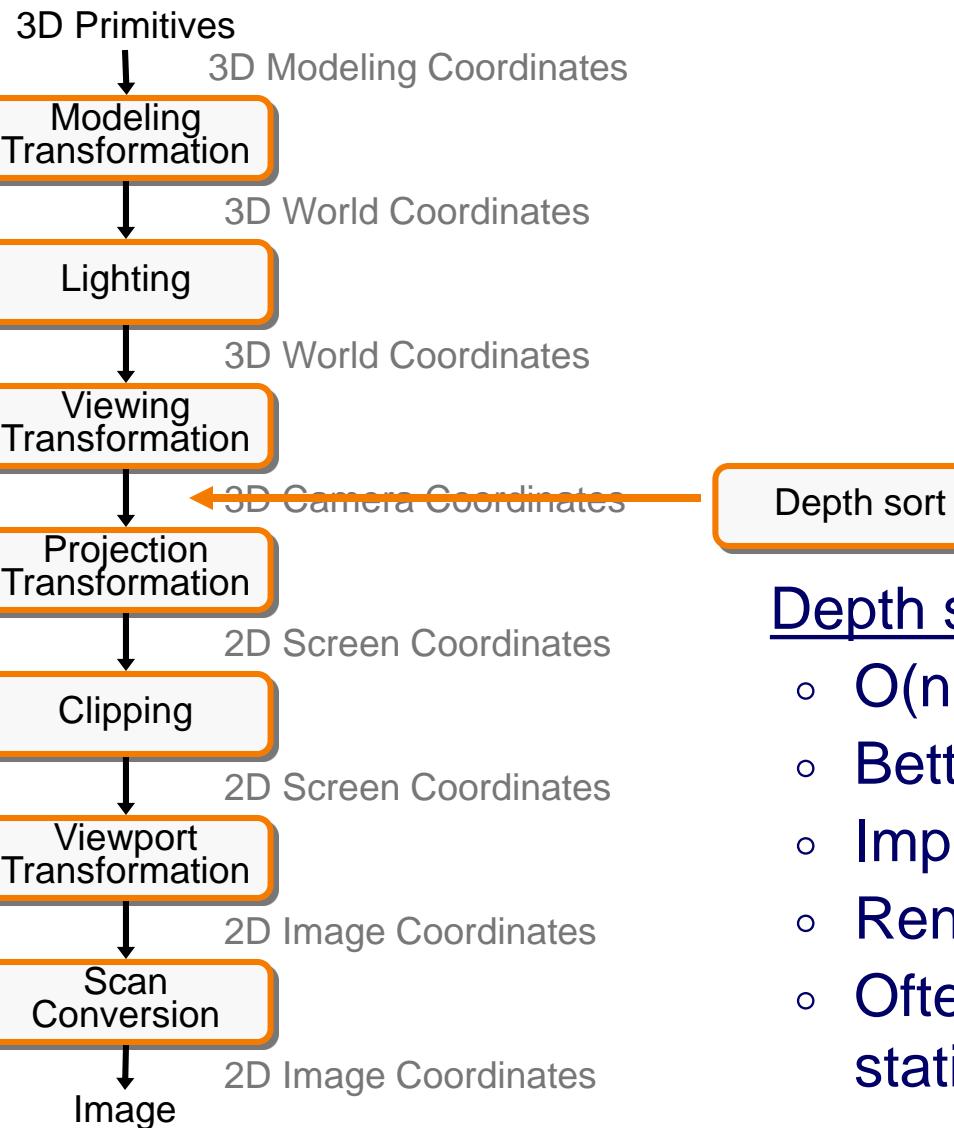
“Painter’s algorithm”

- Sort surfaces in order of decreasing maximum depth
- Scan convert surfaces in **back-to-front** order,
overwriting pixels





3D Rendering Pipeline



Depth sort comments

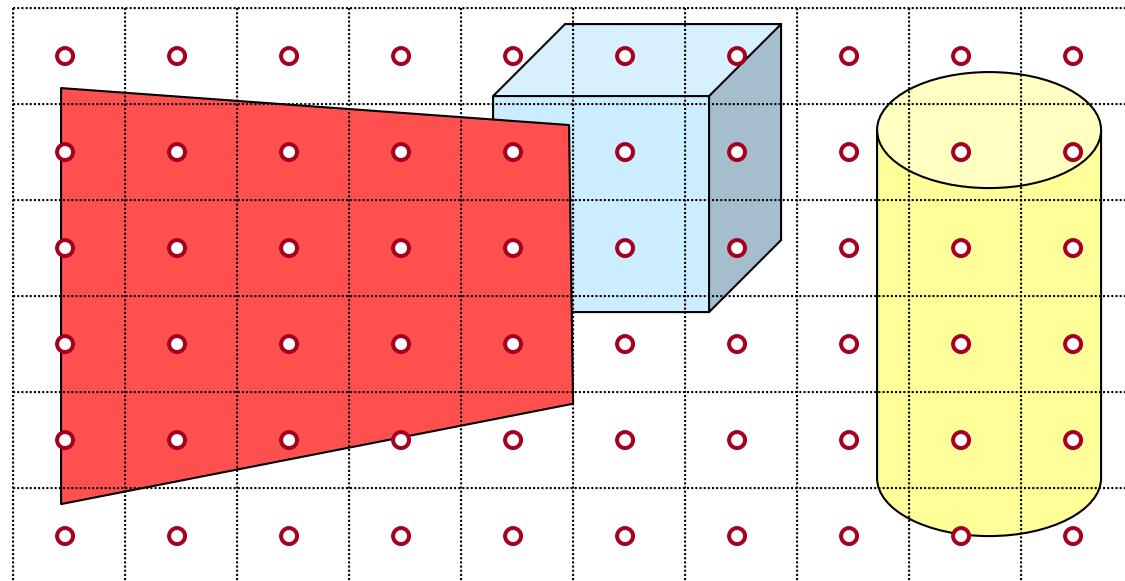
- $O(n \log n)$
- Better with frame coherence?
- Implemented in software
- Render every polygon
- Often use BSP-tree or static list ordering



Z-Buffer

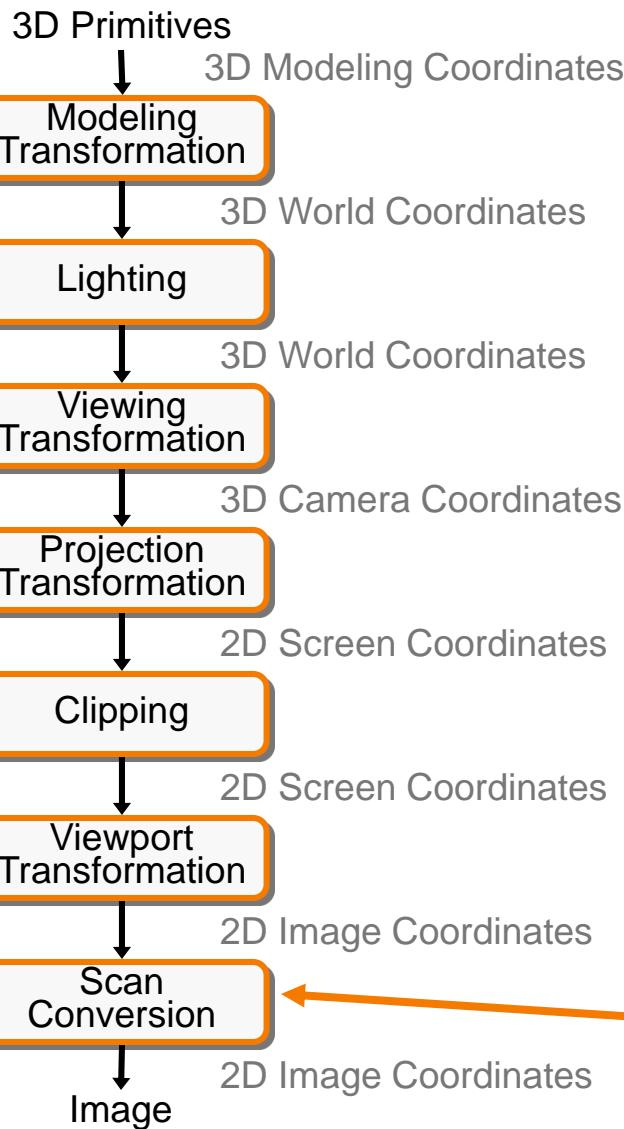
Maintain color & depth of closest object per pixel

- Framebuffer now RGBAz – initialize z to far plane
- Update only pixels with depth closer than in z-buffer
- Depths are interpolated from vertices, just like colors



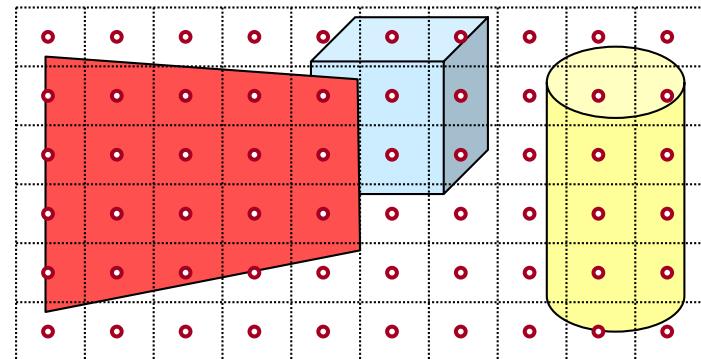


Z-Buffer



Z-buffer comments

- + Polygons rasterized in any order
- + Process one polygon at a time
- + Suitable for hardware pipeline
- Requires extra memory for z-buffer
- Subject to aliasing (A-buffer)
- o Commonly in hardware



Z-Buffer



Hidden Surface Removal Algorithms

36 • I. E. Sutherland, R. F. Sproull, and R. A. Schumacker

A Characterization of Ten Hidden-Surface Algorithms

• 37

Only z-buffer and ray tracing

OPAQUE-OBJECT ALGORITHMS									
COMPARISON ALGORITHMS									
OBJECT SPACE					(partly each)				
edges edges					LIST PRIORITY ALGORITHMS				
edges volumes					a priori priority				
IMAGESPACE					dynamically computed priority				
DEPTH PRIORITY ALGORITHMS					area sampling				
point sampling									
APPEL 1967	GALIMBERTI, et al. 1969	LOUTREL 1967	ROBERTS 1963	SCHUMACKER, et al. 1969	NEWELL, et al. 1972	WARNOCK 1968	HATEINGO 1970	RONNEY, et al. 1967	BOURNIGHT 1969
RESTRICTIONS	TP, NP	TP, NP	TP, NP	TP, CC, CF, NP	CF, NP, LS (TP)	(TR) None	None	TR, CF, NP	
COHERENCE	Promote visibility of a vertex to all edges at vertex	Promote visibility of a vertex to all edges at vertex	Promote visibility of a vertex to all edges at vertex		Frame coherence in depth No X coherence used	Intra-Cluster Priority	Sort faces by depth Scan-line coherence	Scan-line coherence	Scan-line coherence
SORTING	Back Edge Cull 1) Edges separating back-facing planes 2) Dot product with normals & topology 3) Cull 4) List of edges, E_s 5) $E_s \cap E_t$	Back Edge Cull 1) Edges separating back-facing planes 2) Dot product with normals & topology 3) Cull 4) List of edges, E_s 5) $E_s \cap E_t$	Back Edge Cull 1) Edges separating back-facing planes 2) Dot product with normals & topology 3) Cull 4) List of edges, E_s 5) $E_s \cap E_t$	Back Edge Cull 1) Edges separating back-facing planes 2) Dot product with normals & topology 3) Cull 4) List of edges, E_s 5) $E_s \cap E_t$	Intra-Cluster Priority 1) Faces, max Z 2) Dot product of max points 3) Log m 4) Order table 5) $E_s \cap E_t$	Sort faces by depth Scan-line coherence	Sort faces by depth Scan-line X coherence	Sort faces by depth Scan-line coherence	Sort faces by depth Scan-line coherence
(1) what prop- erty									
(2) Method									
(3) Type	Contour Edge Cull 1) Edges separating front & back faces	Contour Edge Cull 1) Edges separating front & back faces	Contour Edge Cull 1) Edges separating front & back faces	Contour Edge Cull 1) Edges separating front & back faces	Contour Edge Cull 1) Edges separating front & back faces	Newell Special 1) Clusters 2) Dot product with separating planes 3) Prefix scan binary tree 4) Radix 4 subdivision with overlap	X Merge 1) Faces, X value 2) Dot product of X and Y, sum of angles 3) Merge (ordered) 4) 2-way linked list 5) E_s, S_k	X Sort 1) Segments, X value 2) Comparison 3) Merge (ordered) 4) Table of lists 5) E_s, S_k	X Sort 1) Edges, X value 2) Comparison 3) Merge (ordered) 4) Table of lists 5) E_s, S_k
(4) Result									
structure									
(5) Number per frame, number of objects									
(merge)									
Number of new entries per frame, length of list									
Number of cache hits per frame, length of list									
Initial Visibility	Initial Visibility 1) Ray to vertex against all faces	Visibility 1) Ray to vertex against all faces	Edge/Volume Test 1) Edges, visibility 2) Line segments, intersection 3) Cull (unsorted)	Edge/Volume Test 1) Edges, visibility 2) Line segments, intersection 3) Cull (unsorted)	Y Sort 1) Face segment by Y range 2) Y intercept 3) Bucket 4) None 5) E_s, F_t	Depth Search 1) Face segment by Y range 2) 4-corner compare 3) Exhaustive search 4) Answer/failure 5) E_s, F_t /factor 1	X Sort 1) Segments, X value 2) Comparison 3) Bubble 4) Active segment 5) n, S_k	X Sort 1) Segments, X value 2) Comparison 3) Bubble 4) 1-way linked list 5) n, S_k	X Sort 1) Edges, X value 2) Comparison 3) Merge (ordered) 4) Table of lists 5) E_s, S_k
Initial Edge Intersections	Initial Edge Intersections 1) Intersections on edge, ordering	Initial Edge Intersections 1) Intersections on edge, ordering	Edge Intersection 1) Intersections on edge, ordering	Edge Intersection 1) Intersections on edge, ordering	Y Merge 1) Faces by Y extent 2) Min-max on X intercept 3) Cull (unsorted)	IV Sort (Opt) Sort windows into scan-line order if needed	Span Cull 1) Segments, overlap with sample span 2) Linear equations and comparison	Search 1) Segments, depth 2) Linear equations and comparison	Search 1) Edges, depth 2) Linear equations and comparison
Intersections with sweep triangle									
Cull (unsorted)									
Intersection list									
E_s, E_c									
Sort Along Edge	Sort Along Edge 1) Intersections on edge, ordering	X Sort 1) Segments 2) Counters 3) Hardware 4) Segments at threshold 5) n, S_k	Y Sort 1) Segments, priority 2) Logic network 3) Logic network 4) Visible segment 5) n, S_k	Priority Search 1) Segments, priority 2) Logic network 3) Logic network 4) Visible segment 5) n, S_k	Search 1) Segments, depth 2) Linear equations and comparison	Search 1) Edges, depth 2) Linear equations and comparison			
$E_s, X_s/E_s$									
(must be done)									

Figure 29. Characterization of ten opaque-object algorithms. b. Comparison of the algorithms.

[Sutherland '74]



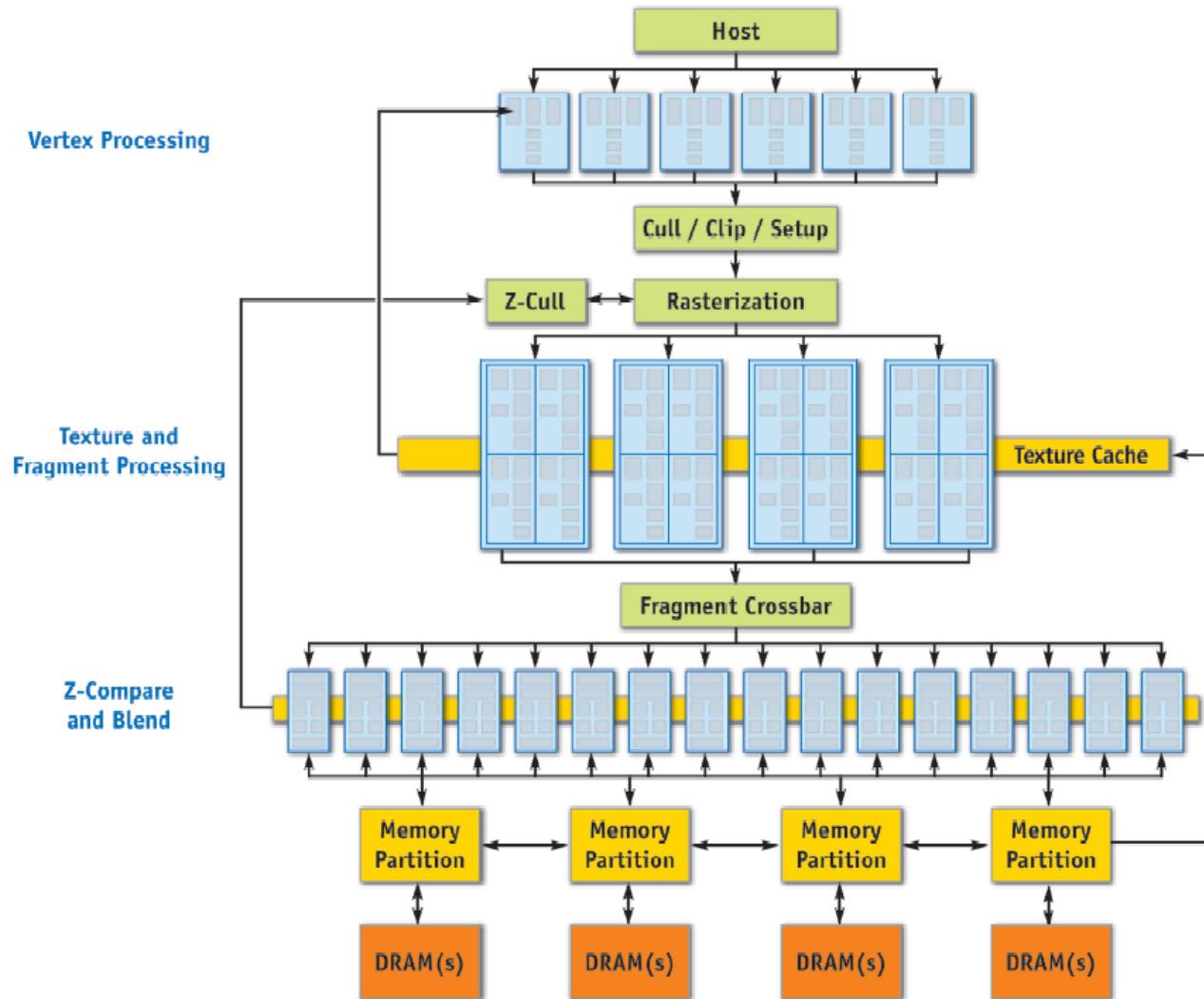
Rasterization Summary

- Scan conversion
 - Sweep-line algorithm
- Shading algorithms
 - Flat, Gouraud
- Texture mapping
 - Mipmaps
- Visibility determination
 - Z-buffer

This is all in hardware



GPU Architecture



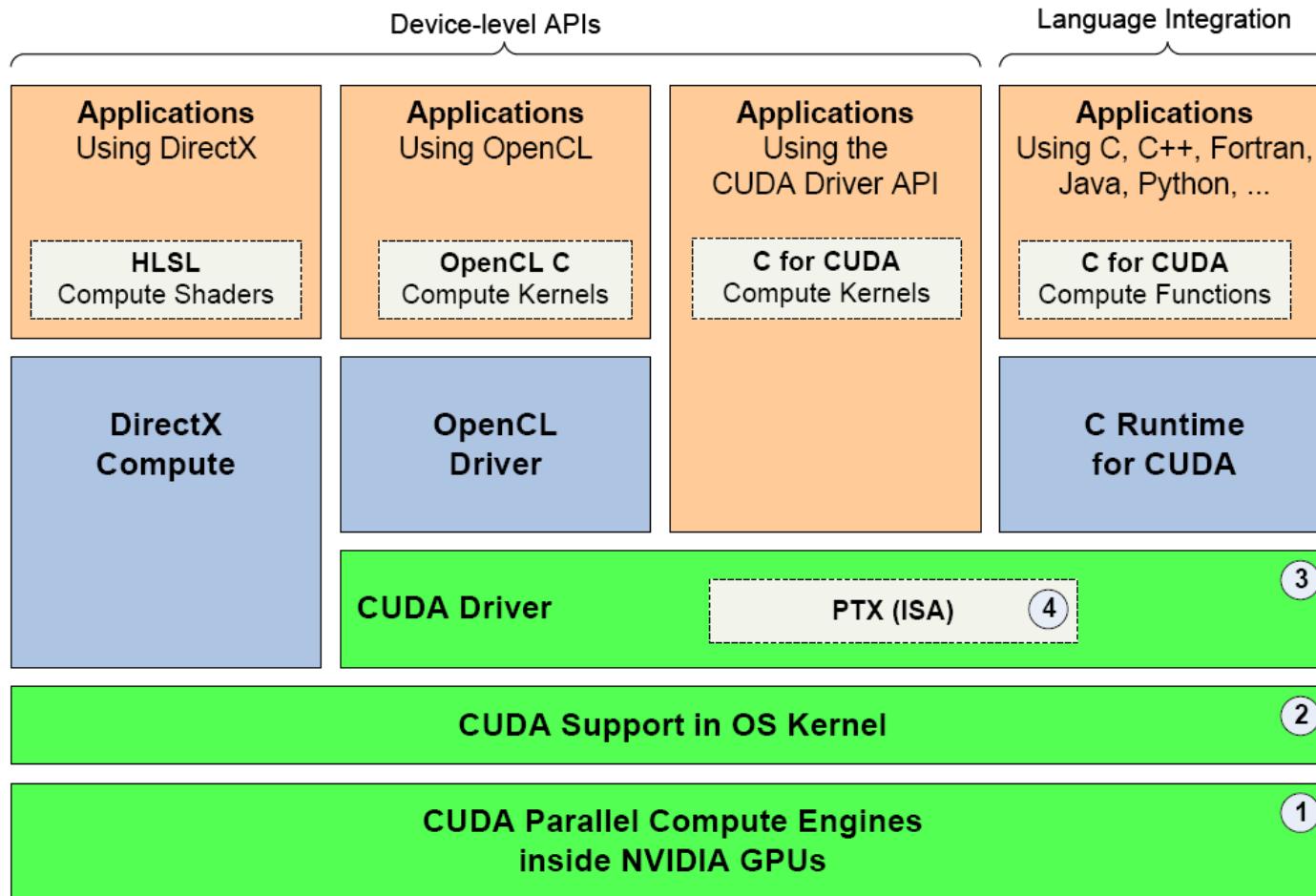
GeForce 6 Series Architecture

GPU Gems 2, NVIDIA



Actually ...

- Graphics hardware is programmable





Trend ...

- GPU is general-purpose parallel computer

