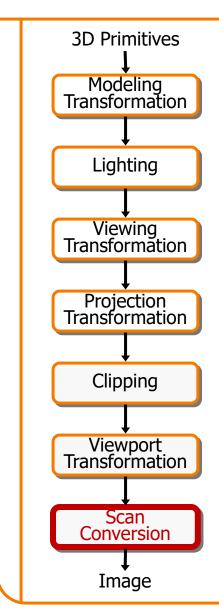


Scan Conversion

COS 426, Spring 2022 Felix Heide Princeton University

3D Rendering Pipeline (for direct illumination)





Rasterization

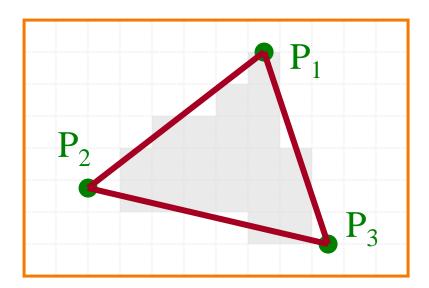


- Scan conversion (last time)
 - Determine which pixels to fill
- ➤ Shading
 - Determine a color for each filled pixel
- Texture mapping
 - Describe shading variation within polygon interiors
- Visible surface determination
 - Figure out which surface is front-most at every pixel

Shading



How do we choose a color for each filled pixel?

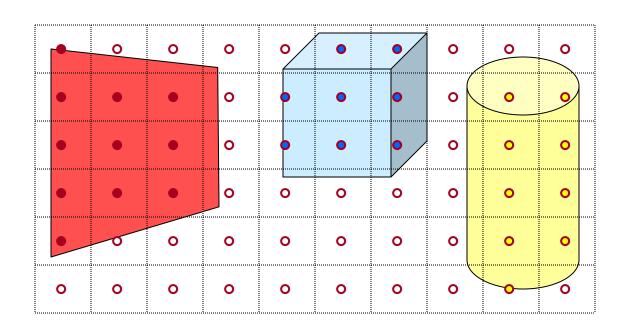


Emphasis on methods that can be implemented in hardware

Ray Casting



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

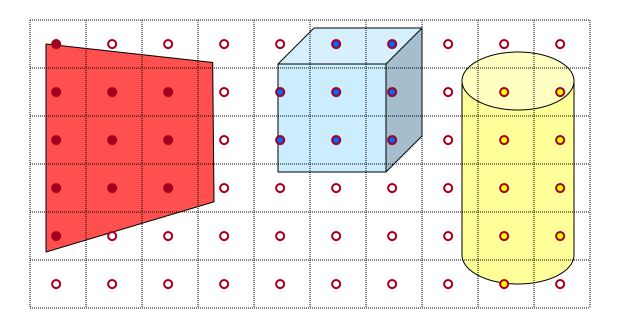


$$I = I_{E} + K_{A}I_{AL} + \sum_{i} \left(K_{D}(N \cdot L_{i})I_{i} + K_{S}(V \cdot R_{i})^{n} I_{i} \right)$$

Polygon Shading



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



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

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 viewed from infinitely far away

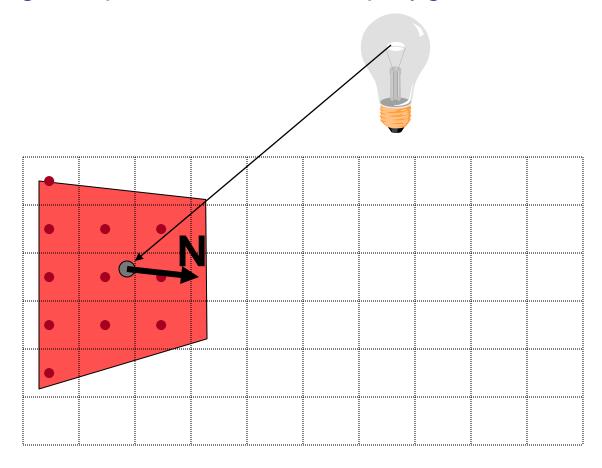


$$I = I_{E} + K_{A}I_{AL} + \sum_{i} \left(K_{D}(N \cdot L_{i})I_{i} + K_{S}(V \cdot R_{i})^{n} I_{i} \right)$$

Flat Shading



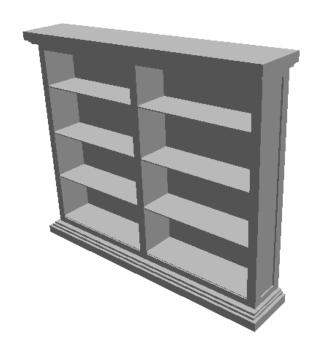
- One illumination calculation per polygon
 - Assign all pixels inside each polygon the same color

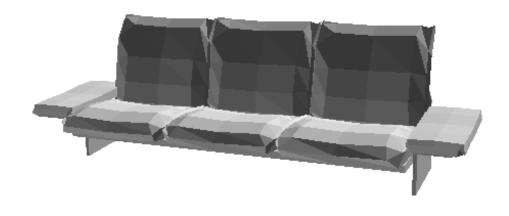


Flat Shading



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

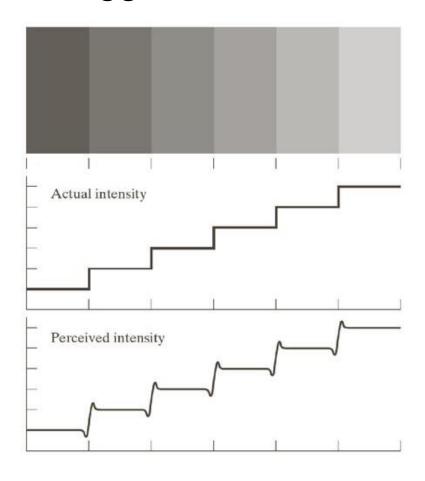




Mach Band Effect



• Edges between adjacent shades of gray are perceived as exaggerated.



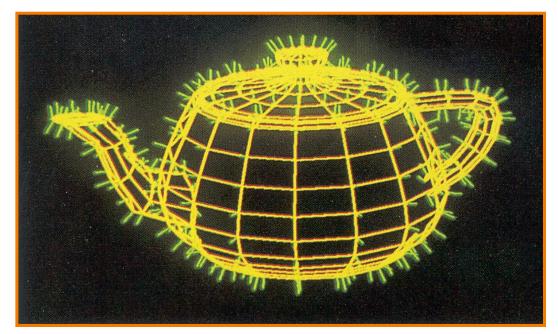
Polygon Shading Algorithms



- Flat Shading
- Gouraud Shading
- Phong Shading



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

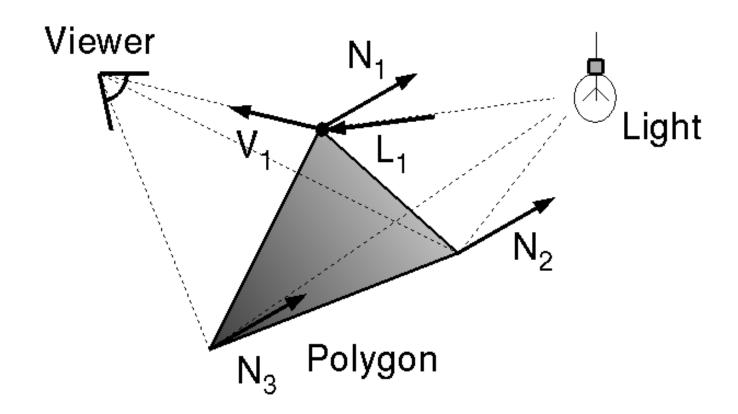


Watt Plate 7

$$I = I_{E} + K_{A}I_{AL} + \sum_{i} \left(K_{D}(N \cdot L_{i})I_{i} + K_{S}(V \cdot R_{i})^{n} I_{i} \right)$$



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

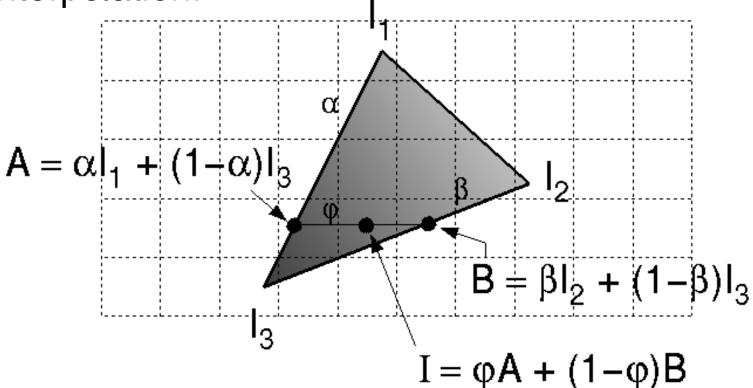




Bilinear interpolation of colors at vertices

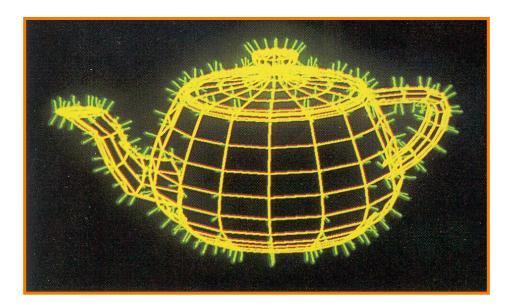
down and across scan lines = barycentric

interpolation!





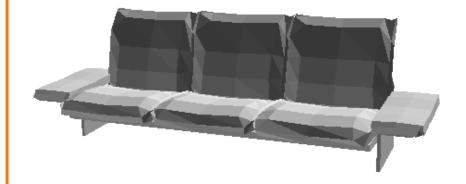
- Smooth shading over adjacent polygons
 - Curved surfaces
 - Illumination highlights
 - Soft shadows



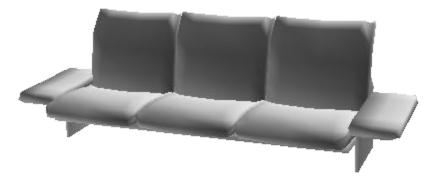
Mesh with shared normals at vertices



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





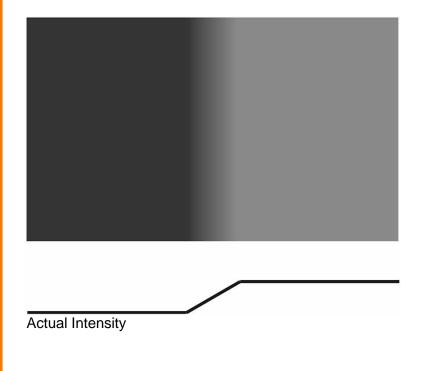


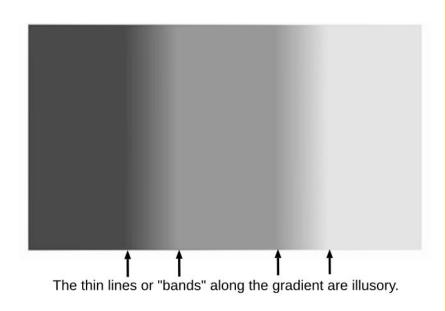
Gourand Shading

Mach Band Effect



 Mach Band Effect also affects Gouraud Shading for piecewise linear interpolation.





Polygon Shading Algorithms

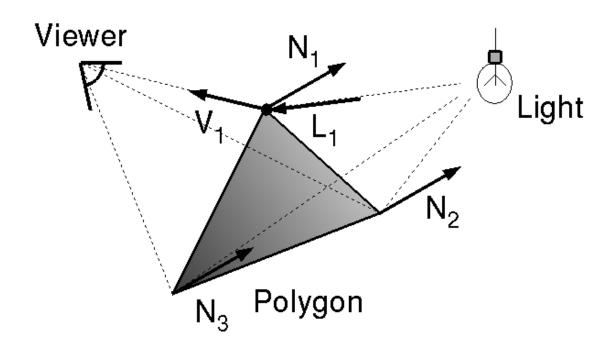


- Flat Shading
- Gouraud Shading
- Phong Shading (≠ Phong reflectance model)

Phong Shading



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

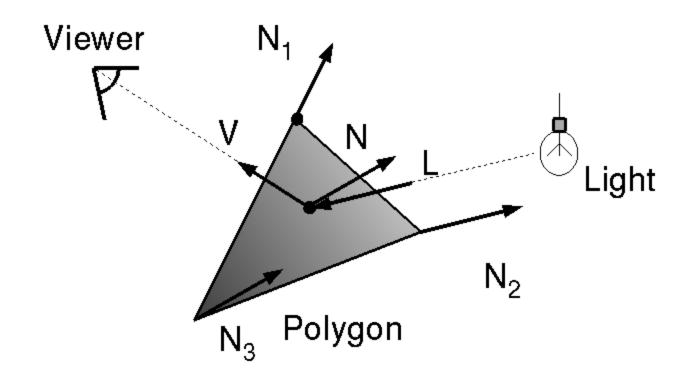


$$I = I_{E} + K_{A}I_{AL} + \sum_{i} \left(K_{D}(N \cdot L_{i})I_{i} + K_{S}(V \cdot R_{i})^{n} I_{i} \right)$$

Phong Shading



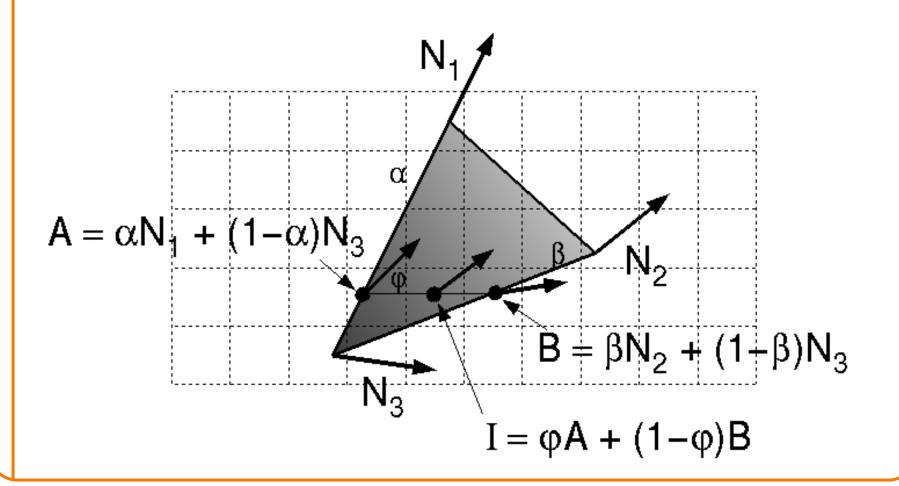
- One lighting calculation per pixel
 - Approximate surface normals for points inside polygons by bilinear interpolation of normals from vertices



Phong Shading



Bilinear interpolation of surface normals at vertices

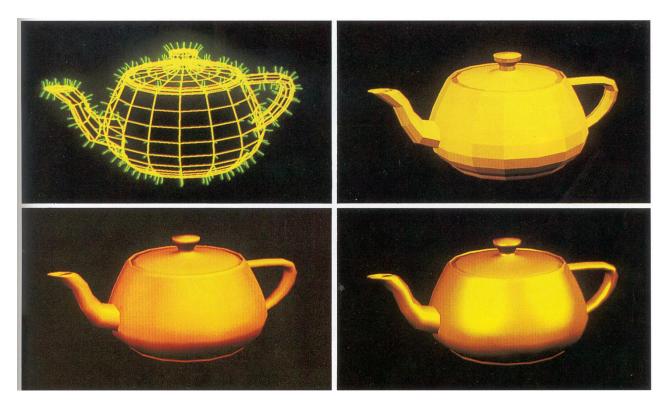


Polygon Shading Algorithms



Wireframe

Flat



Gouraud

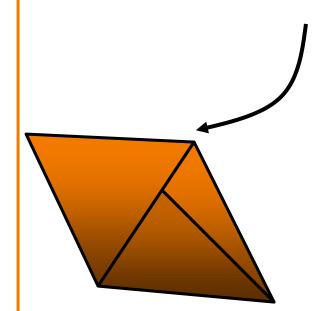
Phong

Demo: https://threejs.org/docs/scenes/material-browser.html#MeshPhongMaterial

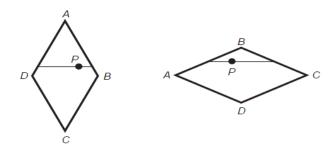
Shading Issues



- Problems with interpolated shading:
 - Polygonal <u>silhouettes</u> still obvious
 - Perspective distortion (due to <u>screen-space interpolation</u>)
 - Problems at T-junctions



The results of interpolated-shading is not independent of the projected polygons position (Foley Figure 14.22).



Rasterization

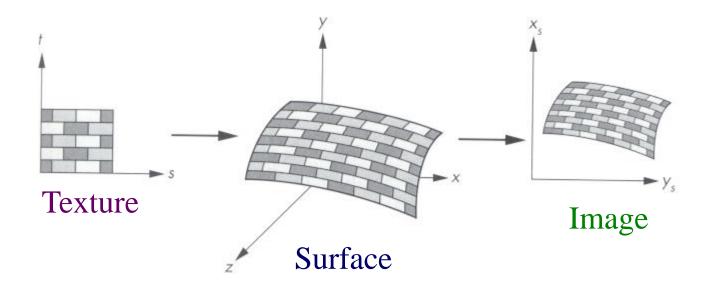


- Scan conversion
 - Determine which pixels to fill
- Shading
 - Determine a color for each filled pixel
- Texture mapping
 - Describe shading variation within polygon interiors
- Visible surface determination
 - Figure out which surface is front-most at every pixel

Textures



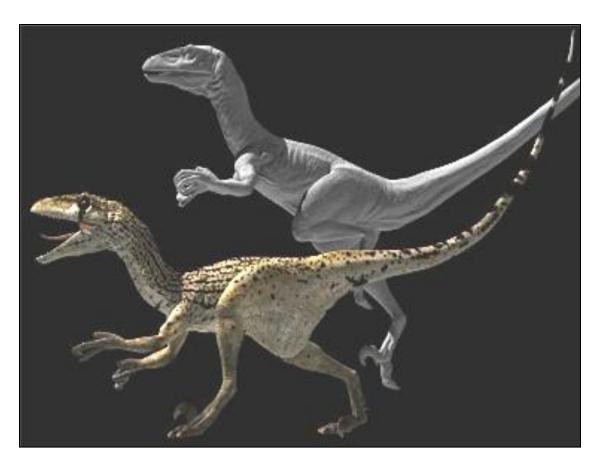
- Describe color variation in interior of 3D polygon
 - When scan converting a polygon, vary pixel colors according to values fetched from a texture image

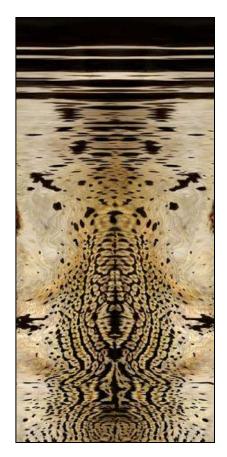


Textures



Add visual detail to surfaces of 3D objects





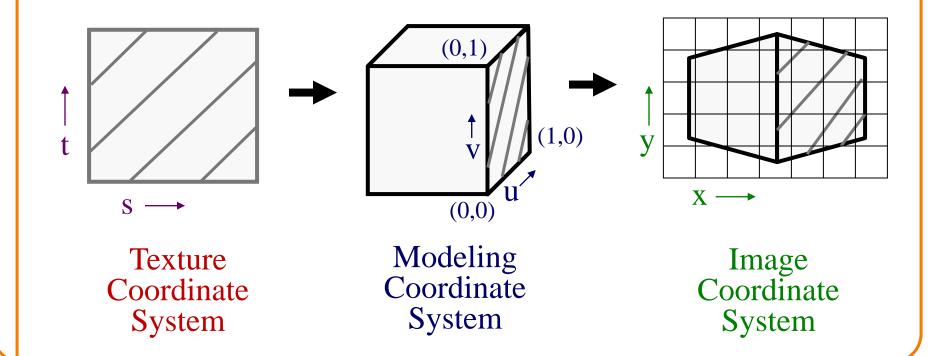
[Daren Horley]

Texture Mapping



Steps:

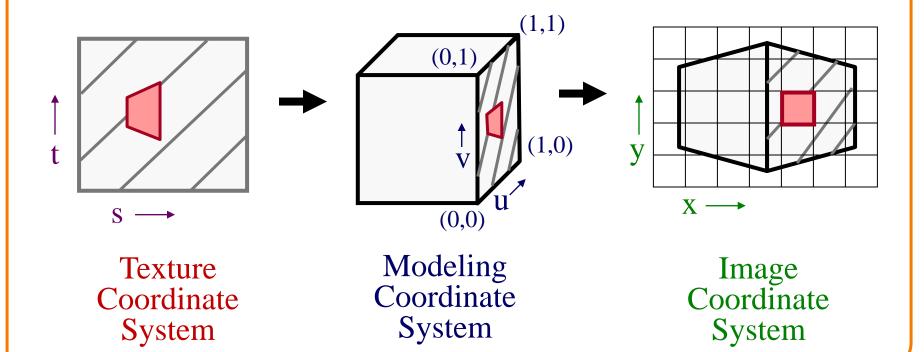
- 1. Define texture
- 2. Specify mapping from texture to surface
- 3. Look up texture values during scan conversion



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



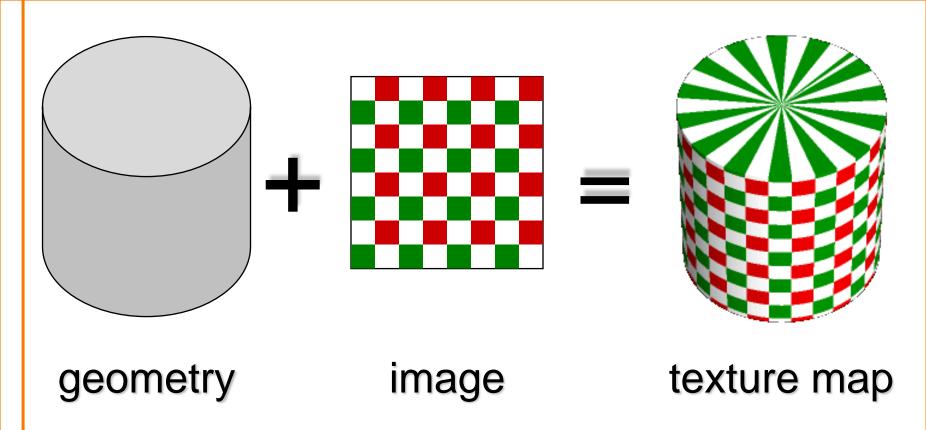
- Texture mapping stages
 - Parameterization
 - Mapping
 - Filtering
- Texture mapping applications
 - Modulation textures
 - Illumination mapping
 - Bump mapping
 - Environment mapping
 - Image-based rendering
 - Non-photorealistic rendering

Texture Overview



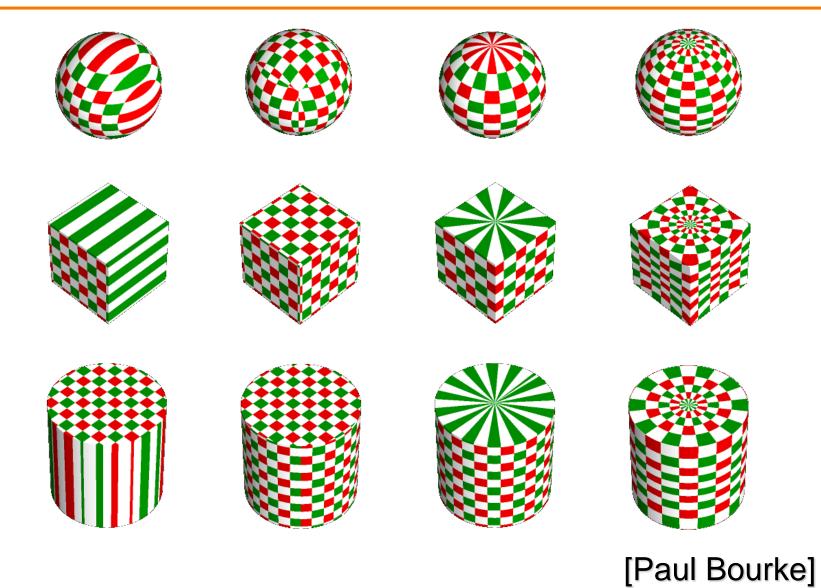
- Texture mapping stages
 - > Parameterization
 - Mapping
 - Filtering
- Texture mapping applications
 - Modulation textures
 - Illumination mapping
 - Bump mapping
 - Environment mapping
 - Image-based rendering
 - Non-photorealistic rendering





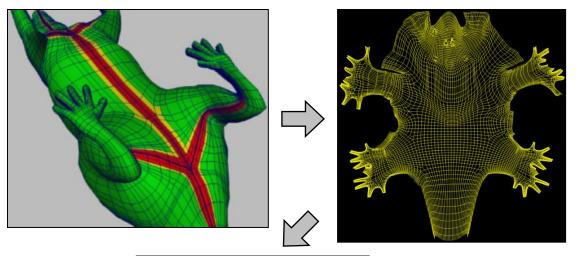
 Q: How do we decide where on the geometry each color from the image should go?



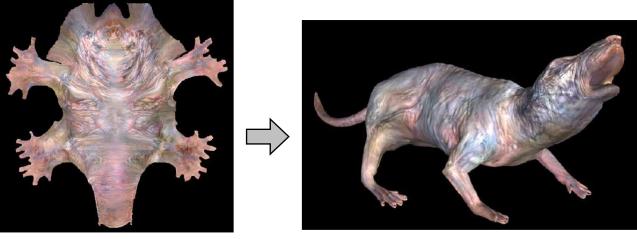




Option1: unfold the surface

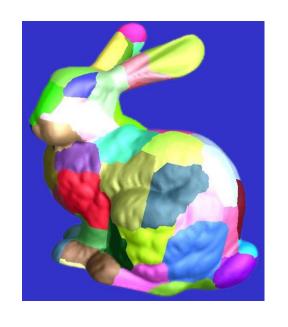


[Piponi2000]

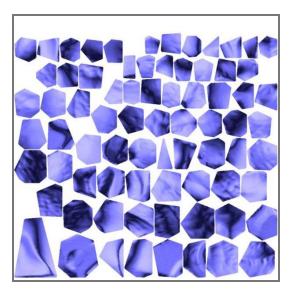




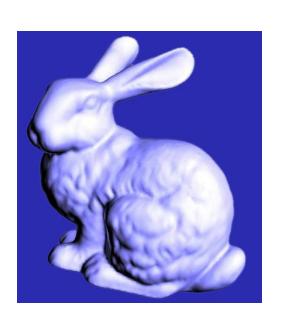
Option2: make an atlas



charts



atlas



surface

[Sander2001]

Texture Overview

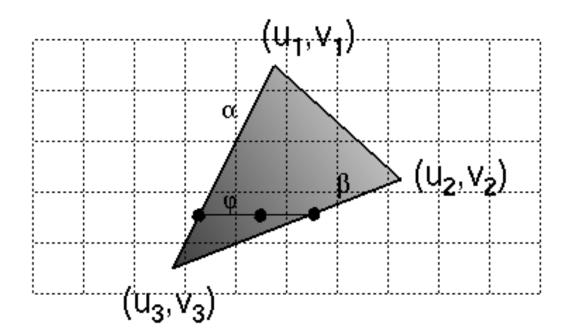


- Texture mapping stages
 - Parameterization
 - Mapping
 - Filtering
- Texture mapping applications
 - Modulation textures
 - Illumination mapping
 - Bump mapping
 - Environment mapping
 - Image-based rendering
 - Non-photorealistic rendering

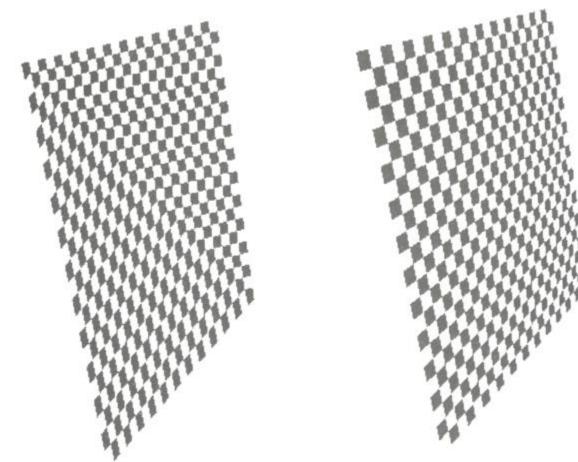
Texture Mapping



- Scan conversion
 - Interpolate texture coordinates down/across scan lines



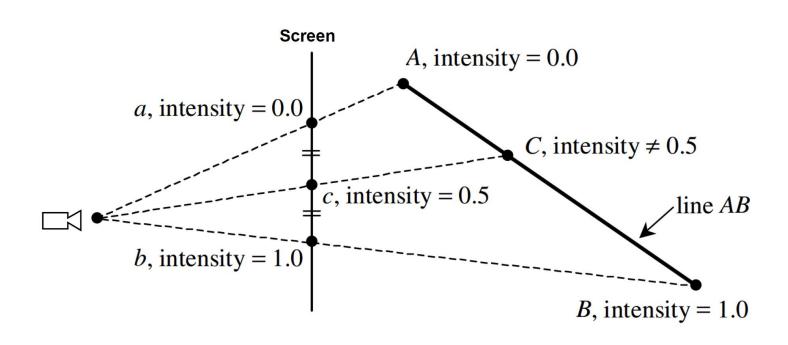




Linear interpolation of texture coordinates

Correct interpolation

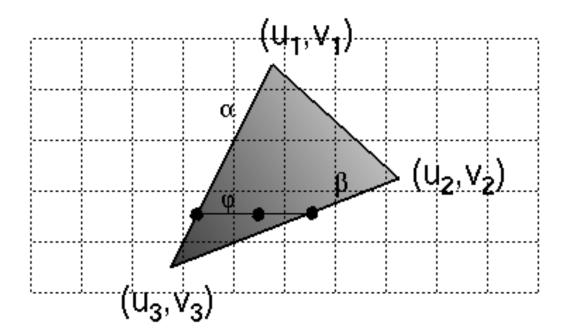




Texture Mapping



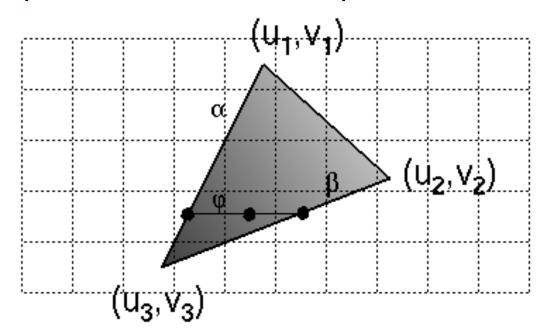
- Scan conversion
 - Interpolate texture coordinates down/across scan lines
 - Distortion due to bilinear interpolation approximation
 - » Cut polygons into smaller ones, or



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





Assume triangle attribute varies linearly across the triangle

Attribute's value at 3D (non-homogeneous) point $P = \begin{bmatrix} x & y & z \end{bmatrix}^T$ is then:

$$f(x, y, z) = ax + by + cz$$

Get 2D homogeneous representation : $\begin{bmatrix} x_{\text{2D-H}} & y_{\text{2D-H}} & w \end{bmatrix}^T = \begin{bmatrix} x & y & z \end{bmatrix}^T$



Assume triangle attribute varies linearly across the triangle

Attribute's value at 3D (non-homogeneous) point $P = \begin{bmatrix} x & y & z \end{bmatrix}^T$ is then:

$$f(x, y, z) = ax + by + cz$$

Get 2D homogeneous representation : $\begin{bmatrix} x_{\text{2D-H}} & y_{\text{2D-H}} & w \end{bmatrix}^T = \begin{bmatrix} x & y & z \end{bmatrix}^T$

Rewrite attribute equation for f in terms of 2D homogeneous coordinates:

$$f = ax_{2D-H} + by_{2D-H} + cw$$



Assume triangle attribute varies linearly across the triangle

Attribute's value at 3D (non-homogeneous) point $P = \begin{bmatrix} x & y & z \end{bmatrix}^T$ is then:

$$f(x, y, z) = ax + by + cz$$

Get 2D homogeneous representation : $\begin{bmatrix} x_{\text{2D-H}} & y_{\text{2D-H}} & w \end{bmatrix}^T = \begin{bmatrix} x & y & z \end{bmatrix}^T$

Rewrite attribute equation for $\,f\,$ in terms of 2D homogeneous coordinates:

$$f = ax_{2D-H} + by_{2D-H} + cw$$

$$\frac{f}{w} = a\frac{x_{\text{2D-H}}}{w} + b\frac{y_{\text{2D-H}}}{w} + c$$



Assume triangle attribute varies linearly across the triangle

Attribute's value at 3D (non-homogeneous) point $P = \begin{bmatrix} x & y & z \end{bmatrix}^T$ is then:

$$f(x, y, z) = ax + by + cz$$

Get 2D homogeneous representation : $\begin{bmatrix} x_{\text{2D-H}} & y_{\text{2D-H}} & w \end{bmatrix}^T = \begin{bmatrix} x & y & z \end{bmatrix}^T$

Rewrite attribute equation for $\,f\,$ in terms of 2D homogeneous coordinates:

$$f = ax_{2D-H} + by_{2D-H} + cw$$

$$\frac{f}{w} = a\frac{x_{2D-H}}{w} + b\frac{y_{2D-H}}{w} + c$$

$$\frac{f}{w} = ax_{2D} + by_{2D} + c$$

Where $\begin{bmatrix} x_{2\mathrm{D}} & y_{2\mathrm{D}} \end{bmatrix}^T$ are projected screen 2D coordinates (after homogeneous divide)



Assume triangle attribute varies linearly across the triangle

Attribute's value at 3D (non-homogeneous) point $P = \begin{bmatrix} x & y & z \end{bmatrix}^T$ is then:

$$f(x, y, z) = ax + by + cz$$

Get 2D homogeneous representation : $\begin{bmatrix} x_{\text{2D-H}} & y_{\text{2D-H}} & w \end{bmatrix}^T = \begin{bmatrix} x & y & z \end{bmatrix}^T$

Rewrite attribute equation for f in terms of 2D homogeneous coordinates:

$$f = ax_{2D-H} + by_{2D-H} + cw$$

$$\frac{f}{w} = a\frac{x_{2D-H}}{w} + b\frac{y_{2D-H}}{w} + c$$

$$\frac{f}{w} = ax_{2D} + by_{2D} + c$$

Where $\begin{bmatrix} x_{2\mathrm{D}} & y_{2\mathrm{D}} \end{bmatrix}^T$ are projected screen 2D coordinates (after homogeneous divide)

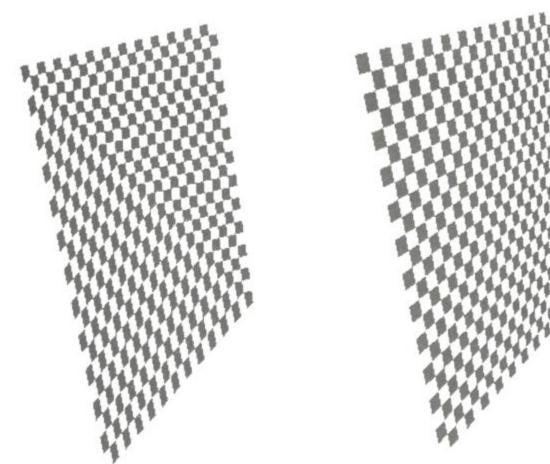
So ... $\frac{f}{dt}$ is affine function of 2D screen coordinates...



- Compute at each vertex <u>after perspective</u> transformation:
 - "Numerators" s/w, t/w
 - "Denominator" 1/w

- Linearly interpolate s/w, and t/w and 1/w across the polygon
- At each pixel:
 - Perform perspective division of interpolated texture coordinates (s/w, t/w) by interpolated 1/w (i.e., numerator over denominator) to get (s, t)





Linear interpolation of texture coordinates

Correct interpolation

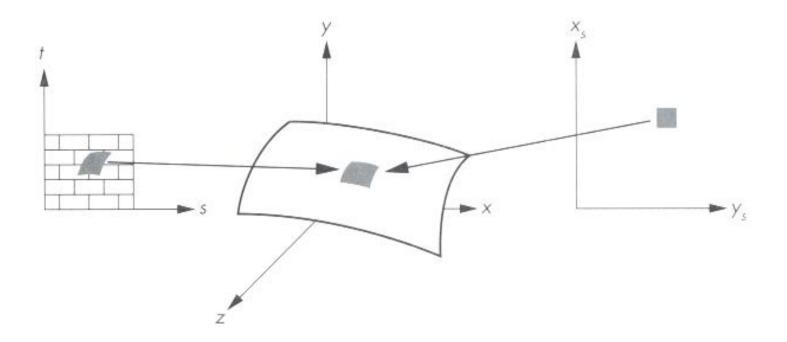
Texture Overview



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

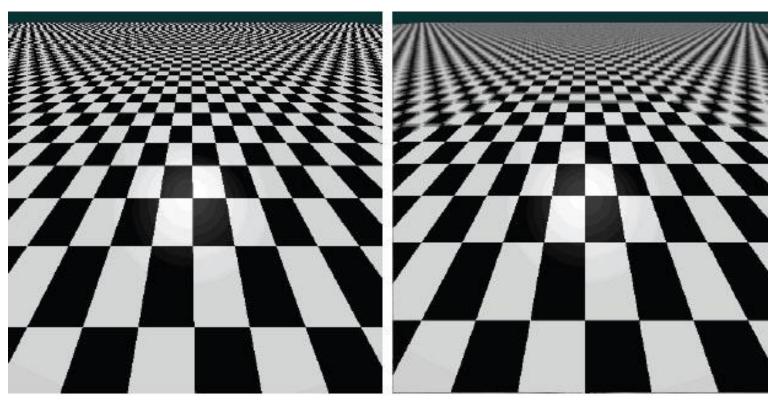


 Must sample texture to determine color at each pixel in image





Aliasing is a problem

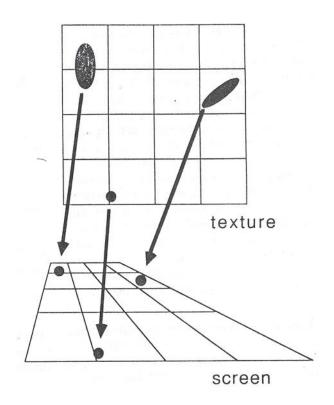


Point sampling

Area filtering



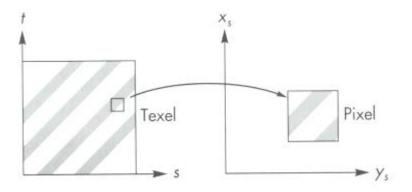
Ideally, use elliptically shaped convolution filters



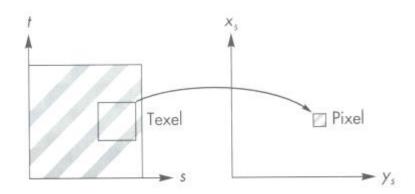
In practice, use rectangles or squares



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



Magnification

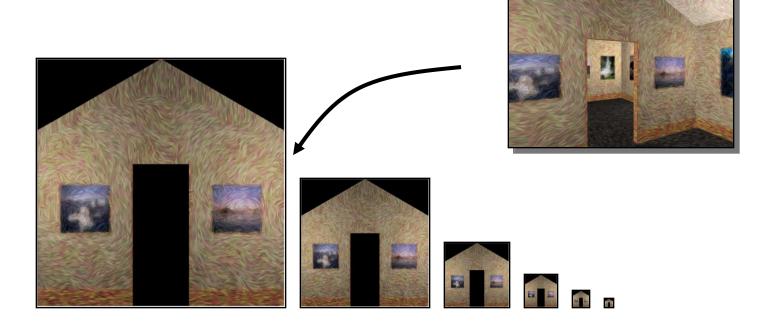


Minification

Mipmaps



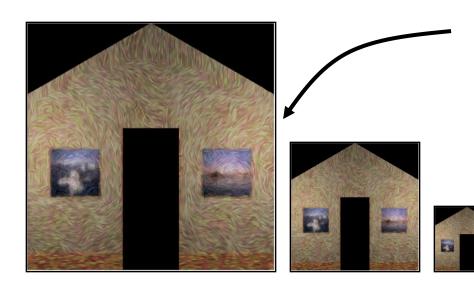
- Keep textures prefiltered at multiple resolutions
 - Usually powers of 2



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





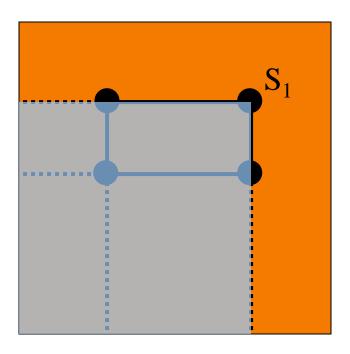






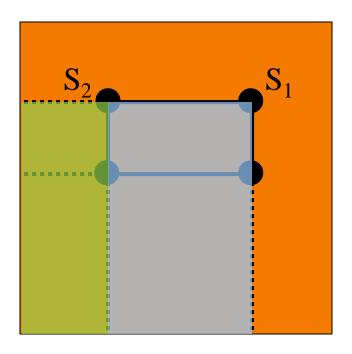


- At each texel keep sum of all values down & left
 - To compute sum of all values within a rectangle,
 simply combine four entries: S₁



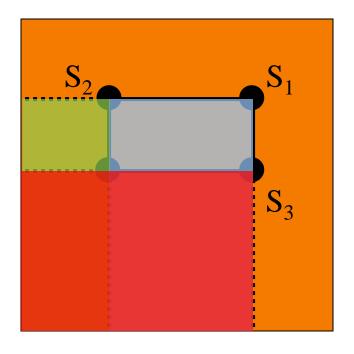


- At each texel keep sum of all values down & left
 - To compute sum of all values within a rectangle,
 simply combine four entries: S₁ S





- At each texel keep sum of all values down & left
 - To compute sum of all values within a rectangle, simply combine four entries: $S_1 S_2 S_3$

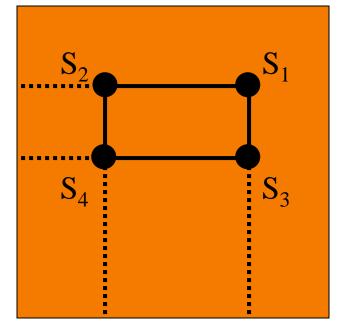




- At each texel keep sum of all values down & left
 - To compute sum of all values within a rectangle, simply combine four entries: $S_1 S_2 S_3 + S_4$

Better ability to capture oblique projections,

but still not perfect



(Mipmaps are more common.)

Texture Overview

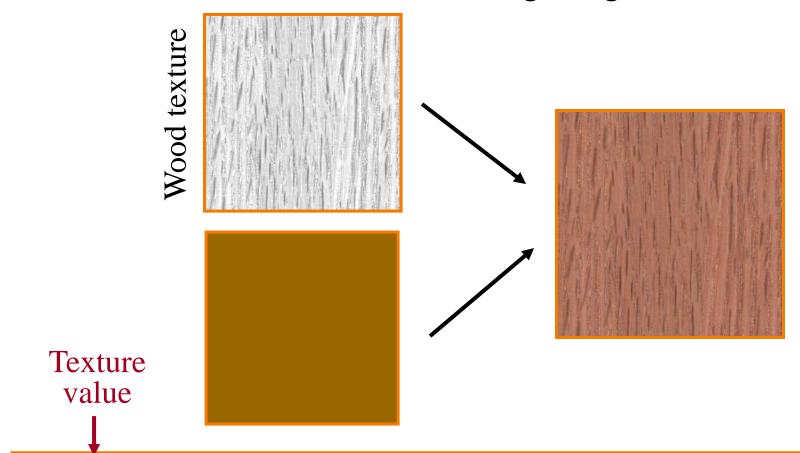


- Texture mapping stages
 - Parameterization
 - Mapping
 - Filtering
- Texture mapping applications
 - Modulation textures
 - Illumination mapping
 - Bump mapping
 - Environment mapping
 - Image-based rendering

Modulation textures



Texture values scale result of lighting calculation



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

Illumination Mapping



Map texture values to surface material parameter

- ∘ K_A
- ∘ K_D
- ∘ K_S
- \circ K_T
- o n



Texture value

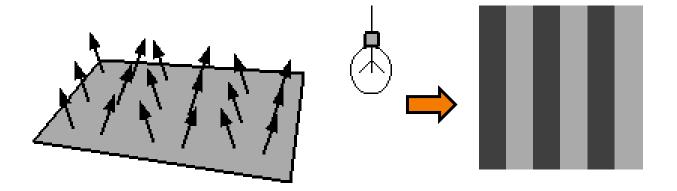
$$I = I_E + K_A I_A + \sum_L (K_D(s,t)(N \cdot L) + K_S(V \cdot R)^n) S_L I_L + K_T I_T + K_S I_S$$

Bump/Normal Mapping



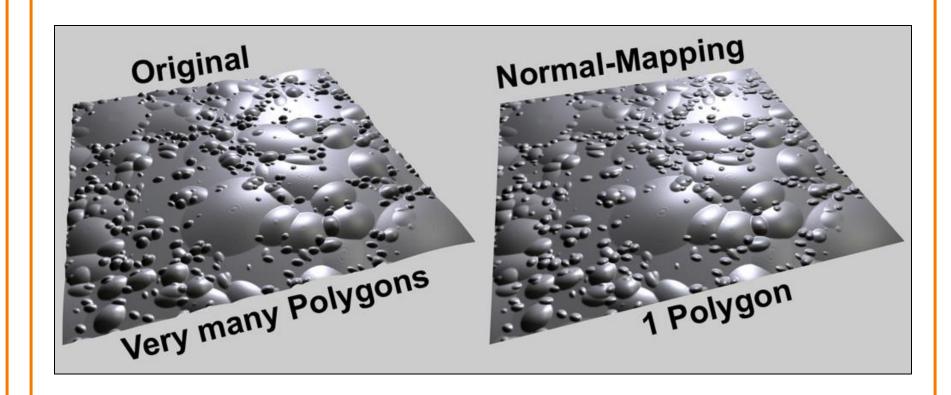
Texture values perturb surface normals:

- Encode normals (or offsets) in RGB
- Use gradient of grayscale image ("bump")



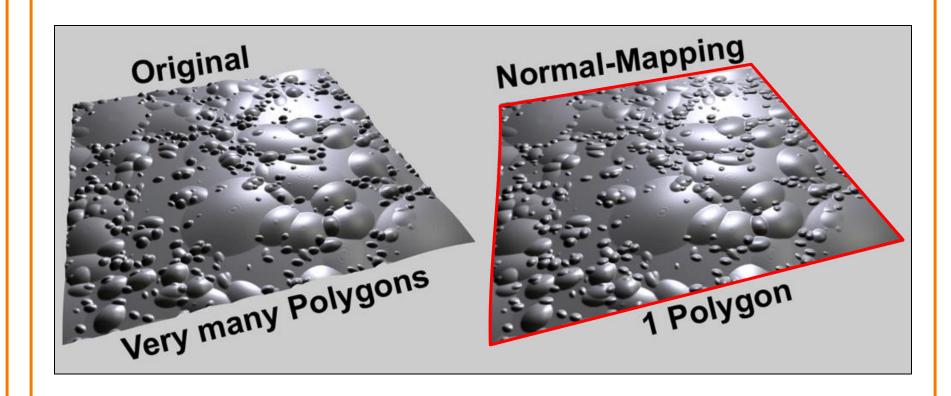
Normal Mapping





Normal Mapping

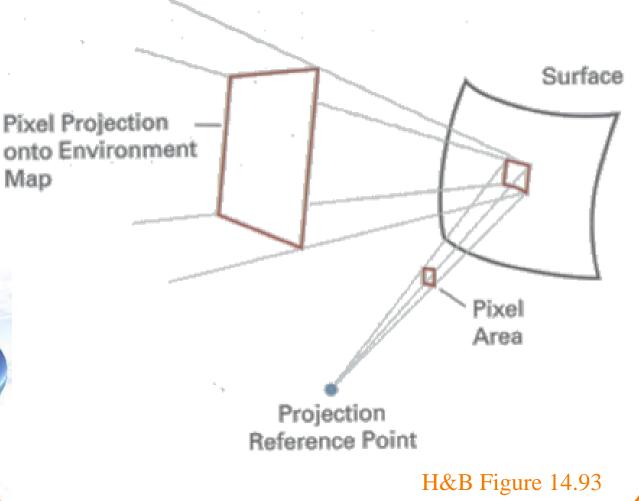




Environment Mapping



Texture values are reflected off surface patch



Gamer3D/Wikipedia

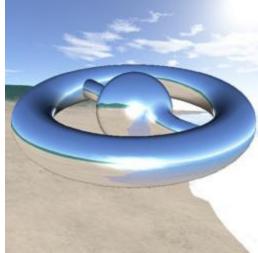


Image-Based Rendering



Map photographic textures to provide details for coarsely detailed polygonal model

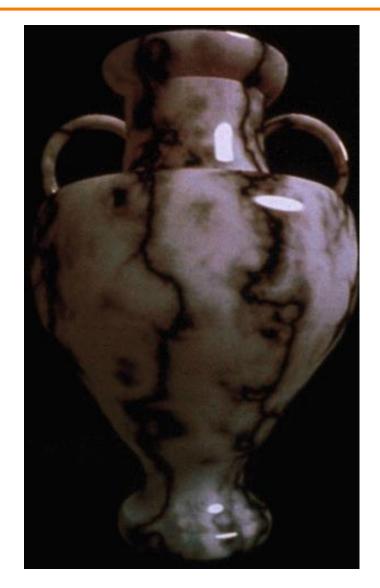


Solid textures



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

• Expensive storage, or

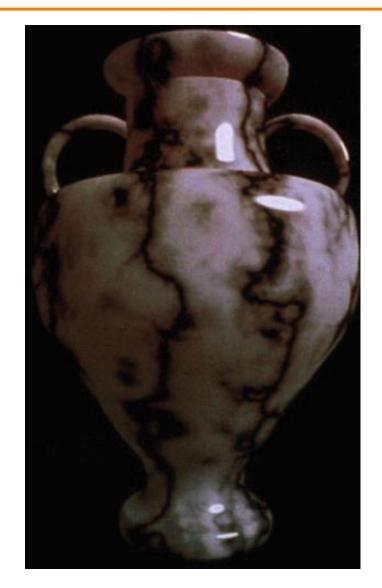


Solid textures



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

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



Rasterization

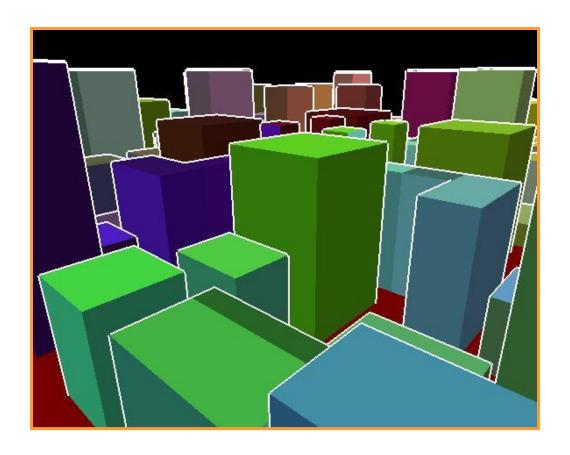


- Scan conversion
 - Determine which pixels to fill
- Shading
 - Determine a color for each filled pixel
- Texture mapping
 - Describe shading variation within polygon interiors
- Visible surface determination
 - Figure out which surface is front-most at every pixel

Visible Surface Determination



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

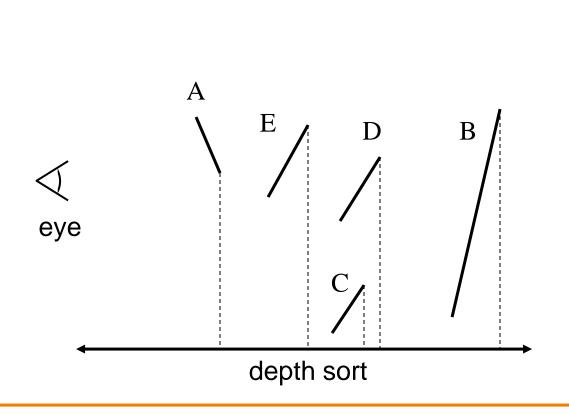


Depth sort



"Painter's algorithm"

1. First **sort surfaces** in order of decreasing <u>maximum</u> depth





Depth sort

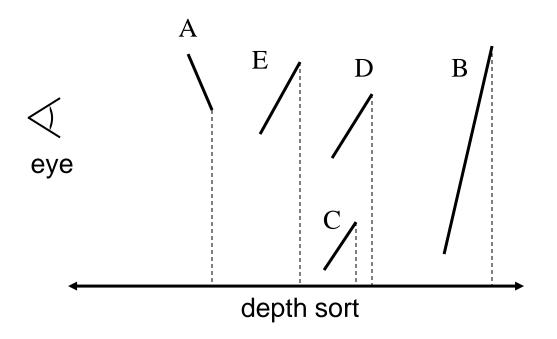


"Painter's algorithm"

1. First **sort surfaces** in order of decreasing <u>maximum</u> depth

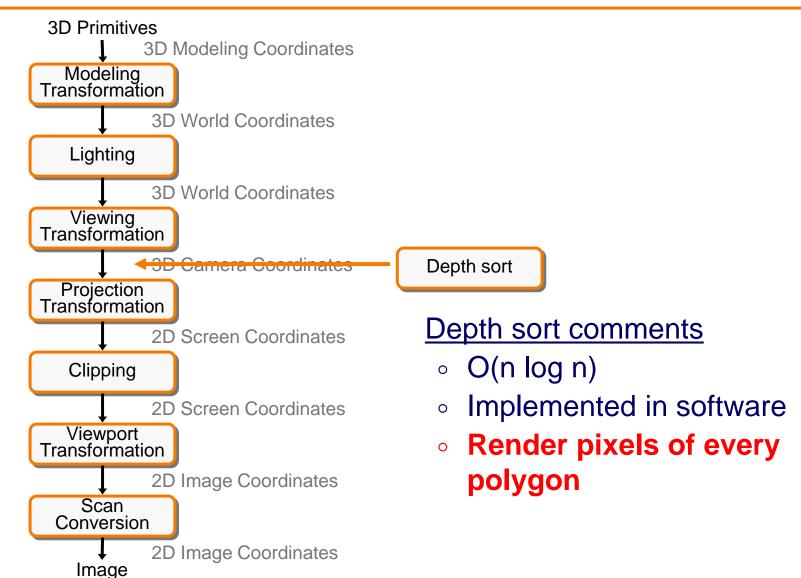
2. Scan convert surfaces in back-to-front order,

overwriting pixels



3D Rendering Pipeline



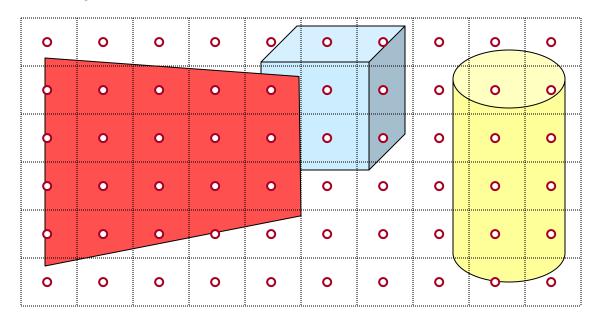


Z-Buffer



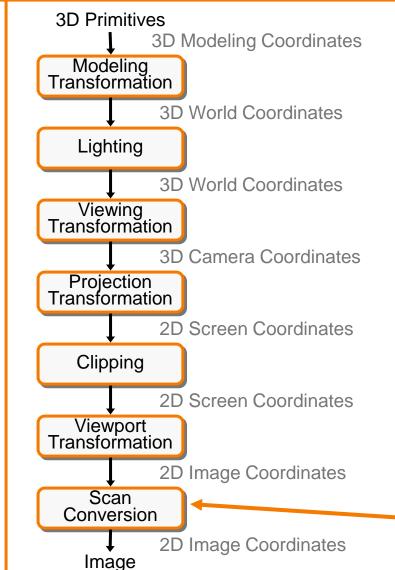
Maintain color & depth of closest object per pixel

- Framebuffer now RGBAz initialize z to far plane
- Update <u>only pixels with depth closer than in z-buffer</u>
- Depths are interpolated for in-primitive pixels from vertices, just like colors



Z-Buffer

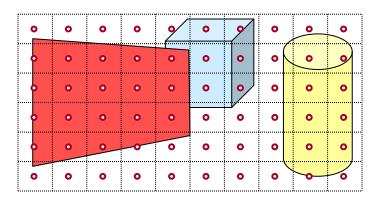




Z-buffer comments

Z-Buffer

- + Polygons rasterized in any order
- + Process one polygon at a time
- + Suitable for hardware pipeline
- Requires extra memory for z-buffer
- Commonly in hardware



Hidden Surface Removal Algorithms

OPAQUE-OBJECT



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

A Characterization of Ten Hidden-Surface Algorithms

ALGORITHMS										
					1					
		COMPARIS	SON ALGORITHMS	OBJECT SPACE	(partly each)	IMAGE SPACE	DEPTH PRIORIT	Y ALCORITHMS		
				ODDET OTHER	(,,,,	IMAGE SPACE	DEFIN TRIORIT	1 ALGORITHMS		
	edges edges volumes						\ .		ANALES AND LE PROPERTIES	10 U
	,			(LIST PRIORITY	`	area sampli	ng	point sampling	
					ALGORITHMS	dynamicall.	1			
					priority	computed	\			
	•	•	•	•	•	•	7			•
	APPEL 1967	GALIMBERTI, et al	LOUTREL 1967	ROBERTS 1963	SCHUMACKER, et al	NEWELL, et al	WARNOCK	WATKINS	ROMNEY, et al	BOUKNIGHT
RESTRICTIONS	TP,NP	TP,NP	TP,NP	TP, CC, CF, NP	CF, NP, LS (TP)	1972 None	1968 (TR) None	1970 Kone	1967 TR,CF,NP	1969
								None	IR,CE,AP	
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	None used	a col	inline x	St line De Co	nline oherence
				Anna cur orași protestat anna est						
SORTING	Back Edge Cull 1) Edges separating	Back Edge Cull Description Back Edge Cull Back Edge Cull	Back Edge Cull 1) Edges separating back-facing planes 2) Dot product with	Back Edge Cull 1) Edges separating	Intra-Cluster Priority	7 507 30 mp	Z Sort (Opt) 1) Faces, max 2	Y Sort	Y S	Y Sort 1) Edges, Min Y
What,	back-facing planes 2) Dot product with normals & topology			2) Dot product with	visibility	max vints	2) Comparison of max points	2) an on 3) a ket	2) Comparison	2) Comparison 3) Bucket
what prop- erty	3) Cull 4) List of edges, E ₅ 5) 1, E _t	3) Cull 4) List of edges, E,	3) Cull 4) List of edges, E _s 5) 1, E _t	3) Cull	normals	3) Ordered table	3) n log m 4) Ordered	5) 1, e os	3) 2 bucket 4) Table of lists 5) 1, F _r	(4) Table of lists (5) 1, E,
(2) Method	5) 1, E _t	5) 1, E _t	5) 1, E _t	5) 1, E _t	Ordere	5) 1, F _T	5) 1, F _T		5) 1, F _r	
(3)	Contour Edge Cull 1) Edges separating			F 22 (D)	ter-Cluster	Newell Special	pec	X Merge	X Sprt	X Merge
Type	front & back faces 2) Dot product with			51 00 10	Priority Clusters 2) Dot product with	I) Faces, p.	7) mini-may	2) Edges, X value 2) Comparison	1) Edges, X value 2) Comparison	X Merge 1) Edges, X value 2) Comparison
(4) Result structure	normals & tanctony	(Omstted)	(Omitted)	Cul	separating planes 3) Prefix scan	b) S, S TOP	n X and Y, sum of angles g3) Radix 4 subdivi-	3) Merge (ordred) 4) 2-way linked list	3) 2 bucket 4) Table of lists	3) Merge (ordered) 4) Linked list
(5)	3) Cull 4) List, E _c 5) 1, E _e	1		15 Es	binary tree 4) Gred tab	(4) ere able	sion with overlap 4) Stacks of	s) Er, Sa	5) n. S ₂	5) Er, 2St (edges)
(5) Number per frame, num-	(8)				5) 7 (5		unordered tables 5) L, F,/factor 1			,
ber of ob-	Initial Visibility 1) Ray to vertex	1) Ra rte	Visibility to vertex	Edge/Volume Test	Cu.	Y Sort 1) Face segment	Depth Search	X Sort	X Priority Search 1) Edges, X value	1) X Sort 1) Edges, X value
(merge) Number of	against all faces	against a faces 2) Depth	against all faces 2) Betweenness.	relative vol. (2) t j duct th	by Y range 2) Y intercept	1) Surrounder faces 2) 4-corner compare 3) Exhaustive	l) Segments, λ left 2) Comparison	(2) Comparison	2) Comparison
new entries	3) Exhau v	i) Exhau e search	surroundedness 3) Exhaustive seare 4) Quantitative	Pro smmil	5) Cull 4) Smaller ordered	3) Bucket 4) None	4) Answer/failure 5) L _v , F _r /factor 2	5) Bubble 4) 2-way linked	3) Priority search 4) Active segment list	3) Bubble 4) 1-way linked list 5) N, 2S _£ (edges)
length of list	4) Quanta tiv) Quantitative isibility of vertex) fobjects, F.	4) Quantitative visibility 5) #objects	5) s edges,	5) 1, F _t	5) F + split faces	7, -4, -4,	115t 5) n, Sa	5) n, m	3) 11, 252 (6424)
umber				Foojects					İ	
earches ength o	1) rsection	Edge Intersection	In sec on one E		Y Cull 1) Faces by Y extent	X Merge	TV Sort (Opt) Sort windows into	Span Cull	2 Search	Z Search 1) Segments, depth
st	2) Penetration	2) Inters	In rsect in		2) Mini-max on X intercepts	X intercept 2) Comparison	scan-line order if	with sample span 2) Double comparison	2 Search 1) Segments, depth 2) Linear equations	1) Segments, depth 2) Linear equations and comparison
	with sweep triangle 3) Cull (unordered) 4) Intersection list	y (rdey)	Cull (unordered) 4) Intersection list		3) Cull (unordered) 4) X intercepts of relevant segments	3) Ordered merge 4) Ordered list		3) Cull ordered list	(1) Search (unordered	3) Search of un- ordered active list
	S) E, Ec	Es	5) E _s , E _s - 1		5) n, E _s	5) S _r , S _v /2		5) n*S _v * f (>1), S	4) Visible segment 5) n*2S _L ,D _C	4) Visible segment 5) n*2S, Dc
	Sort Along Edge 1) Intersections on	1) intersections on	Sort Along Edge 1) Intersections on		X Sort 1) Segments			Z Search 1) Segments, Z	(Omitted if X (priorities same as (last time)	
	edge, ordering	edge, ordering	edge, ordering		2) Counters 3) Hardware			2) Depth by logarithmic search	last time)	1
	3) Bubble 4) Answer	4) Answer 5) E _s , X _v /E _s	4) Answer 5) E _s . X _v /E _s	1	4) Segments at this X			 Search (unordered Visible segment 	1)	
	5) E _s , X _V /E _s Omit if well hidden)		(Omit if well hidden		5) nm, S _t			5) n*S _v *f(>1), D _c		
					Priority Search 1) Segments, priorit	Į.				1
					2) Logic network 3) Logic network	1			-	
	Í				4) Visible segment 5) nm, St					
	I	I.	l,	1		1			ΓC	/1 1

Figure 29. Characterization of ten opaque-object algorithms 6. Comparison of the algorithms.

[Sutherland '74]

Rasterization Summary

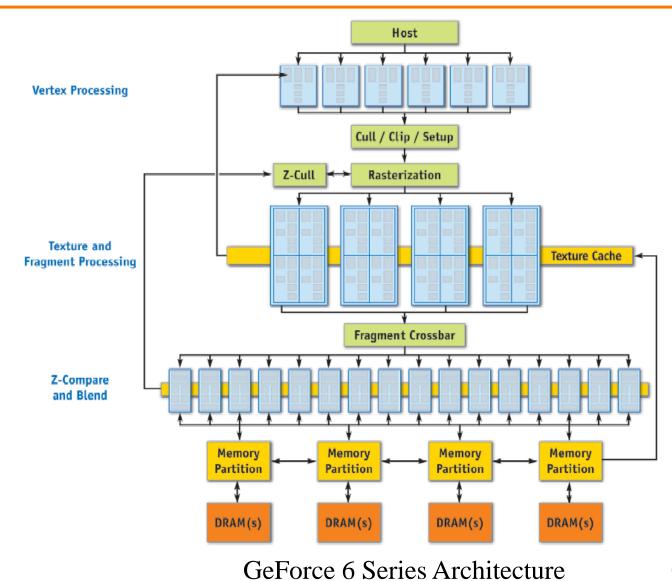


- Scan conversion
 - Sweep-line algorithm
- Shading algorithms
 - Flat, Gouraud
- Texture mapping
 - Mipmaps
- Visibiliity determination
 - Z-buffer

This is all in hardware

GPU Architecture



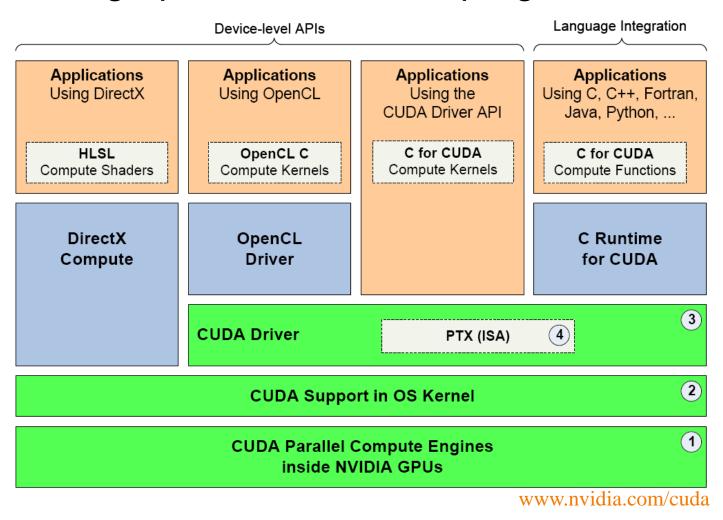


GPU Gems 2, NVIDIA

Actually ...



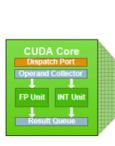
Modern graphics hardware is programmable



Trend ...



GPU is general-purpose parallel computer







www.nvidia.com/cuda