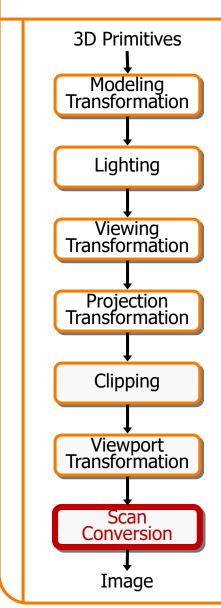


Scan Conversion

COS 426, Spring 2020 Felix Heide Princeton University

3D Rendering Pipeline (for direct illumination)





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

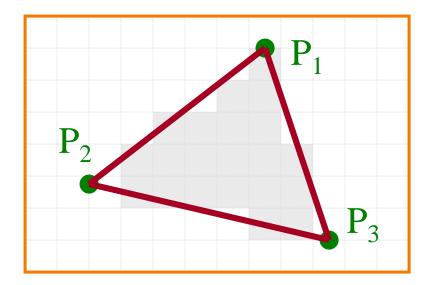


- Scan conversion (last time)
 - Determine which pixels to fill
- ➤ Shading
 - Determine a color for each filled pixel
- Texture mapping
 - Describe shading variation within polygon interiors
- Visible surface determination
 - Figure out which surface is front-most at every pixel

Shading



How do we choose a color for each filled pixel?

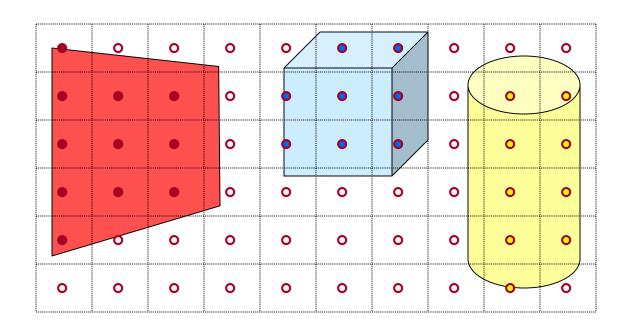


Emphasis on methods that can be implemented in hardware

Ray Casting



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

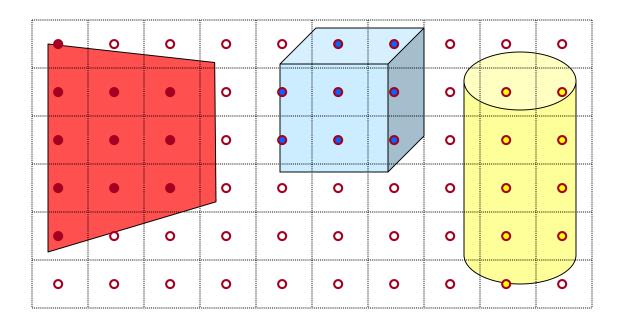


$$I = I_{E} + K_{A}I_{AL} + \sum_{i} \left(K_{D}(N \cdot L_{i})I_{i} + K_{S}(V \cdot R_{i})^{n} I_{i} \right)$$

Polygon Shading



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



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

Polygon Shading Algorithms

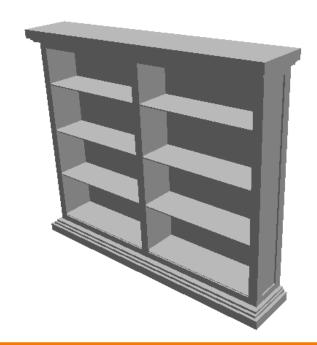


- Flat Shading
- Gouraud Shading
- Phong Shading

Flat Shading



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

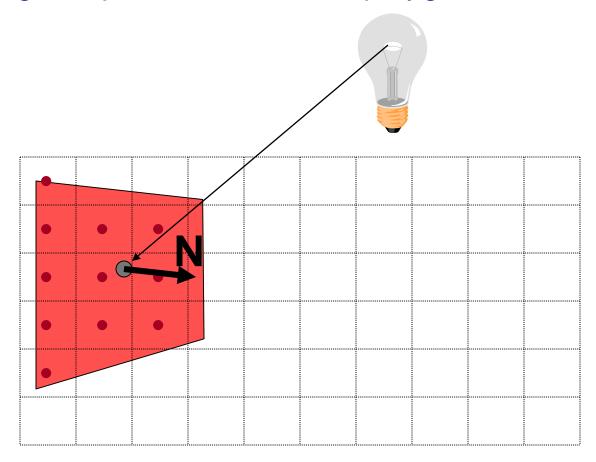


$$I = I_{E} + K_{A}I_{AL} + \sum_{i} (K_{D}(N \cdot L_{i})I_{i} + K_{S}(V \cdot R_{i})^{n}I_{i})$$

Flat Shading



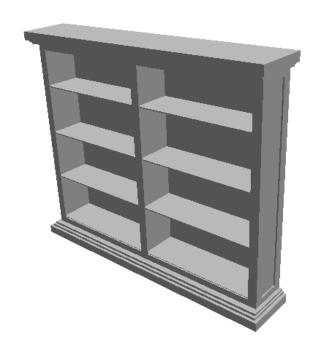
- One illumination calculation per polygon
 - Assign all pixels inside each polygon the same color

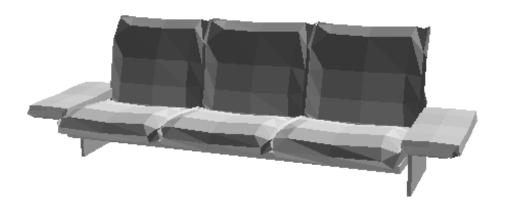


Flat Shading



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

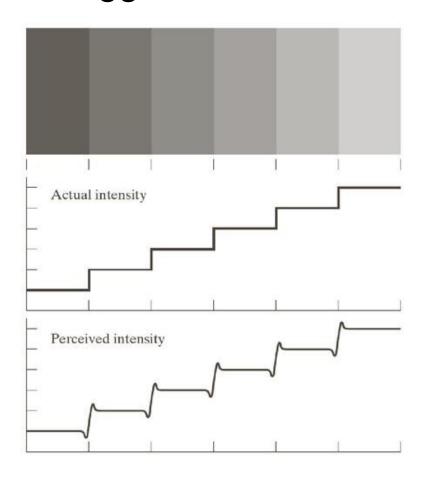




Mach Band Effect



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



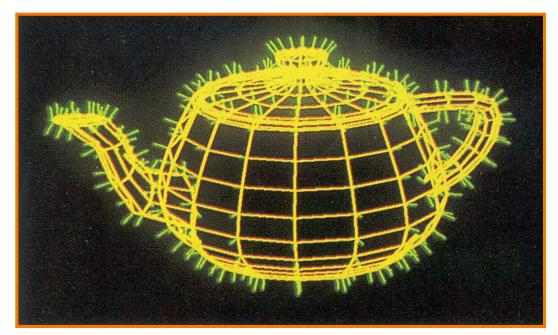
Polygon Shading Algorithms



- Flat Shading
- Gouraud Shading
- Phong Shading



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

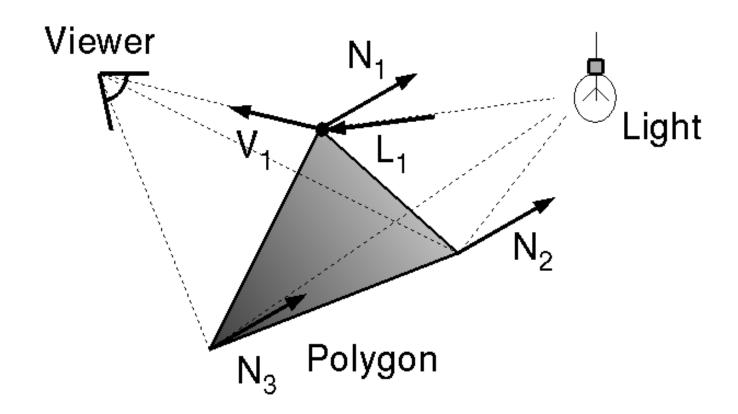


Watt Plate 7

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



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

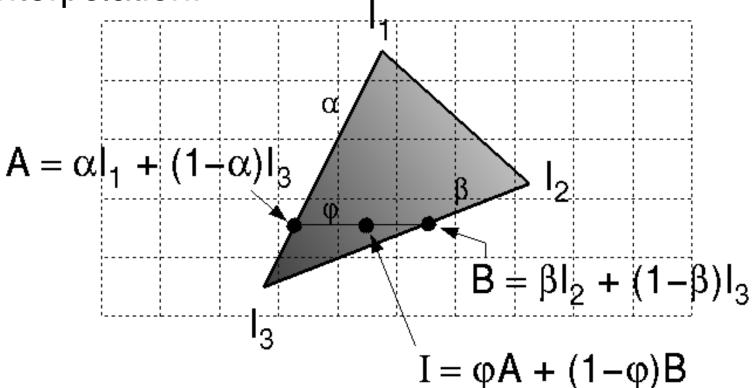




Bilinear interpolation of colors at vertices

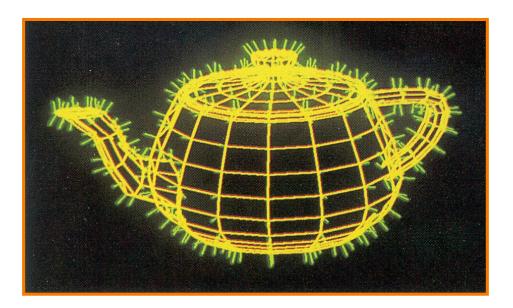
down and across scan lines = barycentric

interpolation!





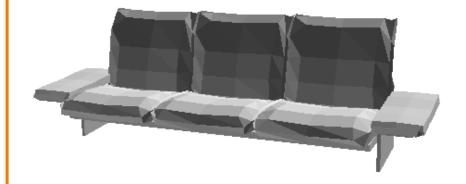
- Smooth shading over adjacent polygons
 - Curved surfaces
 - Illumination highlights
 - Soft shadows



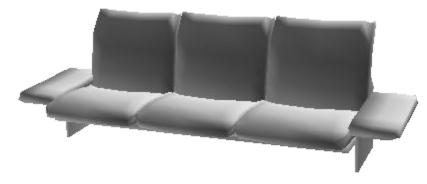
Mesh with shared normals at vertices



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





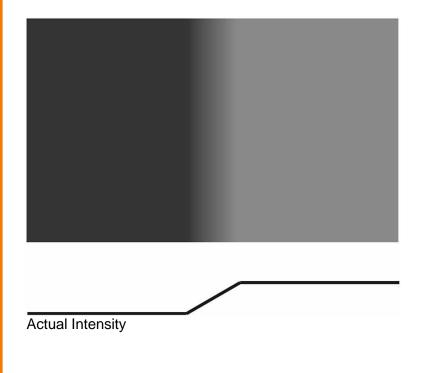


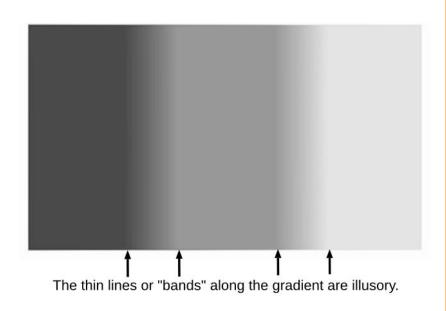
Gourand Shading

Mach Band Effect



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





Polygon Shading Algorithms

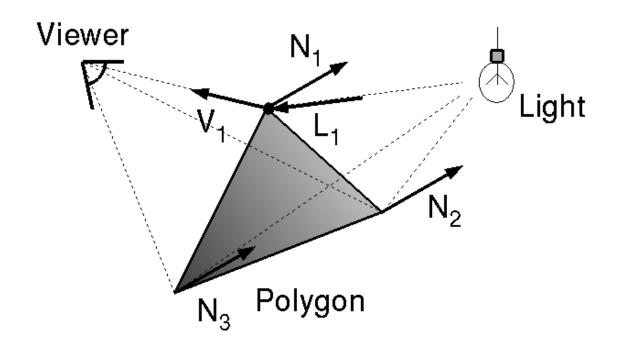


- Flat Shading
- Gouraud Shading
- Phong Shading (≠ Phong reflectance model)

Phong Shading



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

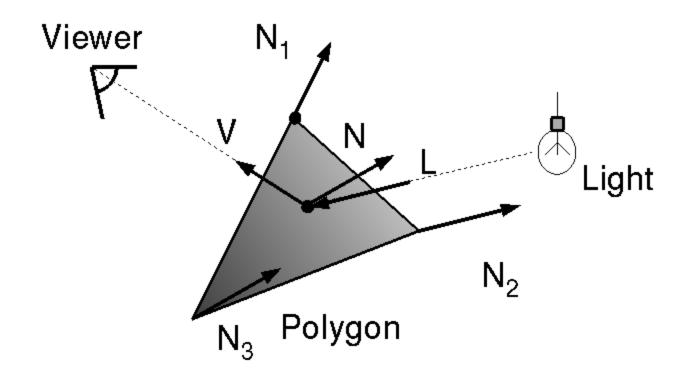


$$I = I_{E} + K_{A}I_{AL} + \sum_{i} \left(K_{D}(N \cdot L_{i})I_{i} + K_{S}(V \cdot R_{i})^{n} I_{i} \right)$$

Phong Shading



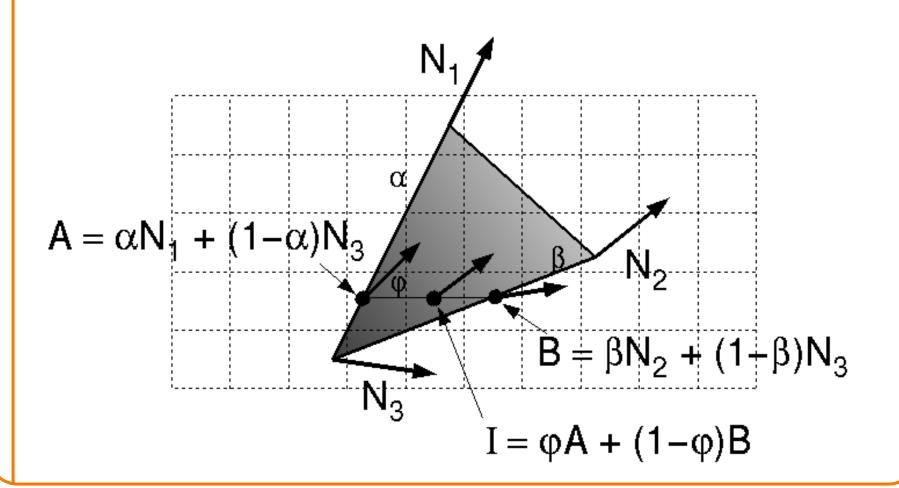
- One lighting calculation per pixel
 - Approximate surface normals for points inside polygons by bilinear interpolation of normals from vertices



Phong Shading



Bilinear interpolation of surface normals at vertices

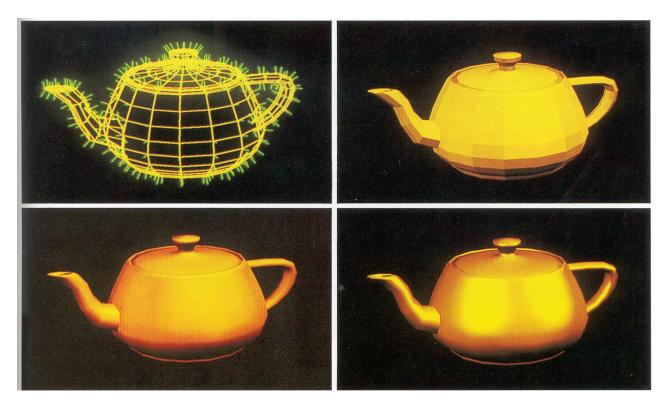


Polygon Shading Algorithms



Wireframe

Flat



Gouraud

Phong

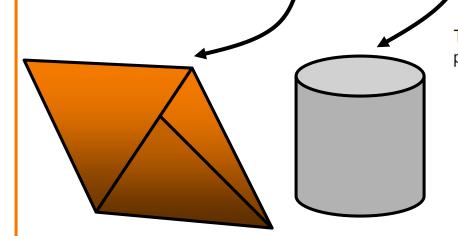
Demo: https://threejs.org/docs/scenes/material-browser.html#MeshPhongMaterial

Watt Plate 7

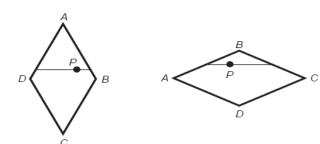
Shading Issues



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



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



Rasterization

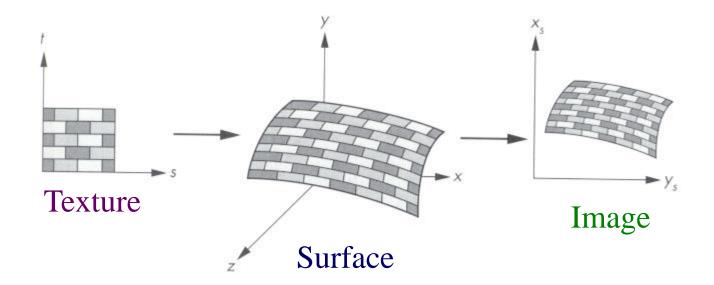


- Scan conversion
 - Determine which pixels to fill
- Shading
 - Determine a color for each filled pixel
- Texture mapping
 - Describe shading variation within polygon interiors
- Visible surface determination
 - Figure out which surface is front-most at every pixel

Textures



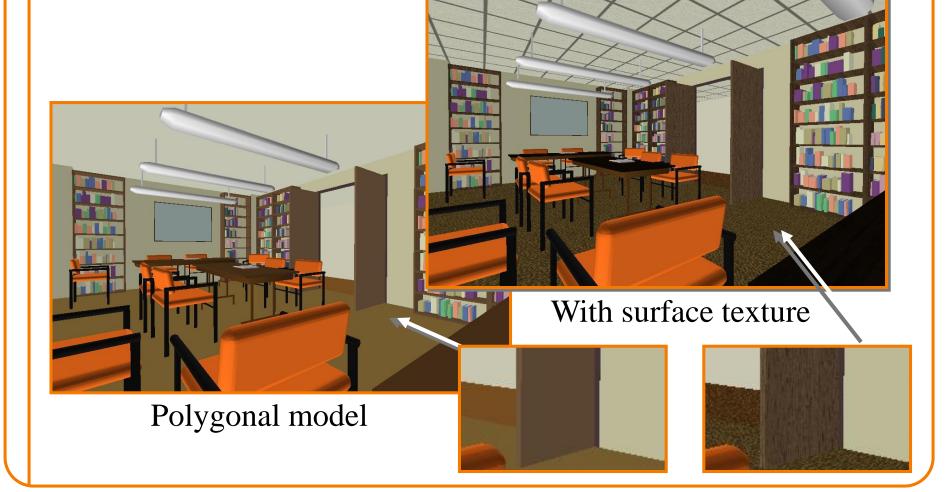
- Describe color variation in interior of 3D polygon
 - When scan converting a polygon, vary pixel colors according to values fetched from a texture image



Surface Textures



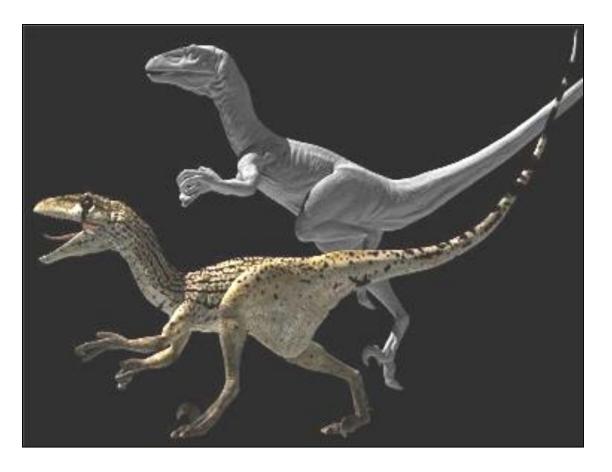
Add visual detail to surfaces of 3D objects



Textures



Add visual detail to surfaces of 3D objects



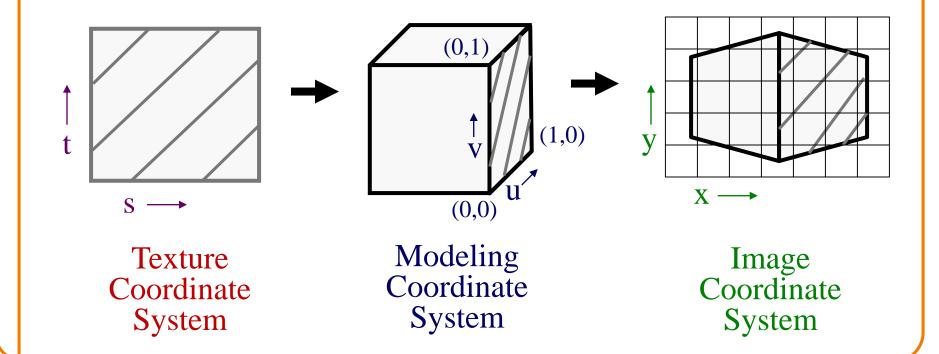


[Daren Horley]

Texture Mapping



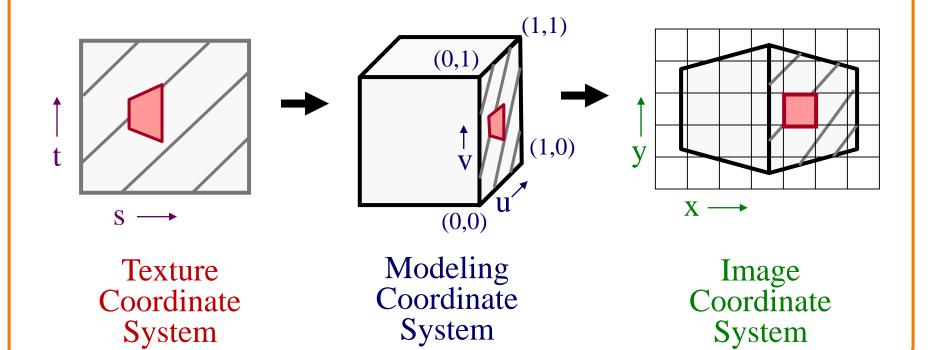
- Steps:
 - 1. Define texture
 - 2. Specify mapping from texture to surface
 - 3. Look up texture values during scan conversion



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

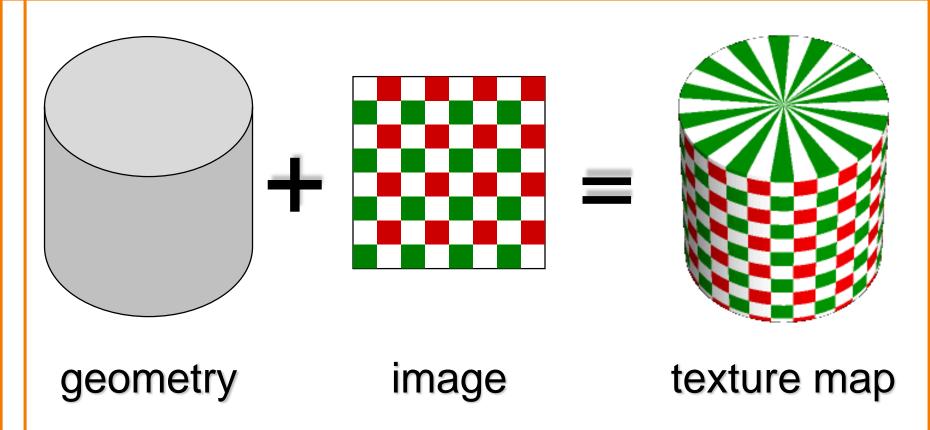
Texture Overview



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

Texture Parameterization

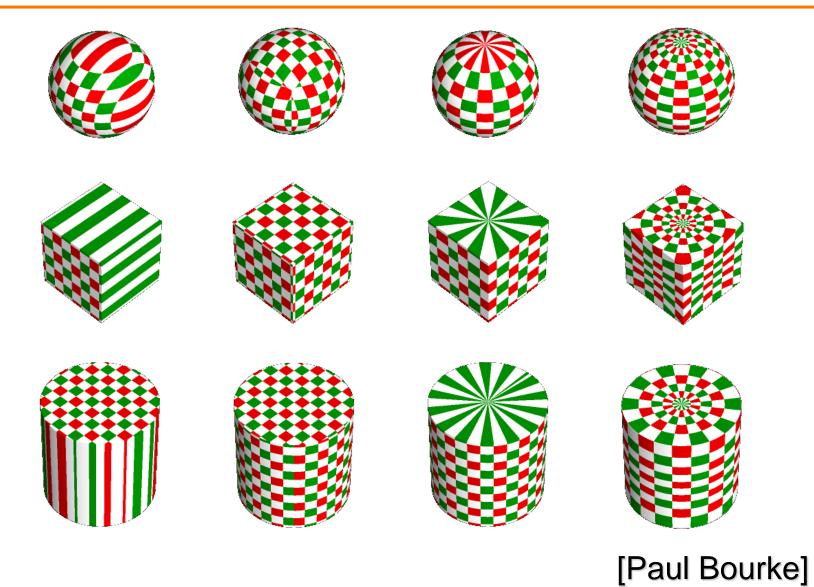




 Q: How do we decide where on the geometry each color from the image should go?

Texture Parameterization

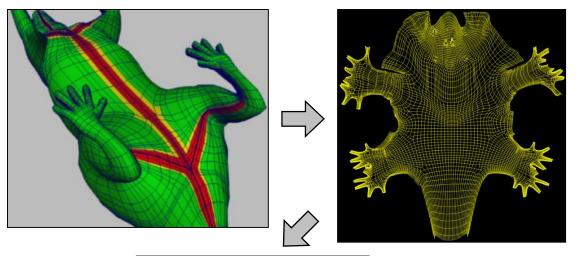




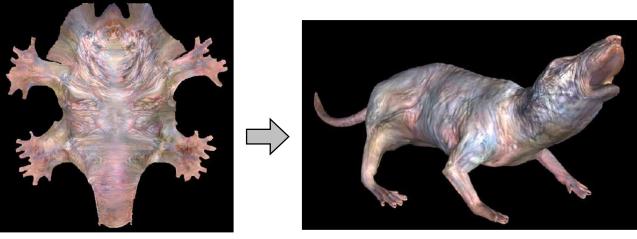
Texture Parameterization



Option1: unfold the surface



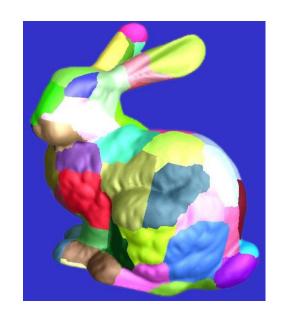
[Piponi2000]



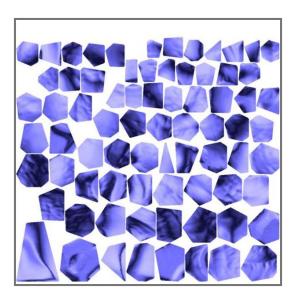
Texture Parameterization



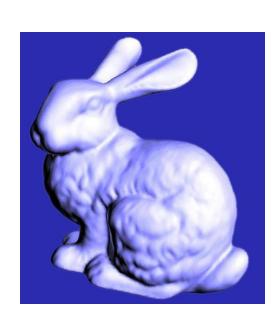
Option2: make an atlas



charts



atlas



surface

[Sander2001]

Texture Overview

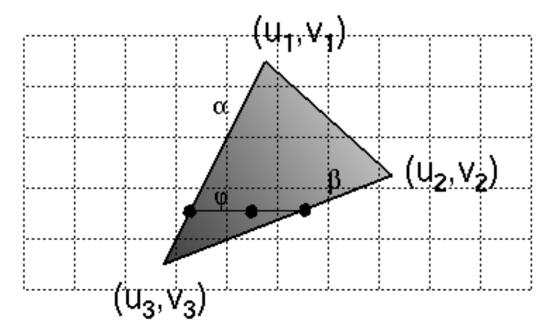


- Texture mapping stages
 - Parameterization
 - Mapping
 - Filtering
- Texture mapping applications
 - Modulation textures
 - Illumination mapping
 - Bump mapping
 - Environment mapping
 - Image-based rendering
 - Non-photorealistic rendering

Texture Mapping

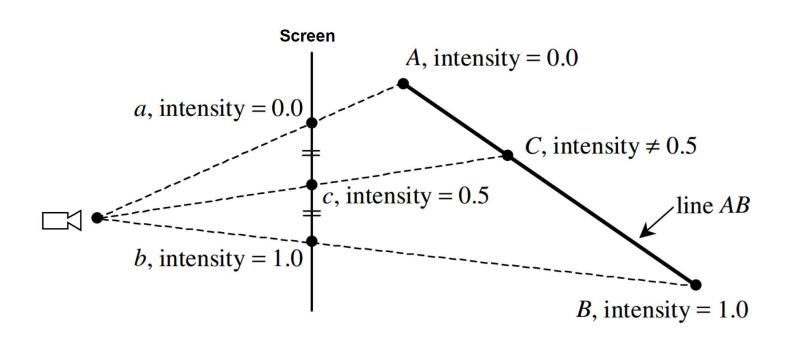


- Scan conversion
 - Interpolate texture coordinates down/across scan lines
 - Distortion due to bilinear interpolation approximation
 - » Cut polygons into smaller ones, or
 - » Perspective divide at each pixel



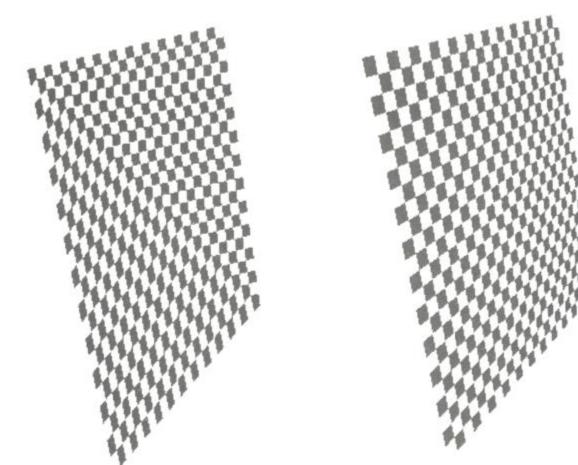
Perspective Divide





Perspective Divide





Linear interpolation of texture coordinates

Correct interpolation with perspective divide

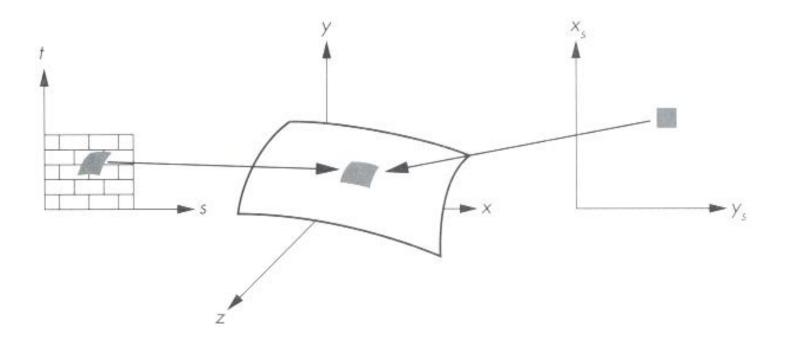
Texture Overview



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

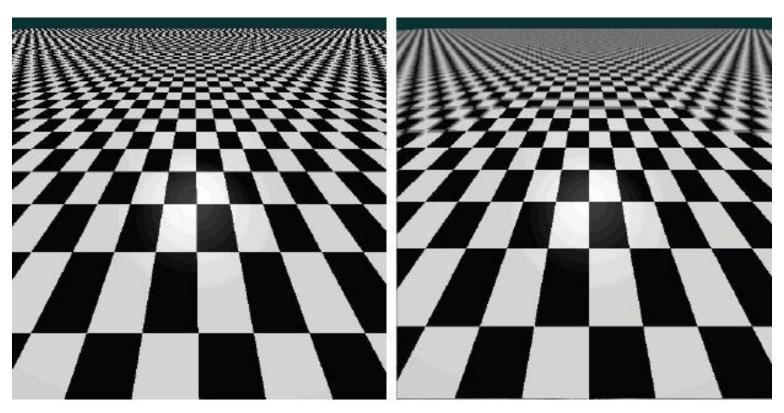


 Must sample texture to determine color at each pixel in image





Aliasing is a problem

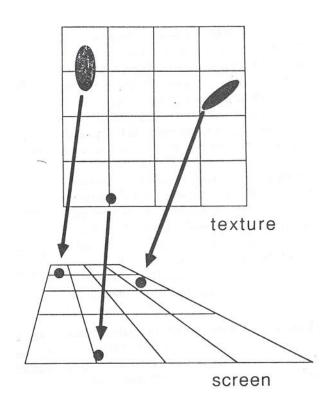


Point sampling

Area filtering



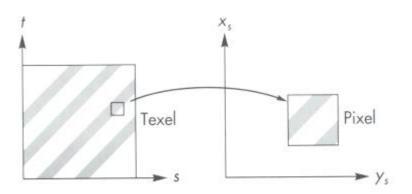
Ideally, use elliptically shaped convolution filters



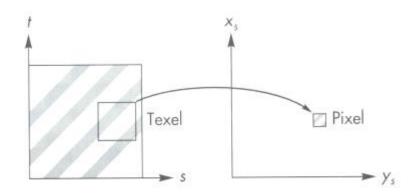
In practice, use rectangles or squares



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



Magnification

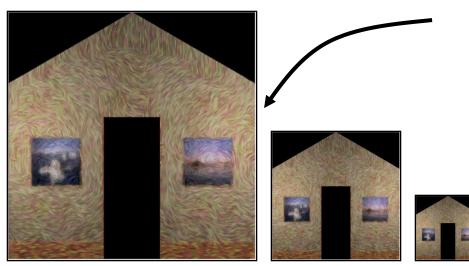


Minification

Mipmaps



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











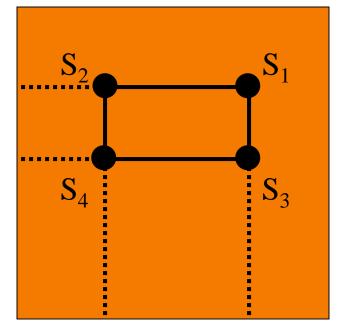
Summed-area tables



- At each texel keep sum of all values down & 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

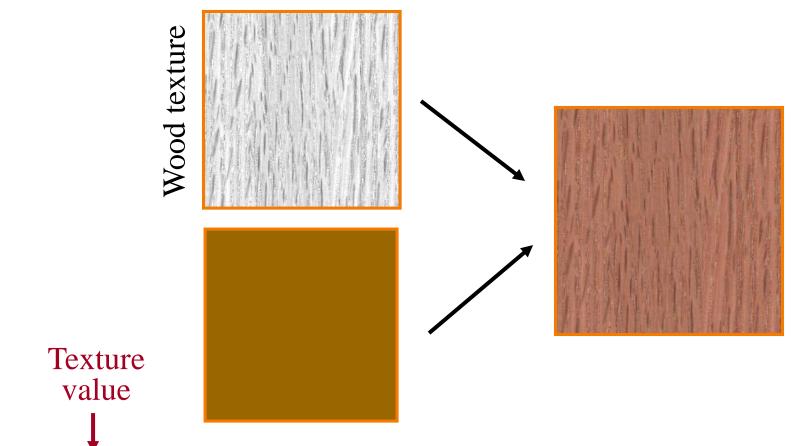


- Texture mapping stages
 - Parameterization
 - Mapping
 - Filtering
- Texture mapping applications
 - Modulation textures
 - Illumination mapping
 - Bump mapping
 - Environment mapping
 - Image-based rendering

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
- $\circ K_S$
- \circ K_T
- o n



Texture value

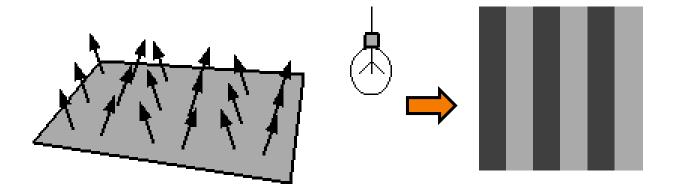
$$I = I_E + K_A I_A + \sum_L (K_D(s,t)(N \cdot L) + K_S(V \cdot R)^n) S_L I_L + K_T I_T + K_S I_S$$

Bump/Normal Mapping



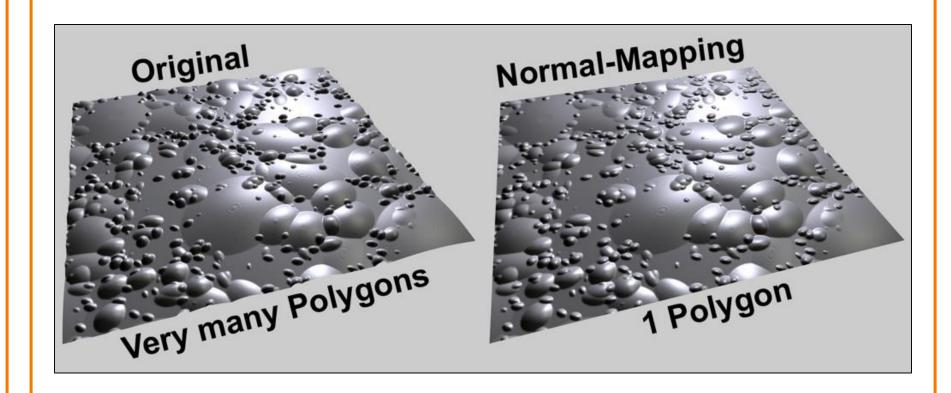
Texture values perturb surface normals:

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



Normal Mapping

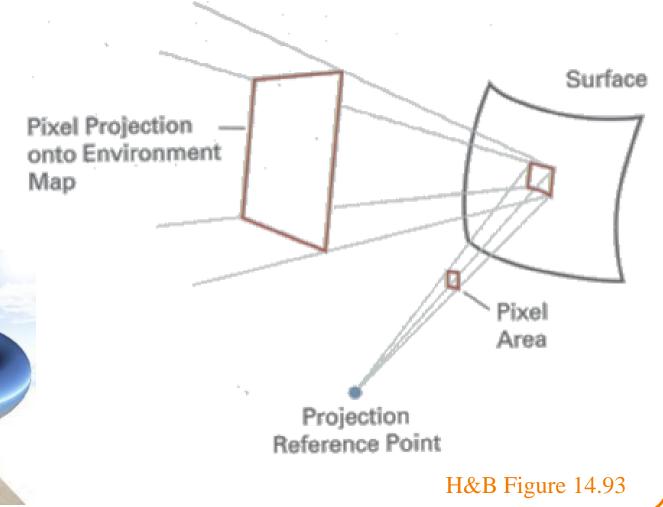




Environment Mapping



Texture values are reflected off surface patch



Gamer3D/Wikipedia

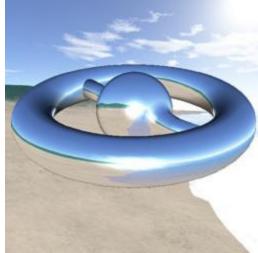
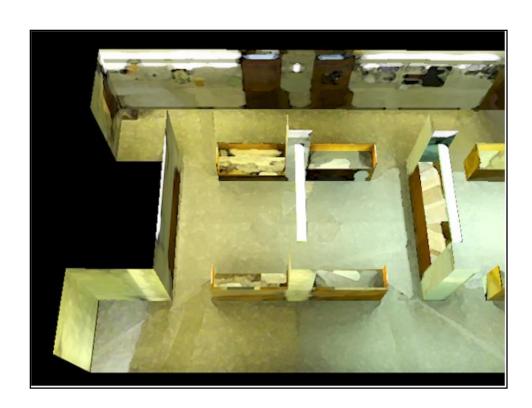


Image-Based Rendering



Map photographic textures to provide details for coarsely detailed polygonal model

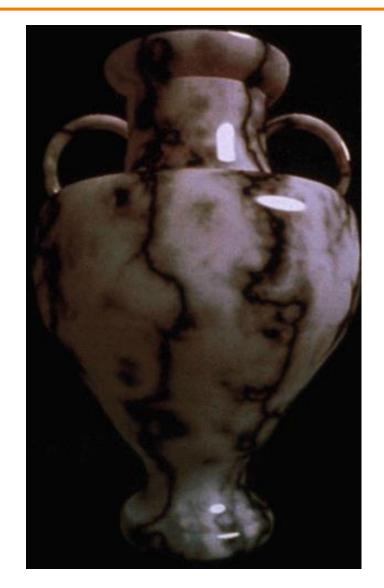


Solid textures



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

- Expensive storage, or
- 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

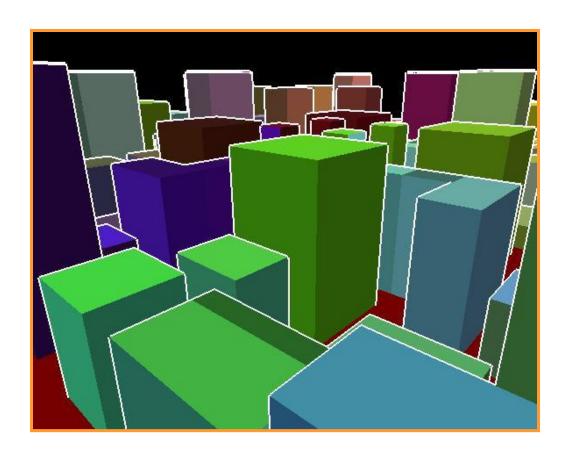


- Scan conversion
 - Determine which pixels to fill
- Shading
 - Determine a color for each filled pixel
- Texture mapping
 - Describe shading variation within polygon interiors
- Visible surface determination
 - Figure out which surface is front-most at every pixel

Visible Surface Determination



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



Depth sort

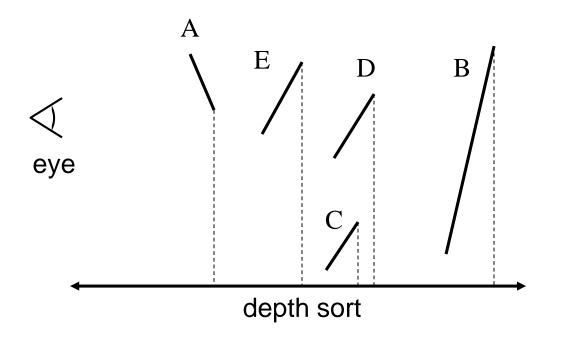


"Painter's algorithm"

First sort surfaces in order of decreasing maximum depth

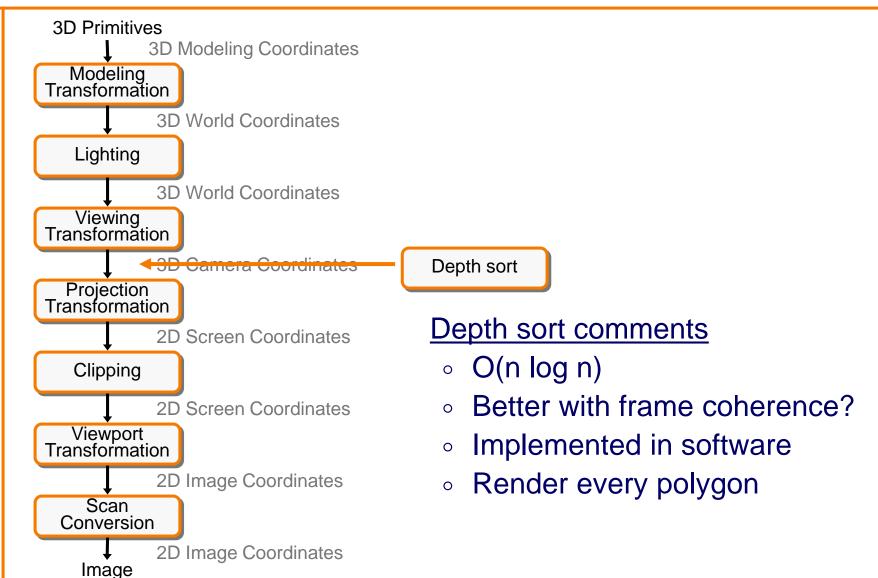
2. Scan convert surfaces in back-to-front order,

overwriting pixels



3D Rendering Pipeline



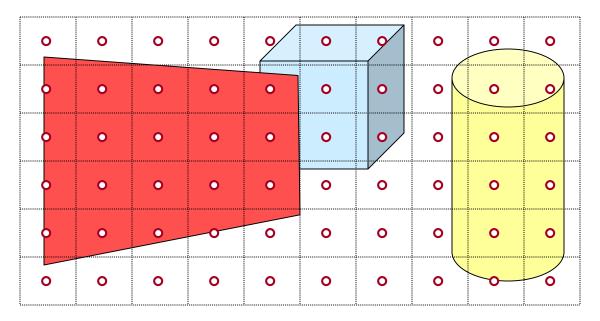


Z-Buffer



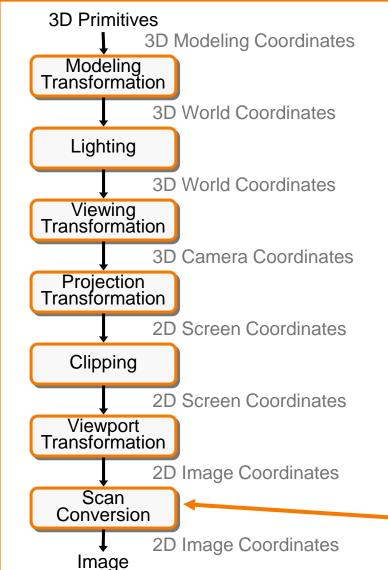
Maintain color & depth of closest object per pixel

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



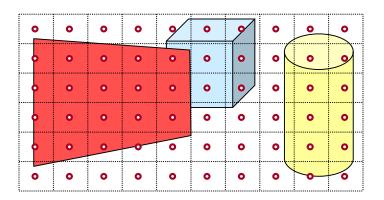
Z-Buffer





Z-buffer comments

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



Z-Buffer

Hidden Surface Removal Algorithms



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

A Characterization of Ten Hidden-Surface Algorithms

ALCORITIONS										
		COMPARTS	SON ALGORITHMS							
		COMPARIS	SON ALGORITHMS	OBJECT SPACE	(partly each)	IMAGE SPACE	DEPTH PRIORIT	Y ALGORITHMS		
		edges edges		edges volumes			\			
. edges edges				Cages volumes	LIST PRIORITY		area sampli	ng	point sampling	
					ALGORITHMS .	dynamicall,	\			
					priority	computed	\			
		•	•		. •	, `	•			•
	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 None	TR,CF,NP	1969
	Promote visibility	Promote visibility	Promote visibility		Frame coherence	None used		inline X		192m Fr N
COHERENCE	of a vertex to all edges at vertex	of a vertex to all edges at vertex	of a vertex to all edges at vertex		in depth No X coherence used			terence	De C	nline
SORTING	Back Edge Cull	Back Edge Cull	Back Edge Cull	Back Edge Cull	Intra-Cluster	2 Sar	Sert (Oct)			
	1) Edges separating back-facing planes 2) Dot product with	Back Edge Cull Description Back-facing planes	Back Edge Cull 1) Edges separating back-facing planes 2) Dot product with	1) Edges separating back-facing planes	1) Faces	2) und mu	1 Sort (Opt) 1) Faces, max 2 2) Comparison of	1) Sort	ygons, Y	Y Sort 1) Edges, Min Y 2) Comparison
mac,	normals & tanclosy			normals & topology	visibility 2) Dot p	3) . ogn	max points 3) n log m	3) 1 (et 4) The on 3		3) Bucket (4) Table of lists
erty (2)	3) Cull 4) List of edges, E _s 5) 1, E _t	4) List of edges, E _s	3) Cull 4) List of edges, E _s 5) 1, E _e	4) List of edges,E 5) 1, E	Exhaust or rc.	5) Ordered table	4) Ordered 5) 1, F _T	5) 1,	3) 2 bucket 4) Table of lists 5) 1, Fr	5) 1, E _r
Method	Contour Edge Cull				ter-Cluster		OU			2.5% (30.5 (c) C) + 47 C
Type (3)	1) Edges separating front & back faces			51 VO 10	Priority	Newell Specis I) Faces, p	mini-may	X Merge (1) Edges, X value (2) Comparison	X Sort 1) Edges, X value 2) Comparison	X Merge 1) Edges, X value 2) Comparison
(4) Result	2) Dot product with normals & topology	(Omitted)	(Omitted)	Cul	2) Dot product with separating planes 3) Prefix scan	b) S, S ra	n X and Y, sum of angles	3) Merge (ordred) 4) 2-way linked	3) 2 bucket 4) Table of lists	3) Merge (ordered) 4) Linked list
structure (5)	3) Cull 4) List, E _c 5) 1, E _e			Es Es	binary tree 4) wered tab	(4) ere able	g3) Radix 4 subdivi- sion with overlap 4) Stacks of	5) E _r , S _k	5) n. 5 ₂	5) E _r , 2S _£ (edges)
Number per frame, num-					5) 1 2	o, i pin races	unordered tables 5) L, F,/factor 1			,
ber of ob- jects	Initial Visibility 1) Ray to vertex	Initial bil	Visibility to vertex	Edge/Volume Test 1) Edges, visib	yl acc	Y Sort 1) Face segment	Depth Search 1) Surrounder faces	X Sort	X Priority Search I) Edges, X value	1) X Sort 1) Edges, X value
(merge) Number of	2) Benth	against a faces 2) Depth	against all faces 2) Betweenness, surroundedness	relative vol. (2) Line Pro manni	2) t y duct th fac ormal 3) cull	by Y range 2) Y intercept	2) 4-corner compare 3) Exhaustive	2) Comparison	(2) Comparison (3) Priority search	2) Comparison
Number of new entries per frame, length of	3) Exhau vi var 4) Quanti (1)	i) Exhau e search	Exhaustive sear Quantitative	a) vsi	1) Smaller ordered	3) Bucket 4) None 5) F. + split faces	4) Answer/failure 5) L _v , F _r /factor 2	5) Bubble 4) 2-way linked list	4) Active segment	4) 1-way linked list 5) N, 2Sg (edges)
list	vir its (va	isibility of vertex) fobjects, Fr	visibility 5) fobjects	5) edges,	5) 1, F _t	5) F + split faces Hf	25 25	5) n, Sa	5) n, n	TE TE
unber	ve l ersection	Edge Intersection	In set on		Y Cull	Y W	TV Same (One)	S 6 11		
	with all E. S	with all B	ith 1 E		 Faces by Y extent Mini-max on 	X intercept	TV Sort (Opt) Sort windows into scan-line order if	1) Segments, overlap with sample span	2 Search 1) Segments, depth 2) Linear equations	Z Search 1) Segments, depth 2) Linear equations
	with sweep triangle 3) Cull (unordered)	picture page, a th	cture plane, depth Cull (unordered)		X intercepts 3) Cull (unordered) 4) X intercepts of	2) Comparison 3) Ordered merge	needed	 Double comparisor Cull ordered list 	and comparison	and comparison
	4) Intersection list 5) E, Ec	Interse ist	4) Intersection list 5) E _s , E _s - 1		relevant segments 5) n, E,	4) Ordered list 5) S _r , S _v /2		4) Active list 5) n*S, * f (>1), S,	4) Visible segment 5) n*2S ₄ ,D _C	ordered active list 4) Visible segment
	Sort Along Edge 1) Intersections on	1) Intersections on	Sort Along Edge 1) Intersections on		X Sort I) Segments			Z Search	(Omitted if X priorities same as	5) n*2S ₁ , D _c
	edge, ordering 2) Comparison	edge, ordering	edge, ordering		2) Counters 3) Hardware			1) Segments, Z 2) Depth by logarithmic search	last time)	
	3) Bubble 4) Answer 5) E _s , X /E _s	4) Answer 5) E _s , X _v /E _s	3) 4) Answer 5) E _s . X _v /E _s	1	4) Segments at this X 5) nm, S,	İ		 Search (unordered Visible segment 	i)	
	Omit if well hidden)		(Omit if well hidden		İ			5) n*S _v *f(>1), D _c		
					Priority Search 1) Segments, priorit 2) Logic network	ļ			ļ	l
					2) Logic network 3) Logic network 4) Wisible segment					
			20		5) nm, S _L				.	

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

[Sutherland '74]

Rasterization Summary

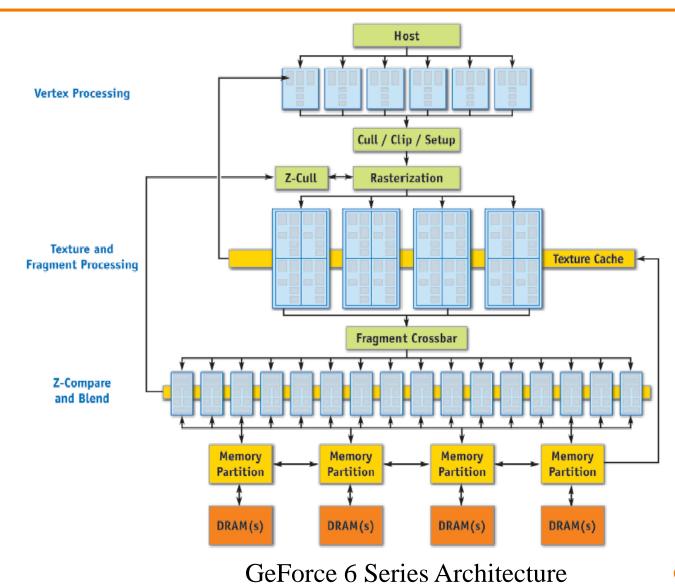


- Scan conversion
 - Sweep-line algorithm
- Shading algorithms
 - Flat, Gouraud
- Texture mapping
 - Mipmaps
- Visibiliity determination
 - Z-buffer

This is all in hardware

GPU Architecture



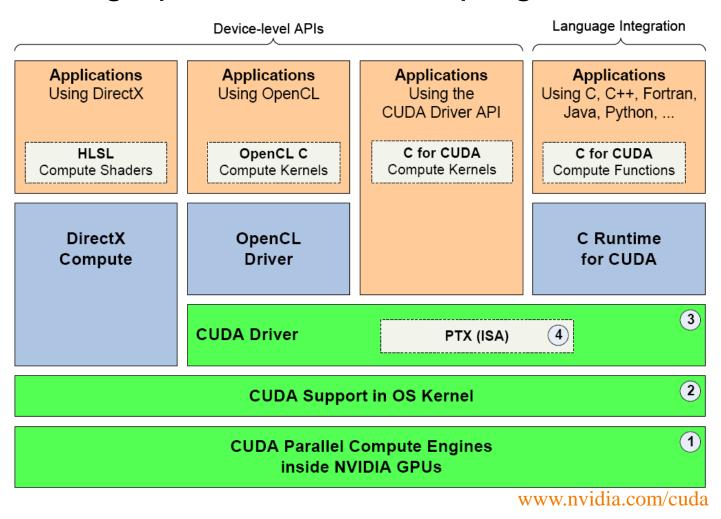


GPU Gems 2, NVIDIA

Actually ...



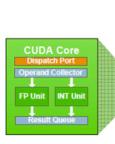
Modern graphics hardware is programmable

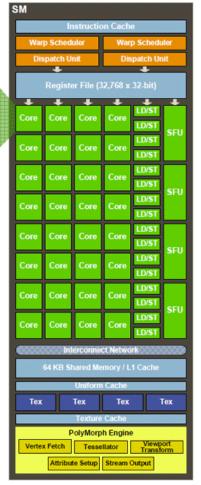


Trend ...



GPU is general-purpose parallel computer







www.nvidia.com/cuda