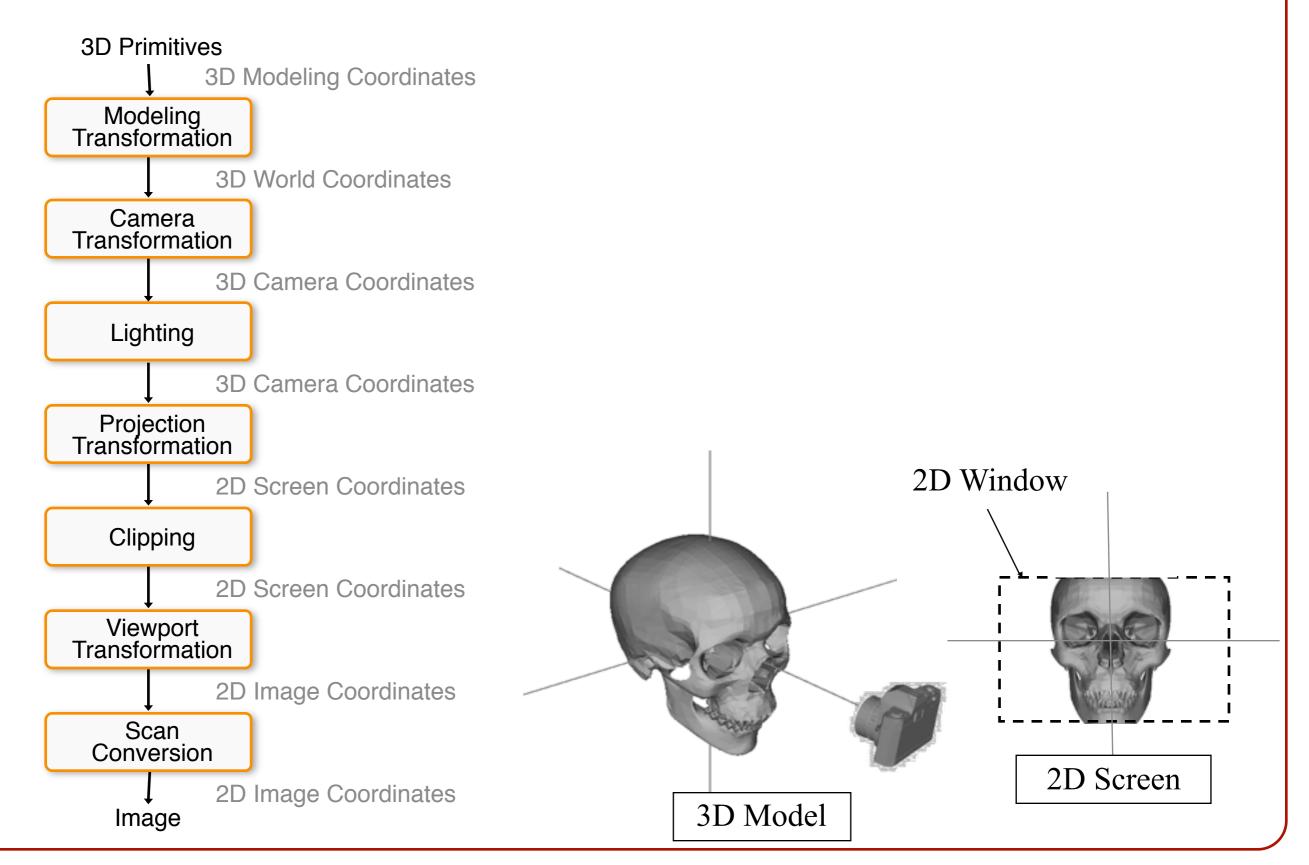
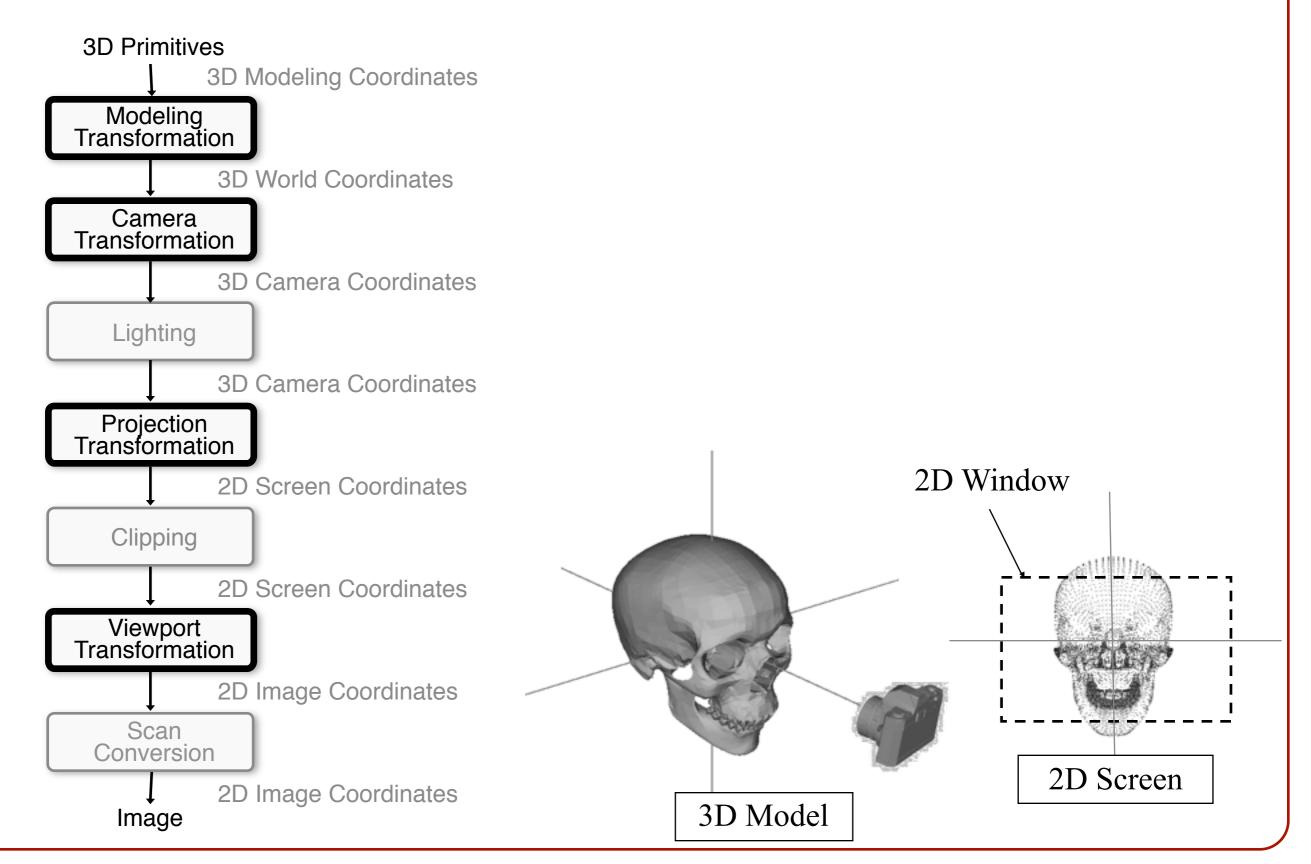
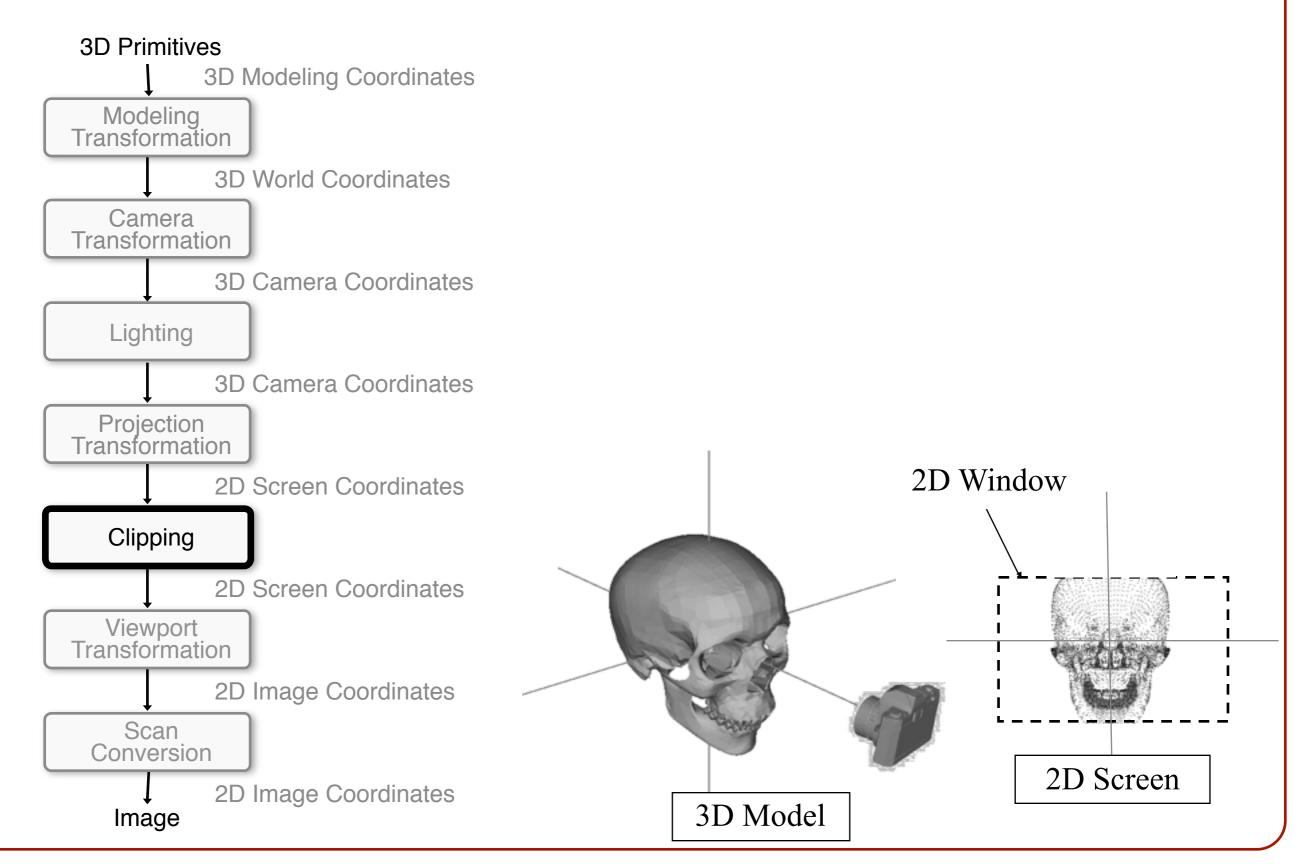
Shading and Visibility

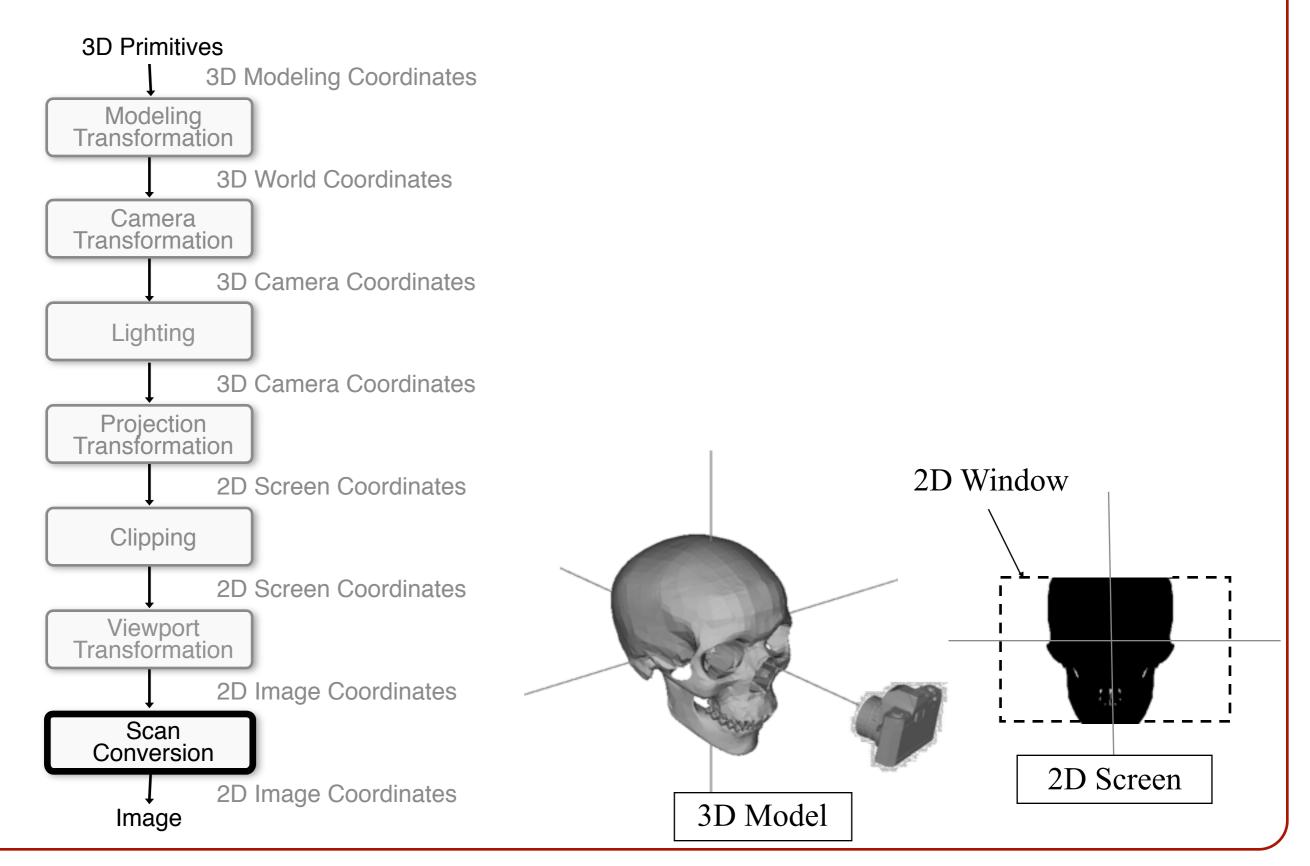
Connelly Barnes CS 4810: Graphics

Acknowledgment: slides by Jason Lawrence, Misha Kazhdan, Allison Klein, Tom Funkhouser, Adam Finkelstein and David Dobkin









Overview

- Scan conversion
 oFigure out which pixels to fill
- Shading
 ODetermine a color for each filled pixel

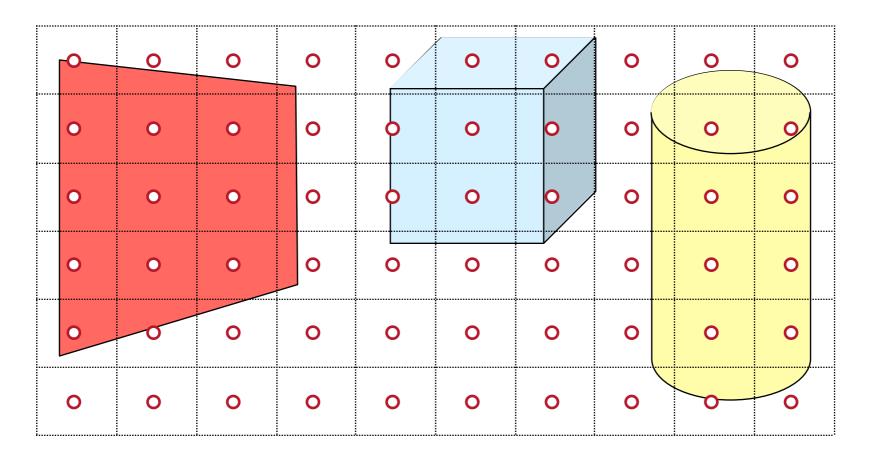
• Depth test

oDetermine when the color of a pixel comes from the frontmost primitive

Polygon Shading

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

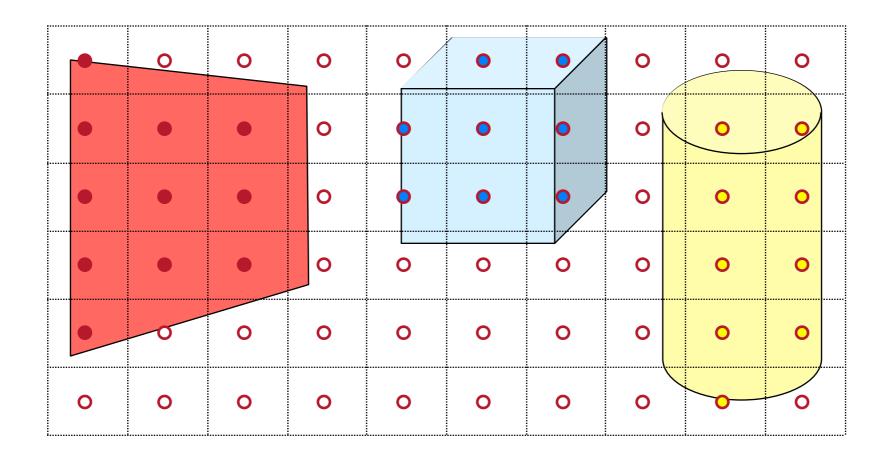
oWhen is this unnecessary?



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

Polygon Shading

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

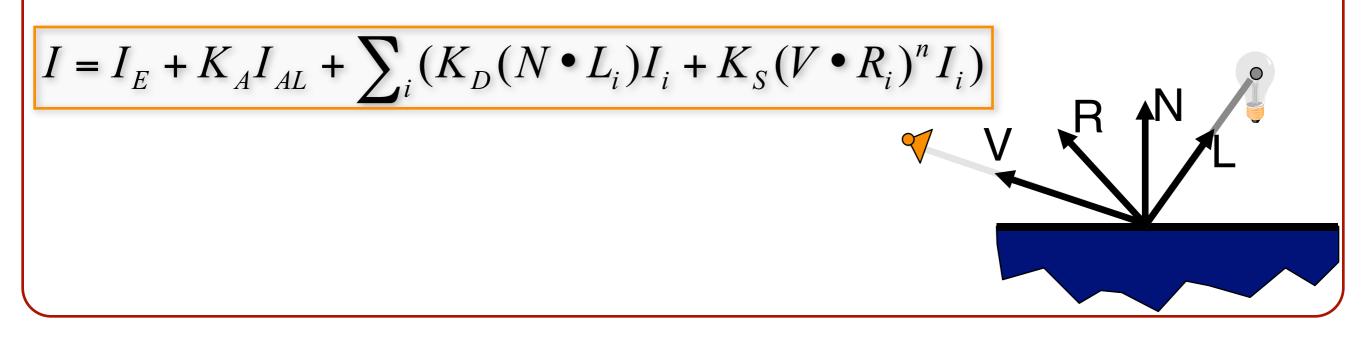


$$I = I_E + K_A I_{AL} + \sum_i (K_D (N \bullet L_i) I_i + K_S (V \bullet R_i)^n I_i)$$

Polygon Shading Algorithms

- Flat Shading
- Gouraud Shading
- Phong Shading

 Can take advantage of spatial coherence
 Make the lighting equation constant over the surface of each primitive



- Can take advantage of spatial coherence
 oMake the lighting equation constant over the surface of each primitive
- If the normal is constant over the primitive, and
- if the light is directional,

the diffuse component is the same for all points on the primitive

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

- Can take advantage of spatial coherence
 oMake the lighting equation constant over the surface of each primitive
- If the normal is constant over the primitive, and
- if the light is directional,

the diffuse component is the same for all points on the primitive

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

- Can take advantage of spatial coherence
 oMake the lighting equation constant
 over the surface of each primitive
- If the normal is constant over the primitive,
- if the light is directional, and
- if the direction to the viewer is constant over the primitive the specular component is the same for all points on the primitive

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

- Can take advantage of spatial coherence
 oMake the lighting equation constant
 over the surface of each primitive
- If the normal is constant over the primitive,
- if the light is directional, and
- if the direction to the viewer is constant over the primitive the specular component is the same for all points on the primitive

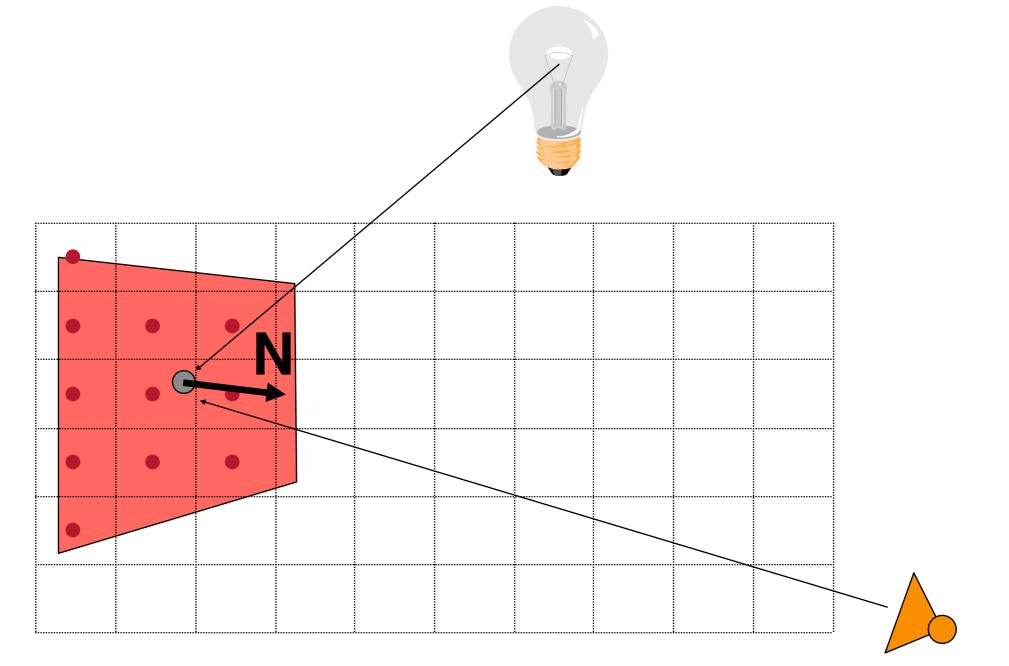
$$I = I_E + K_A I_{AL} + \sum_i (K_D (N \bullet L_i) I_i + K_S (V \bullet R_i)^n I_i)$$

 Illuminate as though all light sources are directional, the polygon is flat, and is viewed from infinitely far away

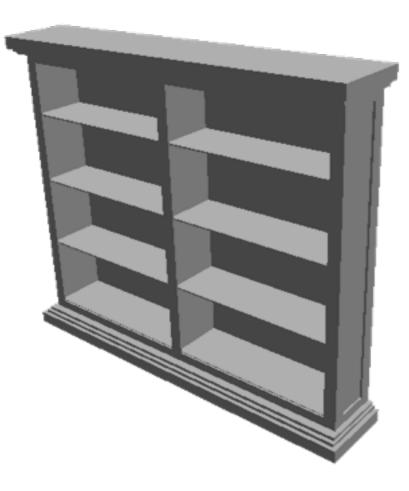
 oN·L_i constant over polygon
 oAttenuation function constant over polygon
 oV·R constant over surface

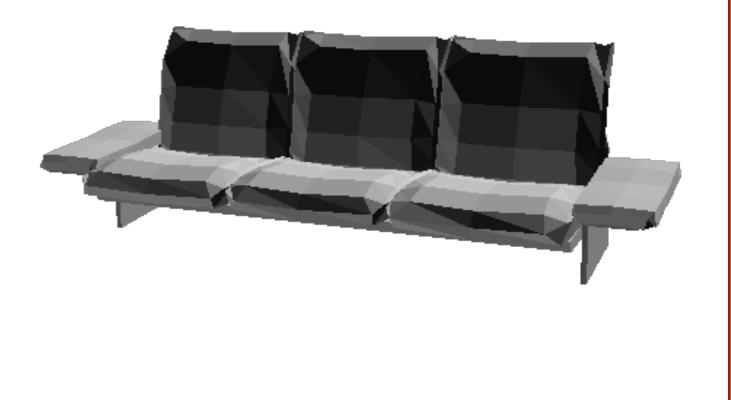
$$I = I_E + K_A I_{AL} + \sum_i (K_D (N \bullet L_i) I_i + K_S (V \bullet R_i))^n I_i$$

One lighting calculation per polygon
 oAssign all pixels inside each polygon the same color



Objects look like they are composed of polygons
 oOK for polyhedral objects
 oNot so good for smooth surfaces

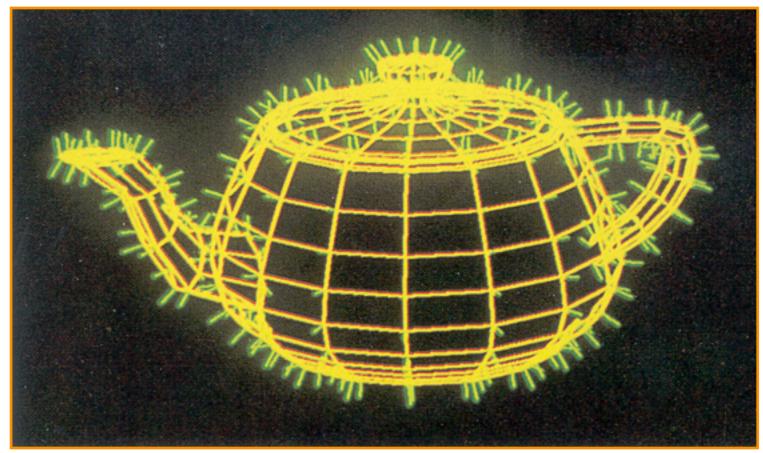




Polygon Shading Algorithms

- Flat Shading
- Gouraud Shading
- Phong Shading

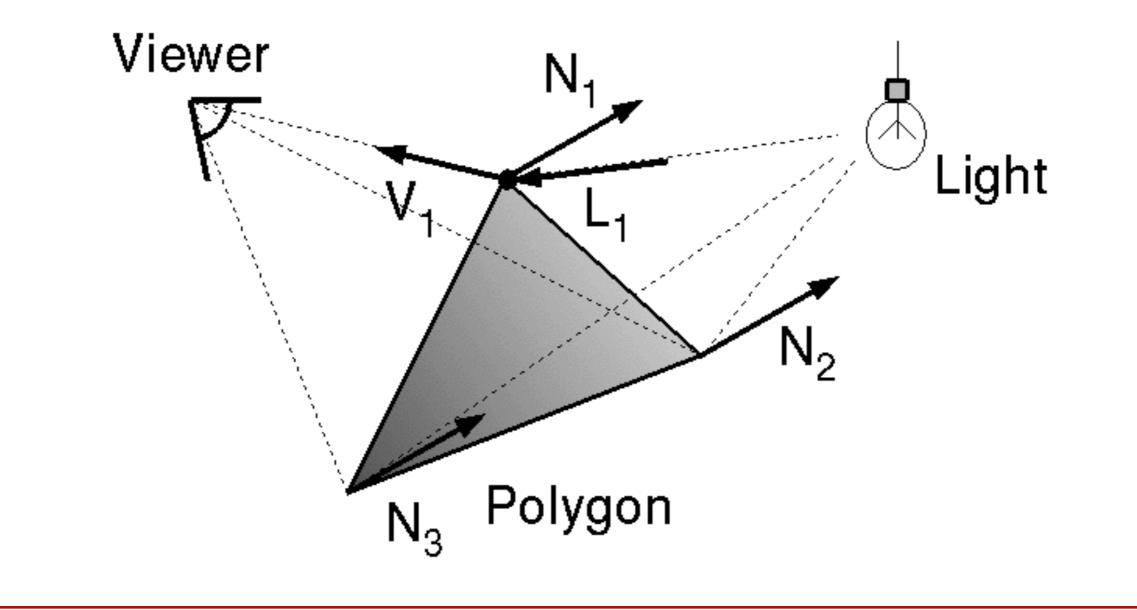
 What if smooth surface is represented by polygonal mesh with a normal at each vertex?



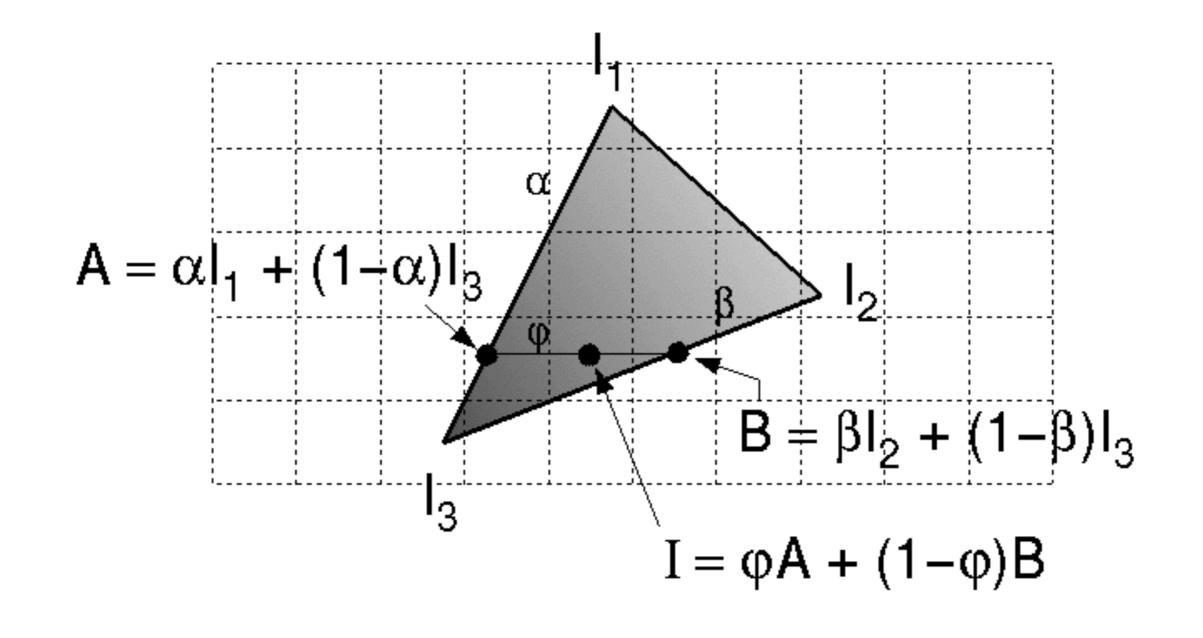
Watt Plate 7

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

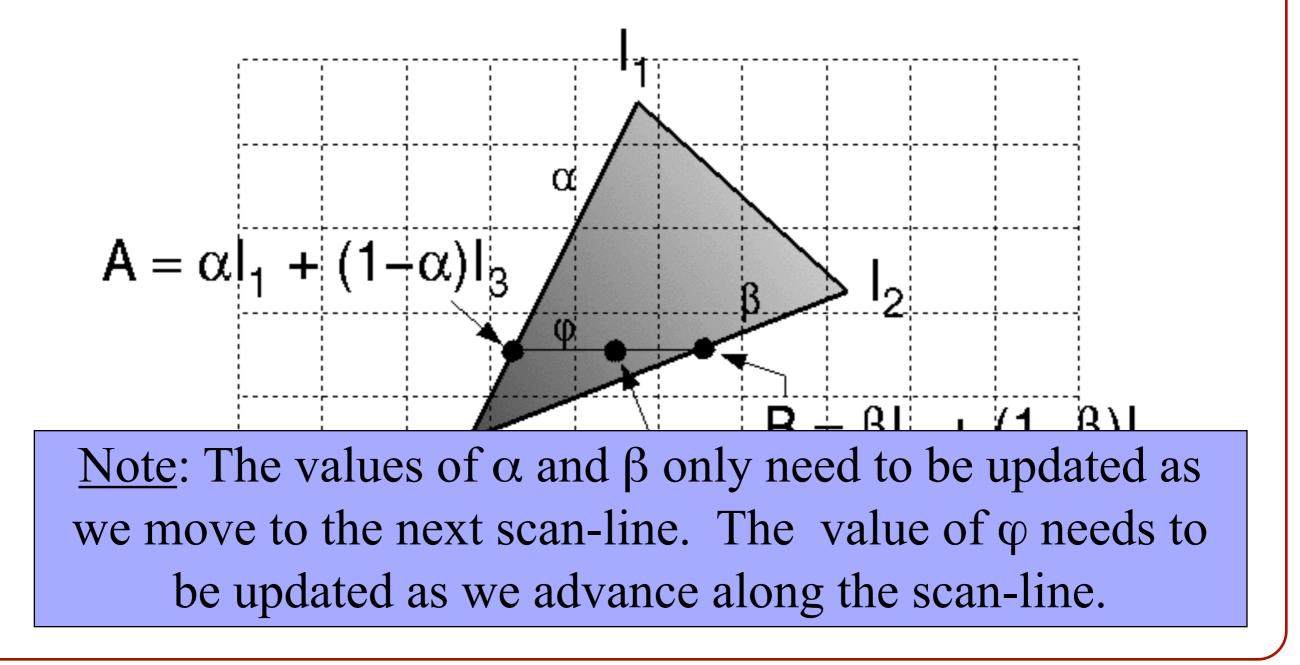
 One lighting calculation per vertex
 oAssign pixel colors inside polygon by interpolating colors computed at vertices



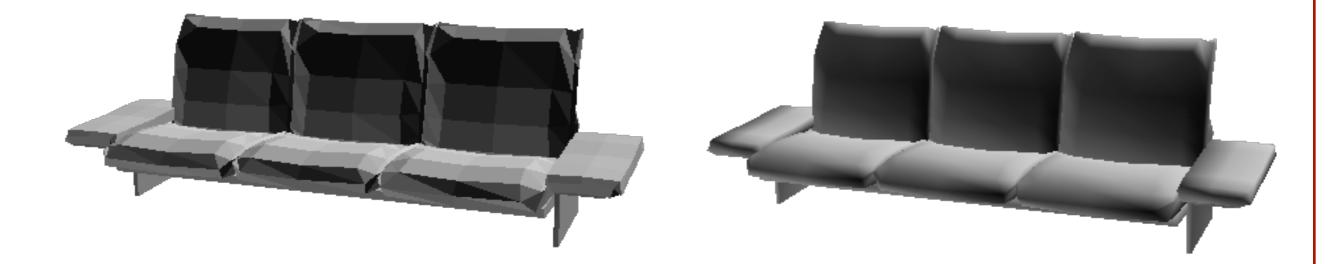
 Bilinearly interpolate colors at vertices down and across scan lines



 Bilinearly interpolate colors at vertices down and across scan lines



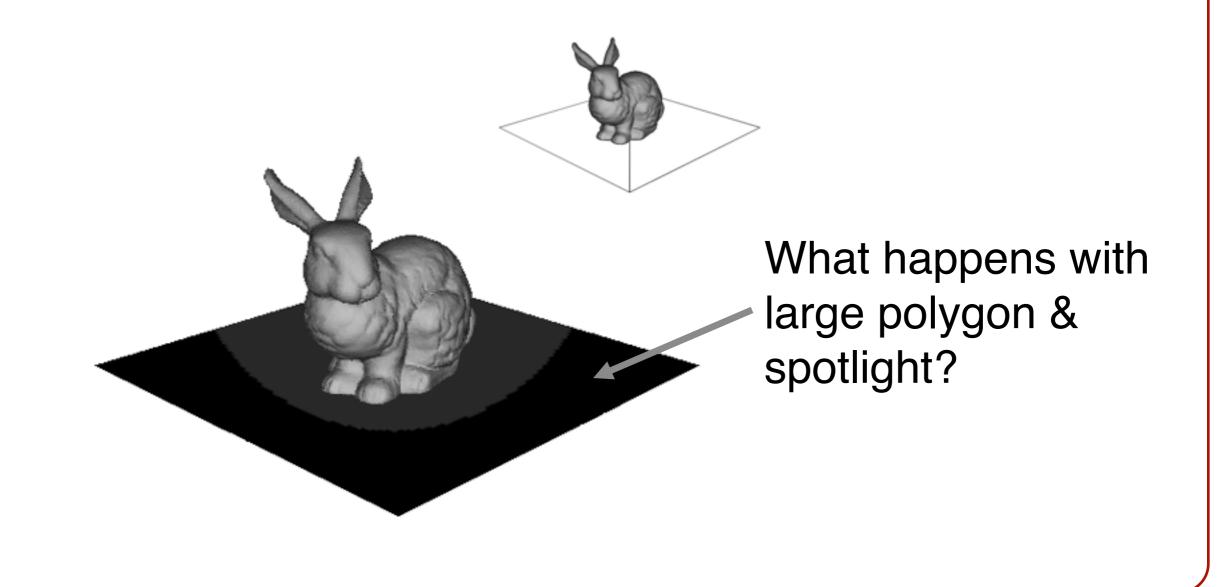
Produces smoothly shaded polygonal mesh
 oSmooth shading over adjacent polygons
 oNeed fine mesh to capture subtle lighting effects



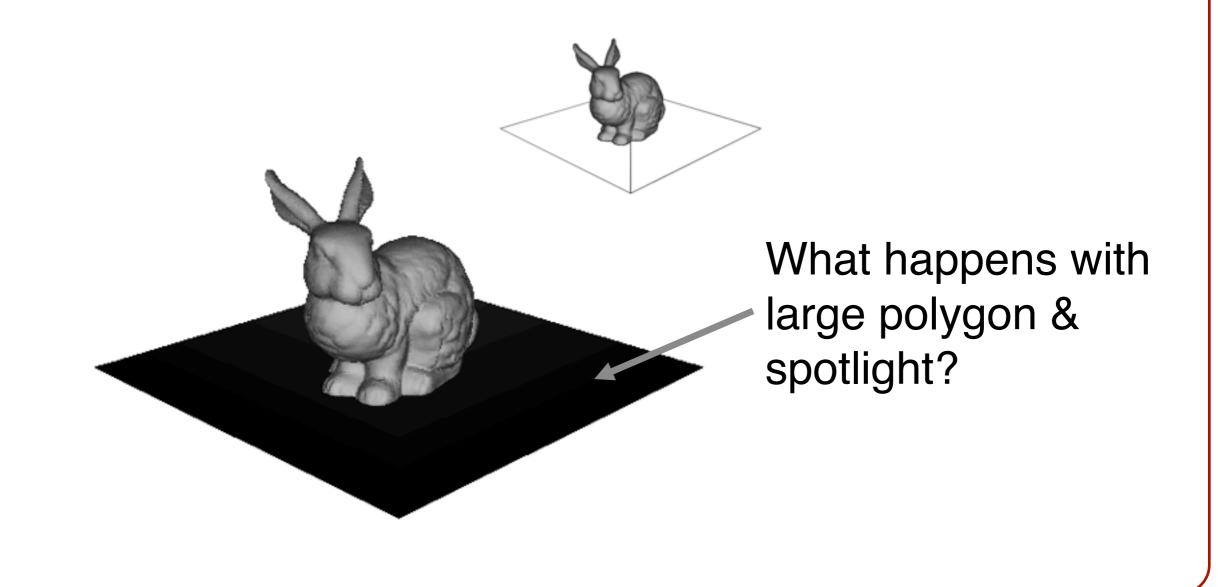
Flat Shading

Gouraud Shading

Produces smoothly shaded polygonal mesh
 oSmooth shading over adjacent polygons
 oNeed fine mesh to capture subtle lighting effects



Produces smoothly shaded polygonal mesh
 oSmooth shading over adjacent polygons
 oNeed fine mesh to capture subtle lighting effects

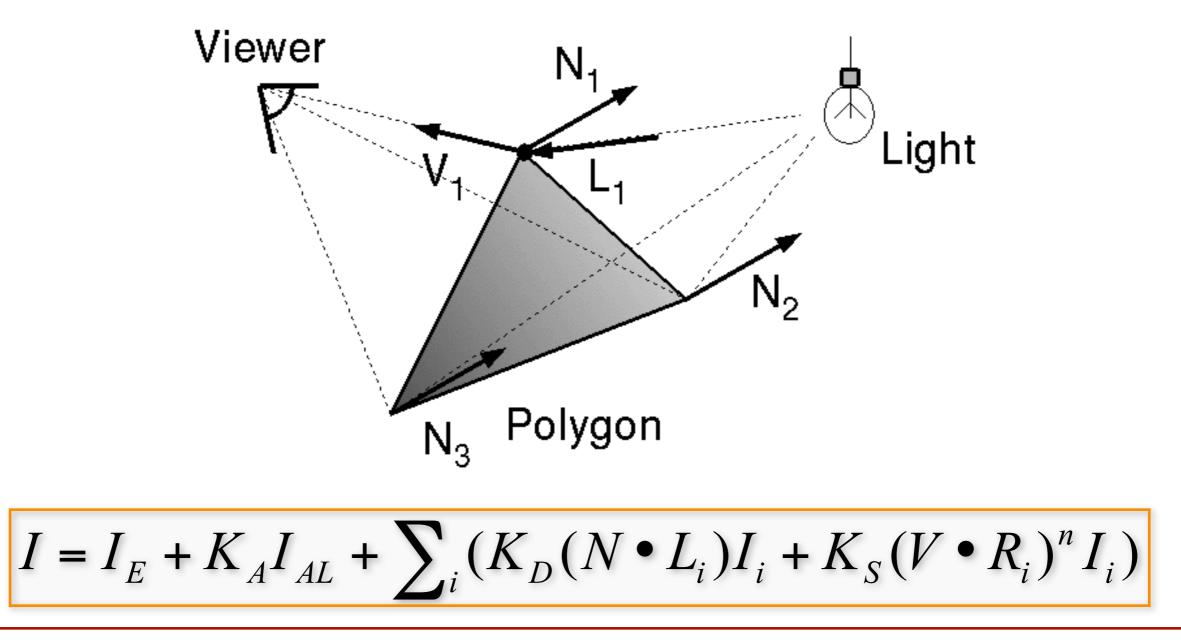


Polygon Shading Algorithms

- Flat Shading
- Gouraud Shading
- Phong Shading

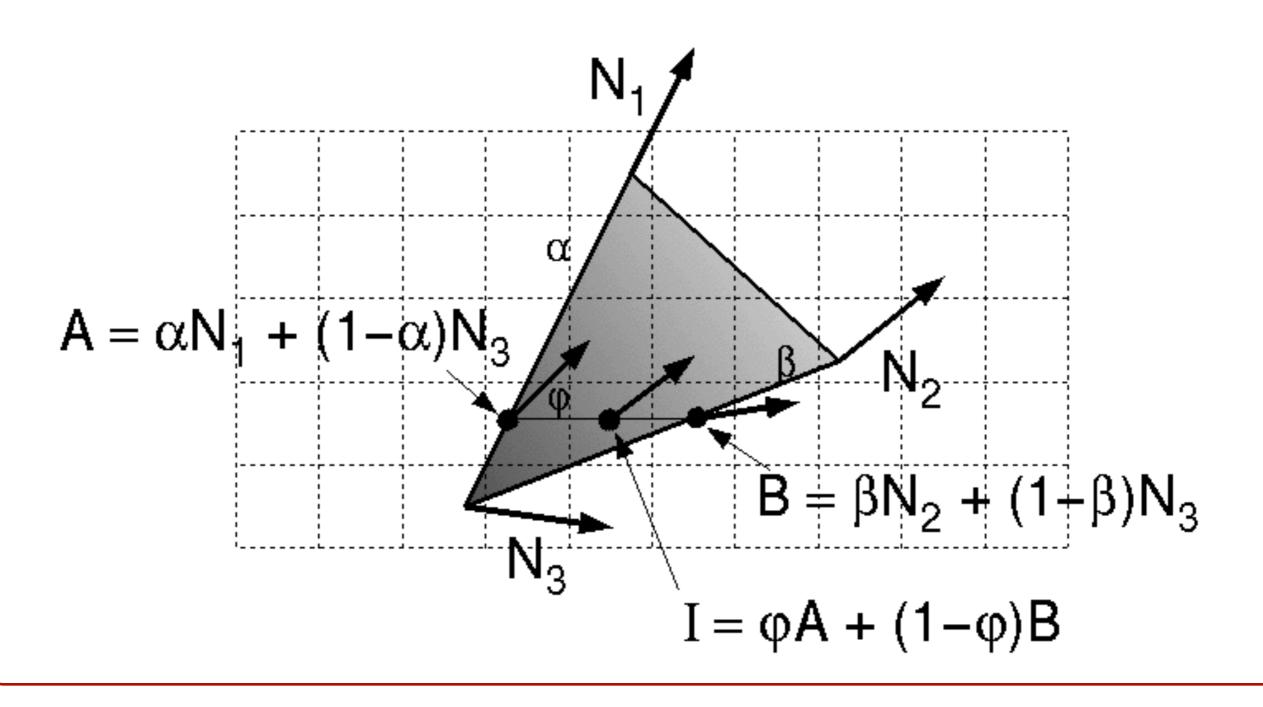
Phong Shading

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



Phong Shading

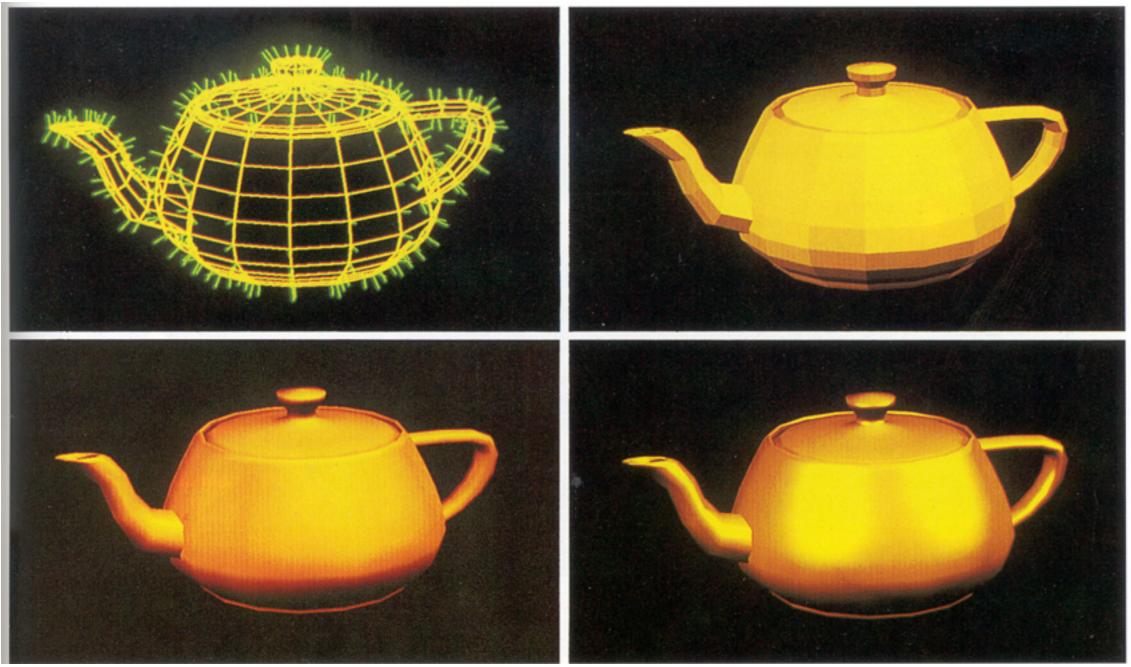
 Bilinearly interpolate surface normals at vertices down and across scan lines



Polygon Shading Algorithms

Wireframe

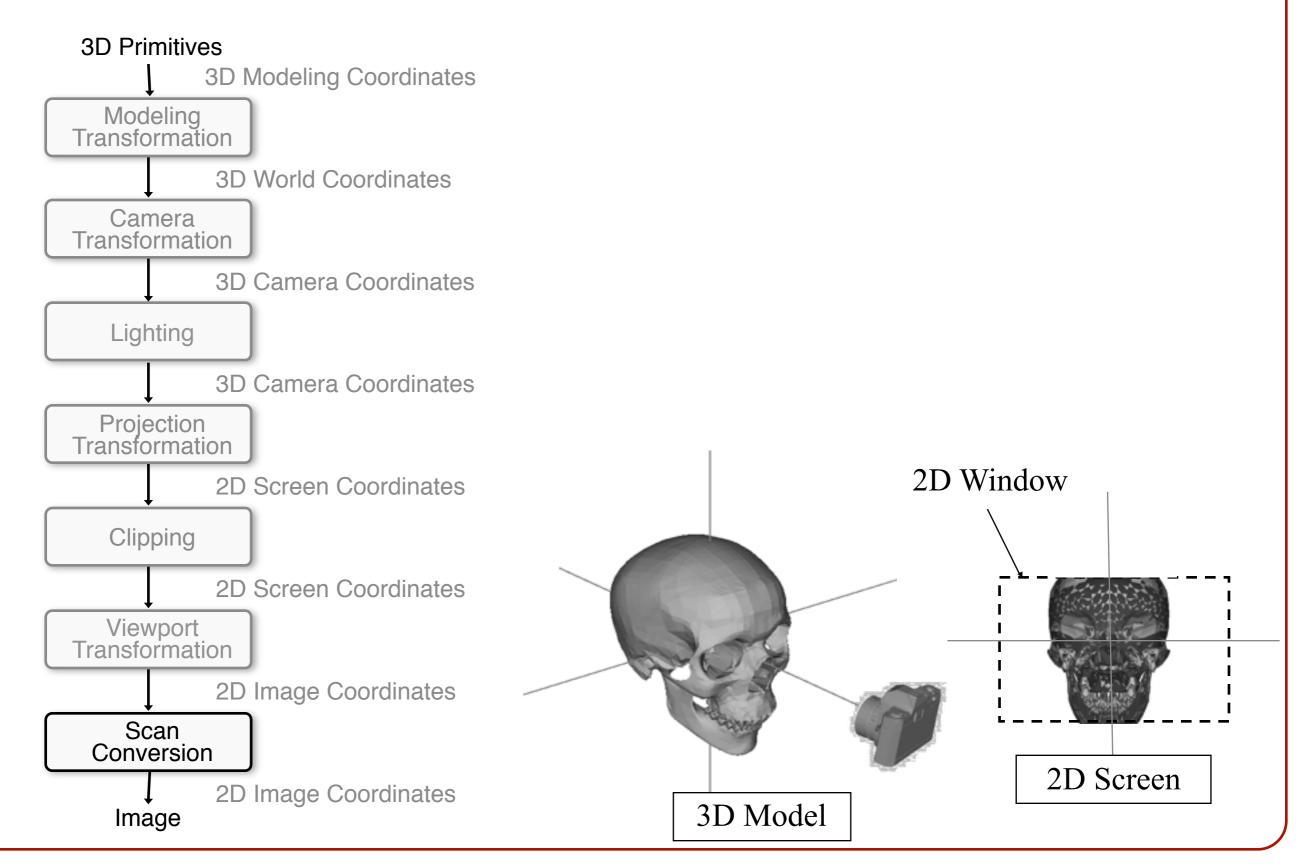
Flat



Gouraud

Phong

Watt Plate 7



Overview

- Scan conversion
 oFigure out which pixels to fill
- Shading
 oDetermine a color for each filled pixel

Depth test

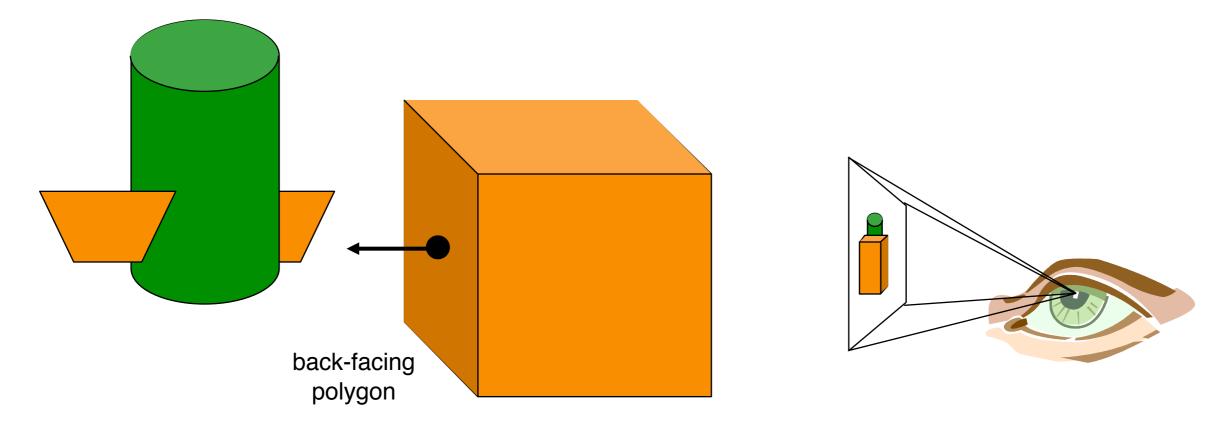
oDetermine when the color of a pixel comes from the frontmost primitive

Hidden Surface Removal

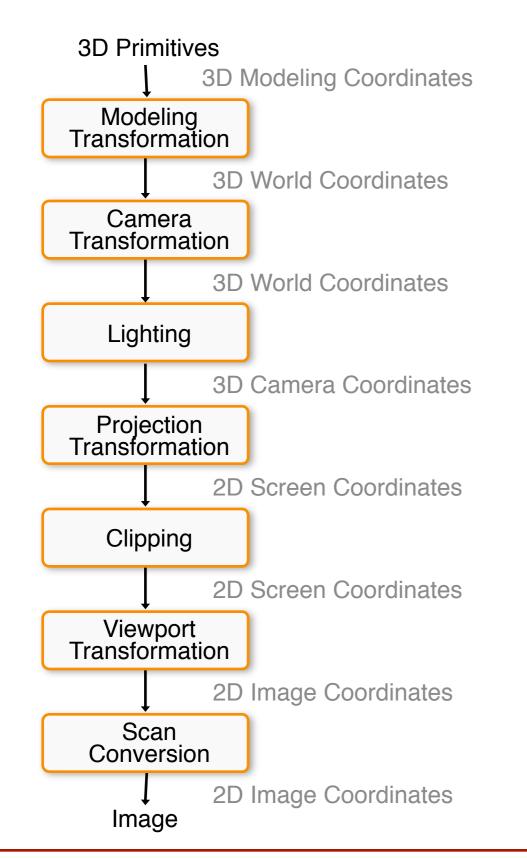
- Motivation
- Algorithms for HSR
 oBack-face detection
 oDepth sort
 oRay casting
 oZ-buffer

Motivation

- In general, we don't want to draw surfaces that are not visible to the viewer:
- Surfaces may be back-facing.
- Surfaces may intersect in 3D.
- Surfaces may intersect in the image plane.



3D Rendering Pipeline



Somewhere in here we have to decide which objects are visible, and which objects are hidden.

Overview

Motivation

Algorithms for HSR
 oBack-face detection
 oBSP-Trees
 oRay casting
 oZ-buffer

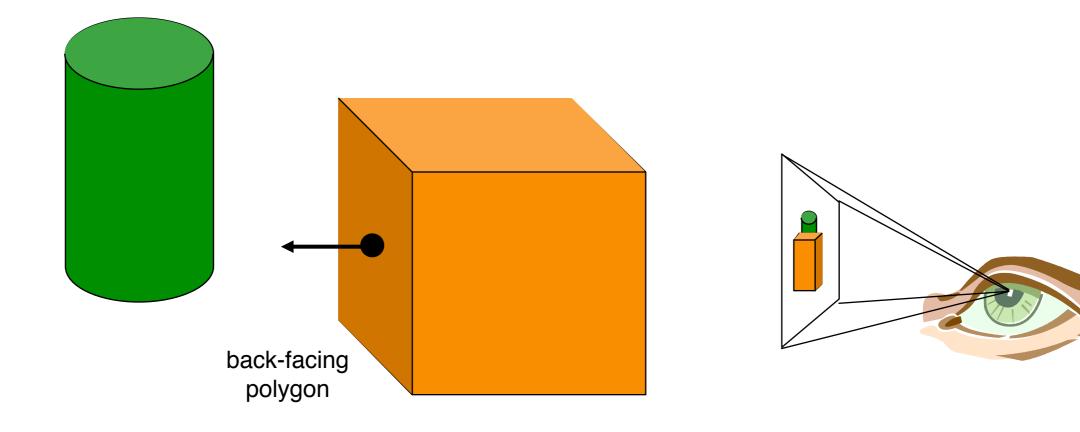
Visibility algorithms

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

				OPAQUI	-OBJECT ITHMS					ourjuo nigornanio
					$\overline{}$					
		COMPARIS	SON ALGORITHMS	OBJECT SPACE	(partly each)	INACE SPACE	DEPTH PRIORIT	Y ALGORITHMS		
		edges edges		edges volumes			ares samplas		point sampling	
		X			ALGORITHMS	×	1		Point sampling	
	/			\backslash	priority	dynamicall, computed priority		/		
	APPEL 1967	GALIMBERTI, et al	LOUTREL	ROBERTS	SCHUMACKER. at al	NEWILL, et al	NABNOCK	MATEINS	•	BOUKNIGHT
RESTRUCTIONS		TP, NP	1967 TP,KP	1963 TP, CC, CF, NP	CF, NP, LS (TP)	1972 Noze	1958 (TB) None	1970 None	ROWNEY, et al 1957 TR,CF,NP	1969
CONERENCE	of a vertex to all	Framote visibility of a vertex to all edges at vertex	Promote visibility of a vertex to all edges at vertex		Frame coherence in depth No I coherence used	None used	Area coheresce	Scanline X cohorenco	Scamline Depth Coherence	Scanline X Coherence
SCRTING What, what prop- erty (2) Hethed	back-facing planes 2) Dot product with normals & topology 3) Cull	 Dot product with hormals & topology Cull 		 Dot product with hormals & topology Cull 	visibility	2 Sort 1) Faces, max 2 2) Comparison of max points 3) a logm 4) Ordered table 5) 1, F _p	2 Sort (Opt) 1) Faces, max 1 2) Comparison of max points 5) n log m 4) Ordered table 5) 1, F _T	Y Sort 1) Edges, min Y 2) Comparison 3) Bucket 4) Table of Lists 5) 1, Er	Y Sort 1) Folygons, Y endpoints 2) Comparison 3) 2 bucket 4) Table of lists 5) 1, Fr	1) Edges, Min Y 2) Comparison 3) Bucket (4) Table of lists 5) 1, E _y
(3) Type (4) Result structure (5) Number per frame, num-	Contour Edge Cull 1) Edges separating front 6 back faces 2) Dot product with normals 6 topology 3) Cull 4) List, E _c 5) 1, E _g	(Omatted)	(Omstred)	Clipping Cull 1) Intersect edge with visible volume 2) Cull 4) I 5) I 5) S 1,	Inter-Cluster Priority 1) Clusters 2) Dot product with separating planes 3) Prefix scan binary tree 4) ordered table 5) 1, C _t	 Bubble, splittin 	sight angles (3) Radix 4 subdays- sight with overlap		X Sort 1) Edges, X value 2) Comparison 3) 2 bucket 4) Table of lists 5) n. S ₂	X Merge 1) Edges, X value 2) Comparison 3) Merge (ordered) 4) Linked list 5) E _p , 25 _L (edges)
ber of ob- jects (merge) Number of new entries per frame, length of list	against all faces 2) Depth, Surroundedness 3) Exhaustive search 4) Quantitative visibility of vertex	Initial Visibility 1) Bay to vertex against all faces 2) Depth. surroundedness 3) Exhaustive search 4) Quantitative visibility of vertex 5) fobjects, Pr	against all faces 2) Betweenness, surroundedness 3) Exhaustive search 4) Quantitative	 Edges, visibilit relative to volumes Linear Programming Mini-max sort Answer 	2) Bot product with face normal 3) Cull 4) Smaller ordered table	Y Sort 1) Face segment by Y range 2) Y intercept 3) Bucket 4) None 5) Fr split faces Hf	Depth Search 1) Surrounder faces 2) 4-corner compare 5) Exhaustive 4) Answer/failure 5) L _p , F _p /factor 2	X left 2) Comparison 5) Rubble	X Priority Search 1) Edges, X value 2) Comparison 3) Priority search 4) Active segment list 5) m, m	1 Sort 1) Edges, X value 2) Comparison 3) Bubble 4) 1-way linked list 5) N, 25g (edges)
(search) Number of searches, length of last	 Penetration With sweep triangle Cull (unordered) Intersection list 	2) Intersett in	Edge Intersection 1) Intersect one 5, with all E, 2) Intersect in picture plane, depth 3) Cull (unordered) 4) Intersection list 5) E, E, 1		Y Coll 1) Faces by Y extent 2) Minimax on 1 intercepts 3) Cull (unordered) 4) X intercepts of relevant segments 5) n, E ₅	X Merge 1) Segments, X intercept 2) Comparison 3) Ordered merge 4) Ordered list 5) 5, 5,72	needed	a) Double comparison b) Cull ordered list	 Linear equations and comparison Search (unordered 4) Visible segment n*25g.0c 	I Search 3) Segments, depth 2) Linear equations and comparison (3) Search of un- ordered active list 4) Visible segment 5) m225, 0,
	Sort Along Edge 1) Intersections on edge, ordering 2) Comparison 3) Bubble 4) Answer 5) E _g , X _y /E _g (Omat of woll hadden)	Sort Along Edge 1) Tetersections on edge, ordering 2) 3) 4) Answer 5) E _s , X _q /E _s (must be done)	Sort Along Boge 1) Intersections on odge, ordering 2) 3) 4) Answer 5) E ₅ , X ₂ /E ₅ (Cmit if well hidden		X Sort 1) Segments 2) Counters 3) Hardware 4) Segments at this X 5) mm, Sg			<pre>1 Search 1) Segments, 1 2) Depth by logarithmic search 3) Search (unordered 4) Timble segment 5) n*5v*f(>1), D_c</pre>	(Omitted of X priorities same as last time)	
					Priority Search 1) Segments, priorit 2) Logic network 3) Logic network 4) Wisible segment 5) nm, Sg				r a	
Figure 29. Characterization of ten opaque-object algorithms & Comparison of the algorithms. [Sutherland '74]										

Back-face detection

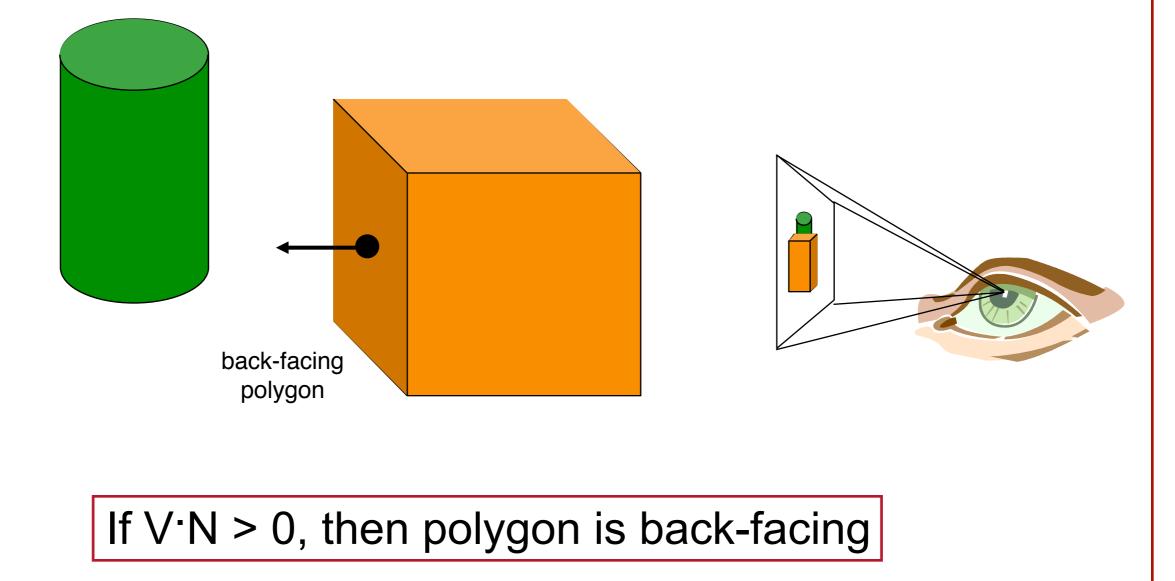
Q: How do we test for back-facing polygons?



Back-face detection

Q: How do we test for back-facing polygons?

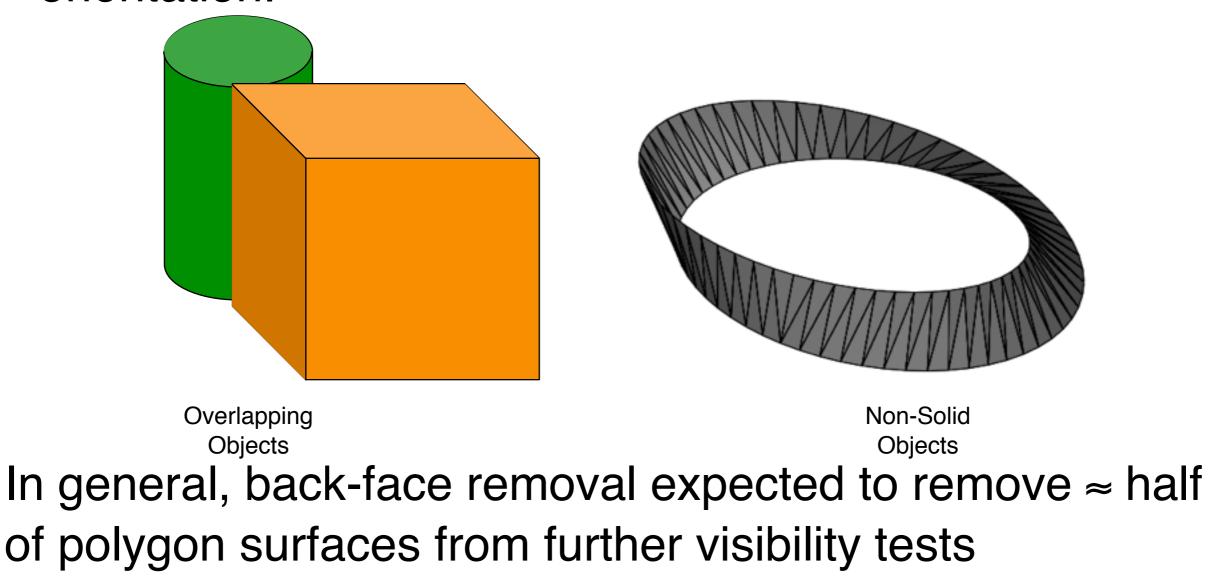
A: Dot product of the normal and view directions.



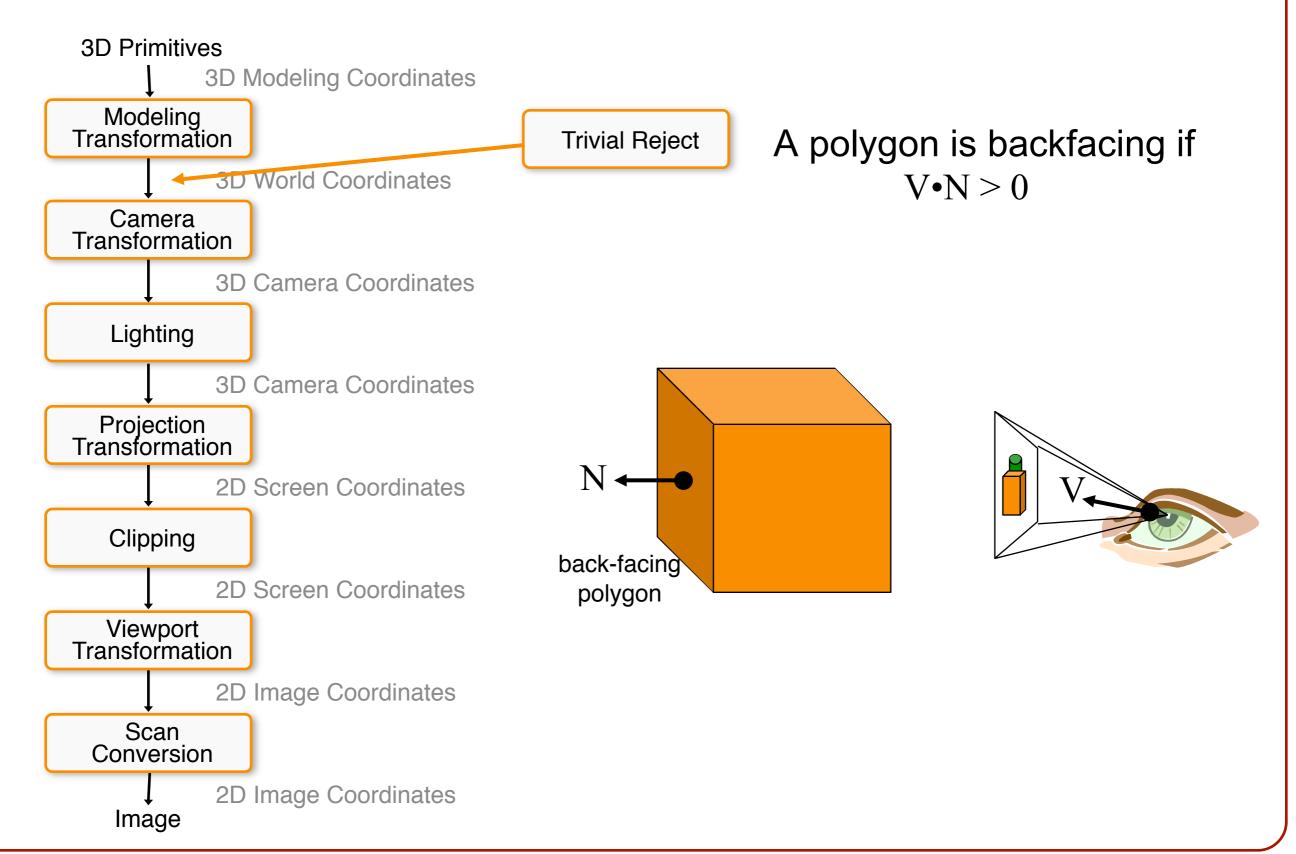
Back-face detection

This method breaks down for:

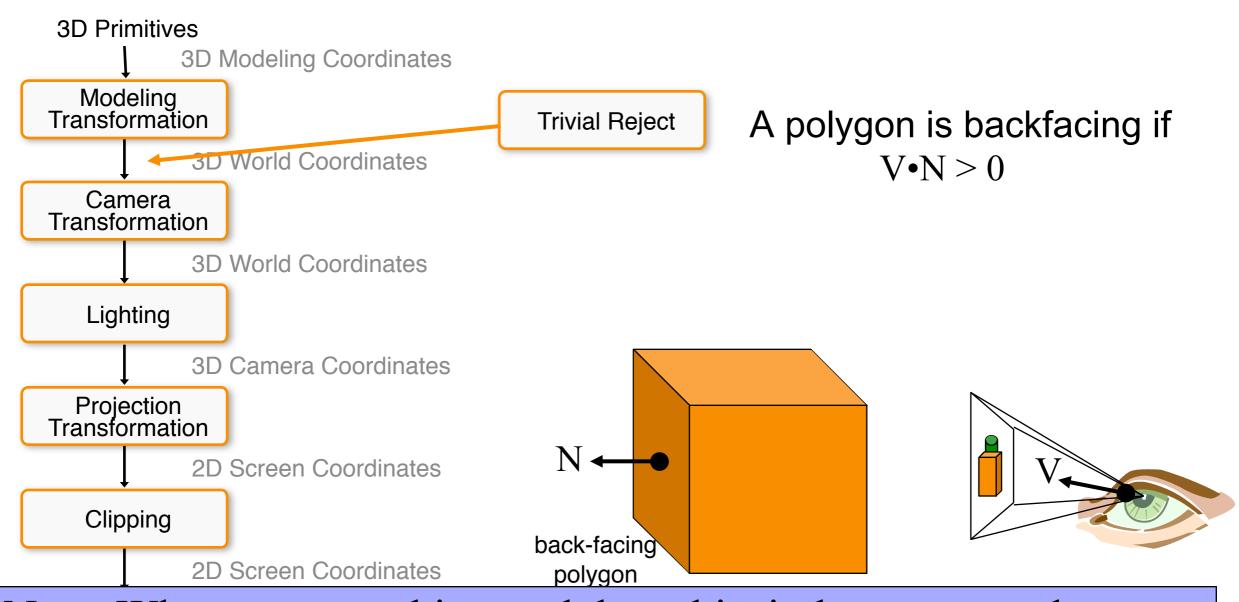
- Overlapping primitives
- Non-solid models and/or models without a well defined orientation.



3D Rendering Pipeline



3D Rendering Pipeline



<u>Note</u>: When your graphics card does this, it does <u>not</u> use the normals you provide at the vertices. Instead it uses the cross-product of the triangle vertices, so make sure that the ordering of the vertices is consistent (e.g. CCW)

Ideal Solution

Painter's Algorithm:

 Sort primitives front to back and draw the back ones first, over-writing pixel values with information from the front primitives as they are processed.

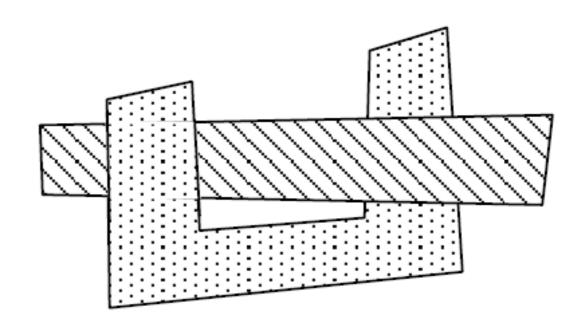
Ideal Solution

Painter's Algorithm:

 Sort primitives front to back and draw the back ones first, over-writing pixel values with information from the front primitives as they are processed.

Problem:

• You can't always sort the primitives.



Ideal Solution

Painter's Algorithm:

 Sort primitives front to back and draw the back ones first, over-writing pixel values with information from the front primitives as they are processed.

Problem:

• You can't always sort the primitives.

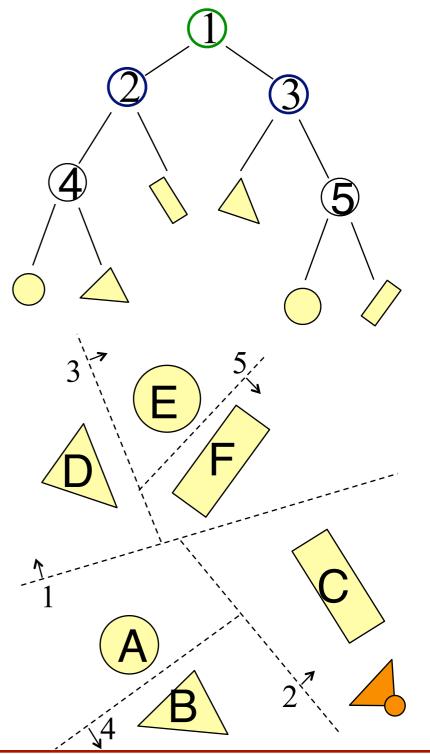
However, in some cases you can sort the primitives – e.g. if all the vertices of one primitive are in front of all the vertices of the second.

 BSP-Trees recursively partition space by planes
 oGiven two primitives on either side of a plane, the one on the opposite side from the camera will always be further away.

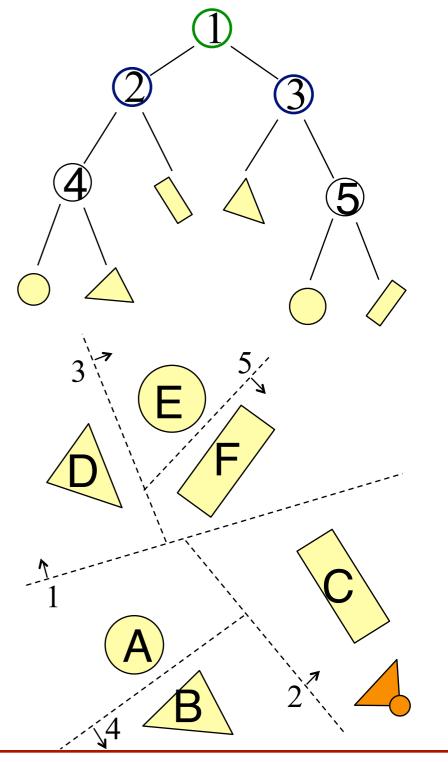
 $(\mathbf{5})$

oDraw the further side first, and then draw the closer one

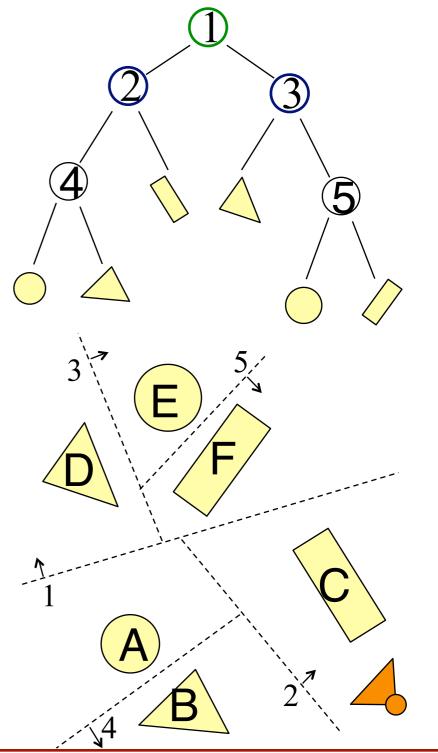
- Draw further half first, then the closer one.
 - Draw right side of 1
 - Draw left side of 1



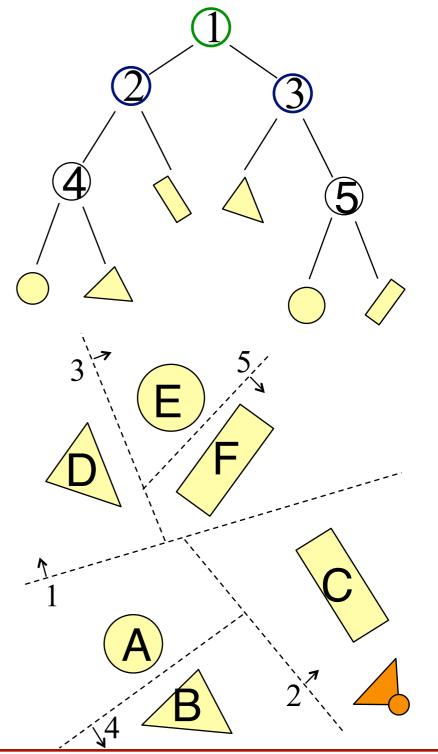
- Draw further half first, then the closer one.
 - Draw right side of 1
 - Draw left side of 3
 - Draw right side of 3
 - Draw left side of 1



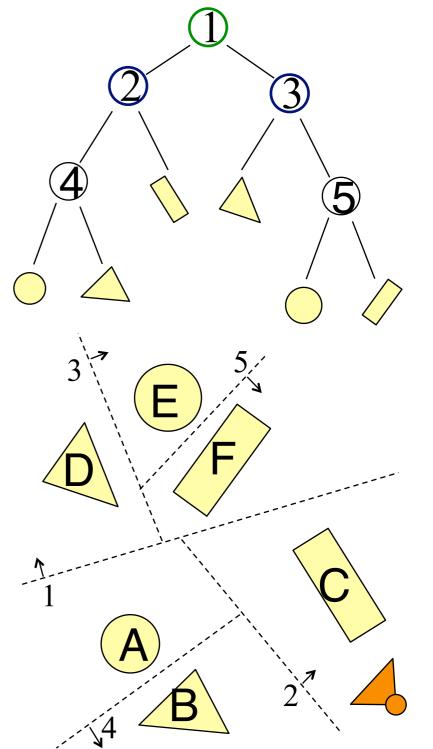
- Draw further half first, then the closer one.
 - Draw right side of 1
 - Draw left side of 3
 - Draw D
 - Draw right side of 3
 - Draw left side of **1**



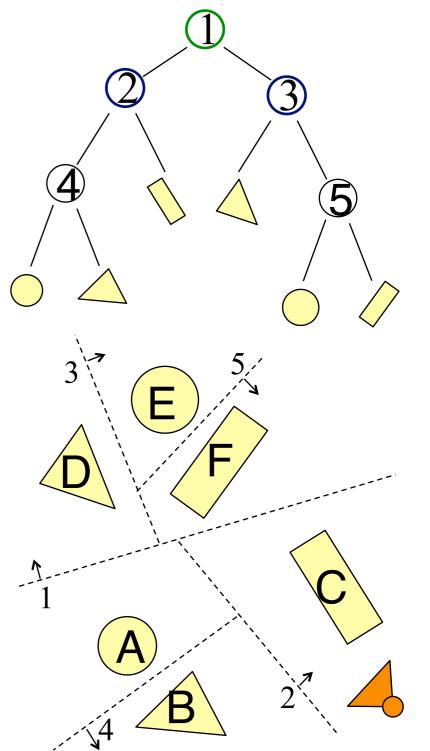
- Draw further half first, then the closer one.
 - Draw right side of 1
 - Draw left side of 3
 - Draw **D**
 - Draw right side of 3
 - Draw left side of 5
 - Draw right side of 5
 - Draw left side of 1



- Draw further half first, then the closer one.
 - Draw right side of 1
 - Draw left side of 3
 - Draw D
 - Draw right side of 3
 - Draw left side of 5
 - Draw E
 - Draw right side of 5
 - Draw left side of 1



- Draw further half first, then the closer one.
 - Draw right side of 1
 - Draw left side of 3
 - Draw **D**
 - Draw right side of 3
 - Draw left side of 5
 - Draw E
 - Draw right side of 5
 - Draw F
 - Draw left side of 1

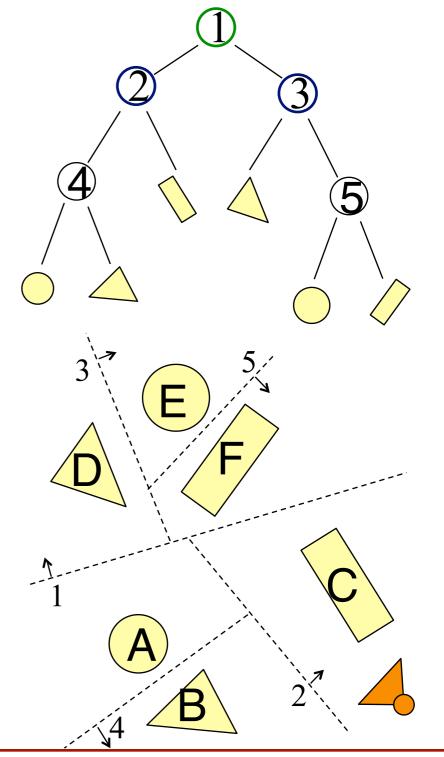


• Draw further half first, then the closer one.

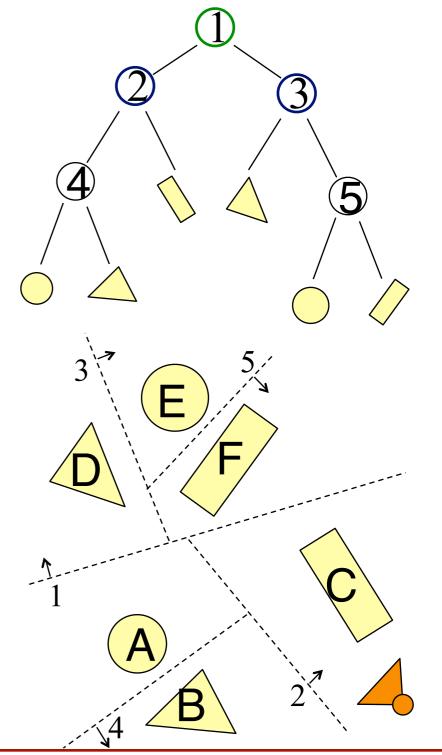
5

- Draw right side of 1
 - Draw left side of 3
 - Draw **D**
 - Draw right side of 3
 - Draw left side of 5
 - Draw E
 - Draw right side of 5
 - Draw F
- Draw left side of 1
 - Draw left side of 2
 - Draw right side of 2

- Draw further half first, then the closer one.
 - Draw right side of 1
 - Draw left side of 3
 - Draw **D**
 - Draw right side of 3
 - Draw left side of 5
 - Draw E
 - Draw right side of 5
 - Draw F
 - Draw left side of 1
 - Draw left side of 2
 - Draw left side of 4
 - Draw right side of 4
 - Draw right side of 2



- Draw further half first, then the closer one.
 - Draw right side of 1
 - Draw left side of 3
 - Draw **D**
 - Draw right side of 3
 - Draw left side of 5
 - Draw E
 - Draw right side of 5
 - Draw F
 - Draw left side of 1
 - Draw left side of 2
 - Draw left side of 4
 - Draw A
 - Draw right side of 4
 - Draw right side of 2

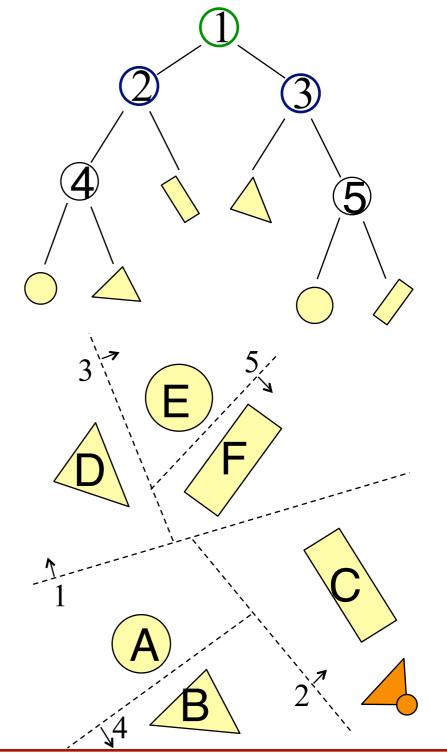


Draw further half first, then the closer one.

5

- Draw right side of 1
 - Draw left side of 3
 - Draw **D**
 - Draw right side of 3
 - Draw left side of 5
 - Draw E
 - Draw right side of 5
 - Draw F
- Draw left side of 1
 - Draw left side of 2
 - Draw left side of 4
 - Draw A
 - Draw right side of 4
 - Draw **B**
 - Draw right side of 2

- Draw further half first, then the closer one.
 - Draw right side of 1
 - Draw left side of 3
 - Draw **D**
 - Draw right side of 3
 - Draw left side of 5
 - Draw E
 - Draw right side of 5
 - Draw F
 - Draw left side of 1
 - Draw left side of 2
 - Draw left side of 4
 - Draw A
 - Draw right side of 4
 - Draw **B**
 - Draw right side of 2
 - Draw C

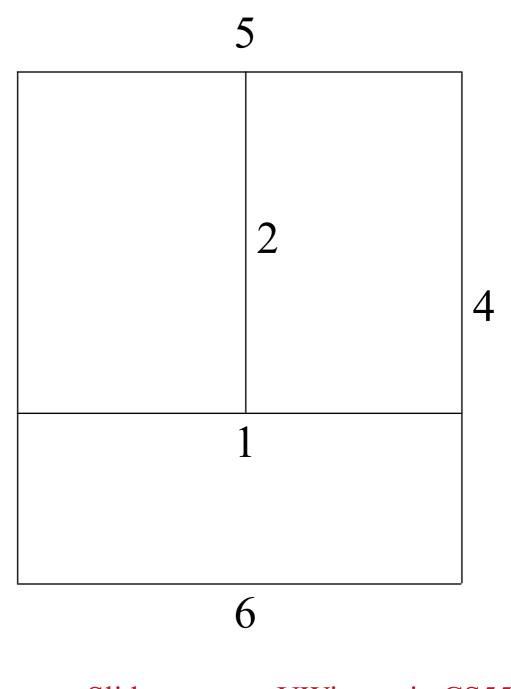


Building BSP-Trees

- Choose polygon (arbitrary)
- Split its cell using plane on which polygon lies
 oMay have to chop polygons in two (Clipping!)
- Continue until each cell contains only one polygon fragment
- Splitting planes could be chosen in other ways, but there is no efficient optimal algorithm for building BSP trees
 Optimal means minimum number of polygon fragments in a balanced tree

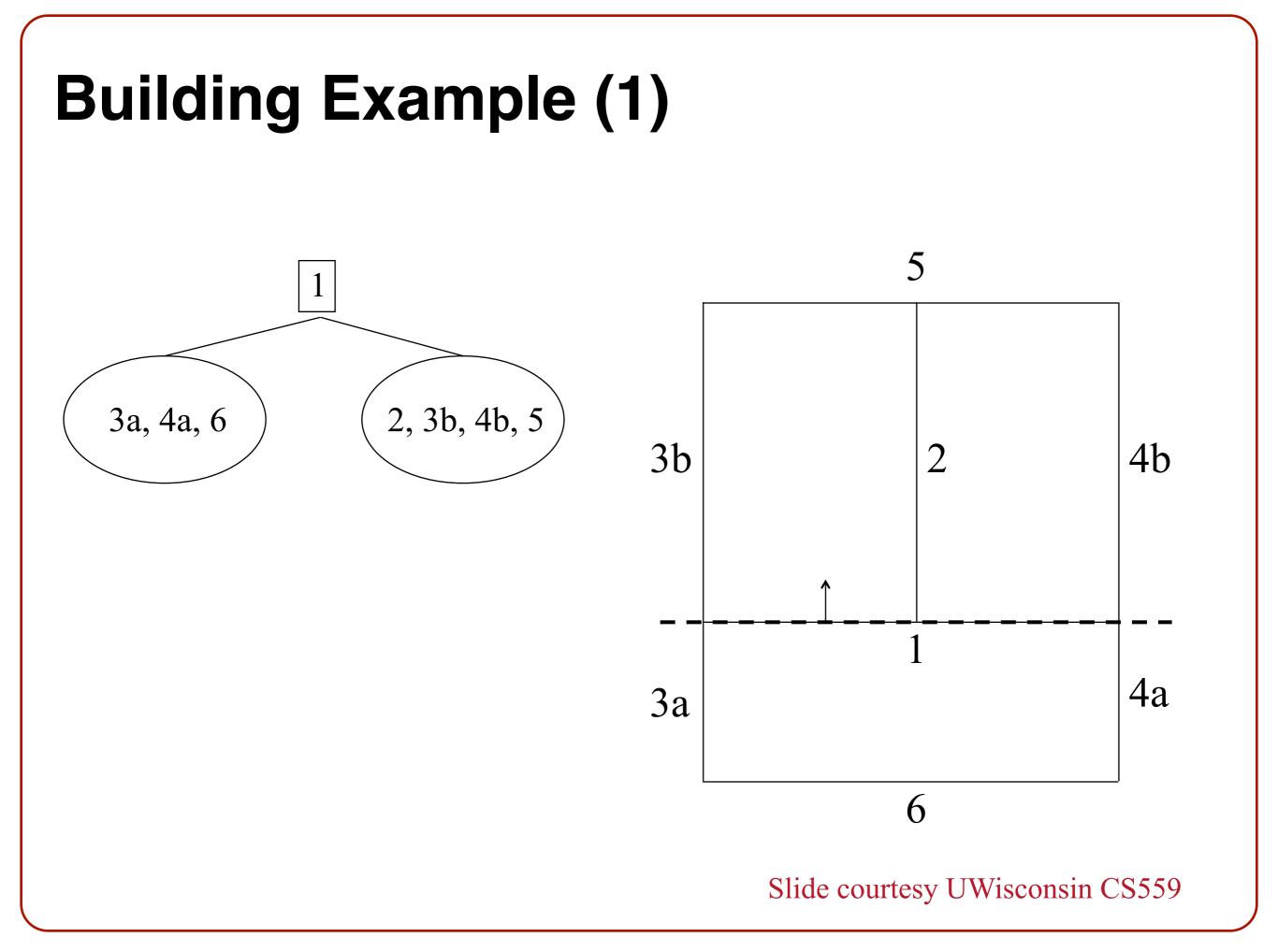
Building Example

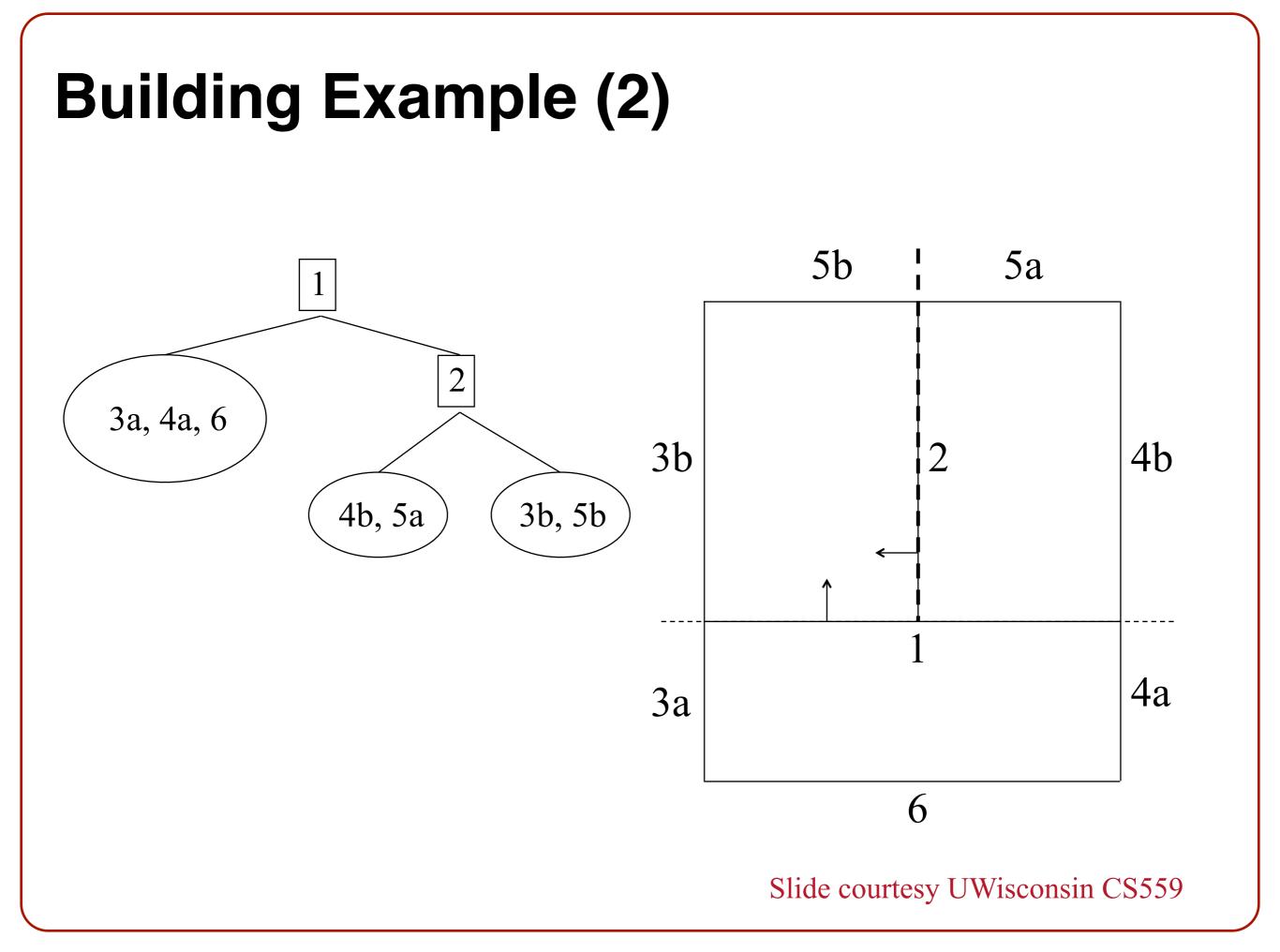
• We will build a BSP tree, in 2D, for a 3 room building

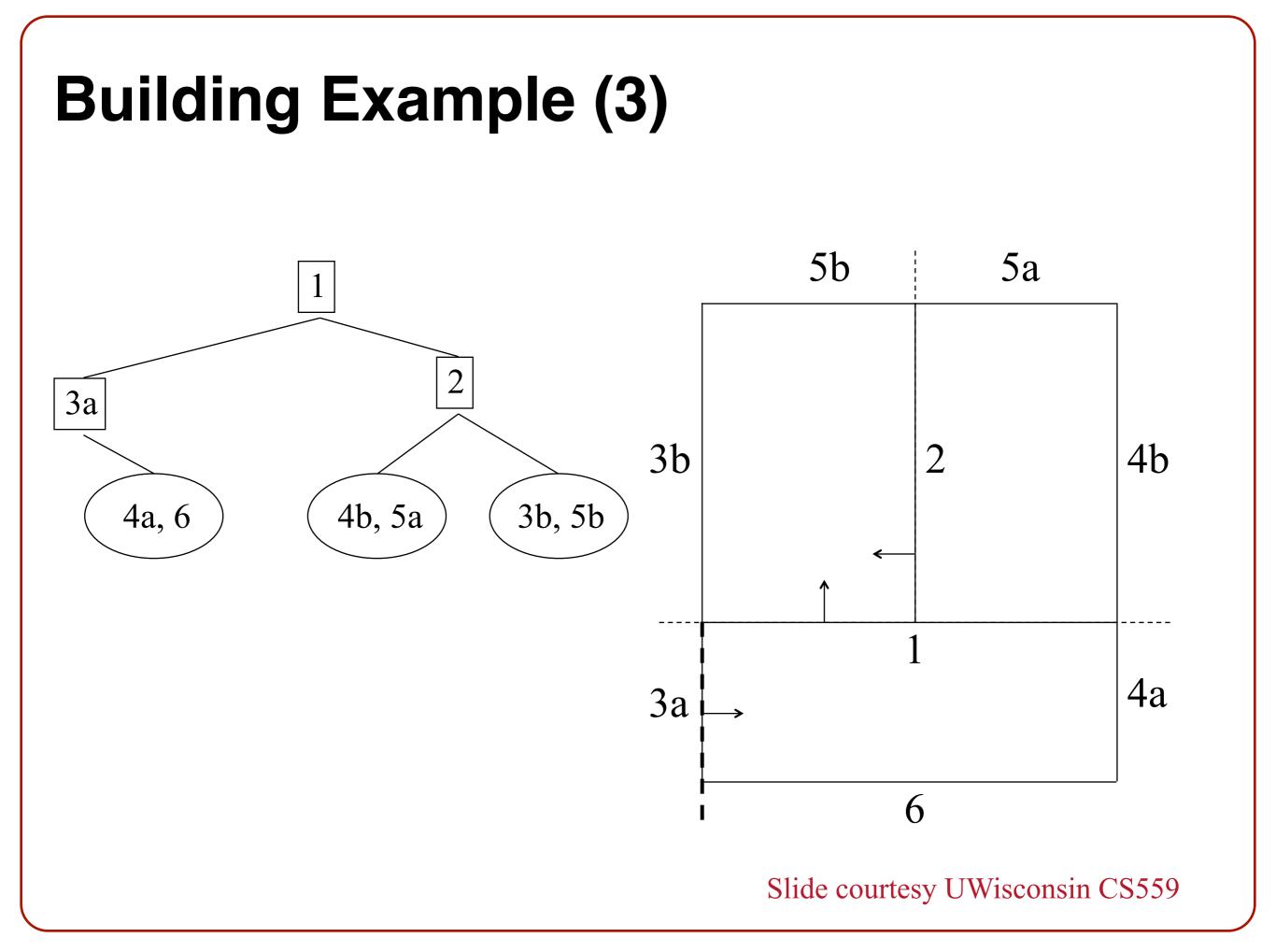


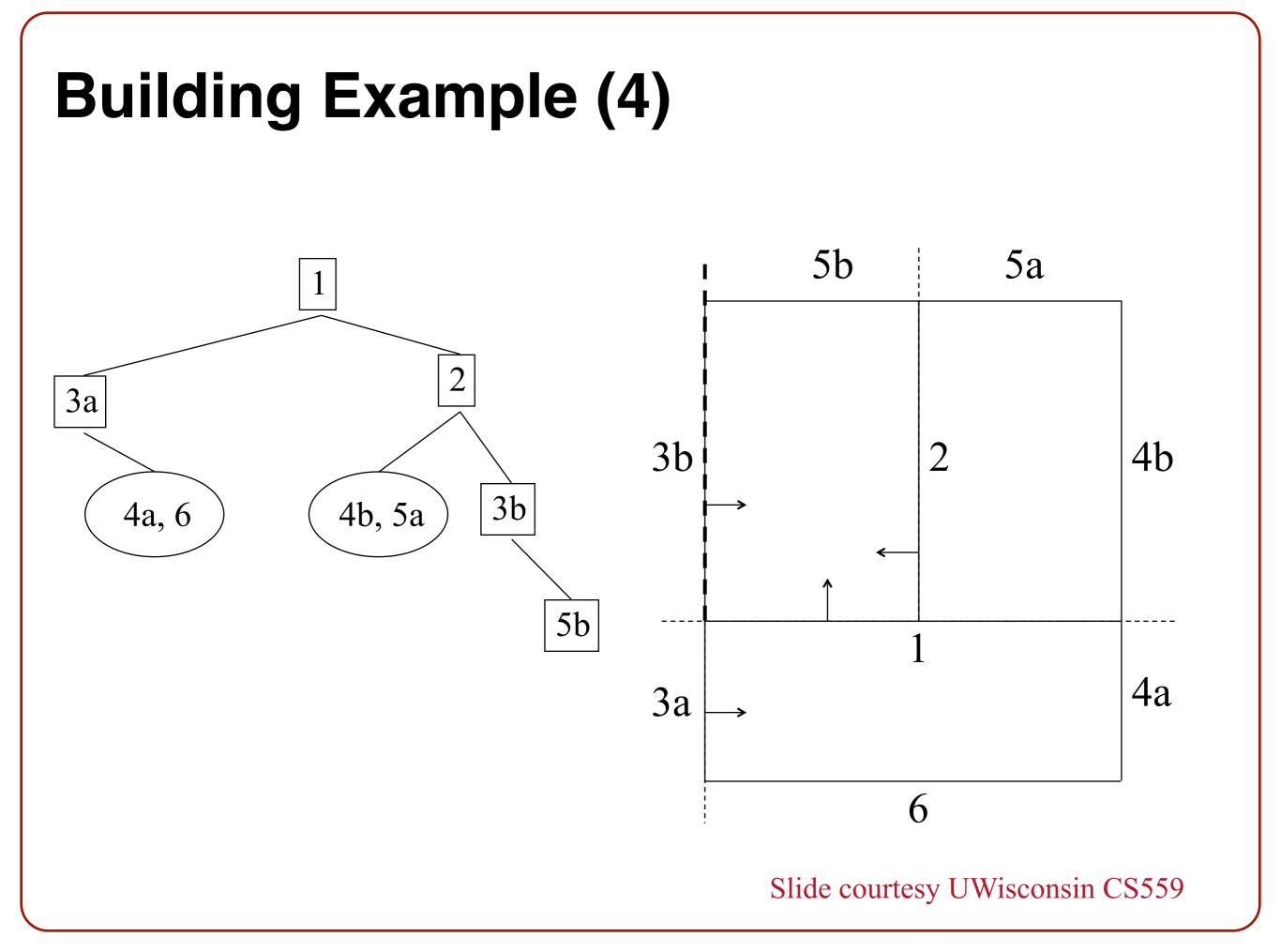
3

Slide courtesy UWisconsin CS559

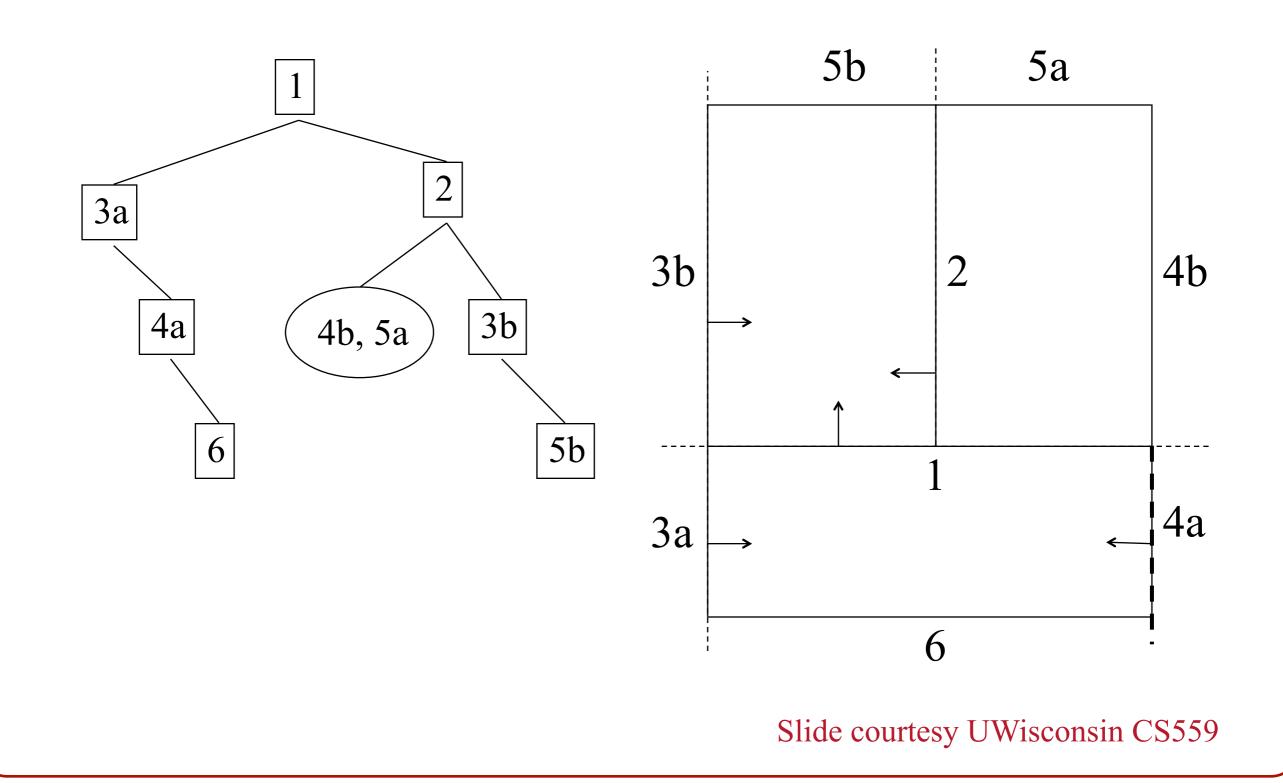




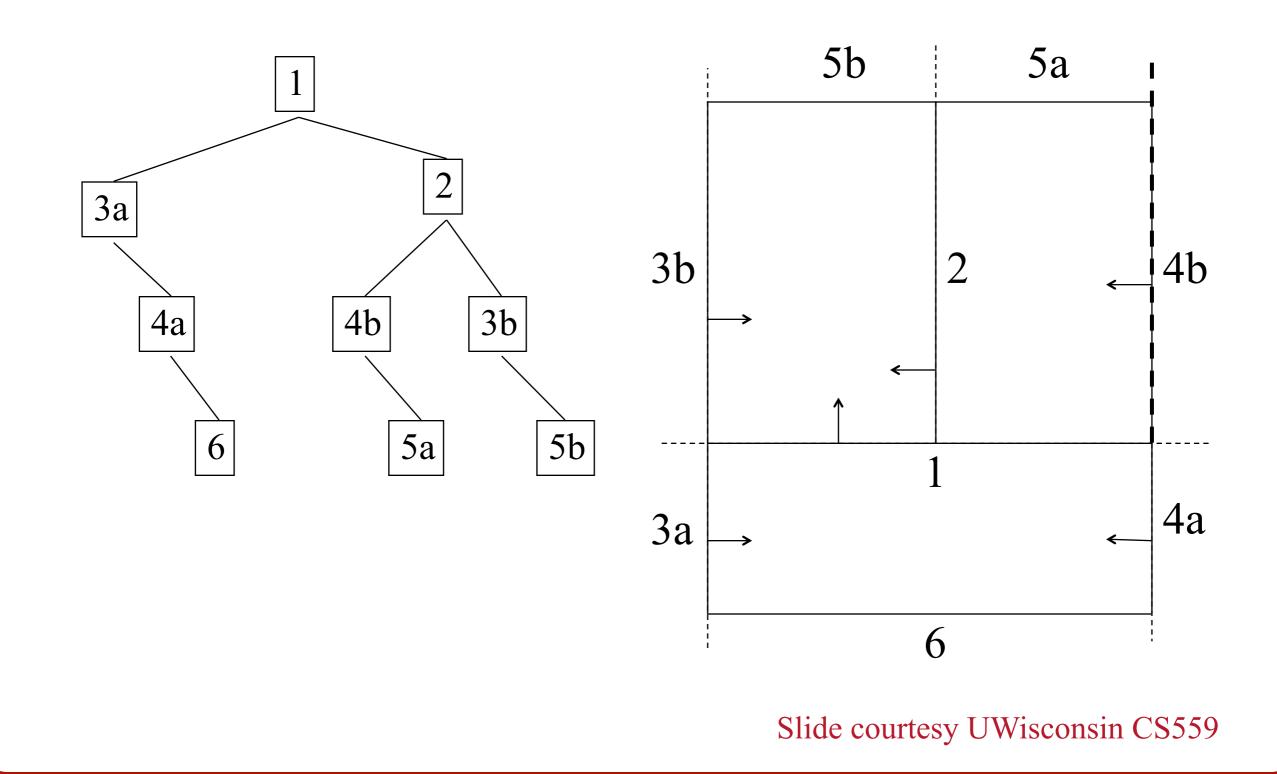




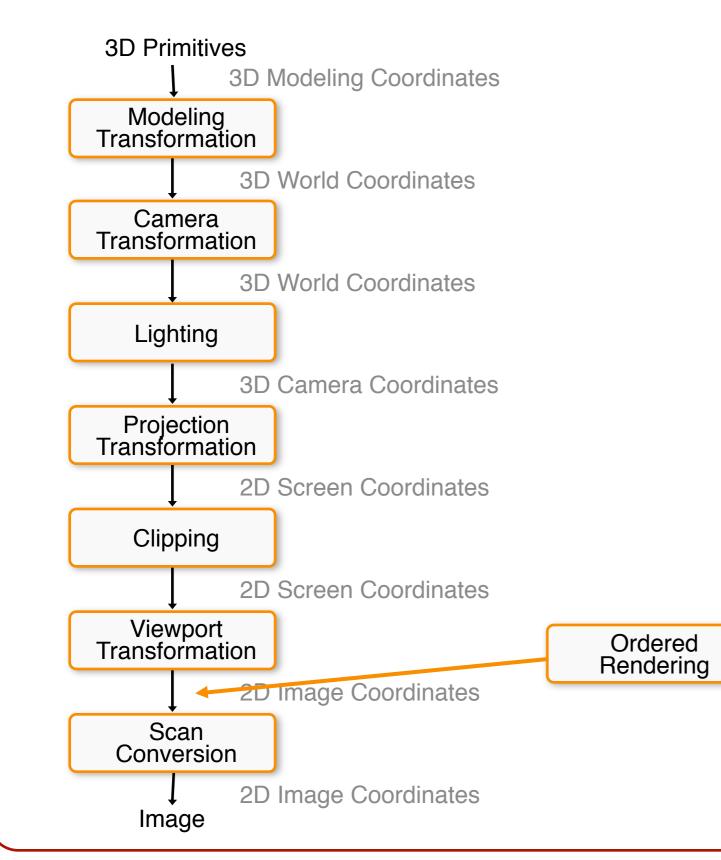
Building Example (5)



Building Example (Done)



3D Rendering Pipeline



Binary Space Partition:

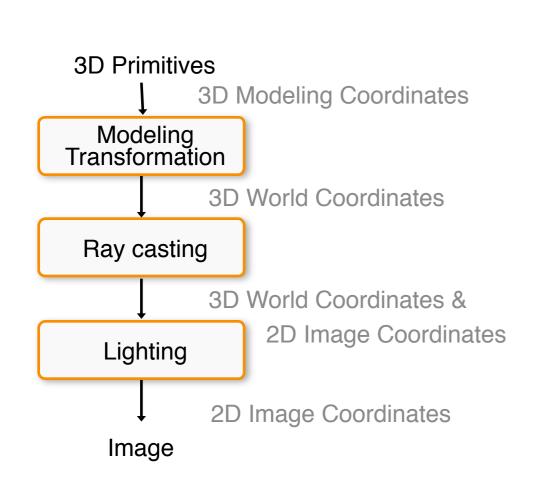
- View Independent
- Linear-time <u>depth</u> <u>sort</u>

Ray Casting

Fire a ray for every pixel olf ray intersects multiple objects, take the closest

0	0	0	0	0	0	9	0	0	0
•	•	0	0	0	0	0	•	0	9
0	•	0	0	•	0	0	ο	0	0
0	•	•	•	•	0	0	0	0	0
0	0	•		0	ο	0	ο	0	0
0	0	0	0	0	0	0	0	<u> </u>	0

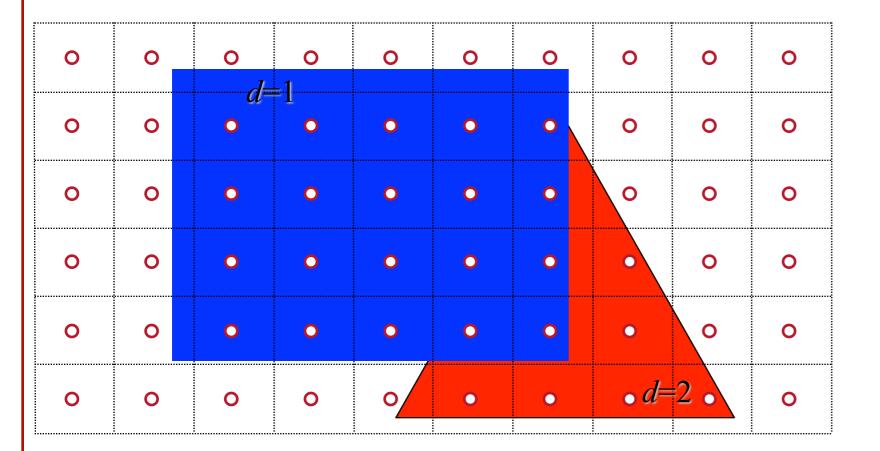
Ray Casting Pipeline



<u>Ray casting comments</u> **o** O(p log n) for p pixels **o** May (or not) use pixel coherence **o** Simple, but generally not used

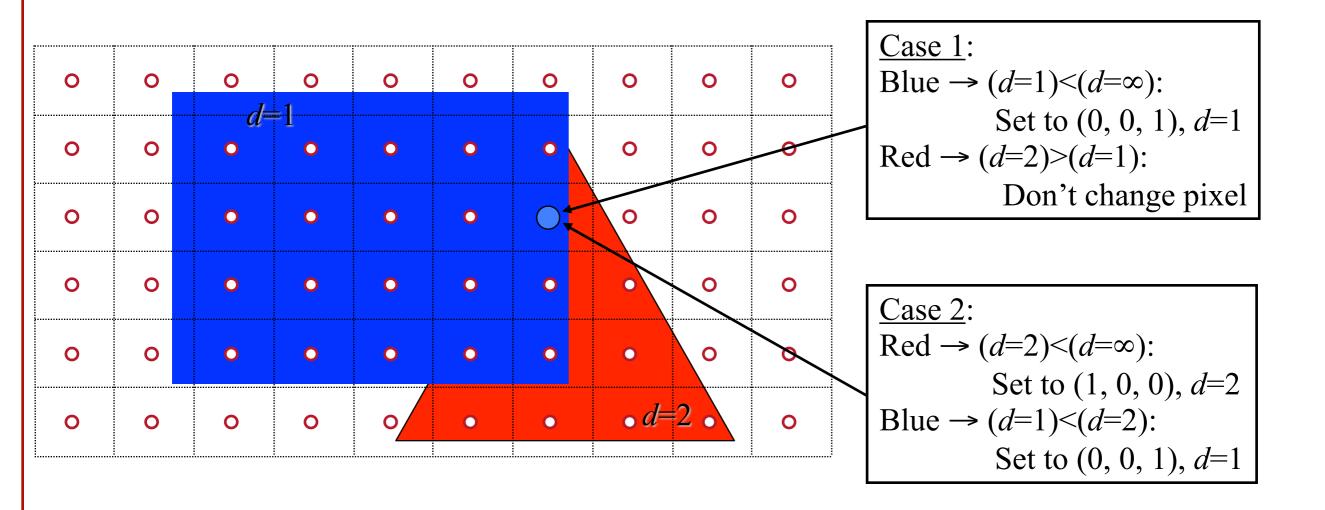
Z-Buffer

Store color & depth of closest object at each pixel
 oInitialize depth of each pixel to ∞
 oUpdate only pixels whose depth is closer than in buffer



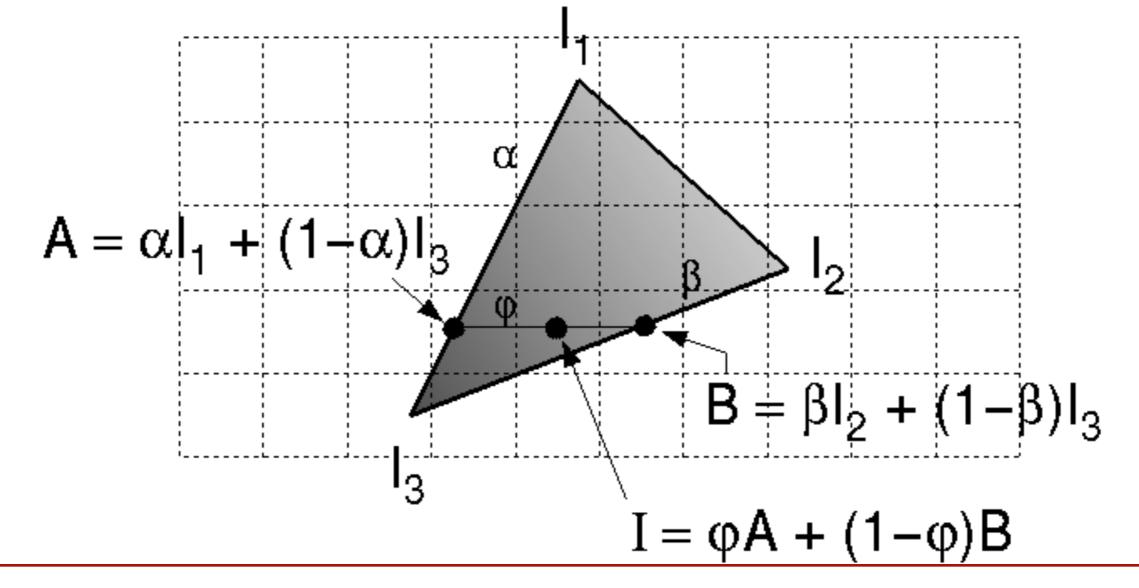
Z-Buffer

Store color & depth of closest object at each pixel
 oInitialize depth of each pixel to ∞
 oUpdate only pixels whose depth is closer than in buffer



Z-Buffer

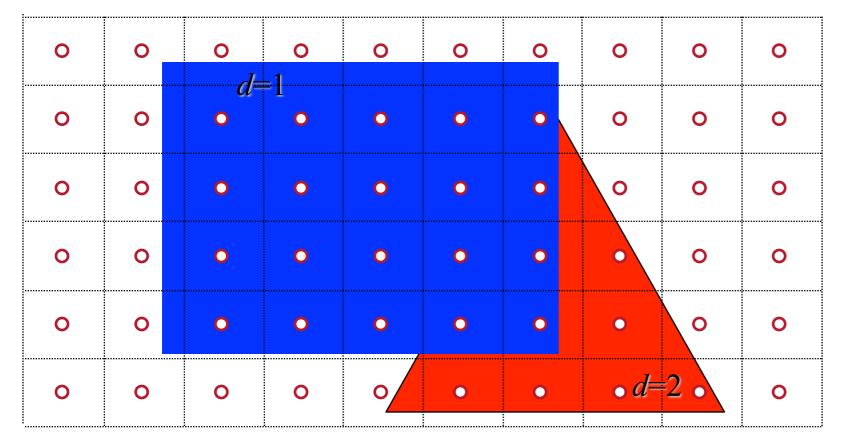
 Store color & depth of closest object at each pixel olnitialize depth of each pixel to ∞
 oUpdate only pixels whose depth is closer than in buffer
 oDepths are interpolated from vertices, just like colors



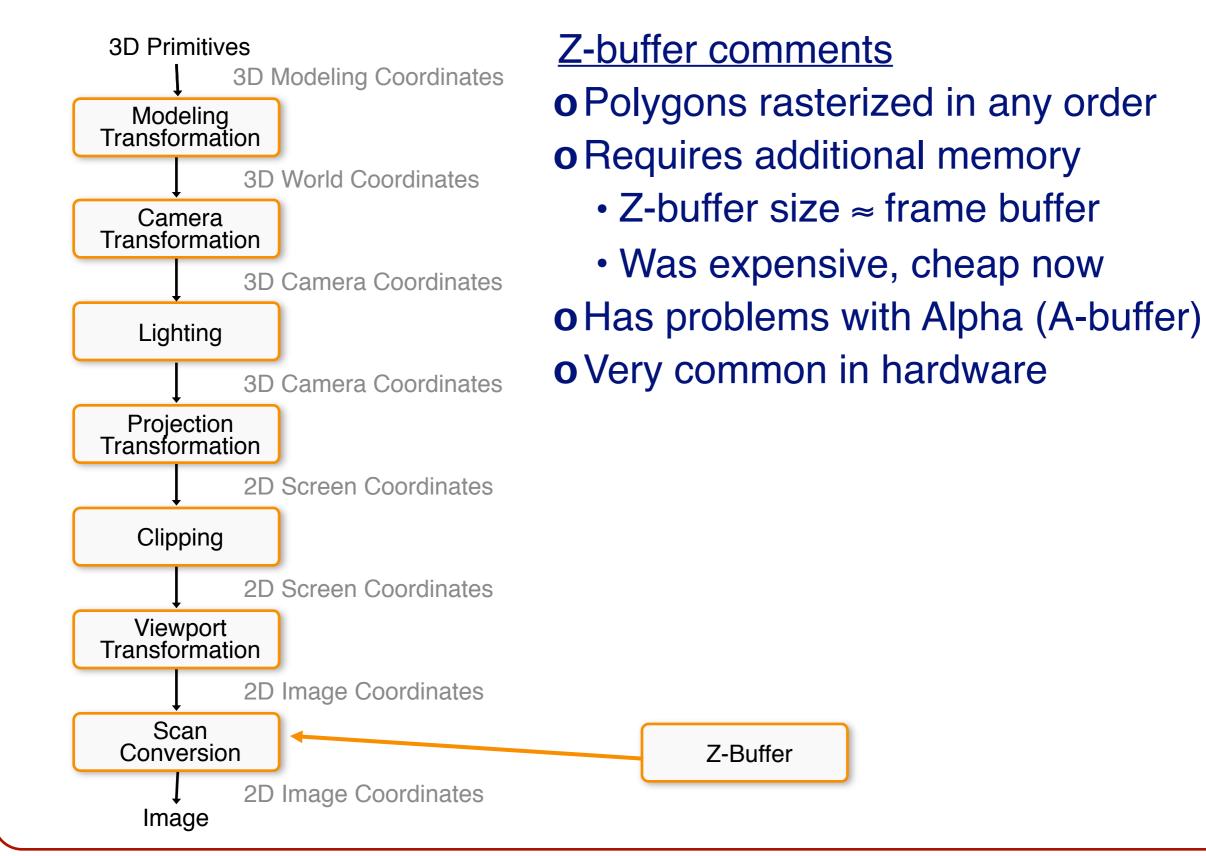
A-Buffer

Alpha values can cause problems:
 oZ-buffer can only find one visible surface at each pixel
 oA-buffer supports linked list of surfaces at each pixel for better transparency support

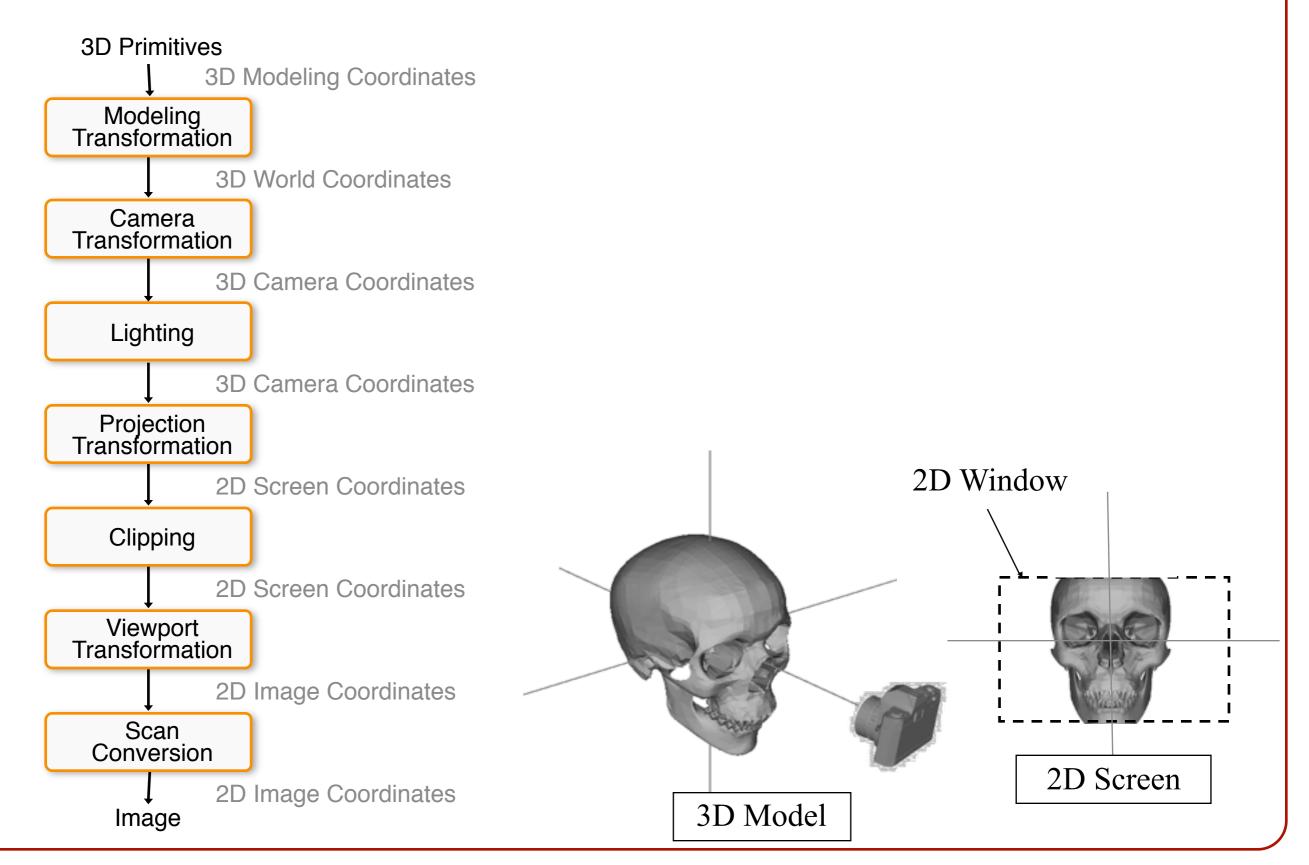
oA-buffer also helps with anti-aliasing



3D Rendering Pipeline



3D Rendering Pipeline (for direct illumination)



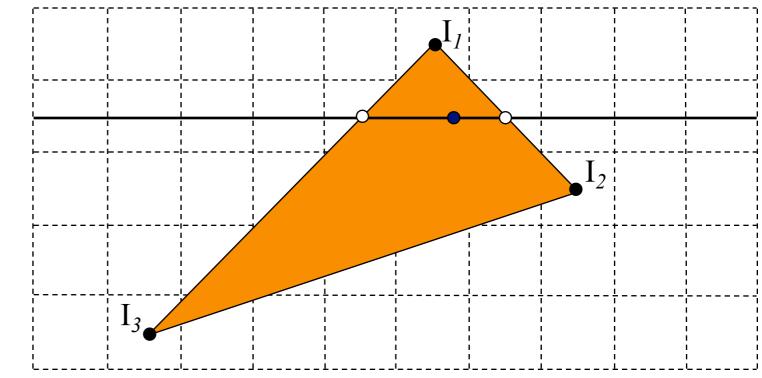
Scan Conversion

How do we average information from the three vertices of a triangle?

- oInterpolate using weights determined by the screen space projection?
- oInterpolate using weights determined by the 3D locations?

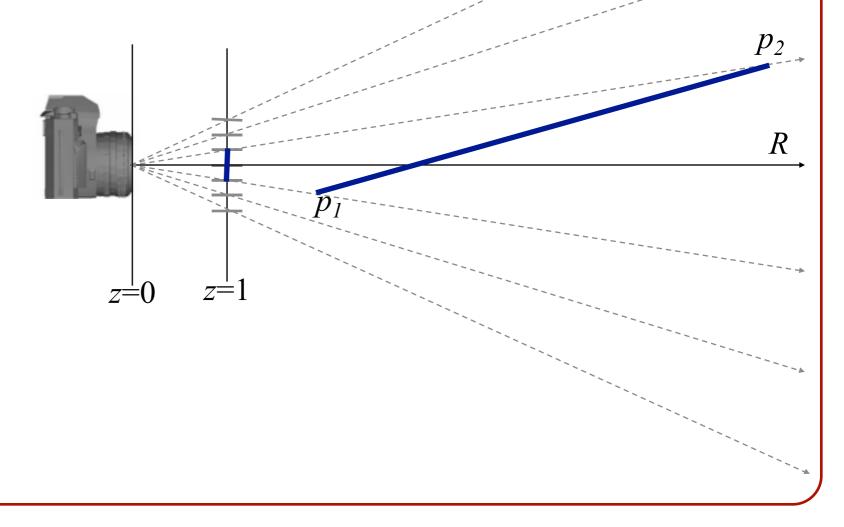
It's easier to do the interpolation in 2D.

Is there a difference?



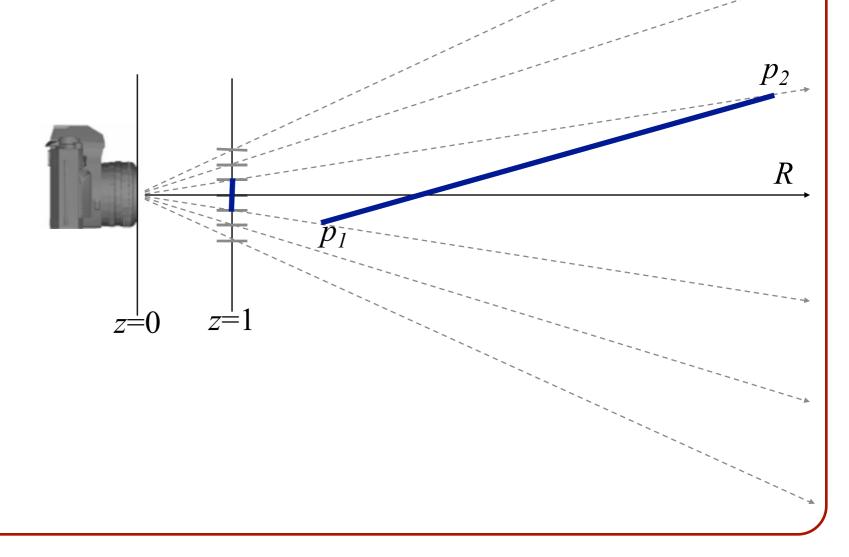
A line segment in 2D projected onto a 1D screen.

How should we interpolate the information from vertices p_1 and p_2 at the pixel corresponding to ray *R*?



A line segment in 2D projected onto a 1D screen.

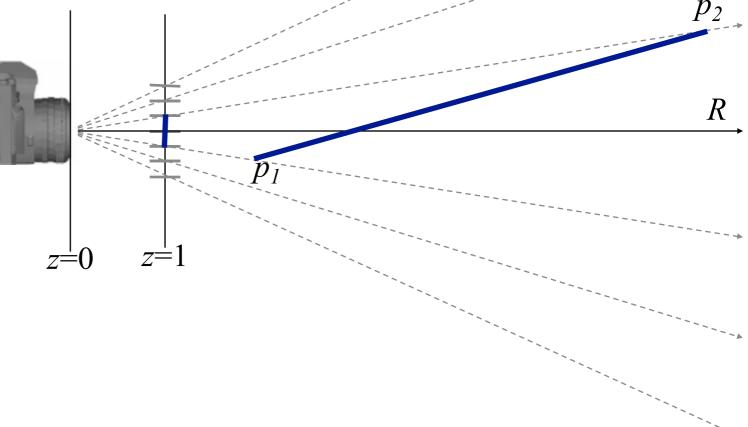
R intersects the projected line segment in the middle:
 o We should use equal contributions from *p*₁ and *p*₂.



A line segment in 2D projected onto a 1D screen.

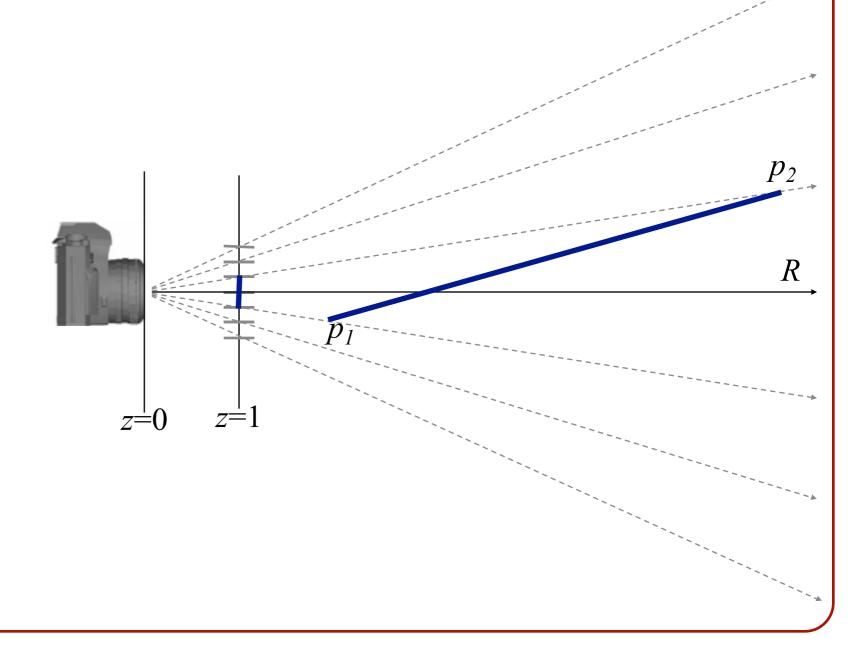
- *R* intersects the projected line segment in the middle:
 o We should use equal contributions from *p*₁ and *p*₂.
- *R* intersects the 2D line segment closer to p_1 :

o We should use more information from p_1 than from p_2 .



A line segment in 2D projected onto a 1D screen.

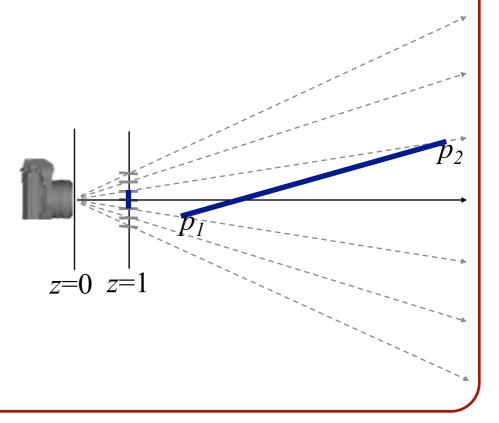
How do we interpolate correctly?



A line segment in 2D projected onto a 1D screen.

• How do we interpolate correctly?

<u>Recall</u>: The 2D point (x, z) maps to the point (x/z) in 1D.



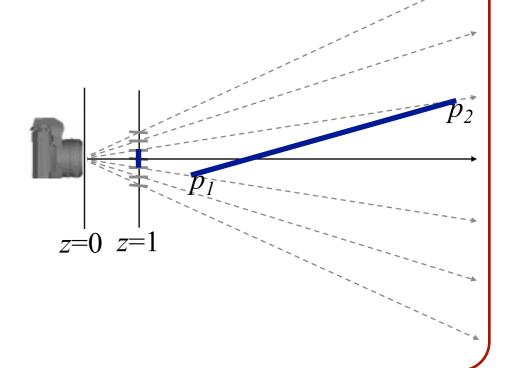
A line segment in 2D projected onto a 1D screen.

• How do we interpolate correctly?

<u>Recall</u>: The 2D point (x, z) maps to the point (x/z) in 1D.

If $p_1 = (x_1, z_1)$ and $p_2 = (x_2, z_2)$, to find the blending value for a pixel at position x in the screen we need to solve for α s. t.:

$$(1 - \alpha)(x_1, z_1) + \alpha(x_2, z_2) \to (x, 1)$$



A line segment in 2D projected onto a 1D screen.

• How do we interpolate correctly?

<u>Recall</u>: The 2D point (x, z) maps to the point (x/z) in 1D.

If $p_1 = (x_1, z_1)$ and $p_2 = (x_2, z_2)$, to find the blending value for a pixel at position x in the screen we need to solve for α s. t.:

A line segment in 2D projected onto a 1D screen.

• How do we interpolate correctly?

<u>Recall</u>: The 2D point (x, z) maps to the point (x/z) in 1D.

If $p_1 = (x_1, z_1)$ and $p_2 = (x_2, z_2)$, to find the blending value for a pixel at position x in the screen we need to solve for α s. t.:

$$(1 - \alpha)(x_1, z_1) + \alpha(x_2, z_2) \to (x, 1)$$

$$((1 - \alpha)x_1 + \alpha x_2, (1 - \alpha)z_1 + \alpha z_2) \to (x, 1)$$

$$(1 - \alpha)x_1 + \alpha x_2$$

$$\frac{(1 - \alpha)x_1 + \alpha x_2}{(1 - \alpha)z_1 + \alpha z_2} = x$$

A line segment in 2D projected onto a 1D screen.

• How do we interpolate correctly?

 $\frac{\text{Recall: The 2D point (x, z) many to the point (x/z) in 1D}{\text{To compute the interpolation weights correctly, we}$ If p need to perform a perspective divide! pixel at position x in the screen we need to solve for α s. t.: $(1-\alpha)(x_1, z_1) + \alpha(x_2, z_2) \to (x, 1)$ $((1-\alpha)x_1 + \alpha x_2, (1-\alpha)z_1 + \alpha z_2) \to (x,1)$ $\frac{(1-\alpha)x_1 + \alpha x_2}{(1-\alpha)z_1 + \alpha z_2} = x$

A line segment in 2D projected onto a 1D screen.

• How do we interpolate correctly?

Recall: The 2D point (x, z) maps to the point (y/z) in 1D.To compute the interpolation weights correctly, weIf p need to perform a perspective divide!pixel at position x in the screen we need to solve for α s. t.:

Note that this is not the same as solving for the blending value in the image plane:

$$\frac{(1-\alpha)x_1 + \alpha x_2}{(1-\alpha)z_1 + \alpha z_2} = x \qquad (1-\alpha)\frac{x_1}{z_1} + \alpha \frac{x_2}{z_2} = x$$