Pipeline and Rasterization

Readings: Chapter 8 (math: section 2.7)

Pipeline

The graphics pipeline

- The stand approach to objectorder graphics
- Many versions exist
 - Software, e.g., Pixar's EYYES architecture
 - Hardware, e.g., graphcs card (Nvidia)
 - Amazing performance: millions of triangles per frame
- Our focus: abstract version of hardware pipeline
- "Pipeline" because of many stages
 - easy to parallel
 - Remarkable performance of graphics card



Primitives

- Points
- Line segments
 - And chains of connected line segments
 - Triangles
- And that is all!
 - Curves? Approximate them with chains of line segments
 - Polygons? Break them up into triangles
 - Curved regions? Approximate them with triangles.
- Trend has been toward minimal primitivies
 - Simple, uniform, repetitive: good for parallelism

Rasterization

Primitives

- First job: enumerate the pixels covered by a primitive
 - Simple, aliased definition: pixels whose centers fall inside
- Second job: interpolate values across the primitive
 - E.g., colors computed at vertices
 - Normals at vertices
 - Will see applications later on

Rasterizing lines

Rasterizing lines



CMSC 435 / 634

August 2013

Pipeline and rasterization 9

Point sampling

																	_				
									•		•	٠			_	۰	•		•	٠	
	•		•		•				•	•		•		•	_	۰			0	•	0
	•	•			•			•	•	•		•				۰			•	•	0
									•	•	•	•	/	0		۰	0		•		
	•		•		•		•			~	-	•		0	_	۰	•		•	۰	
		•	•	•		•	-	~	•	•	•	•		•	_	•			0	•	
	•	•	•	•		•	•	•	•	•		•		•	_	0			•		
	•		-	•	•	•	•		•	•		•	•			•		-	-	-5	
	•		•		•	•	•	•	•			•		•		•			0	۰	0
	•				•		•		•		•					۰	•		•	•	
	•	. •	•		•	•	•	•	•	•		٠		•	_	0			0	•	
	•		•	•	•		•	•	•	•	•	•		•					0	•	
	•	•	•	•			•		•	•	•					0			0	•	0
	•	•			۰	۰	۰	•	•	0		•		۰		0			0	•	
	٠		٠		۰		٠	٠	•			٠		•		٠			•	•	
_													-	_							
	•	0	0	•	•	•		•						, ,	•	+		*			
-	•	0	0	0	•	•	0	0	0	0	•	0))	0	,	0	•	0		
	•	0	0	•	•	•	0	0	0	0	0	0					0 0	•	0	0	
	•	0	•	•	0 0 0	•	•	•	0 0 0	•	•	•	-				0	•	•		
	• • •	0 0 0 0	0 0 0	• • •	• • •	0 0 0	0 0 0	•	• • •	•	•						0 0 0	• • •	0	0	
	• • • •	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	• • • • • • • • • • • • • • • • • • • •	•	•	· · ·	•	•						0 0 0 0	• • • •	0		
	• • • • •	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	•	•	•	· · ·	· · ·	· · ·						0 0 0 0 0	• • • •	0 0 0 0		
	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0	· · ·	•	· · ·	• • • •	· · ·	•	· · ·	· ·	· · · ·						0 0 0 0 0 0	• • • • • • • • • • • • • • • • • • • •	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		
	0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	• • • • •		· · · ·	· · ·	· · ·	· · ·	· · · ·	· · · ·	- - - - - - - - - - - - - - - - - - -							0 0 0 0 0 0 0 0 0 0	0 0 0 0 0		
	0 0 0 0 0 0 0 0 0 0 0 0		· · · ·	· · · ·	• • • • •	· · · ·		· · · ·	· · · · ·	· · · ·	- - - - - - - - - - - - - - - - - - -						0 0 0 0 0 0 0 0 0 0 0 0 0 0	• • • • • • • • • • • • • •			
	0 0 0 0 0 0 0 0 0 0 0 0 0 0		• • • • • • •		• • • • • • •	· · · ·		· · · · ·	· · · · ·	· · · · ·								0 0 0 0 0 0 0 0 0 0			
			• • • • • • • • •		0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0		· · · · ·		· • • • • • • • • • • • • • • • • • •								• • • • • • • • • • • • • • • • • • •			
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		• • • • • • • • • • • • • • • •			0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		0 0 0 0 0 0 0 0 0 0 0 0 0 0			- - - - - - - - - - - - - - - - - - -							· • • • • • • • • • • • • • • • • • • •			
			0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0							· · · · · · · · · · · · · · · · · · ·											
			• • • • • • • • • • • • • • • • • • •																		

•	0	0	•		•	0		0	0		0		•	
•			•	0				•	0	•				
•		•		•		•	•	•	•		•			
												-		
•	•	•		•		•	•			0	-	•		
•		•	•	•		9	-		•					
		•	-		9	•	•		0					
•		•	-	•	•	•	•	0	0	•	•			
۰	•	0	•	0	•	•	•	0	0	•	•			
۰	•	•	•	•		•	•	•	0	•	•			
۰						•	•		•	•				
		•		•		•								
•		•	•	•		•	•			•	•			
٠	•	•	•	•		•	•	•	•	•	•			
			•			•								

	٠	•	•			•	٠				•		•	
	•		•										•	
			•	0				•			•		•	
	•		•	0	•									
•	•		•	0	•	•	0				•		•	
•	•		•		•				•				•	
						•								
•	•		•	۰	•								•	
	•			۰	•		•					٠		
•	٠			•		٠		•	•	•			•	
۰	•		۰	•		٠	•	٠	•	•	۰	٠	•	
•			۰	•	٠	۰		•		٠	۰		٠	
۰	۰	•	•	0	•	۰	۰	۰	•	•	•	٠	۰	
۰	۰	•	۰	•	•	•	•	۰	•	•	۰	٠	۰	
			•										•	

CMSC 435 / 634 A

August 2013

Pipeline and rasterization 10

Rasterizing lines algorithm

				•				•	•						-	•	•	•	•		•	•	•	•	•		•		•	F
					•				•						1	•		•	0			•		•	0				0	Γ
	٠	٠			•				•							•		•	•			•		•	•				•	
	٠			٠	٠				•			•	-	0			•	•			0			•	•					Γ
	*						•		•	~	0	•	•			•	•	•				•	•				•	•	•	
		•	۰	۰	٠			-	•							•		۰			•				۰				•	
	٠		•	•	_	•	•	•	۰			•	•	•			•							۰	۰					
			0	۰		•	•	٠	۰	•	•	•	•			•	•		۰		•	۰	•	۰	•				•	
	٠		۰		•		•		۰	•							٠	•			•	٠		•	۰	٠			•	
	٠		•	•	•		•		۰							۰	۰	۰			•	۰	•	۰	۰				•	
	٠		۰	•	•	•	•	۰	۰		•		•	•		٠	۰	۰	۰		•	۰		•	۰	۰	•	•	•	
	٠	•	۰	•	۰		•	۰	۰	•	•	•		•		٠	۰	٠	۰	•	•	۰		•	۰	٠	۰	٠	۰	
	٠	۰	۰	۰	۰		•	٠	۰		۰		۰	٠		۰	۰	۰	•	•	•	۰		۰	۰	۰	۰	٠	۰	
	٠	•	.0		۰		•	•	•	0	•		•			•	۰	•	•		•	•	•	۰	۰				۰	
					٠				•							•		٠	•		•	•	•	•	•			•	•	
_																														

Line equation: y = b + mx

Simple algorithm: //Evaluate line equation per column: for x=ceil(x0) to floor (x1) y = b + m * x output (x, round(y));

Point sampling result



Optimizing line drawing: Bresenham lines result (midpoint algorithm)



Midpoint algorithm

- At each pixel the only options are E and NE
- d = m(x+1) + b-y
- d>0.5 decides between E and NE
 - Only need to update d for integer steps in x and y; we can do that with addition



Midpoint algorithm

$$x = ceil(x0)$$

y = round(m*x + b)
d = m*(x+1)+b-y
while x0.5
y+=1
d-=1
x+=1
d+=m
Output(x,y)

Attributes interpolation

- Attributes:
 - Color
 - Normal vector

Rasterizing triangles

- The most common case in most applications
- Simple way to think of algorithm follows the pixel-walk interpretation of line rasterization
 - Walk from pixel to pixel over (at least the polygon's area)
 - Evaluate linear functions as you go
 - User those functions to decide which pixels are inside

Rasterizing triangles

- Input:
 - Three 2D points (the triangle coordinates in pixel space)
 - parameter attributes at each vertex
- Output
 - A list of fragments, each with
 - The integer pixel coordinates (x, y)
 - Interpolated parameter values

Rasterizing triangles



(See class notes for the drawing algorithm.)

Barycentric coordinates

 A coordinate system that does not use orthogonal basis

- Algebraic viewpoint:

$$\mathbf{p} = \alpha \mathbf{a} + \beta \mathbf{b} + \gamma \mathbf{c}$$

$$\mathbf{p} = \alpha \mathbf{a} + \beta \mathbf{b} + \gamma$$

$$\alpha + \beta + \gamma = 1$$

- Geometric viewpoint (areas)
 - (refer to in-class notes)



Barycentric coordinates

Properties

Geometric viewpoint: distances



 Linear viewpoint: basis of edges

$$\alpha = 1 - \beta - \gamma$$

$$\mathbf{p} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$$

Barycentric coordinates

Properties

Basis for the plane



– Triangle interior test:

 $\beta > 0; \quad \gamma > 0; \quad \beta + \gamma < 1$

Barycentric coordinates

- Calculation (derivation in class)
- Example: take a triangle and a point in this triangle from the terrain in project 2 and see how we calculate the point barycentric coordinates.



Pixel-walk rasterization



Primitives

- First job: enumerate the pixels covered by a primitive
 - Simple, aliased definition: pixels whose centers fall inside
- Second job: interpolate values across the primitive
 - E.g., colors computed at vertices
 - Normals at vertices
 - Will see applications later on

Compute colors



CMSC 435 / 634 August 2013

Pipeline and rasterization 27

Linear interpolation

- Pixels are not exactly on the line
- Define 2D function by projection on line
 - Linear in 2D
 - Use linear interpolation as the vertex calculation

Triangle coloring interpolation result



Compute normals

Insert normal

 We could associate the same normal/color to every point on the face of a triangle by computing

$$\mathbf{n} = \frac{(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)}{\|(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)\|}$$



Insert normal

 For example, we could associate the same normal/color to every **point** on the face of a triangle by computing

$$\mathbf{n} = \frac{(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)}{\|(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)\|}$$



This gives rise to flat shading/ coloring across the faces

Triangle Normals

Insert normal

Instead

 We could associate **normals** to every vertex:

 $T = ((p_1, n_1), (p_2, n_2), (p_3, n_3))$

so that the normal at point q in the triangle is the interpolation of the **normals** at the vertices



Insert normals

Two insertion results: which is better?



Triangle Normals



More uses: texture mapping



Clipping

- Rasterization tends to assume the triangles are on screen
 - Particularly problematic to have triangles crossing the plane Z=0
- After projection, before perspective divide

Clip against the 6 planes

Clipping a triangle against a plane

- 4 cases, based on the sidedness of vertices
 - -All in (keep)
 - All out (discard)
 - One in, two out (one clipped triangle)
 - Two in, one out (two clipped triangle)



Operations before and after rasterization

Pipeline revisited



Hidden surface removal

- We have discussed how to map primitive to image space
 - Projection and perspective are depth cues
 - Occlusion is another very important cue



Back face culling

 For closed shapes you will never see the inside

 Therefore only draw surfaces that face the camera



Back face culling

- For closed shapes you will never see the inside
 - Therefore only draw surfaces that face the camera
 - Implemented by checking
 n.v



The z buffer

- Draw in any order, keep track of closest
 - Allocate extra channel per pixel to keep track of closest depth so far
 - When drawing, compare object's depth to current closest depth and discard if greater