CIS 441/541: Intro to Computer Graphics
Lecture 5: interpolation of colors/depth
Announcements
541

• 220 Deschutes, 1pm-2pm
OH

• Tomorrow 11-12
• Dan Li: Weds, Thurs 4-530

• Looking forward:
  – Monday @ 9, 10, 1, 3?
  – Tues @ 10, 2, 3?
  – Weds @ 9, 10, 1?
  – Thurs @ 11, 2?
  – Fri @ 10, 2, 3?
Trying to Add the Class?

- Let’s talk after class...
Canvas is Operational

• Are you getting emails?
• Submit 1A/1B to Canvas...
Questions About Final Projects
Planning Ahead

• Going too fast in the lectures
  – Normally 2x80mins, but this time 3x50mins
  – This is bad ... I want lectures and assignments to stay in sync

• May devote Monday to a group OH for 1C
  – 1C is hard!
  – Only would be helpful if folks start 1C this weekend
Review
What is a field?

Example field (2D): temperature over the United States
How much data is needed to make this picture?

Example field (2D): temperature over the United States
Linear Interpolation for Scalar Field $F$
Linear Interpolation for Scalar Field F

• General equation to interpolate:
  – \( F(X) = F(A) + t*(F(B)-F(A)) \)

• \( t \) is proportion of \( X \) between \( A \) and \( B \)
  – \( t = (X-A)/(B-A) \)
Quiz Time #4

• F(3) = 5, F(6) = 11
• What is F(4)? = 5 + (4-3)/(6-3)*(11-5) = 7

• General equation to interpolate:
  \[ F(X) = F(A) + t*(F(B)-F(A)) \]

• t is proportion of X between A and B
  \[ t = (X-A)/(B-A) \]
Consider a single scalar field defined on a triangle.
Consider a single scalar field defined on a triangle.

\[
F(V2) = 2 \\
F(V1) = 10 \\
F(V3) = -2
\]
What is \( F(V4) \)?

- \( F(V1) = 10 \)
- \( F(V2) = 2 \)
- \( F(V3) = -2 \)

Point \( V4 \) is at \((0.5, 0.25)\).
What is $F(V4)$?
• Steps to follow:
  – Calculate V5, the left intercept for Y=0.25
  – Calculate V6, the right intercept for Y=0.25
  – Calculate V4, which is between V5 and V6
What is the X-location of V5?

Y-axis
Y=1
Y=0.5
Y=0

X-axis
X=0
X=0.5
X=1

V1, F(V1) = 10
V2, F(V2) = 2

V4, at (0.5, 0.25)

F(v) = A + ((v-v1)/(v2-v1))*(B-A):

F(v) = 0.25, find v
0.25 = 0 + ((v-0)/(1-0))*(1-0)
v = 0.25
What is the F-value of V5?

F(V1) = A \rightarrow F(0) = 10
F(V2) = B \rightarrow F(1) = 2
F(v) = A + ((v-v1)/(v2-v1))*(B-A):

v = 0.25, find F(v)

F(v) = 10 + ((0.25-0)/(1-0))*(2-10)
   = 10 + 0.25*(-8) = 10 -2 = 8
What is the X-location of V6?

F(v1) = A → F(1) = 1
F(v2) = B → F(2) = 0
F(v) = A + ((v-v1)/(v2-v1))*(B-A):

F(v) = 0.25, find v

0.25 = 1 + ((v-1)/(2-1))*(0-1)
    = 1 + (v-1)*(-1)
0.25 = 2 - v
v = 1.75
What is the F-value of V6?

F(v1) = A → F(1) = 2
F(v2) = B → F(2) = -2
F(v) = A + ((v-v1)/(v2-v1))*(B-A):

v = 1.75, find F(v)

F(v) = 2 + ((1.75-1)/(2-1))*(-2 - 2)
    = 2 + (.75)*(-4)
    = 2 - 3
    = -1
What is the F-value of V5?
What is the F-value of V5?

F(v1) = A  \rightarrow  F(0.25) = 8
F(v2) = B  \rightarrow  F(1.75) = -1
F(v) = A + ((v-v1)/(v2-v1))*(B-A):

v = 0.5, find F(v)

F(v) = 8 + ((0.5-0.25)/(1.75-0.25))*(-1-8)
= 8 + (0.25/1.5)*-9 = 8 - 1.5 = 6.5
Project 1C
Arbitrary Triangles

• The description of the scanline algorithm from Lecture 2 is general.

• But the implementation for these three triangles vary:

  Solve for location of this point and then solve two “base cases”.
Arbitrary Triangles

• Project #1B: implement the scanline algorithm for triangles with “flat bottoms”
• Project #1C: arbitrary triangles
Project #1C (3 points !!!), Due October 12th

- Goal: apply the scanline algorithm to arbitrary triangles and output an image.
- Extend your project1B code
- File proj1c_geometry.vtk available on web (80MB)
- File “get_triangles.cxx” has code to read triangles from file.
- No Cmake, project1c.cxx
Colors
What about triangles that have more than one color?
The color is in three channels, hence three scalar fields defined on the triangle.
Scanline algorithm

• Determine rows of pixels triangles can possibly intersect
  – Call them rowMin to rowMax
    • rowMin: ceiling of smallest Y value
    • rowMax: floor of biggest Y value

• For r in [rowMin à rowMax] ; do
  – Find end points of r intersected with triangle
    • Call them leftEnd and rightEnd
  – For c in [ceiling(leftEnd) à floor(rightEnd) ] ; do
    • ImageColor(r, c) ← triangle color
Scanline algorithm w/ Color

• Determine rows of pixels triangles can possibly intersect
  – Call them rowMin to rowMax
    • rowMin: ceiling of smallest Y value
    • rowMax: floor of biggest Y value

• For r in [rowMin \( \rightarrow \) rowMax] ; do
  – Find end points of r intersected with triangle
    • Call them leftEnd and rightEnd
  – Calculate Color(leftEnd) and Color(rightEnd) using interpolation from triangle vertices

  – For c in [ceiling(leftEnd) \( \rightarrow \) floor(rightEnd) ] ; do
    • Calculate Color(r, c) using Color(leftEnd) and Color(rightEnd)
    • ImageColor(r, c) \( \leftarrow \) Color(r, c)
Simple Example

What is the color at (2, 1)?

V(0,0) RGB = (1,0,0)
V(1,1) RGB = (0.5,0.5,0)
V(2,1) RGB = (0.25,0.5,0.25)
V(2,2) RGB = (0,1,0)
V(3,1) RGB = (0,0.5,0.5)
V(4,0) RGB = (0,0,1)
Scanline algorithm w/ Color

• Determine rows of pixels triangles can possibly intersect
  – Call them rowMin to rowMax
    • rowMin: ceiling of smallest Y value
    • rowMax: floor of biggest Y value

• For r in [rowMin → rowMax] ; do
  – Find end points of r intersected with triangle
    • Call them leftEnd and rightEnd
  – Calculate Color(leftEnd) and Color(rightEnd) using interpolation from triangle vertices
  – For c in [ceiling(leftEnd) → floor(rightEnd) ] ; do
    • Calculate Color(r, c) using Color(leftEnd) and Color(rightEnd)
    • ImageColor(r, c) ← Color(r, c)
Important

• ceiling / floor: needed to decide which pixels to light up
  – used: rowMin / rowMax, leftEnd / rightEnd
  – not used: when doing interpolation

Color(leftEnd) and Color(rightEnd) should be at the intersection locations ... no ceiling/floor.
How To Resolve When Triangles Overlap:
The Z-Buffer
Imagine you have a cube where each face has its own color.

<table>
<thead>
<tr>
<th>Face</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front</td>
<td>Blue</td>
</tr>
<tr>
<td>Right</td>
<td>Green</td>
</tr>
<tr>
<td>Top</td>
<td>Red</td>
</tr>
<tr>
<td>Back</td>
<td>Yellow</td>
</tr>
<tr>
<td>Left</td>
<td>Purple</td>
</tr>
<tr>
<td>Bottom</td>
<td>Cyan</td>
</tr>
</tbody>
</table>

View from “front/top/right” side
Imagine you have a cube where each face has its own color.

How do we render the pixels that we want and ignore the pixels from faces that are obscured?
Consider a scene from the right side

Camera/eyeball

Camera oriented directly at Front face, seen from the Right side

<table>
<thead>
<tr>
<th>Face</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front</td>
<td>Blue</td>
</tr>
<tr>
<td>Right</td>
<td>Green</td>
</tr>
<tr>
<td>Top</td>
<td>Red</td>
</tr>
<tr>
<td>Back</td>
<td>Yellow</td>
</tr>
<tr>
<td>Left</td>
<td>Purple</td>
</tr>
<tr>
<td>Bottom</td>
<td>Cyan</td>
</tr>
</tbody>
</table>
Consider the scene from the top side.

Camera/eyeball

Camera oriented directly at Front face, seen from the Top side

<table>
<thead>
<tr>
<th>Face</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front</td>
<td>Blue</td>
</tr>
<tr>
<td>Right</td>
<td>Green</td>
</tr>
<tr>
<td>Top</td>
<td>Red</td>
</tr>
<tr>
<td>Back</td>
<td>Yellow</td>
</tr>
<tr>
<td>Left</td>
<td>Purple</td>
</tr>
<tr>
<td>Bottom</td>
<td>Cyan</td>
</tr>
</tbody>
</table>
What do we render?

Green, Red, Purple, and Cyan all “flat” to camera. Only need to render Blue and Yellow faces (*).

<table>
<thead>
<tr>
<th>Face</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front</td>
<td>Blue</td>
</tr>
<tr>
<td>Right</td>
<td>Green</td>
</tr>
<tr>
<td>Top</td>
<td>Red</td>
</tr>
<tr>
<td>Back</td>
<td>Yellow</td>
</tr>
<tr>
<td>Left</td>
<td>Purple</td>
</tr>
<tr>
<td>Bottom</td>
<td>Cyan</td>
</tr>
</tbody>
</table>

Camera/eyeball

Camera oriented directly at Front face, seen from the Top side
What do we render?

What should the picture look like? What’s visible? What’s obscured?

Camera/eyeball

Camera oriented directly at Front face, seen from the Top side

<table>
<thead>
<tr>
<th>Face</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front</td>
<td>Blue</td>
</tr>
<tr>
<td>Right</td>
<td>Green</td>
</tr>
<tr>
<td>Top</td>
<td>Red</td>
</tr>
<tr>
<td>Back</td>
<td>Yellow</td>
</tr>
<tr>
<td>Left</td>
<td>Purple</td>
</tr>
<tr>
<td>Bottom</td>
<td>Cyan</td>
</tr>
</tbody>
</table>
New field associated with each triangle: depth

• Project 1B,1C:
  class Triangle
  {
    public:
      Double X[3];
      Double Y[3];
      ...
  };

• Now...
  Double Z[3];
What do we render?

Camera oriented directly at Front face, seen from the Top side

<table>
<thead>
<tr>
<th>Face</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front</td>
<td>Blue</td>
</tr>
<tr>
<td>Right</td>
<td>Green</td>
</tr>
<tr>
<td>Top</td>
<td>Red</td>
</tr>
<tr>
<td>Back</td>
<td>Yellow</td>
</tr>
<tr>
<td>Left</td>
<td>Purple</td>
</tr>
<tr>
<td>Bottom</td>
<td>Cyan</td>
</tr>
</tbody>
</table>
Using depth when rendering

• Use Z values to guide which geometry is displayed and which is obscured.

• Example....
Consider 4 triangles with constant Z values

- Z = -0.35
- Z = -0.5
- Z = -0.65
- Z = -0.8
Consider 4 triangles with constant $Z$ values

$Z = -0.35$

$Z = -0.5$

$Z = -0.65$

$Z = -0.8$

How do we make this picture?
Idea #1

• Sort triangles “back to front” (based on Z)
• Render triangles in back to front order
  – Overwrite existing pixels
Idea #2

• Sort triangles “front to back” (based on Z)
• Render triangles in front to back order
  – Do not overwrite existing pixels.
But there is a problem...
The Z-Buffer Algorithm

• The preceding 10 slides were designed to get you comfortable with the notion of depth/Z.
• The Z-Buffer algorithm is the way to deal with overlapping triangles when doing rasterization.
  – It is the technique that GPUs use.
• It works with opaque triangles, but not transparent geometry, which requires special handling
  – Transparent geometry discussed week 7.
  – Uses the front-to-back or back-to-front sortings just discussed.
The Z-Buffer Algorithm: Data Structure

- **Existing:** for every pixel, we store 3 bytes:
  - Red channel, green channel, blue channel
- **New:** for every pixel, we store a floating point value:
  - Depth buffer (also called “Z value”)

- Now 7 bytes per pixel (*)
  - (*): 8 with RGBA
The Z-Buffer Algorithm: Initialization

• Existing:
  – For each pixel, set R/G/B to 0.

• New:
  – For each pixel, set depth value to -1.

  – Valid depth values go from -1 (back) to 0 (front)
  – This is partly convention and partly because it “makes the math easy” when doing transformations.
Scanline algorithm

• Determine rows of pixels triangles can possibly intersect
  – Call them rowMin to rowMax
    • rowMin: ceiling of smallest Y value
    • rowMax: floor of biggest Y value

• For r in [rowMin → rowMax] ; do
  – Find end points of r intersected with triangle
    • Call them leftEnd and rightEnd
  – For c in [ceiling(leftEnd) → floor(rightEnd) ] ; do
    • ImageColor(r, c) ← triangle color
Scanline algorithm w/ Z-Buffer

• Determine rows of pixels triangles can possibly intersect
  – Call them rowMin to rowMax
    • rowMin: ceiling of smallest Y value
    • rowMax: floor of biggest Y value

• For r in [rowMin \rightarrow rowMax] ; do
  – Find end points of r intersected with triangle
    • Call them leftEnd and rightEnd
  – Interpolate z(leftEnd) and z(rightEnd) from triangle vertices
  – For c in [ceiling(leftEnd) \rightarrow floor(rightEnd) ] ; do
    • Interpolate z(r,c) from z(leftEnd) and z(rightEnd)
    • If (z(r,c) > depthBuffer(r,c))
      – ImageColor(r, c) \leftarrow triangle color
      – depthBuffer(r,c) = z(r,c)
The Z-Buffer Algorithm: Example

(0,0) (12,0)

(12,12)

Y=5

(0,12) (12,12)

(2.5, 10.5, -0.25)

(2.5, 2.5, -0.5)

(10.5, 2.5, -1)
The Z-Buffer Algorithm: Example
Interpolation and Triangles

• We introduced the notion of interpolating a field on a triangle
• We used the interpolation in two settings:
  – 1) to interpolate colors
  – 2) to interpolate depths for z-buffer algorithm
• Project 1D: you will be adding color interpolation and the z-buffer algorithm to your programs.