Announcements
Canvas is Operational

• Piazza
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
• Will Monday to a group OH for 1C
  – 1C is hard!
  – Only would be helpful if folks start 1C this weekend
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
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 \( F(4) ? = 5 + \frac{(4-3)}{(6-3)}*(11-5) = 7 \)

• General equation to interpolate:
  \[
  F(X) = F(A) + t*(F(B)-F(A))
  \]
• \( t \) is proportion of \( X \) between \( A \) and \( B \)
  \[
  t = \frac{(X-A)}{(B-A)}
  \]
What is $F(V4)$?
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
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 → 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/ 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)?
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)

Calculating multiple color channels here!
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....

View from “front/top/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>
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?

View from “front/top/right” side

View from “back/bottom/left” side
Consider a scene from the right side

Camera/eyeball

<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 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 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?

<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 oriented directly at Front face, seen from the Top side
Using depth when rendering

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

• Example....
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 TODAY.
  – 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 → 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) → floor(rightEnd)] ; do
    • Interpolate z(r, c) from z(leftEnd) and z(rightEnd)
    • If (z(r, c) > depthBuffer(r, c))
      – ImageColor(r, c) ← triangle color
      – depthBuffer(r, c) = z(r, c)
Zbuffer example

Camera/eyeball

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

Consider pixel along this ray

<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>
**Zbuffer example**

Consider pixel along this ray

Initial values (for this pixel):
- RGB = 0/0/0
- Z = -1.0

Which geometry is rendered next (blue or yellow)?
Zbuffer example: yellow then blue

Initial values (for this pixel):
RGB = 0/0/0
Z = -1.0

After yellow quad:
RGB = 255/255/0
Z = -0.8

After blue quad?:
RGB = 0/0/255
Z = -0.2
Zbuffer example: blue then yellow

Consider pixel along this ray

Initial values (for this pixel):
RGB = 0/0/0
Z = -1.0

After blue quad:
RGB = 0/0/255
Z = -0.2

After yellow quad?:
RGB = 0/0/255
Z = -0.2
Transparent Geometry
Compositing and Blending

Ed Angel
Professor of Computer Science, Electrical and Computer Engineering, and Media Arts
University of New Mexico
Opacity and Transparency

• Opaque surfaces permit no light to pass through
• Transparent surfaces permit all light to pass
• Translucent surfaces pass some light

translucency = 1 – opacity (\(\alpha\))
Transparency

• If you have an opaque red square in front of a blue square, what color would you see?
  – Red

• If you have a 50% transparent red square in front of a blue square, what color would you see?
  – Purple

• If you have a 100% transparent red square in front of a blue square, what color would you see?
  – Blue
(One) Formula For Transparency

- Front = (Fr,Fg,Fb,Fa)
  - $a = \alpha$, transparency factor
    - Sometimes percent
    - Typically 0-255, with 255 = 100%, 0 = 0%
- Back = (Br,Bg,Bb,Ba)
- Equation = $(Fa \cdot Fr + (1-Fa) \cdot Br,
  Fa \cdot Fg + (1-Fa) \cdot Bg,
  Fa \cdot Fb + (1-Fa) \cdot Bb,
  Fa + (1-Fa) \cdot Ba)$

Alpha component is important! Any observations?
Transparency

• If you have an 25% transparent red square (255,0,0) in front of a blue square (0,0,255), what color would you see (in RGB)?
  – (192,0,64)

• If you have an 25% transparent blue square (0,0,255) in front of a red square (255,0,0), what color would you see (in RGB)?
  – (64,0,192)
Implementation

- Per pixel storage:
  - RGB: 3 bytes
  - Alpha: 1 byte
  - Z: 4 bytes

- Alpha used to control blending of current color and new colors
New vocab term: fragment

- Fragment is the contribution of a triangle to a single pixel

**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 $\rightarrow$ rowMax] ; do
    - Find end points of r intersected with triangle
      - Call them leftEnd and rightEnd
    - For c in [ceiling(leftEnd) $\rightarrow$ floor(rightEnd)] ; do
      - ImageColor(r, c) $\leftarrow$ triangle color
Examples

• Imagine pixel \((i, j)\) has:
  – RGB = 255/255/255
  – Alpha=255
  – Depth = -0.5
• And we contribute fragment:
  – RGB=0/0/0
  – Alpha=128
  – Depth = -0.25
• What do we get?
• Answer: 128/128/128, \(Z = -0.25\)
• What’s the alpha?
Examples

• Imagine pixel \((i, j)\) has:
  – RGB = 255/255/255
  – Alpha = 128
  – Depth = -0.25

• And we contribute fragment:
  – RGB = 0/0/0
  – Alpha = 255
  – Depth = -0.5

• What do we get?
• Answer: (probably) 128/128/128, \(Z = -0.25\)
• What’s the alpha?
System doesn’t work well for transparency

• Contribute fragments in this order:
  – Z=-0.1
  – Z=-0.9
  – Z=-0.5
  – Z=-0.4
  – Z=-0.6

• Model is too simple. Not enough info to resolve!
Order Dependency

• Is this image correct?
  - Probably not
  - Polygons are rendered in the order they pass down the pipeline
  - Blending functions are order dependent
How do you sort?

1) Calculate depth of each triangle center.
2) Sort based on depth
   - Not perfect, but good

In practice: sort along X, Y, and Z and use “dominant axis” and only do “perfect sort” when rotation stops
But there is a problem...
Depth Peeling

• a multi-pass technique that renders transparent polygonal geometry without sorting

• Pass #1:
  – render as opaque, but note opacity of pixels placed on top
  – treat this as “top layer”
  – save Z-buffer and treat this as “max”

• Pass #2:
  – render as opaque, but ignore fragments beyond “max”

• repeat, repeat...