CIS 441/541: Intro to Computer Graphics
Lecture 2: The Scanline Algorithm
Outline

• Project 1A
• Scanline Algorithm
• Project 1B
• Tips on floating point precision
Outline

• Project 1A
• Scanline Algorithm
• Project 1B
• Tips on floating point precision
Project #1A

- **Goal:** write a specific image
- **Due:** 11:59pm, Oct. 4th ... in 84 hours (!)
- **% of grade:** 2%
- **Q:** Why do I only get 3 1/2 days to complete this project?
- **A:** We need to need to get multi-platform issues shaken out ASAP.
- **May be a little painful**

**Setup:**
1. Download and install CMake. Use version 2.8.12.2
2. Download, build, and install VTK. Use version 6.1.0
3. Make directory called "project1A"
4. Download file project1A.cxx and CMakelists.txt from class website and copy them into directory project1A
5. Run CMake. This will create build files.
6. Compile the program. For Unix/Mac, this means “make”
7. Run the program.
8. It should output an image that is 1024x1024 called oneRedPixel.png. The first pixel of the file should be red (although that might be hard to eyeball)

**Assignment:**
1. You are to make an image that is 1024x1350.
   a. The image will be broken into 27 horizontal strips, with each strip of 50 pixels
   b. The color for the Xth strip should be:
      a. (X) % 3 = 0 \rightarrow R=0
      b. (X) % 3 = 1 \rightarrow B=128
      c. (X) % 3 = 2 \rightarrow G=255
      d. (X/3) % 3 = 0 \rightarrow G = 0
      e. (X/3) % 3 = 1 \rightarrow G=128
      f. (X/3) % 3 = 2 \rightarrow G=255
      g. X/9 = 0 \rightarrow B=0
      h. X/9 = 1 \rightarrow R=128
      i. X/9 = 2 \rightarrow B=255
   2) Examples:
      a. The first strip (which is at the beginning of the image buffer and at the bottom of the image) is to be black. R=0, G=0, B=0
      b. The strip immediately above that should be dark blue, R=0, G=0, B=255
      c. Above that should be bright blue R=0, G=0, B=255
      d. Above that should be dark green, R=0, G=128, B=0

The correct answer is located on the class website.

There is also an image differencer program on the class website. You can use that to verify that your image is correct. You should do this for this assignment.

When you are done, verify you have the correct image via the differencer. Then submit the following:
1. your source code
2. a screen capture showing the output of differencer
Project #1A: background

• Definitions:
  – Image: 2D array of pixels
  – Pixel: A minute area of illumination on a display screen, one of many from which an image is composed.

• Pixels are made up of three colors: Red, Green, Blue (RGB)

• Amount of each color scored from 0 to 1
  – 100% Red + 100% Green + 0% Blue = Yellow
  – 100% Red + 0% Green + 100 %Blue = Purple
  – 0% Red + 100% Green + 0% Blue = Cyan
  – 100% Red + 100% Blue + 100% Green = White
Project #1A: background

• Colors are 0->1, but how much resolution is needed? How many bits should you use to represent the color?
  – Can your eye tell the difference between 8 bits and 32 bits?
  – → No. Human eye can distinguish ~10M colors.
  – 8bits * 3 colors = 24 bits = ~16M colors.

• Red = (255,0,0)
• Green = (0,255,0)
• Blue = (0,0,255)
Project #1A: background

• An “M by N” 8 bit image consists of MxNx3 bytes.
  – It is stored as:
    P0/R, P0/G, P0/B, P1/R, P1/G, P1/B, ... P(MxN)/R, P(MxN)/G, P(MxN)/B

• P0 is the bottom, left pixel
• P(M-1) is the bottom, right pixel
• P((MxN)-M+1) is the top, left pixel
• P(MxN) is the top, right pixel
What is the image I’m supposed to make?
What do I do again?

- Install CMake & VTK.
- Download “project1A.cxx” from class website
- Download “CMakeLists.txt” from class website
- Run CMake
- Modify project1A.cxx to complete the assignment
- And...
- Submit the source and image to Blackboard by Saturday midnight.
Outline

- Project 1A
- Scanline Algorithm
- Project 1B
Reminder: ray-tracing vs rasterization

- Two basic ideas for rendering: rasterization and ray-tracing
- Ray-tracing: cast a ray for every pixel and see what geometry it intersects.
  - $O(n\text{Pixels})$
    - (actually, additional computational complexity for geometry searches)
  - Allows for beautiful rendering effects (reflections, etc)
  - Will discuss at the end of the quarter
Reminder: ray-tracing vs rasterization

• Two basic ideas for rendering: rasterization and ray-tracing

• Rasterization: examine every triangle and see what pixels it covers.
  – $O(n\text{Triangles})$
    • (actually, additional computational complexity for painting in pixels)
  – GPUs do rasterization very quickly
  – Our focus for the next 5 weeks
What color should we choose for each of these four pixels?
What color should we choose for each of these four pixels?
What color should we choose for each of these four pixels?
What color should we choose for each of these four pixels?
What color should we choose for each of these four pixels?
The middle and lower-left variants are half-pixel translations of the other.

This is the convention we will use for the projects.)
Where we are...

• We haven’t talked about how to get triangles in position.
  – Arbitrary camera positions through linear algebra
• We haven’t talked about shading
• Today, we are tackling this problem:
  How to deposit triangle colors onto an image?
Problem: how to deposit triangle colors onto an image?

• Let’s take an example:
  – 12x12 image
  – Red triangle
    • Vertex 1: (2.5, 1.5)
    • Vertex 2: (2.5, 10.5)
    • Vertex 3: (10.5, 1.5)
    • Vertex coordinates are with respect to pixel locations
Our desired output

How do we make this output? Efficiently?
Don’t need to consider any pixels outside these lines.
Scanline algorithm: consider all rows that can possibly overlap

Don't need to consider any pixels outside these lines.
Scanline algorithm: consider all rows that can possibly overlap

We will extract a “scanline”, i.e. calculate the intersections for one row of pixels

Don’t need to consider Pixels outside these lines

Y=5
– Red triangle
  
  • Vertex 1: (2.5, 1.5)
  • Vertex 2: (2.5, 10.5)
  • Vertex 3: (10.5, 1.5)
– Red triangle
  • Vertex 1: (2.5, 1.5)
  • Vertex 2: (2.5, 10.5)
  • Vertex 3: (10.5, 1.5)
– Red triangle
  • Vertex 1: (2.5, 1.5)
  • Vertex 2: (2.5, 10.5)
  • Vertex 3: (10.5, 1.5)

What are the end points?
Red triangle

• Vertex 1: (2.5, 1.5)
• Vertex 2: (2.5, 10.5)
• Vertex 3: (10.5, 1.5)

What are the end points?

(2.5, 5)
Red triangle

- Vertex 1: (2.5, 1.5)
- Vertex 2: (2.5, 10.5)
- Vertex 3: (10.5, 1.5)

What are the end points?

(2.5, 5)

Y = 5

Algebra!
Red triangle
- Vertex 1: (2.5, 1.5)
- Vertex 2: (2.5, 10.5)
- Vertex 3: (10.5, 1.5)

$Y = mx + b$
- $10.5 = m \times 2.5 + b$
- $1.5 = m \times 10.5 + b$

\[ \rightarrow \]
- $9 = -8m$
- $m = -1.125$
- $b = 13.3125$
- $5 = -1.125 \times x + 13.3125$
- $x = 7.3888$ (2.5, 5)

What are the end points?
Scanline algorithm: consider all rows that can possibly overlap

Don’t need to consider any

Pixels outside these lines

2.5

7.3888

Y=5
Scanline algorithm: consider all rows that can possibly overlap

Don't need to consider any pixels outside these lines

Color is deposited at (3,5), (4,5), (5,5), (6,5), (7,5)
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

- Determine rows of pixels triangles can possibly intersect
- For \( r \) in \([\text{rowMin} \rightarrow \text{rowMax}]\) ; do
  - Find end points of \( r \) intersected with triangle
    - Call them leftEnd and rightEnd
  - For \( c \) in \([\text{ceiling}(\text{leftEnd}) \rightarrow \text{floor}(\text{rightEnd})]\) ; do
    - \( \text{ImageColor}(r, c) \leftarrow \text{triangle color} \)

For \( r = 5 \), we call \( \text{ImageColor} \) with \((5,3), (5,4), (5,5), (5,6), (5,7)\)

\( \text{Y values from 1.5 to 10.5 mean rows 2 through 10} \)

For \( r = 5 \), leftEnd = 2.5, rightEnd = 7.3888
Arbitrary Triangles

• The description of the scanline algorithm in the preceding slides is general.
• But the implementation for these three triangles vary:
Supersampling: use the scanline algorithm a bunch of times to converge on the “average” picture.
Where we are...

• We haven’t talked about how to get triangles into position.
  – Arbitrary camera positions through linear algebra
• We haven’t talked about shading
• Today, we tackled this problem:
  How to deposit triangle colors onto an image?
  Still don’t know how to:
  1) Vary colors (easy)
  2) Deal with triangles that overlap
Outline

• Project 1A
• Scanline Algorithm
• Project 1B
• Tips on floating point precision
Arbitrary Triangles

• You will implement the scanline algorithm for triangles with “flat bottoms”
Goal: apply the scanline algorithm to “flat bottom” triangles and output an image.

File “project1B.cxx” has triangles defined in it.

Due: Weds, Oct. 8th

% of grade: 6%
Project #1C

- You will implement the scanline algorithm for arbitrary triangles ... plan ahead
Outline

• Project 1A
• Scanline Algorithm
• Project 1B
• Tips on floating point precision
Project 1B

- `cout/cerr` can be misleading:

```cpp
fawcett:Downloads child$ cat t2.C
#include <iostream.h>
#include <iomanip>

int main()
{
    double X=188;
    X-=1e-12;
    cerr << X << endl;
    cerr << std::setprecision(16) << X << endl;
}
fawcett:Downloads child$ ./a.out
188
187.999999999999999
The limited accuracy of cerr/cout can cause other functions to appear to be wrong:

```c
#include <iostream.h>
#include <iomanip>
#include <math.h>

int main()
{
    double X=188;
    X+=1e-12;
    cerr << "The floor of " << X << " is " << floor(X) << endl;
}
```

```
fawcett:Downloads child$ cat t3.C
The floor of 188 is 187
```
Floating point precision is an approximation of the problem you are trying to solve.

Tiny errors are introduced in nearly every operation you perform.

- Exceptions for integers and denominators that are a power of two.

Fundamental problem:

- Changing the sequence of these operations leads to *different* errors.

Example: \((A+B)+C \neq A+(B+C)\)
For project 1B, we are making a binary decision for each pixel: should it be colored or not?

Consider when a triangle vertex coincides with the bottom left of a pixel:

We all do different variations on how to solve for the endpoints of a line, so we all get slightly different errors.
Our algorithm incorporates floor and ceiling functions.

- This is the right place to bypass the precision problem.
- I have included "floor441" and "ceil441" in project prompt. You need to use them, or you will get one pixel differences.
Project 1B: other thoughts

- You will be building on this project ...
  - think about magic numbers (e.g. screen size of 1000)
  - add safeguards against cases that haven’t shown up yet
  - Assume nothing!