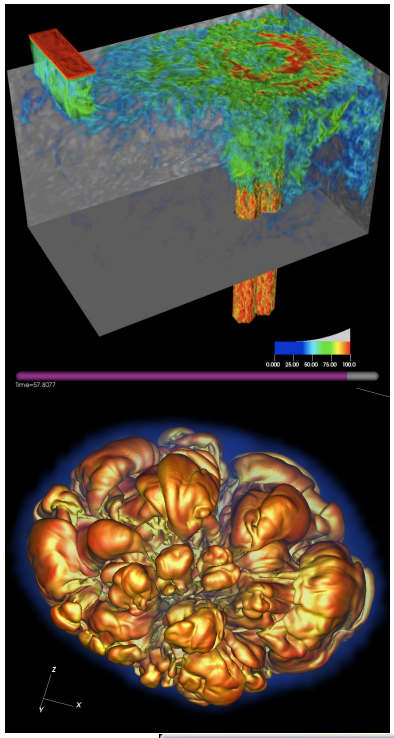
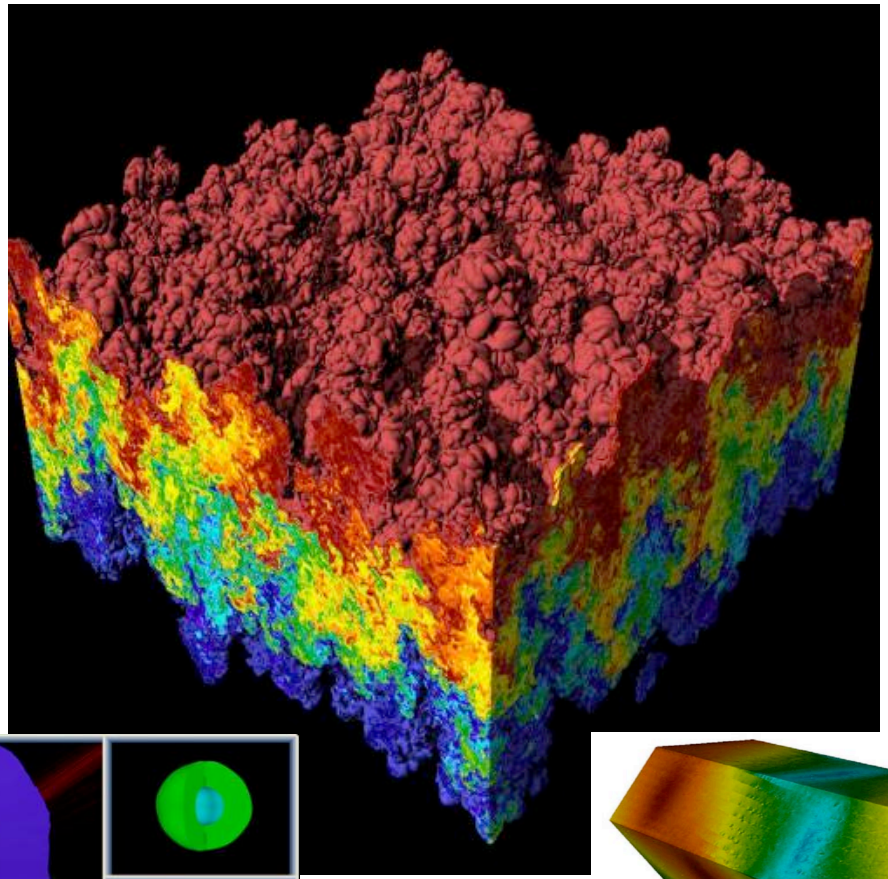
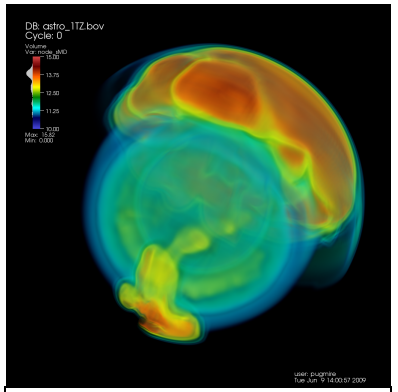
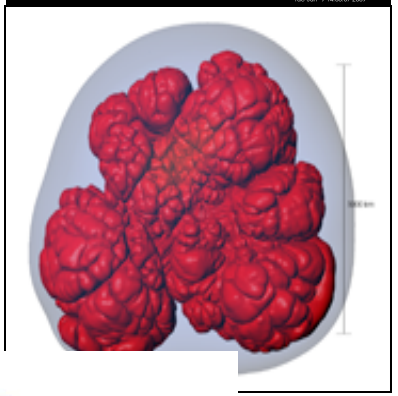
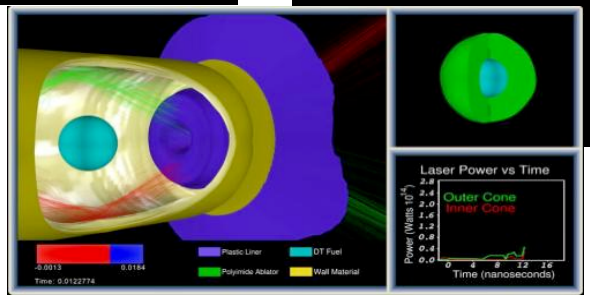
CIS 441/541: Intro to Computer Graphics
Lecture 2: The Scanline Algorithm

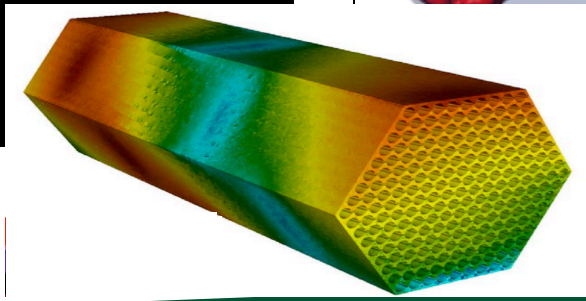












April 1, 2021

Hank Childs, University of Oregon

Duo Two-Step

UNIVERSITY OF OREGON

Office of the Provost

Dear Colleagues,

This spring, Duo two-step login becomes mandatory for all UO students. If you're teaching this term, I would appreciate it if you could help raise awareness of the Duo enrollment deadlines that start April 7.

Those deadlines will require students to register for Duo before they can access Canvas or Zoom. Students risk being late to remote classes on their assigned deadlines if they must stop and register a device on the spot. Preparing students now will ease their experience and reduce disruptions to your classes.

Projects

- 1A: due Saturday
- 1B: assigned today, due Wednesday (4/7)
- 1C: assigned Tuesday (4/6), due following Wednesday (4/14)

Office Hours This Week

- Hank: Thursday 1-2pm
  - Zoom link: check Canvas
- Abhishek: Fri 8-11, 2-5

Office Hours Weeks 2-10

- Abhishek:
  - Monday 10-11
  - Tuesday 945-1045 (partial overlap with 422)
  - Thursday 945-1045 (partial overlap with 422)
- Hank
  - Weds 230-330

Project #1A

Project #1A

- Goal: write a specific image
- Due: Sat April 3rd
- % of grade: 2%
- May be a little painful

CIS 441/541: Project #1A Due 11:59pm April 3rd, 2021
(which means 6am April 4th, 2021)

Worth 2% of your grade

Setup:
1) Download and install CMake. I recommend you use at least version 3.16.
2) Download, build, and install VTK. I strongly recommend you use version 8.2.0.
3) Make a directory called "project1A"
4) Download file project1A.cxx and CMakeLists.txt from class website and copy them into directory project1A.
5) Update the CMakeLists.txt file to point at the correct location for your VTK.
6) Run CMake. This will create build files.
7) Compile the project1A program. For Unix/Mac, this means "make"
8) Run the program.
9) It should output an image that is 1024x1024 called proj1A.png. The first pixel of the file should be red (although that might be hard to eyeball, especially if the corners of your window are curved, like with Macs).

Assignment:
1) You are to make an image that is 1024x1350. (1024 in width, 1350 in height)
    a. The image will be broken into 27 horizontal strips, with each strip consisting of 50 rows of pixels
2) The color for the Xth strip should be:
    a. X % 3 = 0 → B=0
    b. X % 3 = 1 → B=127
    c. X % 3 = 2 →B=255
    d. (X/3) % 3 = 0 → G = 0
    e. (X/3) % 3 = 1 → G=127
    f. (X/3) % 3 = 2 → G=255
    g. X/9 = 0 → R=0
    h. X/9 = 1 → R=127
    i. X/9 = 2 → R=255
3) 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=127
    c. Above that should be bright blue R=0, G=0, B=255
    d. Above that should be dark green, R=0, G=127, 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.

If your program produces the wrong output, you can receive no more half credit. (It is critical you check your work with the differencer).

What to submit: your source code (project1A.cxx)

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 top, left pixel
- P(M-1) is the top, right pixel
- P((MxN)-M) is the bottom, left pixel
- P(MxN-1) is the bottom, right pixel

Project #1A: background

- The red contributions are called the “red channel”
  - Ditto blue & green
- There are 3 channels in the image described above
- There is sometimes a fourth channel, called “alpha”
  - It is used for transparency
- → Images are either RGB or RGBA

Project #1A in a nutshell

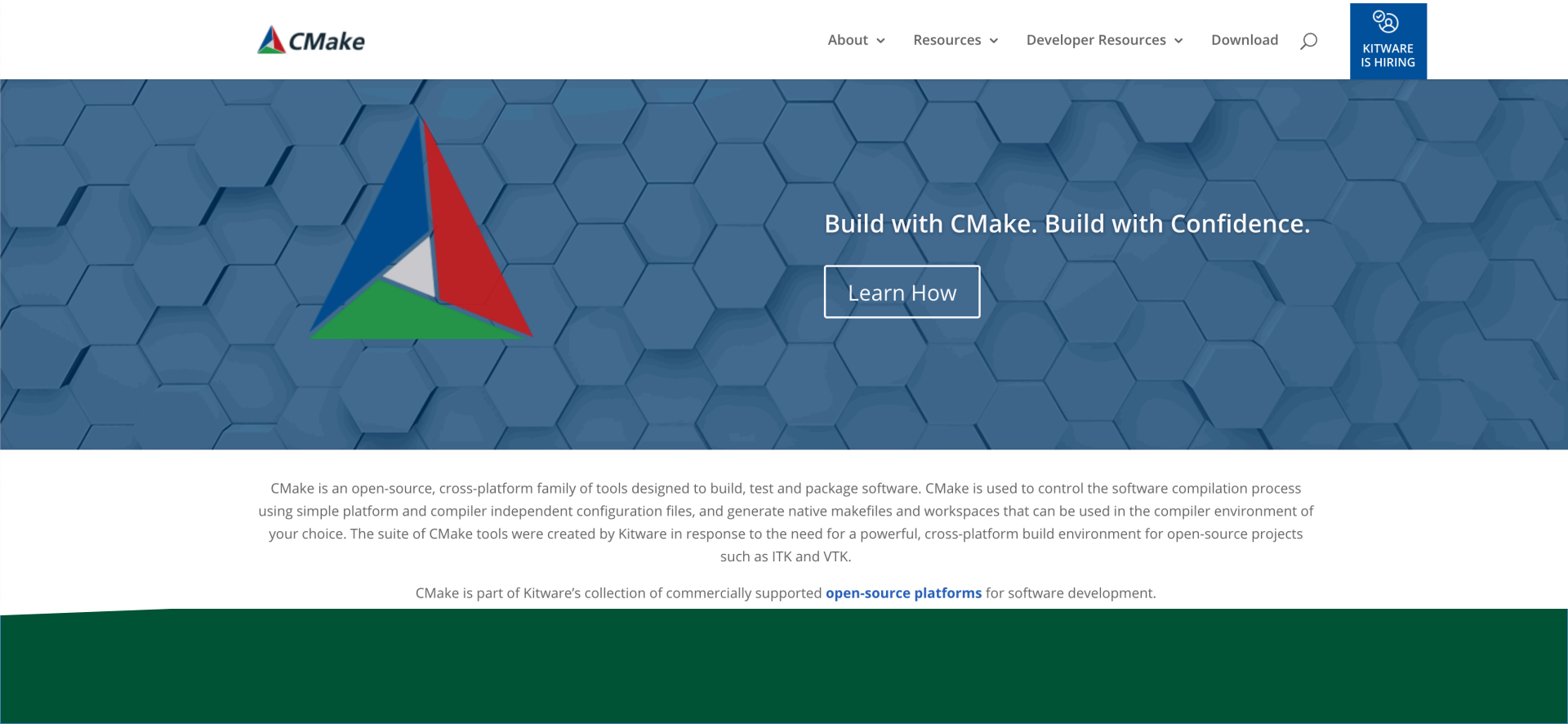
- Assignment:
  - Install CMake
  - Install VTK
  - Modify template program to output specific image

What is  ?

- Cmake is a cross-platform, open-source build system.
- CMake is a family of tools designed to build, test and package software.
- CMake is used to control the software compilation process using simple platform and compiler independent configuration files.
- CMake generates native makefiles and workspaces that can be used in the compiler environment of your choice.

How do you install CMake?

- Go to www.cmake.org & follow the directions

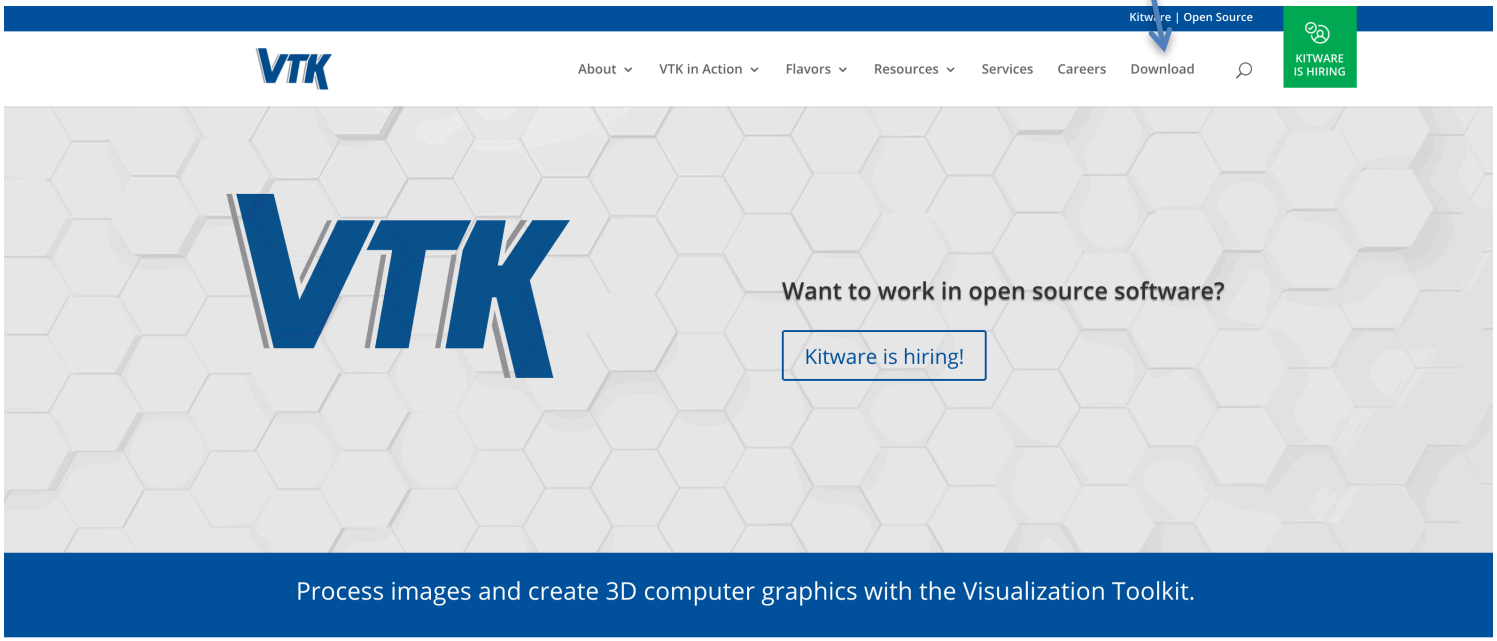


What is the Visualization Toolkit ?

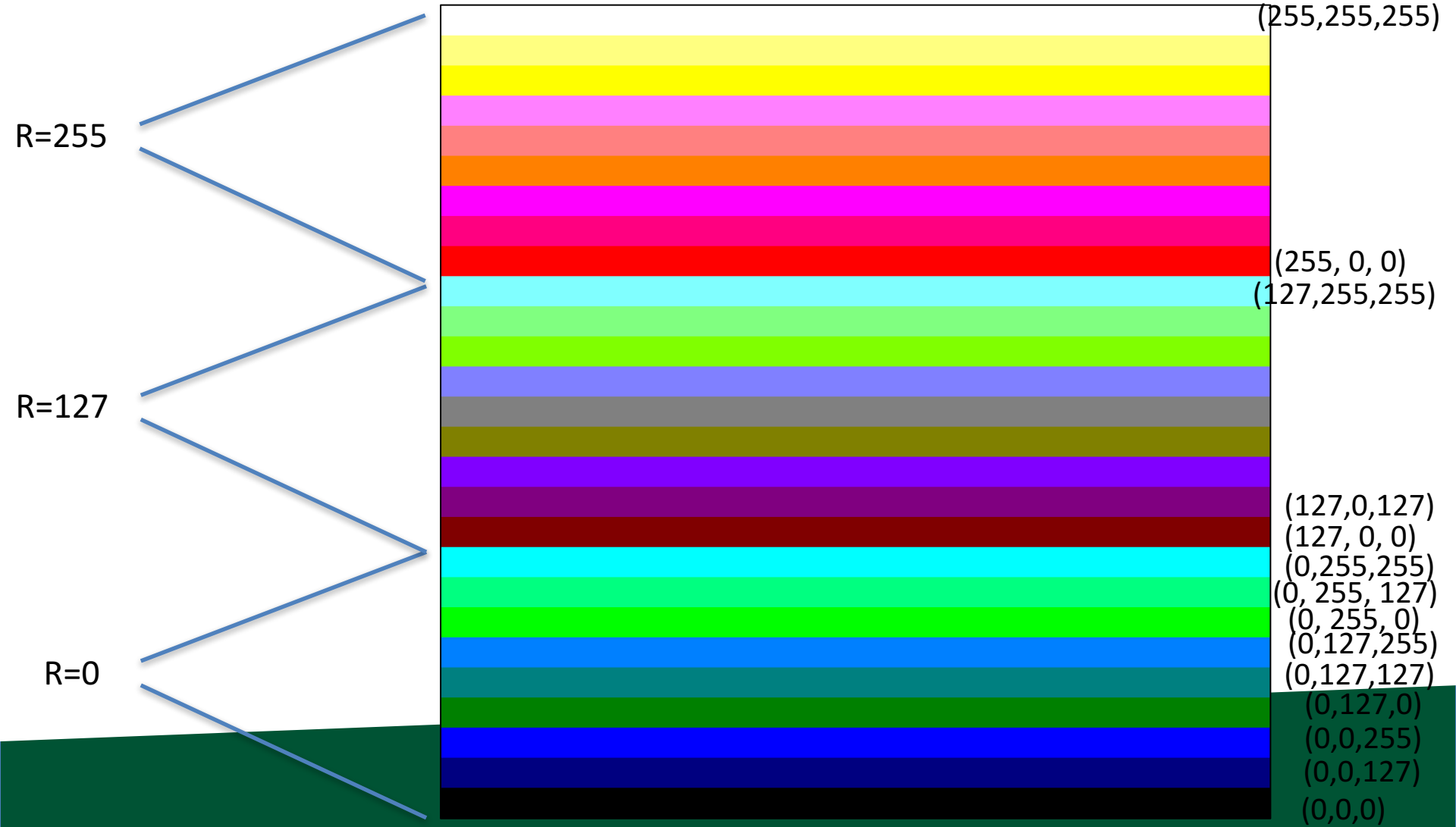
- The Visualization Toolkit (VTK) is an open-source, freely available software system for 3D computer graphics, image processing and visualization.
- VTK consists of a C++ class library and several interpreted interface layers including Tcl/Tk, Java, and Python.
- VTK is cross-platform and runs on Linux, Windows, Mac and Unix platforms.

How do you install VTK?

- Go to www.vtk.org , go to Download and follow the directions



What is the image I'm supposed to make?

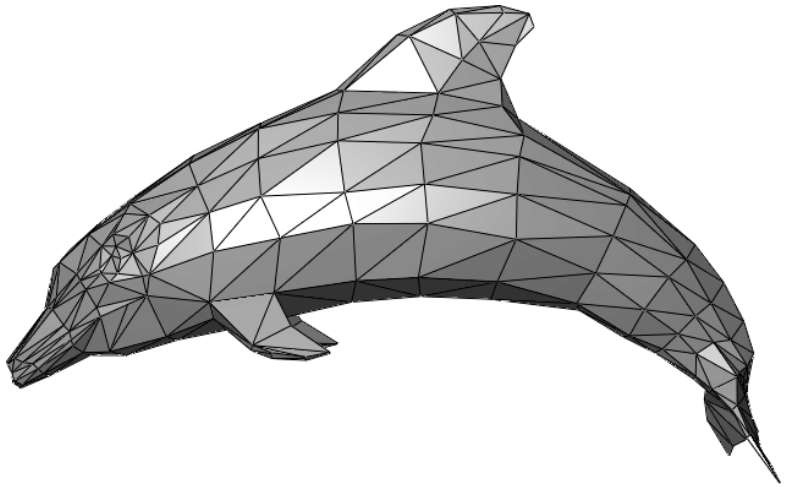


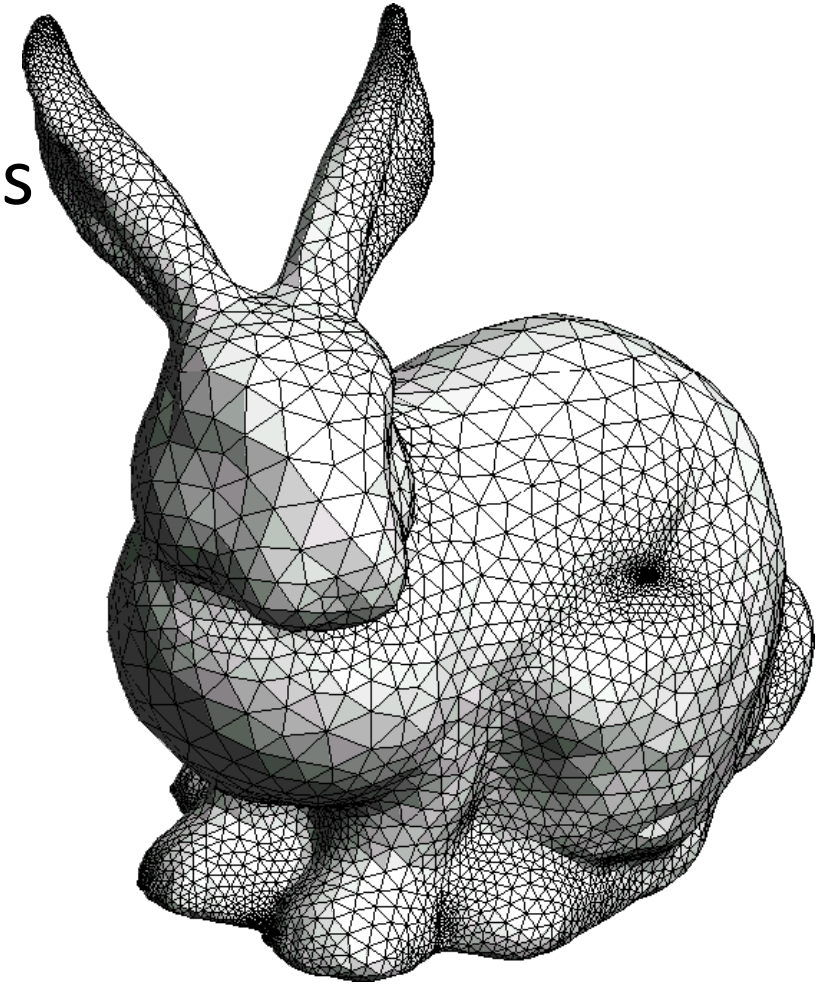
What do I do again?

- 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 to Canvas the source by Saturday midnight

Computer Graphics Models

- Usually made up of triangles
  - + tricks for shading





Picture credit: wikipedia, cs.mun.ca

The Scanline Algorithm

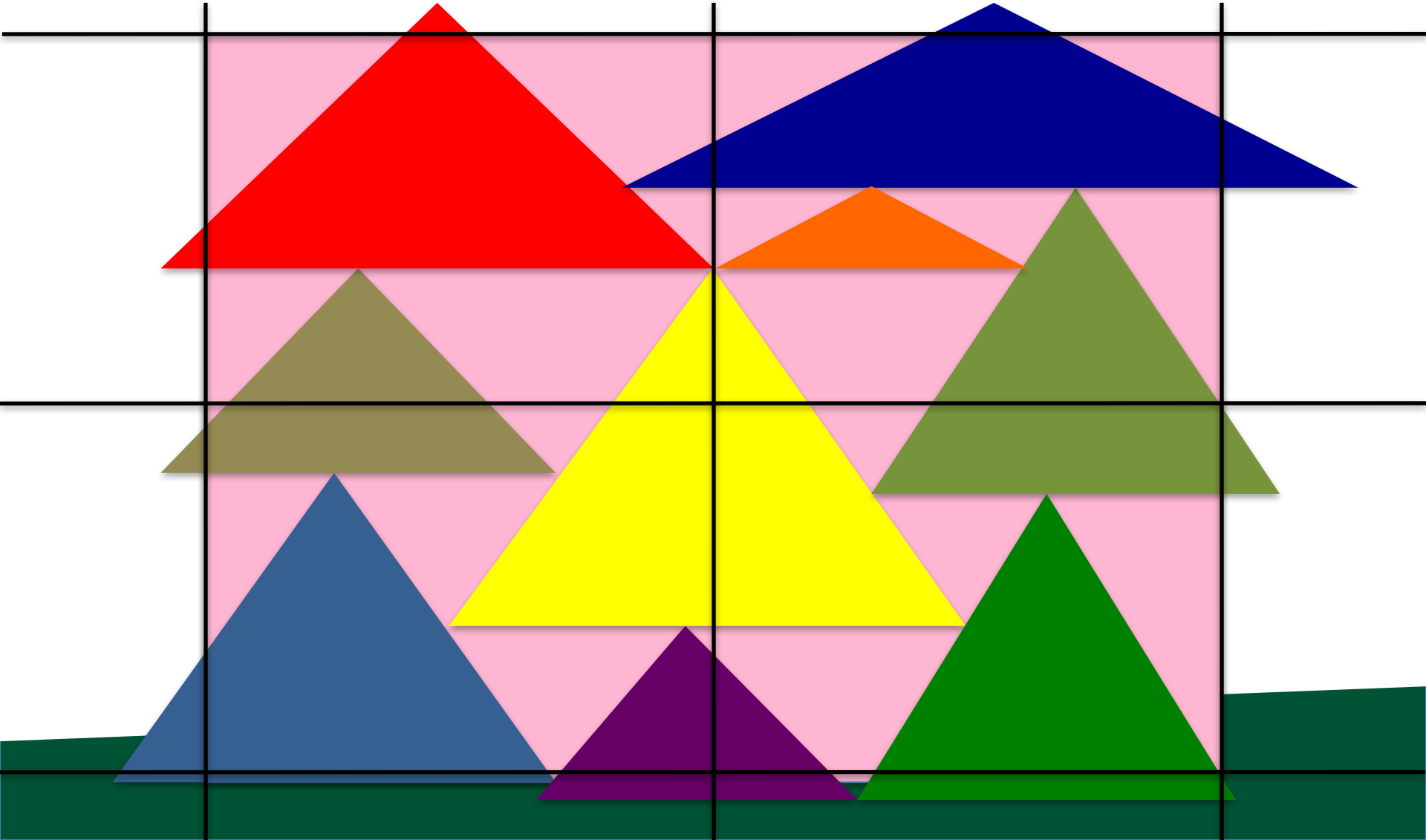
Reminder: ray-tracing vs rasterization

- Two basic ideas for rendering: rasterization and ray-tracing
- Ray-tracing: cast a ray for every pixel and see what geometry it intersects.
  - O(nPixels)
    - (actually, additional computational complexity for geometry searches)
  - Allows for beautiful rendering effects (reflections, etc)
  - Will discuss later in the quarter

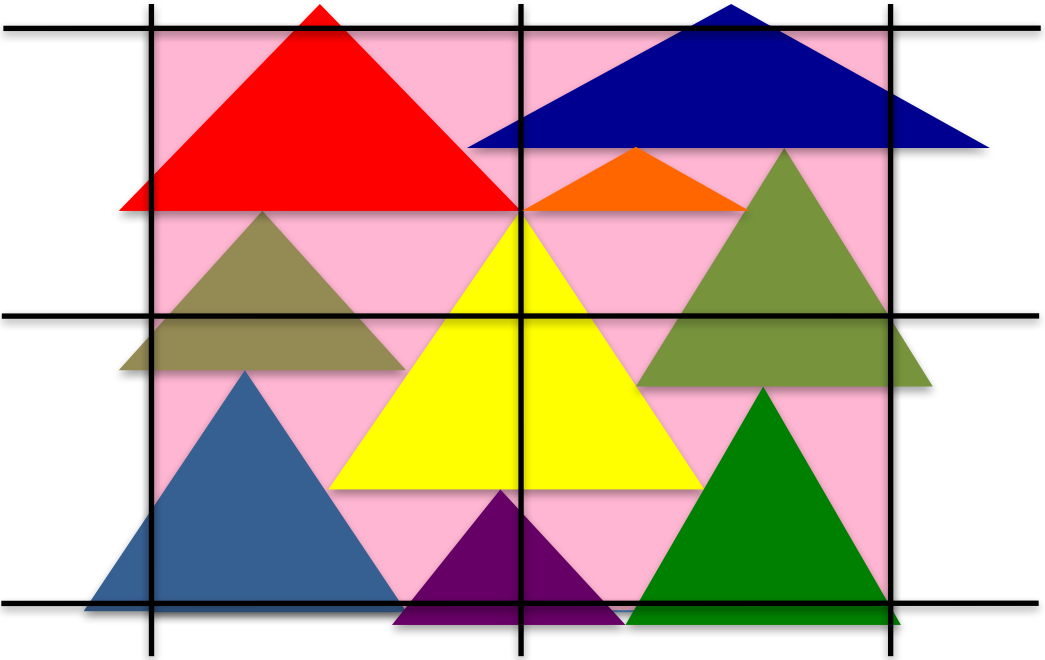
Reminder: ray-tracing vs rasterization

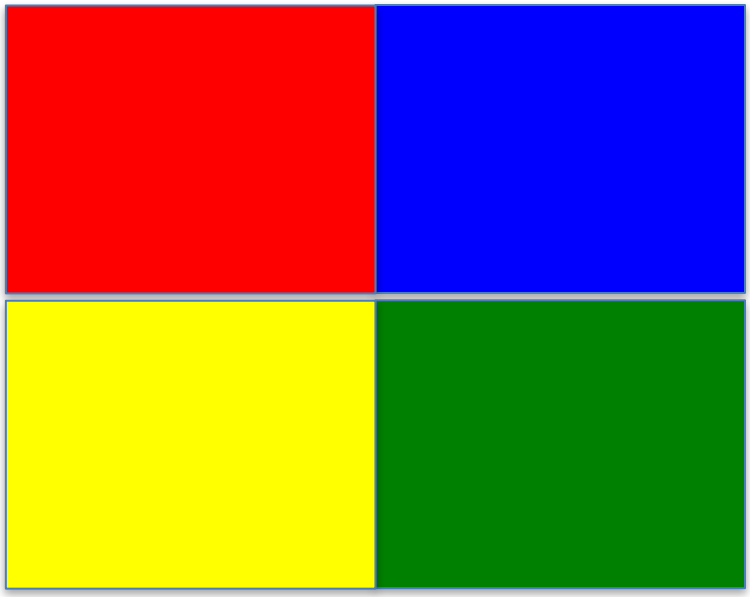
- Two basic ideas for rendering: rasterization and ray-tracing
- Rasterization: examine every triangle and see what pixels it covers.
  - O(nTriangles)
    - (actually, additional computational complexity for painting in pixels)
  - GPUs do rasterization very quickly
  - Our focus for the next 5 weeks

What color should we choose for each of these four pixels?



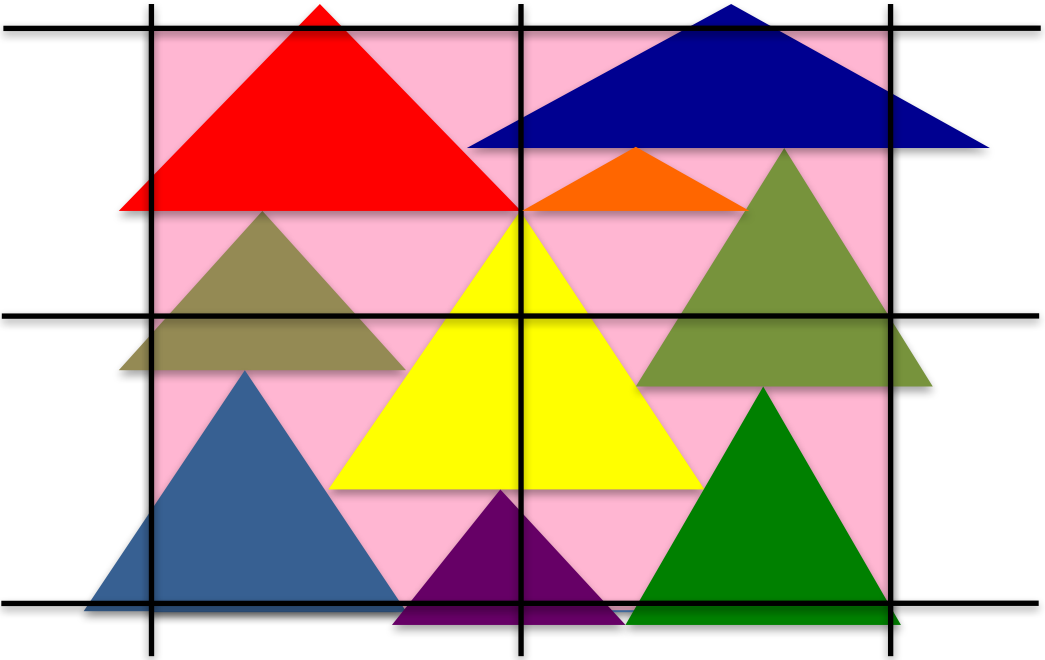
What color should we choose for each of these four pixels?

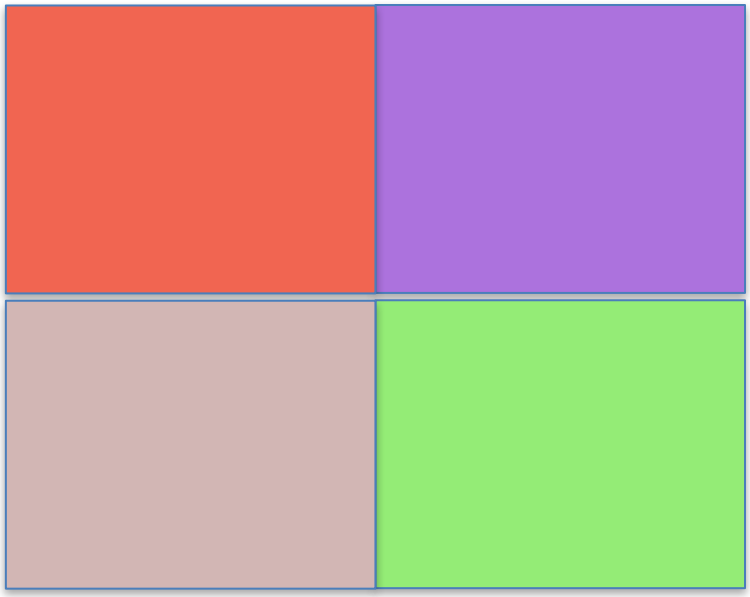




Most dominant triangle

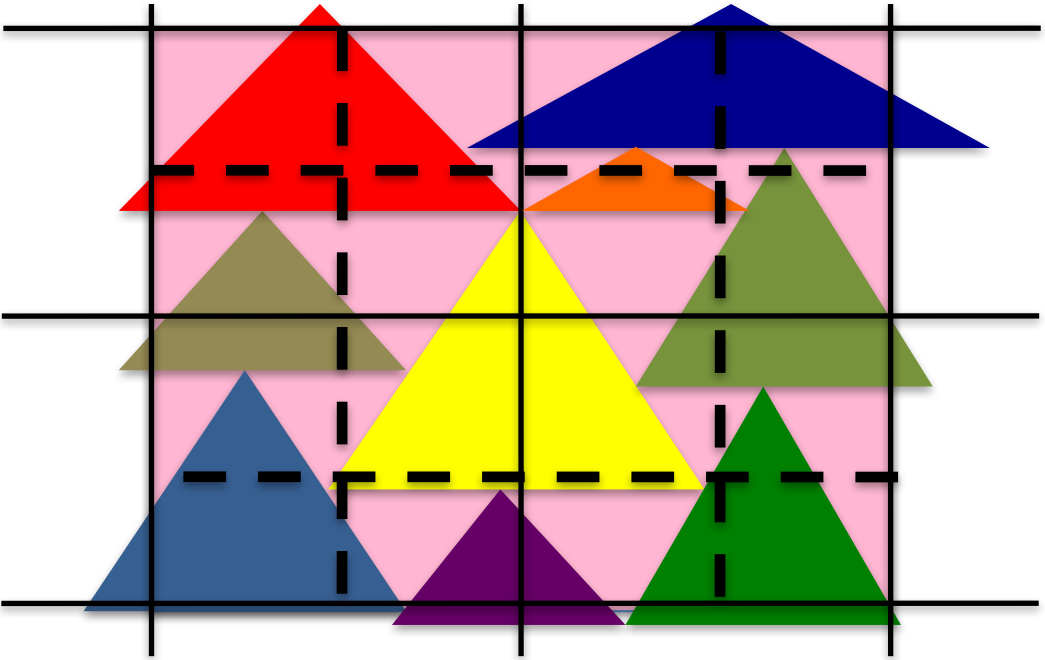
What color should we choose for each of these four pixels?

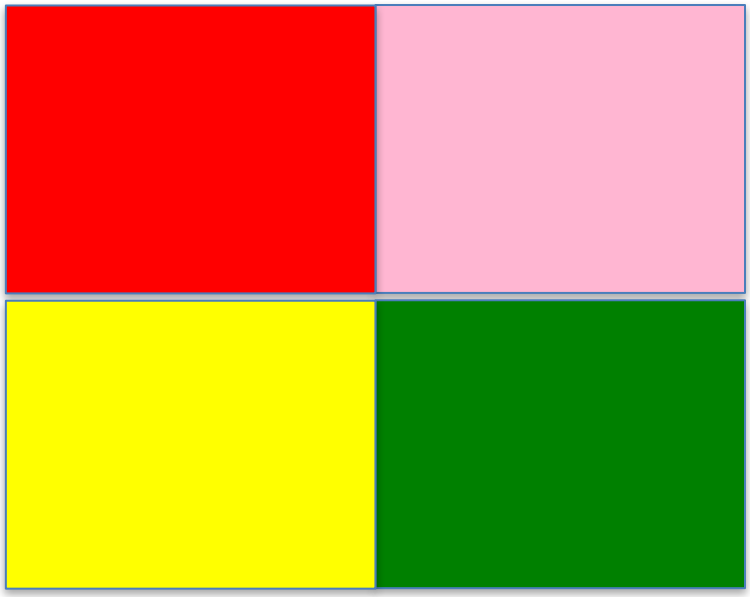




Average

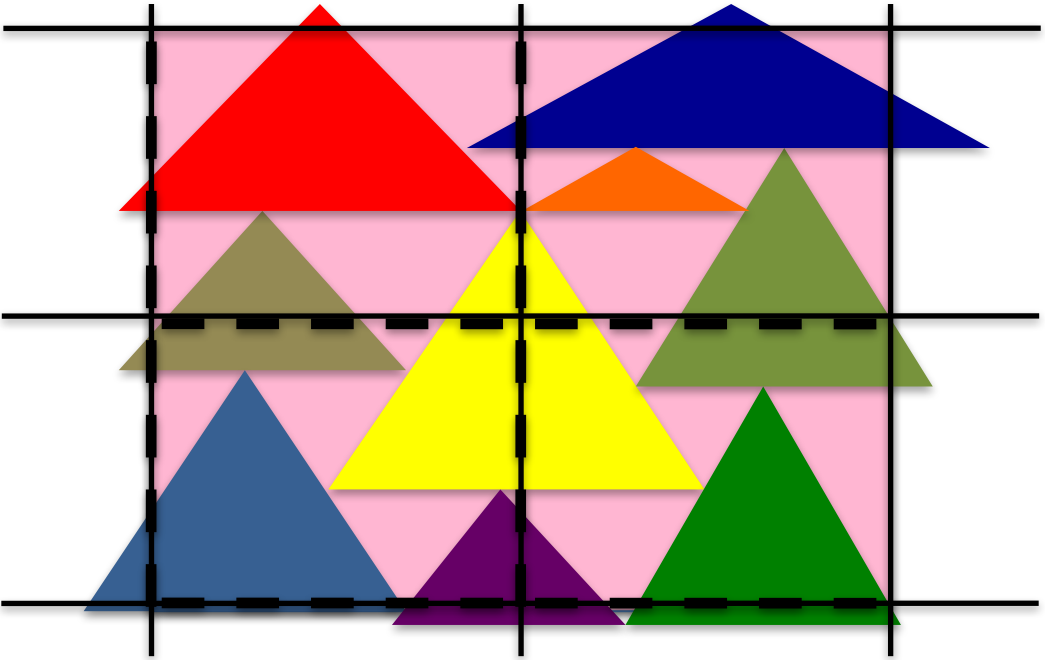
What color should we choose for each of these four pixels?

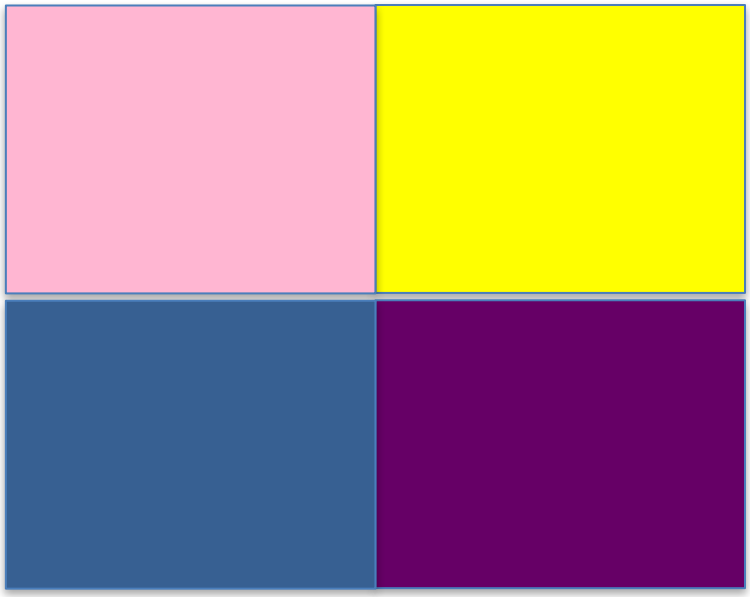




Pixel center

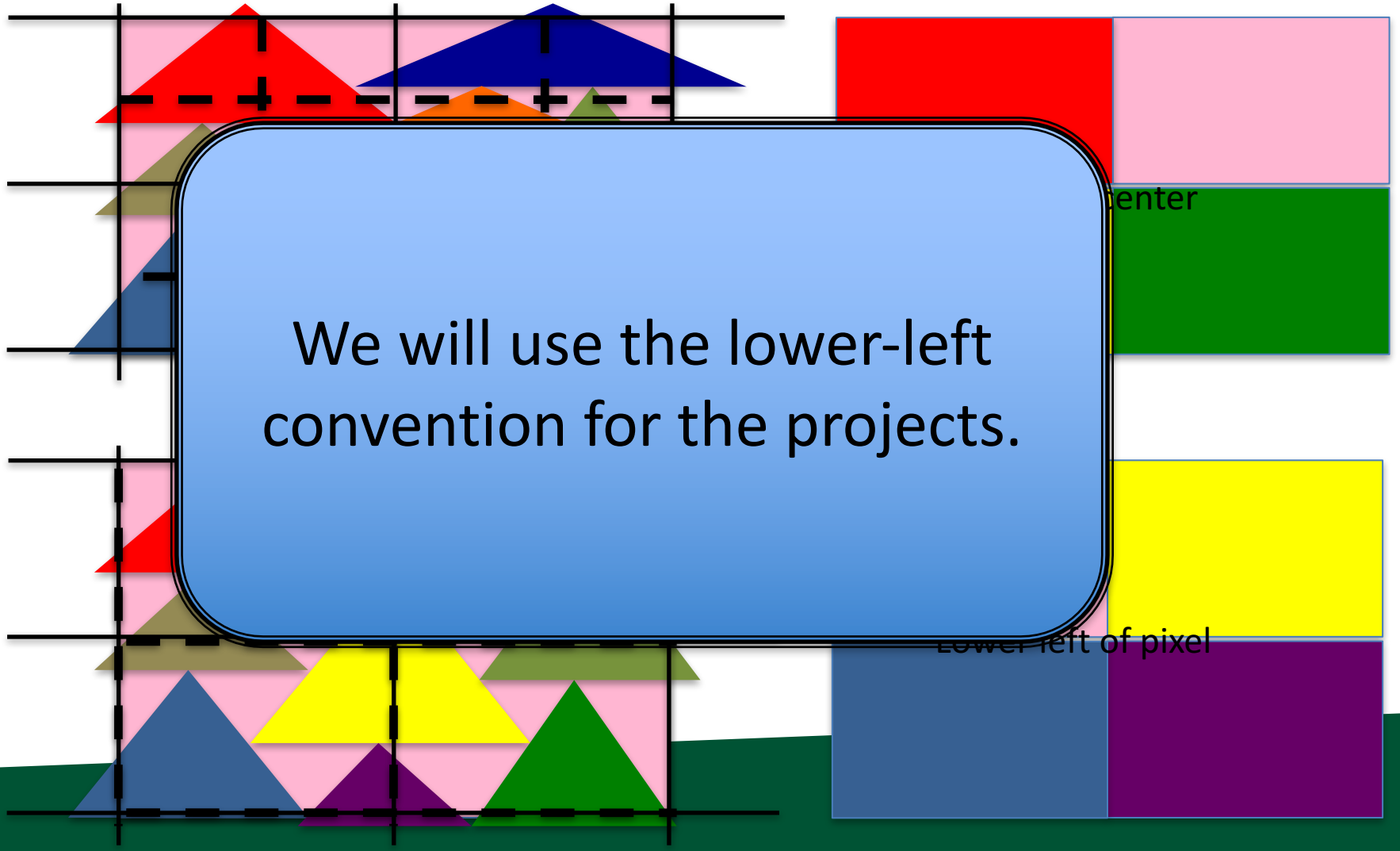
What color should we choose for each of these four pixels?





Lower left of pixel

The middle and lower-left variants are half-pixel translations of the other



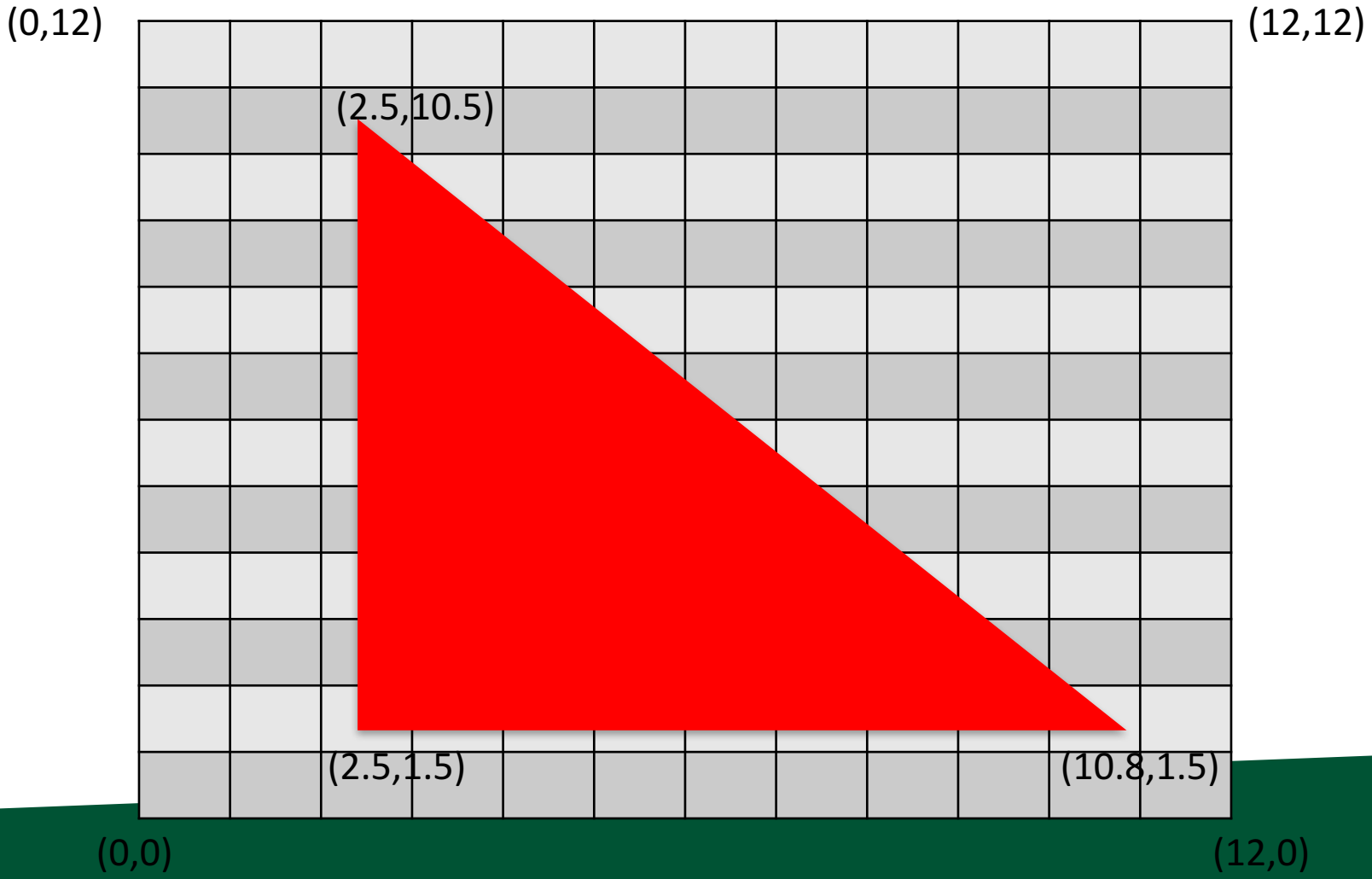
Where we are…

- 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:

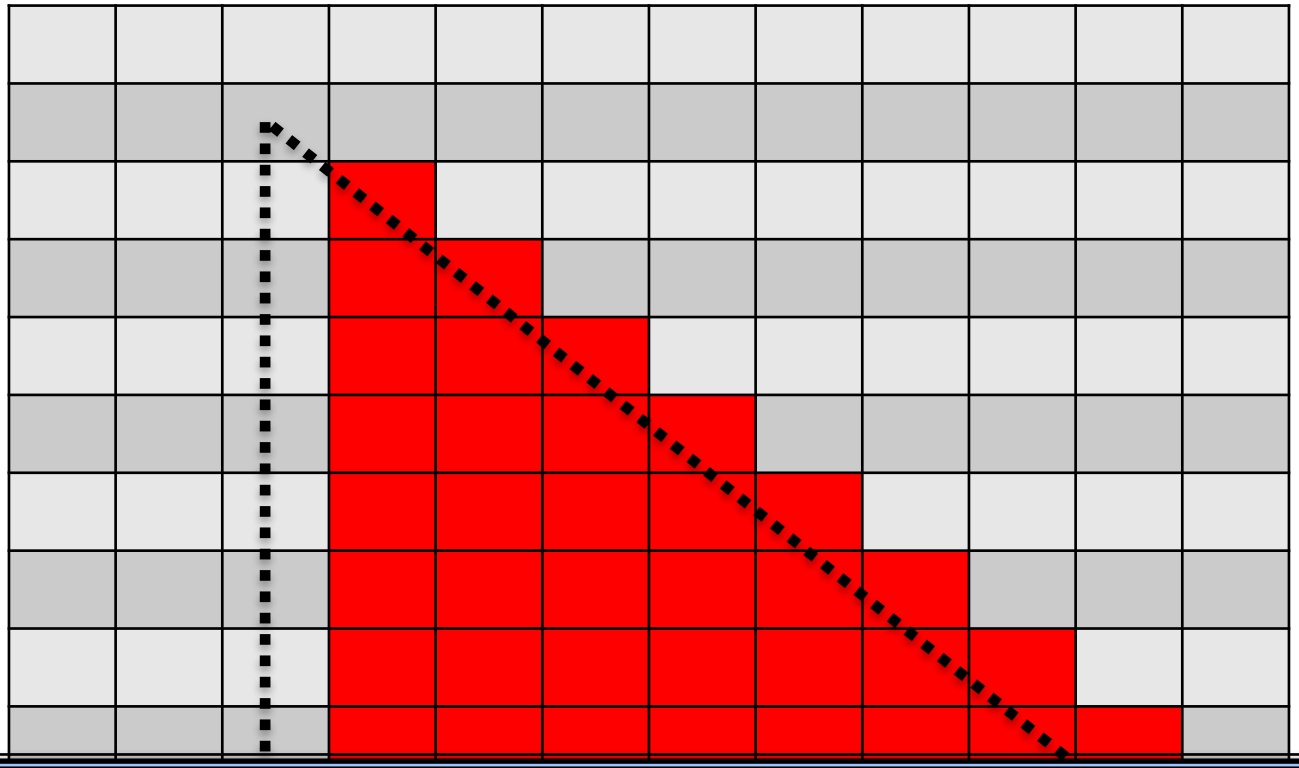
  How to deposit triangle colors onto an image?

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

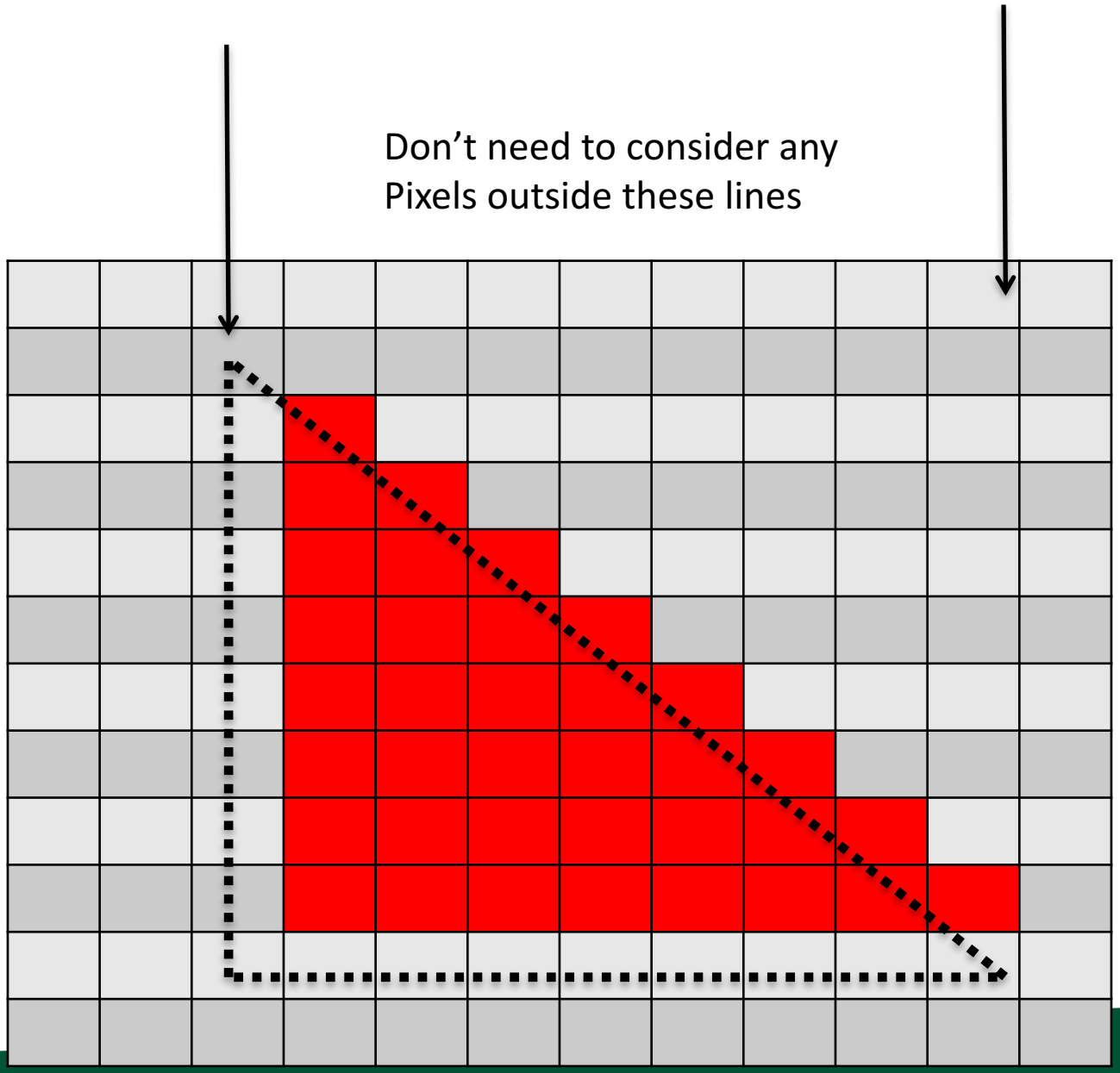

(0,12)
(12,12)
(2.5,10.5)
(2.5,1.5)
(10.8,1.5)
(0,0)
(12,0)

Our desired output

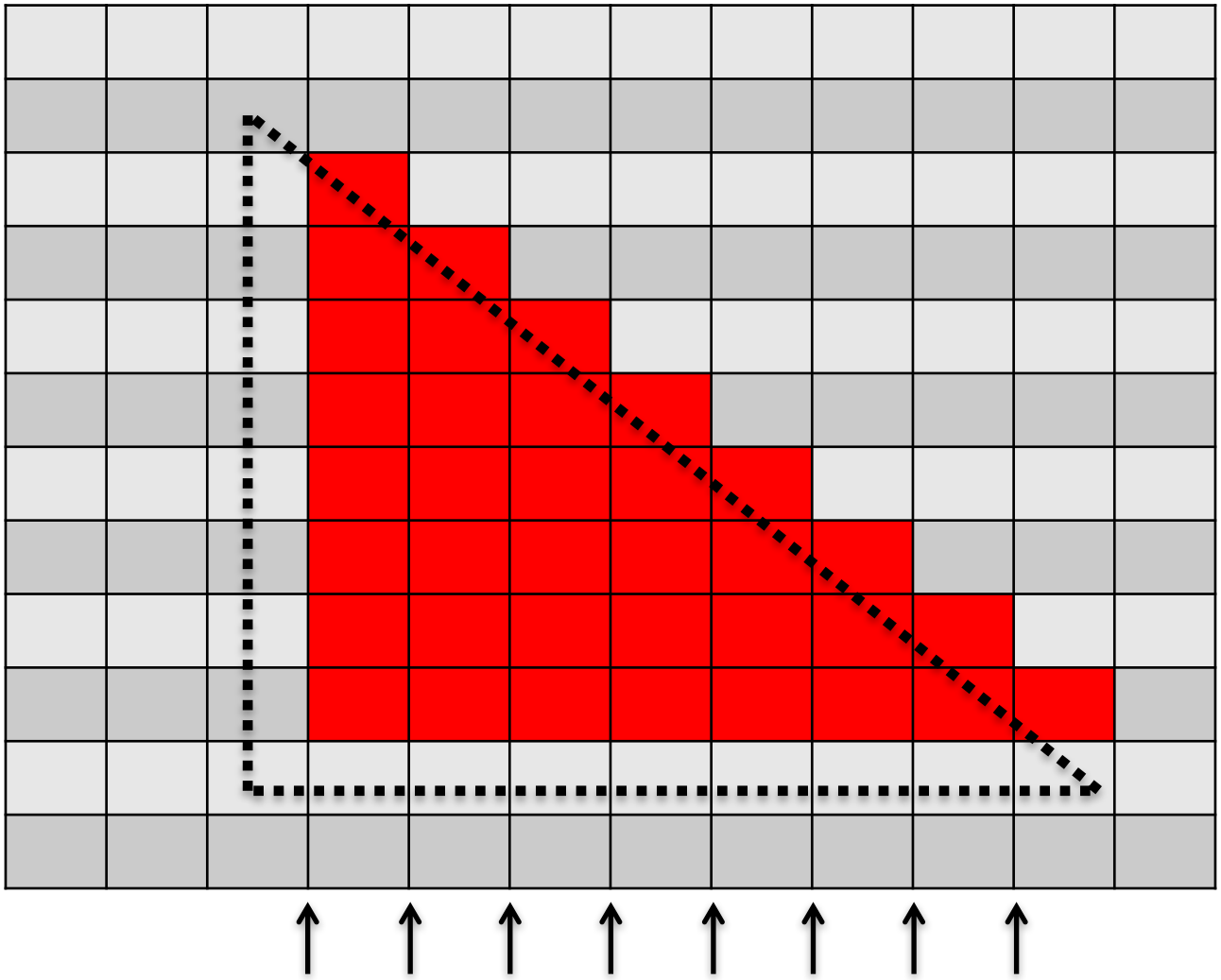


How do we make this output? Efficiently?

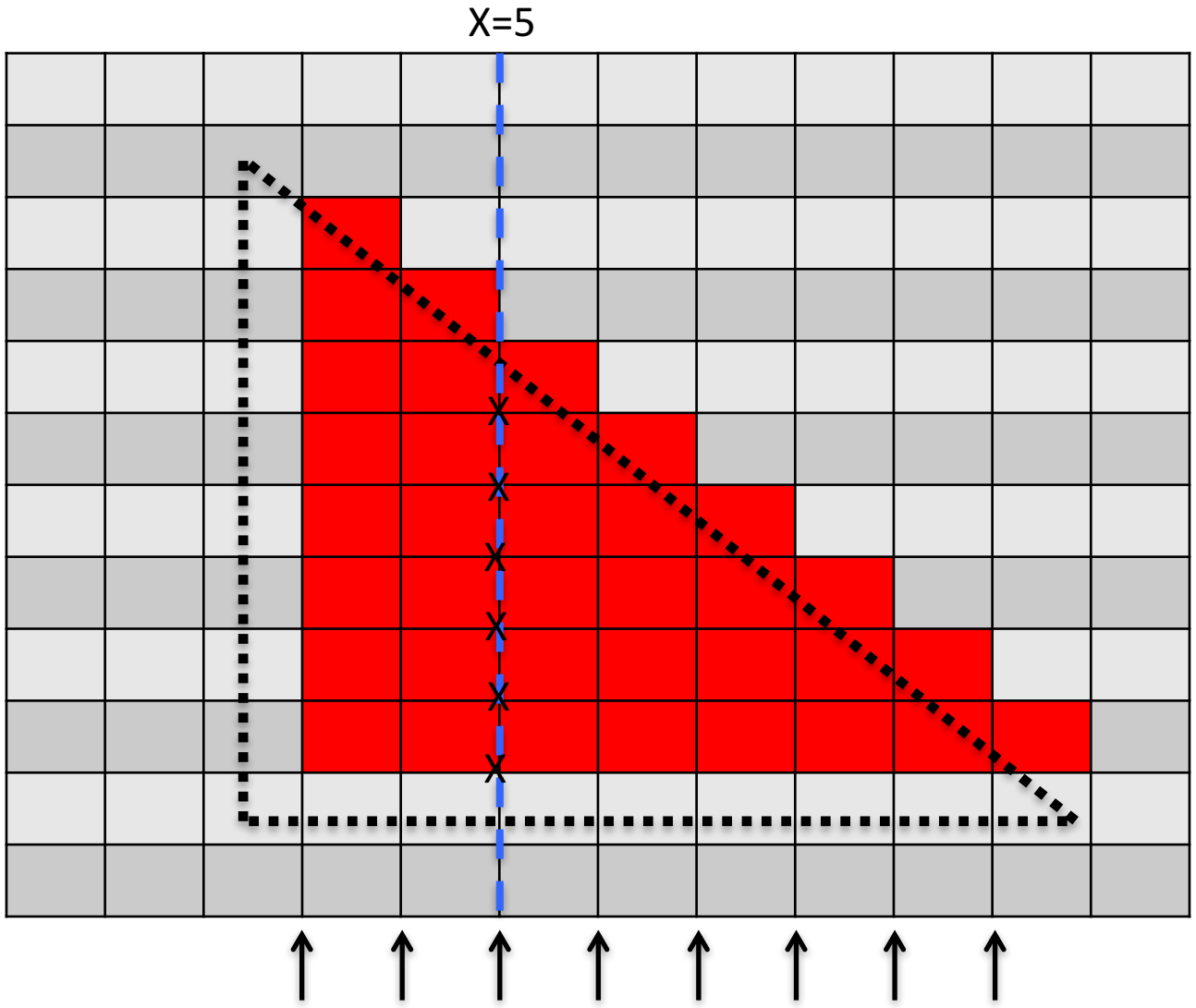
Don’t need to consider any
Pixels outside these lines



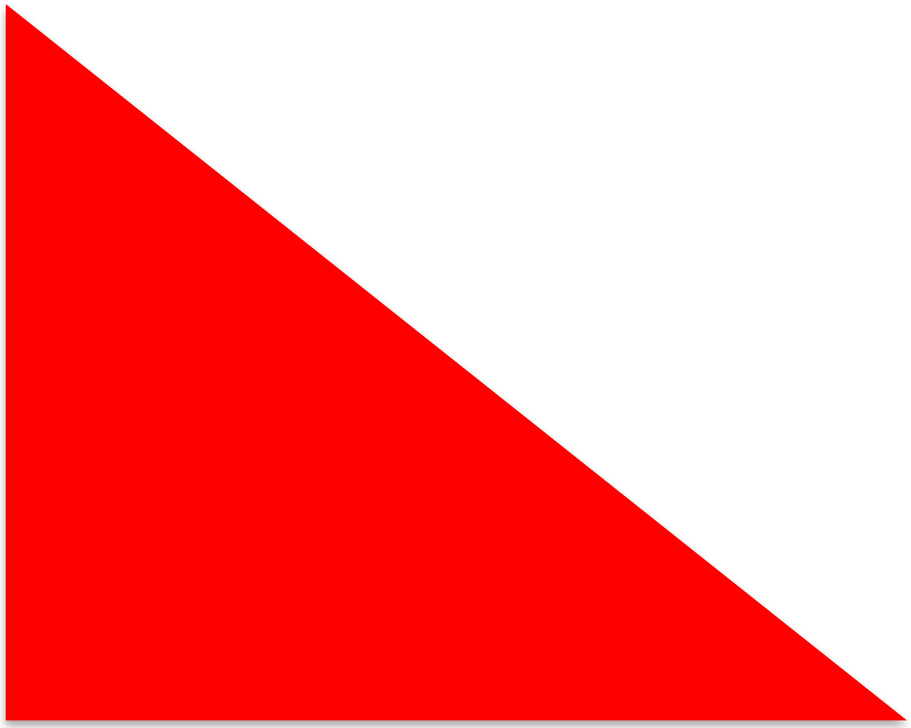
Scanline algorithm: consider all ~~rows~~ columns that can possibly overlap



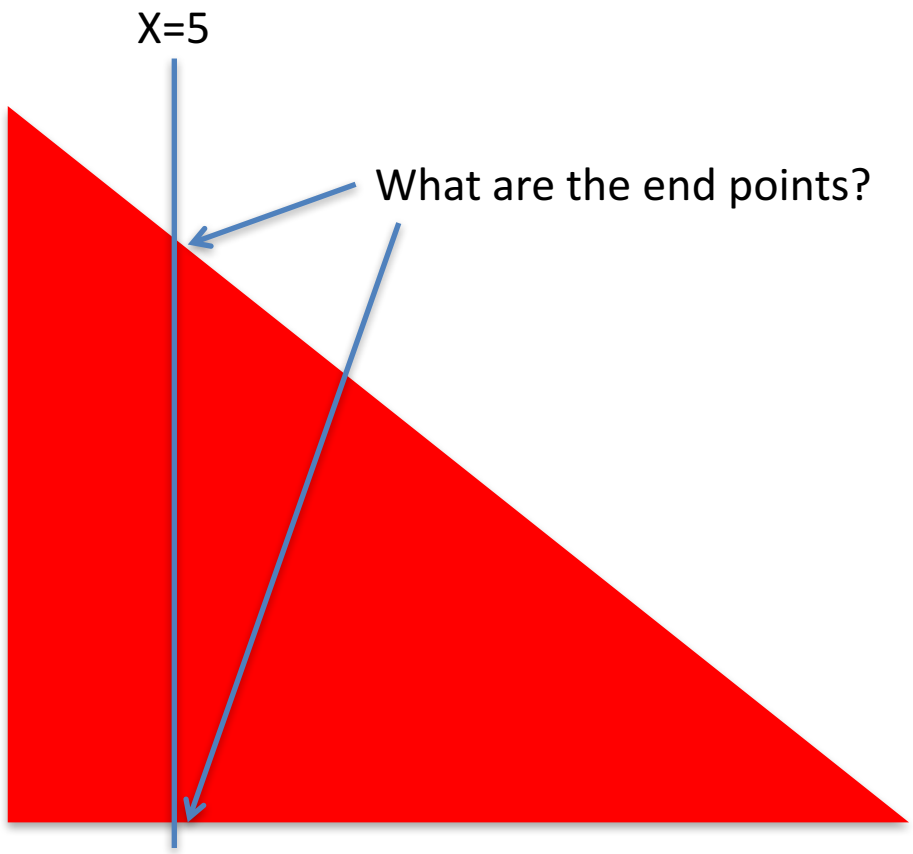
We will extract a “scanline,” i.e., calculate the intersections for one column of pixels



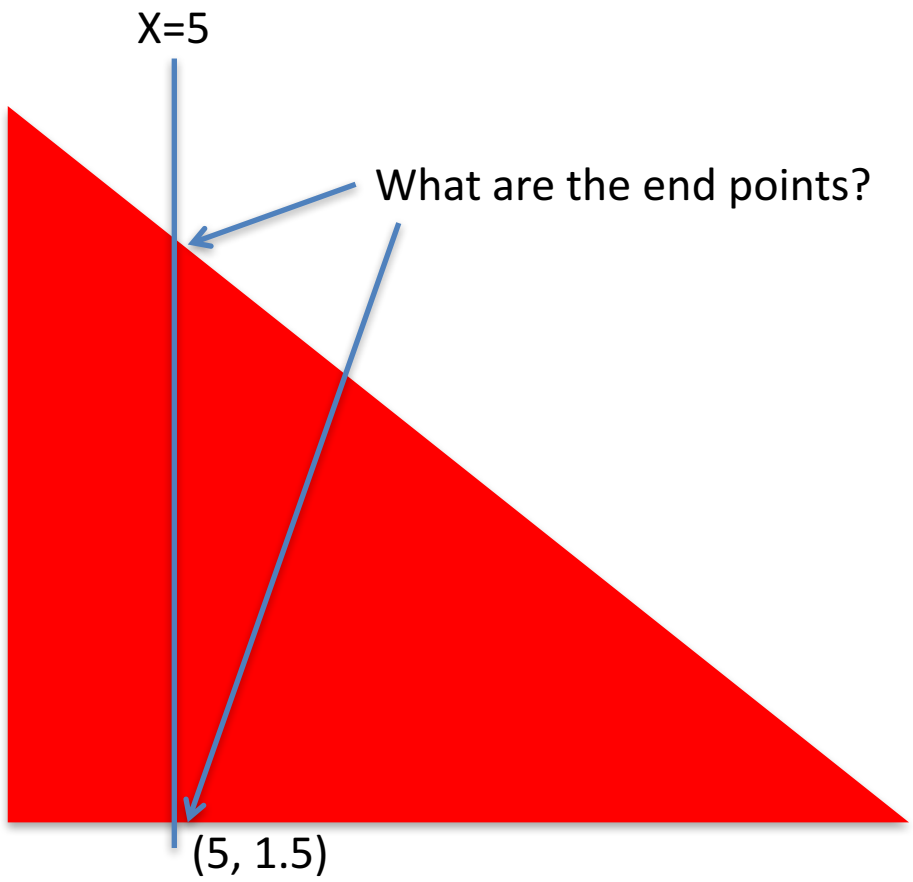
- Red triangle
  - Vertex 1: (2.5, 1.5)
  - Vertex 2: (2.5, 10.5)
  - Vertex 3: (10.5, 1.5)



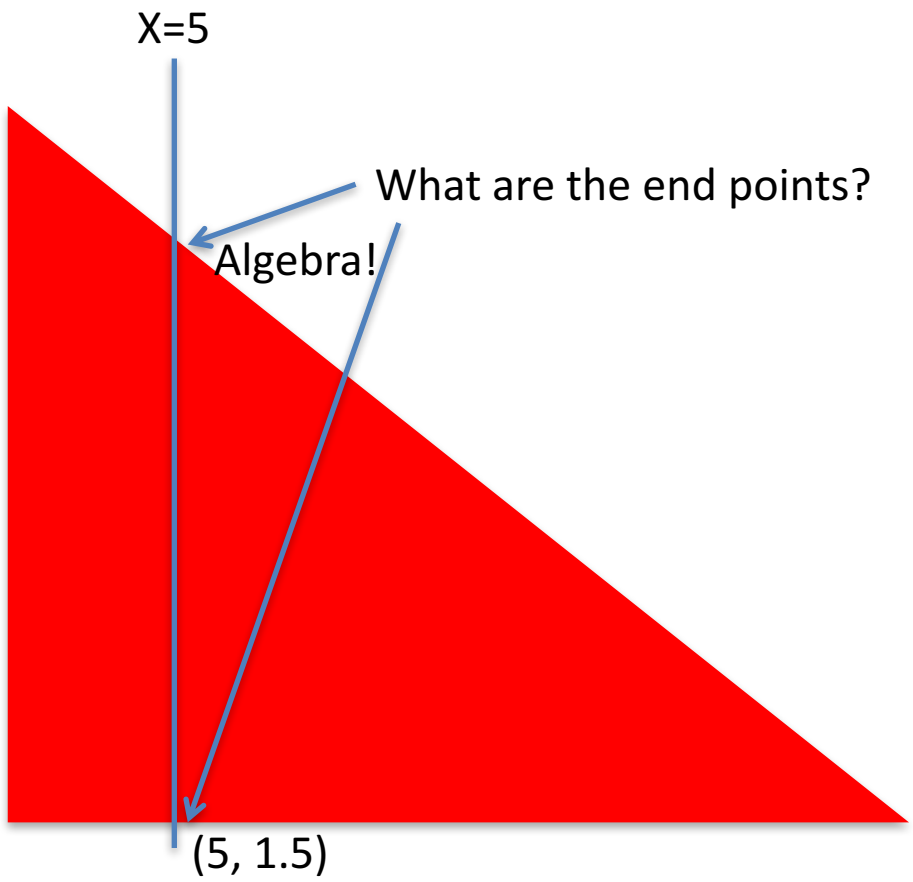
- Red triangle
  - Vertex 1: (2.5, 1.5)
  - Vertex 2: (2.5, 10.5)
  - Vertex 3: (10.5, 1.5)



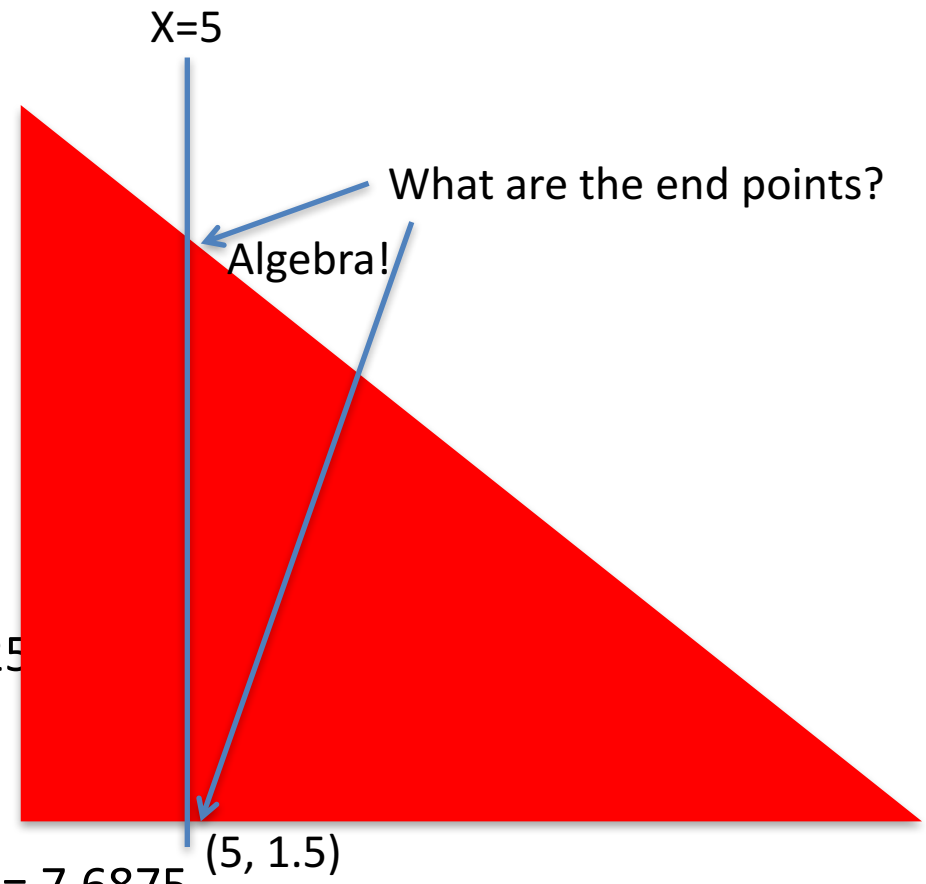
- Red triangle
  - Vertex 1: (2.5, 1.5)
  - Vertex 2: (2.5, 10.5)
  - Vertex 3: (10.5, 1.5)



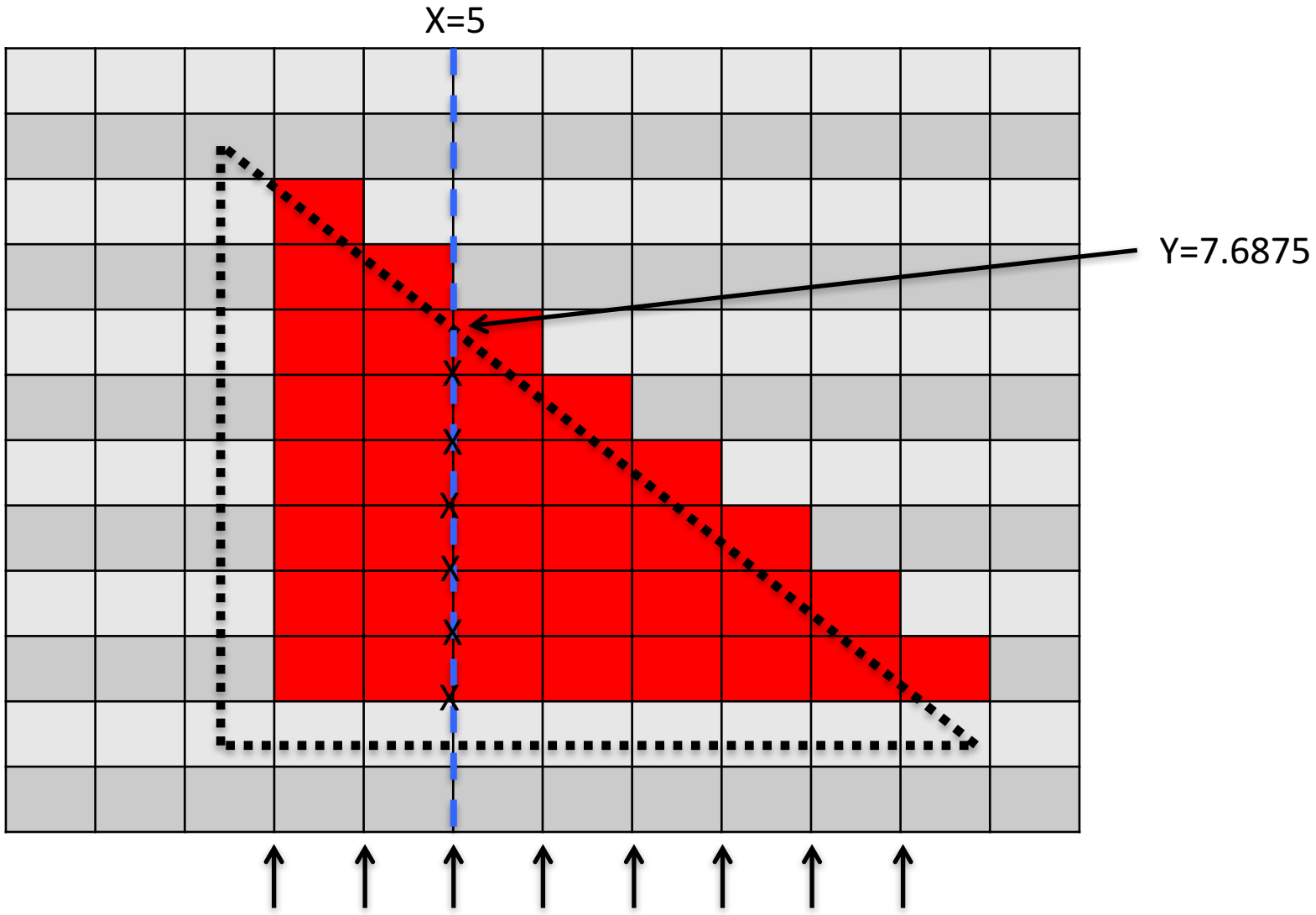
- Red triangle
  - Vertex 1: (2.5, 1.5)
  - Vertex 2: (2.5, 10.5)
  - Vertex 3: (10.5, 1.5)



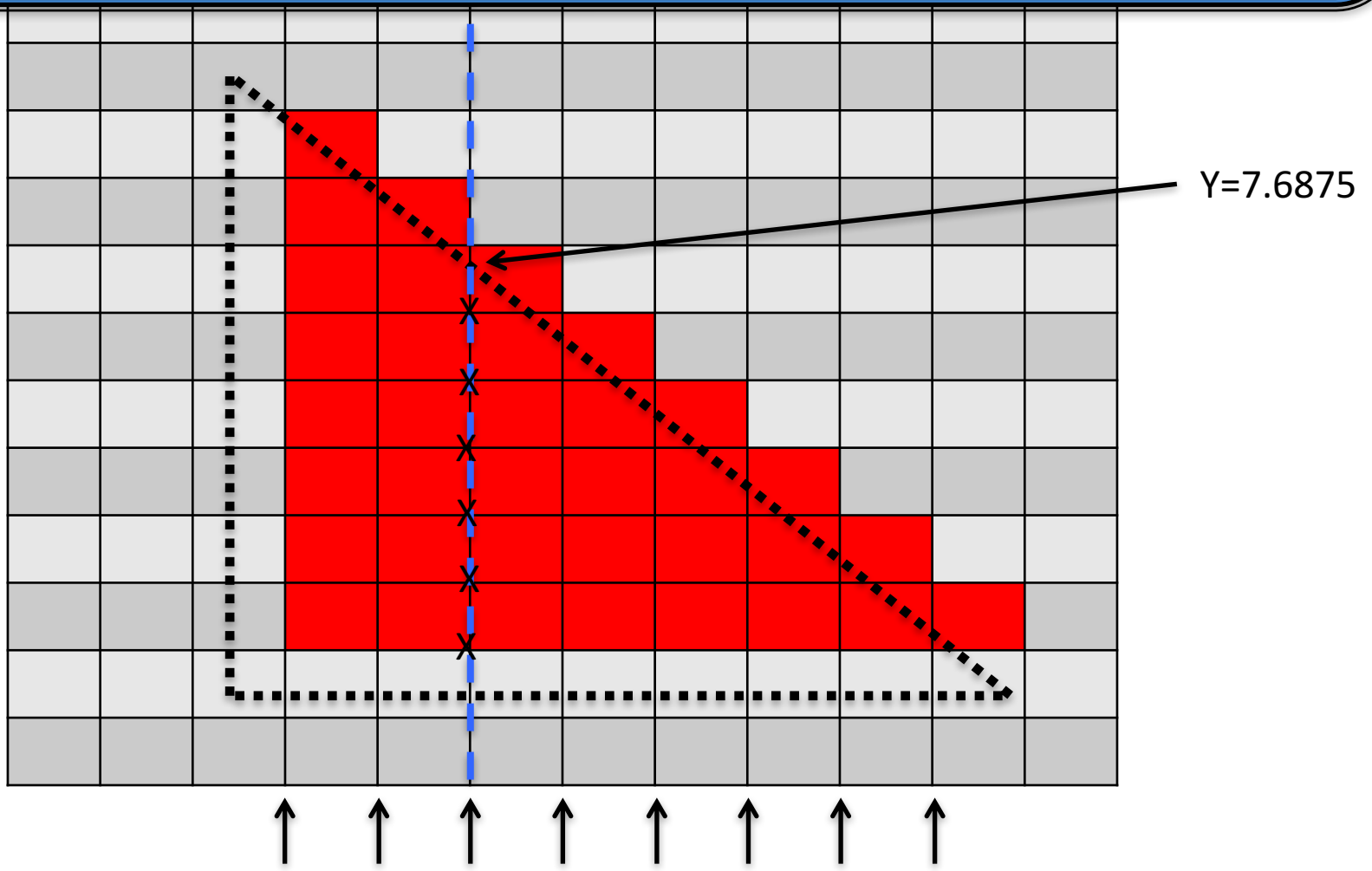
- Red triangle
  - Vertex 1: (2.5, 1.5)
  - Vertex 2: (2.5, 10.5)
  - Vertex 3: (10.5, 1.5)
- y = mx+b
- m=(1.5-10.5)/(10.5-2.5) = -9/8
- b=y-mx = 1.5-10.5*-9/8 = 13.3125
- →
- y = -9/8x + 13.3125
- @ X==5 → y = -9/8*5 + 13.3125 = 7.6875



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



If (r, c) is pixel at row r and column c, then scanline for X=5 leads to colors deposited at:
(2,5), (3,5), (4,5), (5,5), (6, 5), (7, 5)



Scanline algorithm for one triangle

- Determine columns of pixels the triangle can possibly intersect
  - Call them columnMin to columnMax
    - columnMin: ceiling of smallest X value
    - columnMax: floor of biggest X value
- For c in [columnMin → columnMax] ; do
  - Find end points of c intersected with triangle
    - Call them bottomEnd and topEnd
  - For r in [ceiling(bottomEnd) → floor(topEnd)] ; do
    - ImageColor(r, c) ← triangle color

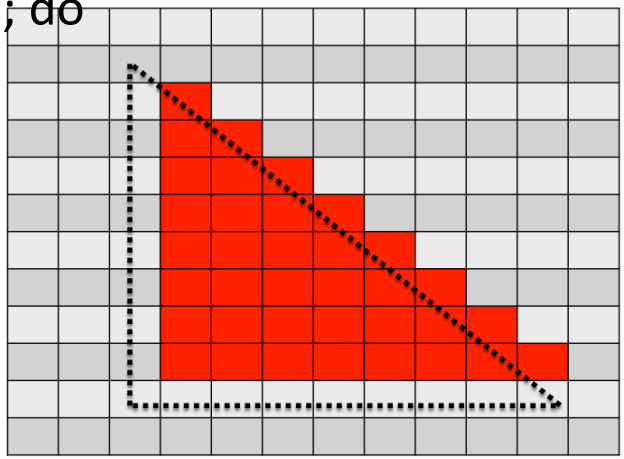
Scanline algorithm

- Determine columns of pixels triangles can possibly intersect
  - Call them columnMin to columnMax
    - columnMin: ceiling of smallest X value
    - columnMax: floor of biggest X value
- For c in [columnMin → columnMax] ; do
  - Find end points of c intersected with triangle
    - Call them bottomEnd and topEnd
  - For r in [ceiling(bottomEnd) → floor(topEnd)] ; do
    - ImageColor(r, c) ← triangle color

Y values from 2.5 to 10.5 means rows 3 through 10

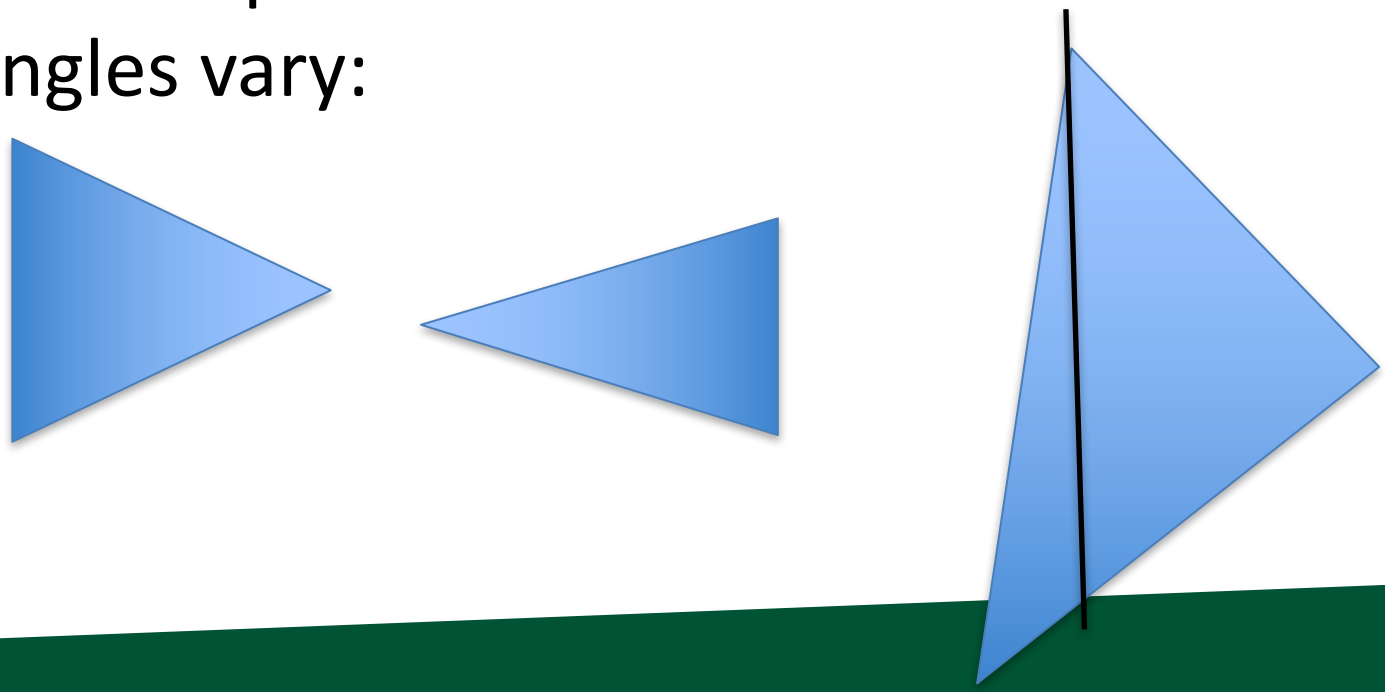
For c==5, bottomEnd = 1.5, topEnd = 7.6875

For c = 5, we call ImageColor with (2,5), (3,5), (4,5), (5,5), (6, 5), (7, 5)

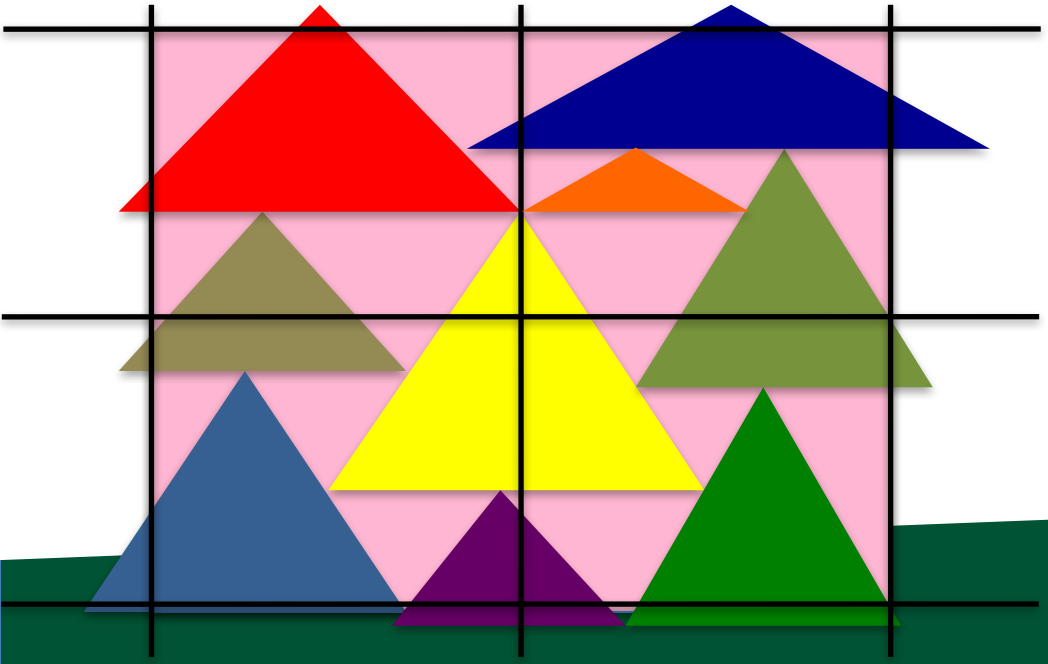


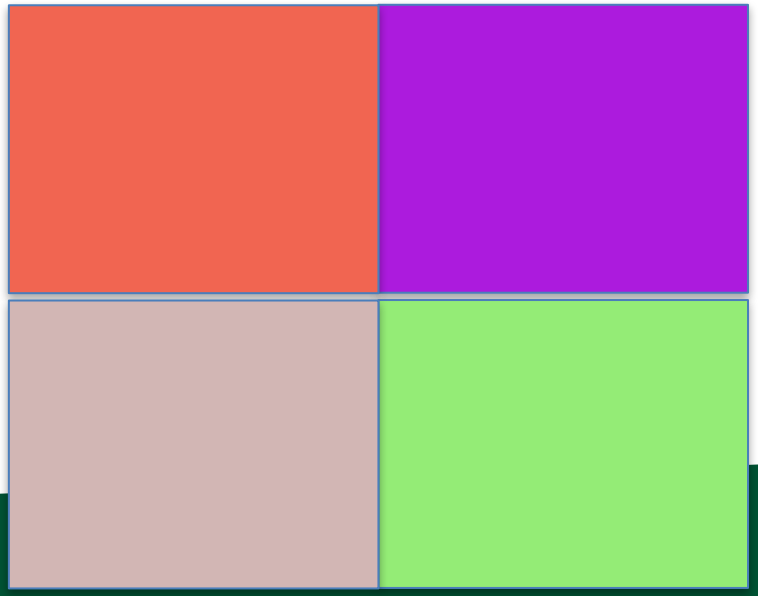
Arbitrary Triangles

- 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:

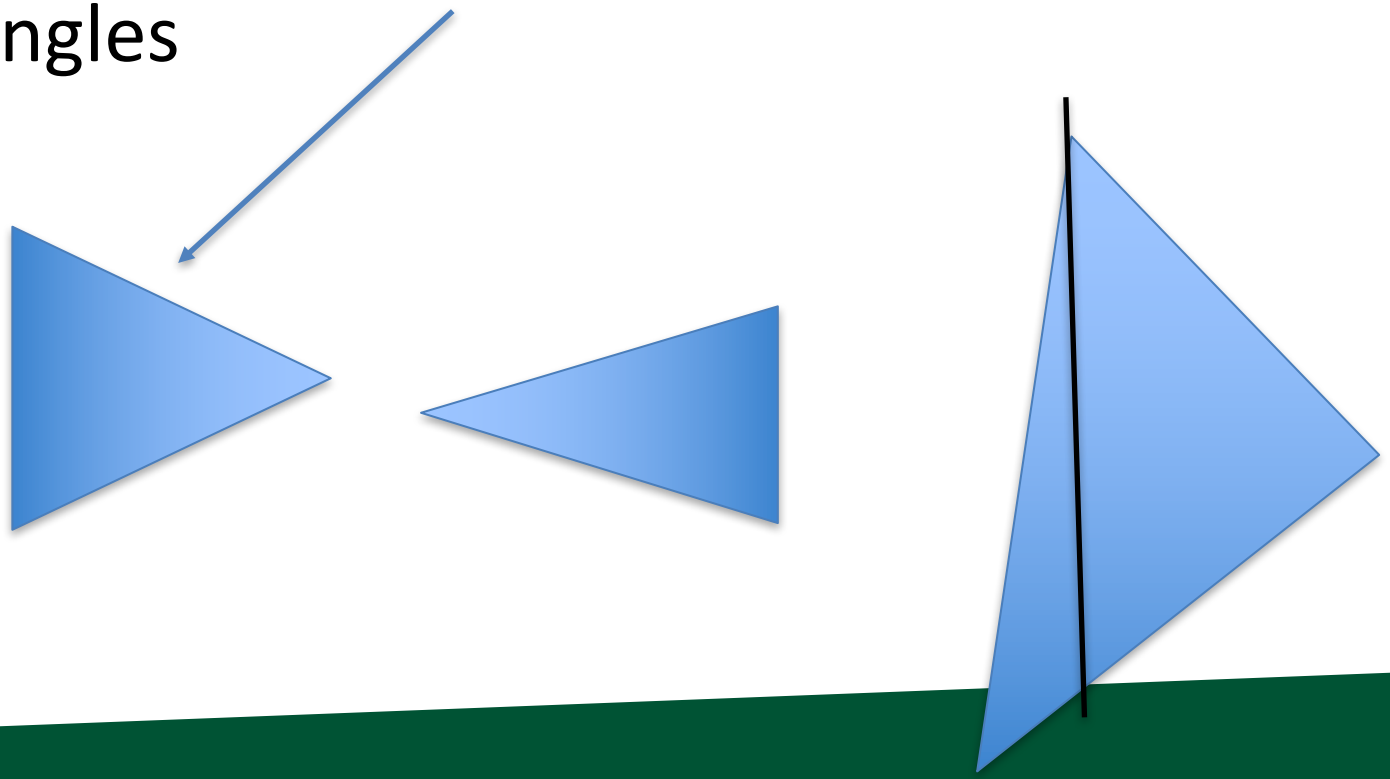
  How to deposit triangle colors onto an image?

  Still don't know how to:
  1) Vary colors (easy)
  2) Deal with triangles that overlap

Project 1B

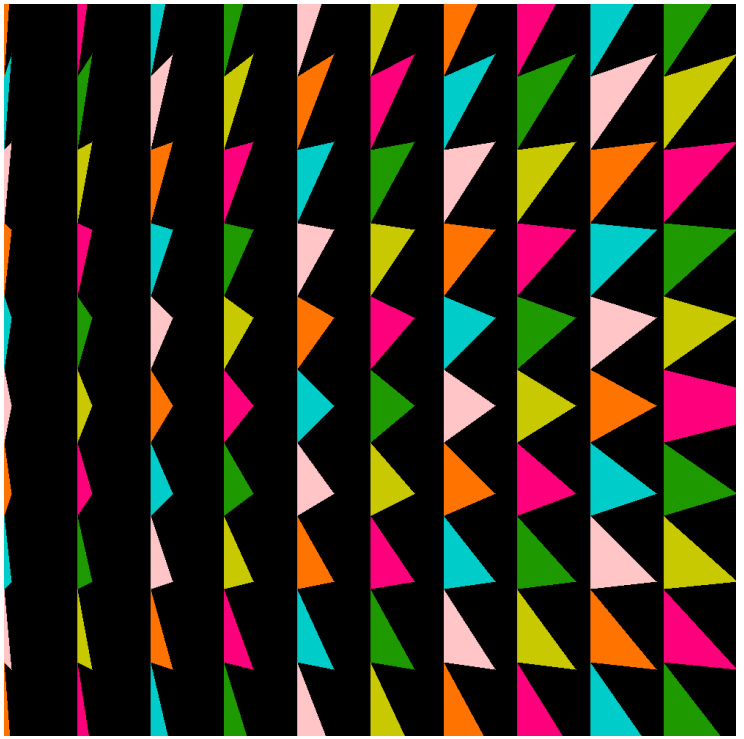
Arbitrary Triangles

- You will implement the scanline algorithm for “going right” triangles



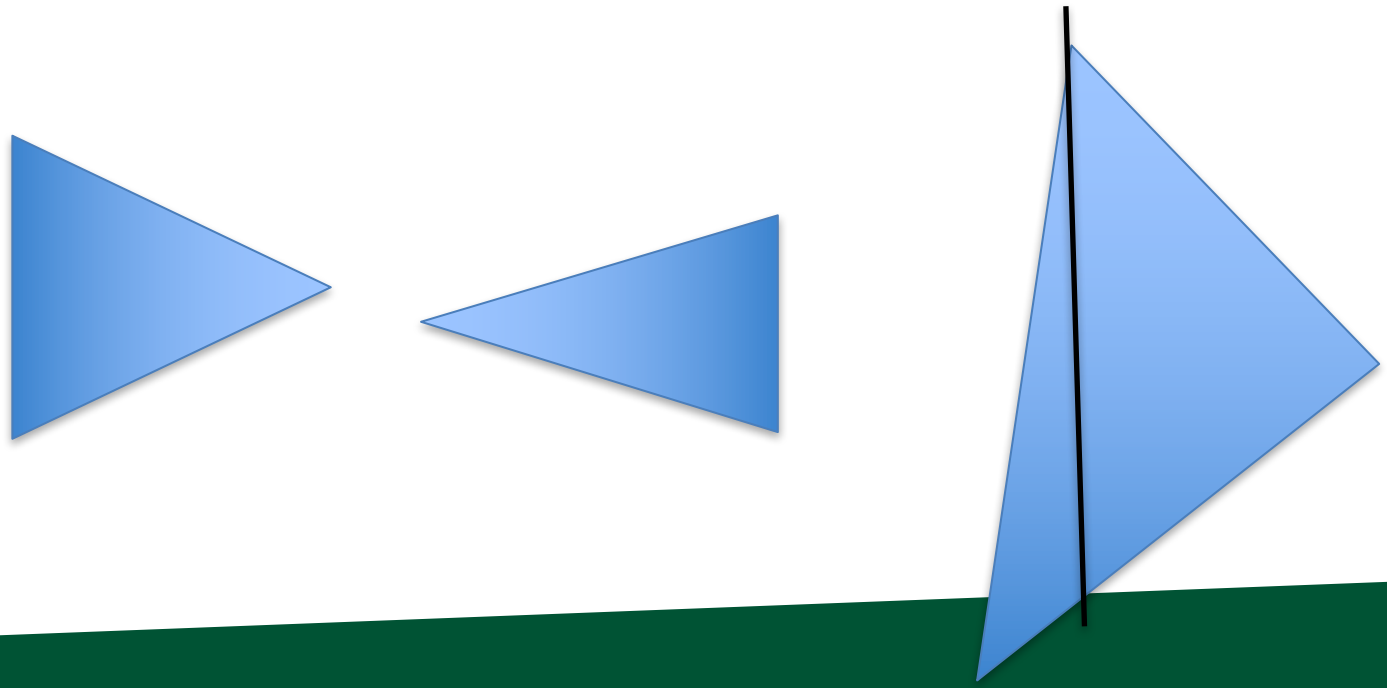
Project #1B

- Goal: apply the scanline algorithm to "going right" triangles and output an image
- File "project1B.cxx" has triangles defined in it
- Due: Weds April 7
- % of grade: 3%



Project #1C

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



Tips On Floating Point Precision

Project 1B

- Cout/cerr can be misleading:

```
fawcett:Downloads childs$ 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 childs$ ./a.out
188
187.999999999999
```

Project 1B

- The limited accuracy of cerr/cout can cause other functions to be appear to be wrong:

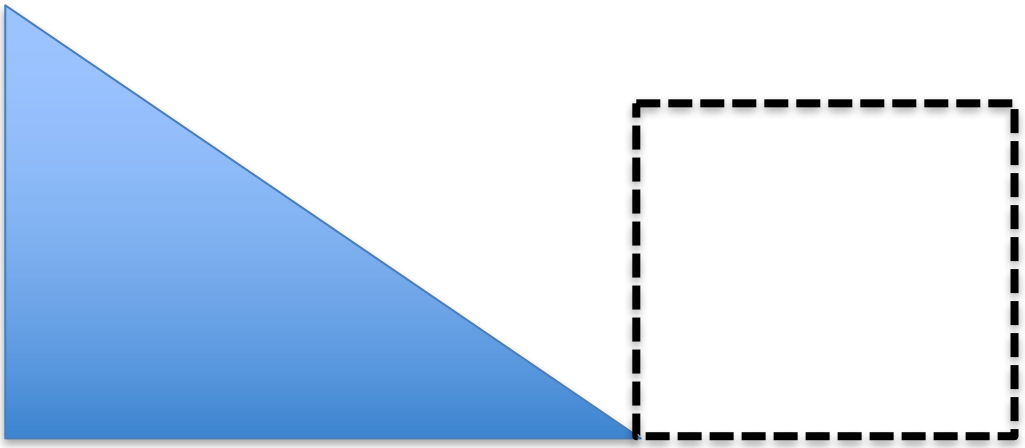
```
fawcett:Downloads childs$ cat t3.C
#include <iostream.h>
#include <iomanip>
#include <math.h>

int main()
{
   double X=188;
   X-=1e-12;
   cerr << "The floor of " << X  << " is " << floor(X) << endl;
}
fawcett:Downloads childs$ ./a.out
The floor of 188 is 187
```

Project 1B

- 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 ≠ A+(B+C)

Project 1B

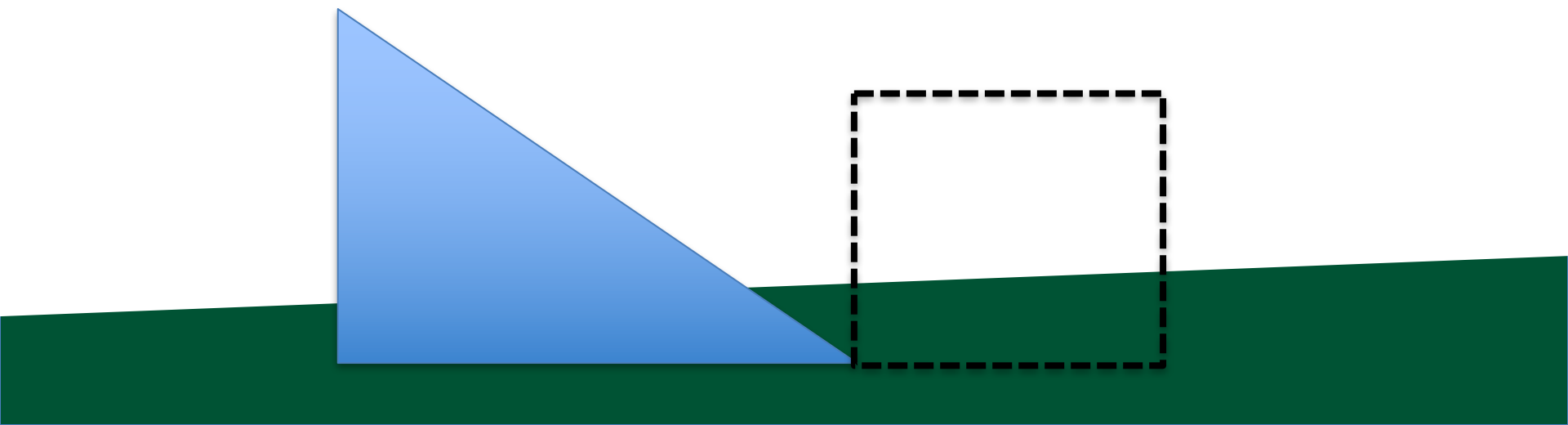
- For project 1B, we are making a binary decision for each pixel: should it be colored or not?
- Consider when a triangle vertex coincides with the bottom left of a pixel:



- We all do different variations on how to solve for the endpoints of a line, so we all get slightly different errors.

Project 1B

- Our algorithm incorporates floor and ceiling functions.
  - This is the right place to bypass the precision problem.
  - I have included “floor__441” and “ceil__441” 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!