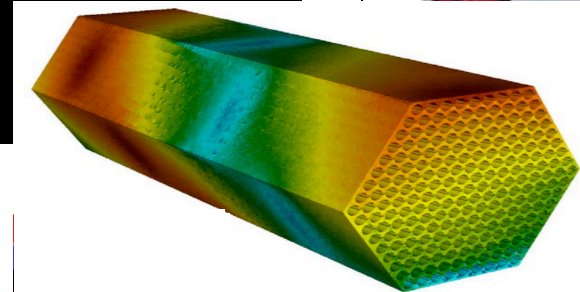
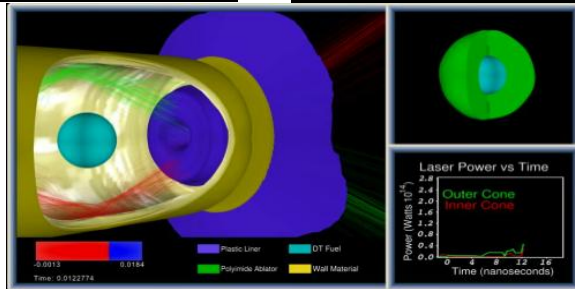
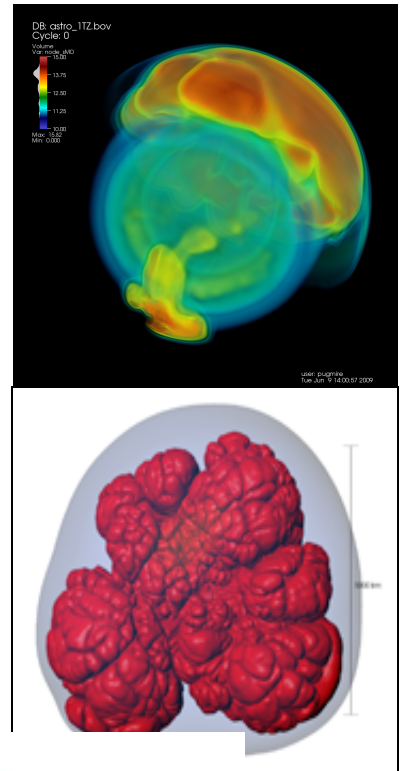
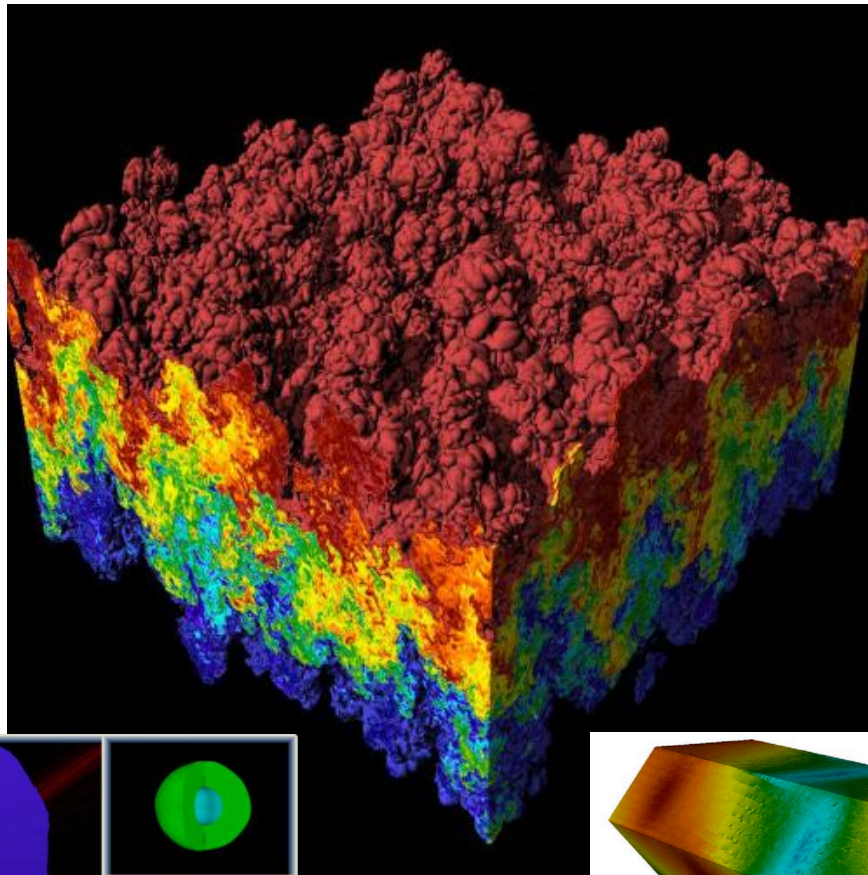
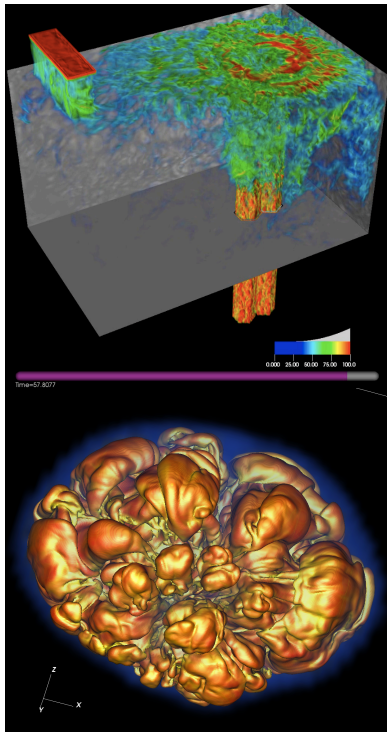


CIS 441/541: Intro to Computer Graphics

Lecture 2: The Scanline Algorithm





Announcements



Trying to Add the Class?

- Let's talk after class...



Projects

- 1A due Saturday
- 1B: assigned today, due Monday
- 1C: assigned Tuesday, due following Wednesday (1/24)



Announcements

- I am adding a 3rd late pass
- Not everyone will be able to get 1B done by Monday
- BUT: I really want you to try, since I want to be able to offer you lots of OH support



Office Hours

- Hank
 - Thursday 5pm (help with 1A)
 - Friday: ??? (help with 1A)
 - Sunday: 3pm-430pm (help with 1B)
 - Monday: ??? (help with 1B)
- Roscoe



Project 1A



Project #1A

- Goal: write a specific image
- Due: Sat Jan 12th
- % of grade: 2%
- May be a little painful

CIS 441/541: Project #1A Due 11:59pm September 30th, 2016
(which means 6am October 1st)

Worth 2% of your grade

Setup:

- 1) Download and install CMake. Use version 2.8.12.1 or higher
- 2) Download, build, and install VTK. Use version 6.X.
- 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
- 2) The color for the Xth strip should be:
 - a. $X \% 3 = 0 \rightarrow B=0$
 - b. $X \% 3 = 1 \rightarrow B=128$
 - c. $X \% 3 = 2 \rightarrow B=255$
 - 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. $X/9 = 0 \rightarrow R=0$
 - h. $X/9 = 1 \rightarrow R=128$
 - i. $X/9 = 2 \rightarrow R=255$
- 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, $R=0, G=0, B=255$
 - 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 $M \times N \times 3$ bytes.
 - It is stored as:
 $P_0/R, P_0/G, P_0/B, P_1/R, P_1/G, P_1/B, \dots P(M \times N)/R, P(M \times N)/G, P(M \times N)/B$
- P_0 is the top, left pixel
- $P(M-1)$ is the top, right pixel
- $P((M \times N) - M + 1)$ is the bottom, left pixel
- $P(M \times N)$ 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* ?

- ❑ 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



The screenshot shows the CMake website homepage. At the top left is the CMake logo, a 3D triangle with red, blue, and green faces. To its right is the word "CMake" in a large, bold, sans-serif font. In the top right corner, there is a "Kitware" logo and a search bar. Below the header, there are navigation links: "PROJECT", "RESOURCES", "HELP", and "OPEN SOURCE". The main content area on the left contains a welcome message: "Welcome to **CMake**, the 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." Below this is a "News" section with a list of recent releases and a "More News >" link. On the right side, there is a large banner for "CMake 2.8.10 Available for Download" featuring the CMake logo and a "Download Now >" link.

CMake

Welcome to **CMake**, the 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.

News [More News >](#)

- 11.07.2012** CMake 2.8.10 Just Released
- 08.09.2012** CMake 2.8.9 is Now Available!
- 07.18.2012** Kitware Announces New Fall Courses
- 04.19.2012** CMake 2.8.8 is Now Available
- 03.02.2012** CDash 2.0.2 Now Available

CMake 2.8.10 Available for Download

[Download Now >](#)

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 Resources->Download and follow the directions



The screenshot shows the Kitware VTK website. The top navigation bar includes links for PROJECT, RESOURCES, HELP, and OPEN SOURCE. A blue arrow points to the 'RESOURCES' link. The main content area features a detailed description of VTK as an open-source software system for 3D computer graphics, image processing, and visualization. It lists various supported languages (C++, Tcl/Tk, Java, Python) and visualization techniques (scalar, vector, tensor, texture, volumetric methods, etc.). A 'News' section on the left lists recent updates, including mobile visualization support, ParaView 3.98.0 availability, DARPA funding, new fall courses, and VTK 5.10 availability. On the right, a banner announces that Kitware received HPCwire's Editors' Choice Award for VTK in 2011, accompanied by a 3D visualization of a complex, multi-colored structure.

Kitware Search
Tell us what you think

Visualization Toolkit

PROJECT RESOURCES HELP OPEN SOURCE

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. **Kitware**, whose team created and continues to extend the toolkit, offers [professional support and consulting](#) services for VTK. VTK supports a wide variety of visualization algorithms including: scalar, vector, tensor, texture, and volumetric methods; and advanced modeling techniques such as: implicit modeling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation. VTK has an extensive information visualization framework, has a suite of 3D interaction widgets, supports parallel processing, and integrates with various databases on GUI toolkits such as Qt and Tk. VTK is cross-platform and runs on Linux, Windows, Mac and Unix platforms.

News [More News >](#)

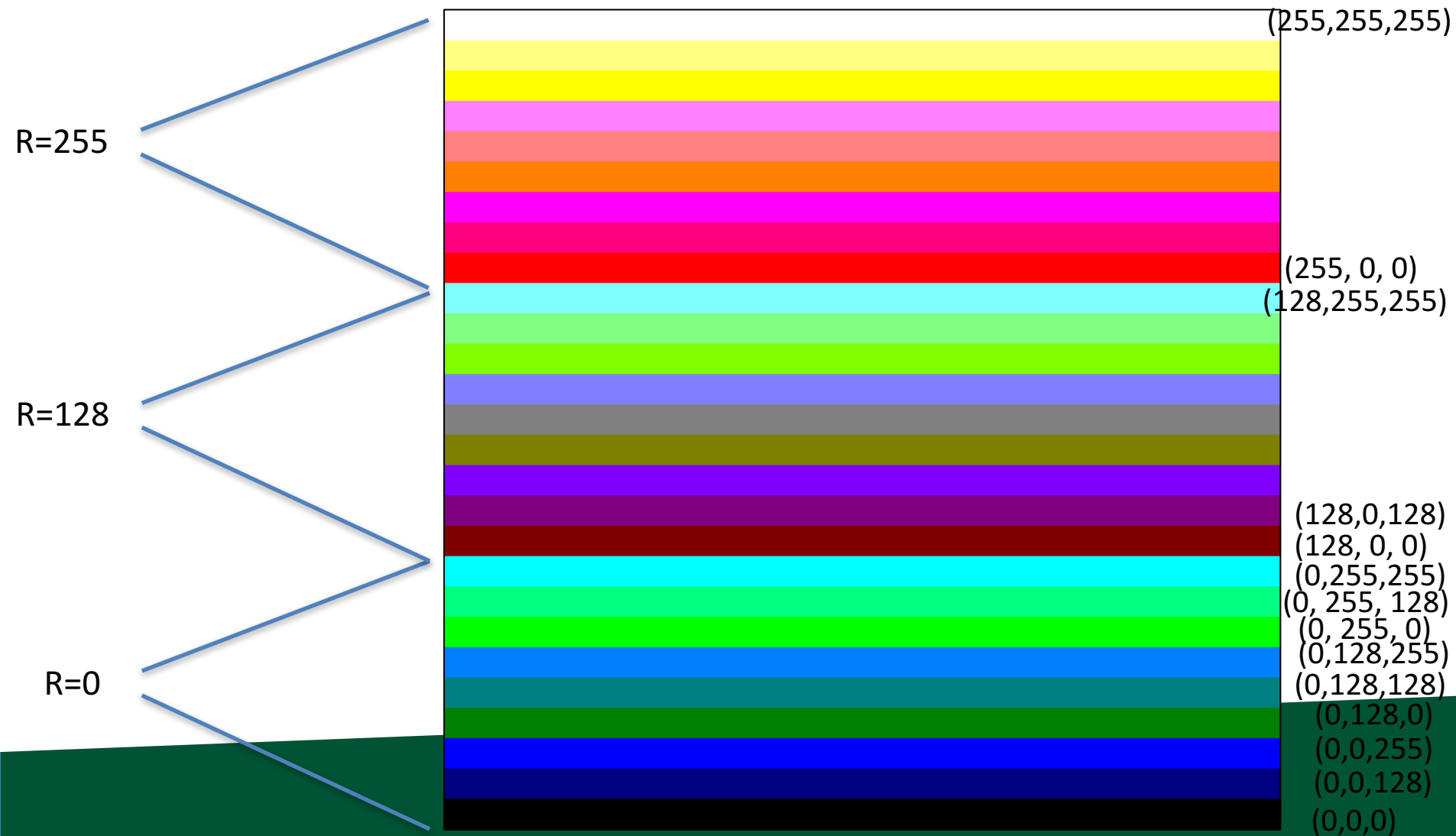
- 01.31.2013** Kitware Provides Mobile Visualization Support for the Visible Pat...
- 12.03.2012** ParaView 3.98.0 Now Available
- 11.27.2012** Kitware Receives DARPA Funding to Develop a Visualization Design ...
- 07.18.2012** Kitware Announces New Fall Courses
- 05.16.2012** VTK 5.10 Now Available

Kitware receives HPCwire's Editors' Choice Award for VTK.

[Learn More >](#)

HPC wire
Editors' Choice Awards 2011

make?



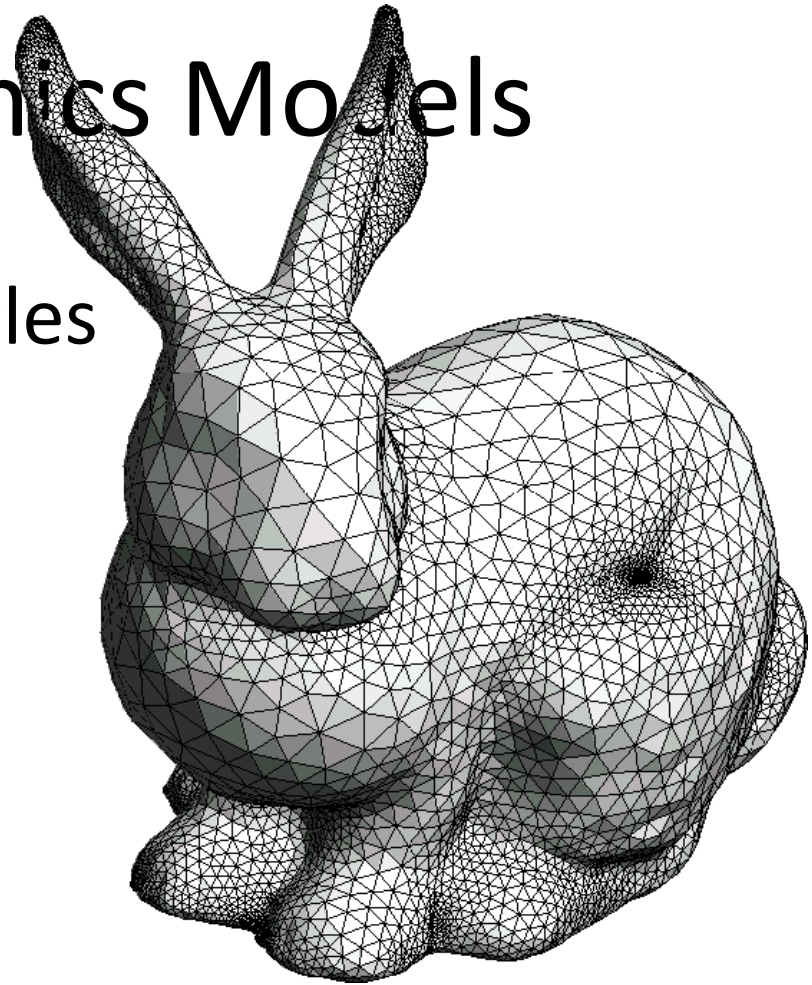
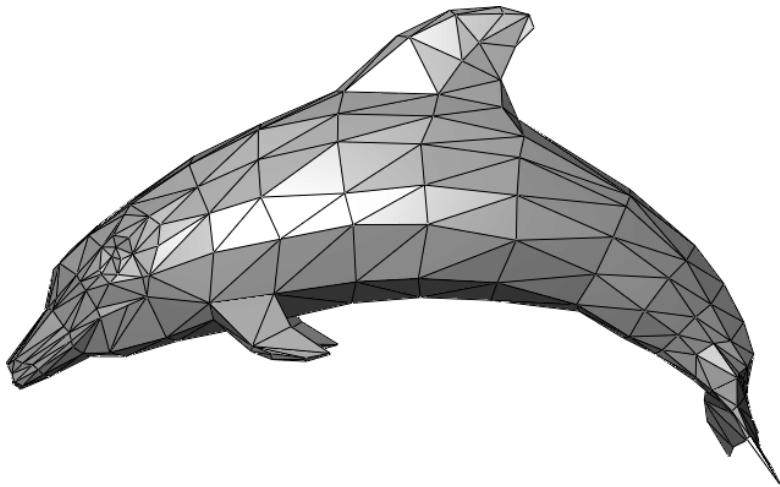


What do I do again?

- 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 and image result by Friday midnight.

Computer Graphics Models

- Usually made up of triangles
 - + tricks for shading





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.
 - $O(n\text{Pixels})$
 - (actually, additional computational complexity for geometry searches)
 - Allows for beautiful rendering effects (reflections, etc)
 - Will discuss at the end of the quarter



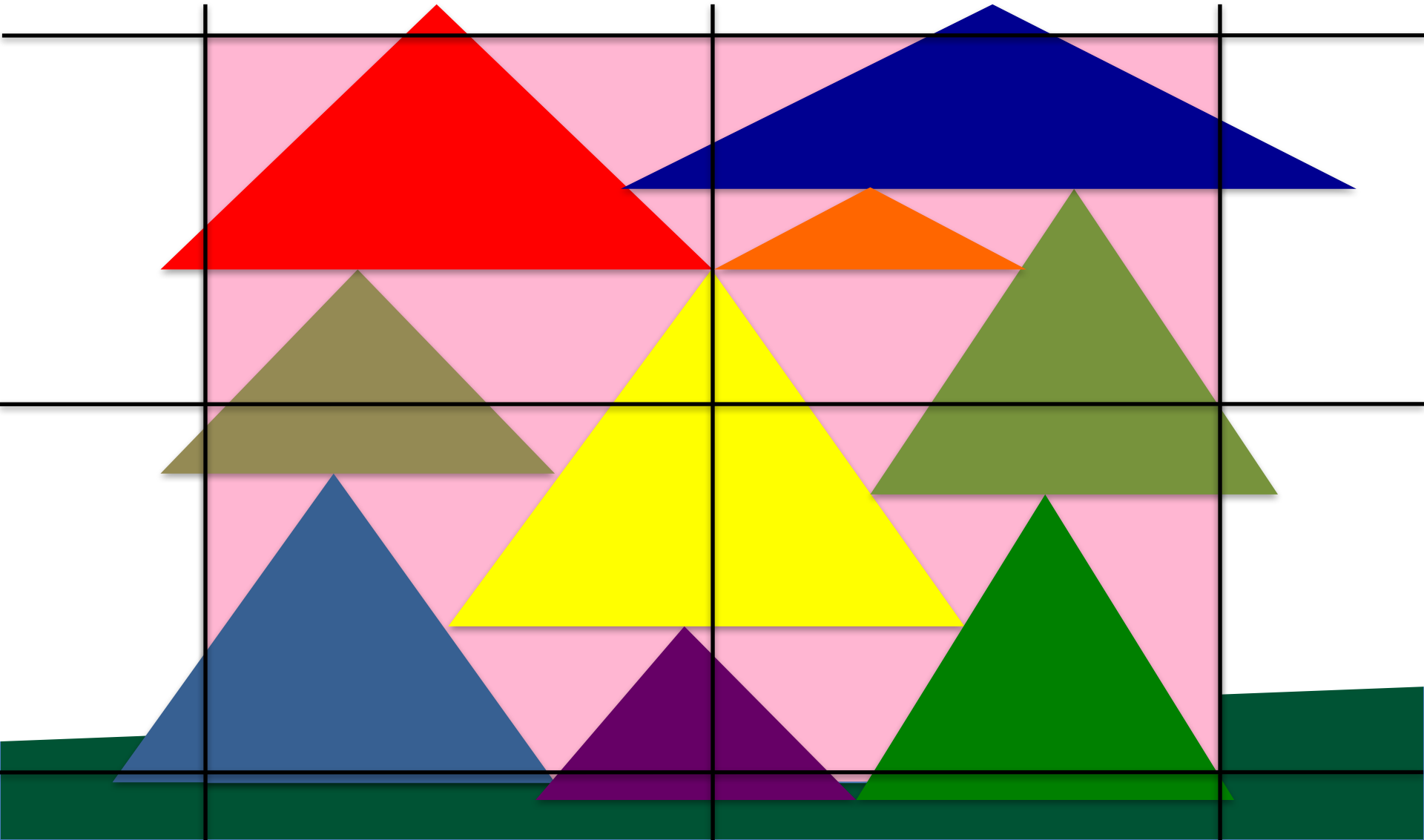


Reminder: ray-tracing vs rasterization

- 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



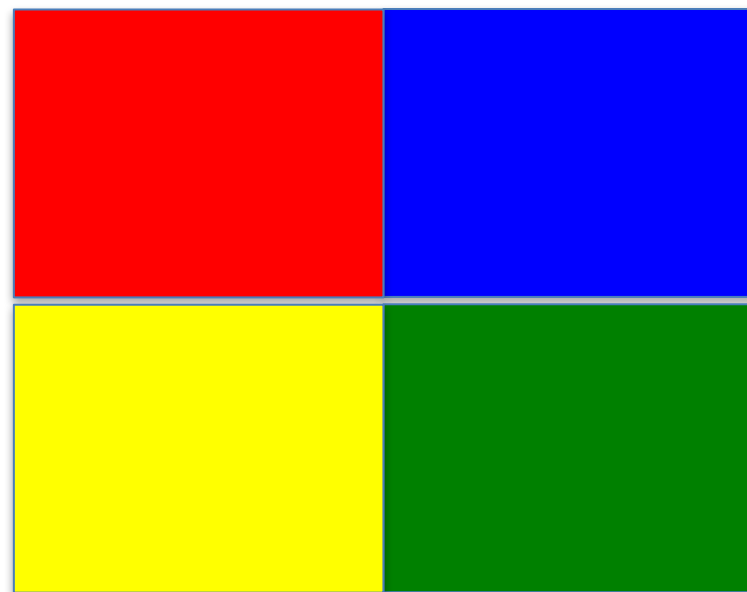
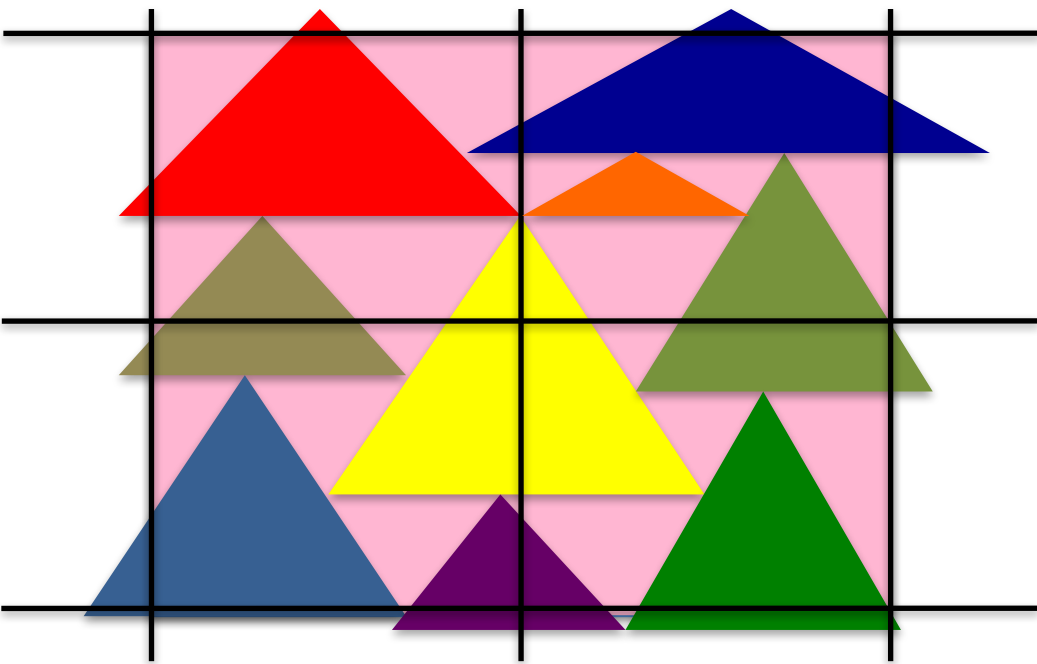
What color should we choose for each of these four pixels?





UNIVERSITY OF OREGON

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

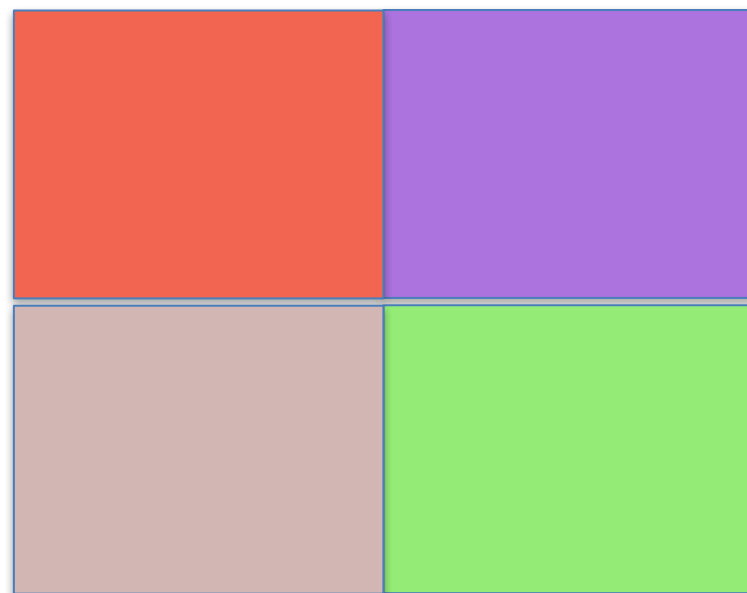
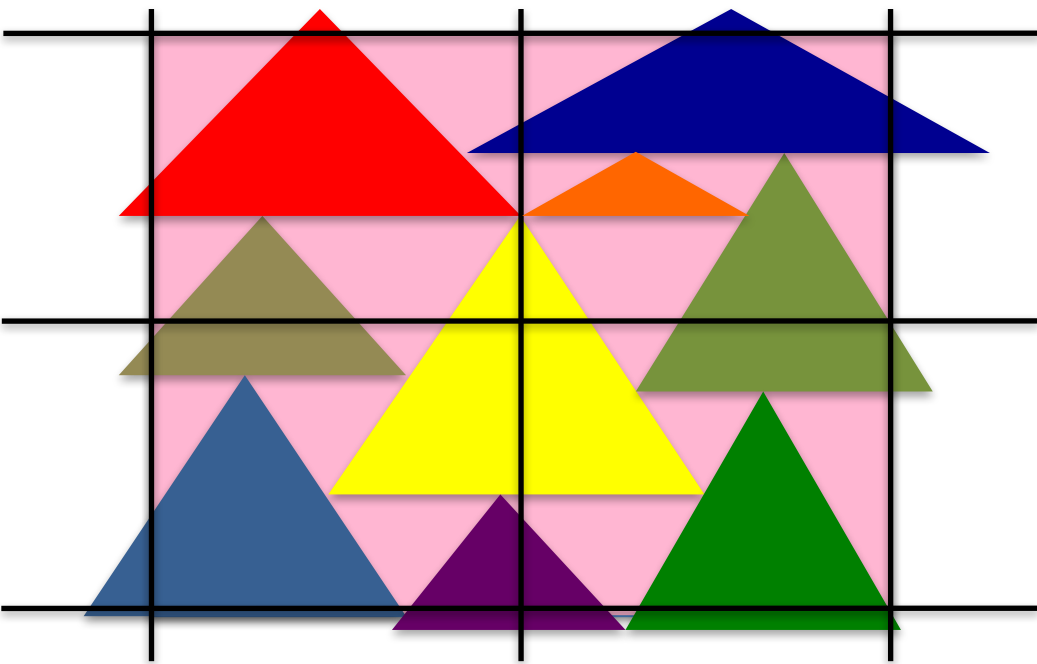


Most dominant triangle



UNIVERSITY OF OREGON

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

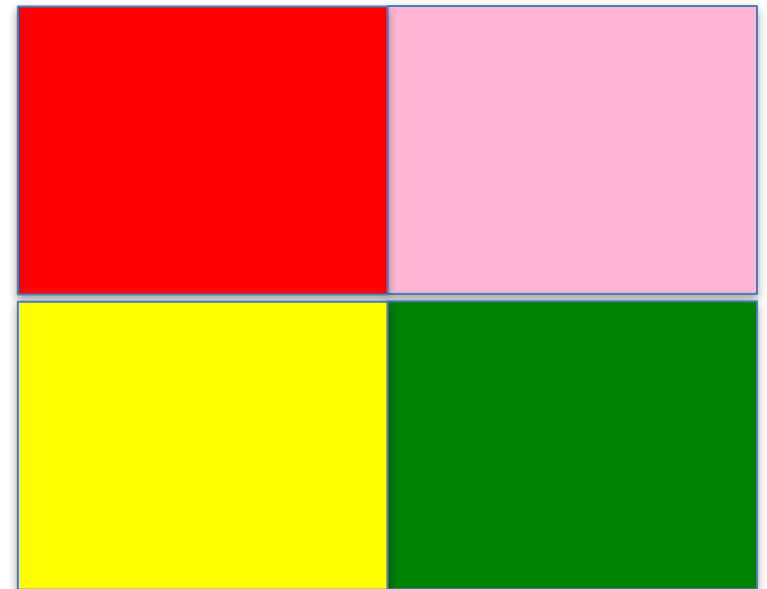
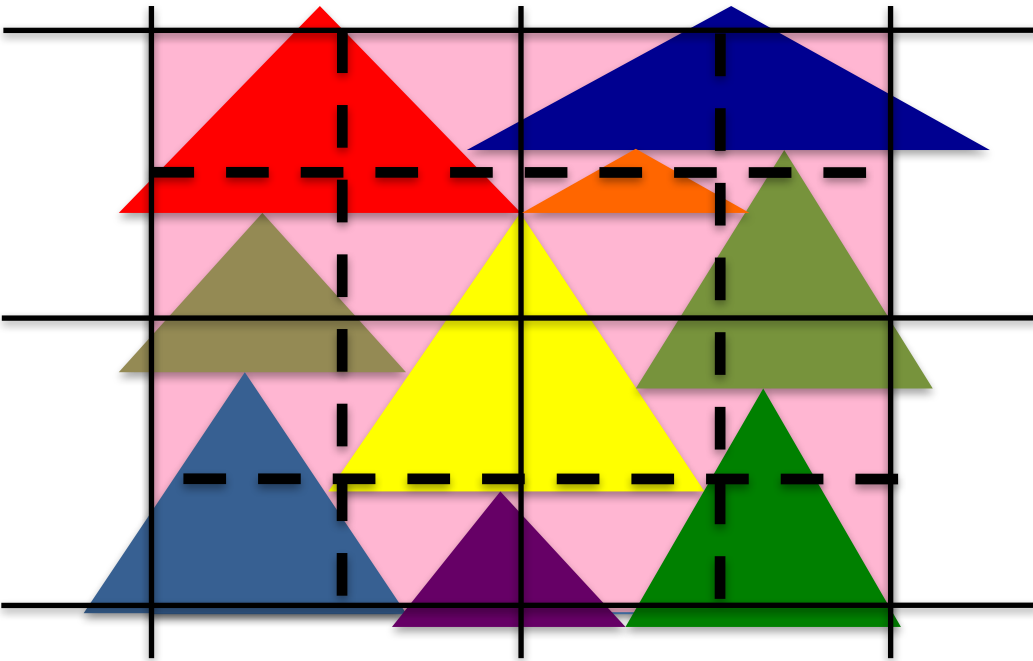


Average



UNIVERSITY OF OREGON

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

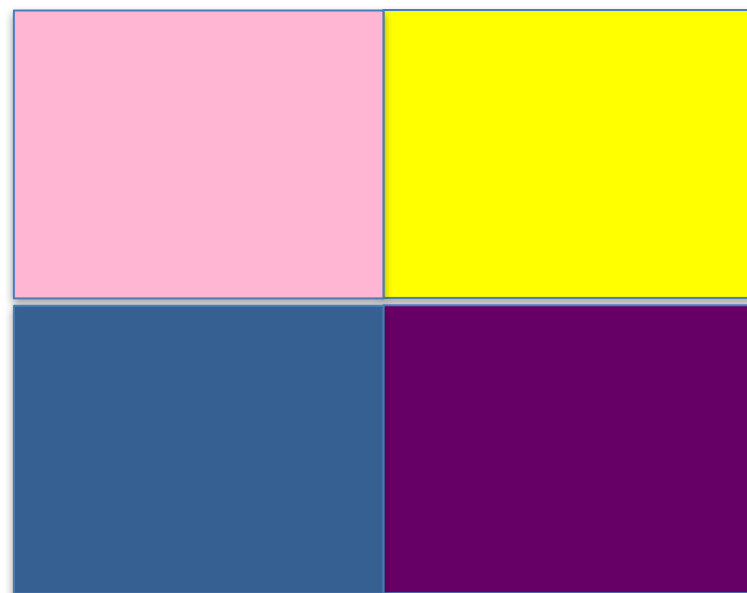
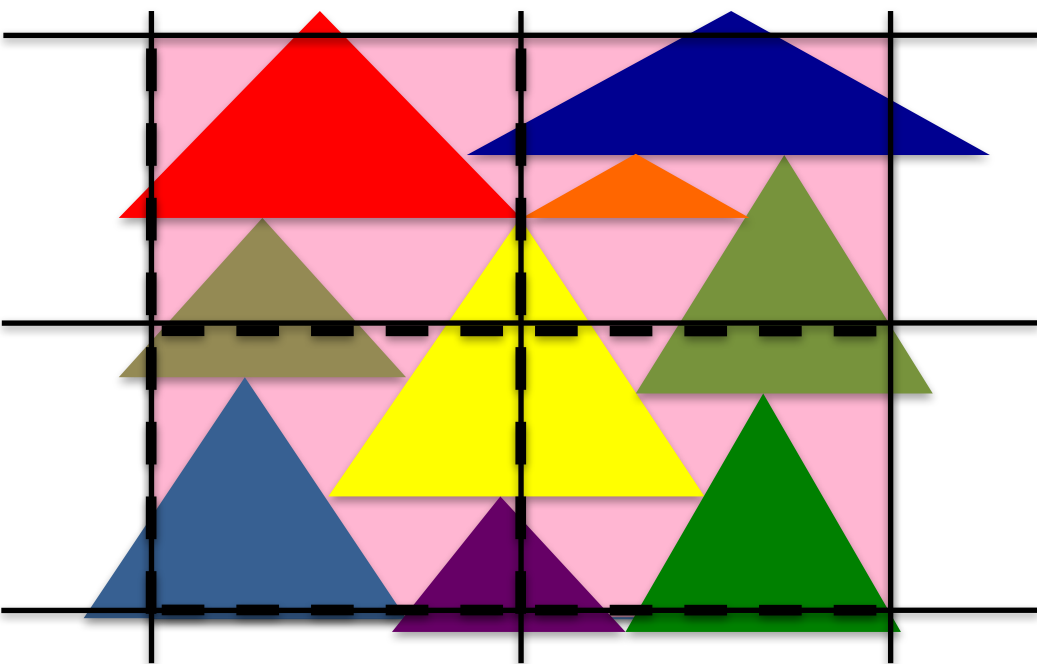


Pixel center



UNIVERSITY OF OREGON

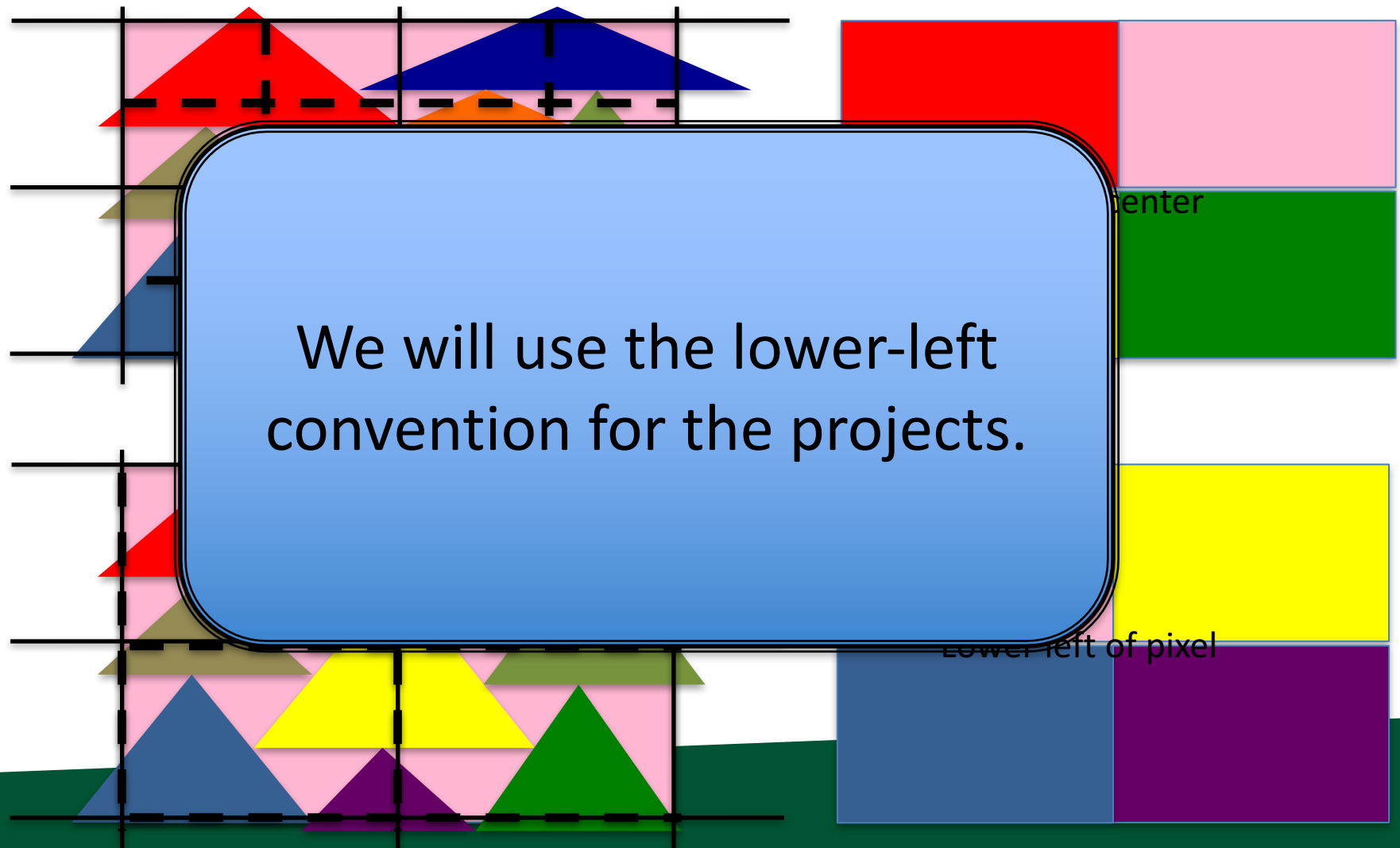
What color should we choose for each of these four pixels?



Lower left of pixel



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





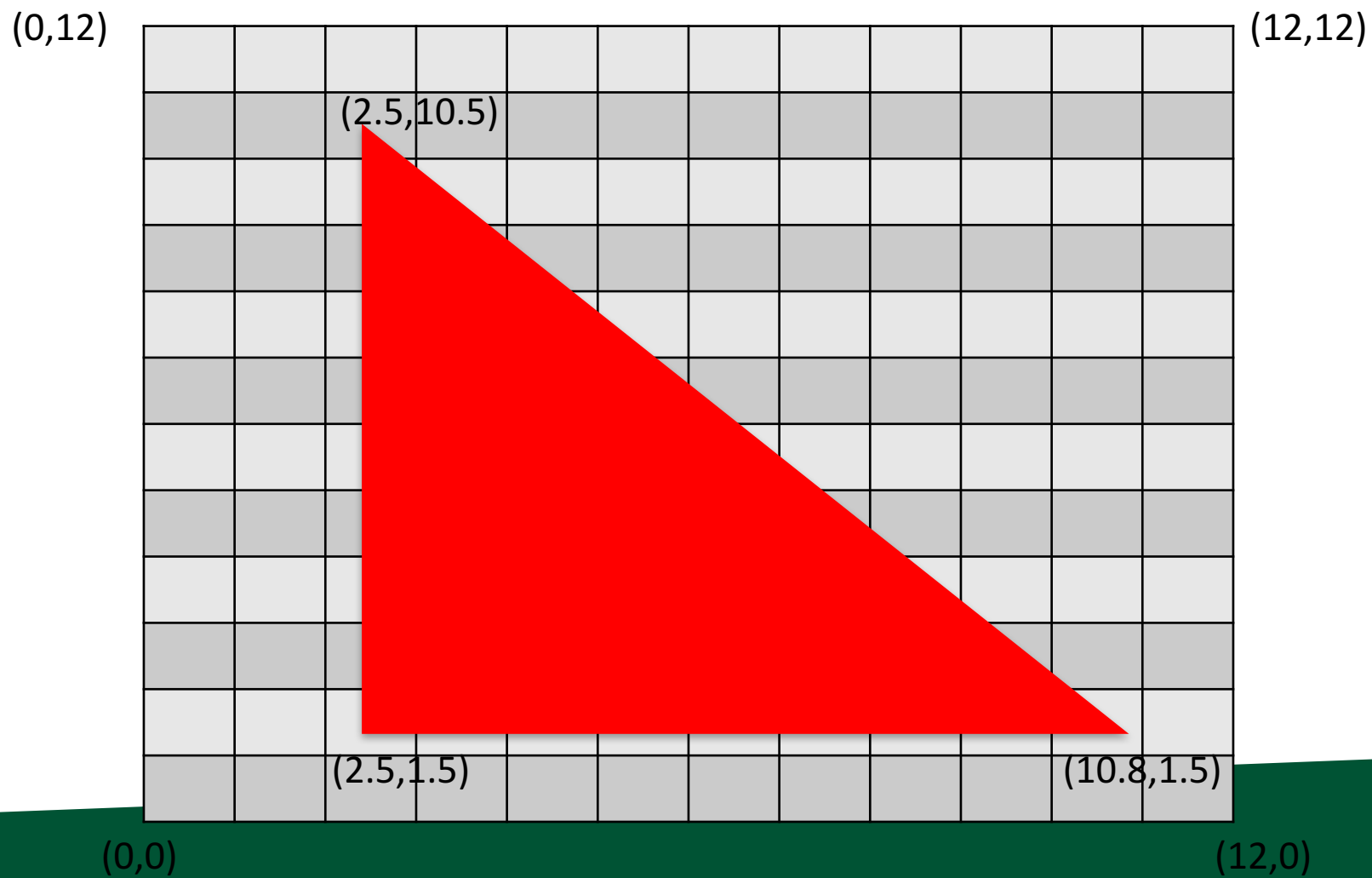
Where we are...

- 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?

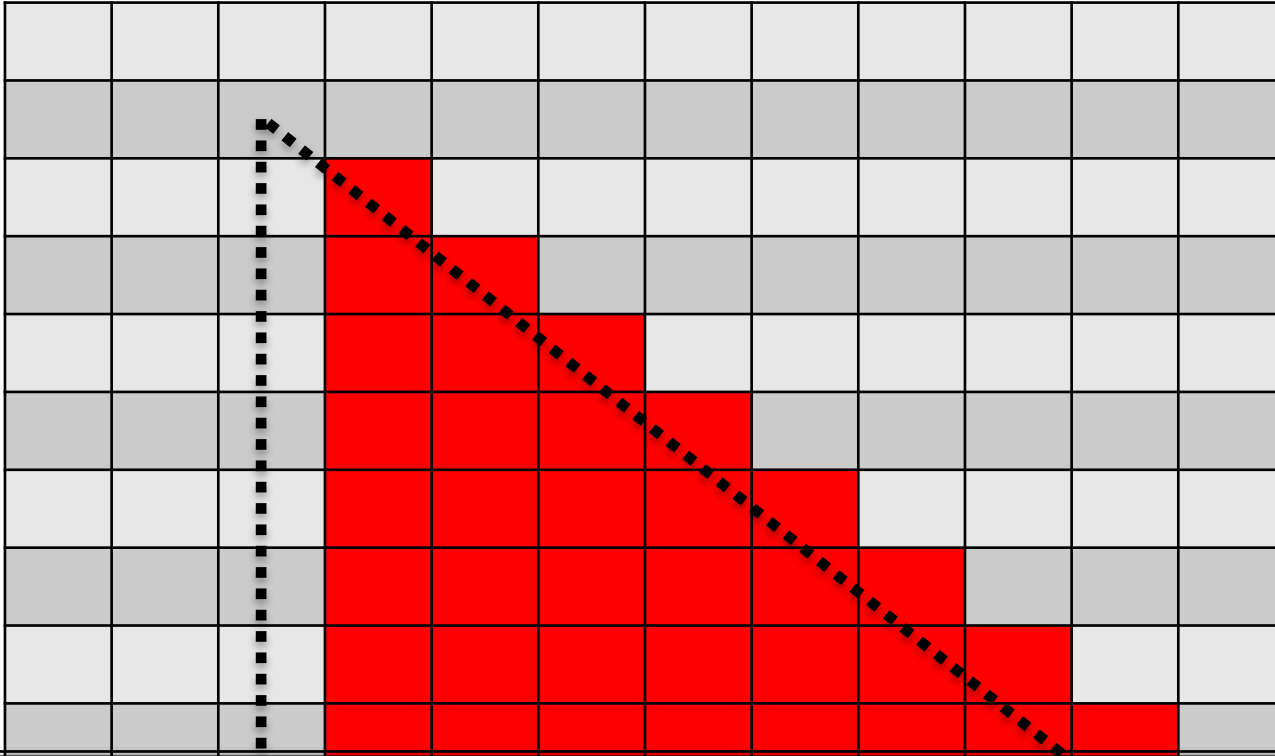


Problem: how to deposit triangle colors onto an image?

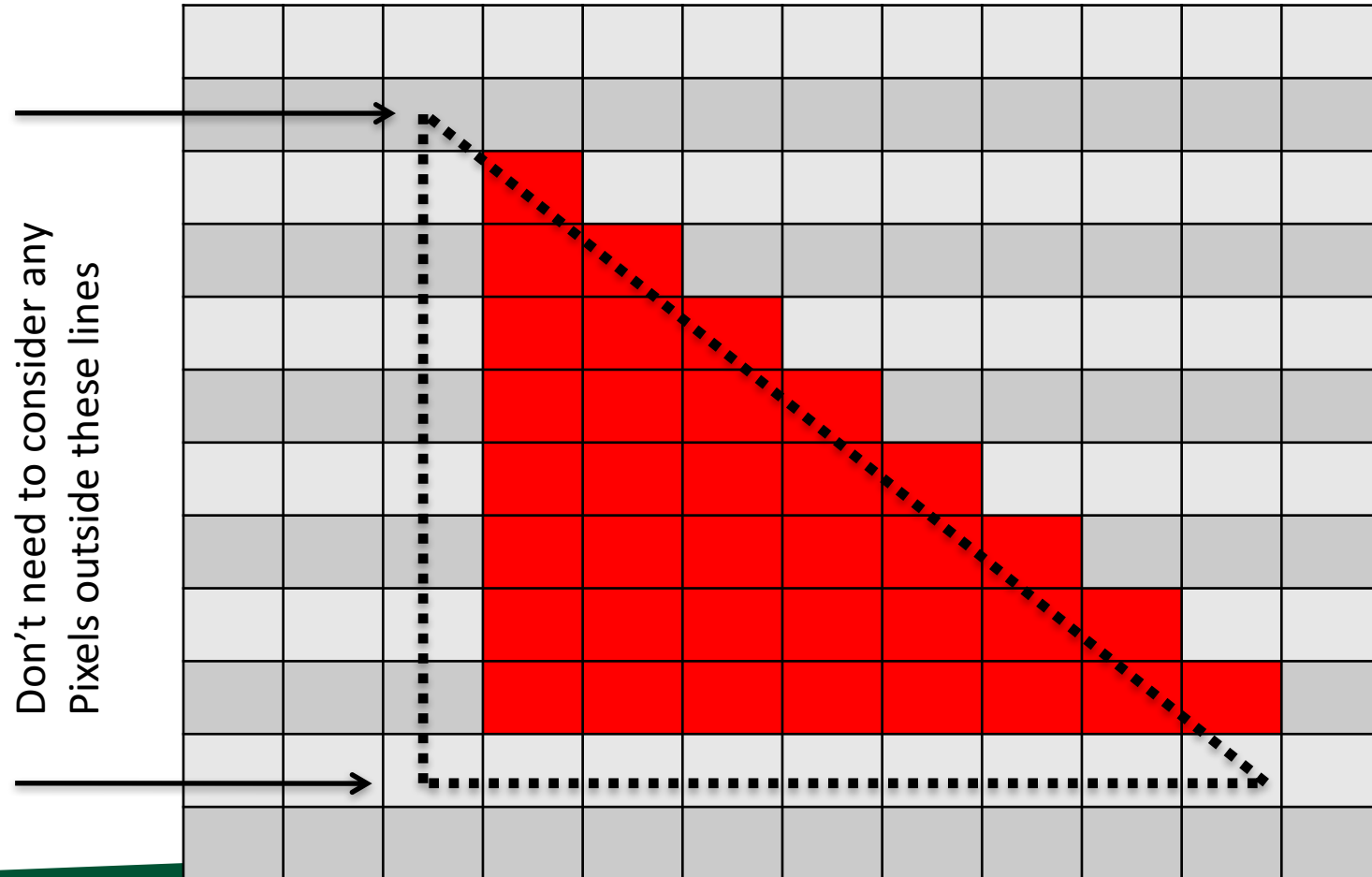
- Let's take an example:
 - 12x12 image
 - Red triangle
 - Vertex 1: (2.5, 1.5)
 - Vertex 2: (2.5, 10.5)
 - Vertex 3: (10.5, 1.5)
 - Vertex coordinates are with respect to pixel locations



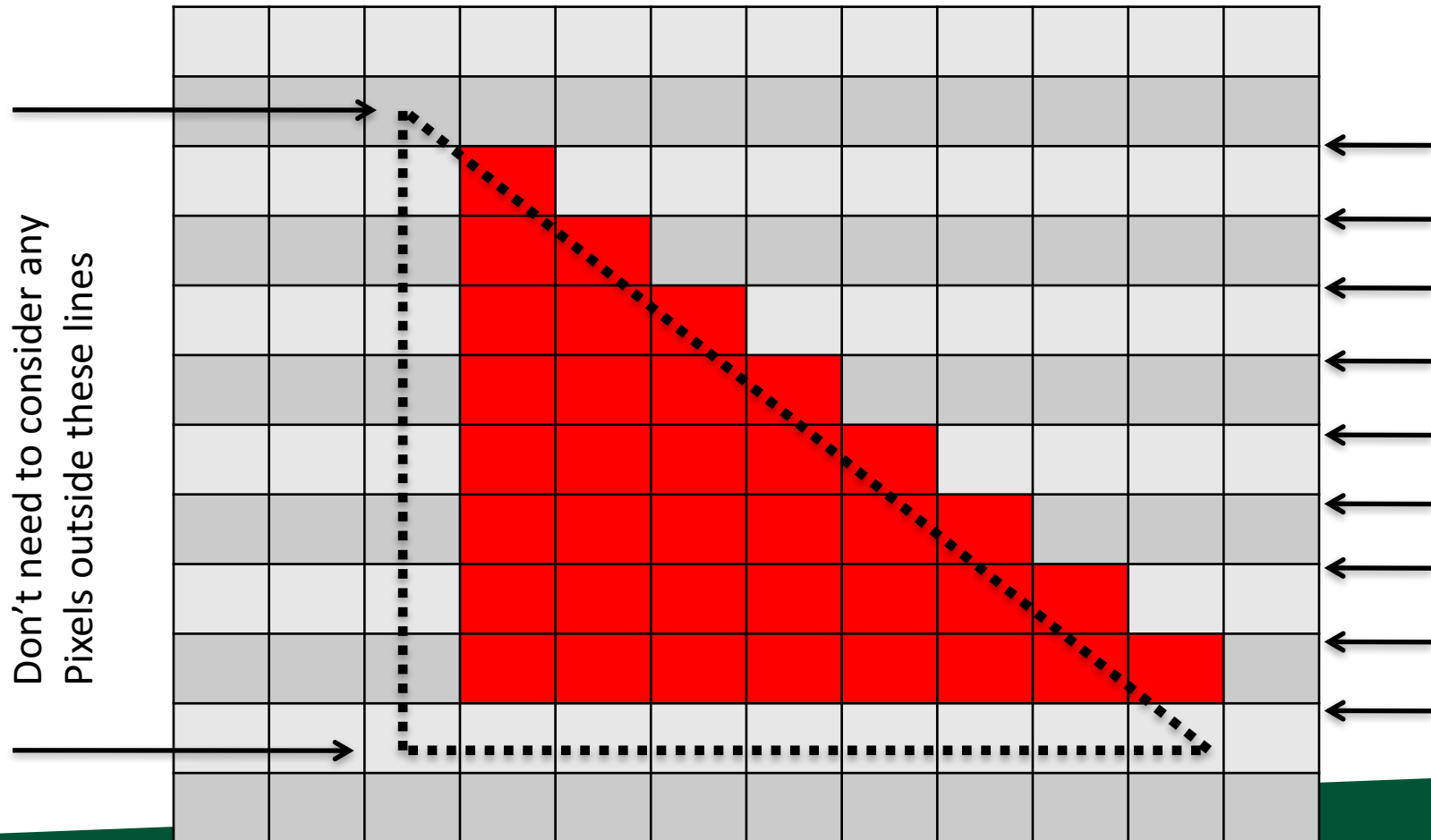
Our desired output



How do we make this output? Efficiently?



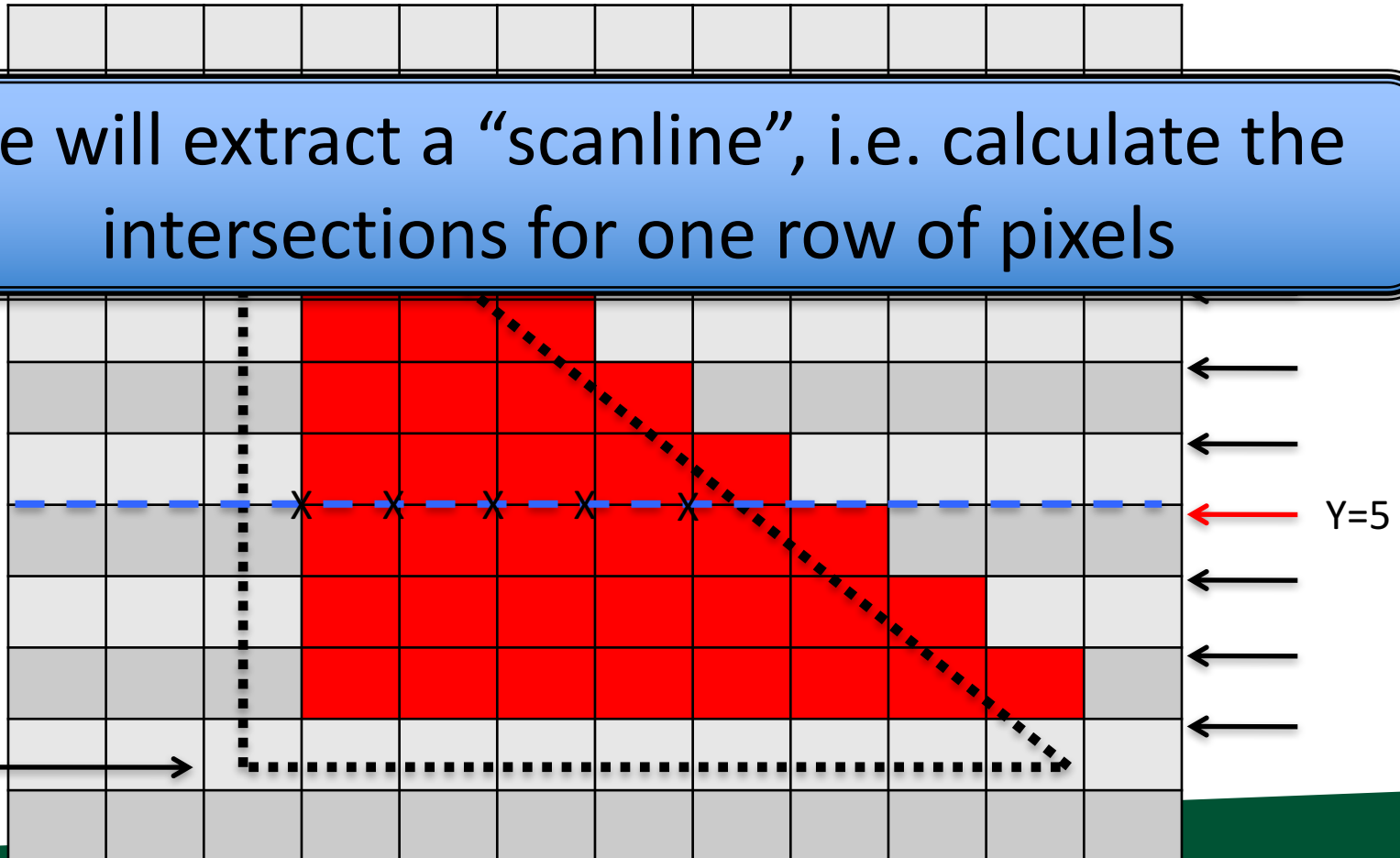
Scanline algorithm: consider all rows that can possibly overlap



Scanline algorithm: consider all rows that can possibly overlap

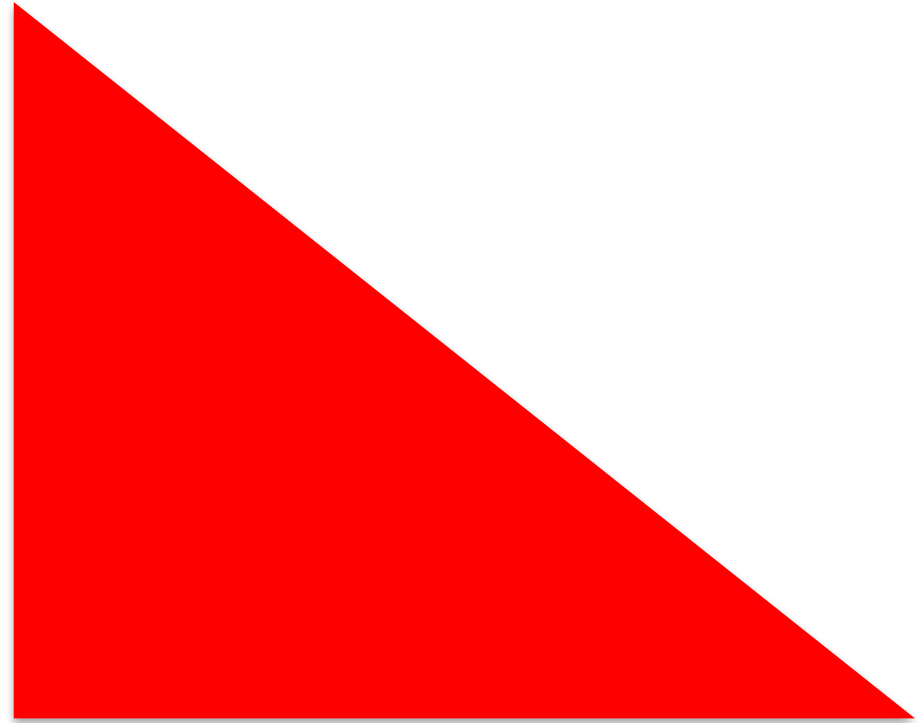
We will extract a “scanline”, i.e. calculate the intersections for one row of pixels

Don't need to consider
Pixels outside these lines



– Red triangle

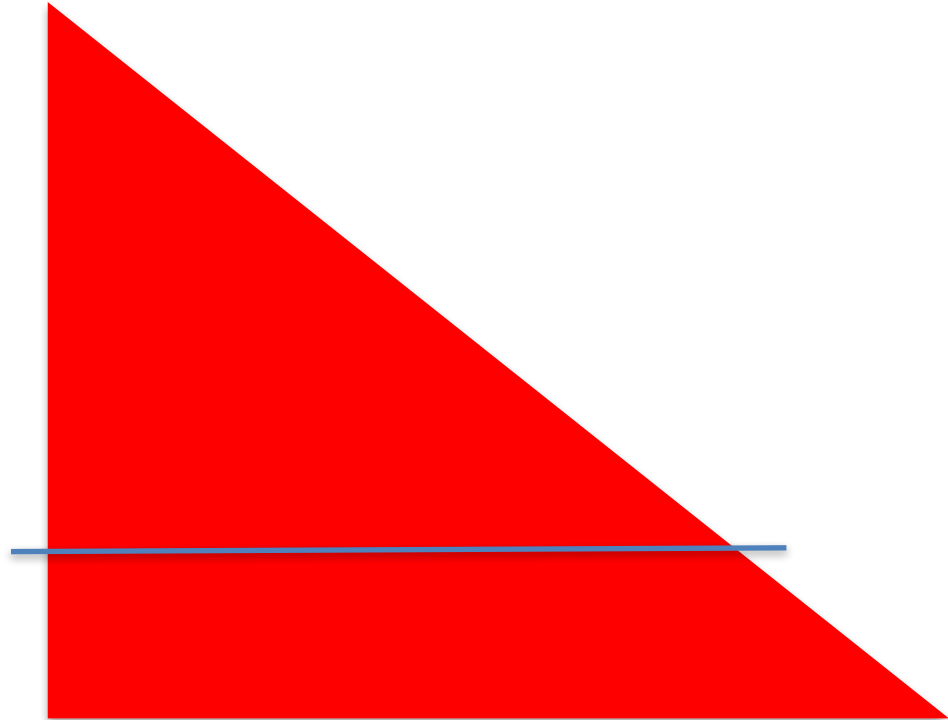
- Vertex 1: (2.5, 1.5)
- Vertex 2: (2.5, 10.5)
- Vertex 3: (10.5, 1.5)



– Red triangle

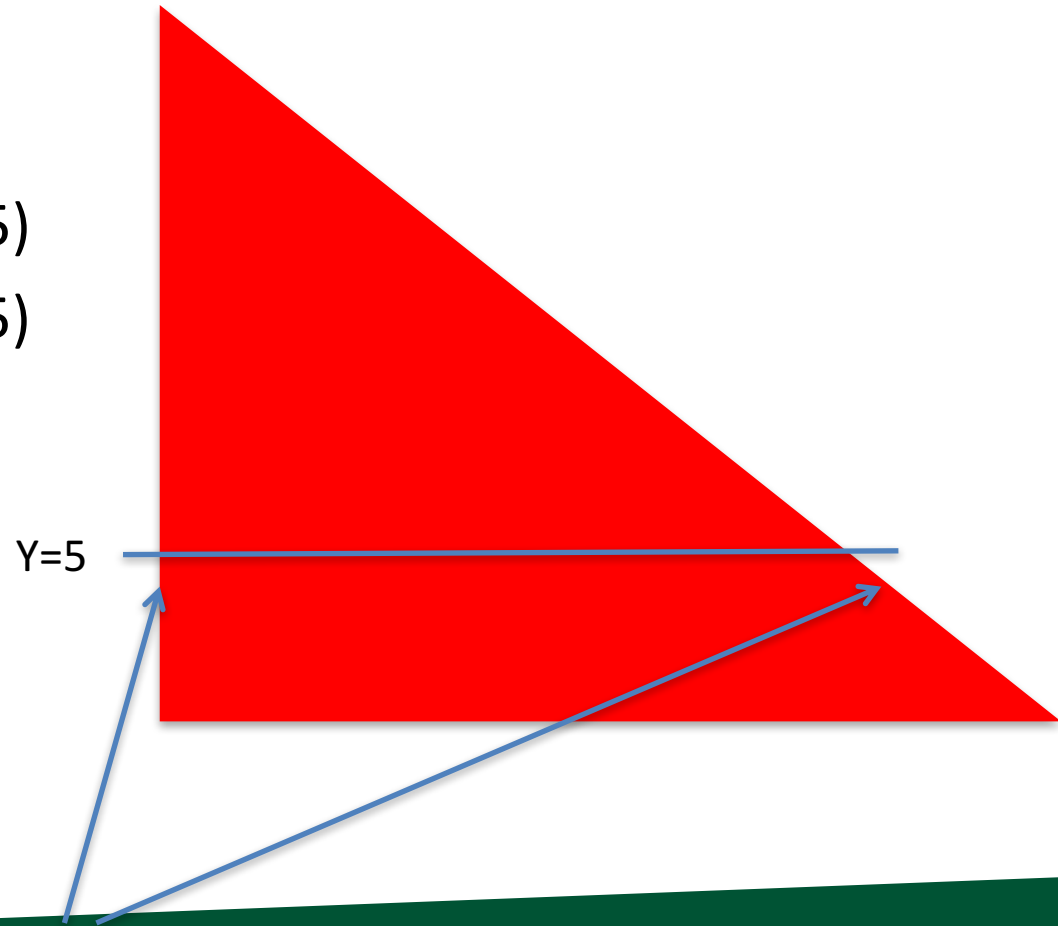
- Vertex 1: (2.5, 1.5)
- Vertex 2: (2.5, 10.5)
- Vertex 3: (10.5, 1.5)

Y=5



– Red triangle

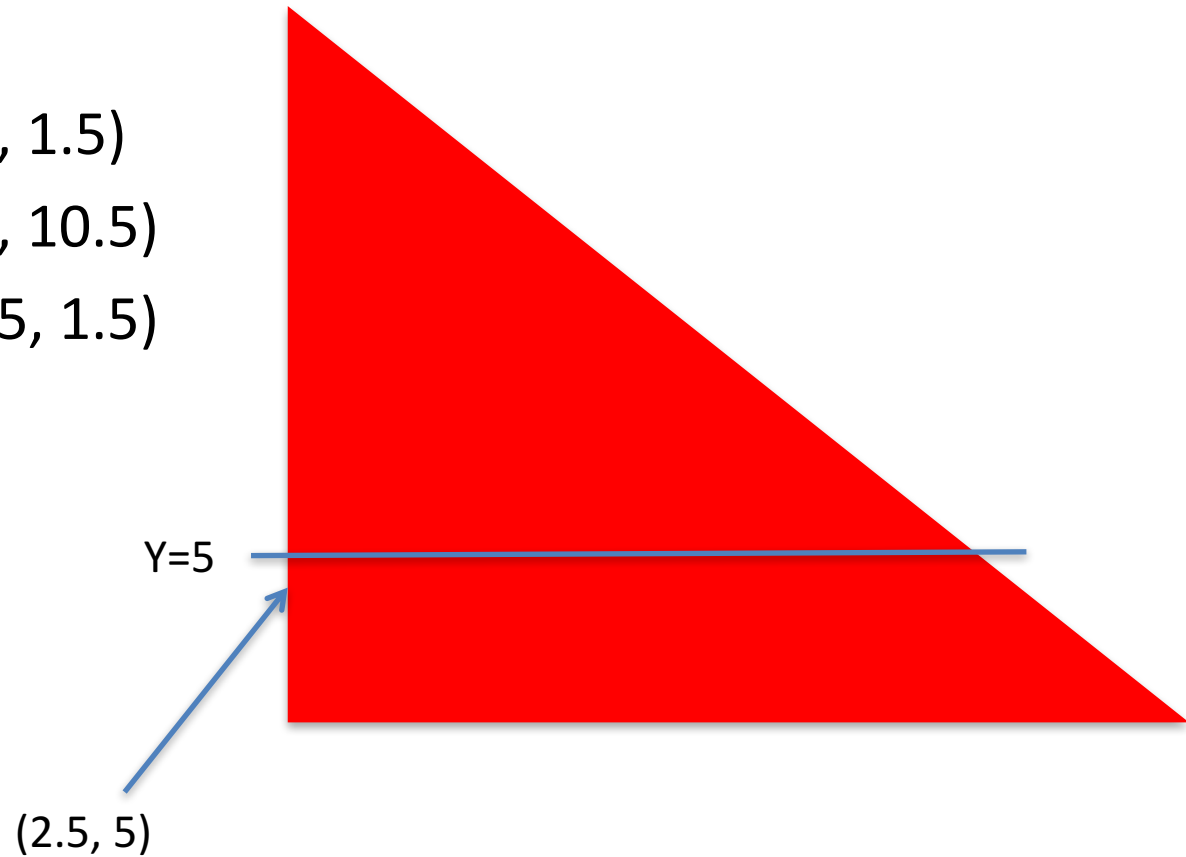
- Vertex 1: (2.5, 1.5)
- Vertex 2: (2.5, 10.5)
- Vertex 3: (10.5, 1.5)



What are the end points?

– Red triangle

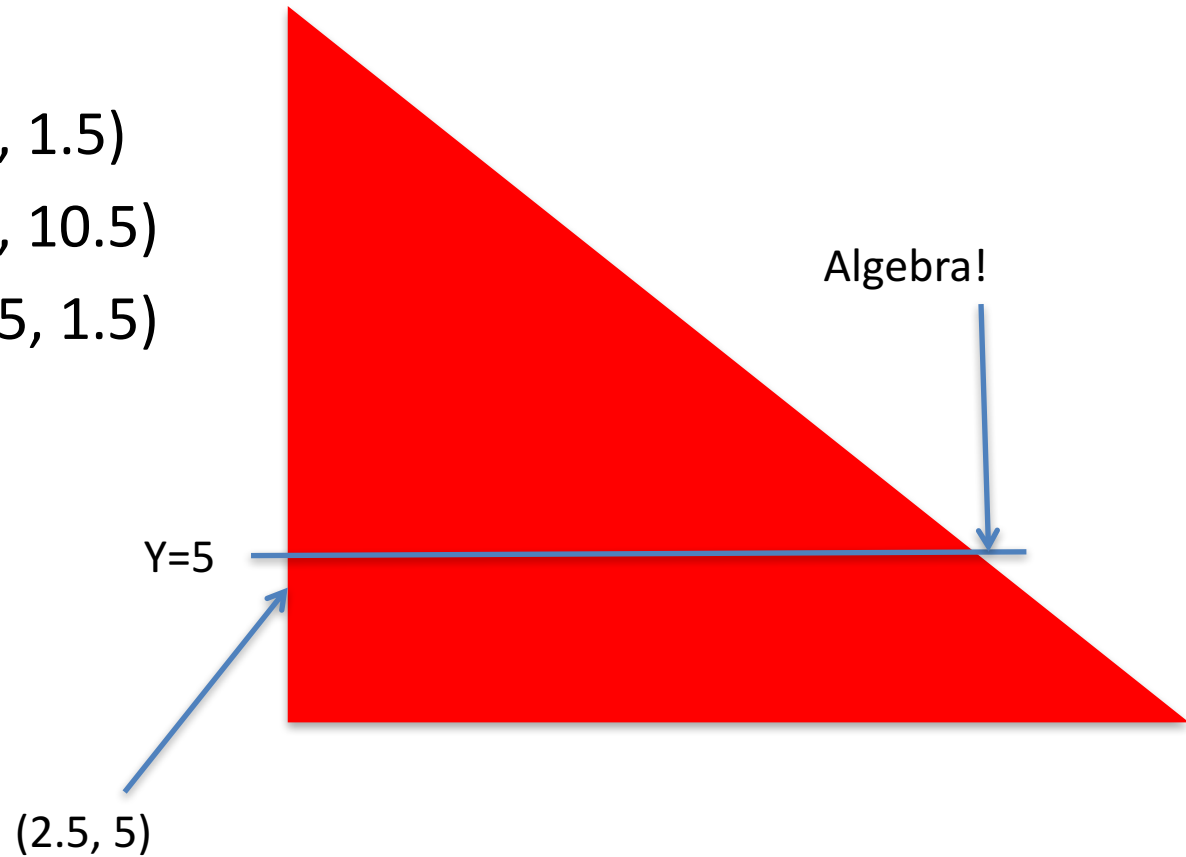
- Vertex 1: (2.5, 1.5)
- Vertex 2: (2.5, 10.5)
- Vertex 3: (10.5, 1.5)



What are the end points?

– 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)

- $Y = mx + b$

- $10.5 = m * 2.5 + b$

- $1.5 = m * 10.5 + b$

- \rightarrow

- $9 = -8m$

- $m = -1.125$

- $b = 13.3125$

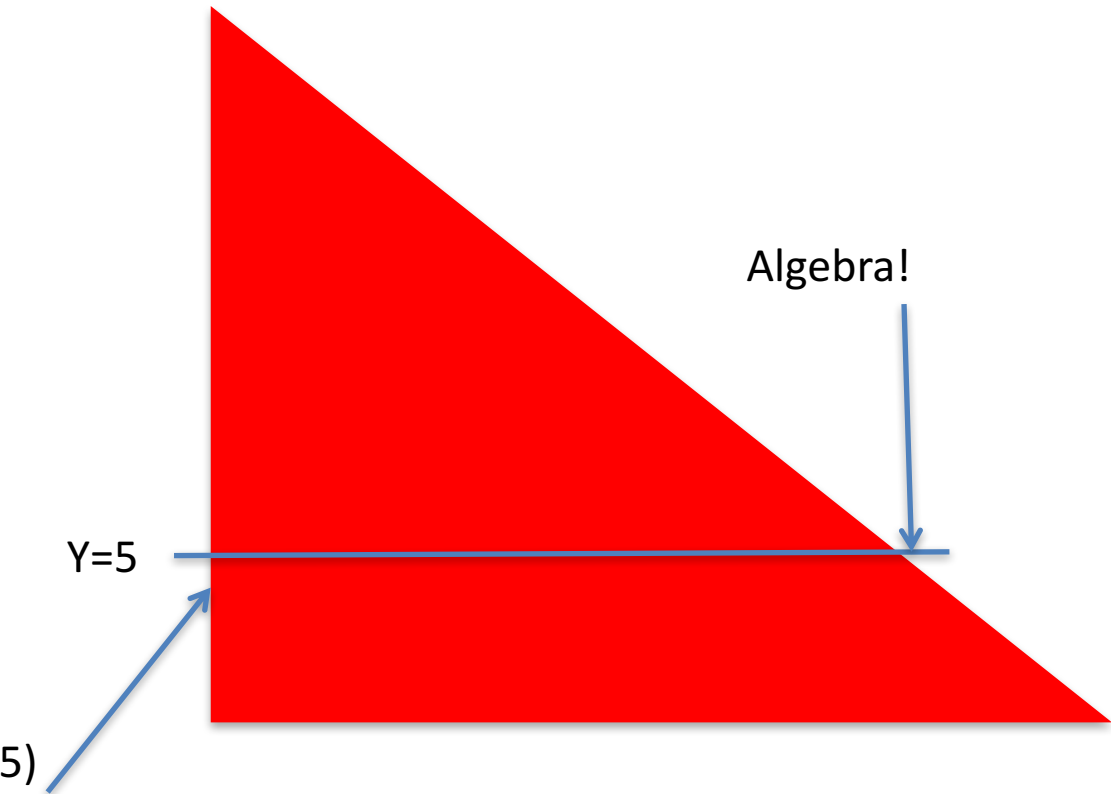
- $5 = -1.125 * x + 13.3125$

- $x = 7.3888$

(2.5, 5)

Y=5

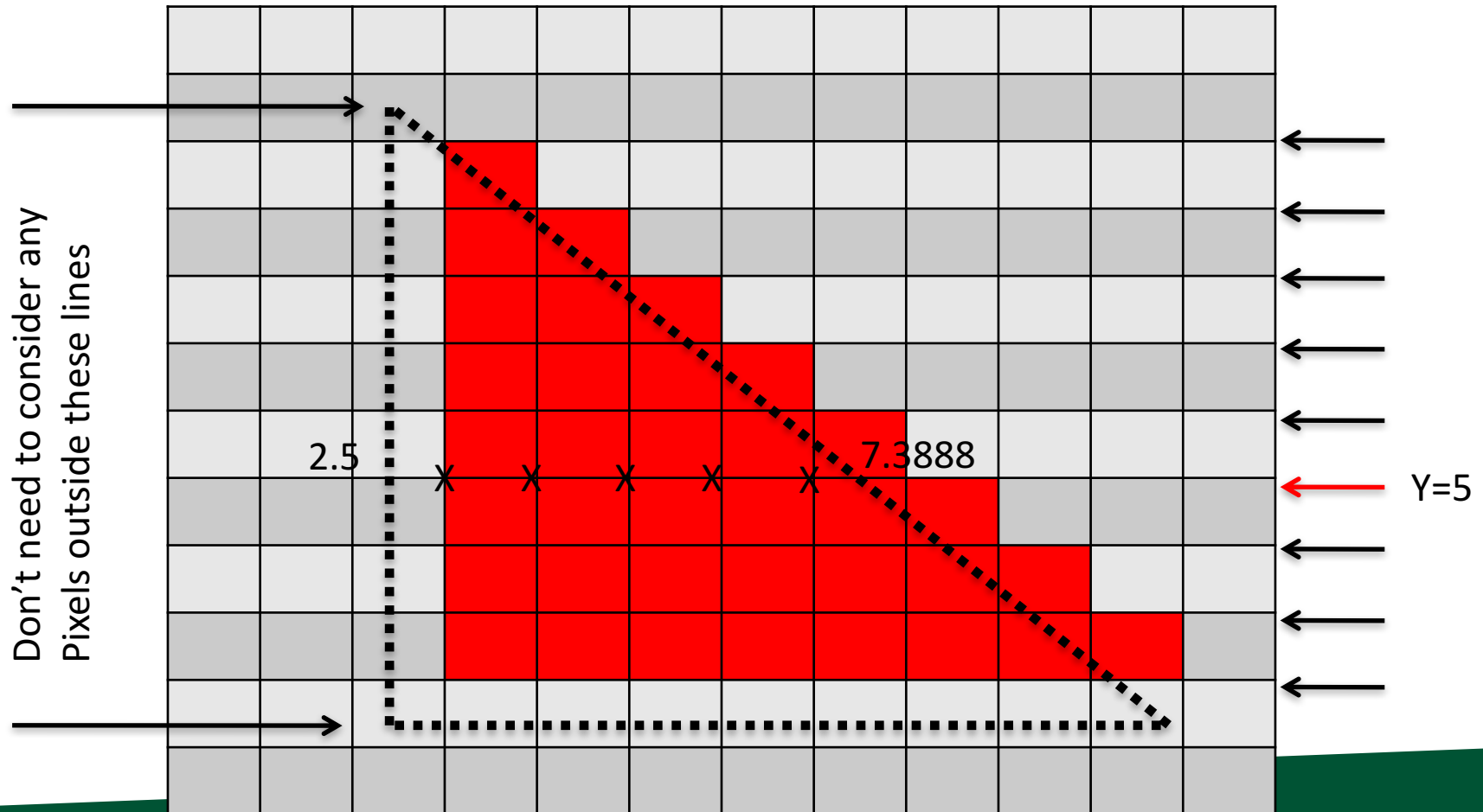
Algebra!



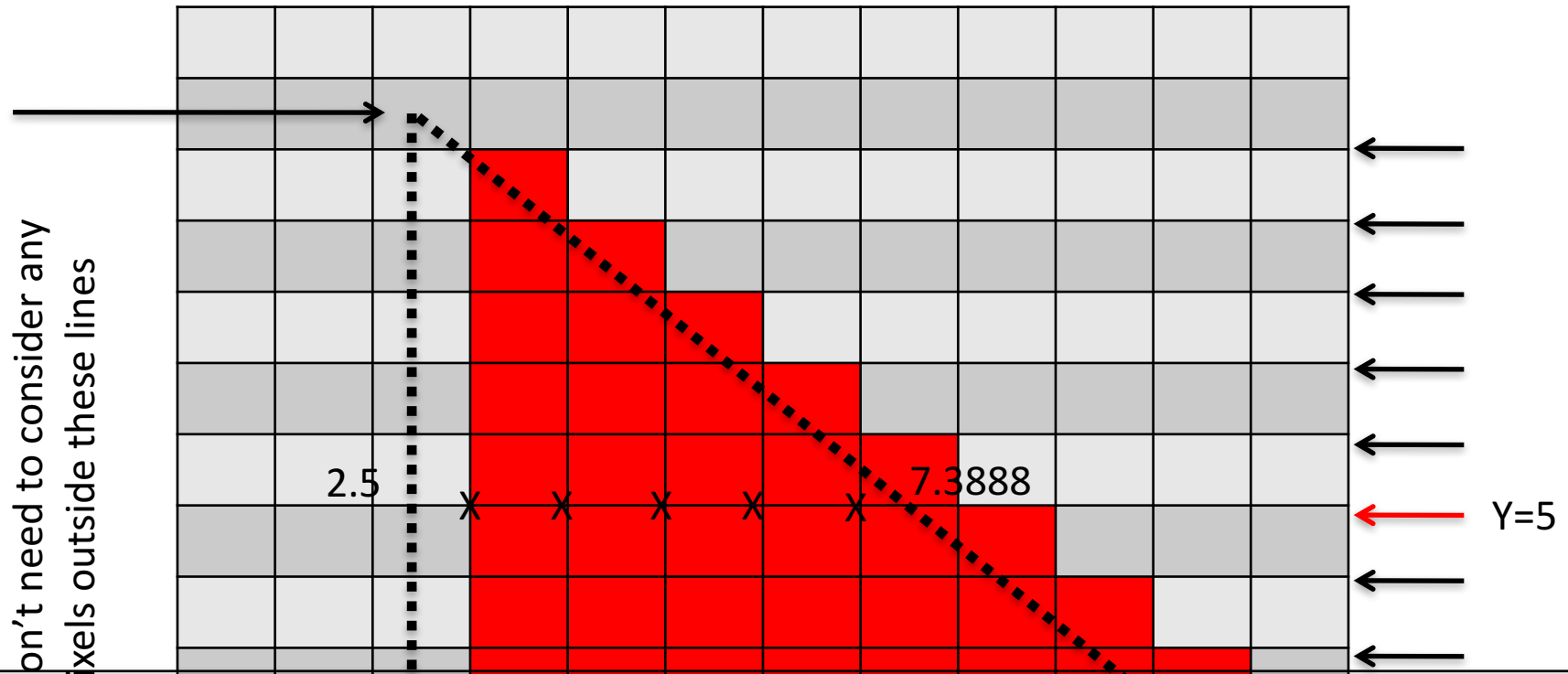
What are the end points?



Scanline algorithm: consider all rows that can possibly overlap



Scanline algorithm: consider all rows that can possibly overlap





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

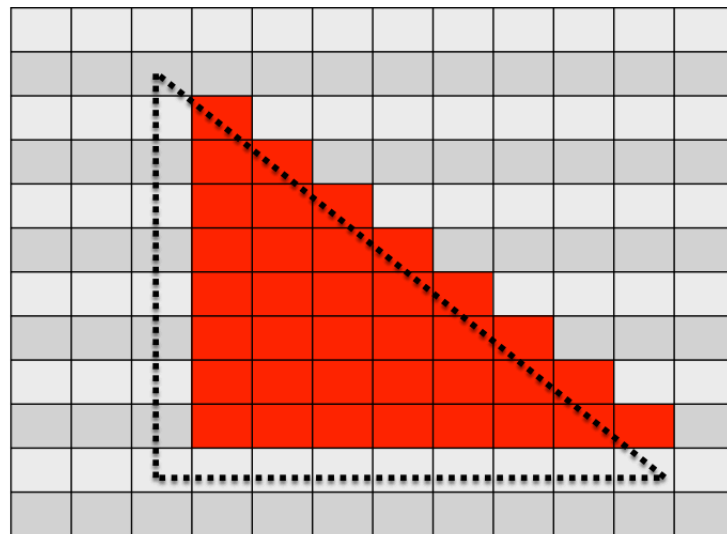
Scanline algorithm

- 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 $[\text{rowMin} \rightarrow \text{rowMax}]$; 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
 - $\text{ImageColor}(r, c) \leftarrow \text{triangle color}$

Scanline algorithm

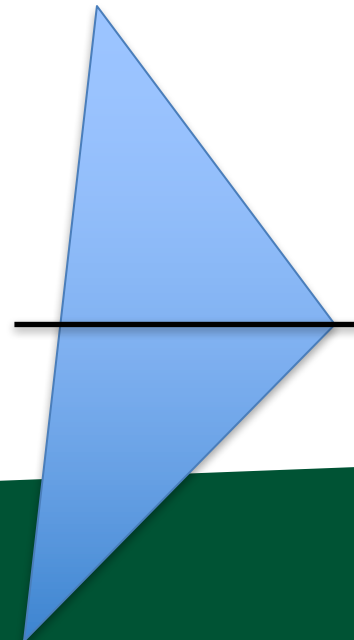
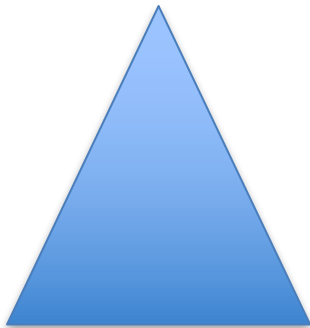
- Determine rows of pixels triangles can possibly intersect  Y values from 1.5 to 10.5 mean rows 2 through 10
- For r in $[\text{rowMin} \rightarrow \text{rowMax}]$; do
 - Find end points of r intersected with triangle
 - Call them leftEnd and rightEnd  For $r = 5$, $\text{leftEnd} = 2.5$, $\text{rightEnd} = 7.3888$
 - 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 ImageColor with
 $(5,3), (5,4), (5,5), (5,6), (5,7)$

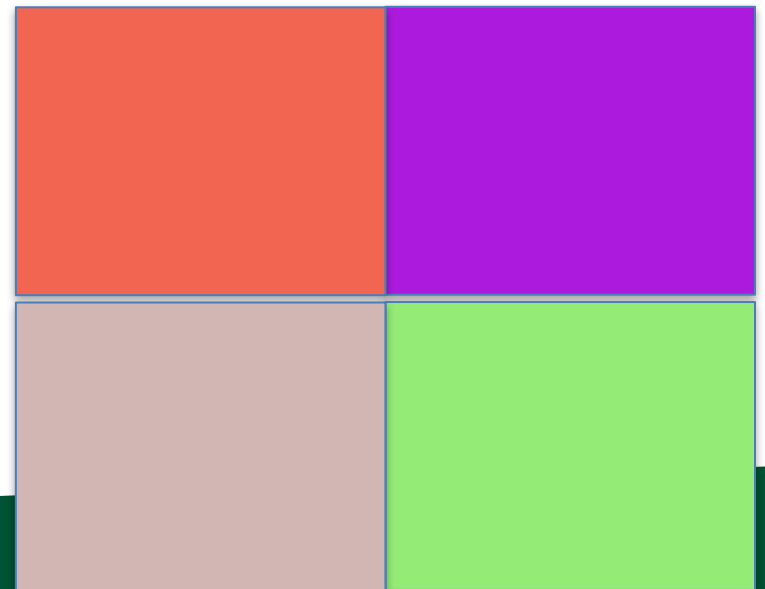
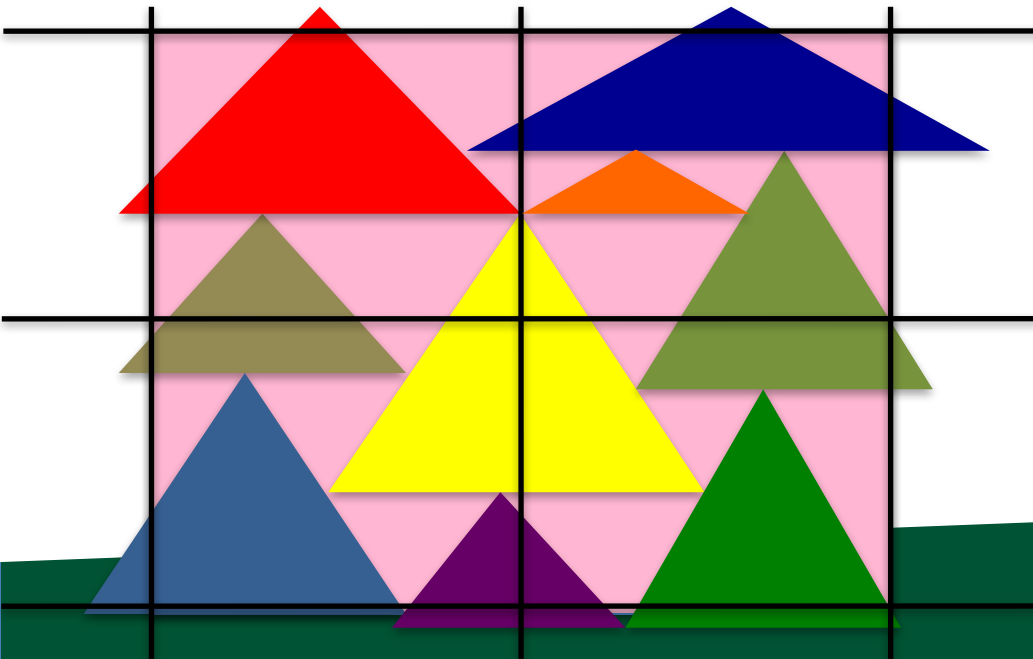


Arbitrary Triangles

- 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:

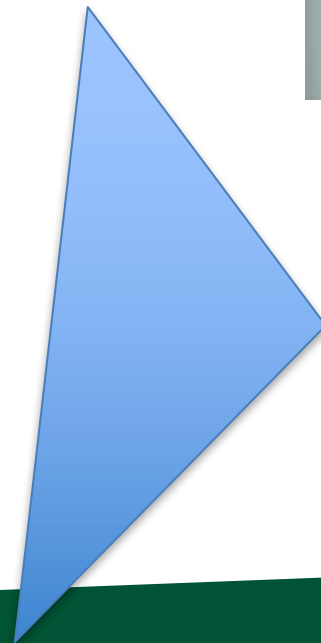
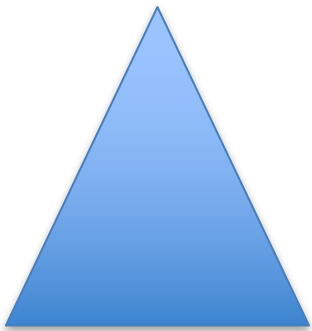
- 1) Vary colors (easy)
- 2) Deal with triangles that overlap



Project 1B

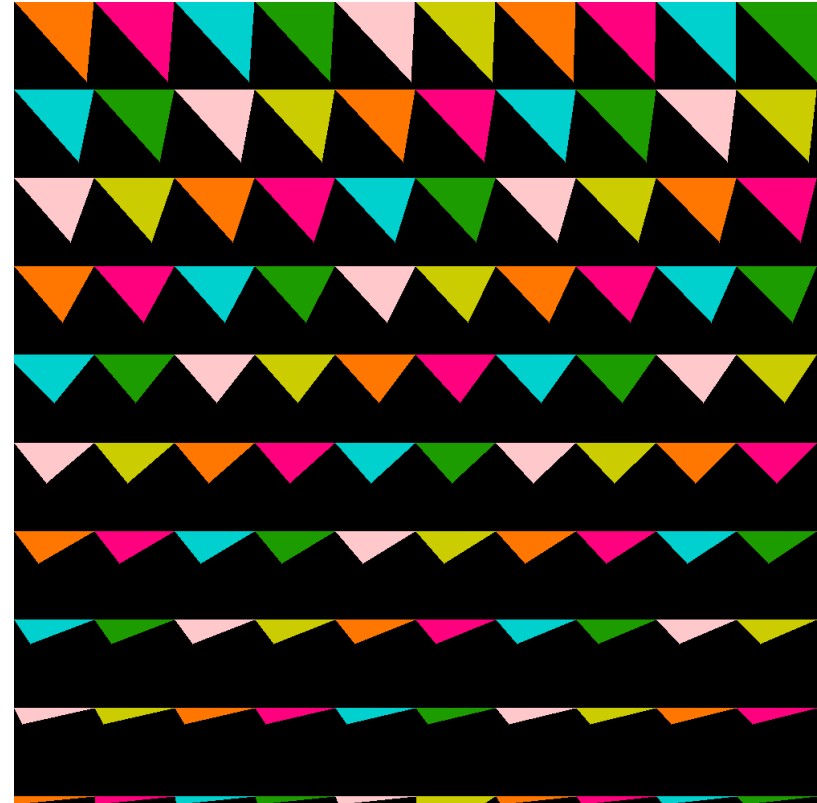
Arbitrary Triangles

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



Project #1B

- Goal: apply the scanline algorithm to “going down” triangles and output an image.
- File “project1B.cxx” has triangles defined in it.
- Due: Monday, Jan 14th
- % 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
187.99999999999999
```

Project 1B

- The limited accuracy of cerr/cout can cause other functions to 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
```

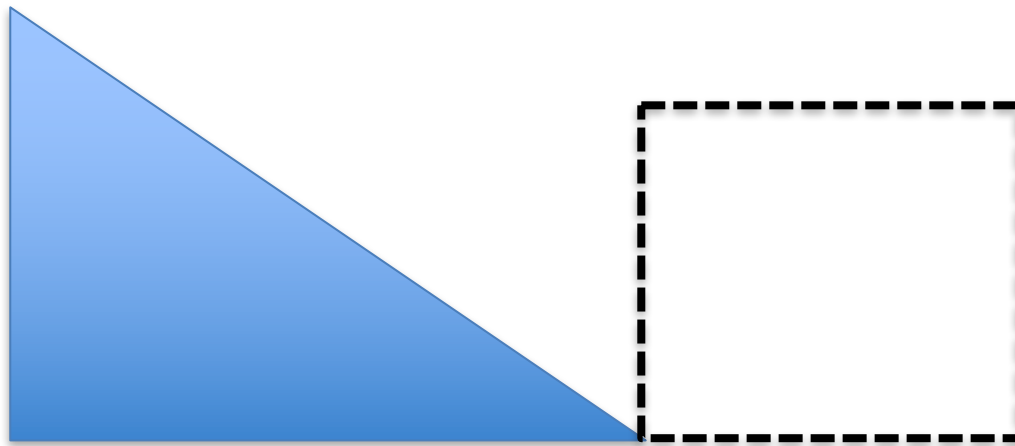


Project 1B

- 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: $(A+B)+C \neq A+(B+C)$

Project 1B

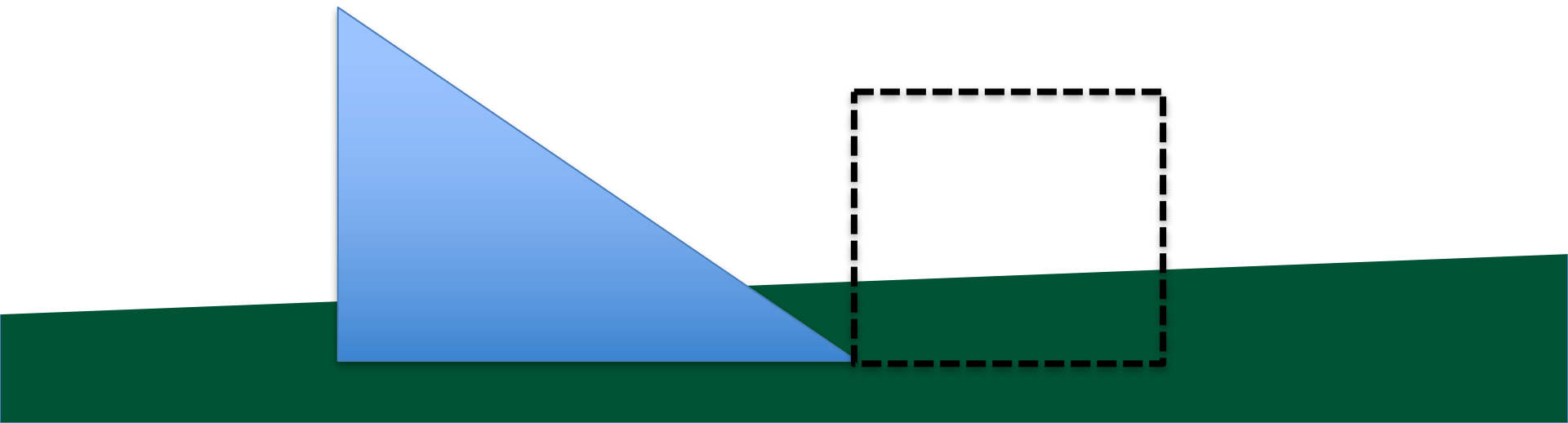
- For project 1B, we are making a binary decision for each pixel: should it be colored or not?
- Consider when a triangle vertex coincides with the bottom left of a pixel:



- 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 “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!