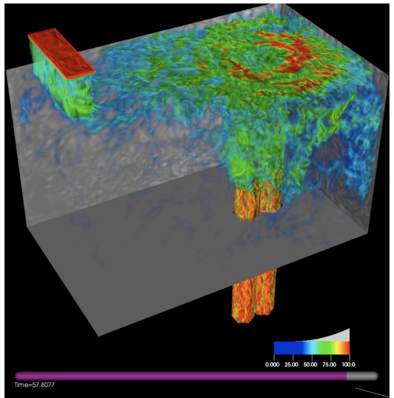
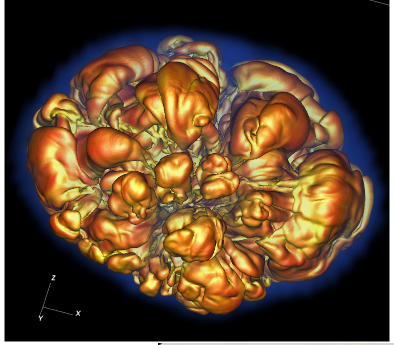
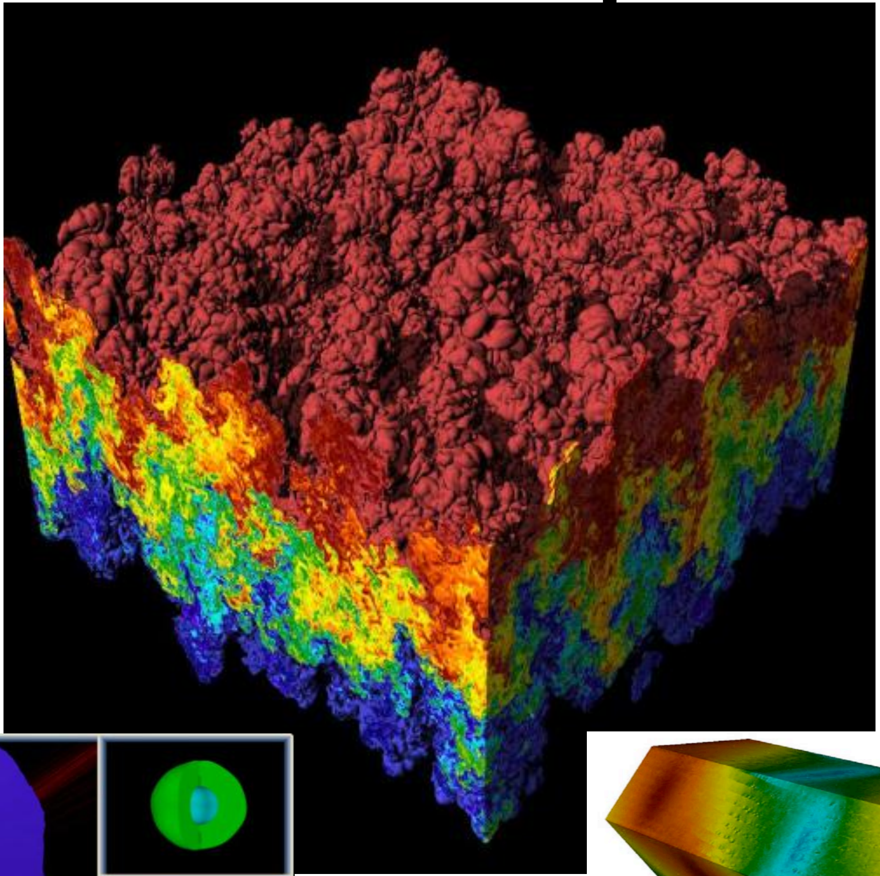
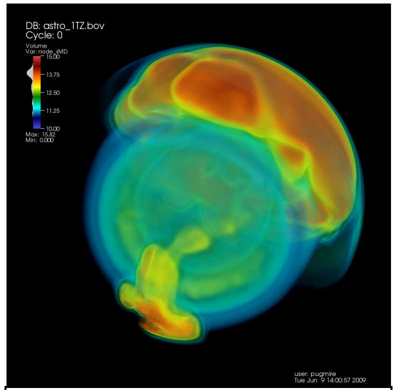
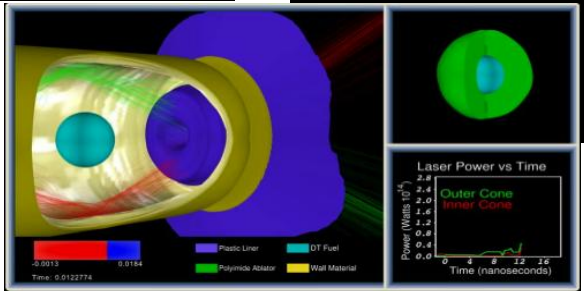
# CIS 441/541: Intro to Computer Graphics
# Lecture 4: Interpolation

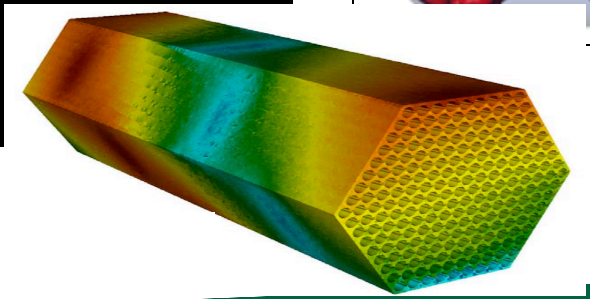












January 15, 2019

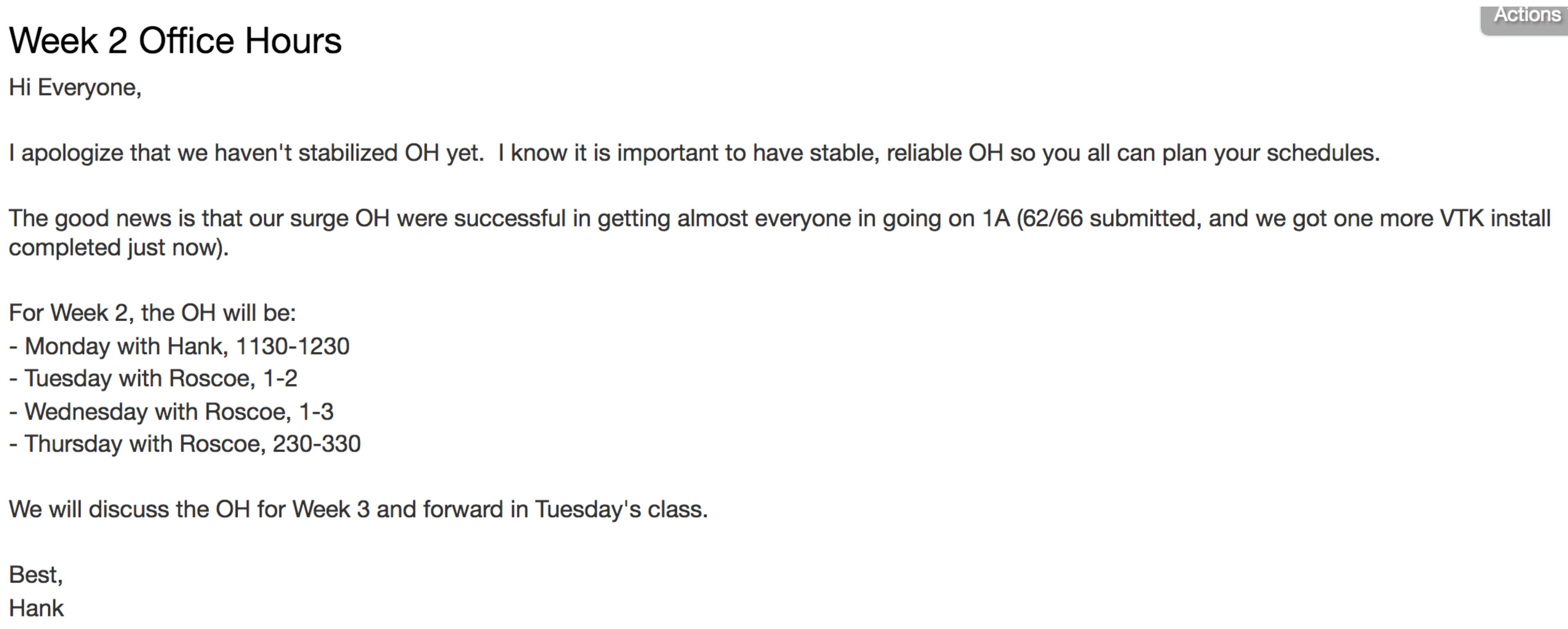
Hank Childs, University of Oregon

## Announcements

## No Class Thursday

- And no more YouTube lectures coming this week
  - We have what we need for 1C. I would prefer to time extra content for 1D/1E/1F around their due dates.

# Week 2 Office Hours



## Week 2 Office Hours

Hi Everyone,

I apologize that we haven't stabilized OH yet. I know it is important to have stable, reliable OH so you all can plan your schedules.

The good news is that our surge OH were successful in getting almost everyone in going on 1A (62/66 submitted, and we got one more VTK install completed just now).

For Week 2, the OH will be:
- Monday with Hank, 1130-1230
- Tuesday with Roscoe, 1-2
- Wednesday with Roscoe, 1-3
- Thursday with Roscoe, 230-330

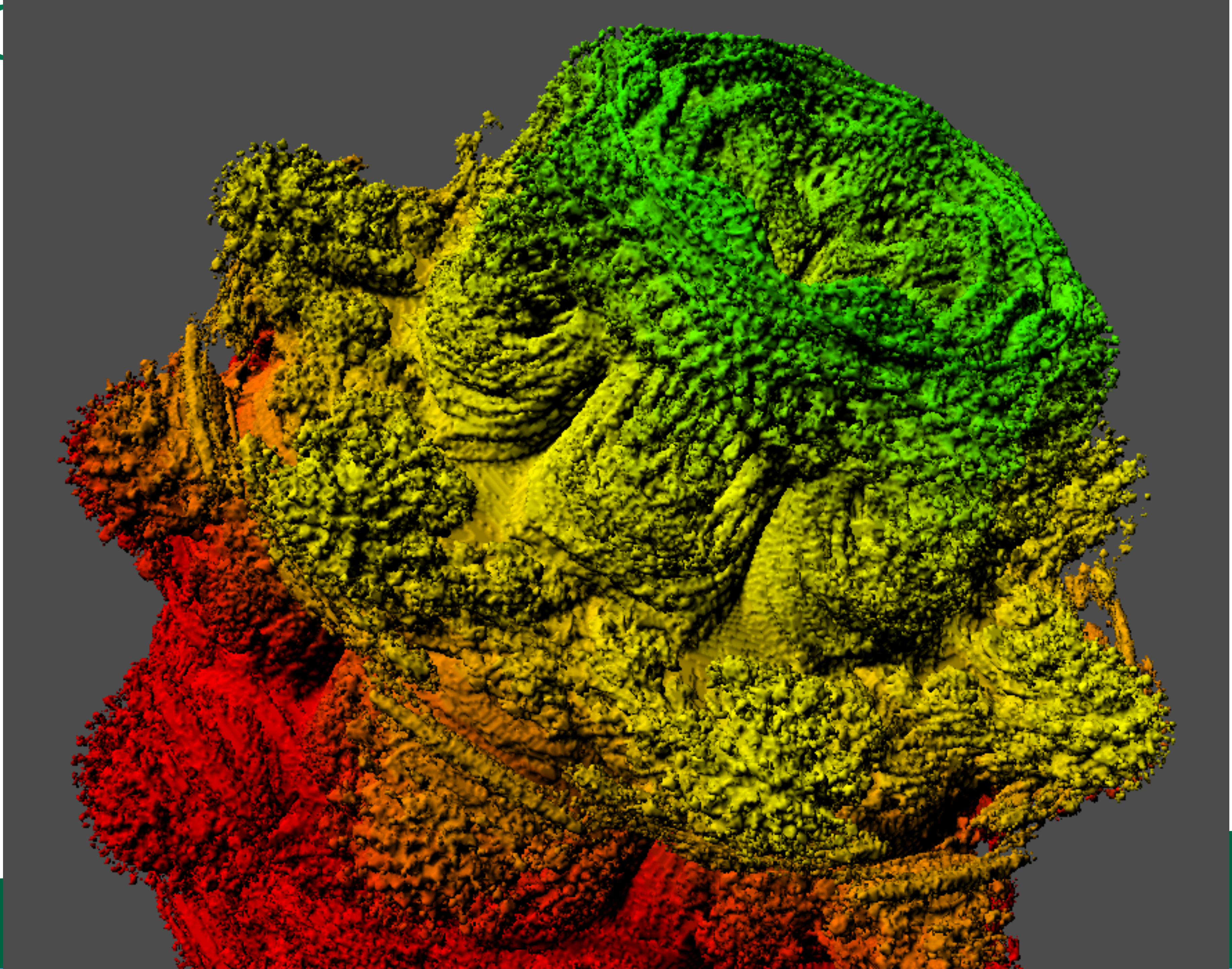
We will discuss the OH for Week 3 and forward in Tuesday's class.

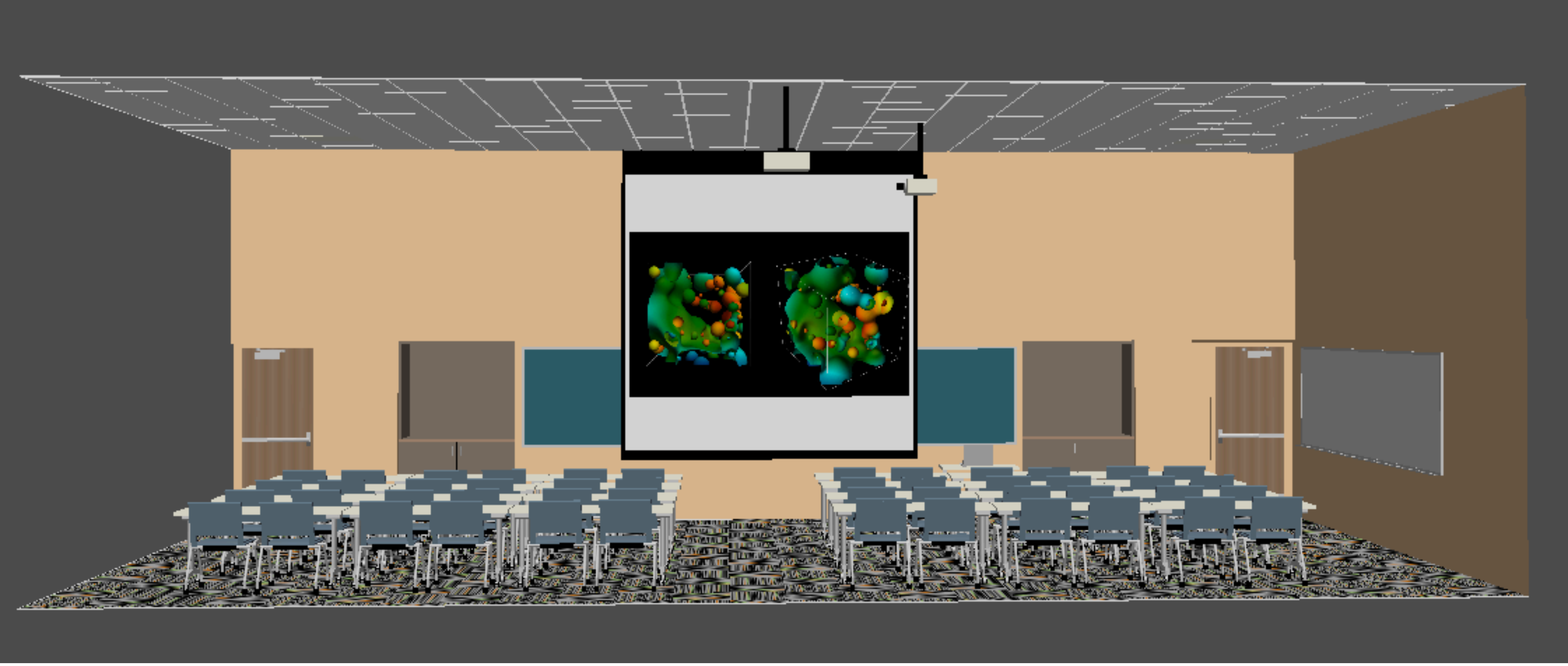
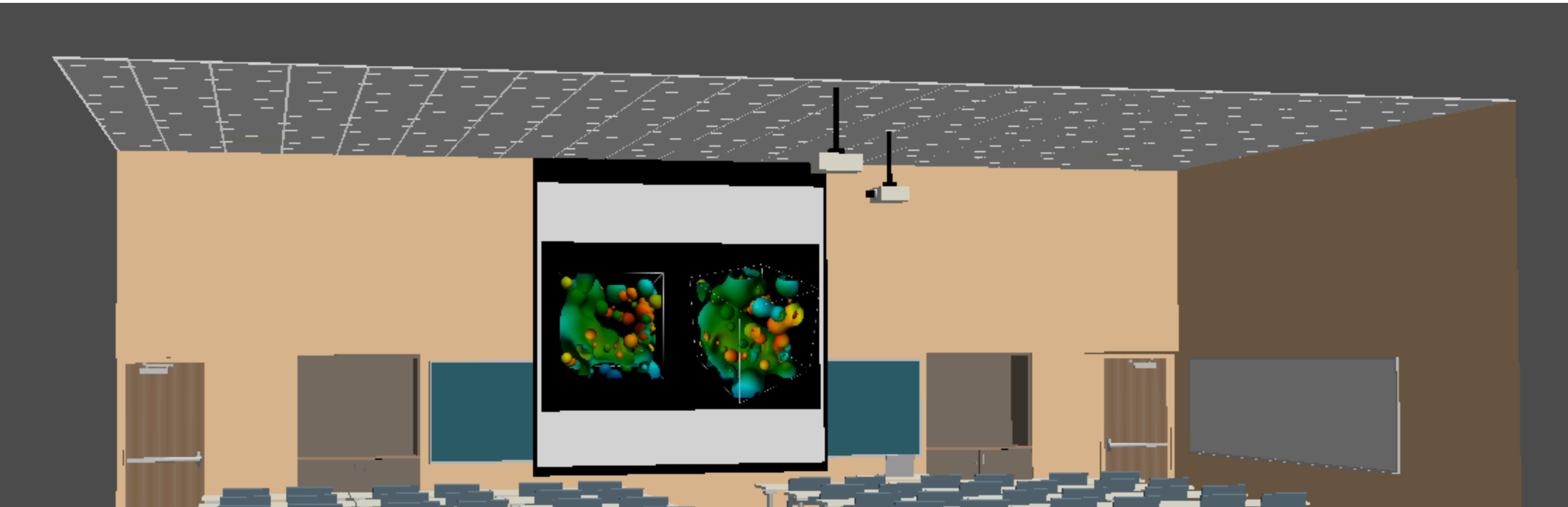
Best,
Hank

## Office Hours: Weeks 3-10

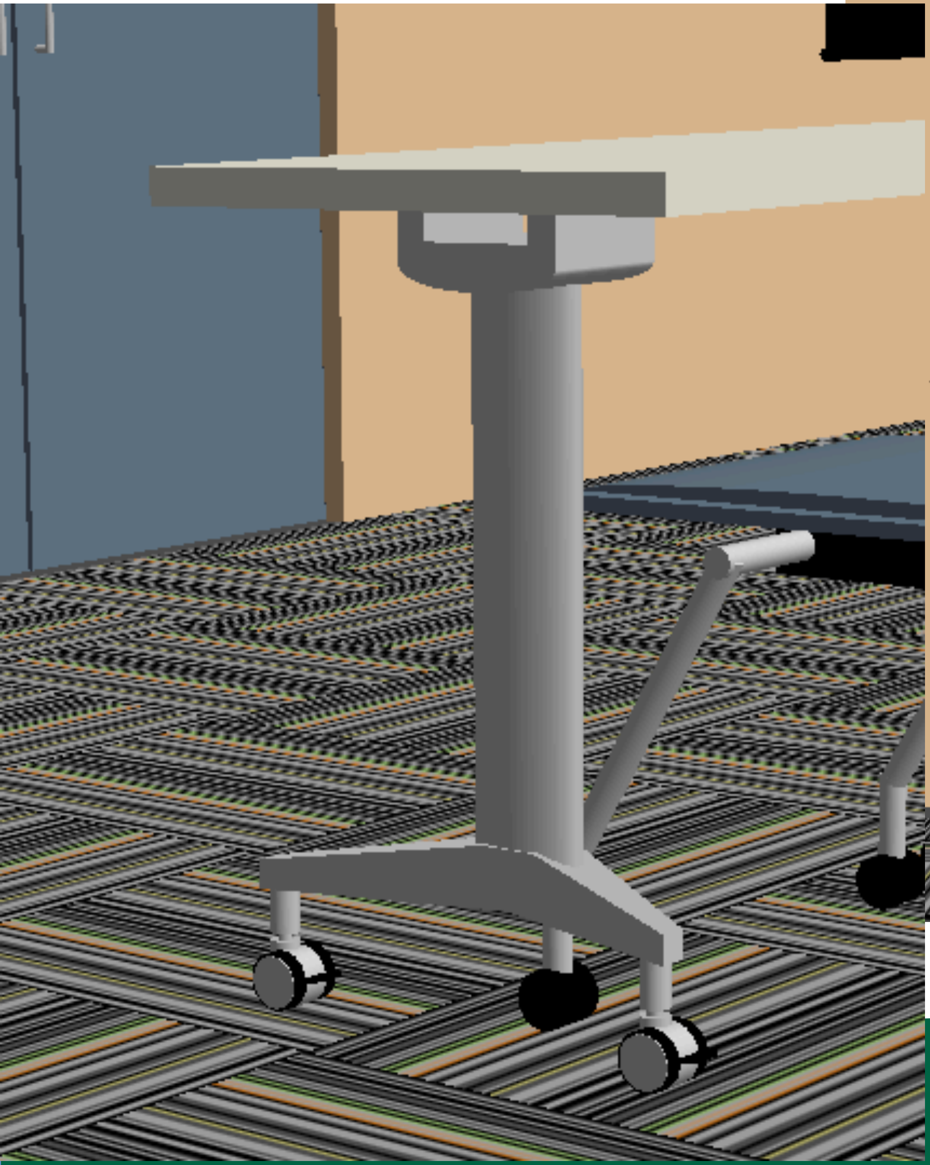
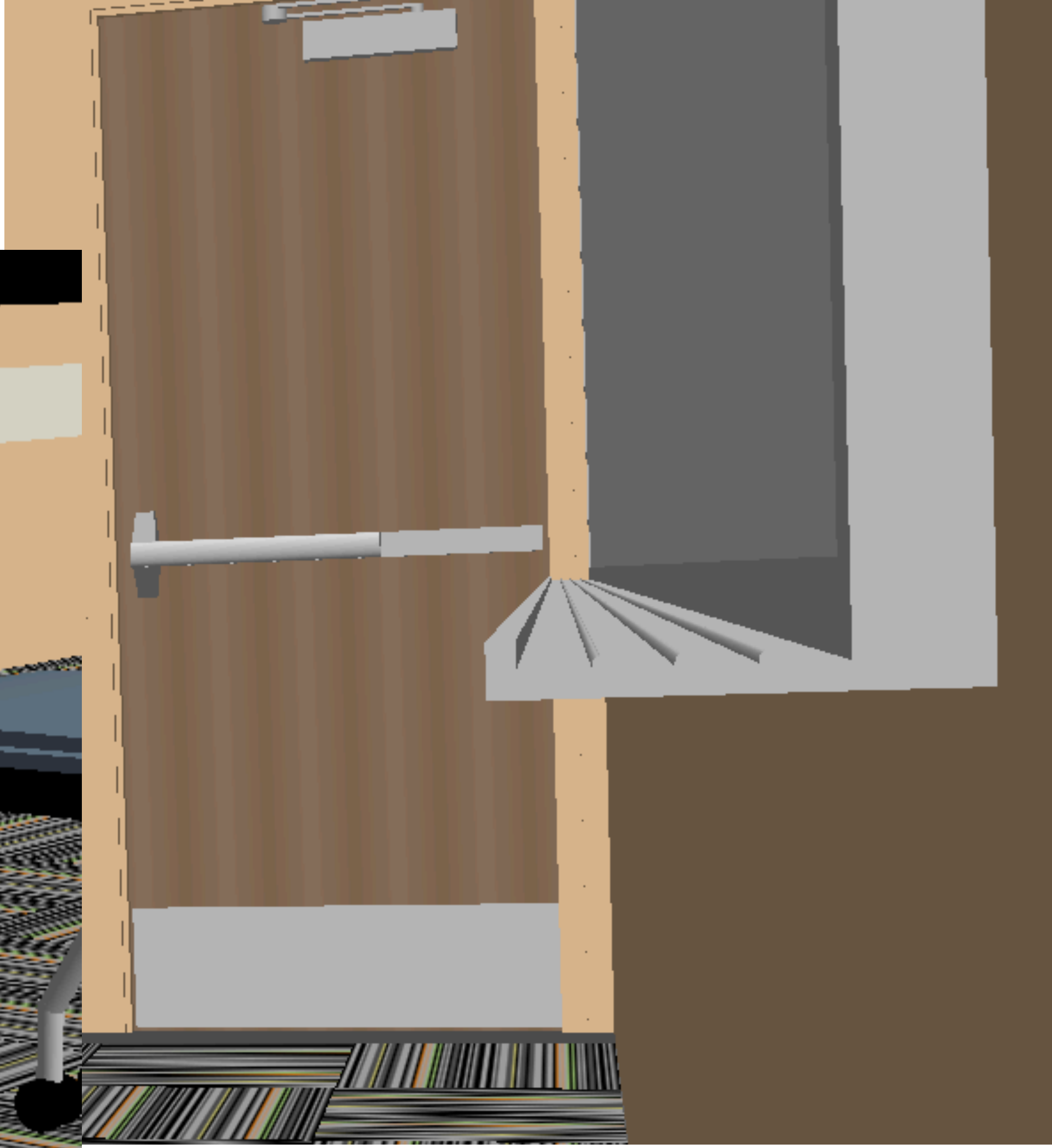
- Monday: 1-2 (Roscoe)
- Tuesday: 1-2 (Roscoe)
- Wednesday: 1-3 (Roscoe)
- Thursday: WHEN? (Hank)
- Friday: WHEN? (Hank)

## Some Sample Final Projects



UNIVERSITY OF OREGON

## Quick Review

## What Are We Rendering?

- Models made up of polygons
- Usually triangles
- Lighting tricks make surfaces look non-faceted
- More on this later…

## NEW Slide

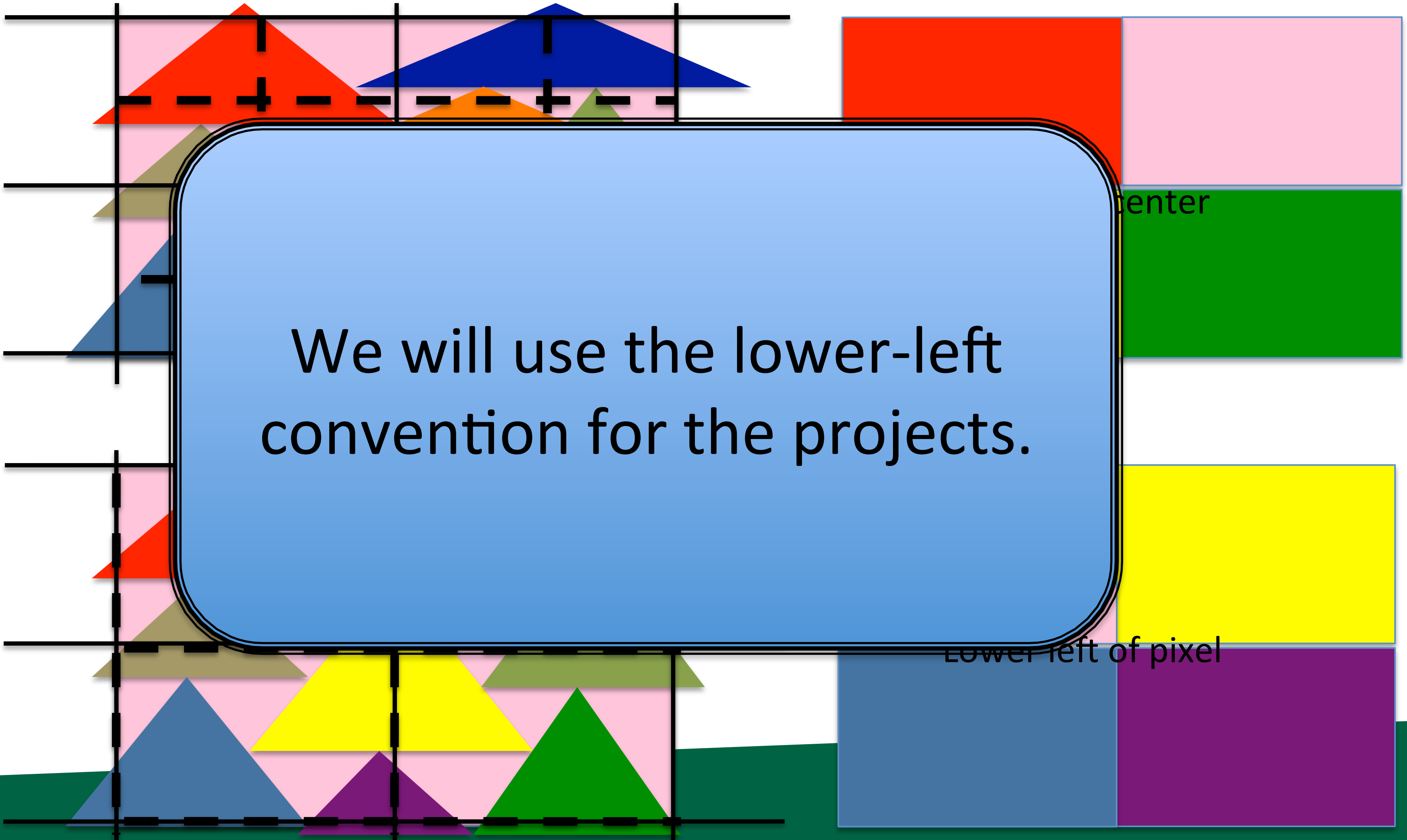
- Multiple coordinate spaces
- “World space”
  - Specify an origin and locations with respect to that origin
  - (x,y,z)
- “Screen space”
  - Everything relative to pixels on the screen
  - Triangle vertex (10.5, 20.5) lies in pixel (10, 20)

## NEW Slide

- Later, we will figure out how to:
  - Define a camera position
  - Transform triangle vertices from world space to screen space
  - Currently: assuming the transform has happened, and operating on triangle vertices already in screen space

UNIVERSITY OF OREGON

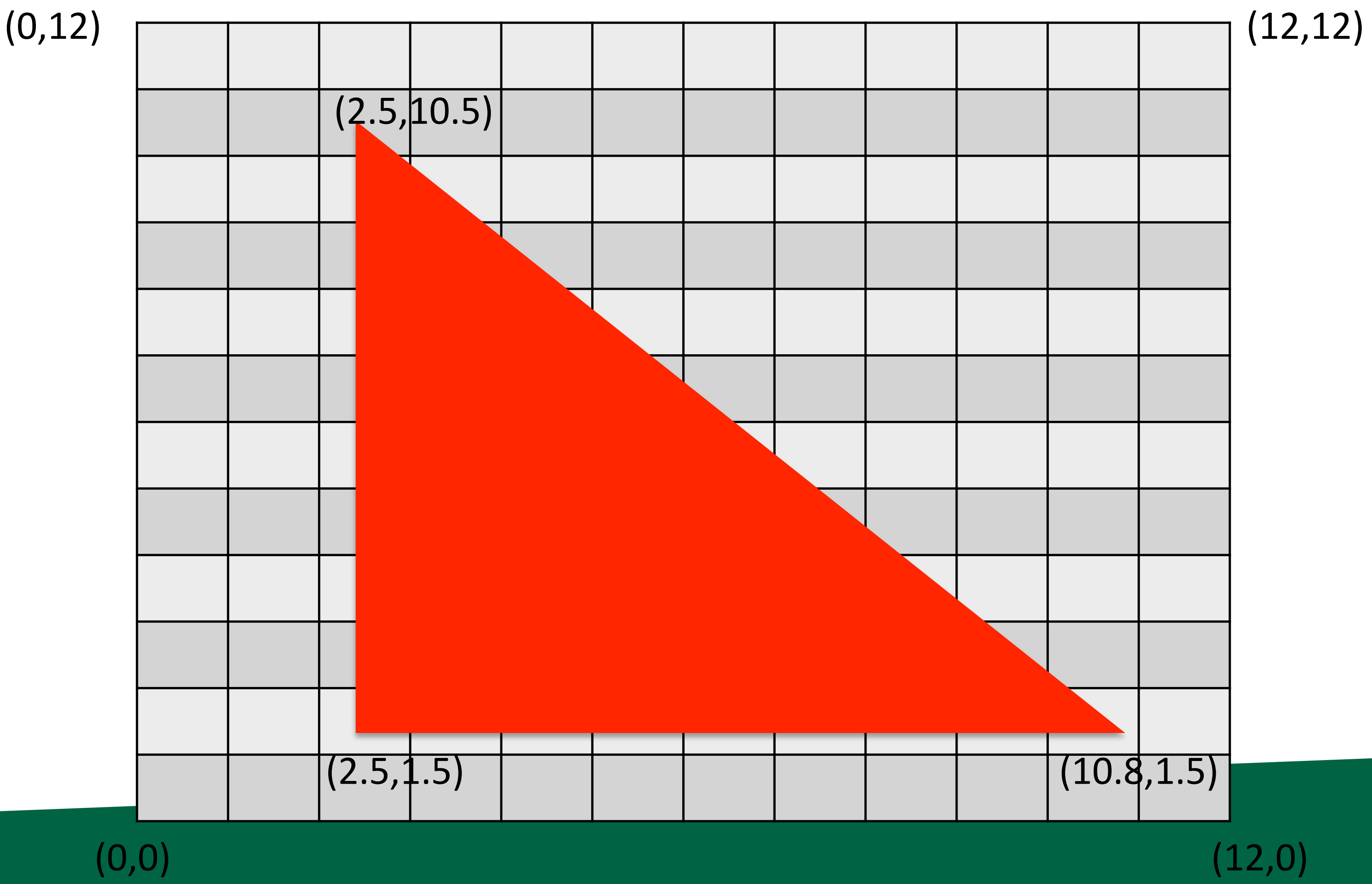
## The middle and lower-left variants are half-pixel translations of the other



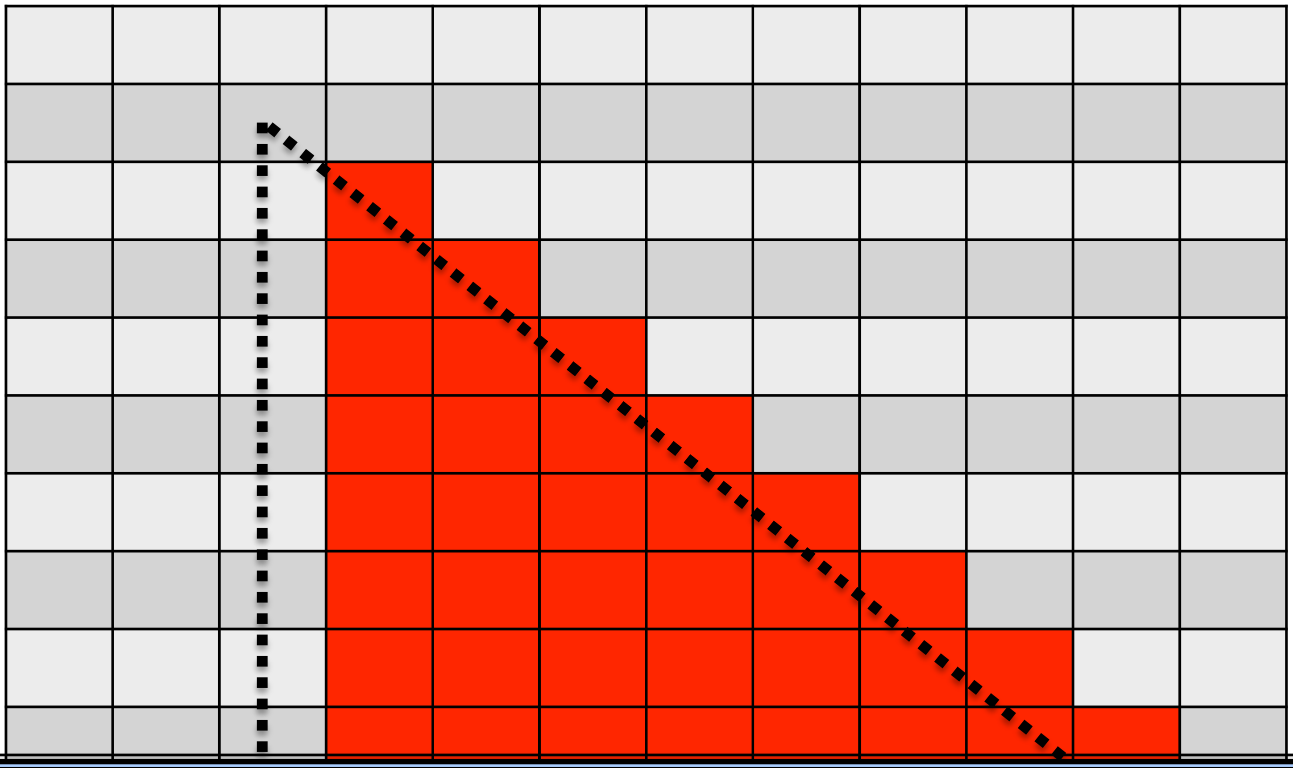
We will use the lower-left convention for the projects.

## 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)

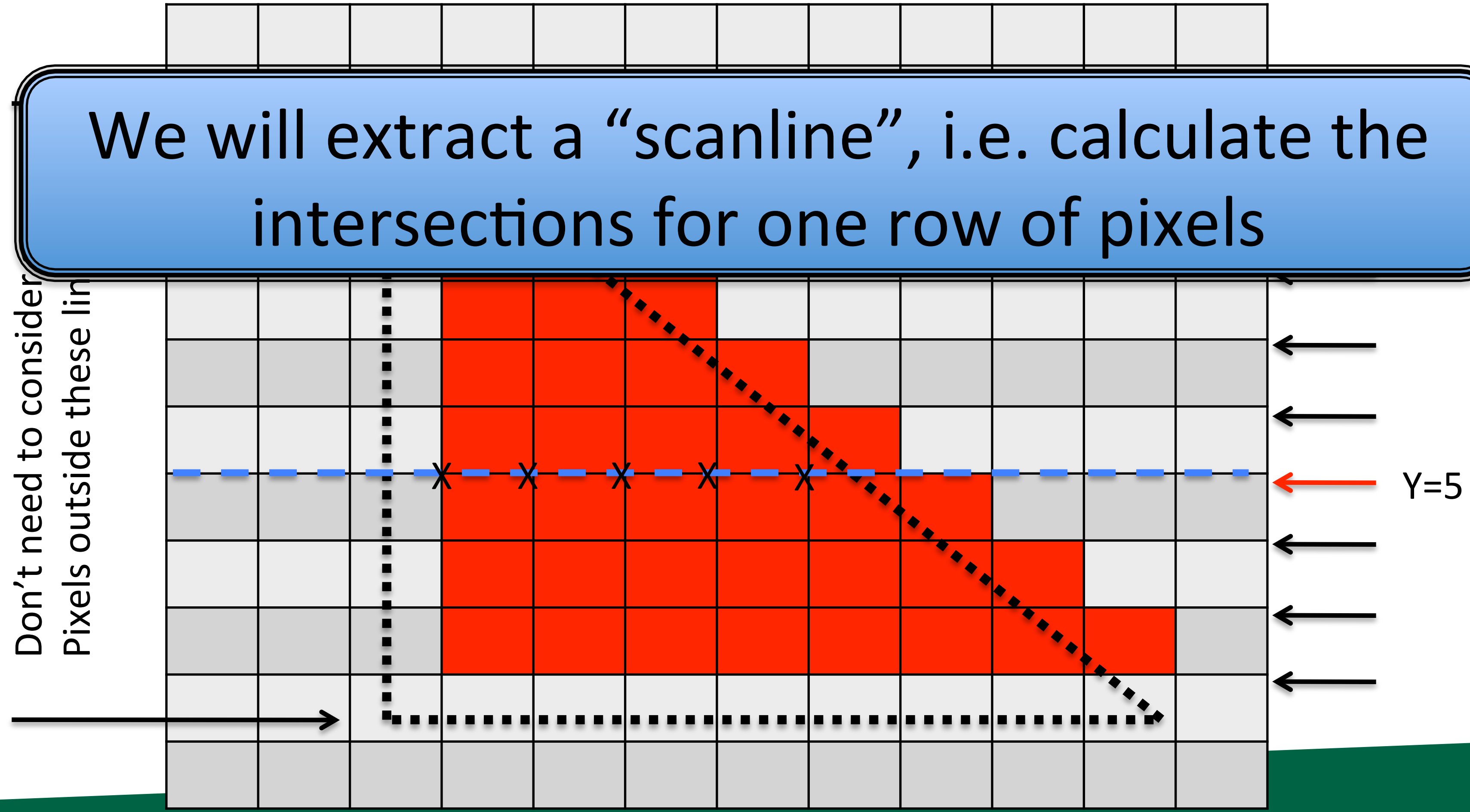
# Our desired output



How do we make this output? Efficiently?

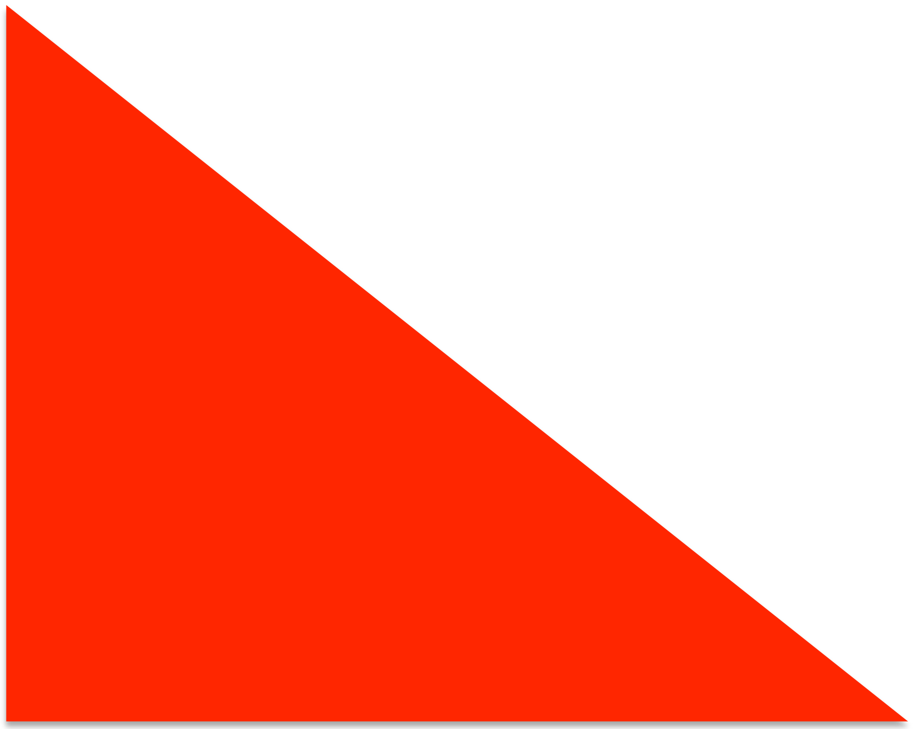
UNIVERSITY OF OREGON

## Scanline algorithm: consider all rows that can possibly overlap

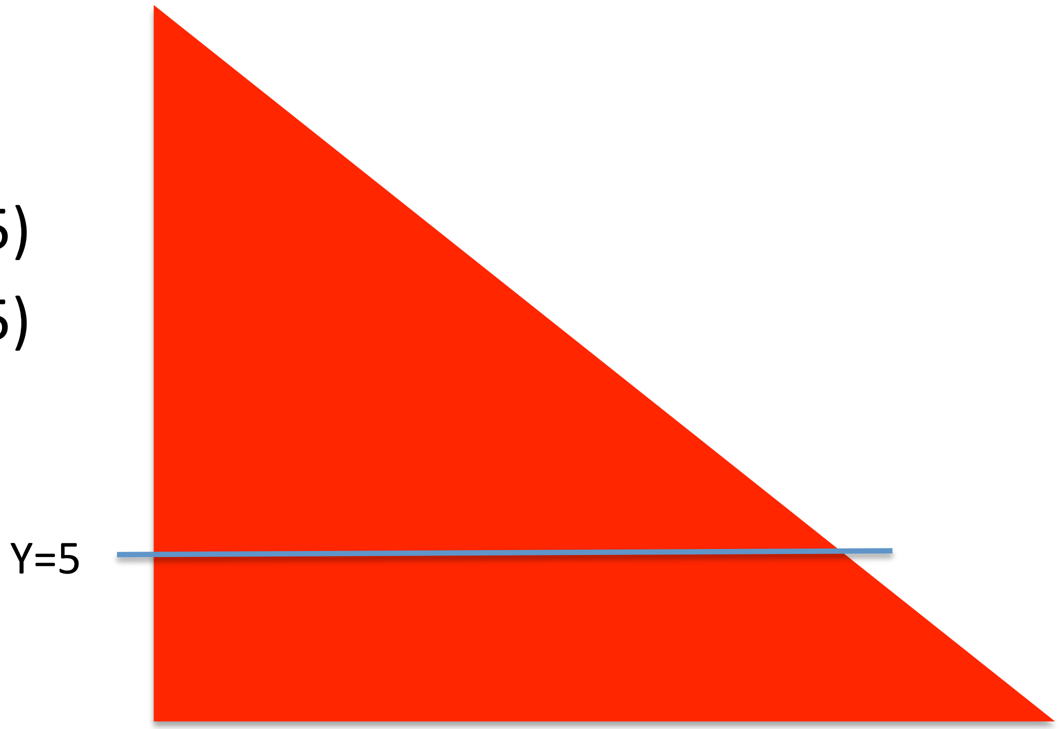


We will extract a "scanline", i.e. calculate the intersections for one row of pixels

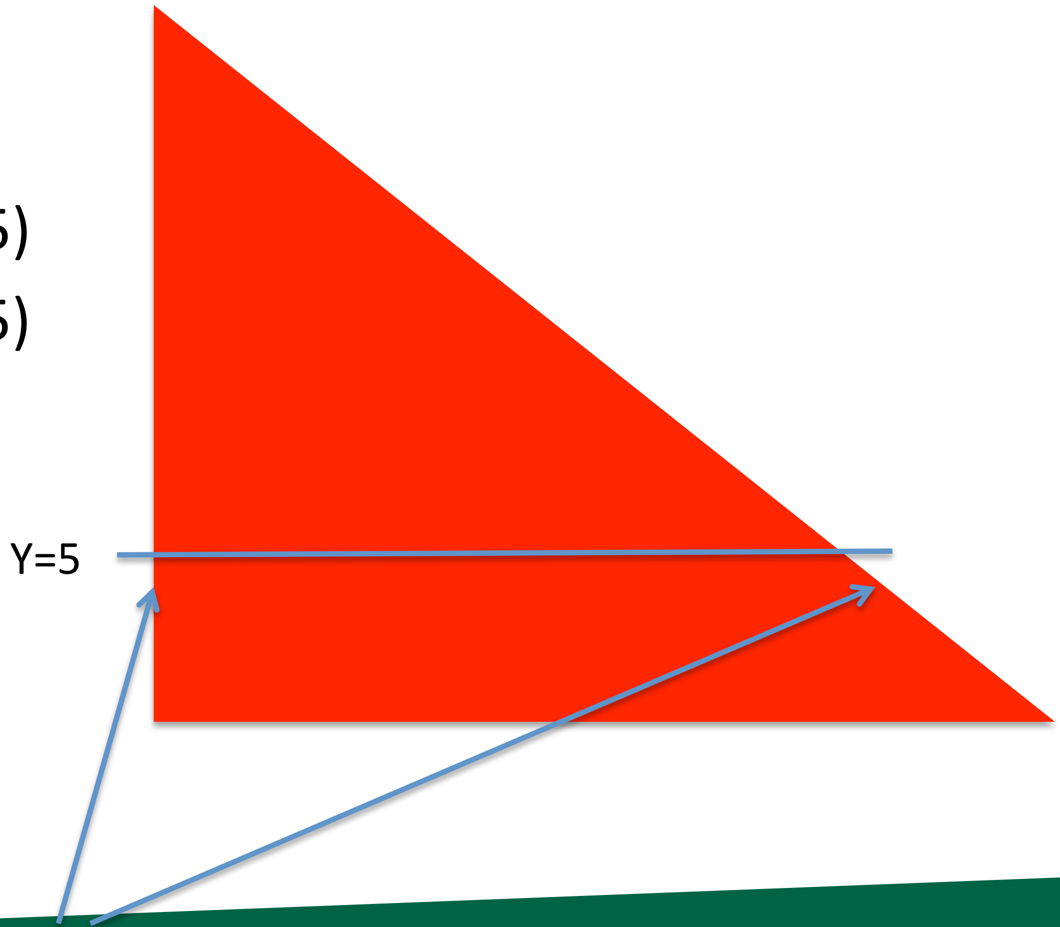
- Red triangle
  - Vertex 1: (2.5, 1.5)
  - Vertex 2: (2.5, 10.5)
  - Vertex 3: (10.5, 1.5)



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

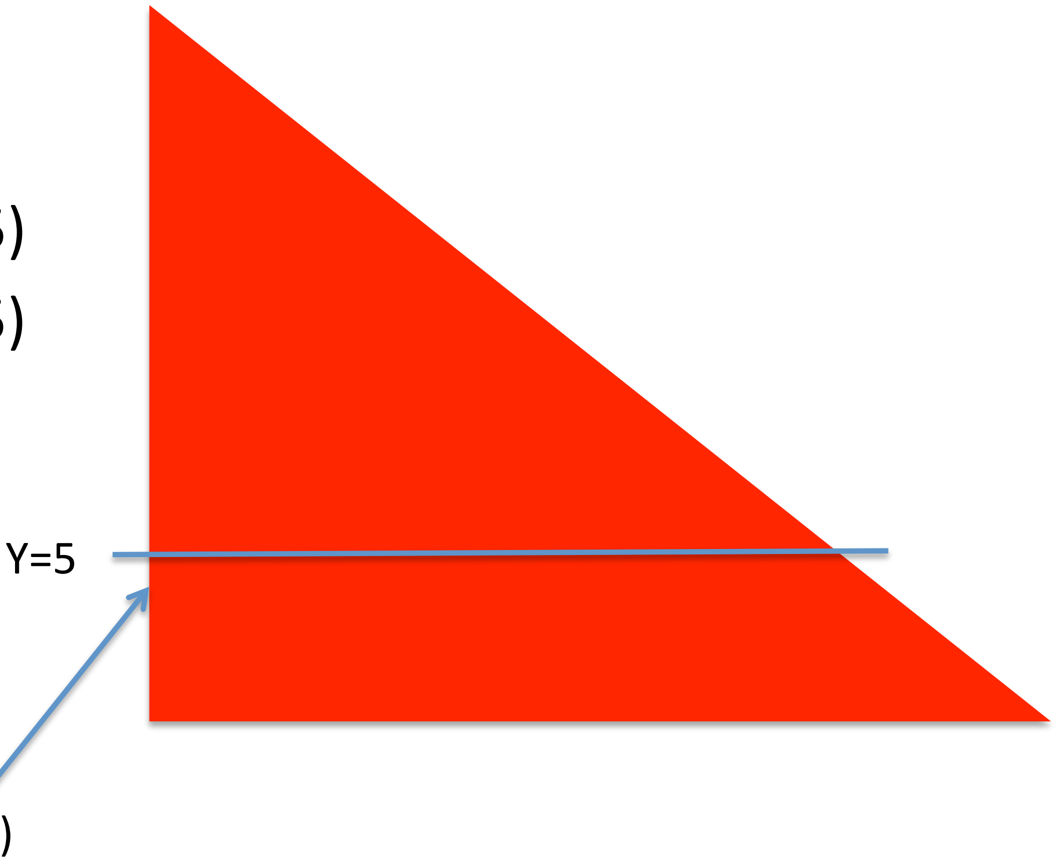


- Red triangle
  - Vertex 1: (2.5, 1.5)
  - Vertex 2: (2.5, 10.5)
  - Vertex 3: (10.5, 1.5)



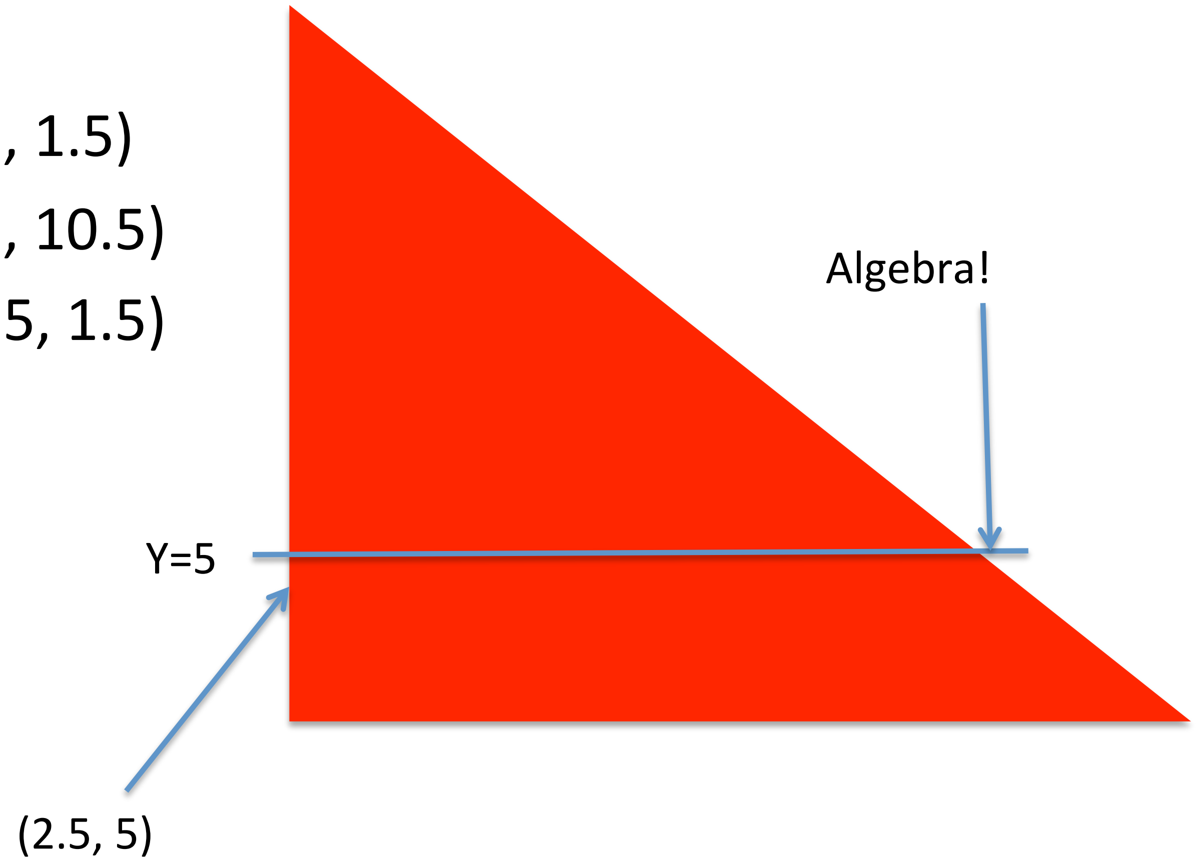
What are the end points?

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



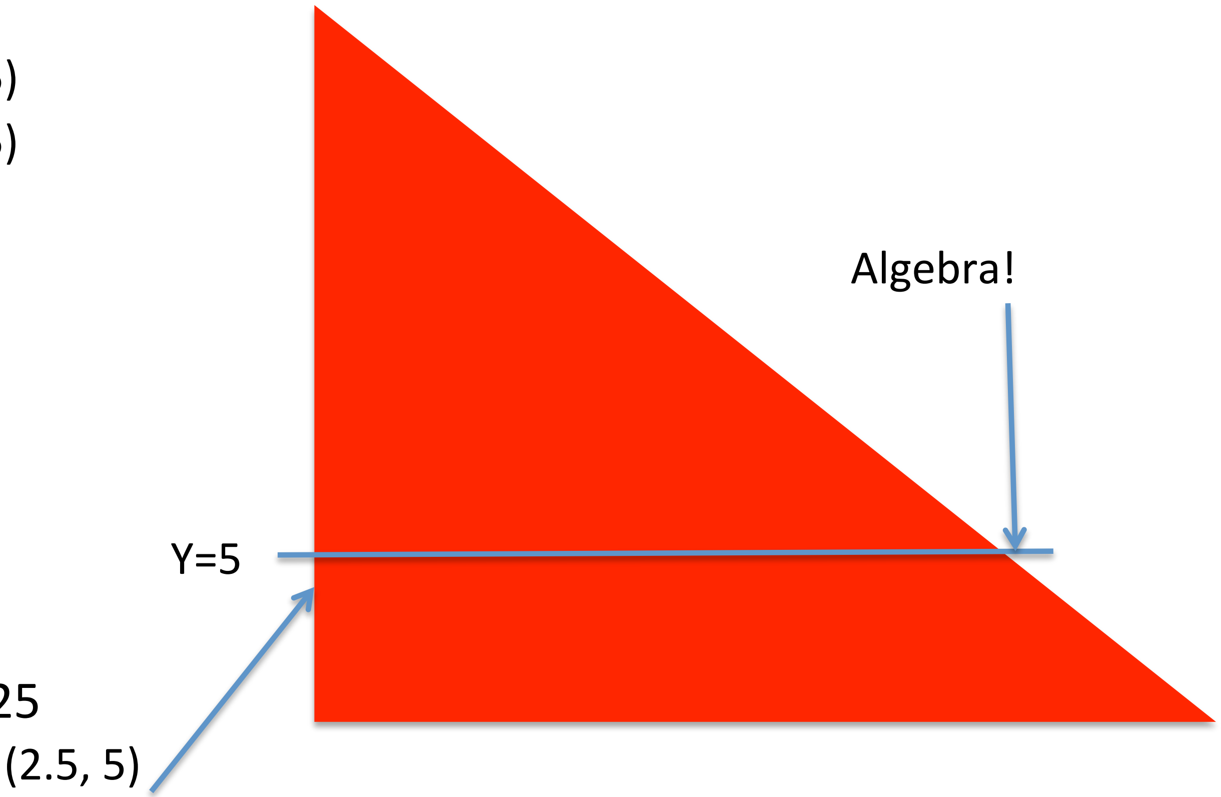
What are the end points?

- 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?

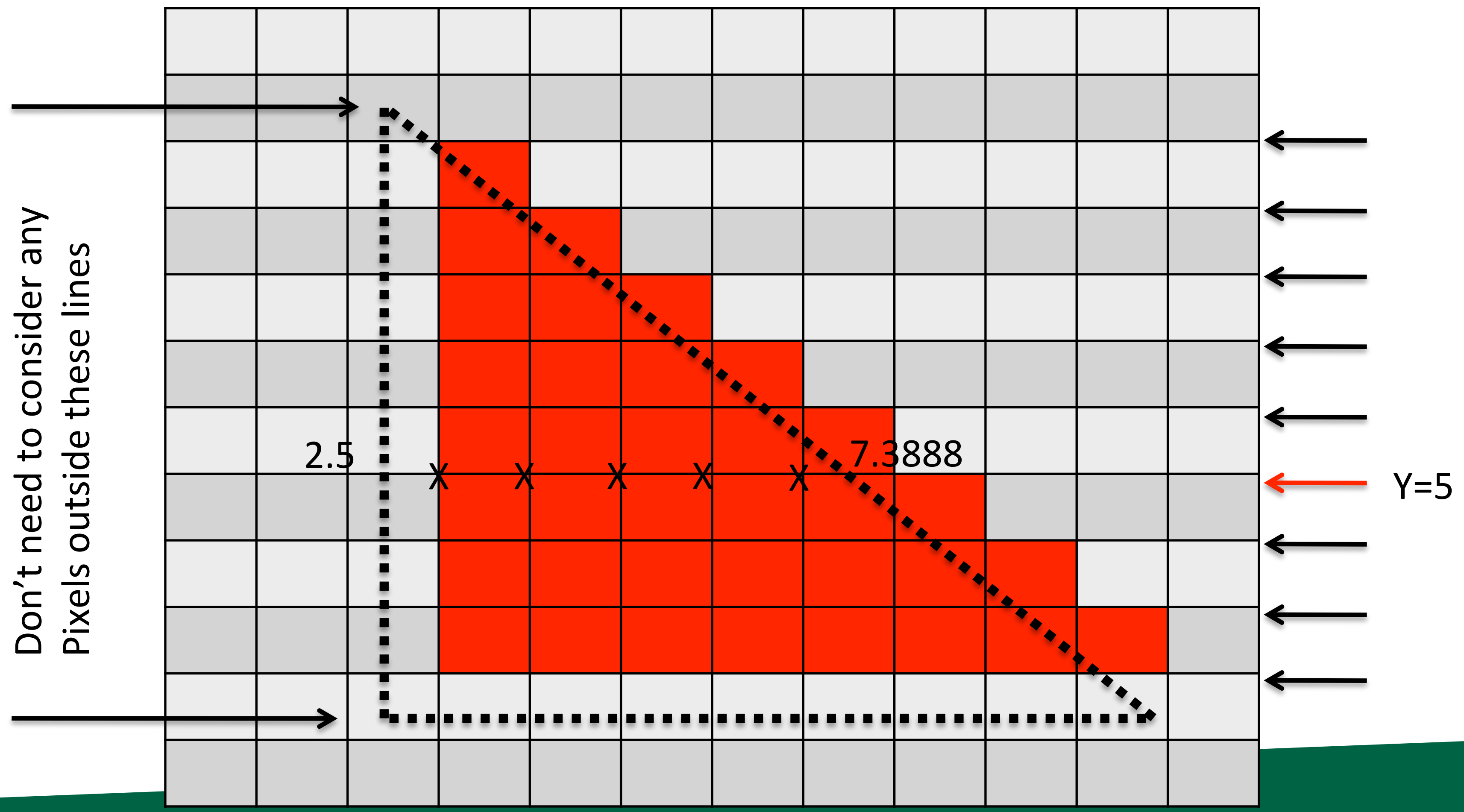
- Red triangle
  - Vertex 1: (2.5, 1.5)
  - Vertex 2: (2.5, 10.5)
  - Vertex 3: (10.5, 1.5)
- Y = mx+b
- 10.5=m*2.5+b
- 1.5 = m*10.5+b
- $\rightarrow$
- 9 = -8m
- m = -1.125
- b = 13.3125
- 5 = -1.125*x + 13.3125
- x = 7.3888



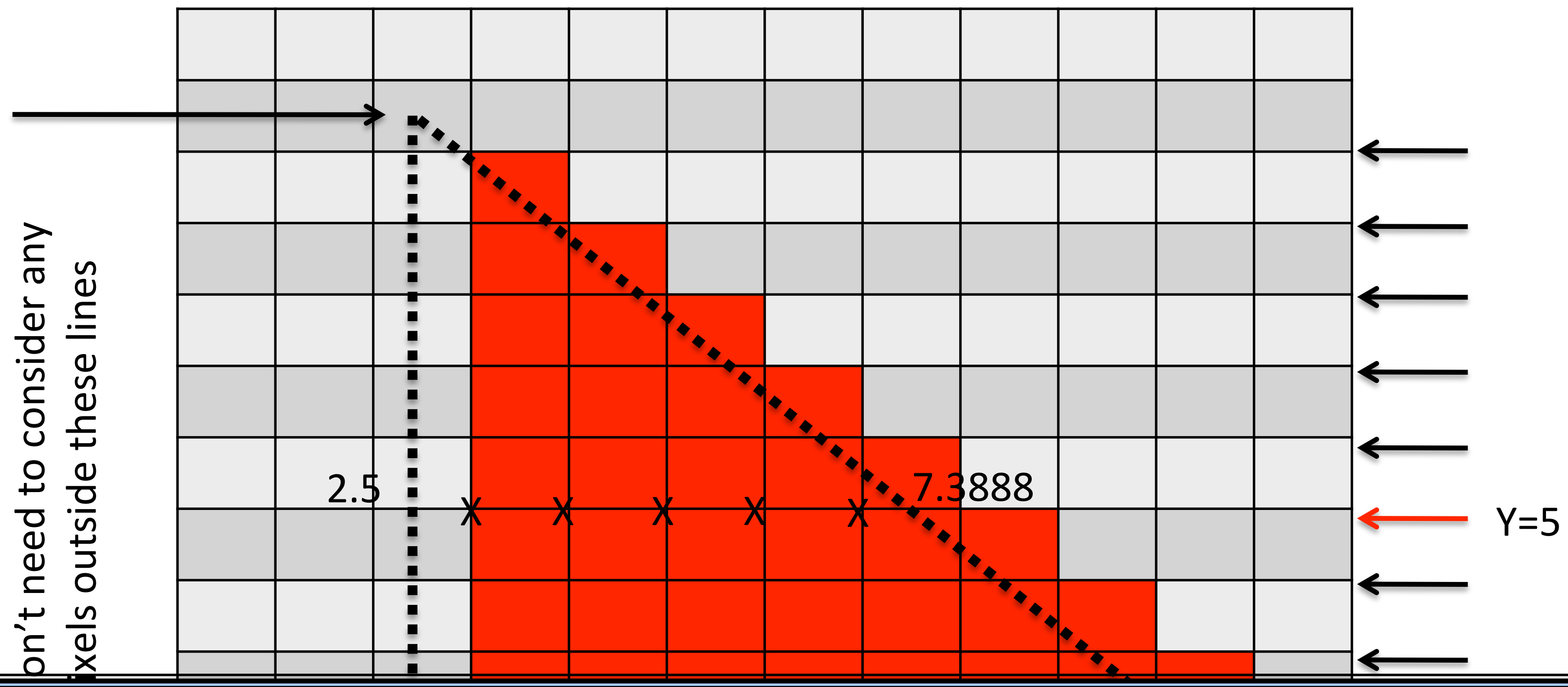
What are the end points?

UNIVERSITY OF OREGON

# 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 [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

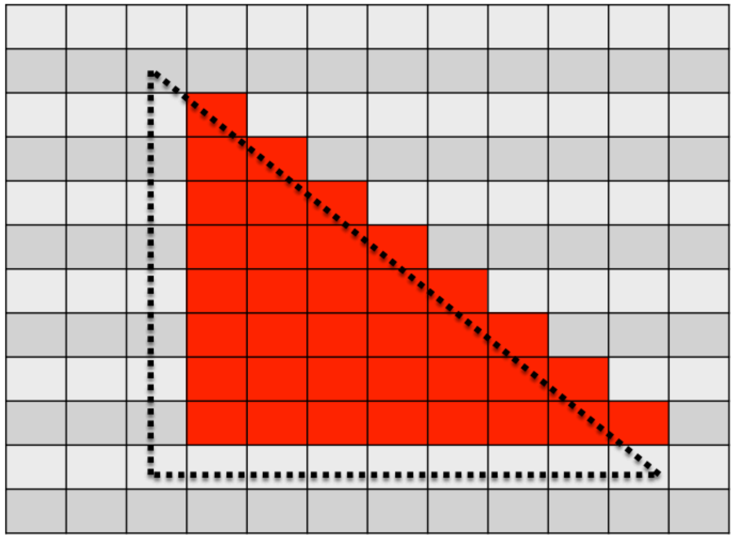
# Scanline algorithm

- Determine rows of pixels triangles can possibly intersect
- For r in [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

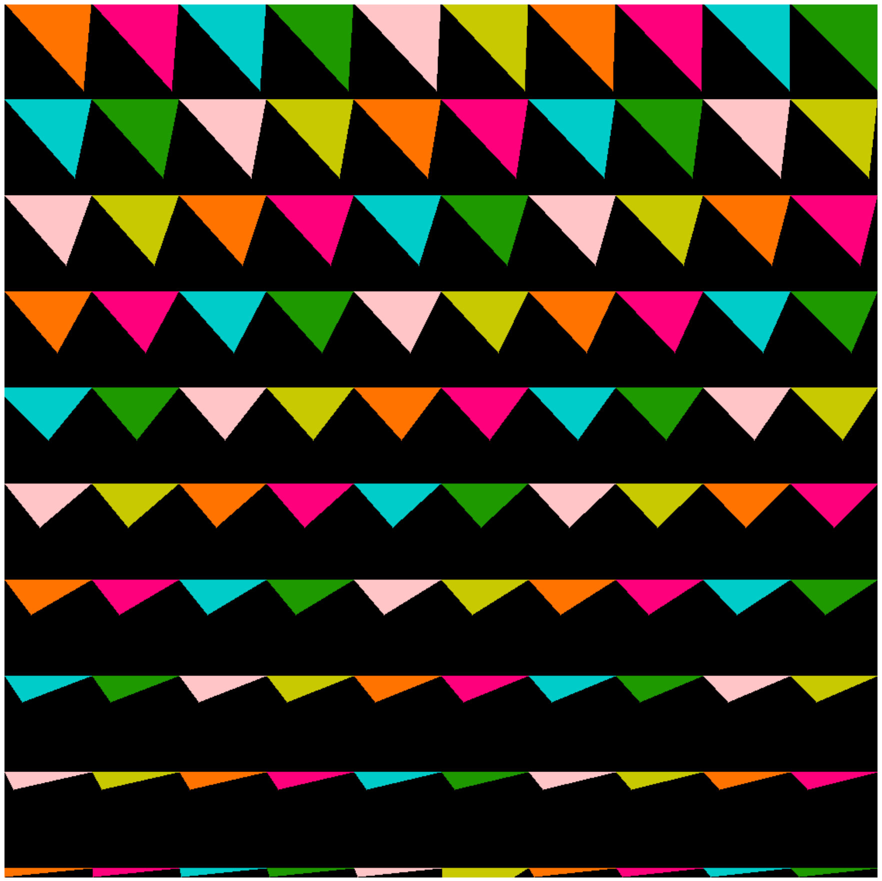
Y values from 1.5 to 10.5 mean rows 2 through 10

For r = 5, leftEnd = 2.5, rightEnd = 7.3888

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



# Project 1B (due last night): Questions?



CIS 441/541: Project #1B
Due January 14th, 2019 (meaning 6am Jan 15th)
Worth 3% of your grade

Instructions

1) Download and build project1B.cxx. Re-use your CMakeLists.txt from project1A, and just replace "1A" with "1B"
2) Project1B.cxx contains a routine that will generate 100 triangles.
    a. All of these triangles have two points with the same Y value and the third point with a lower Y value (i.e., "going down" triangles)
3) Implement the scanline algorithm to these triangles and fill up the image buffer with their colors.
4) The correct image (100triangles.png) is posted to the website.

When you are done, upload your code to Canvas.

If your code does not produce exactly the same image, you should expect to get less than half credit. You can confirm that it produces the same image with the difference program and the reference image on the website (100triangles.png).

Some notes:

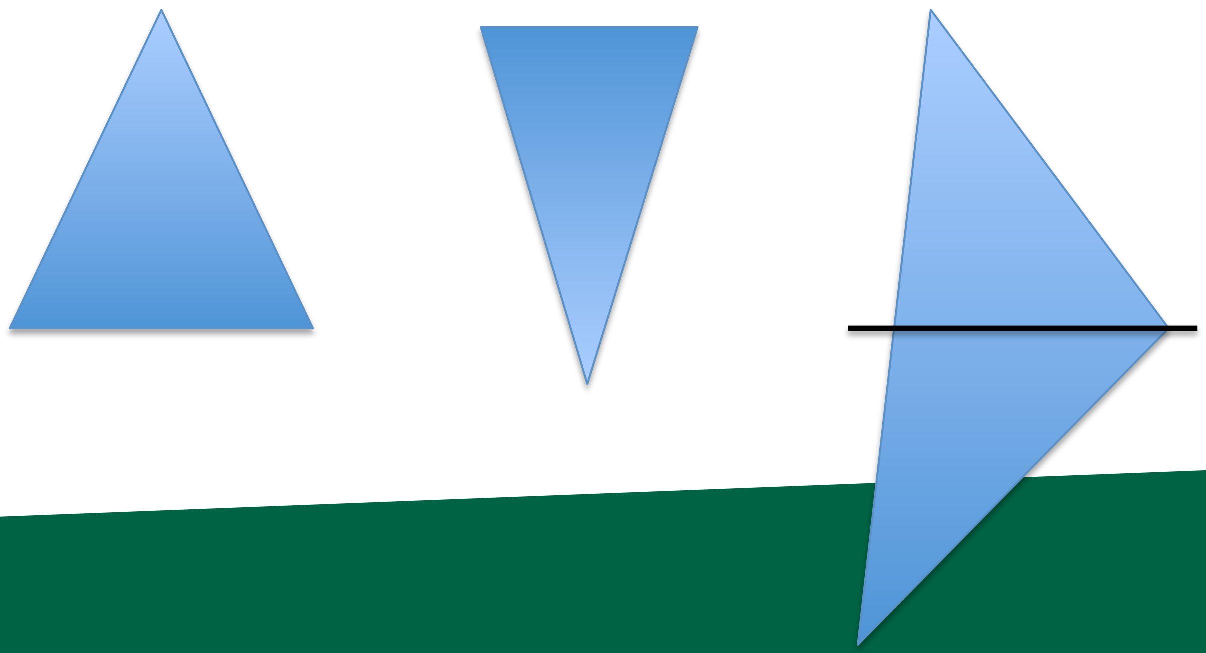
1) I began my implementation by figuring out which vertex was which, i.e., which was top-left, which was top-right, and which was bottom.
2) There are vertical lines, and they need special handling. Otherwise you get divide-by-zero in slope calculations.
3) Some pixels may be outside the screen. Plan for that.
4) Don't forget to use double precision and the floor_441 and ceil_441 functions.
5) What I printed when debugging:
    a. The triangle (I added a print method)
    b. Which vertices were which (top-left, top-right, bottom)
    c. The range of scanlines for a triangle
    d. ALSO: I sometimes modified the for loop to only do one triangle at a time, so it would be fewer print statements.

UNIVERSITY OF OREGON

## End Review

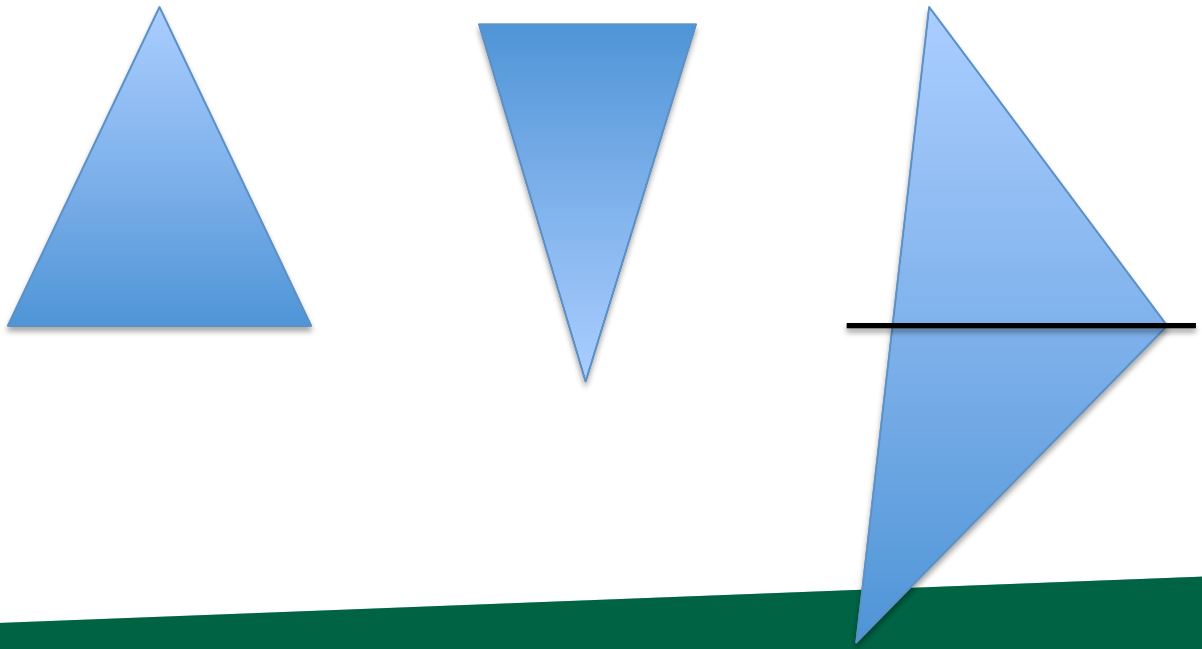
## Arbitrary Triangles

- The description of the scanline algorithm in the preceding slides is general.
- But the implementation for these three triangles vary:



# Arbitrary Triangles

- Project #1B: implement the scanline algorithm for “going down” triangles
- Project #1C: arbitrary triangles

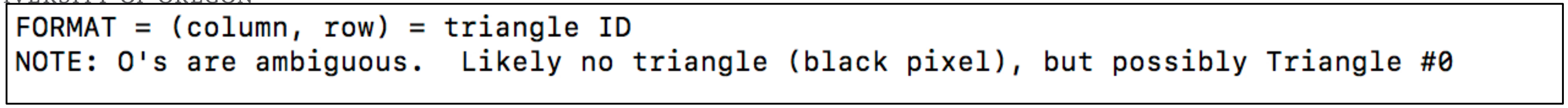


# Arbitrary Triangles

- Function: RasterizeGoingDownTriangle
  - (You have this from 1B)
- Function: RasterizeGoingUpTriangle
  - (You can write this by modifying RasterizeGoingDownTriangle)
- Function: RasterizeArbitraryTriangle
  - Split into two triangles
  - Call RasterizeGoingUpTriangle and RasterizeGoingDownTriangle

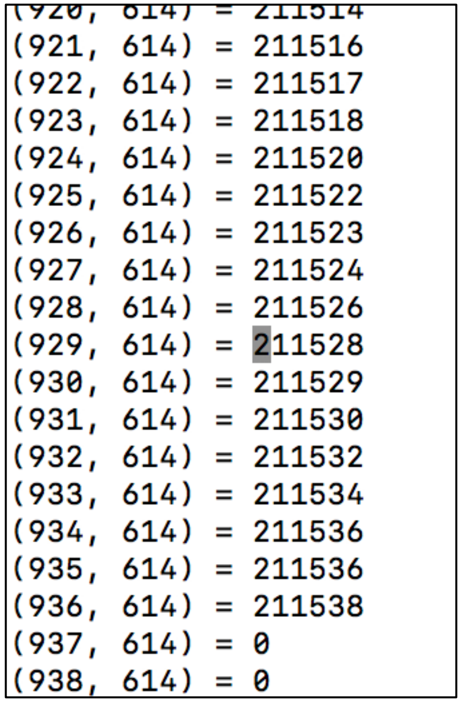
## Project #1C (6%), Due (Jan 23rd)

- 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 “reader.cxx” has code to read triangles from file.
- No Cmake, project1c.cxx



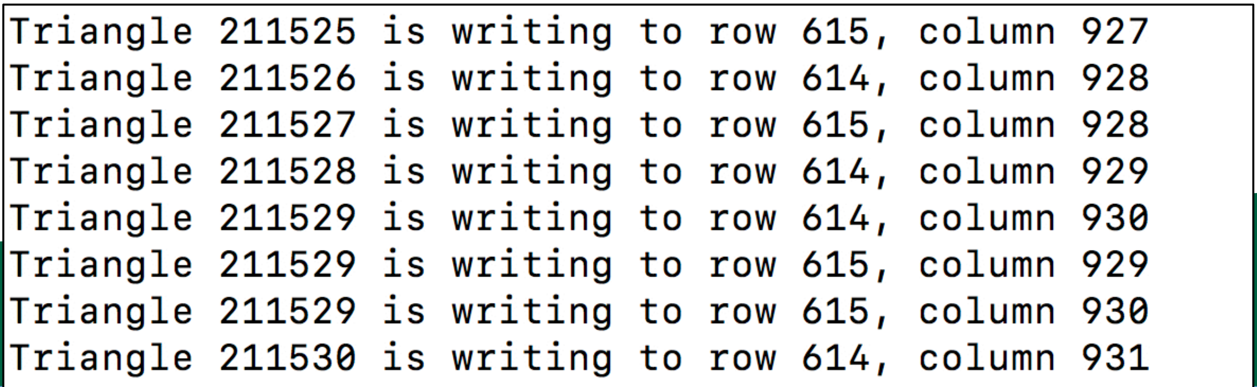
```
FORMAT = (column, row) = triangle ID
NOTE: O's are ambiguous.  Likely no triangle (black pixel), but possibly Triangle #0
```

- File triangle_ids



```
(920, 614) = 211514
(921, 614) = 211516
(922, 614) = 211517
(923, 614) = 211518
(924, 614) = 211520
(925, 614) = 211522
(926, 614) = 211523
(927, 614) = 211524
(928, 614) = 211526
(929, 614) = 211528
(930, 614) = 211529
(931, 614) = 211530
(932, 614) = 211532
(933, 614) = 211534
(934, 614) = 211536
(935, 614) = 211536
(936, 614) = 211538
(937, 614) = 0
(938, 614) = 0
```

- Output from my program



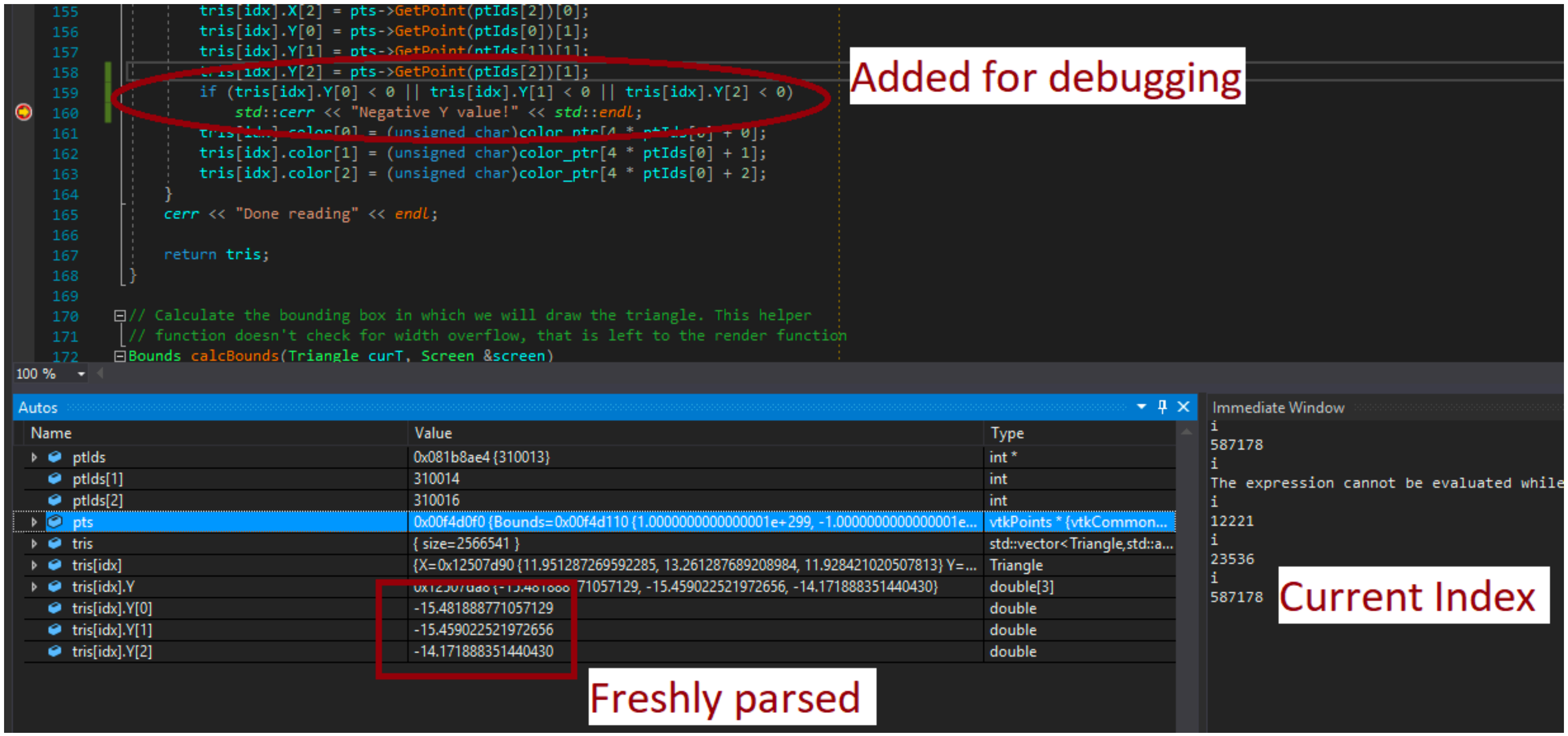
```
Triangle 211525 is writing to row 615, column 927
Triangle 211526 is writing to row 614, column 928
Triangle 211527 is writing to row 615, column 928
Triangle 211528 is writing to row 614, column 929
Triangle 211529 is writing to row 614, column 930
Triangle 211529 is writing to row 615, column 929
Triangle 211529 is writing to row 615, column 930
Triangle 211530 is writing to row 614, column 931
```

## How did I get my output?

```
int triangleID = -1;
void Screen::SetPixel(int r, int c, unsigned char *col)
{
    cerr << "Triangle " << triangleID << " is writing to row " << r << ", column " << c << endl;
```

```
for (int i = 0 ; i < triangles.size() ; i++)
{
     triangleID = i;  // triangleID is a global
     Triangle &t = triangles[i];
     //t.Print(cerr);
     RasterizeTriangle(t, screen);
}
```

# Negative Y values



```cpp
            tris[idx].X[2] = pts->GetPoint(ptIds[2])[0];
            tris[idx].Y[0] = pts->GetPoint(ptIds[0])[1];
            tris[idx].Y[1] = pts->GetPoint(ptIds[1])[1];
            tris[idx].Y[2] = pts->GetPoint(ptIds[2])[1];
            if (tris[idx].Y[0] < 0 || tris[idx].Y[1] < 0 || tris[idx].Y[2] < 0)
                std::cerr << "Negative Y value!" << std::endl;
            tris[idx].color[0] = (unsigned char)color_ptr[4 * ptIds[0] + 0];
            tris[idx].color[1] = (unsigned char)color_ptr[4 * ptIds[0] + 1];
            tris[idx].color[2] = (unsigned char)color_ptr[4 * ptIds[0] + 2];
        }
        cerr << "Done reading" << endl;

        return tris;
}

// Calculate the bounding box in which we will draw the triangle. This helper
// function doesn't check for width overflow, that is left to the render function
Bounds calcBounds(Triangle curT, Screen &screen)
```

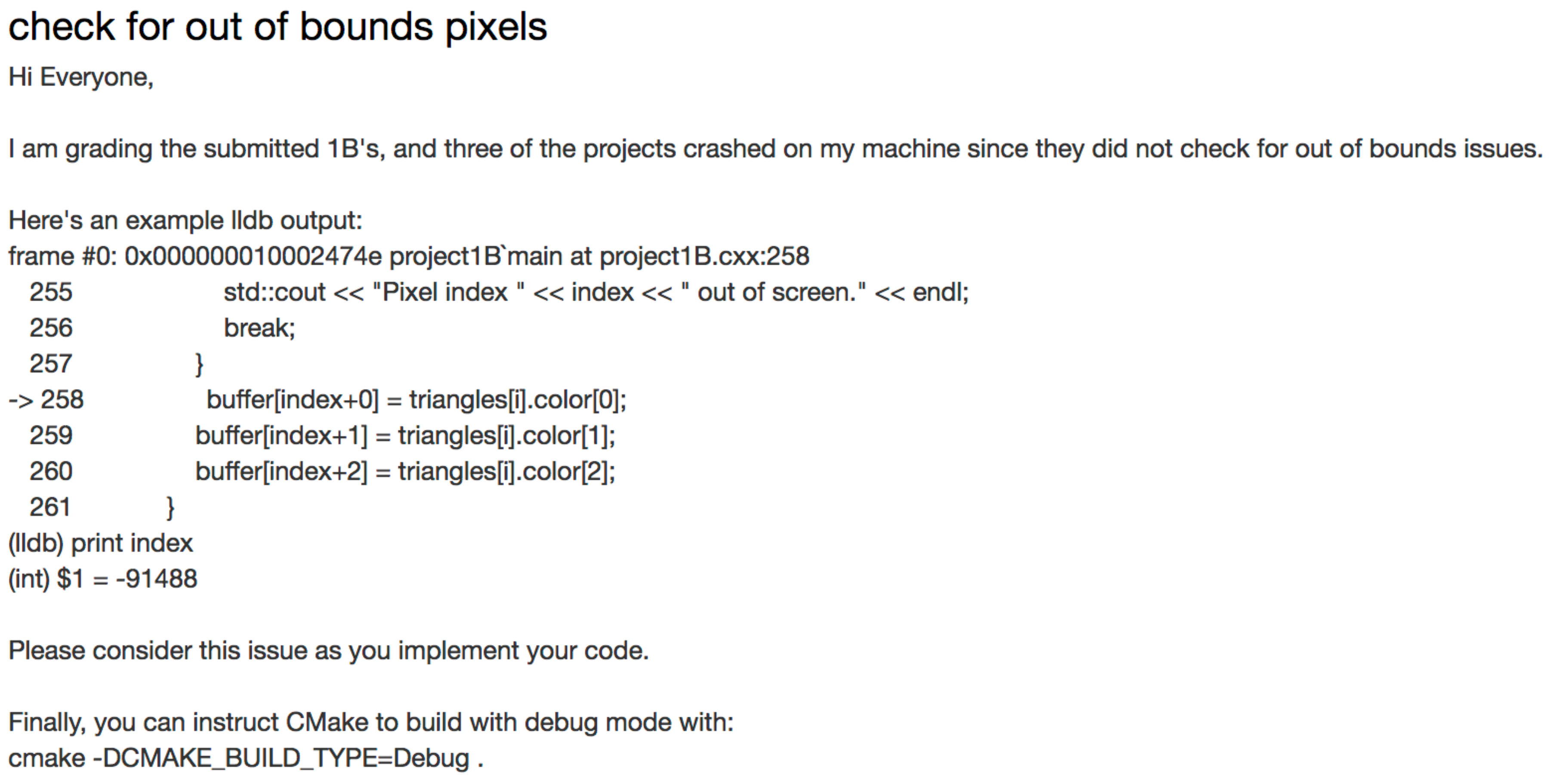
100 %

Autos

| Name | Value | Type |
|---|---|---|
| ptIds | 0x081b8ae4 {310013} | int * |
| ptIds[1] | 310014 | int |
| ptIds[2] | 310016 | int |
| pts | 0x00f4d0f0 {Bounds=0x00f4d110 {1.0000000000000001e+299, -1.0000000000000001e... | vtkPoints * {vtkCommon... |
| tris | { size=2566541 } | std::vector<Triangle,std::a... |
| tris[idx] | {X=0x12507d90 {11.951287269592285, 13.261287689208984, 11.928421020507813} Y=... | Triangle |
| tris[idx].Y | 0x12507da8 {-15.481888771057129, -15.459022521972656, -14.171888351440430} | double[3] |
| tris[idx].Y[0] | -15.481888771057129 | double |
| tris[idx].Y[1] | -15.459022521972656 | double |
| tris[idx].Y[2] | -14.171888351440430 | double |

Immediate Window

```
i
587178
i
The expression cannot be evaluated while
i
12221
i
23536
i
587178
```



## check for out of bounds pixels

Hi Everyone,

I am grading the submitted 1B's, and three of the projects crashed on my machine since they did not check for out of bounds issues.

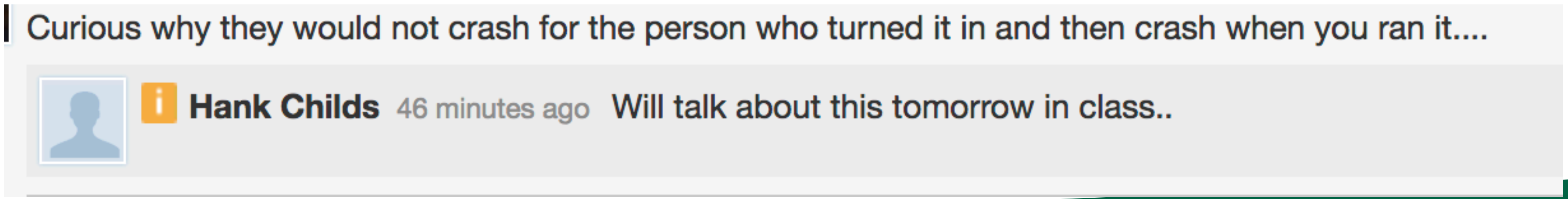
Here's an example lldb output:

```
frame #0: 0x000000010002474e project1B`main at project1B.cxx:258
   255                     std::cout << "Pixel index " << index << " out of screen." << endl;
   256                     break;
   257                 }
-> 258                 buffer[index+0] = triangles[i].color[0];
   259                buffer[index+1] = triangles[i].color[1];
   260                buffer[index+2] = triangles[i].color[2];
   261            }
(lldb) print index
(int) $1 = -91488
```

Please consider this issue as you implement your code.

Finally, you can instruct CMake to build with debug mode with:

```
cmake -DCMAKE_BUILD_TYPE=Debug .
```



Curious why they would not crash for the person who turned it in and then crash when you ran it....

**Hank Childs** 46 minutes ago Will talk about this tomorrow in class..

## 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
- On Thursday, we tackled this problem:

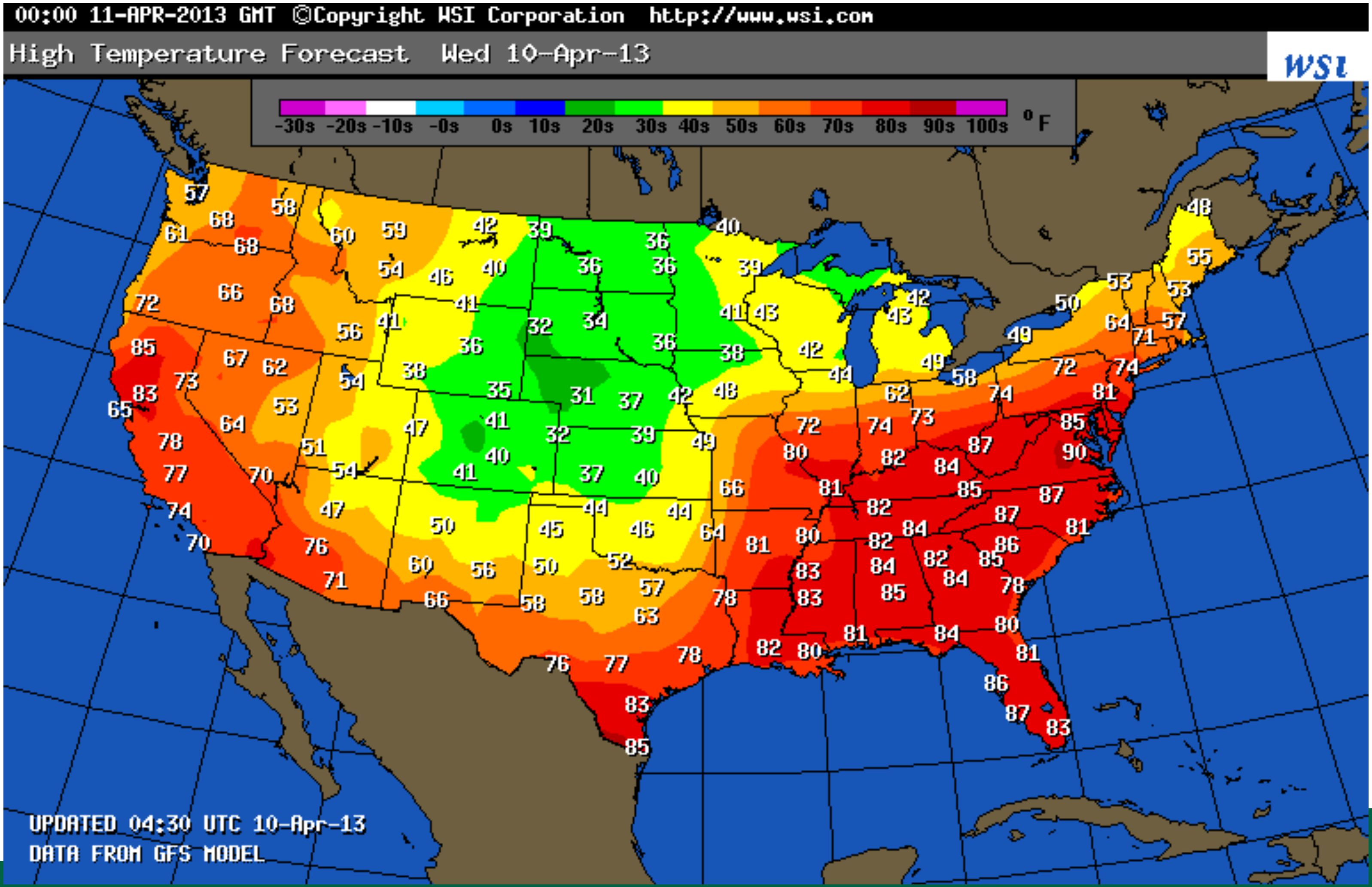
How to deposit triangle colors onto an image?

Still don't know how to:

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

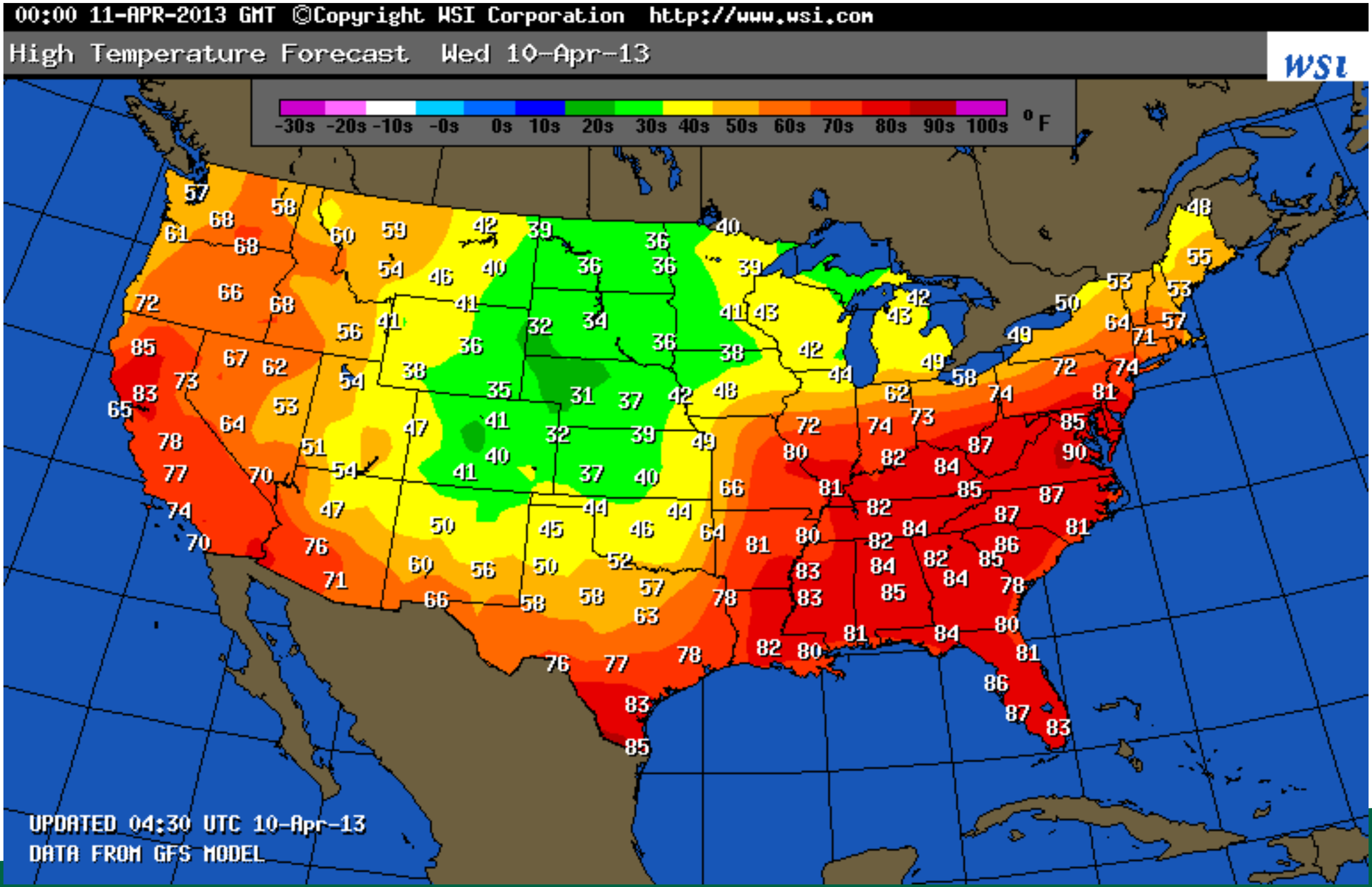
Today's lecture will go over the key operation to do these two.

## What is a field?


00:00 11-APR-2013 GMT ©Copyright WSI Corporation http://www.wsi.com
High Temperature Forecast Wed 10-Apr-13
WSI
-30s -20s -10s -0s 0s 10s 20s 30s 40s 50s 60s 70s 80s 90s 100s °F
UPDATED 04:30 UTC 10-Apr-13
DATA FROM GFS MODEL

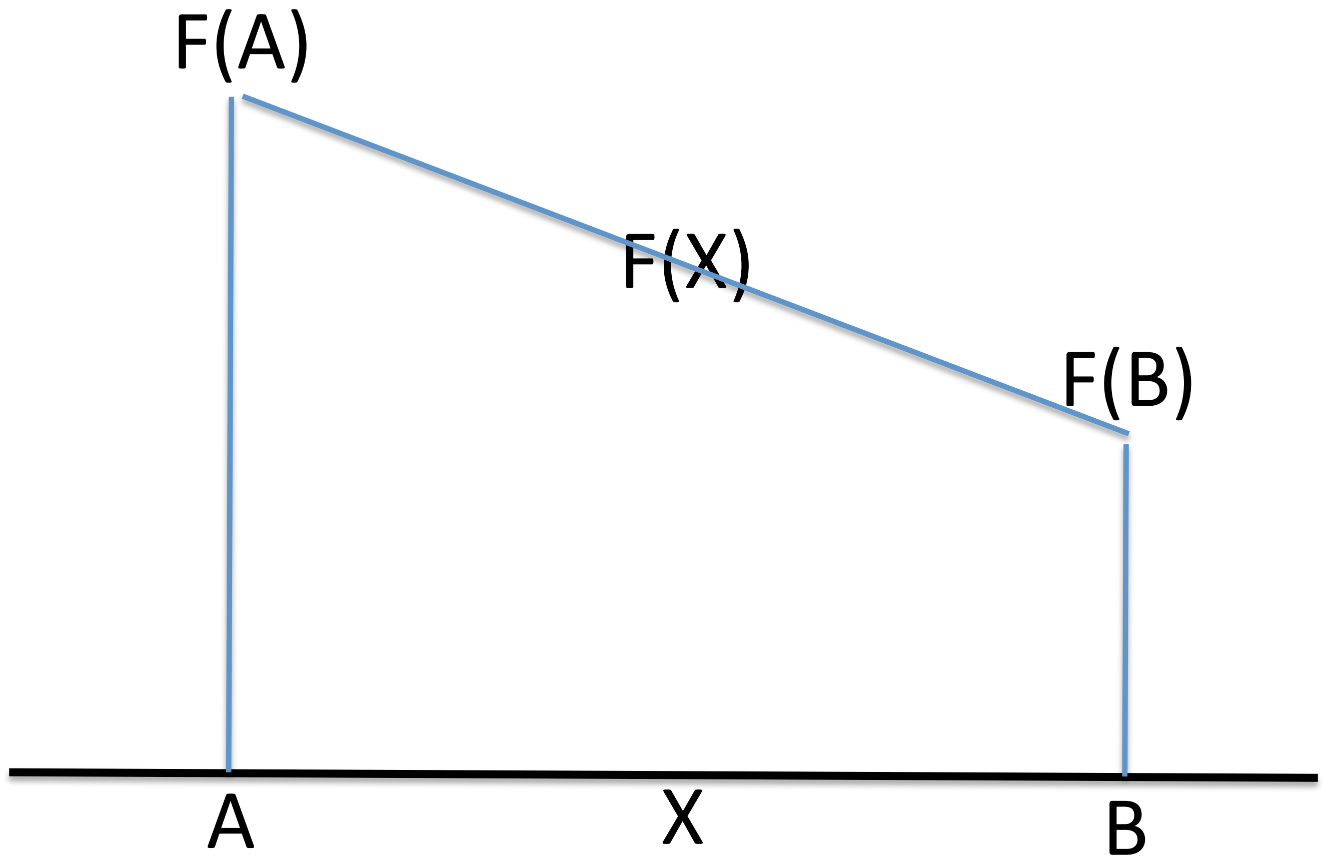
Example field (2D): temperature over the United States

# How much data is needed to make this picture?


00:00 11-APR-2013 GMT ©Copyright WSI Corporation http://www.wsi.com
High Temperature Forecast Wed 10-Apr-13
WSI
-30s -20s -10s -0s 0s 10s 20s 30s 40s 50s 60s 70s 80s 90s 100s °F
UPDATED 04:30 UTC 10-Apr-13
DATA FROM GFS MODEL

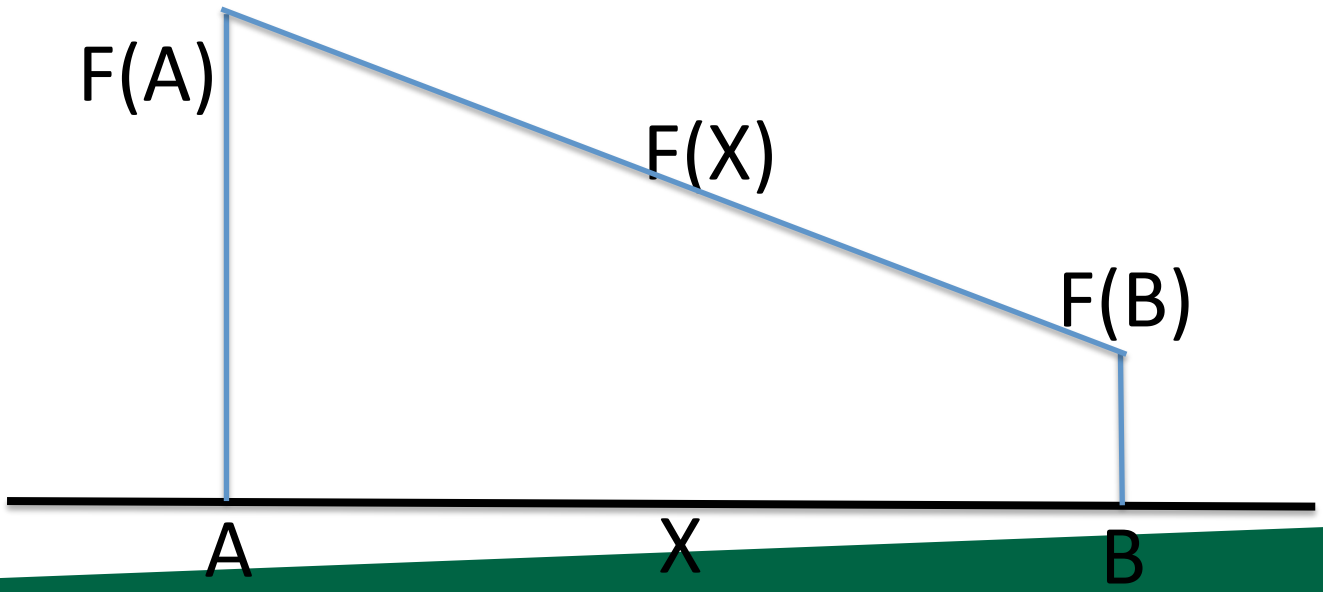
Example field (2D): temperature over the United States

# 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)

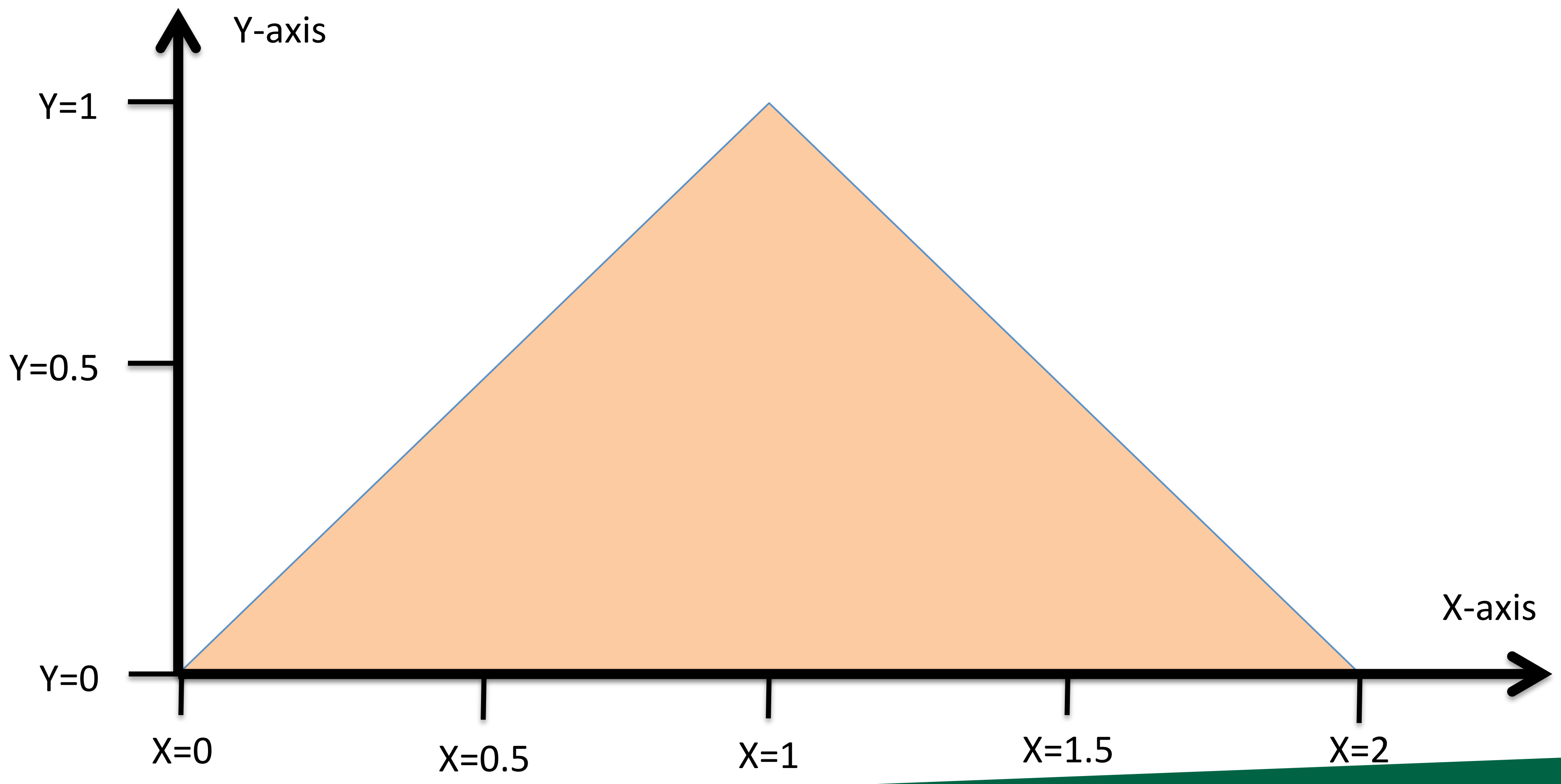


## Quiz Time #4

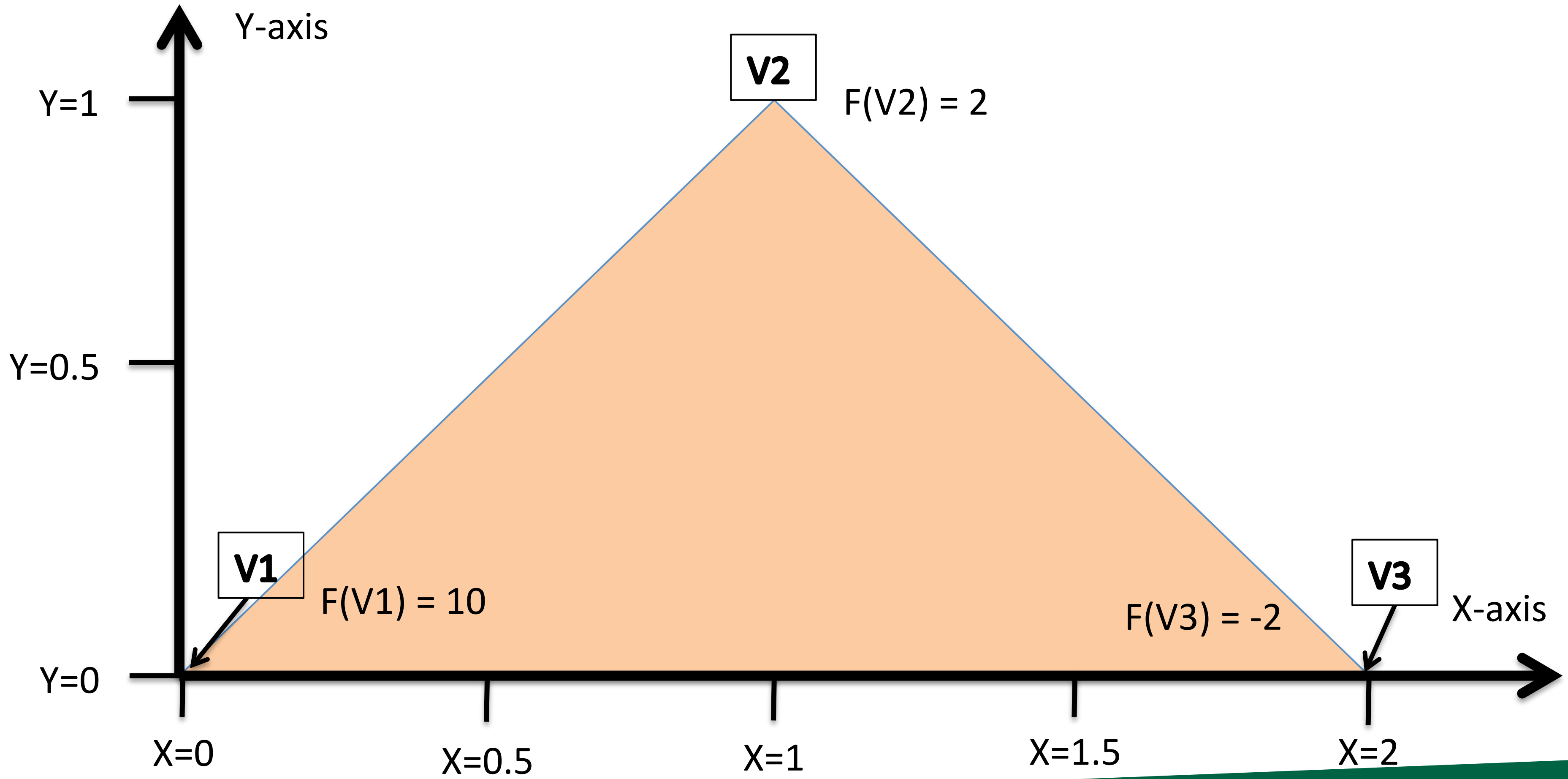
- $F(3) = 5$, $F(6) = 11$
- What is $F(4)$? $= 5 + (4-3)/(6-3)*(11-5) = 7$

- General equation to interpolate:
  - $F(X) = F(A) + t*(F(B)-F(A))$
- t is proportion of X between A and B
  - $t = (X-A)/(B-A)$

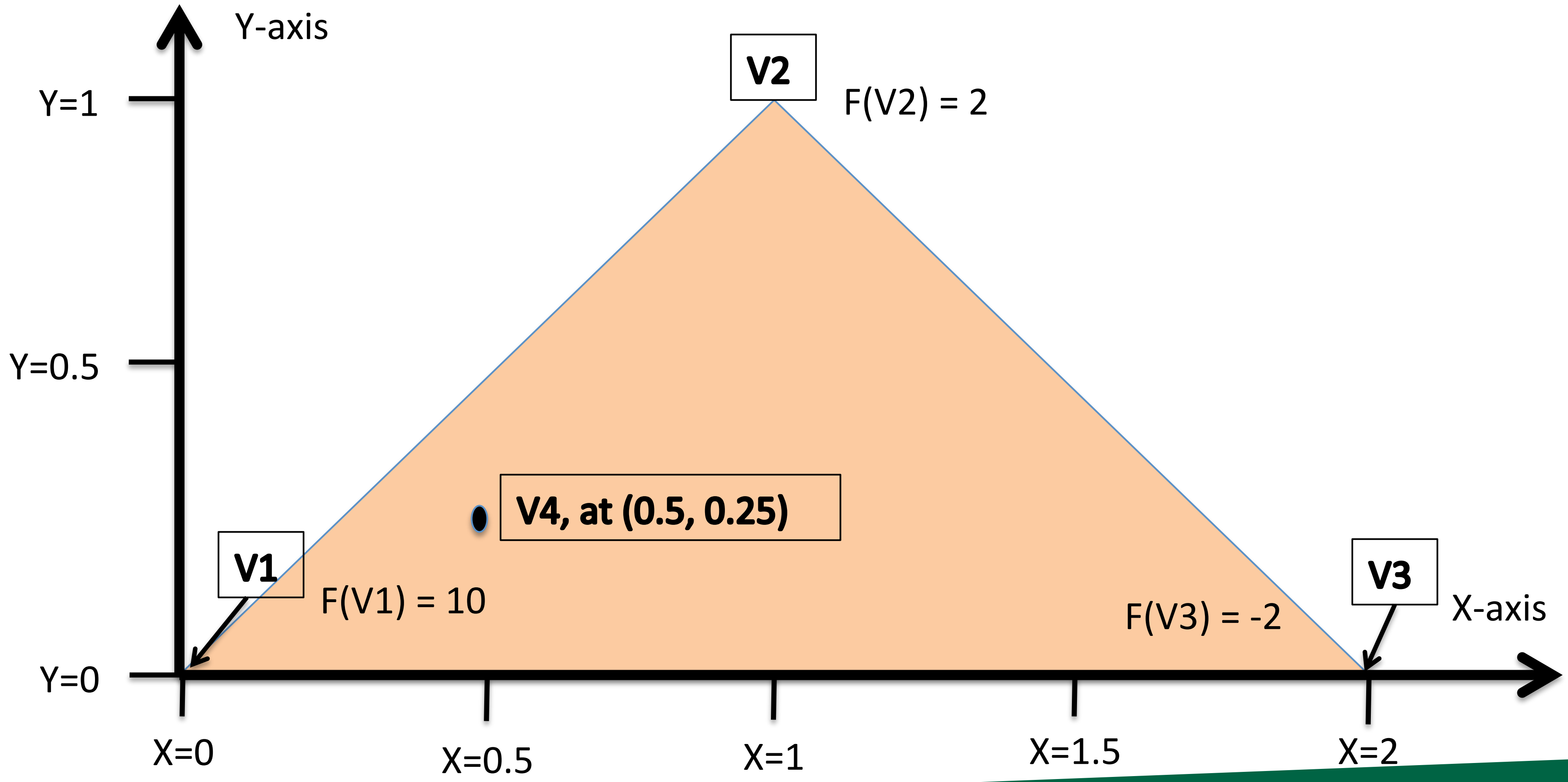
## Consider a single scalar field defined on a triangle.



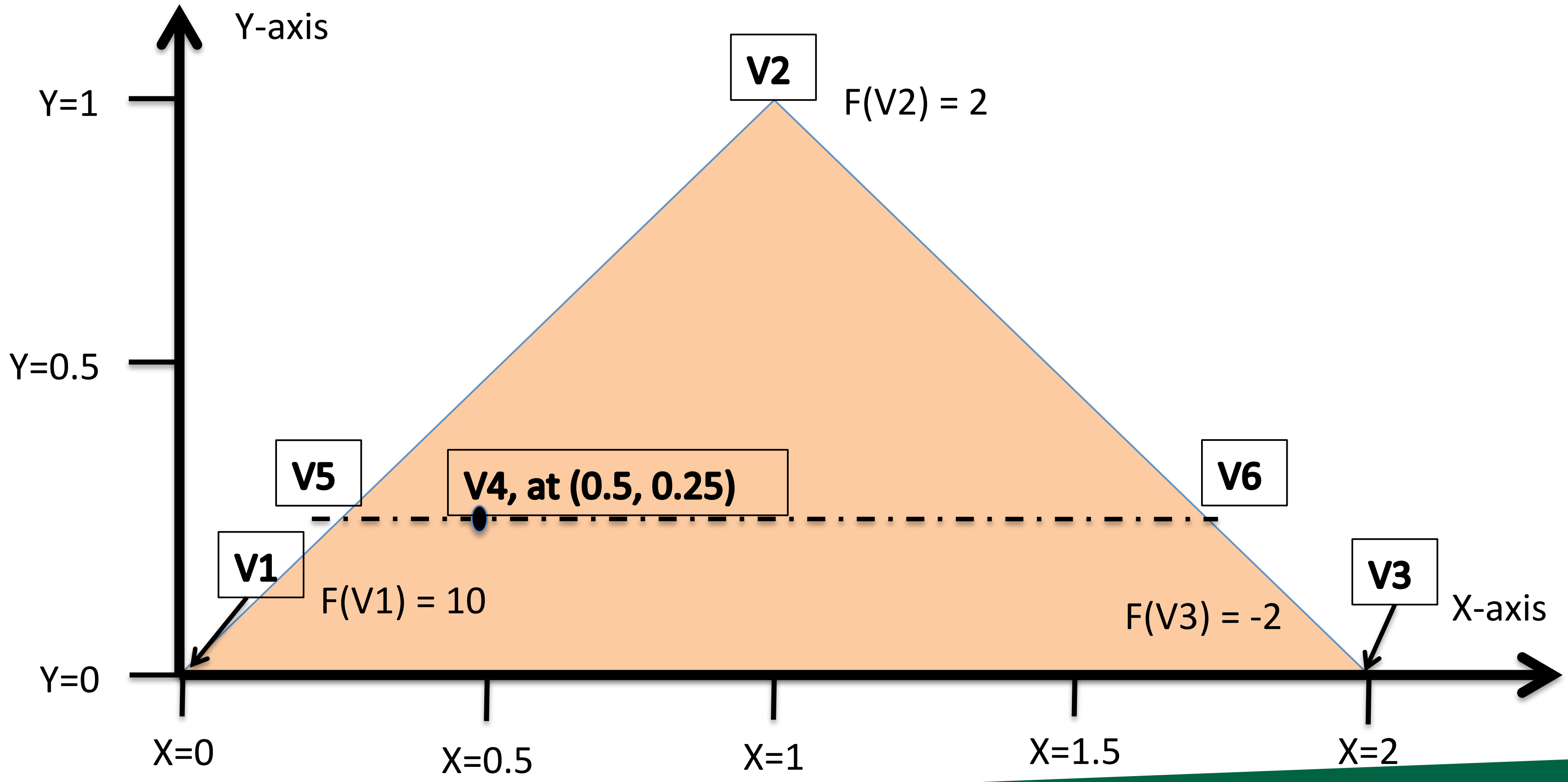
# Consider a single scalar field defined on a triangle.



# What is F(V4)?

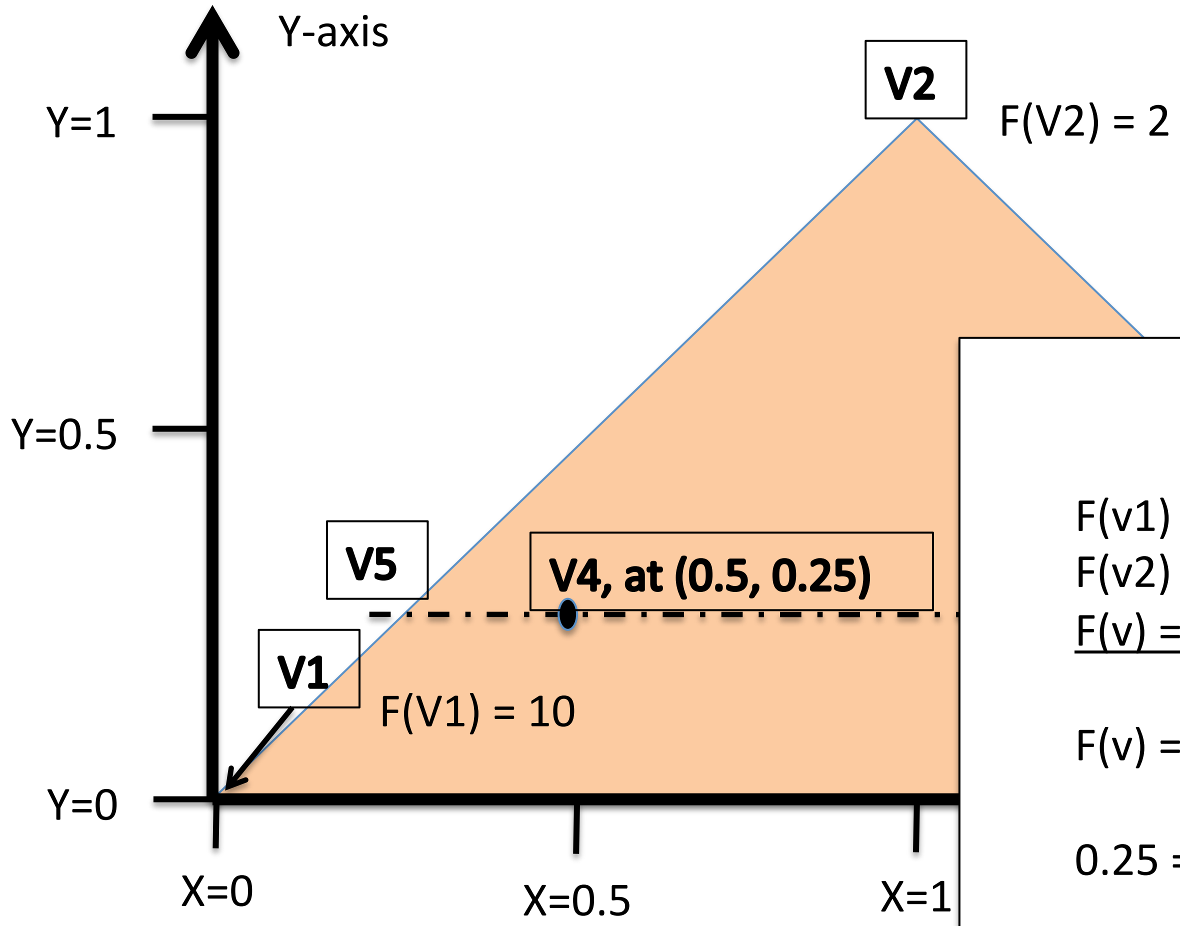

Y-axis
Y=1
Y=0.5
Y=0
V2
F(V2) = 2
V4, at (0.5, 0.25)
V1
F(V1) = 10
F(V3) = -2
V3
X-axis
X=0
X=0.5
X=1
X=1.5
X=2

# What is F(V4)?


Y-axis
Y=1
Y=0.5
Y=0
V2
F(V2) = 2
V5
V4, at (0.5, 0.25)
V6
V1
F(V1) = 10
V3
F(V3) = -2
X-axis
X=0
X=0.5
X=1
X=1.5
X=2

- 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?

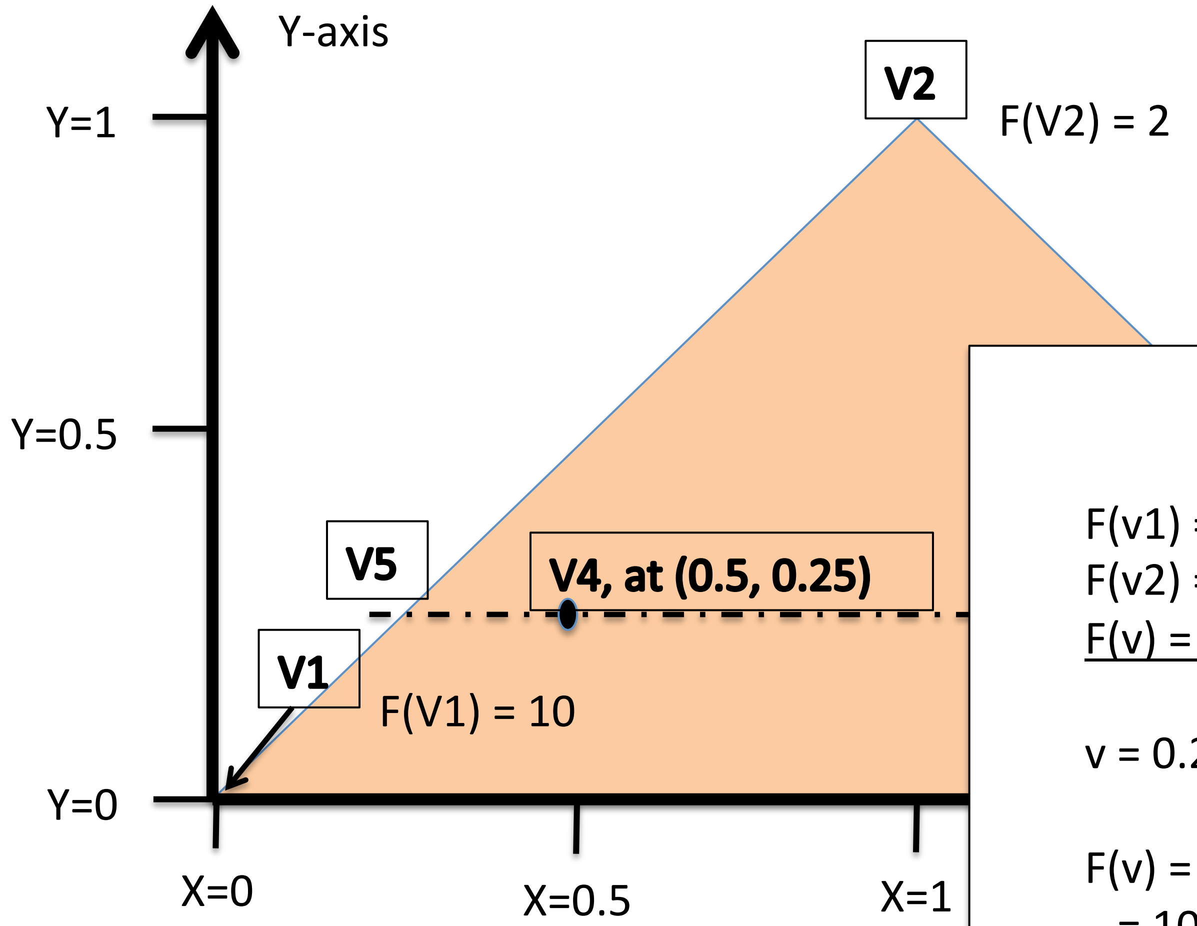


F(v1) = A → F(0) = 0
F(v2) = B → F(1) = 1
<u>F(v) = A + ((v-v1)/(v2-v1))*(B-A):</u>

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?

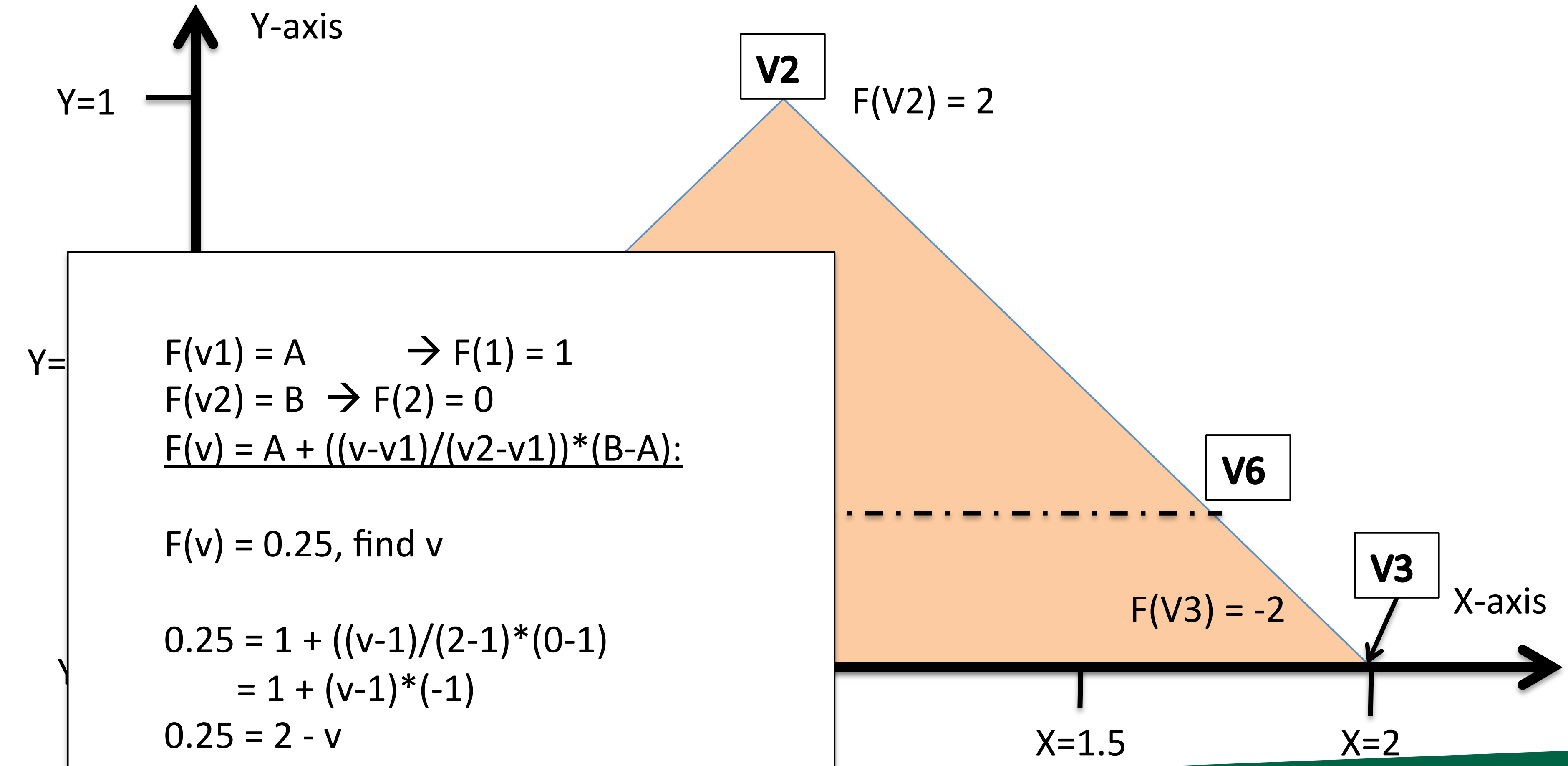


F(v1) = A $\rightarrow$ F(0) = 10
F(v2) = B $\rightarrow$ F(1) = 2
<u>F(v) = A + ((v-v1)/(v2-v1))*(B-A):</u>

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?

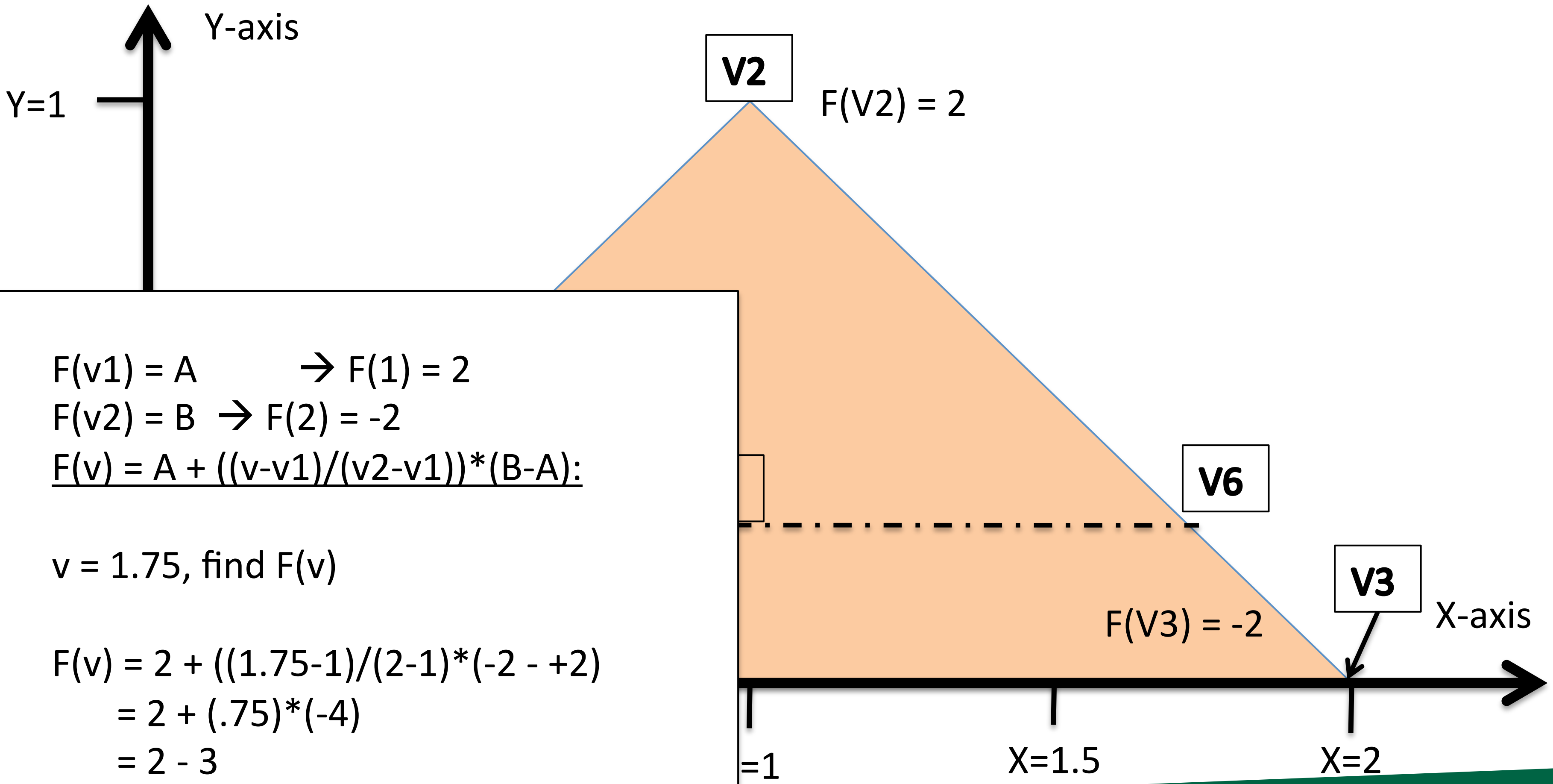


F(v1) = A → F(1) = 1
F(v2) = B → F(2) = 0
F(v) = A + ((v-v1)/(v2-v1))*(B-A):

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?

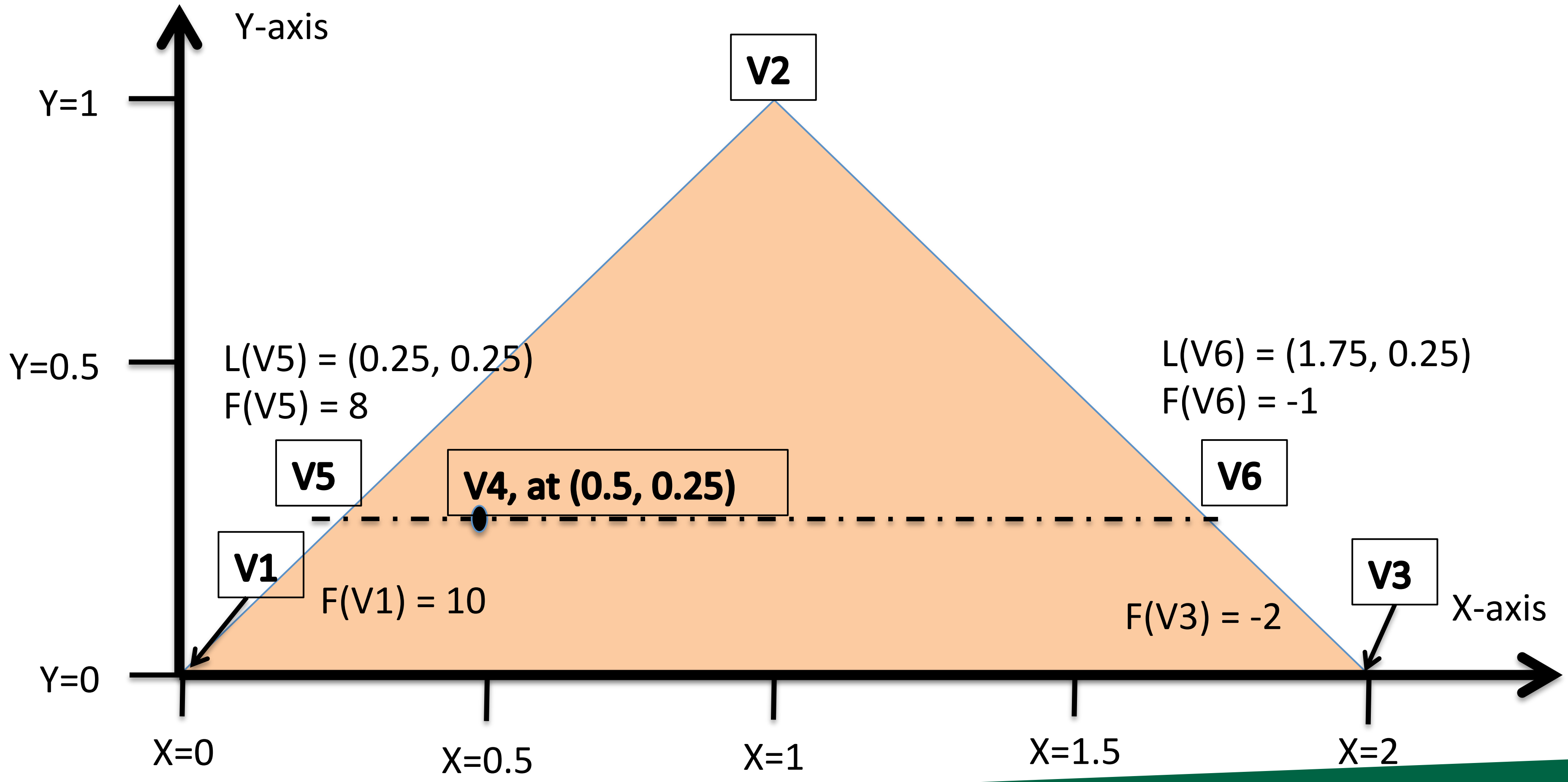


F(v1) = A → F(1) = 2
F(v2) = B → F(2) = -2
F(v) = A + ((v-v1)/(v2-v1))*(B-A):

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?

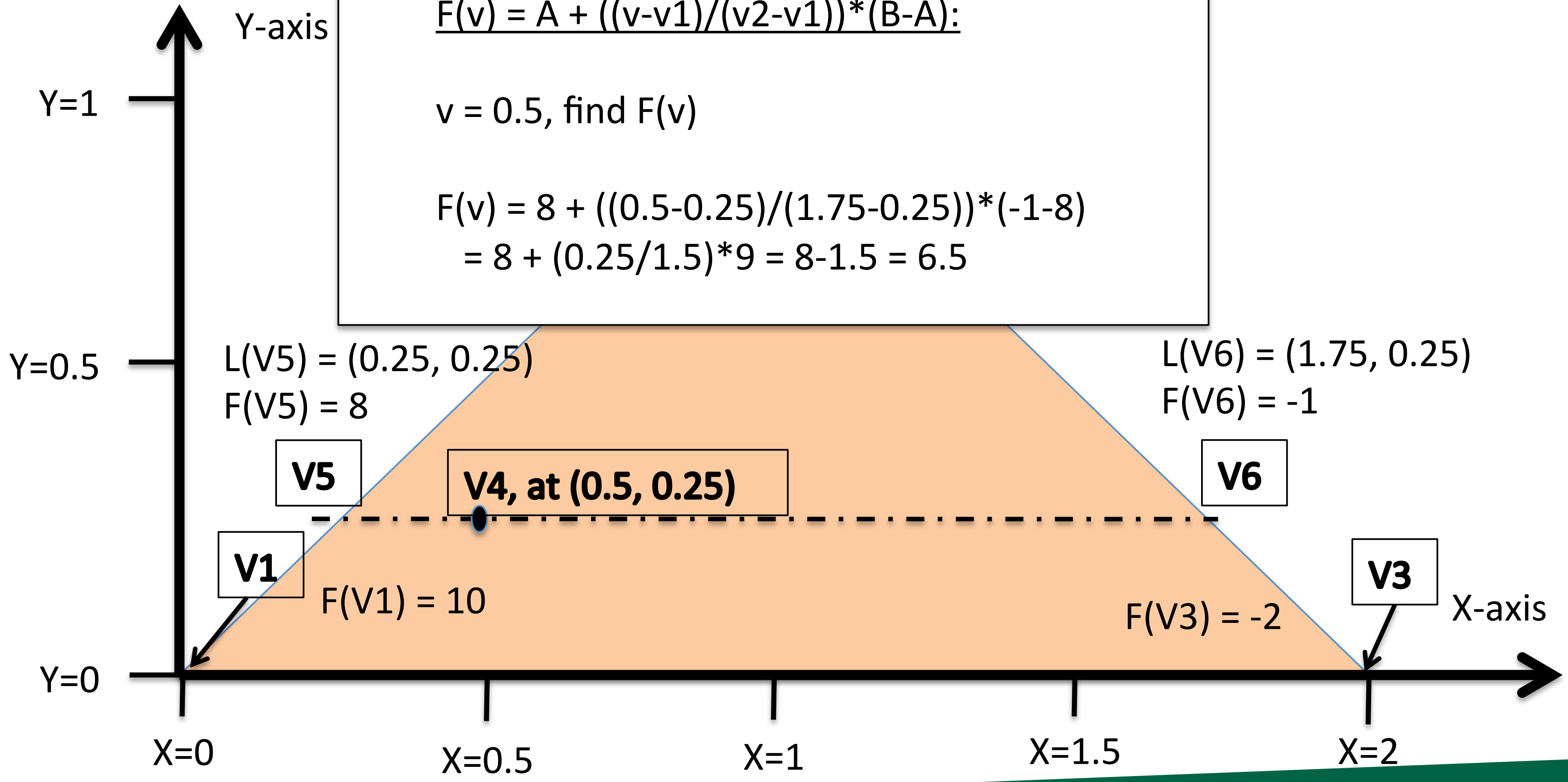


# What is the F value of V5?

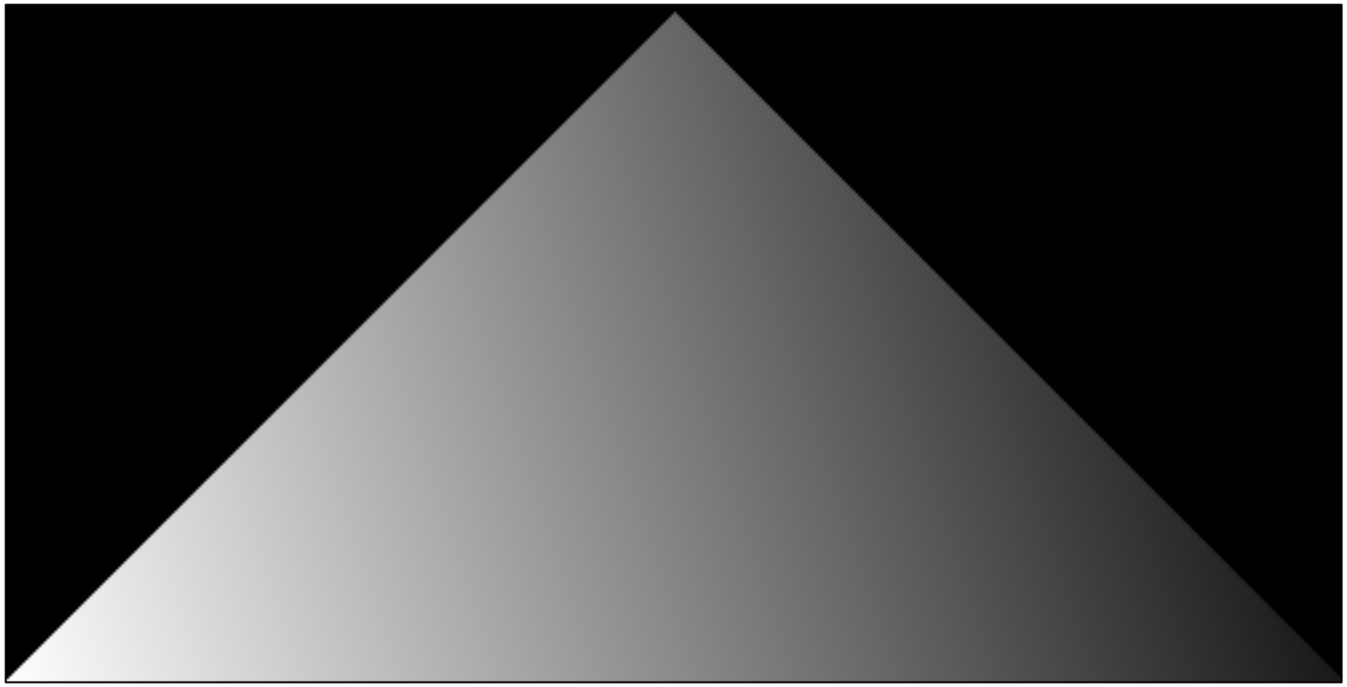
F(v1) = A → F(0.25) = 8
F(v2) = B → F(1.75) = -1
<u>F(v) = A + ((v-v1)/(v2-v1))*(B-A):</u>

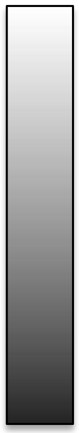
v = 0.5, find F(v)

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



# Visualization of F





■ Not defined

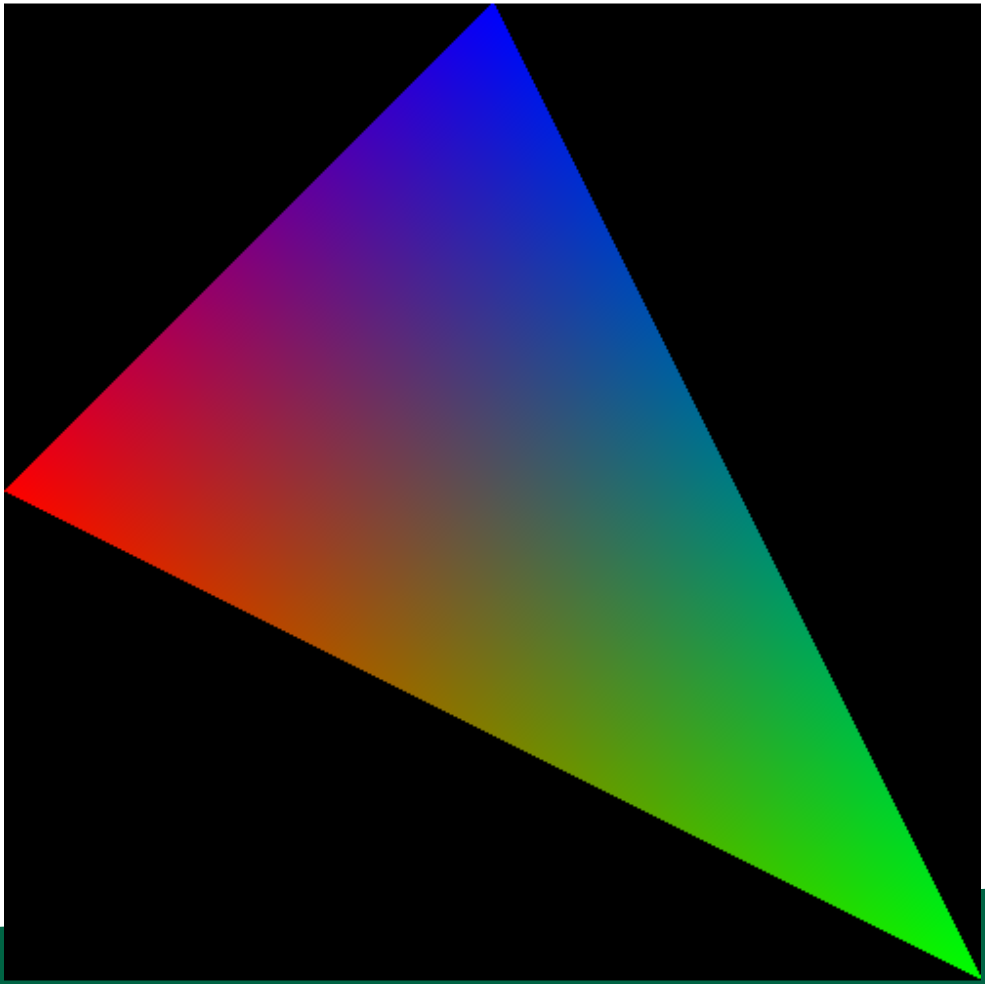
How do you think this picture was made?

## Now We Understand Interpolation
## Let’s Use It For Two New Ideas:
## Color Interpolation
## & Z-buffer Interpolation

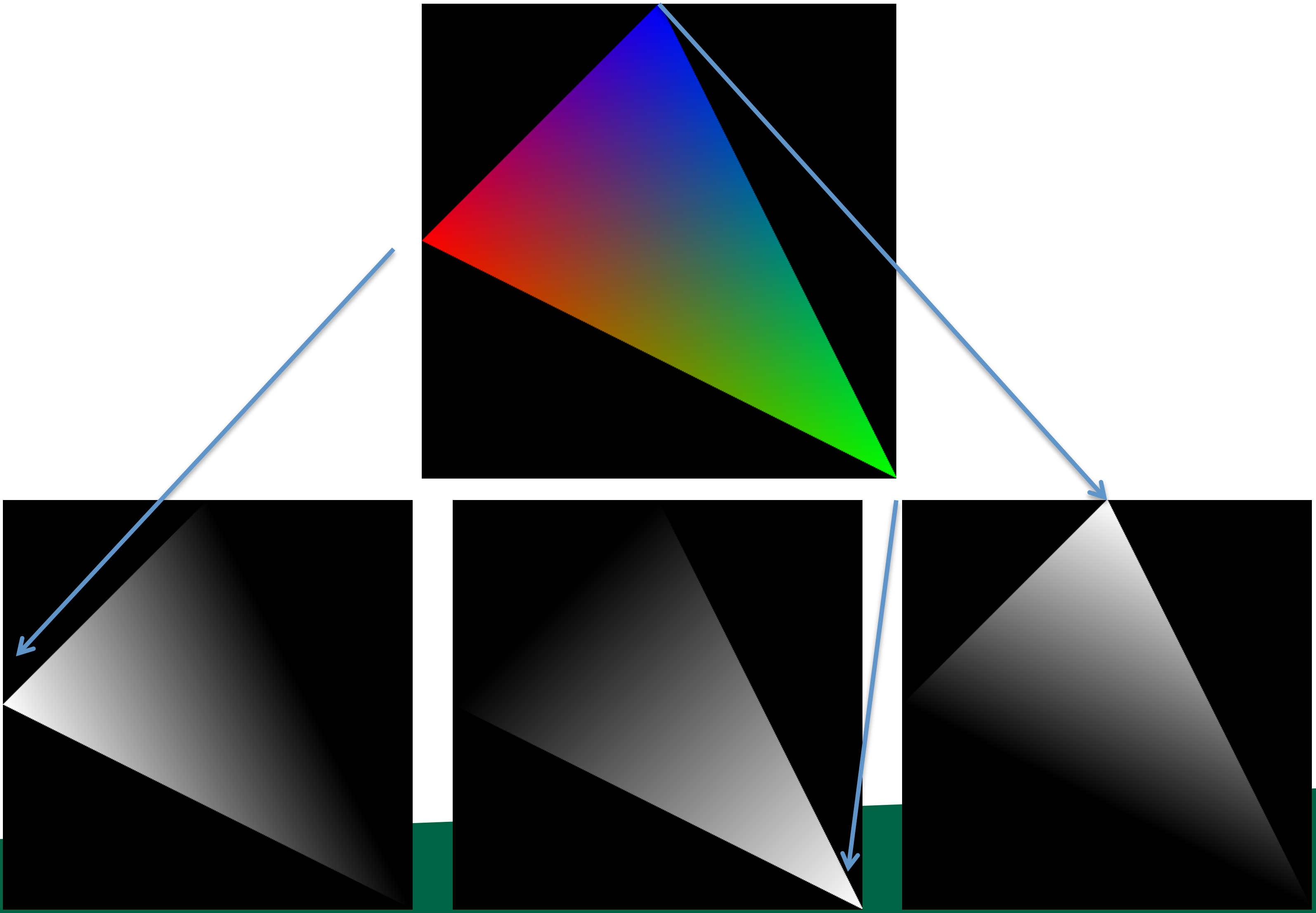
UNIVERSITY OF OREGON

## Colors

## What about triangles that have more than one color?



## The color is in three channels, hence three scalar fields defined on the triangle.



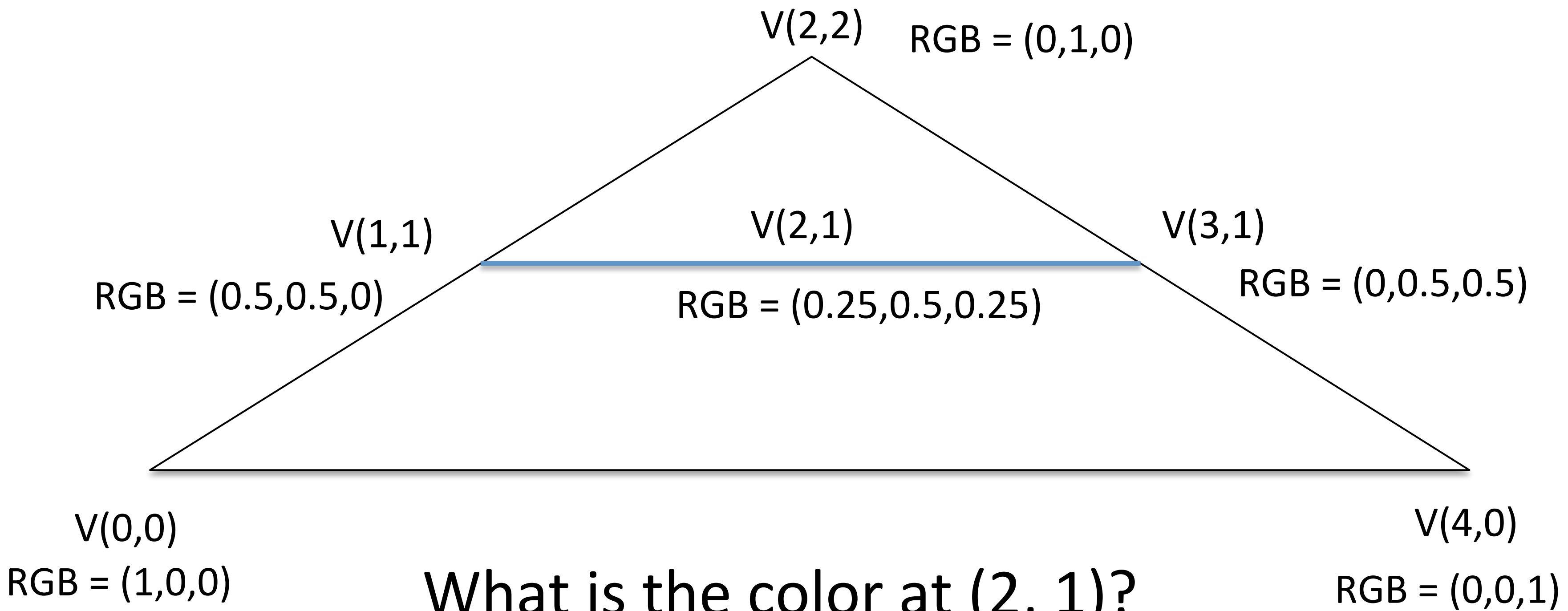
## Scanline algorithm

- 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 → 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)

## 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!

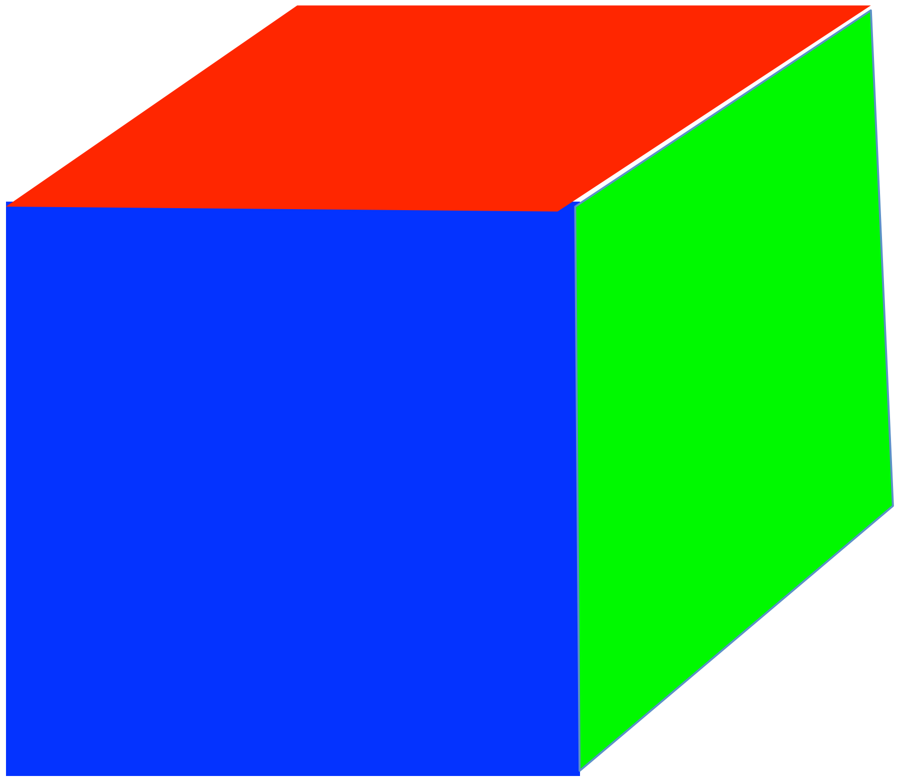
## Important

- ceiling / floor: needed to decide which pixels to deposit colors to
  - 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….



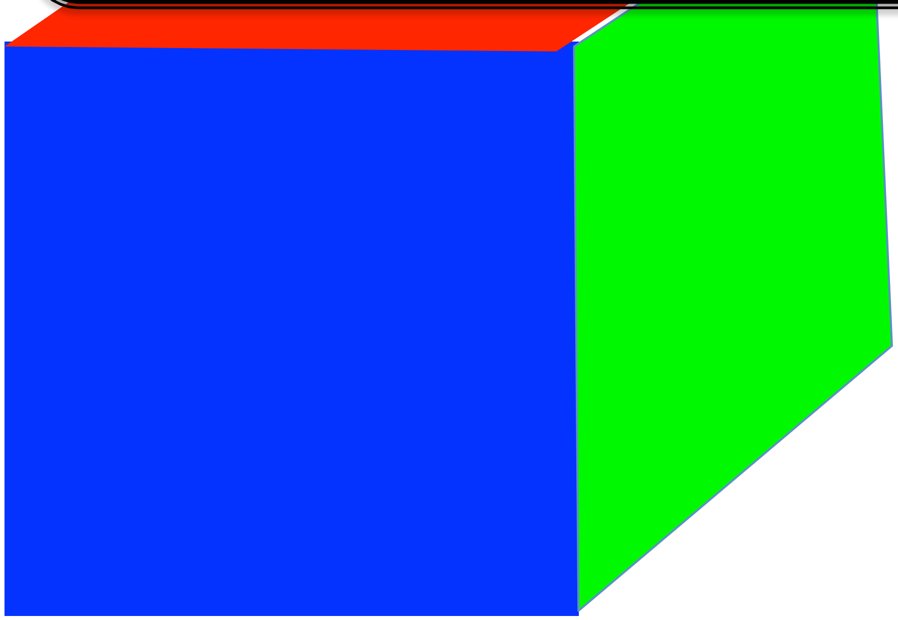
| Face | Color |
|---|---|
| Front | Blue |
| Right | Green |
| Top | Red |
| Back | Yellow |
| Left | Purple |
| Bottom | Cyan |

View from "front/top/right" side

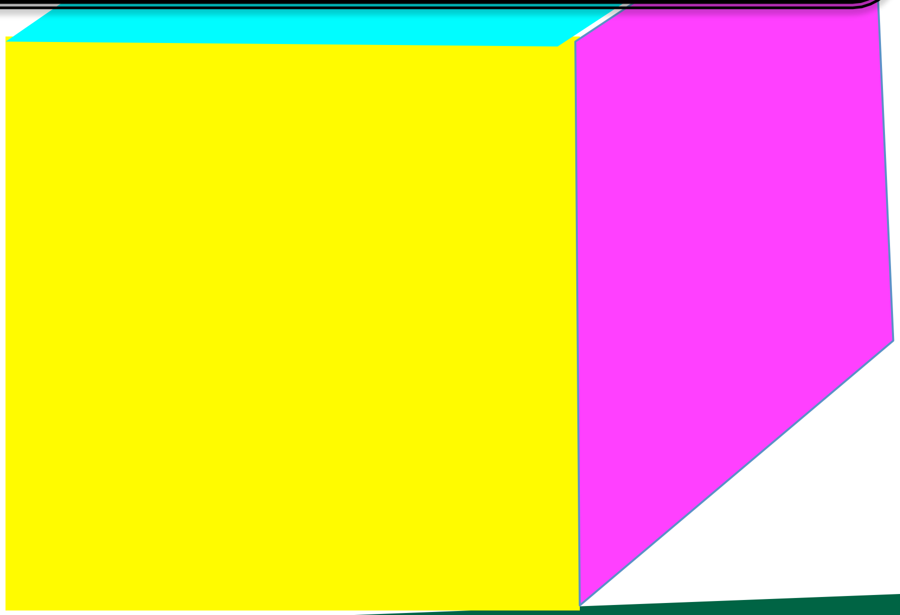
UNIVERSITY OF OREGON

# 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?

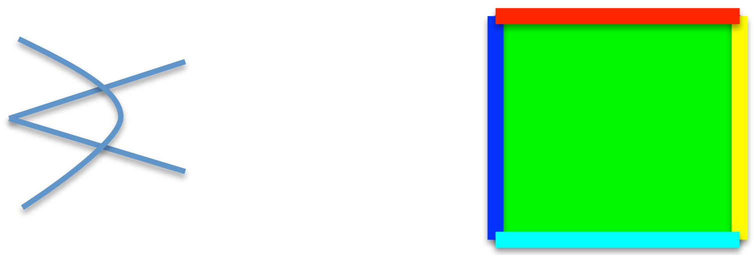


View from "front/top/right" side



View from "back/bottom/left" side

# Consider a scene from the right side

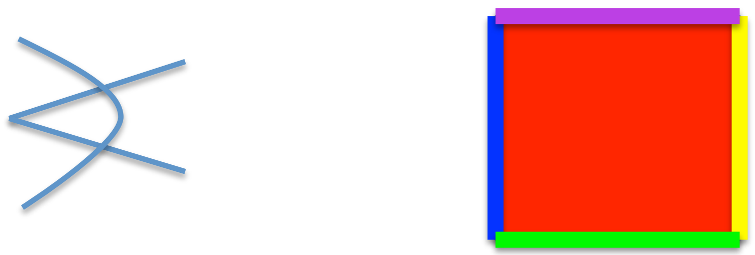


Camera/eyeball

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

| Face | Color |
|---|---|
| Front | Blue |
| Right | Green |
| Top | Red |
| Back | Yellow |
| Left | Purple |
| Bottom | Cyan |

# Consider the scene from the top side



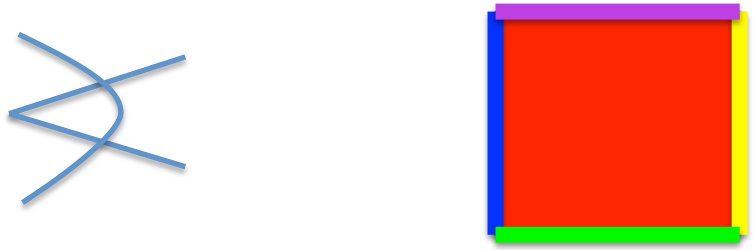
Camera/eyeball

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

| Face | Color |
|---|---|
| Front | Blue |
| Right | Green |
| Top | Red |
| Back | Yellow |
| Left | Purple |
| Bottom | Cyan |

# What do we render?

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



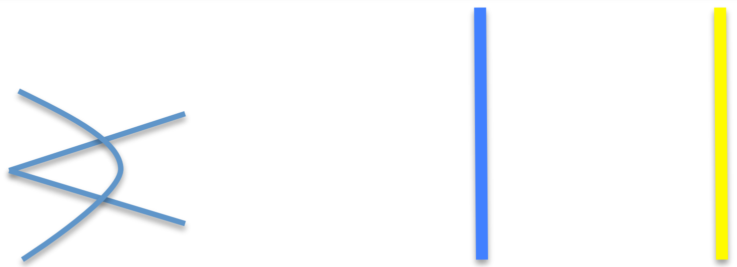
Camera/eyeball

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

| Face | Color |
|---|---|
| Front | Blue |
| Right | Green |
| Top | Red |
| Back | Yellow |
| Left | Purple |
| Bottom | Cyan |

# 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

| Face | Color |
|---|---|
| Front | Blue |
| Right | Green |
| Top | Red |
| Back | Yellow |
| Left | Purple |
| Bottom | Cyan |

# New field associated with each triangle: depth

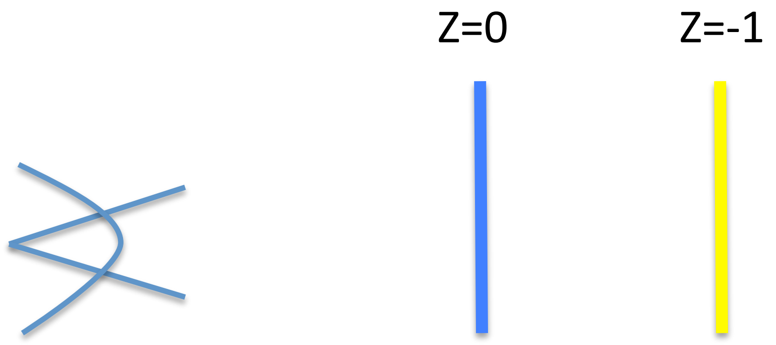
- Project 1B,1C:

```
class Triangle
{
  public:
          Double  X[3];
          Double  Y[3];
          …
};
```

- Now…

```
          Double  Z[3];
```

# What do we render?



Camera/eyeball

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

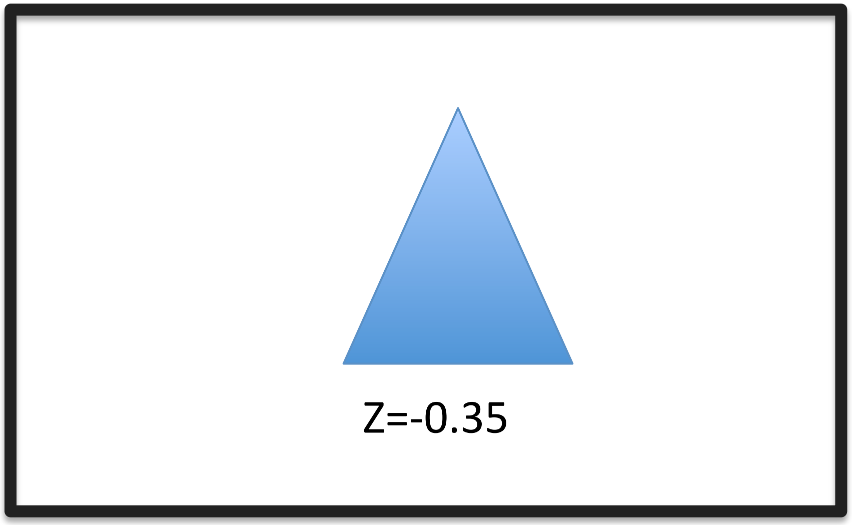
| Face | Color |
|---|---|
| Front | Blue |
| Right | Green |
| Top | Red |
| Back | Yellow |
| Left | Purple |
| Bottom | Cyan |

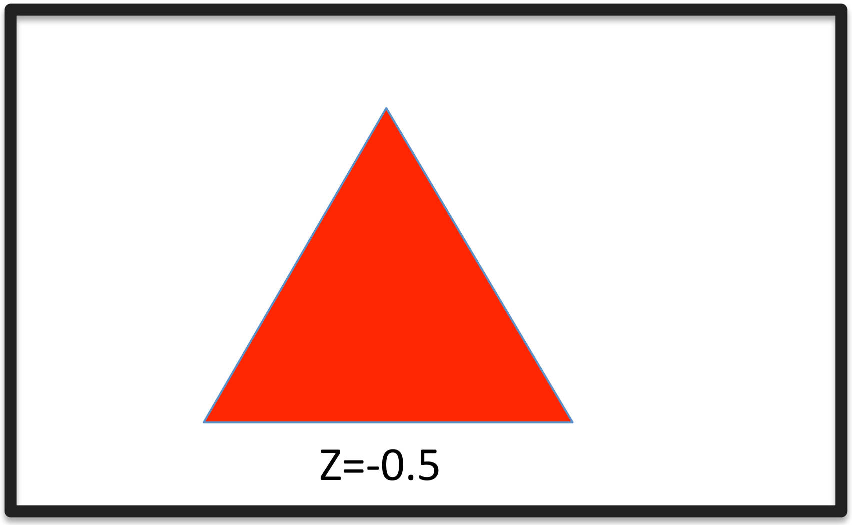
## Using depth when rendering

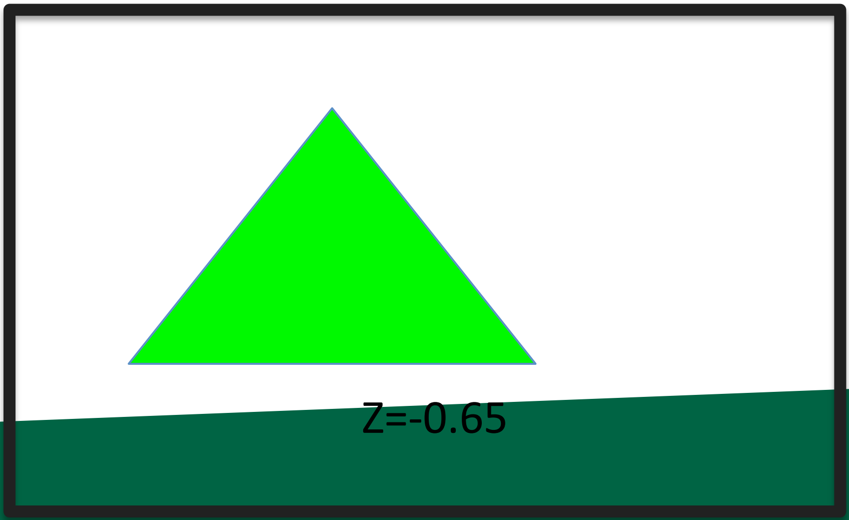
- Use Z values to guide which geometry is displayed and which is obscured.
- Example….

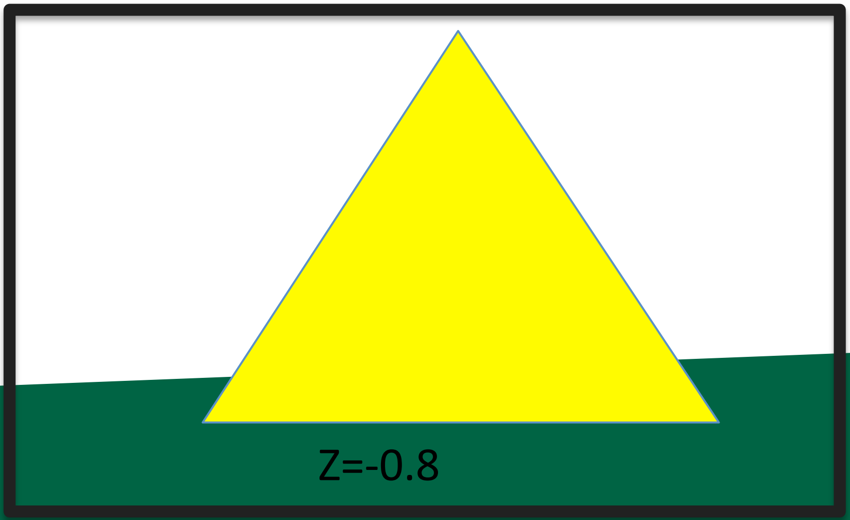
UNIVERSITY OF OREGON

# Consider 4 triangles with constant Z values

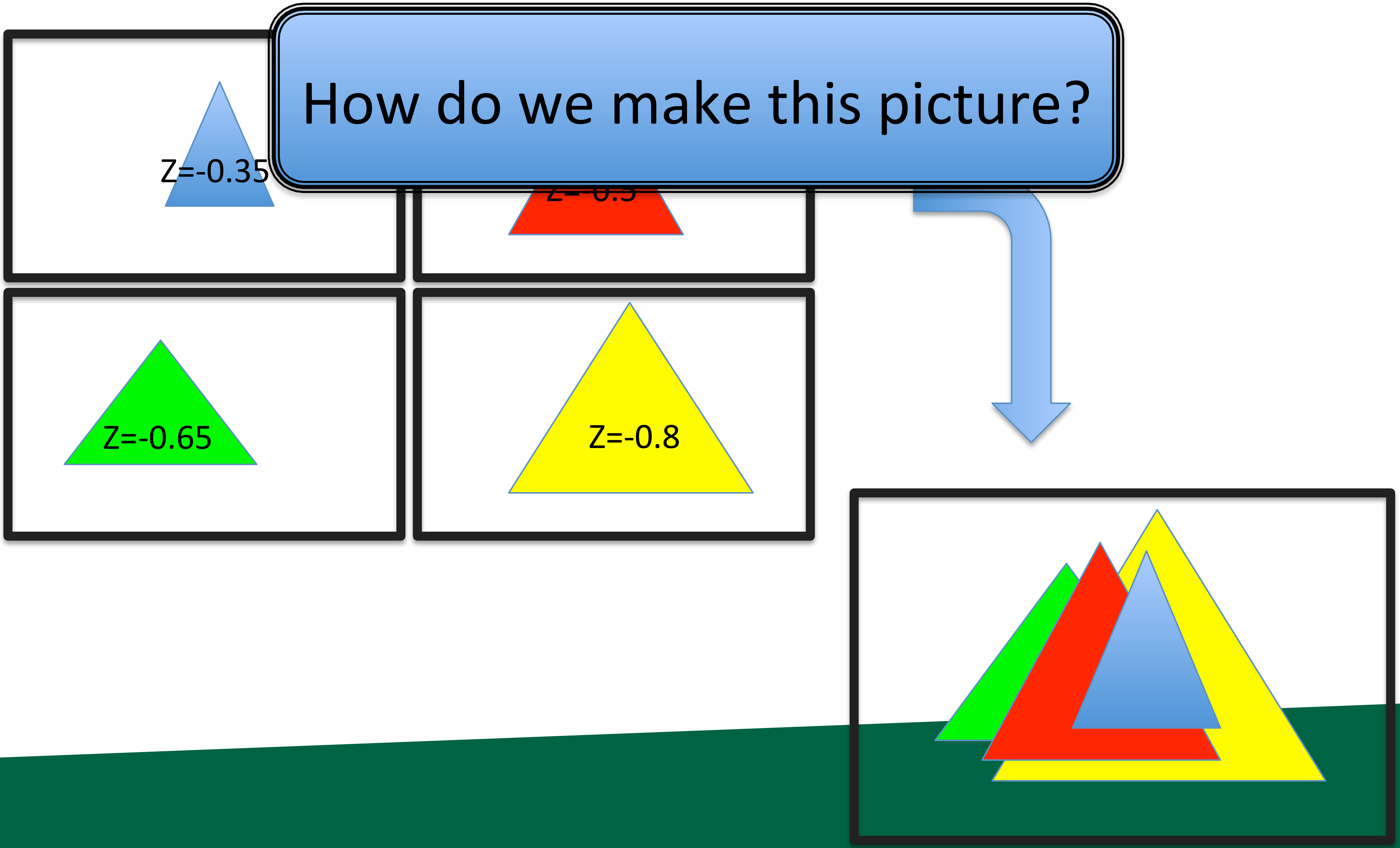

Z=-0.35


Z=-0.5


Z=-0.65


Z=-0.8

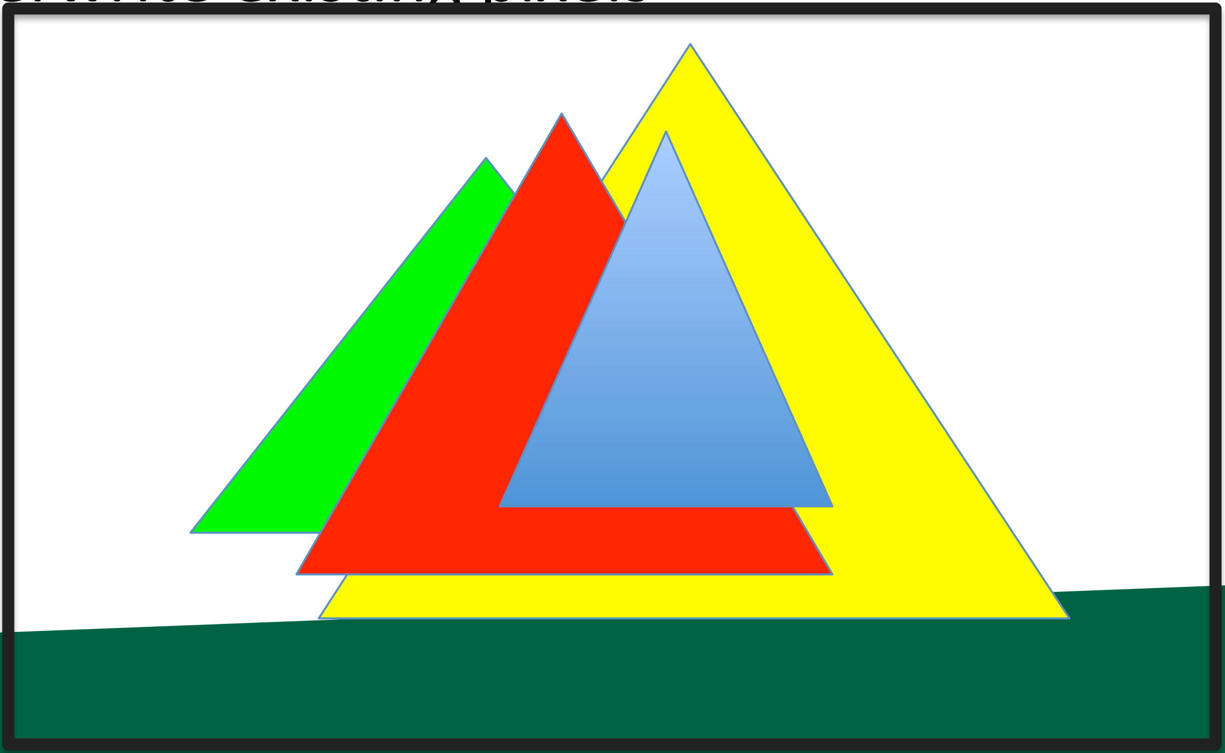
# Consider 4 triangles with constant Z values



UNIVERSITY OF OREGON

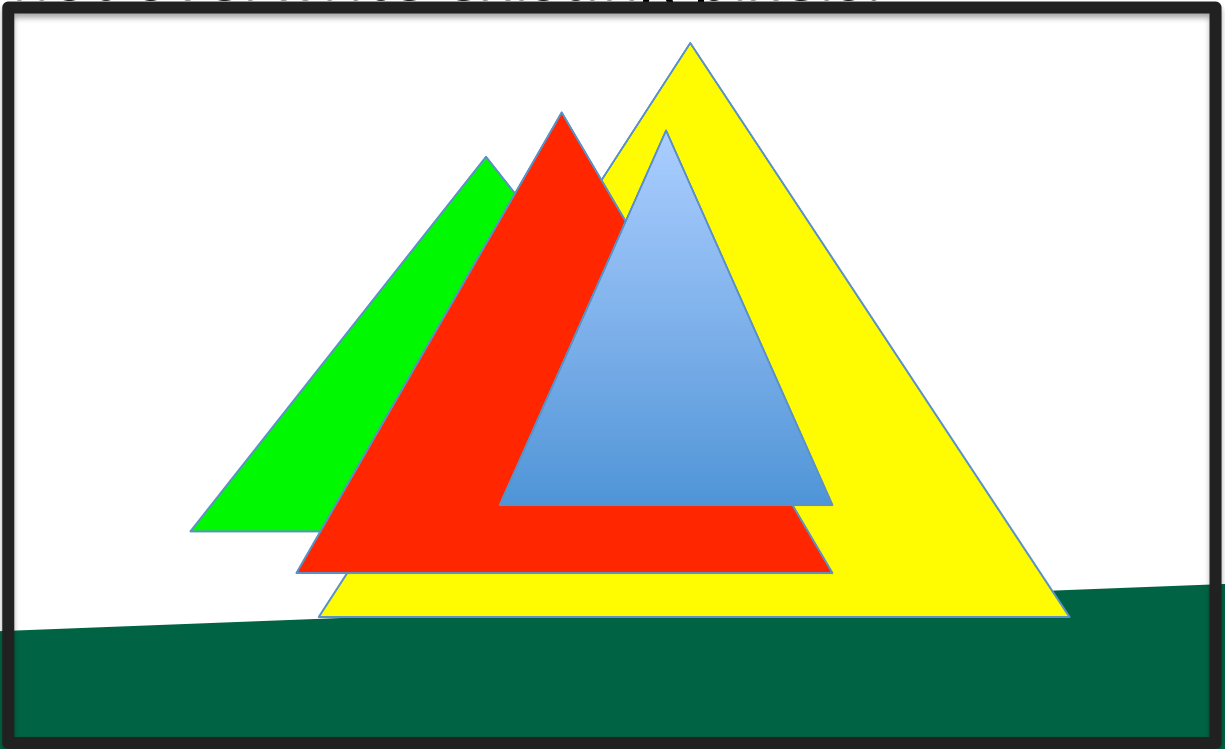
## Idea #1

- Sort triangles “back to front” (based on Z)
- Render triangles in back to front order
  - Overwrite existing pixels

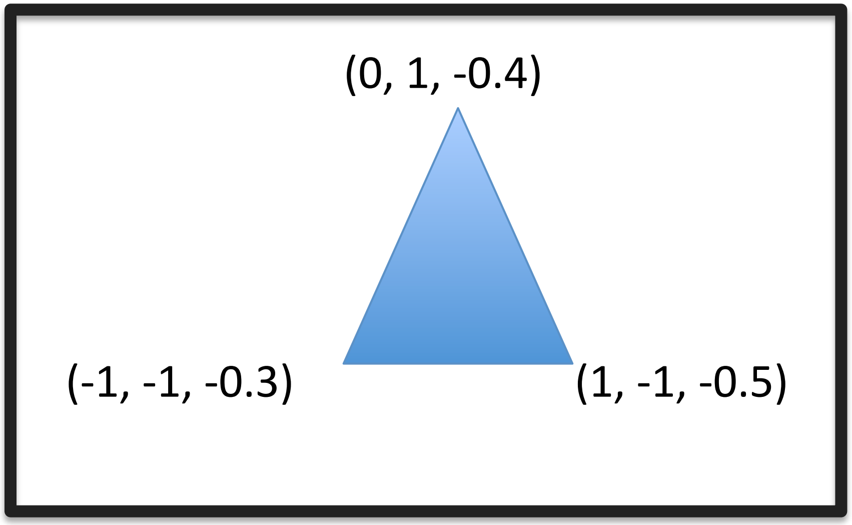


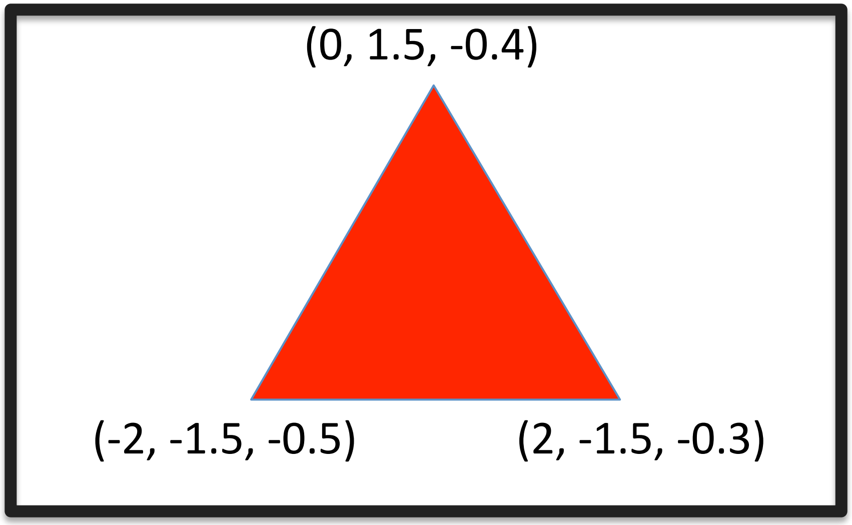
## Idea #2

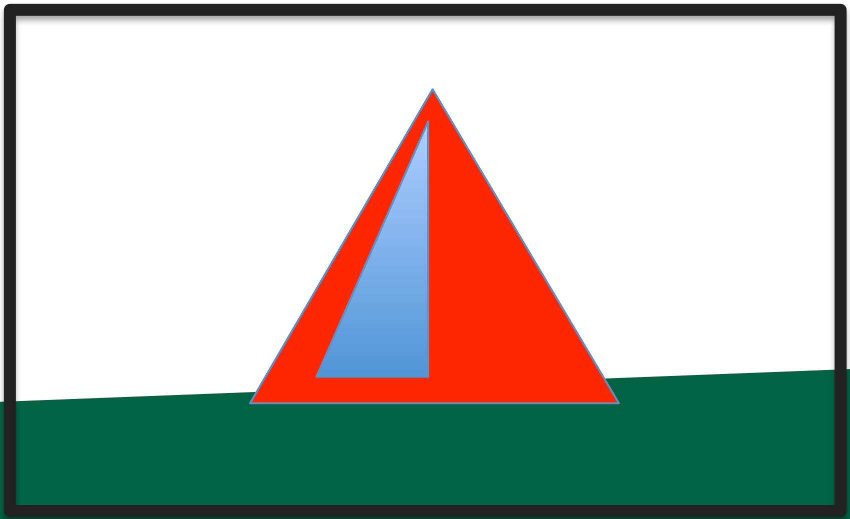
- Sort triangles "front to back" (based on Z)
- Render triangles in front to back order
  - Do not overwrite existing pixels.



## 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.

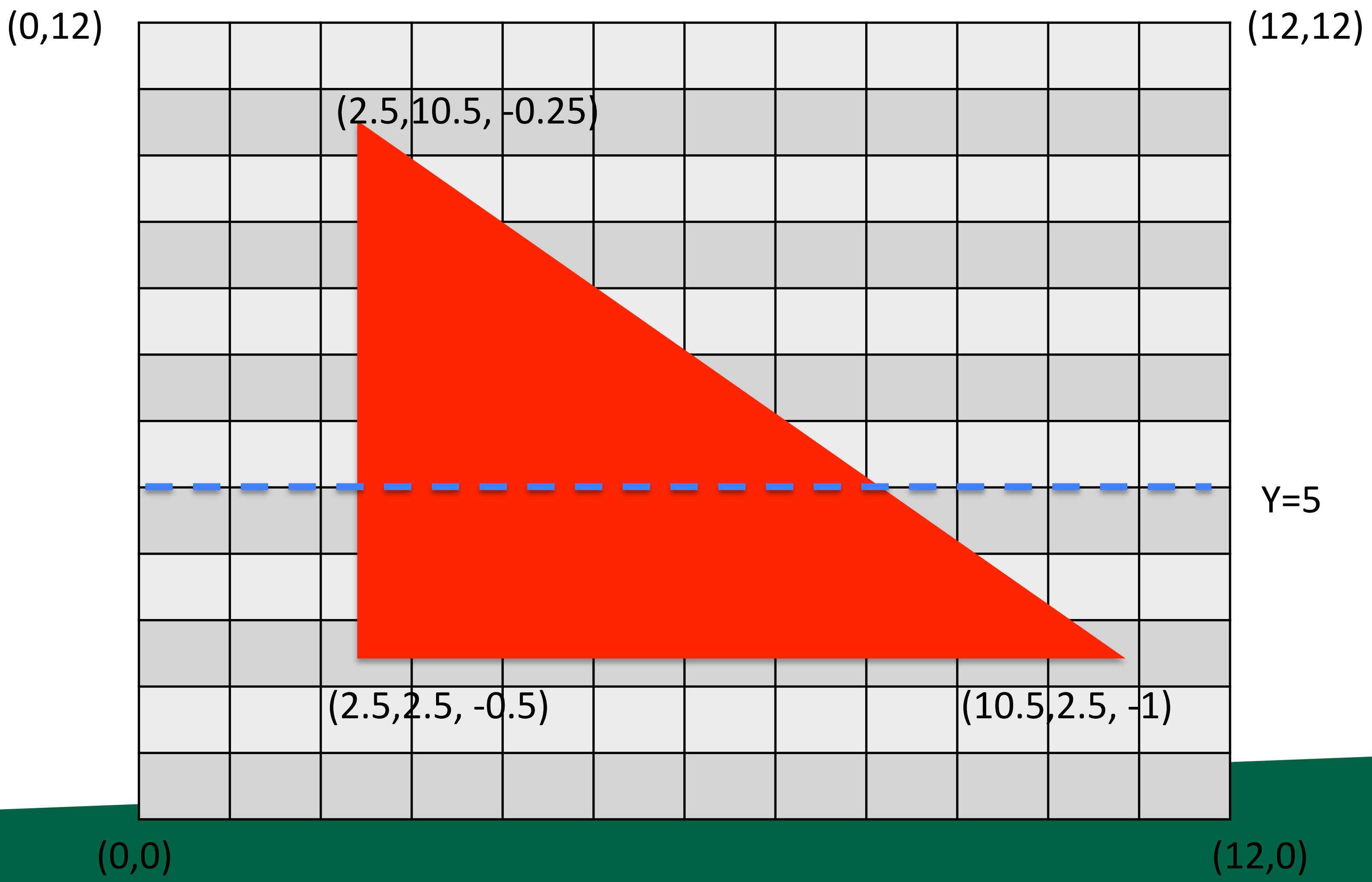
# Scanline algorithm

- 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

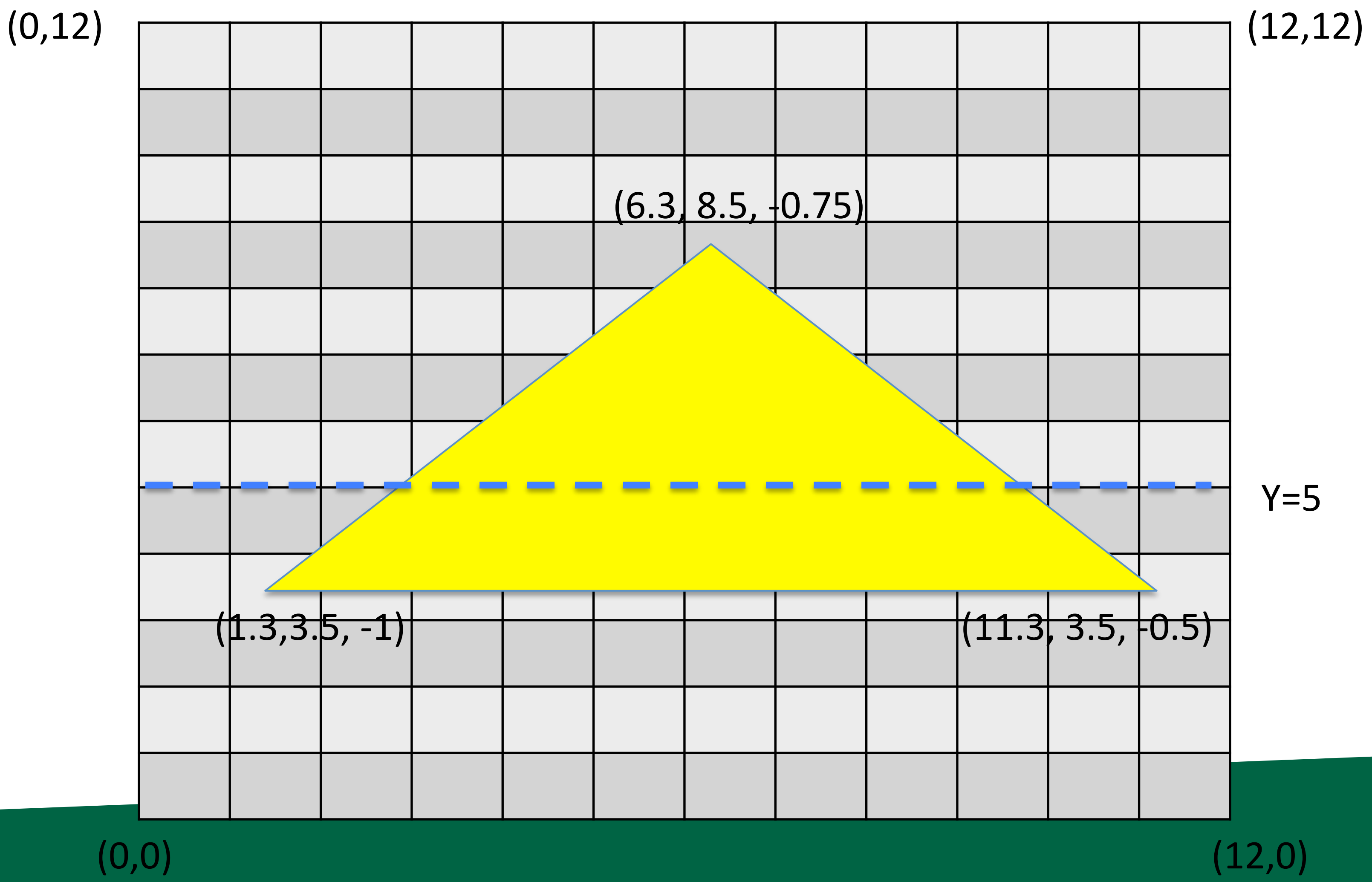
# Scanline algorithm w/ Z-Buffer

- 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 → 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)

# The Z-Buffer Algorithm: Example



# 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.