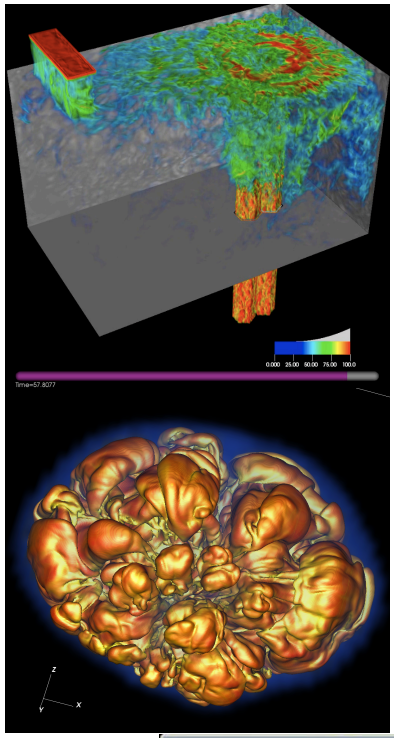
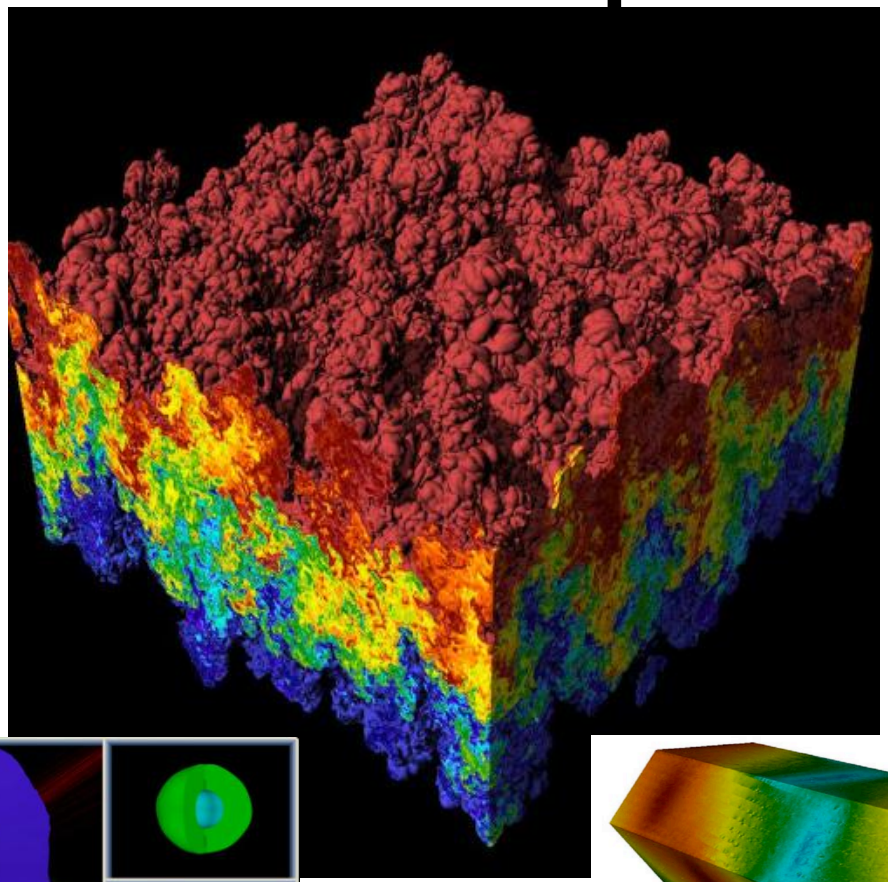
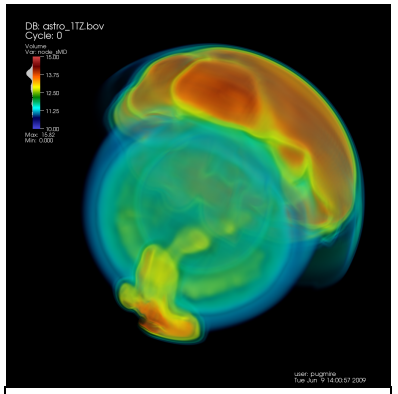
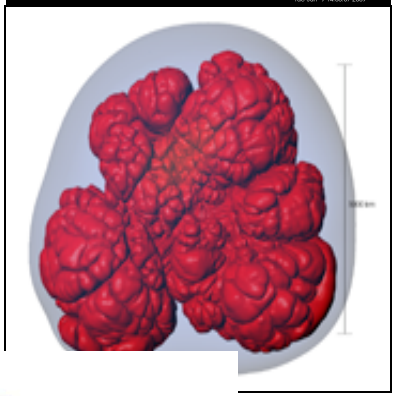
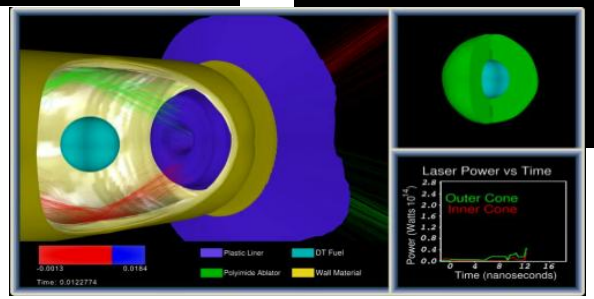
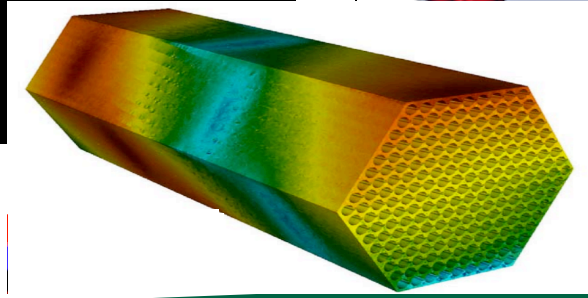
CIS 441/541: Intro to Computer Graphics
Lecture 4: Interpolation



January 22, 2019

Hank Childs, University of Oregon

Taylor…

No Class Tuesday, 1/29

- Will definitely be a YouTube lecture to replace that one.

Office Hours: Weeks 4-10

- Monday: 1-2 (Roscoe)
- Tuesday: 1-2 (Roscoe)
- Wednesday: 1-3 (Roscoe)
- Thursday: 1130-1230 (Hank)
- Friday: 1130-1230 (Hank)

Office Hours: Week 3

- Monday: 415-530 (Hank)
- Tuesday: 1-2, 2-3 (Roscoe)
- Wednesday: 1-3 (Roscoe)
- Thursday: 1130-1230 (Hank)
- Thursday: 1230-230 (Roscoe)

Timeline

- 1C: due Weds Jan 23rd
- 1D: assigned today, due Thurs Jan 31st
- 1E: assigned Thurs Jan 31st, due Weds Feb 6th
  - → will be extra support with this. Tough project.
- 1F: assigned Feb 7th, due Feb 19th
  - → not as tough as 1E
- 2A: will be assigned during week of Feb 11th

Sun	Mon	Tues	Weds	Thurs	Fri	Sat
Jan 20	Jan21	Jan 22 Lec4	Jan 23 1C due	Lec 5 1D assigned	Jan 25	Jan 26
Jan 27	Jan 28	Jan 29 (YouTube)	Jan 30	Lec 6 1D due 1E assigned	Feb 1	Feb 2
Feb 3	Feb 4	Feb 5 Lec 7	Feb 6	Lec 8 1E due 1F assigned	Feb 8	Feb 9

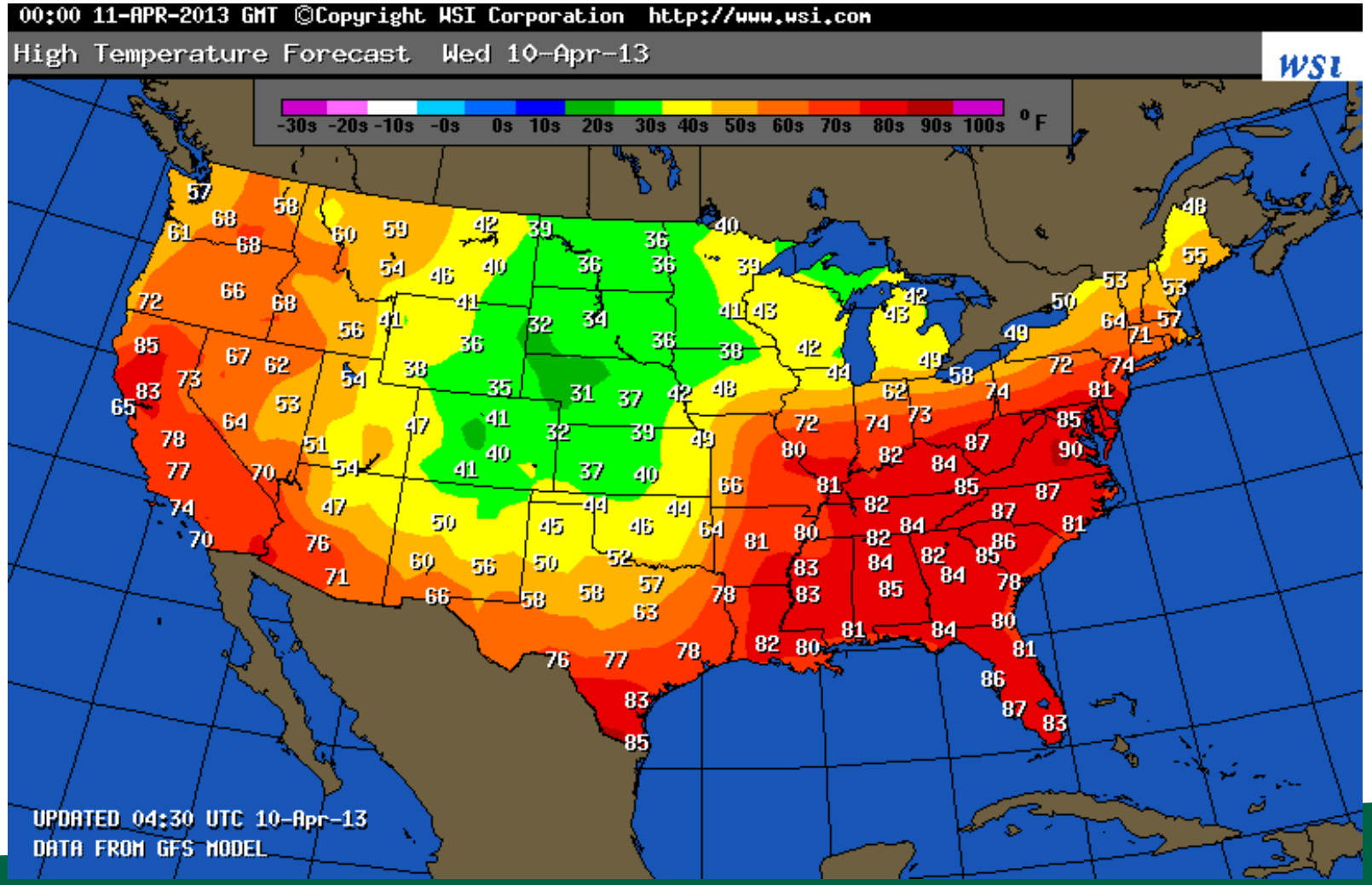
Likely: pre-SuperBowl OH

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;
```

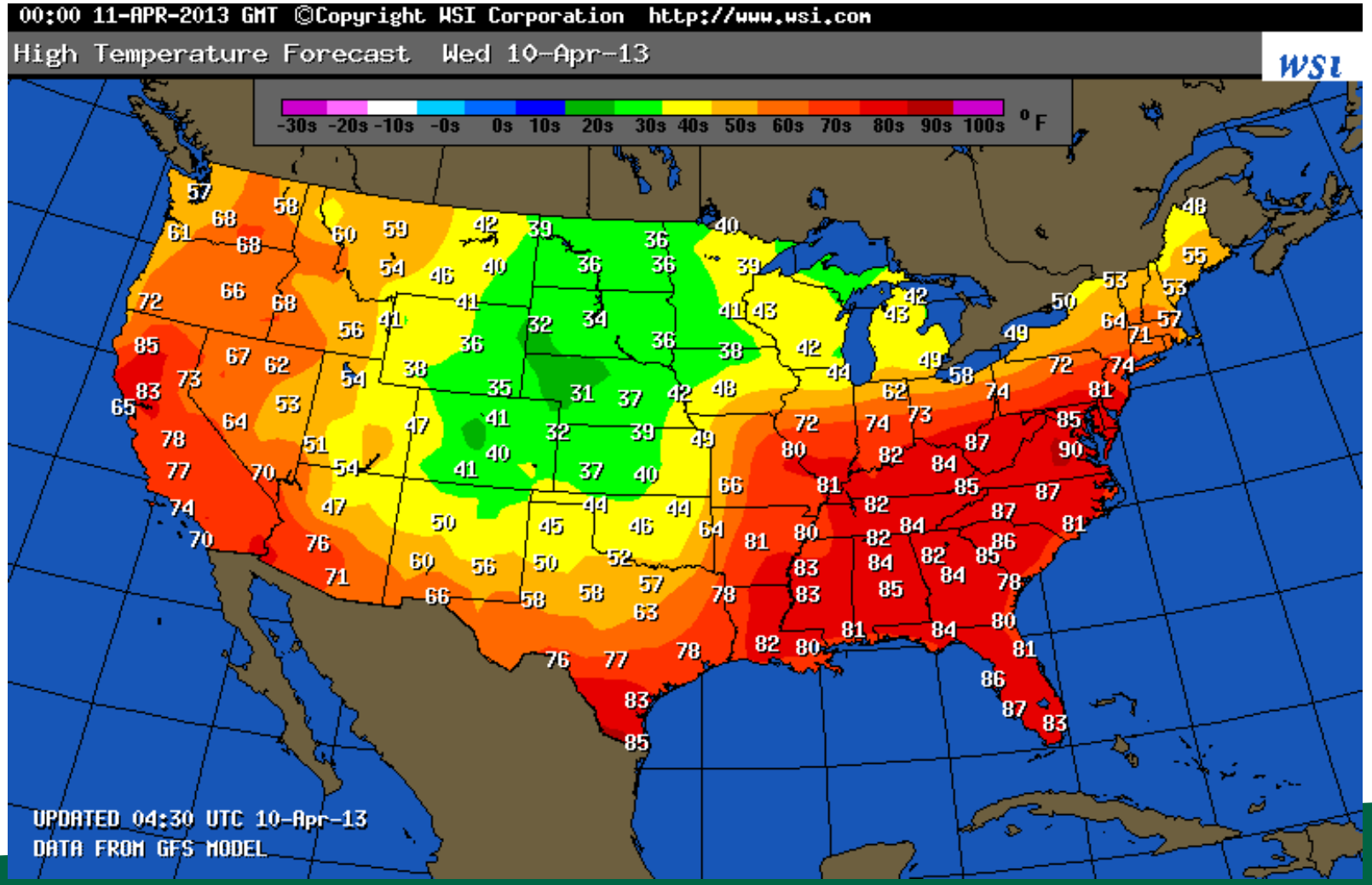
```
for (int i = 0 ; i < triangles.size() ; i++)
{
     triangleID = i;  // triangleID is a global
     Triangle &t = triangles[i];
     //t.Print(cerr);
     RasterizeTriangle(t, screen);
}
```

What is a field?



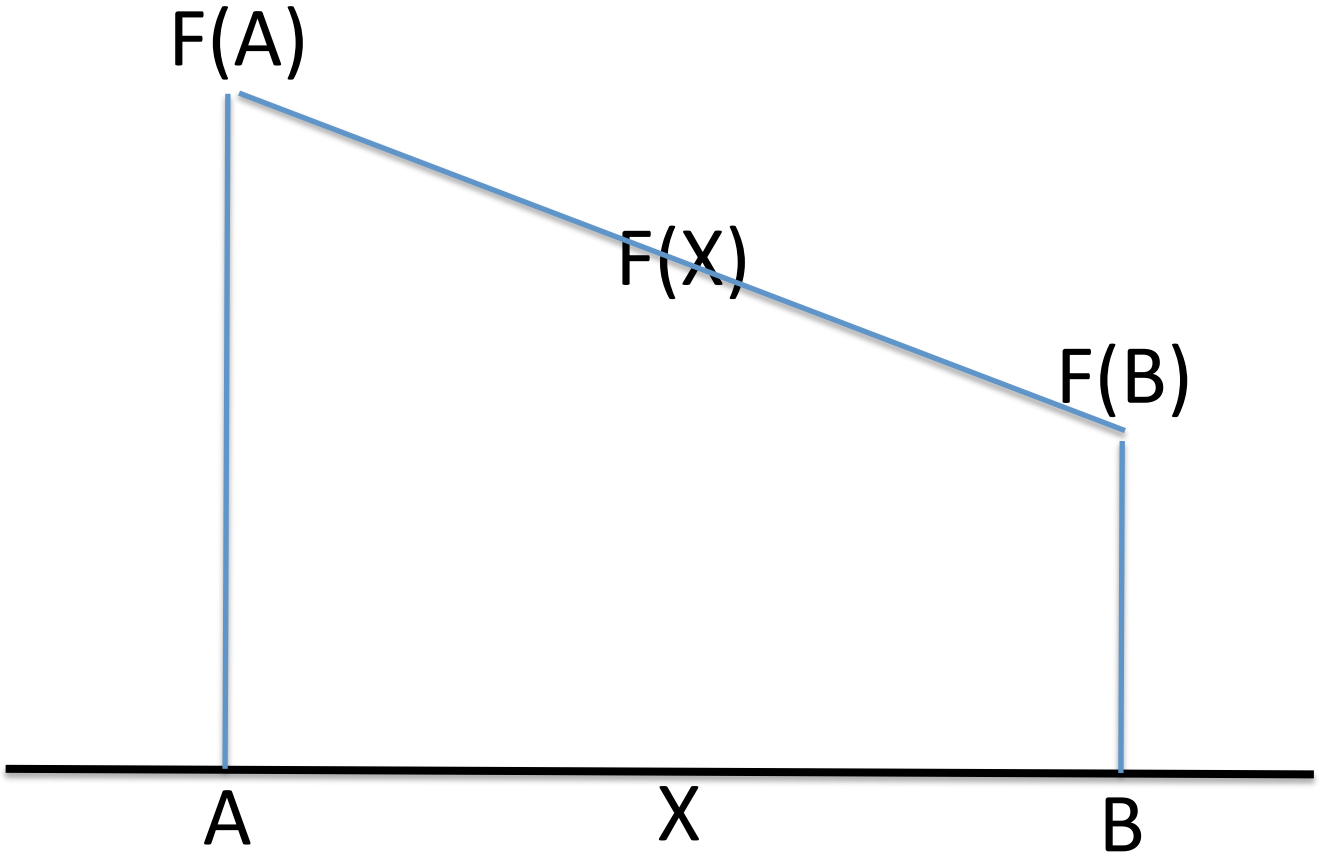
Example field (2D): temperature over the United States

How much data is needed to make this picture?



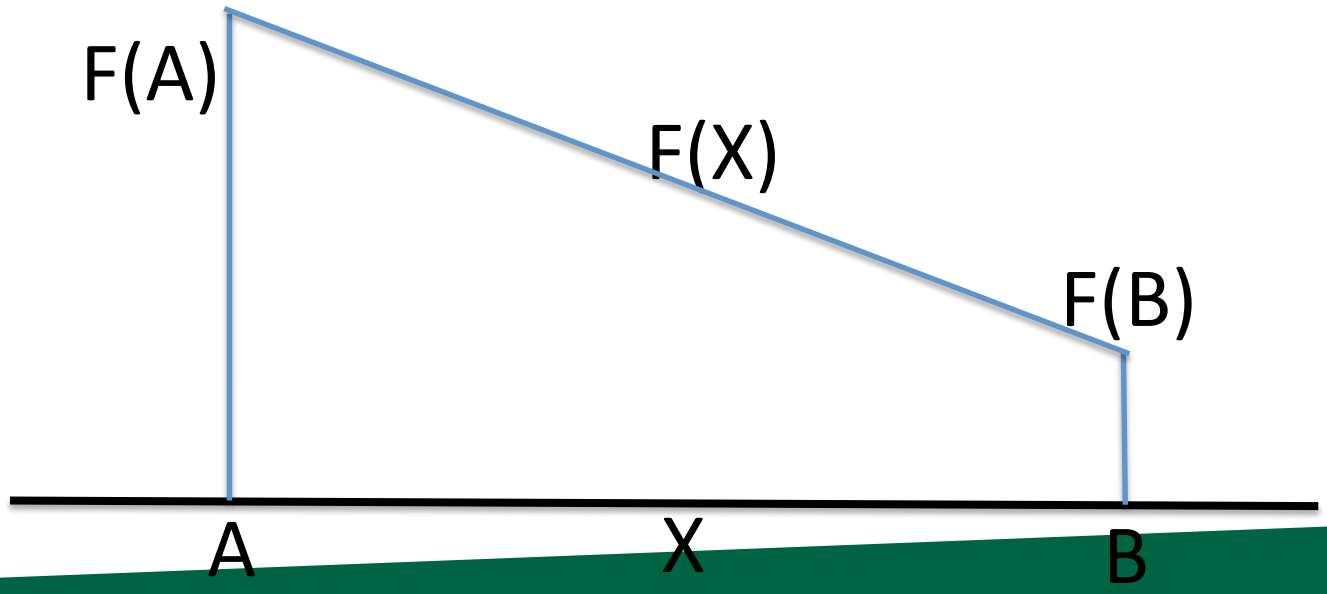
Example field (2D): temperature over the United States

Linear Interpolation for Scalar Field F



Linear Interpolation (LERP) 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)

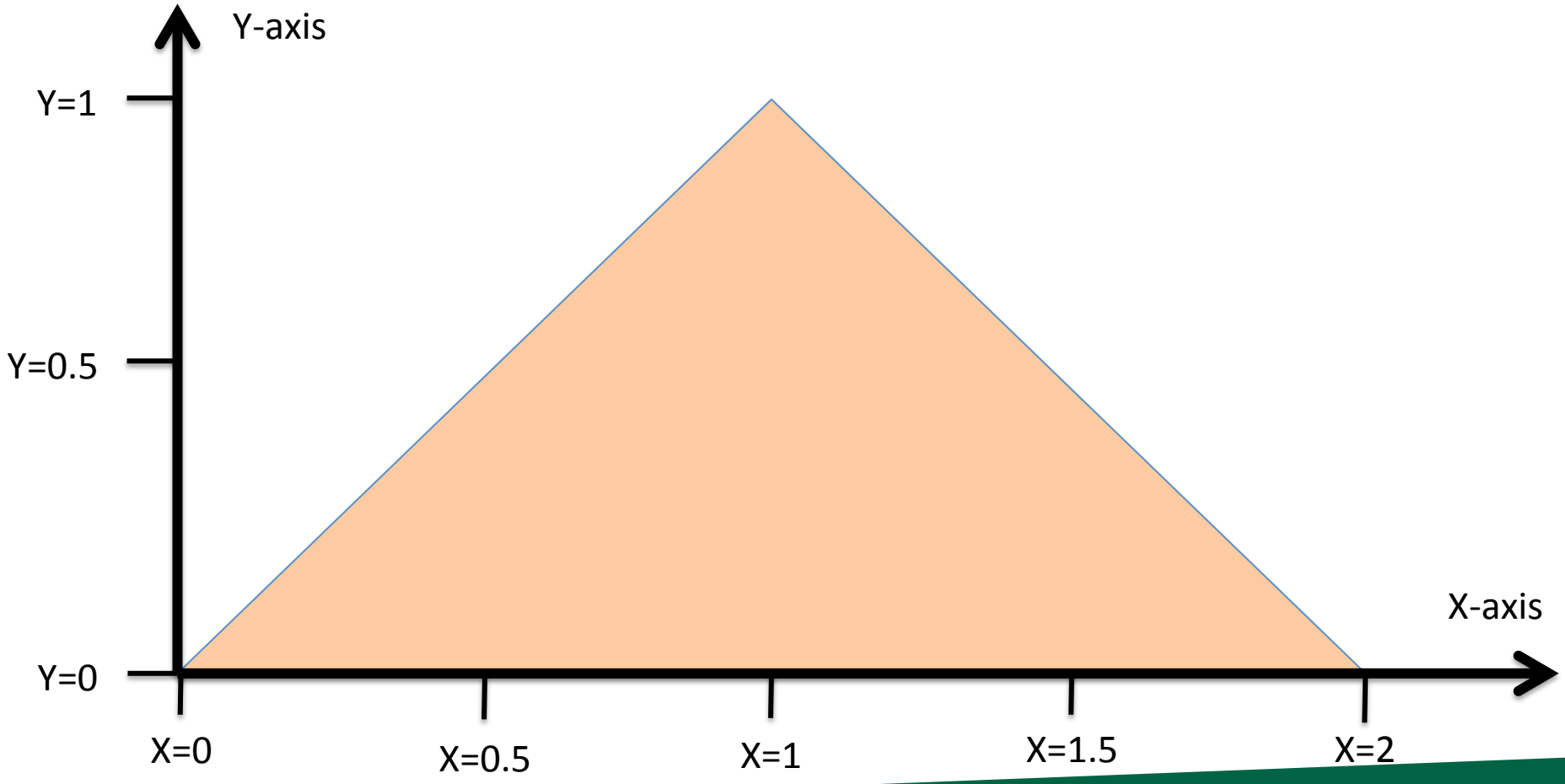


Quiz Time #4

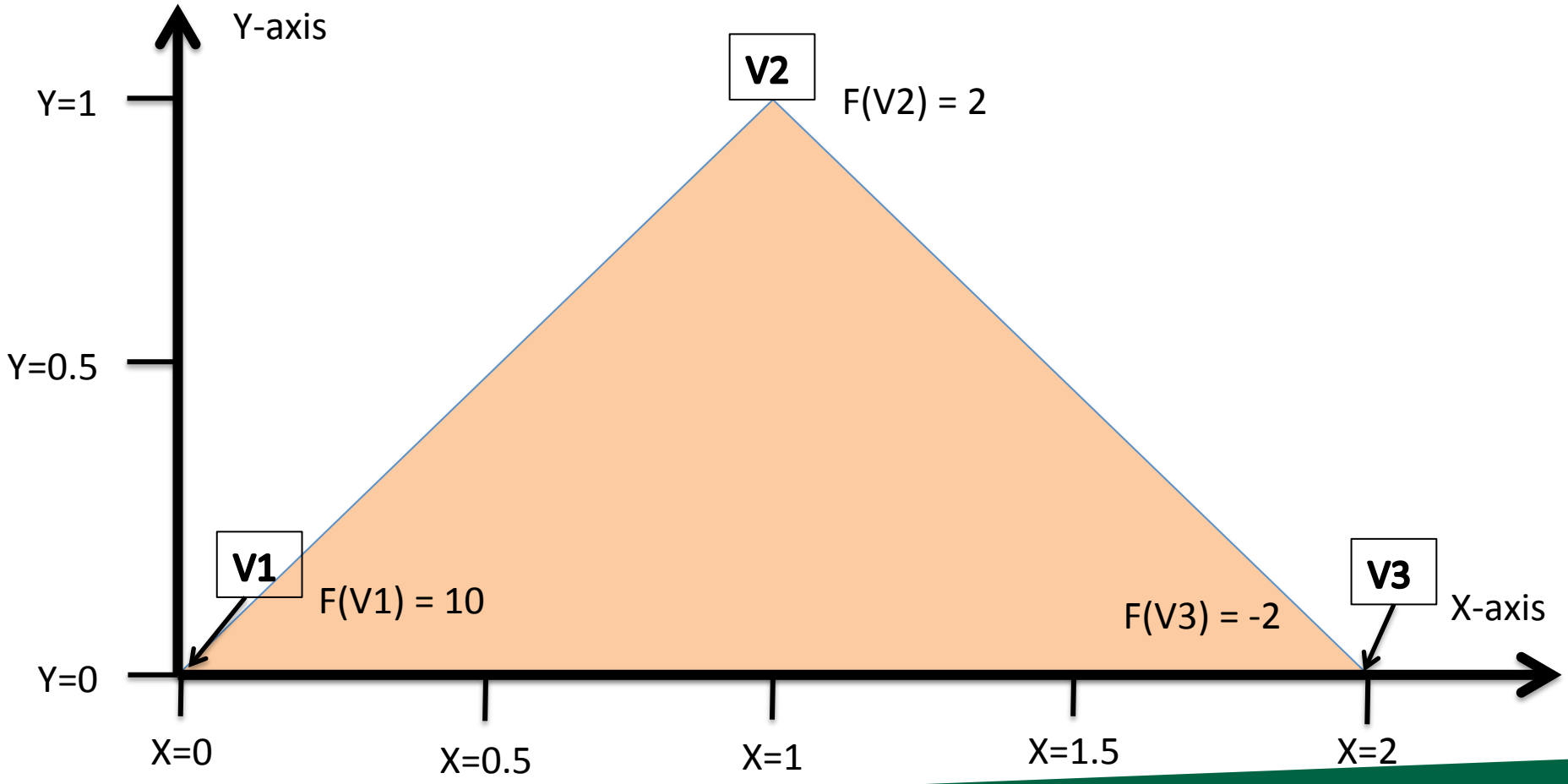
- F(3) = 5, F(6) = 11
- What is F(4)? = 5 + (4-3)/(6-3)*(11-5) = 7

- 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)

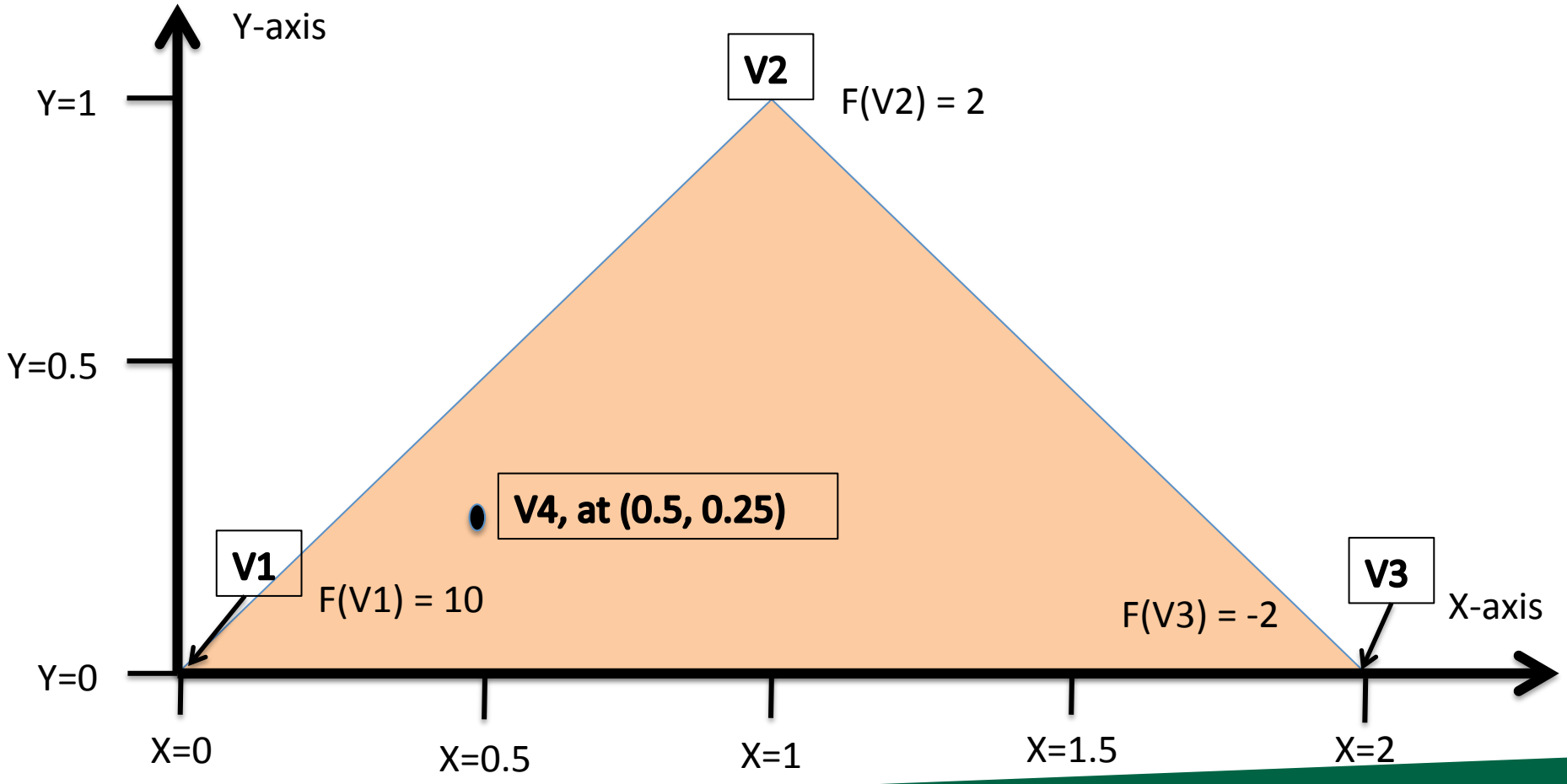
Consider a single scalar field defined on a triangle.



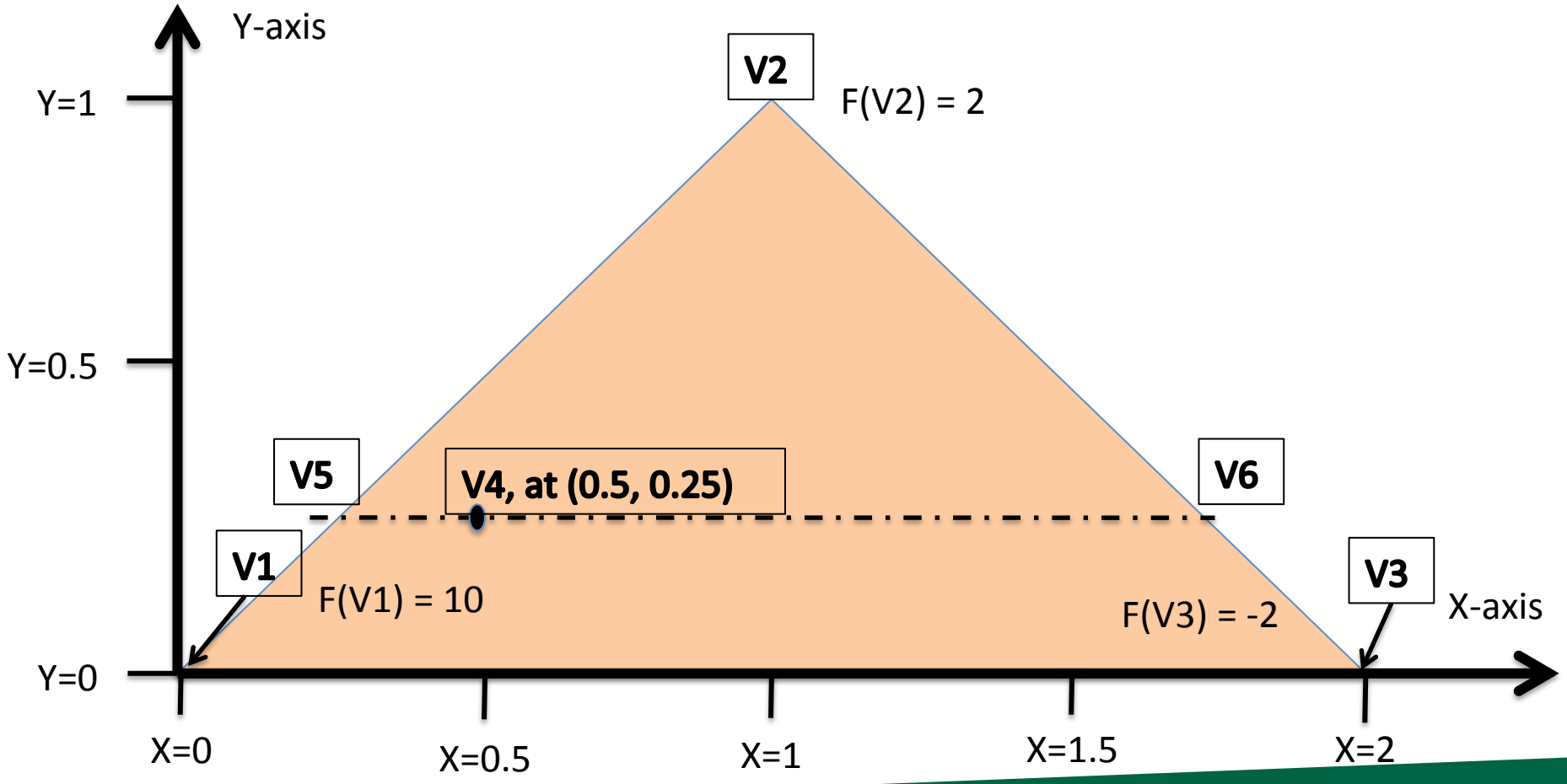
Consider a single scalar field defined on a triangle.



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
V3
F(V3) = -2
X-axis
X=0
X=0.5
X=1
X=1.5
X=2

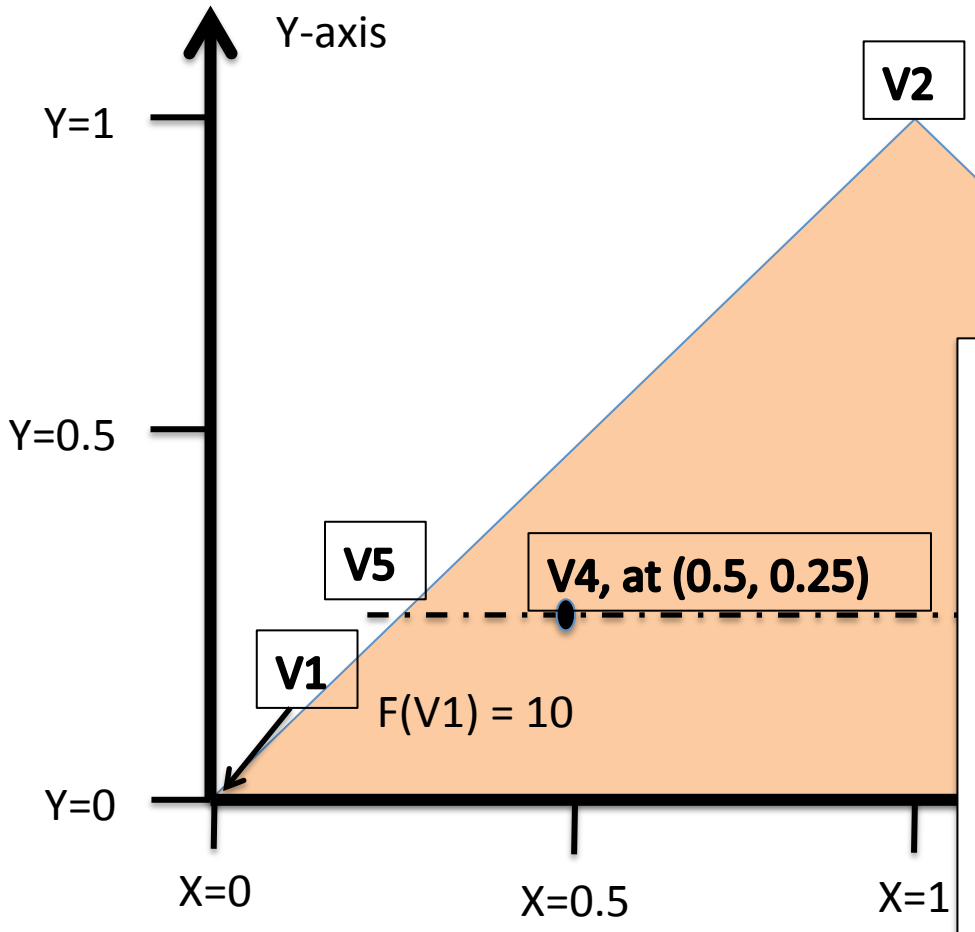
What is F(V4)?


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

(Here's the slides I screwed up on last week)

What is the X-location of V5?

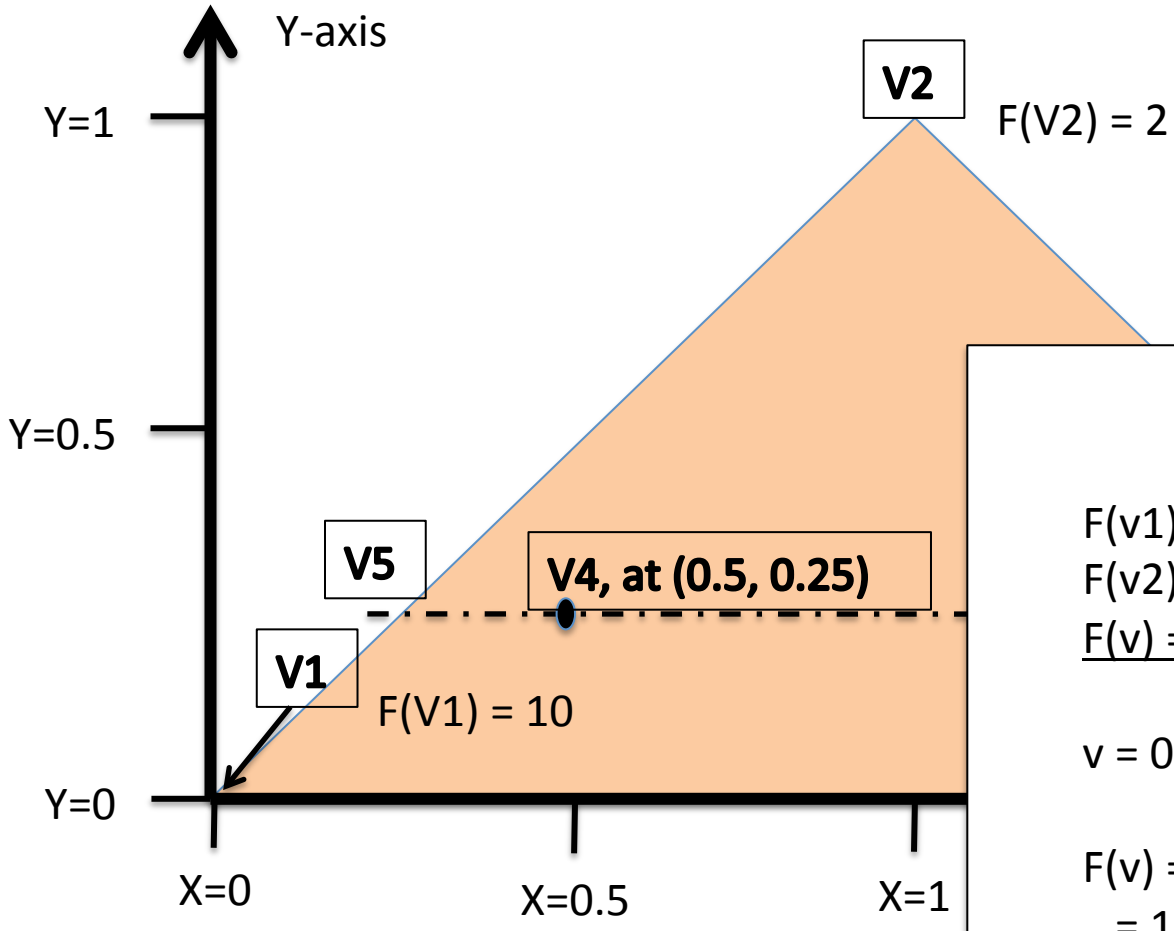


F(v1) = A → F(0) = 0
F(v2) = B → F(1) = 1
F(v) = A + ((v-v1)/(v2-v1))*(B-A):

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 → F(0) = 10
F(v2) = B → F(1) = 2
F(v) = A + ((v-v1)/(v2-v1))*(B-A):

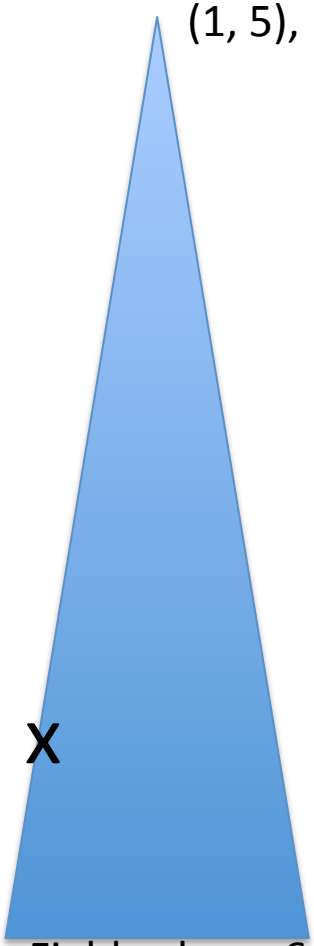
v = 0.25, find F(v)

F(v) = 10 + ((0.25-0)/(1-0))*(2-10)
= 10 + 0.25*-8 = 10 -2 = 8

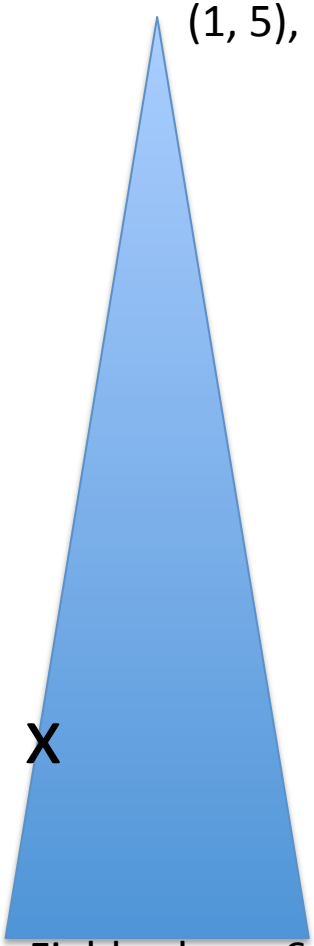
Why did I screw up?

- The logic in the slide?
- → interpolate to X
- → once you know X, use that to find V
- Why did this screw me up?
- → you can interpolate to V directly

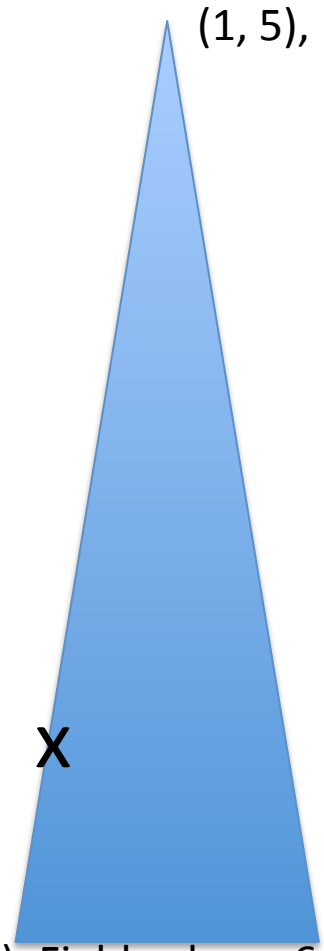
- The previous slide is particularly confusing because it is along the line Y=X.



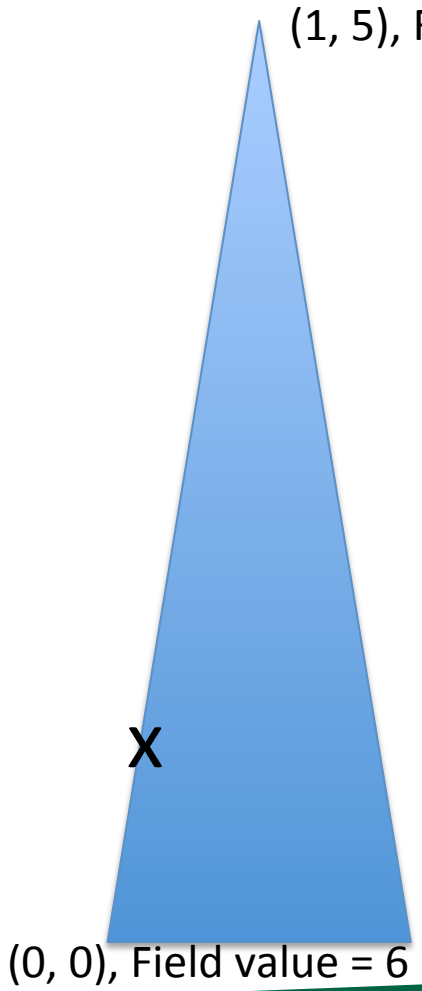
If y-value == 1, then two questions:
What is X value?
What is field value?



If y-value == 1, then two questions:
What is X value?
What is field value?

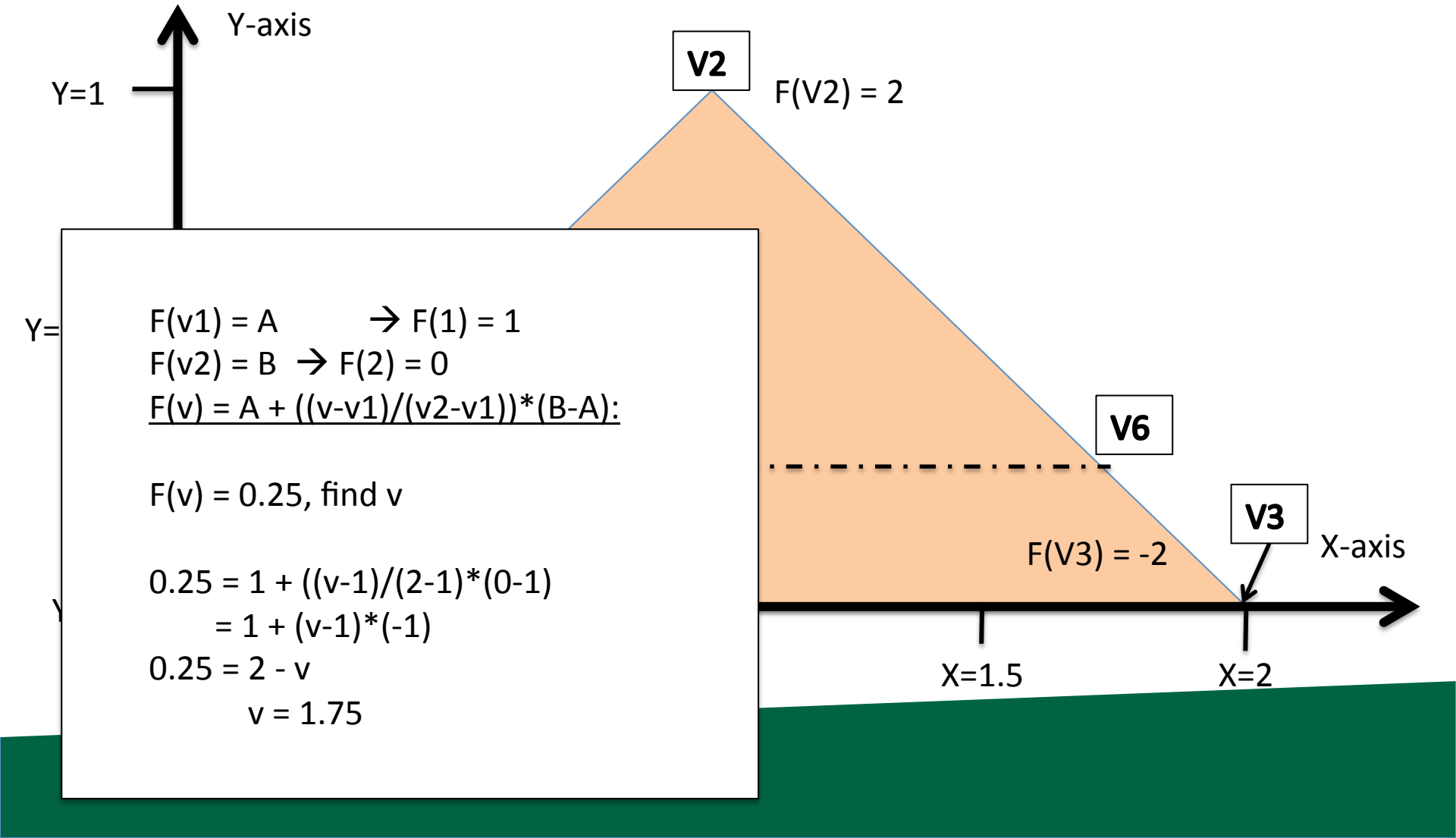


If y-value == 1, then two questions:
What is X value?
→
Option 1: solve for line (what we did before)
(could actually use LERP formula too)



If y-value == 1, then two questions:
What is field value?
→
Use LERP formula
F(1) = F(0) + t*(F(5)-F(0))
= 6 + 0.2 *(3-6) = 5.4
(all F values are the y-coordinates → F(0) means the field value when Y is 0
t = (1-0)/(5-0) = 0.2
= (the y-coord of the point we want to find
- the y-coord of one point we know)
/ (the y—coord of the other point we know
- the y-coord of the first point we know)

What is the X-location of V6?

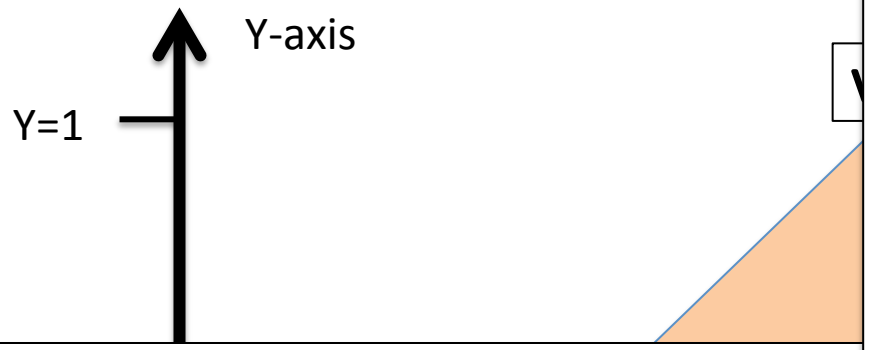


F(v1) = A → F(1) = 1
F(v2) = B → F(2) = 0
F(v) = A + ((v-v1)/(v2-v1))*(B-A):

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?

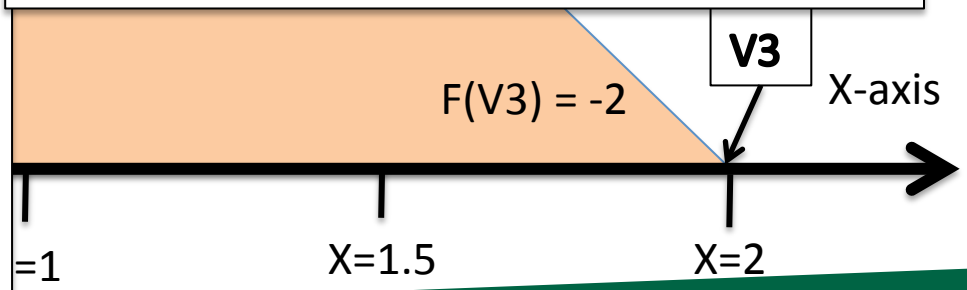


This one is really bad:
Simpler version:
T = 0.25 → (0.25-0)/(1-0)
F(v) = -2 + 0.25(2 - -2) → -2+0.25*4 = -1

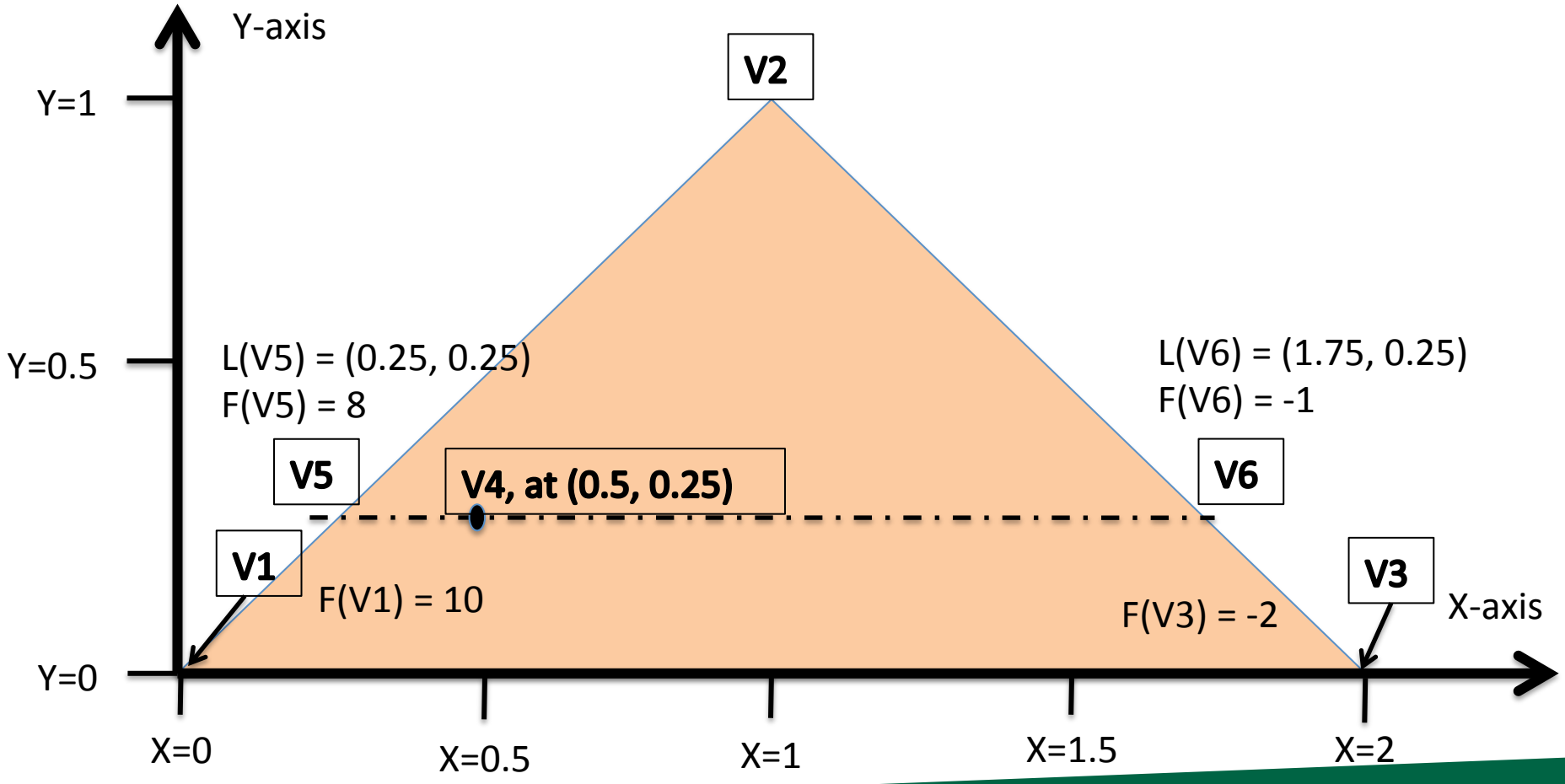
F(v1) = A → F(1) = 2
F(v2) = B → F(2) = -2
F(v) = A + ((v-v1)/(v2-v1))*(B-A):

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?

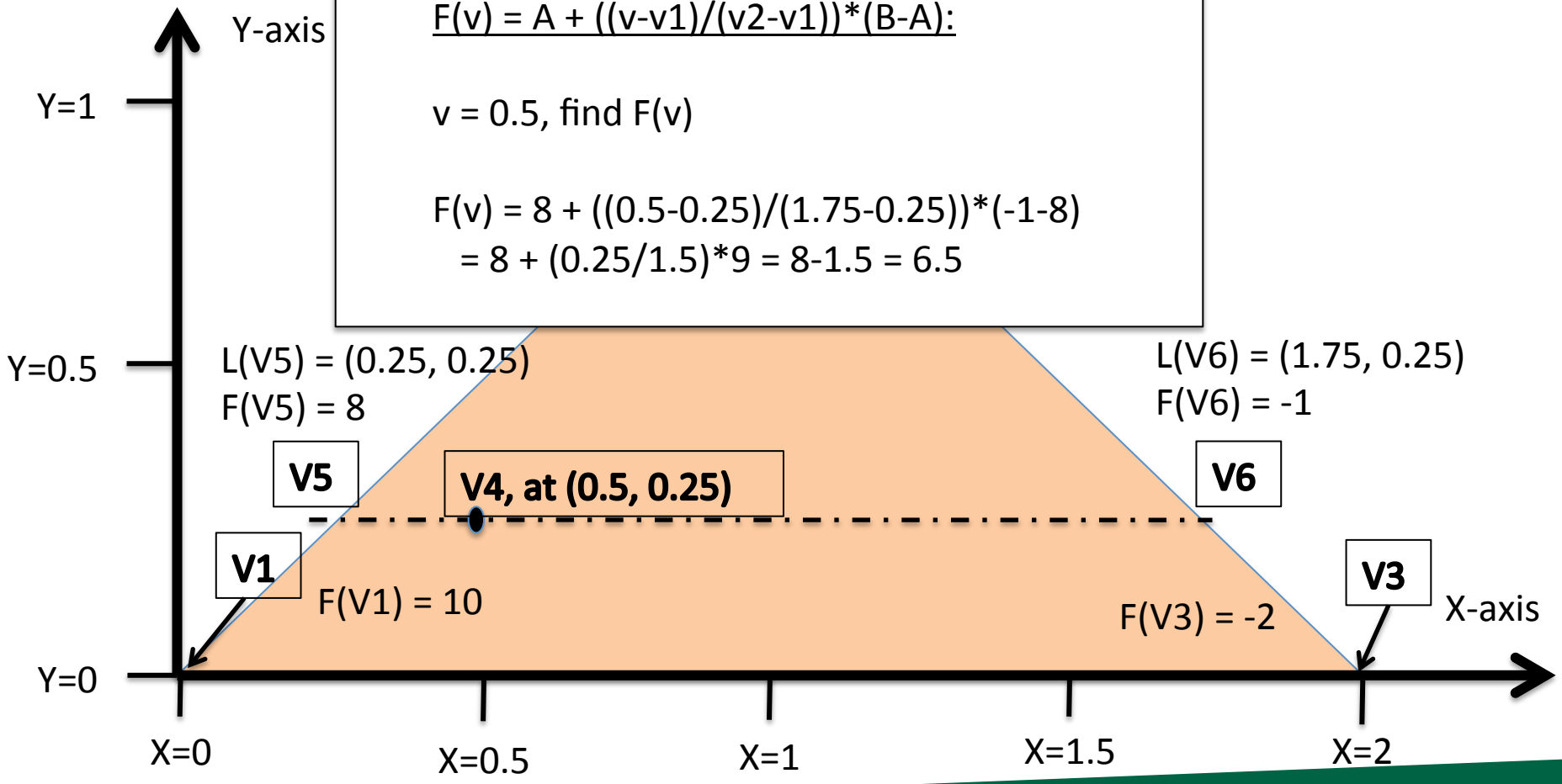


What is the F value of V5?

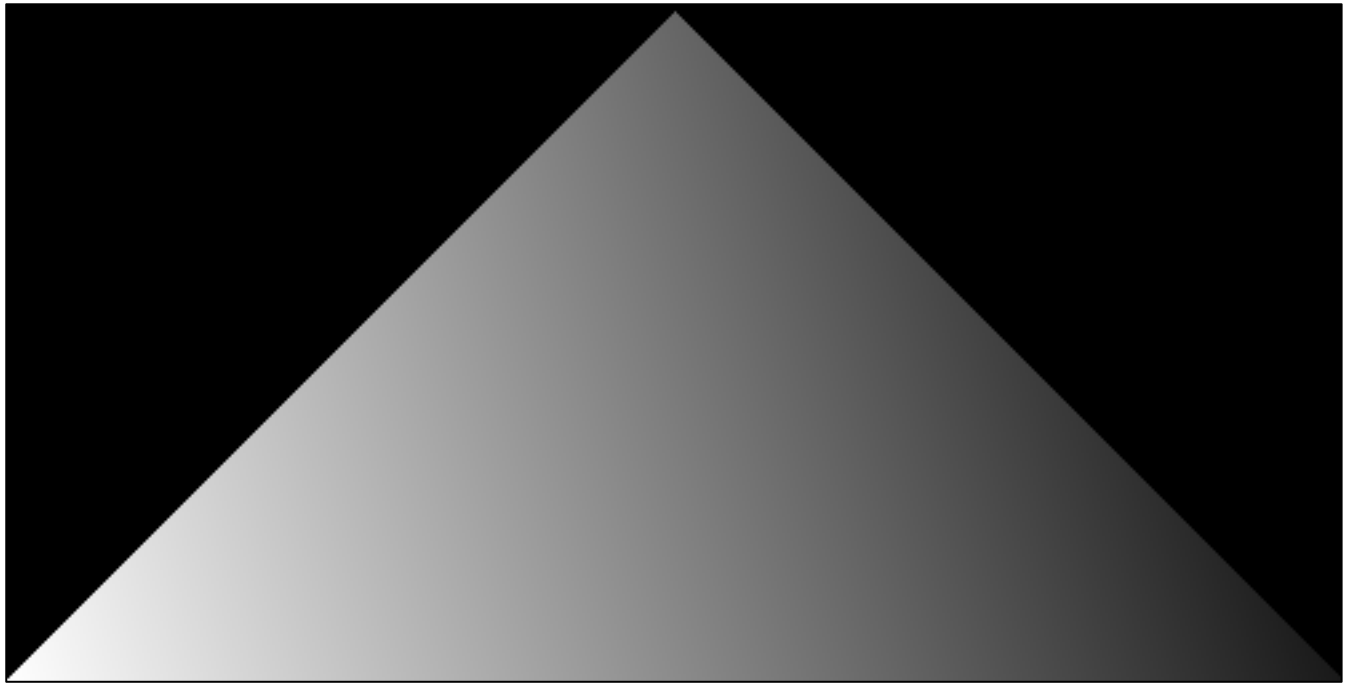
F(v1) = A → F(0.25) = 8
F(v2) = B → F(1.75) = -1
F(v) = A + ((v-v1)/(v2-v1))*(B-A):

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

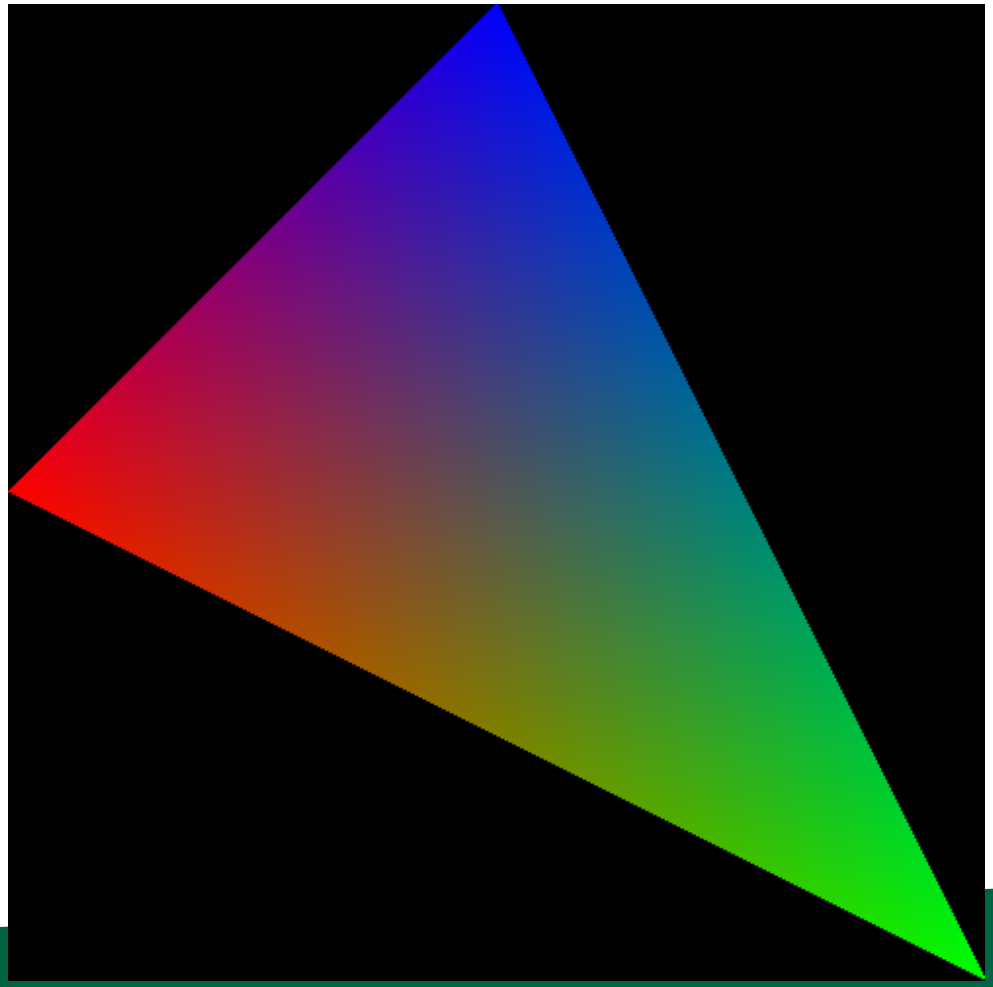


How do you think this picture was made?

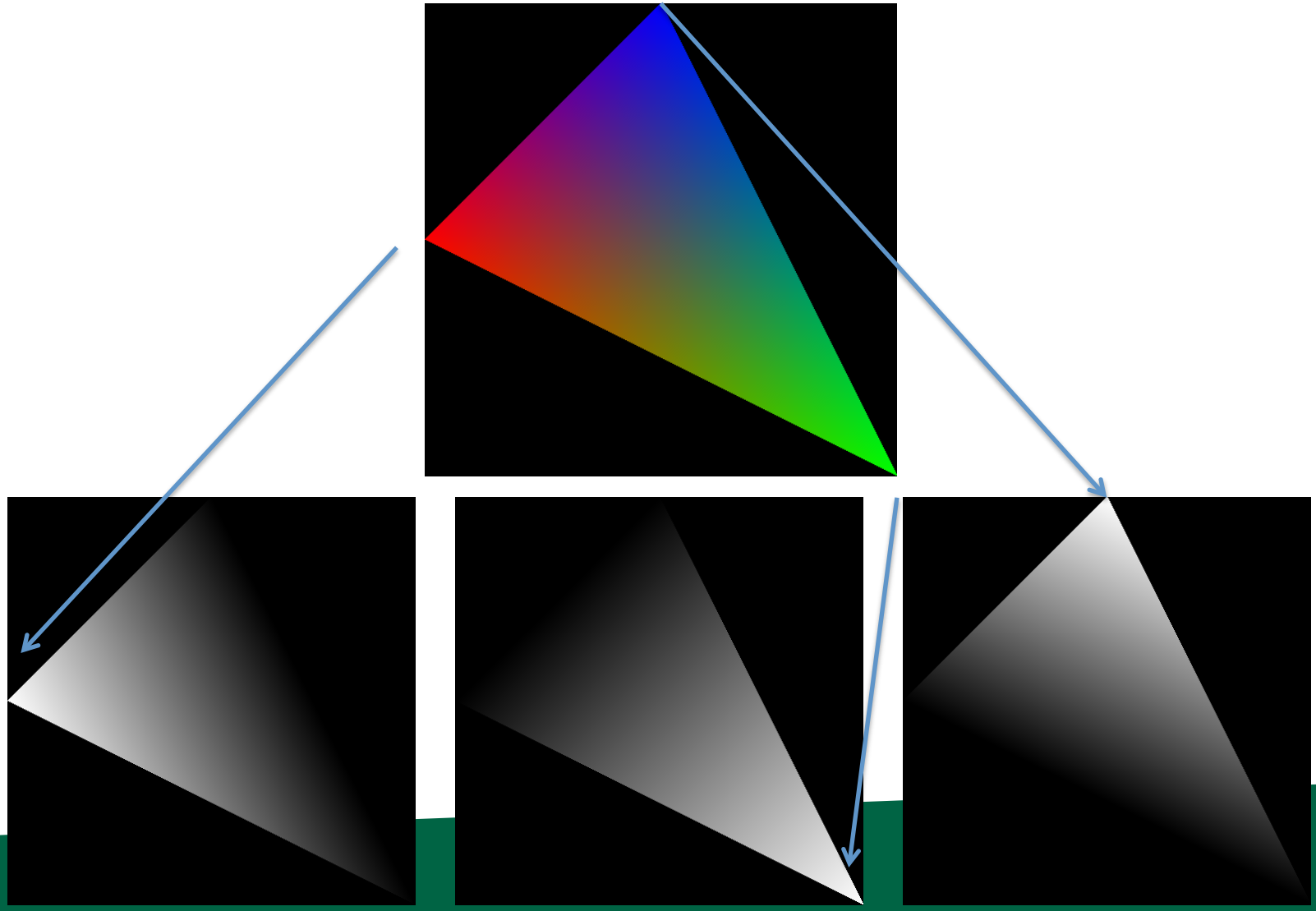
Now We Understand Interpolation
Let's Use It For Two New Ideas:
Color Interpolation
& Z-buffer Interpolation

Colors

What about triangles that have more than one color?



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



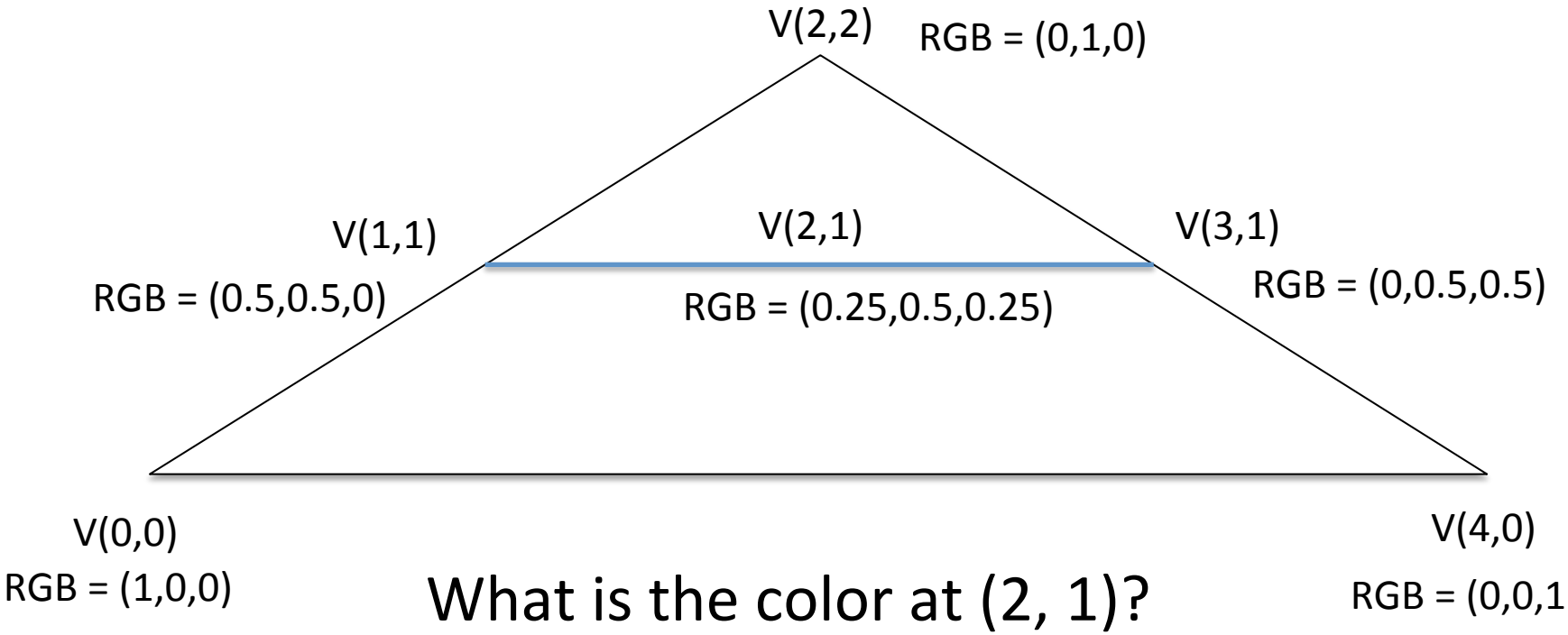
Scanline algorithm

- 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!

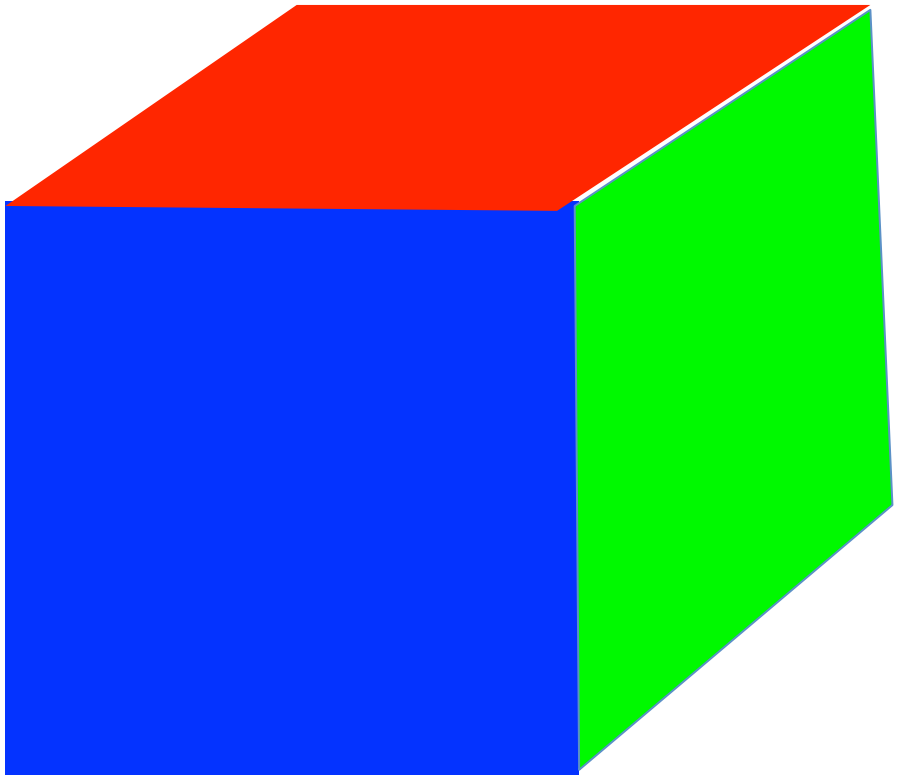
Important

- 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….

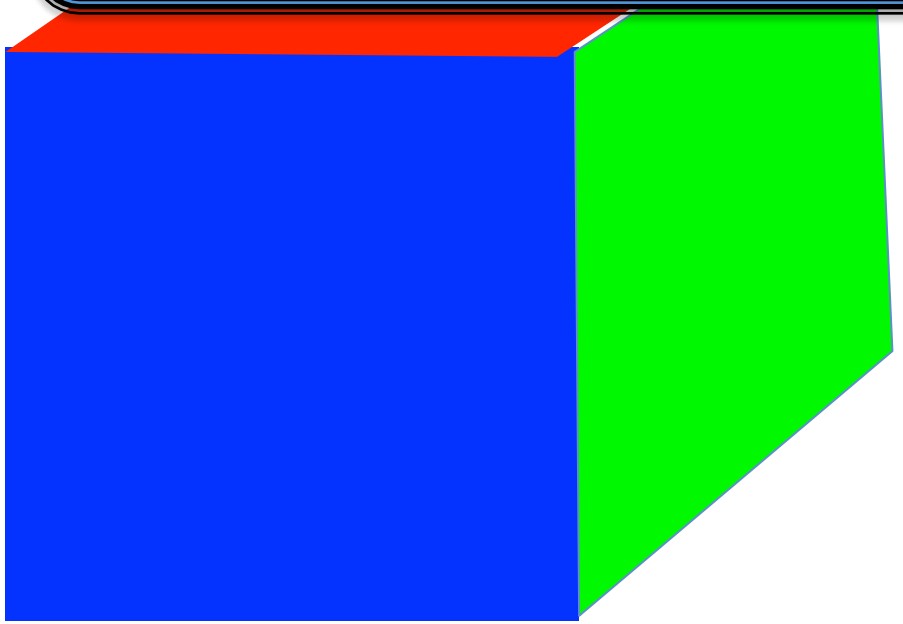


Face	Color
Front	Blue
Right	Green
Top	Red
Back	Yellow
Left	Purple
Bottom	Cyan

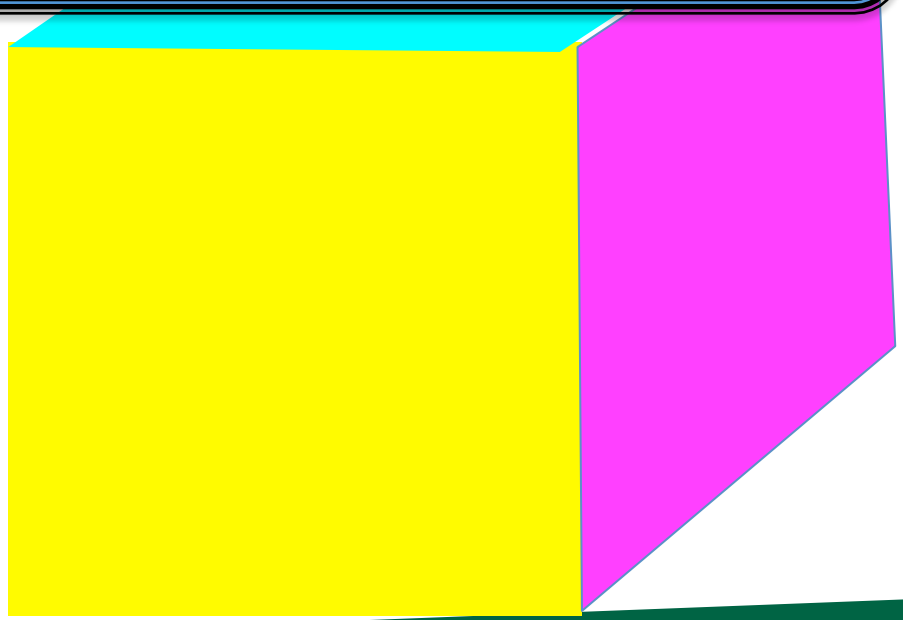
View from “front/top/right” side

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?

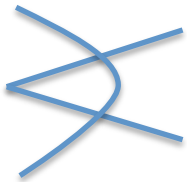


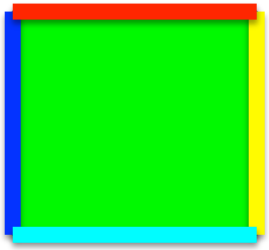
View from “front/top/right” side



View from “back/bottom/left” side

Consider a scene from the right side



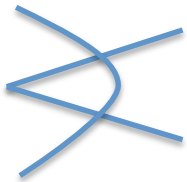


Camera/eyeball

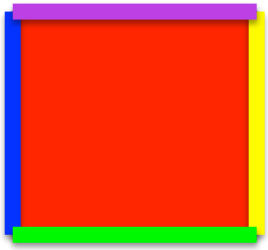
Camera oriented directly at Front face,
seen from the Right side

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

Consider the scene from the top side



Camera/eyeball

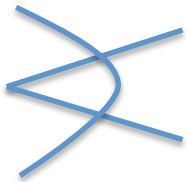


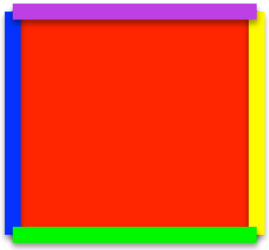
Camera oriented directly at Front face, seen from the Top side

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

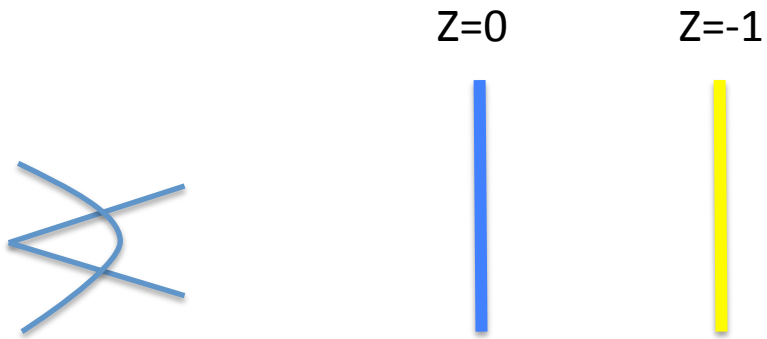
- Project 1B,1C:

```
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

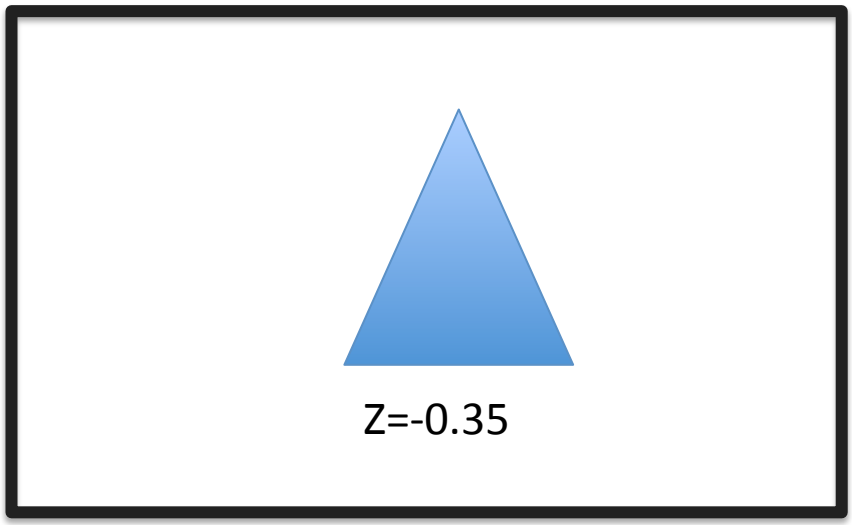
Face	Color
Front	Blue
Right	Green
Top	Red
Back	Yellow
Left	Purple
Bottom	Cyan

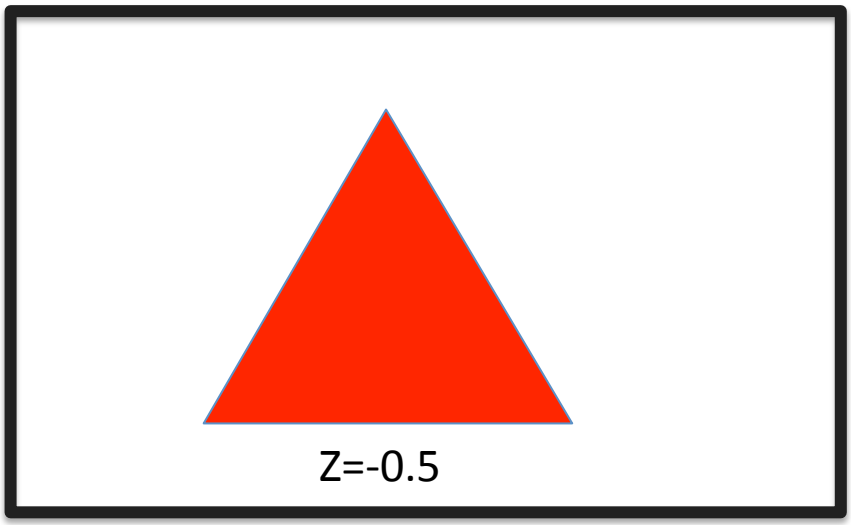
Using depth when rendering

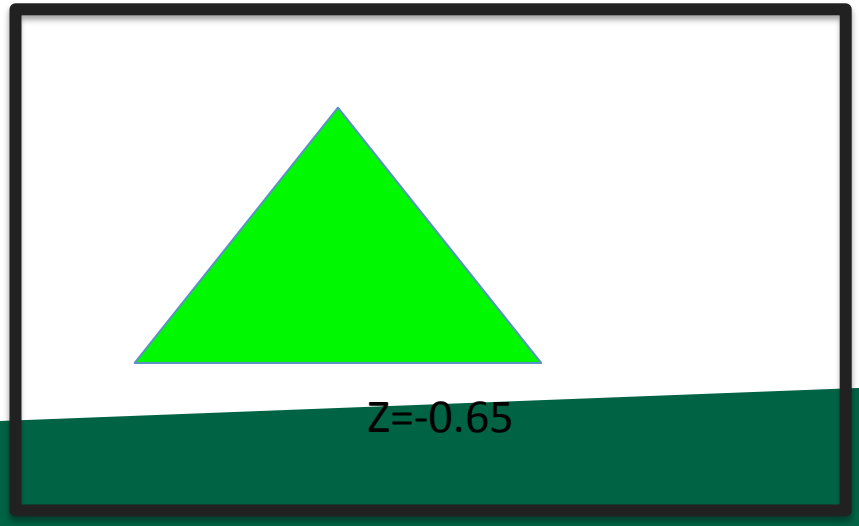
- Use Z values to guide which geometry is displayed and which is obscured.

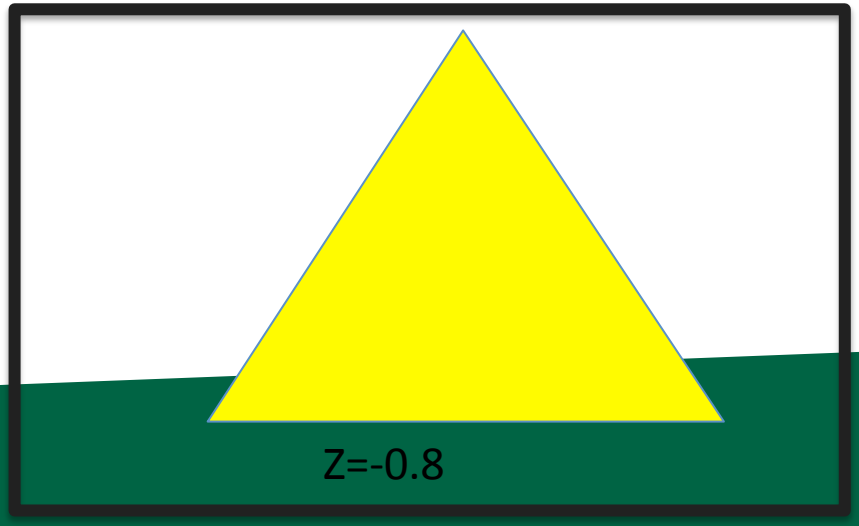
- Example….

Consider 4 triangles with constant Z values



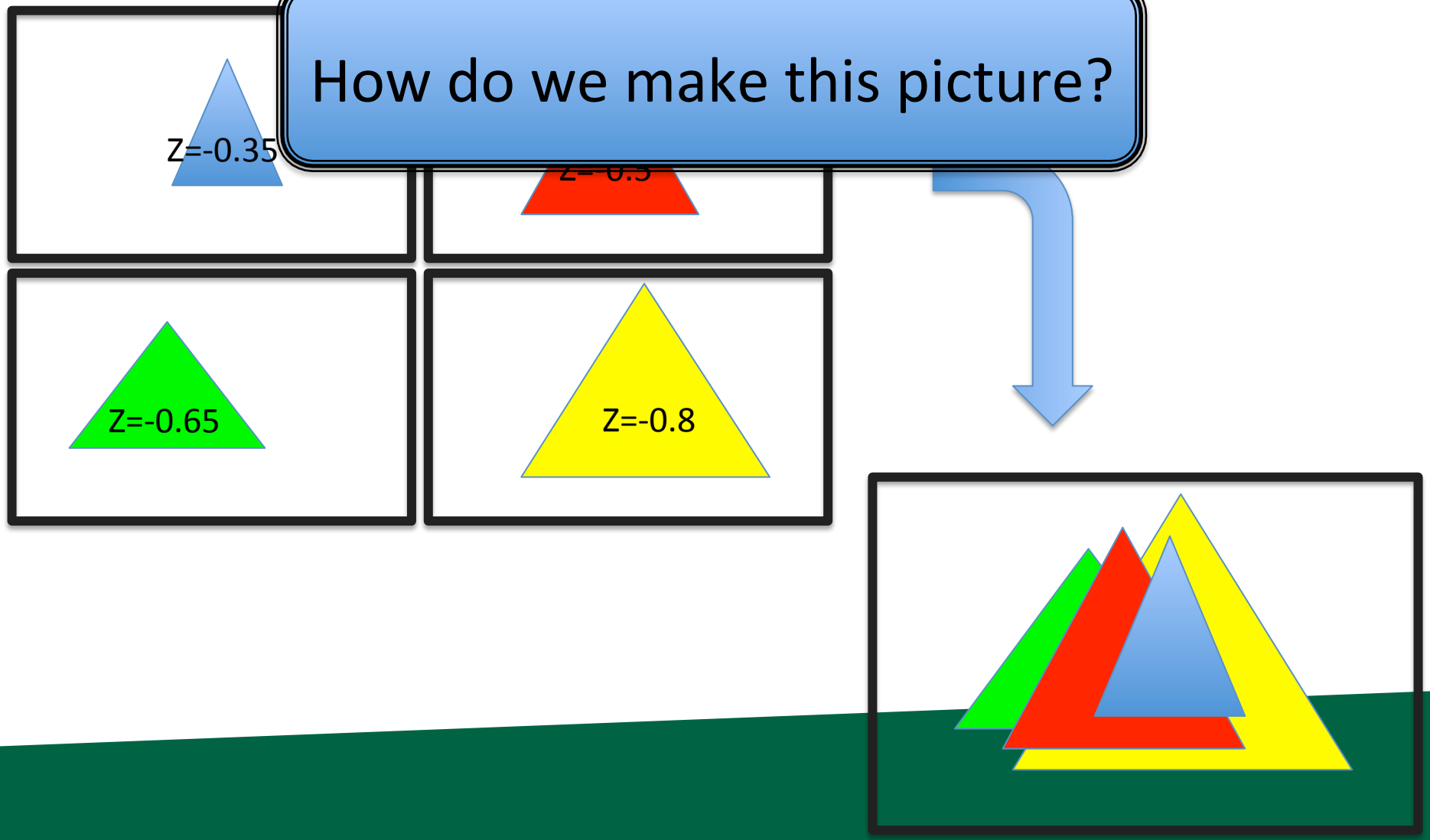






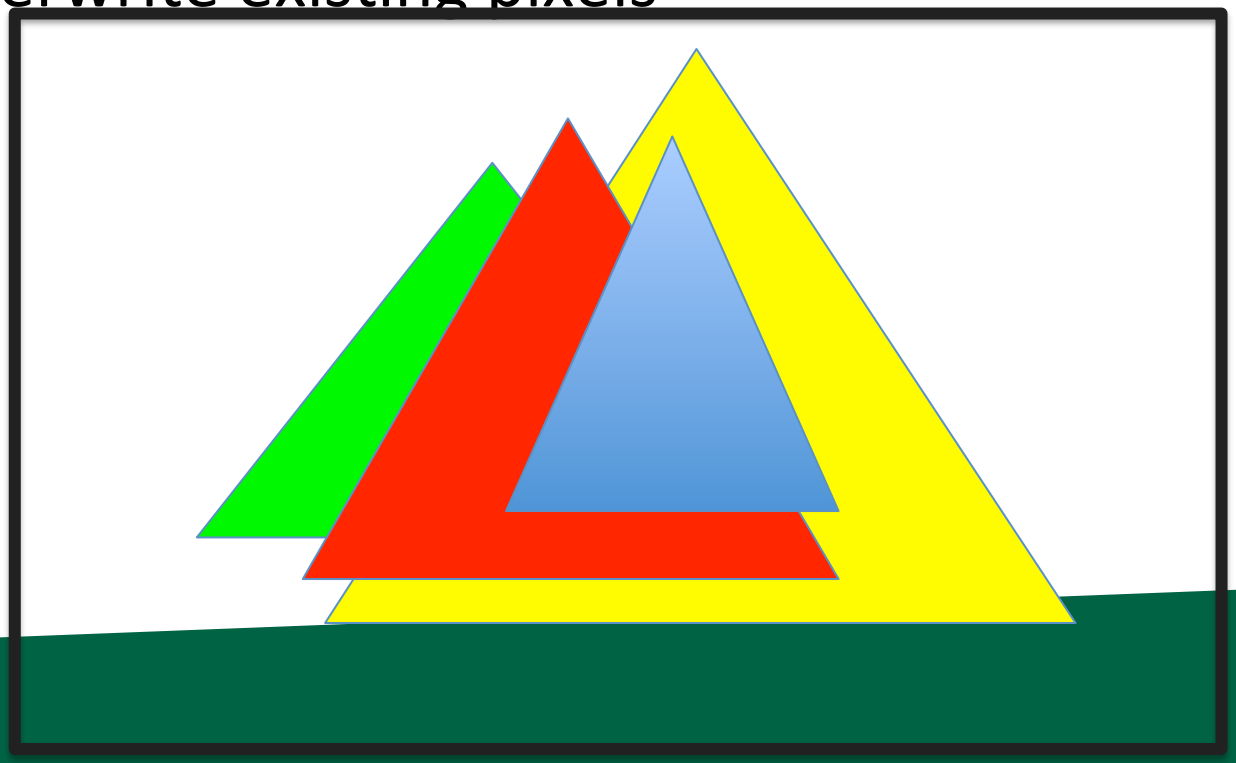
Consider 4 triangles with constant Z values

How do we make this picture?



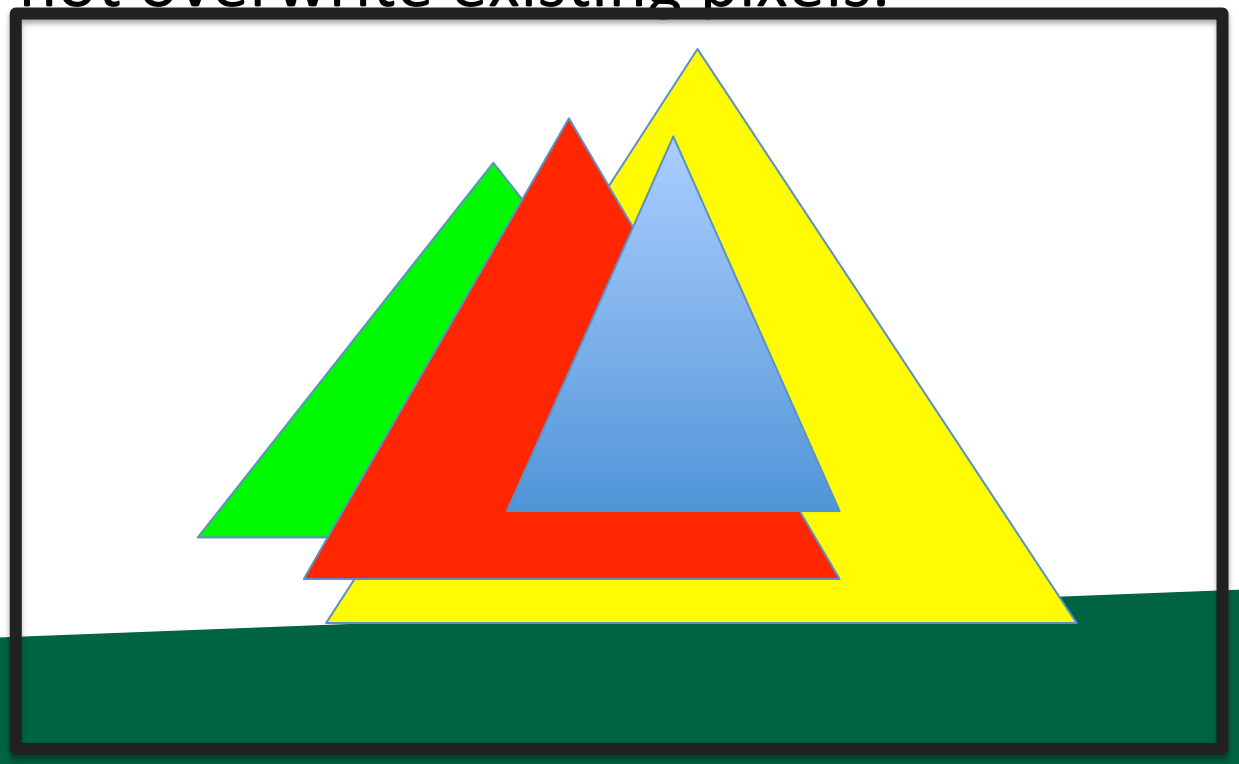
Idea #1

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

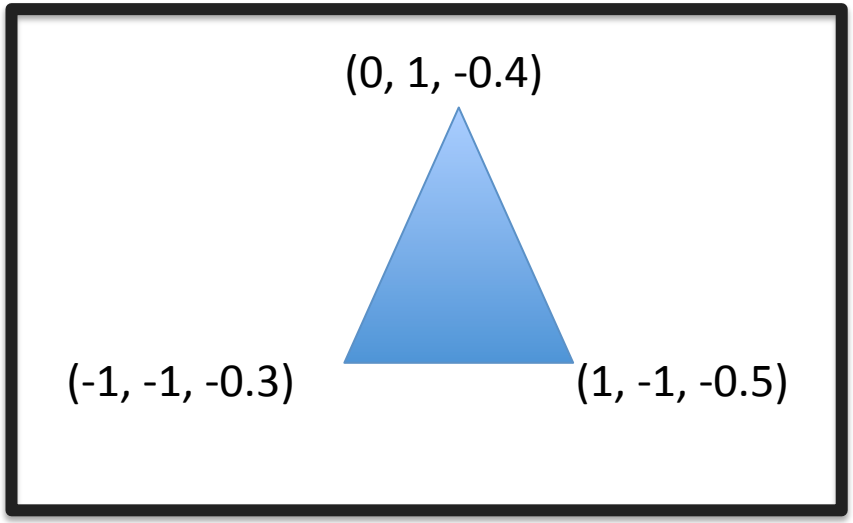


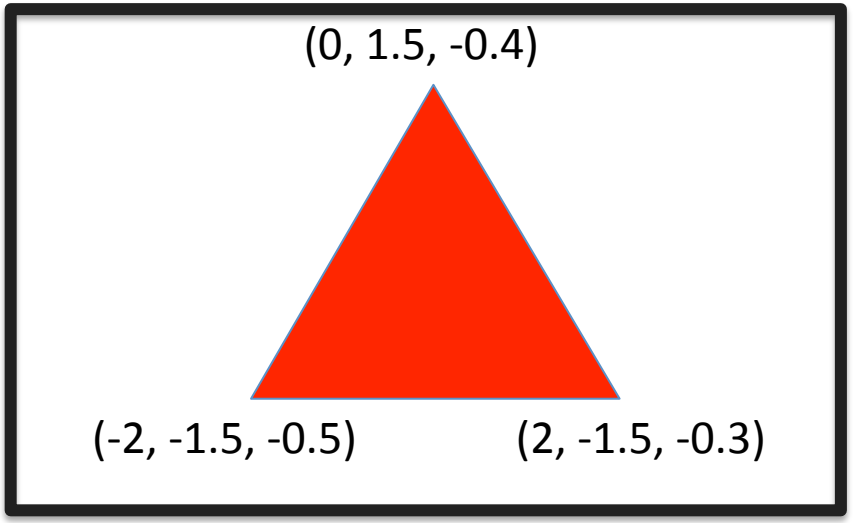
Idea #2

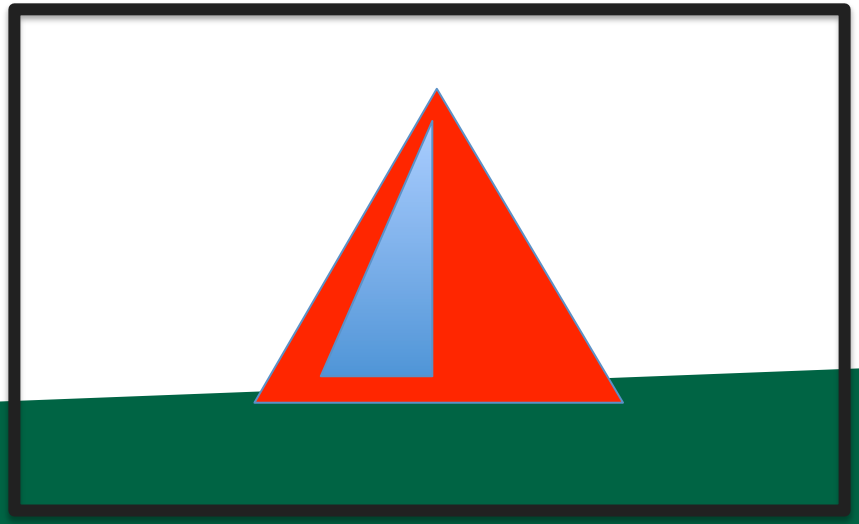
- Sort triangles “front to back” (based on Z)
- Render triangles in front to back order
  - Do not overwrite existing pixels.



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.

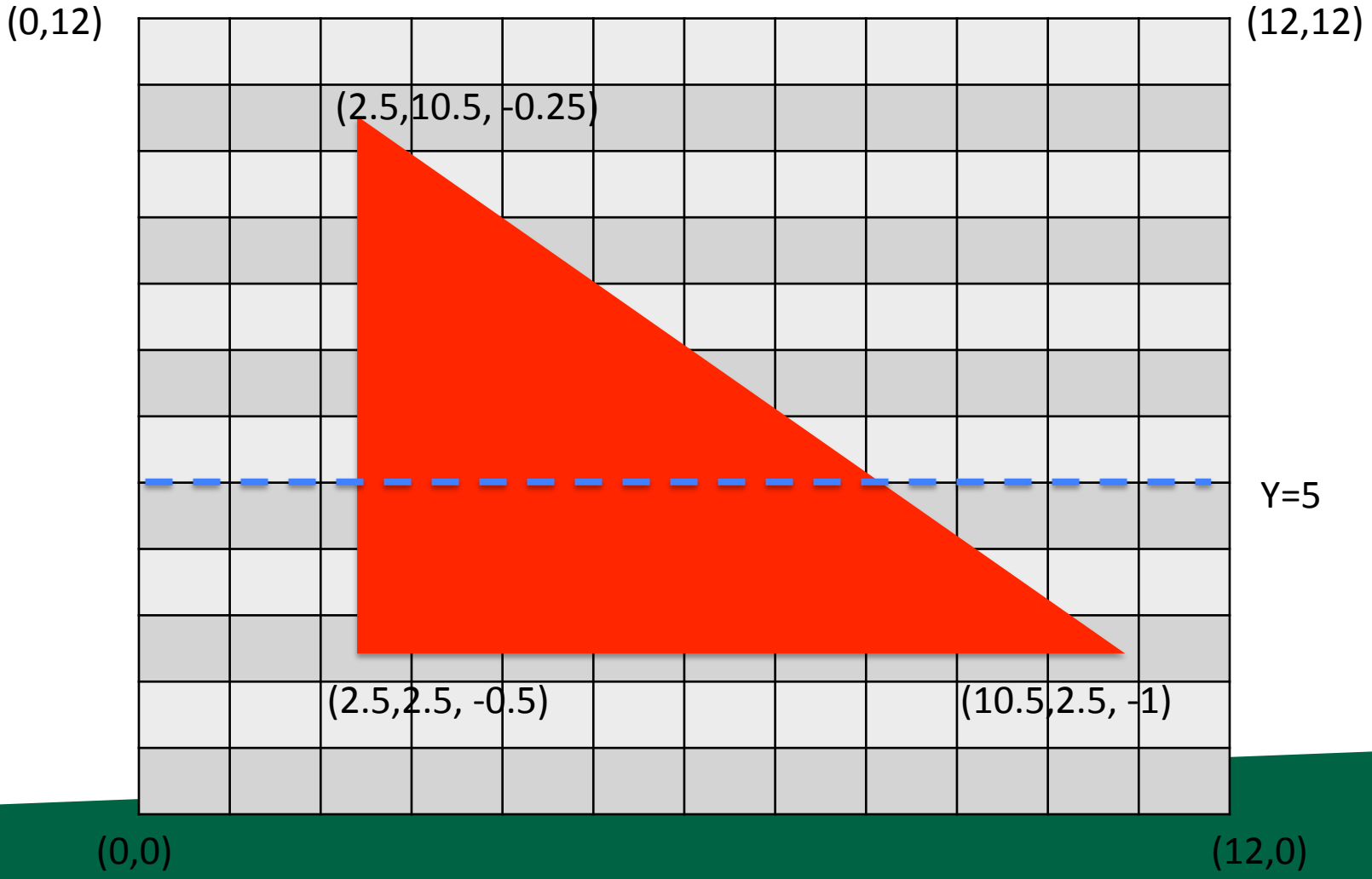
Scanline algorithm

- 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

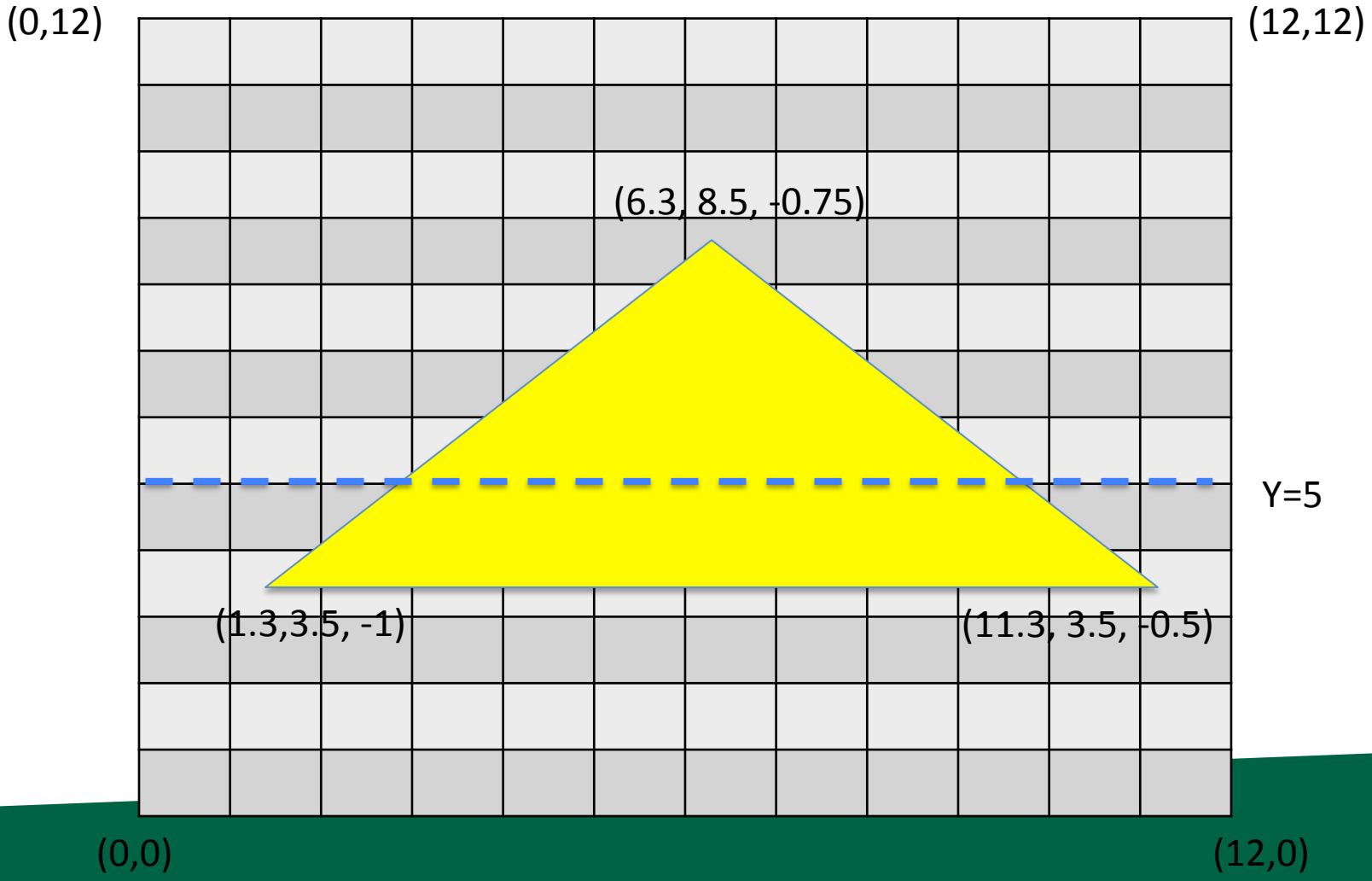
Scanline algorithm w/ Z-Buffer

- 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

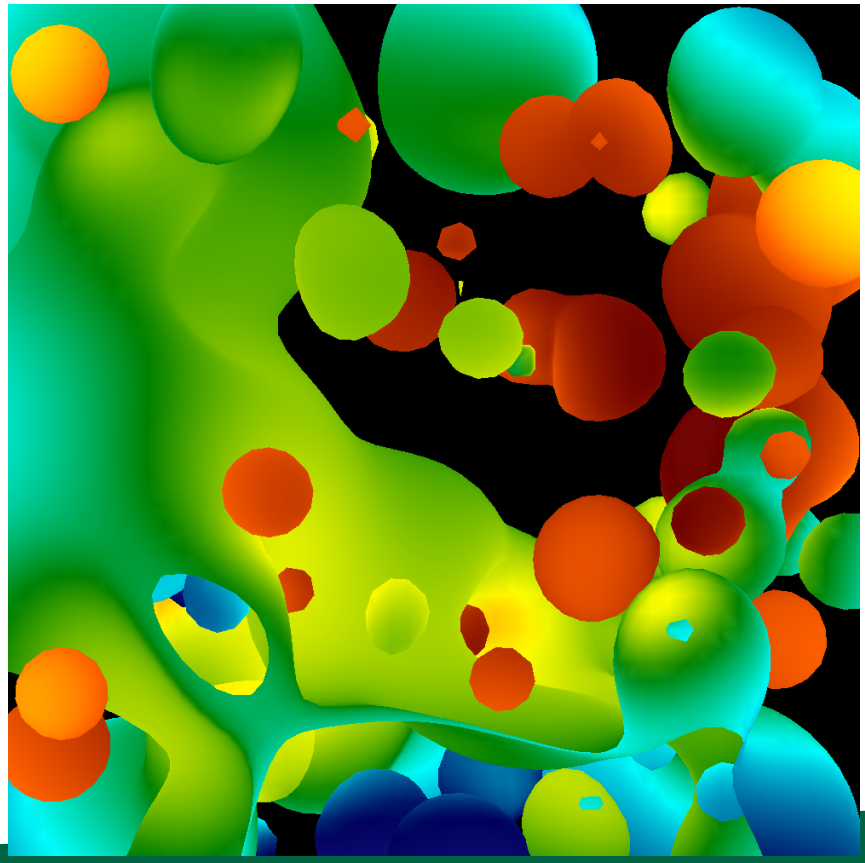


Interpolation and Triangles

- 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.

Project #1D (5%), Due Thurs Jan 31st

- Goal: interpolation of color and zbuffer
- Extend your project1C code
- File proj1d_geometry.vtk available on web (1.4MB)
- File “reader1d.cxx” has code to read triangles from file.
- No Cmake, project1d.cxx



Color is now floating-point

- We will be interpolating colors, so please use floating point (0 → 1)
- Keep colors in floating point until you assign them to a pixel
- Fractional colors? → use ceil_441…
  - ceil_441(value*255)

Changes to data structures

```
class Triangle
{
  public:
     double X[3], Y[3], Z[3];
     double colors[3][3];
};
```

→ reader1d.cxx will not compile until you make these changes

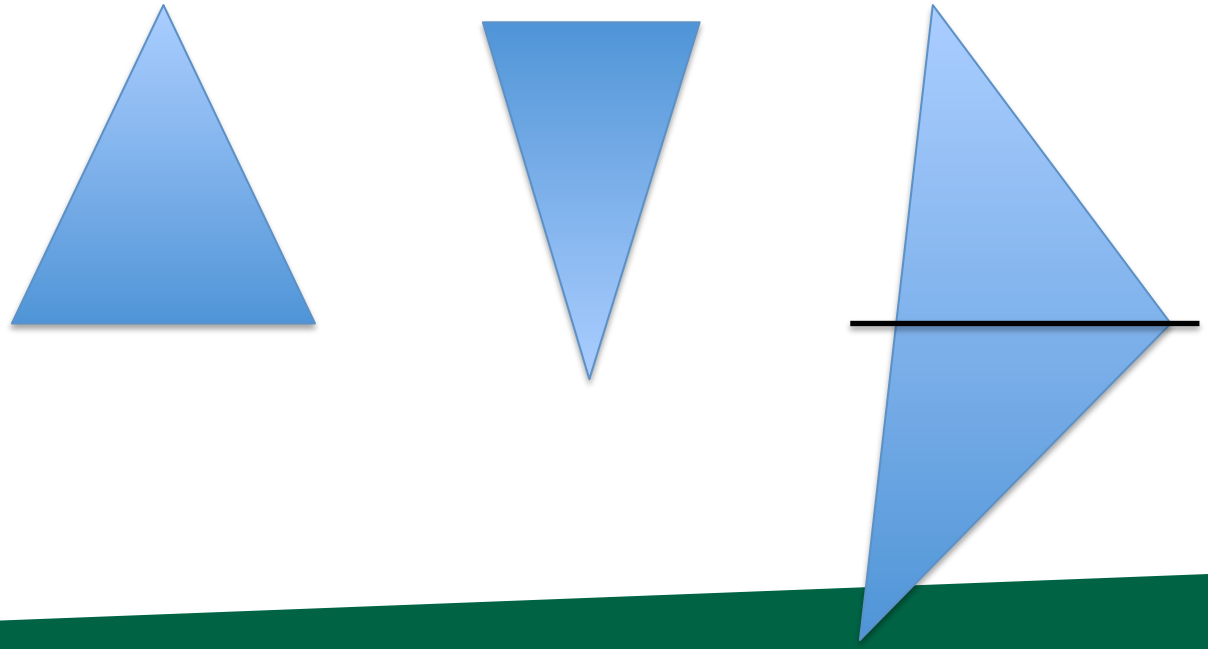
Project 1C

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
```

New debugging stuff…