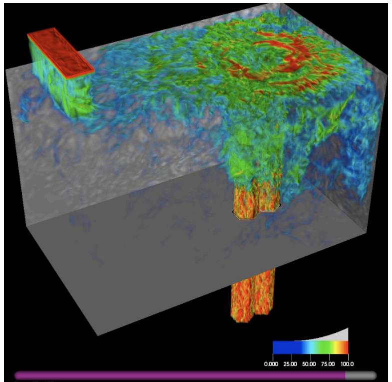
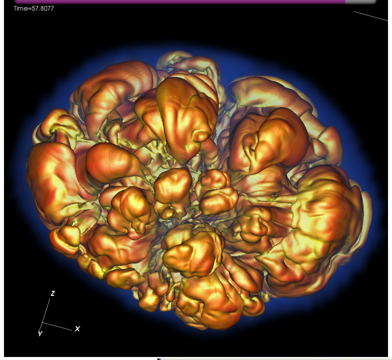
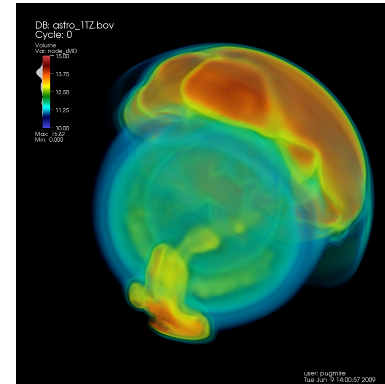
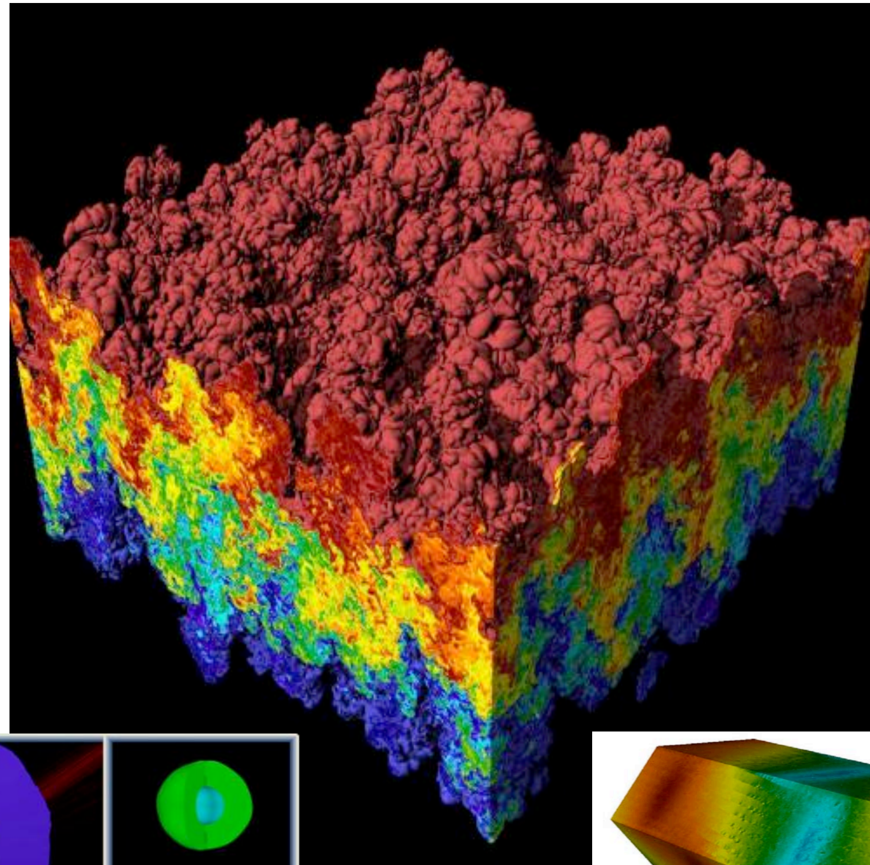
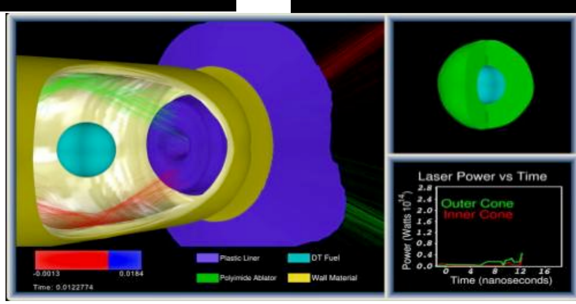
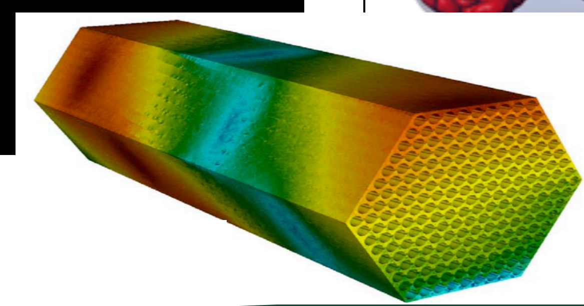
CIS 441/541: Intro to Computer Graphics
Lecture 3: Interpolation

April 6, 2021

Hank Childs, University of Oregon

Class Thursday

- Starts at 9am
- 9am-930am: Q&A / group OH on topics related to project 1, graphics
- 930am-940am: quiz
  - You must be present for these 10 minutes to take the quiz
  - If you cannot be present, you must (1) contact me by 12noon on Weds or (2) be in an emergency situation
  - Still determining how to make custom quizzes – know your UO ID
- This “lecture” will not be recorded

Let's do a practice quiz

UO ID	Column A	Column B
951994649	268	449
951907704	64	915
951790388	758	3
951062645	861	584
951166497	137	742
951648310	570	98
951239287	638	14
951322463	710	789
951193827	368	188
951075481	420	802
951260966	545	343
951448235	611	659
951422849	320	636
951219327	861	997
951774503	152	128
951432579	605	857
951709896	955	535
951129767	101	906
951622873	463	288
951858116	560	863
951807629	678	118
951292153	477	651
951652318	678	309
951907604	891	8
951062452	24	194
951302405	653	421
951727929	150	538
951449684	830	727
951958265	143	843
951329660	401	750
951645337	837	500
951650461	845	363
951407127	63	549
951749776	934	424
951690768	529	776
951339608	685	371
951035559	338	406
951848919	81	52
951035905	413	451
951957181	960	487
951190320	473	589

Quiz #1 (THIS IS A FAKE QUIZ FOR US TO PRACTICE WITH)

Question 1: enter column A for your UO ID
Question 2: enter column B for your UO ID
Question 3: what is A+B?
Question 4: what is A-B?

Note: all of these UO IDs are fake

Virtual Delivery is Changing How This Class is Delivered

- Despite my best efforts, I have done a lot of repetition in previous offerings
  - In this setting, repeating myself seems like a waste of your time
    - Borderline disrespectful
- We will figure this issue out as we go
  - Positive aspect: I was already considering using Thursdays in non-lecture format

May Have Too Much Lecture Today

- We will get as far as we can

Week 2 Office Hours

Published | Edit

How to access Office Hours

Hank Childs

Apr 4 at 2:02pm

All Sections

Hi Everyone,

We currently have an asymmetry for accessing Hank and Abhishek's Office Hours.

As of now, Abhishek's are always at: COVERED UP (THIS IS POSTED ONLINE)

And Hank's are accessible via the Zoom Meetings area in Canvas.

Let's chat on Tuesday about the most standard way to do this.

Finally, here is the OH schedule again:

Monday (Abhishek): 10am-11am
Tuesday (Abhishek): 945am-1045am
Wednesday (Hank): 230pm-330pm
Thursday (Abhishek): 945am-1045am

Best,
Hank

~~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)

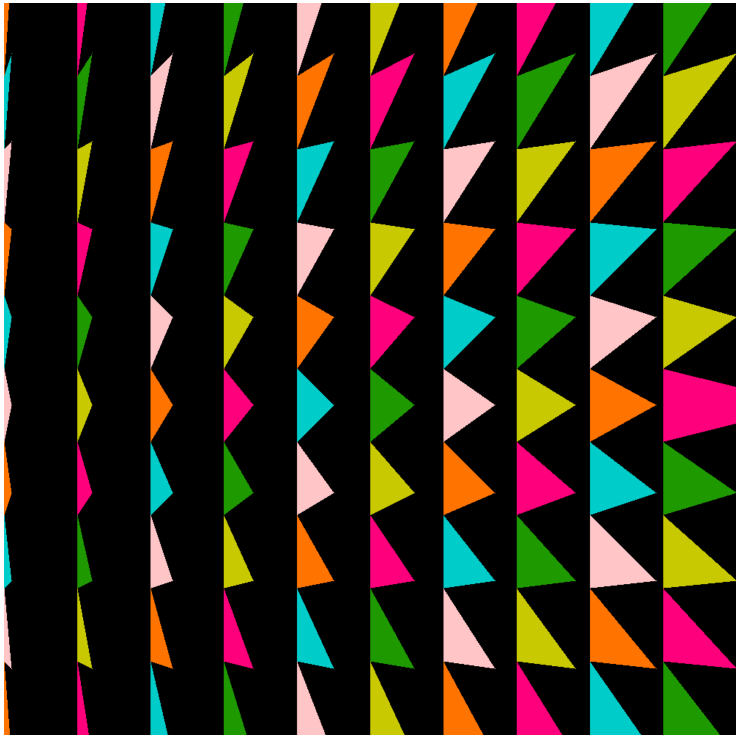
NEW Slide

- 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

These are REPEAT slides I traditionally have repeated this lecture (although quickly)

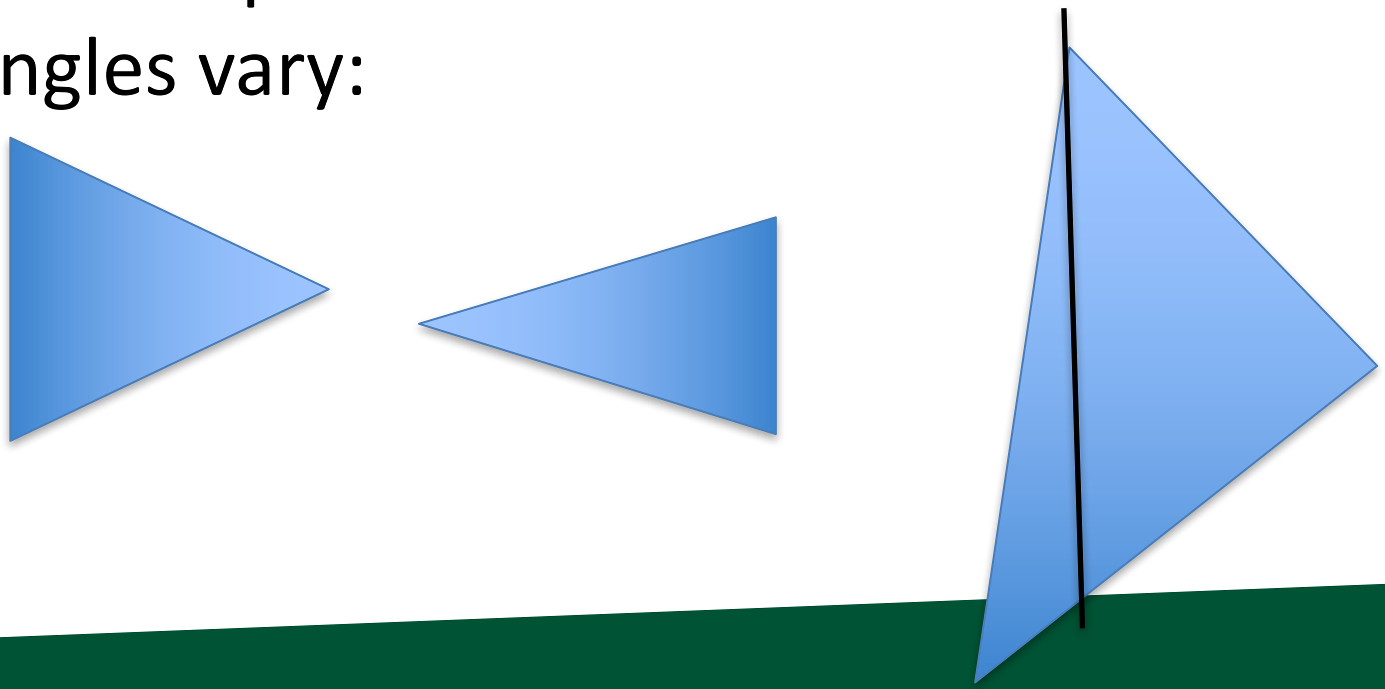
Project #1B (due tomorrow): Questions?

- 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%



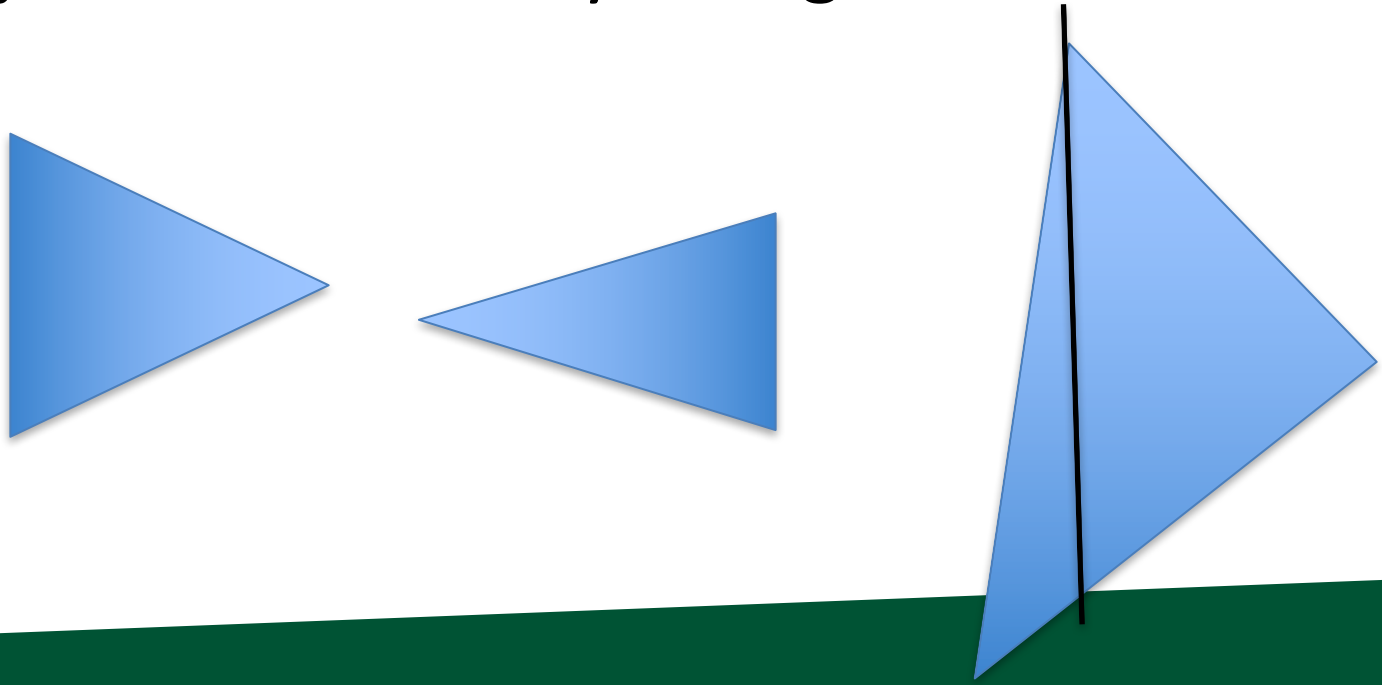
Arbitrary Triangles

- 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 right" triangles
- Project #1C: arbitrary triangles



Arbitrary Triangles

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

Project #1C (6%), Due (April 14th)

- 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
- POSTED SOON

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:

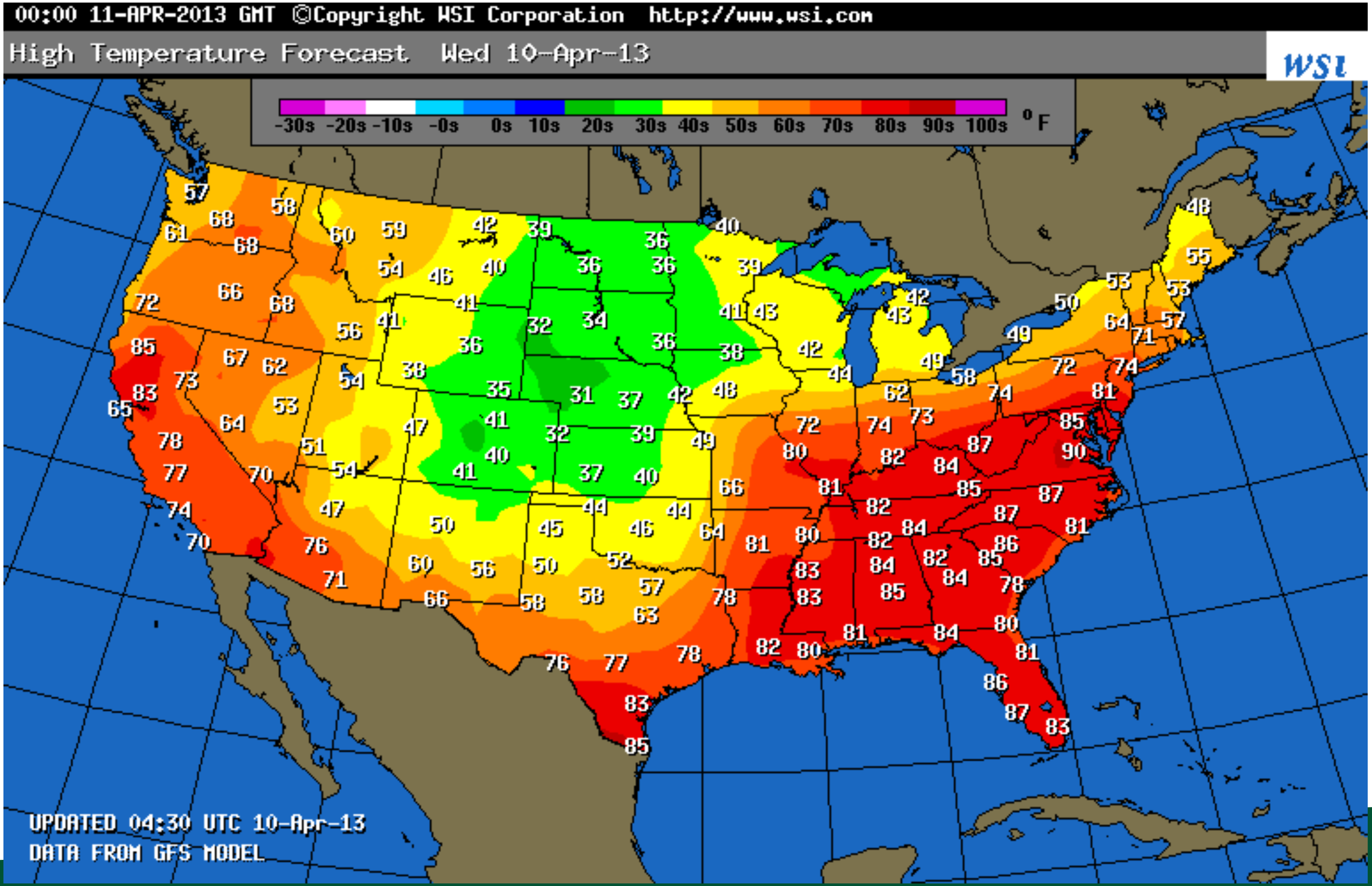
How to deposit triangle colors onto an image?

Still don't know how to:

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

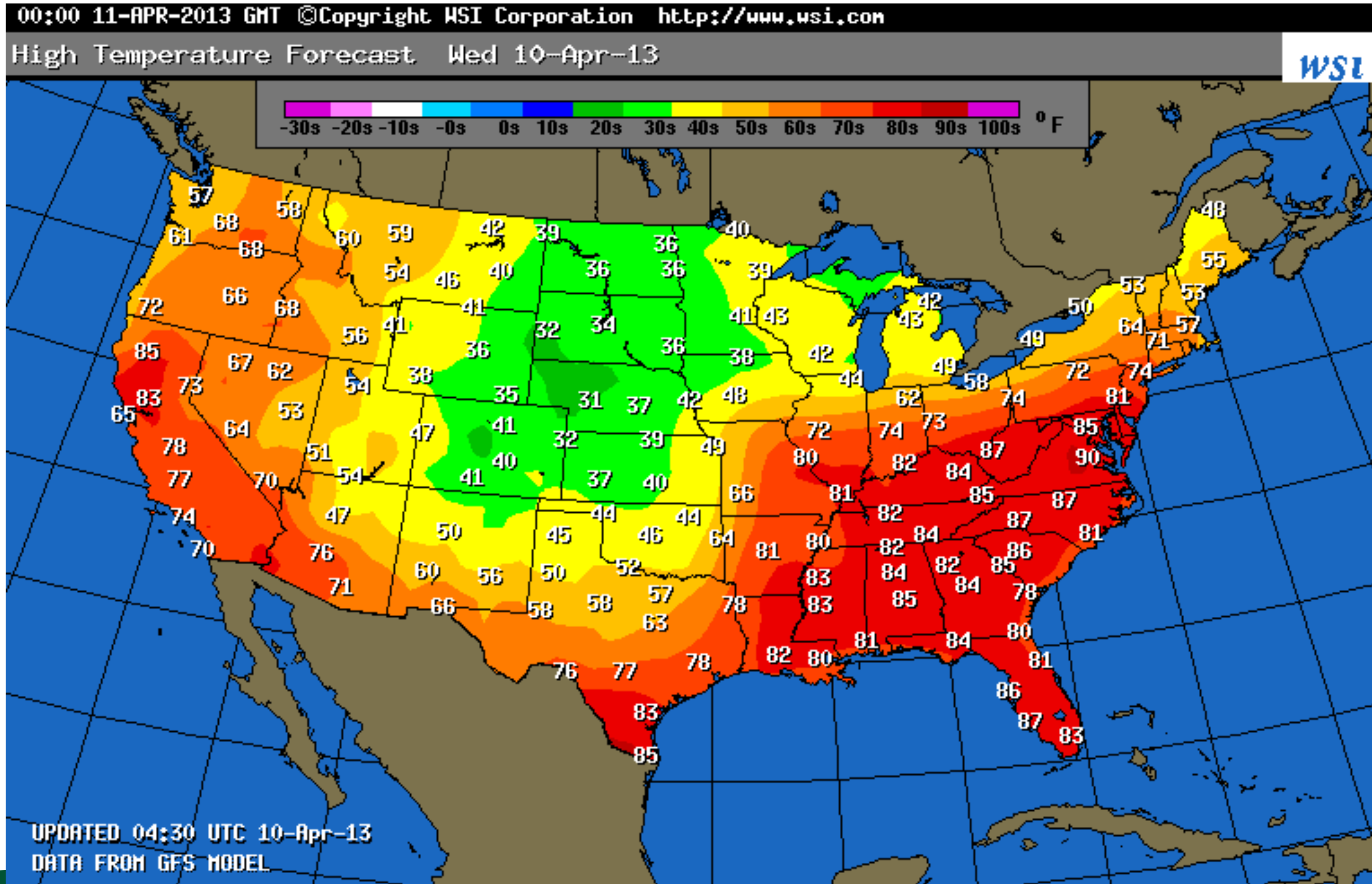
Today's lecture will go over the key operation to do these two.

What is a field?



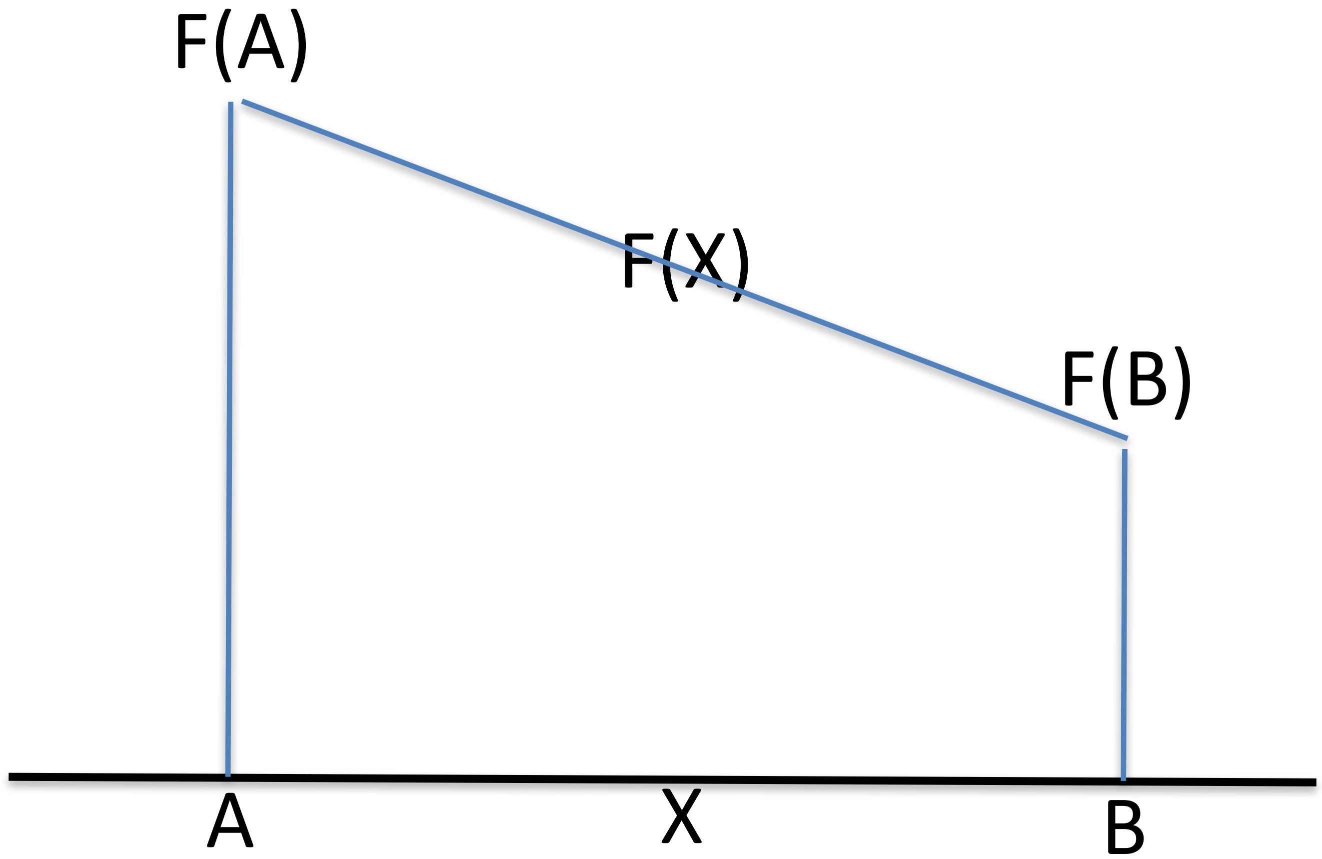
Example field (2D): temperature over the United States

How much data is needed to make this picture?



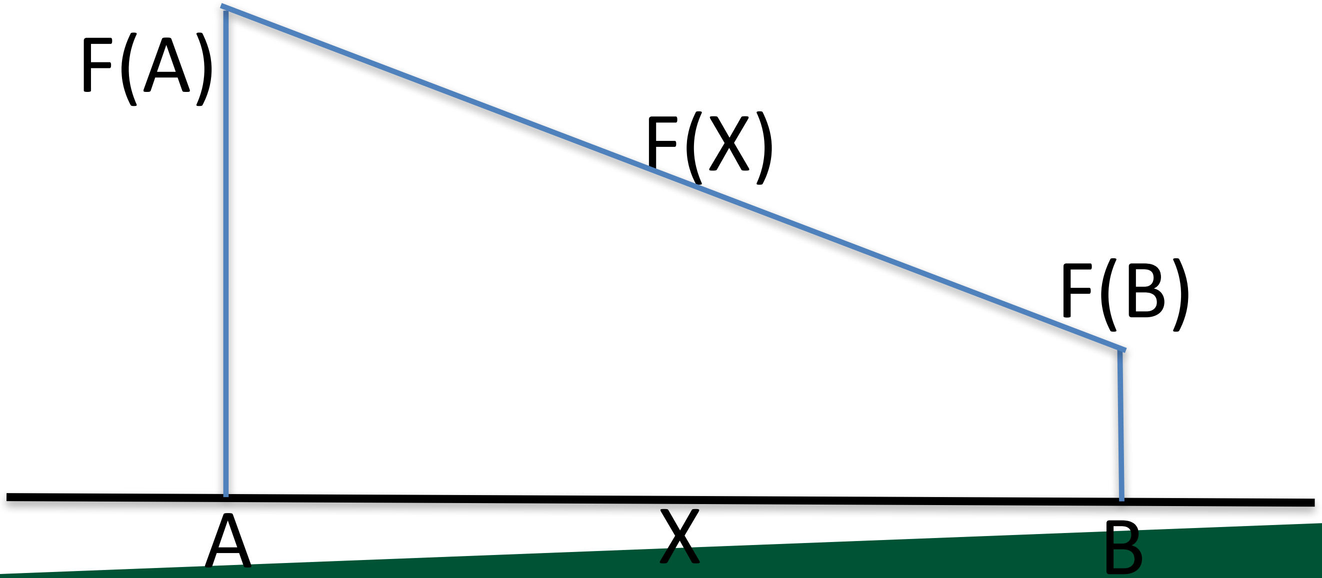
Example field (2D): temperature over the United States

Linear Interpolation for Scalar Field F



Linear Interpolation 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)

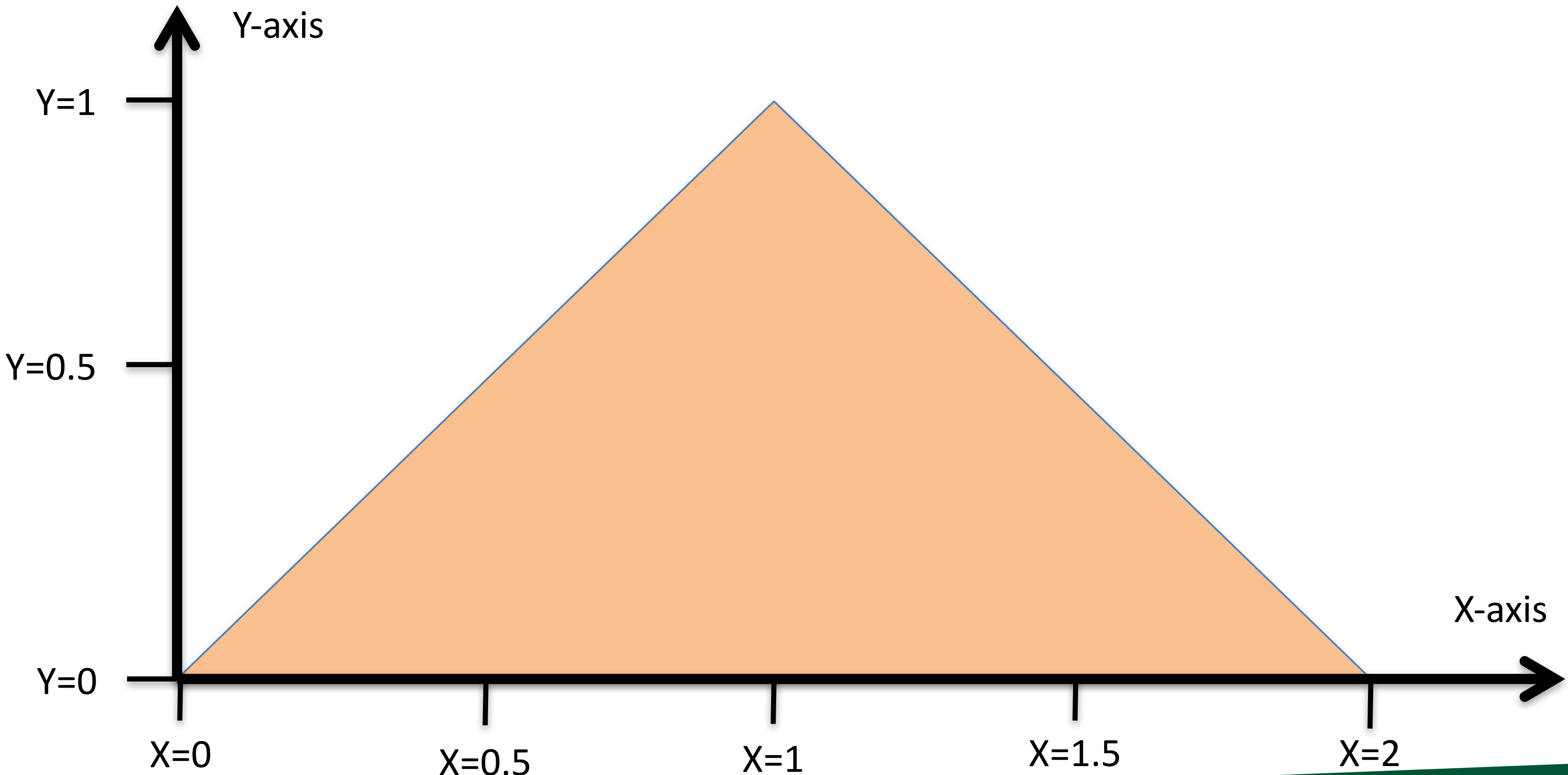


Quiz Time

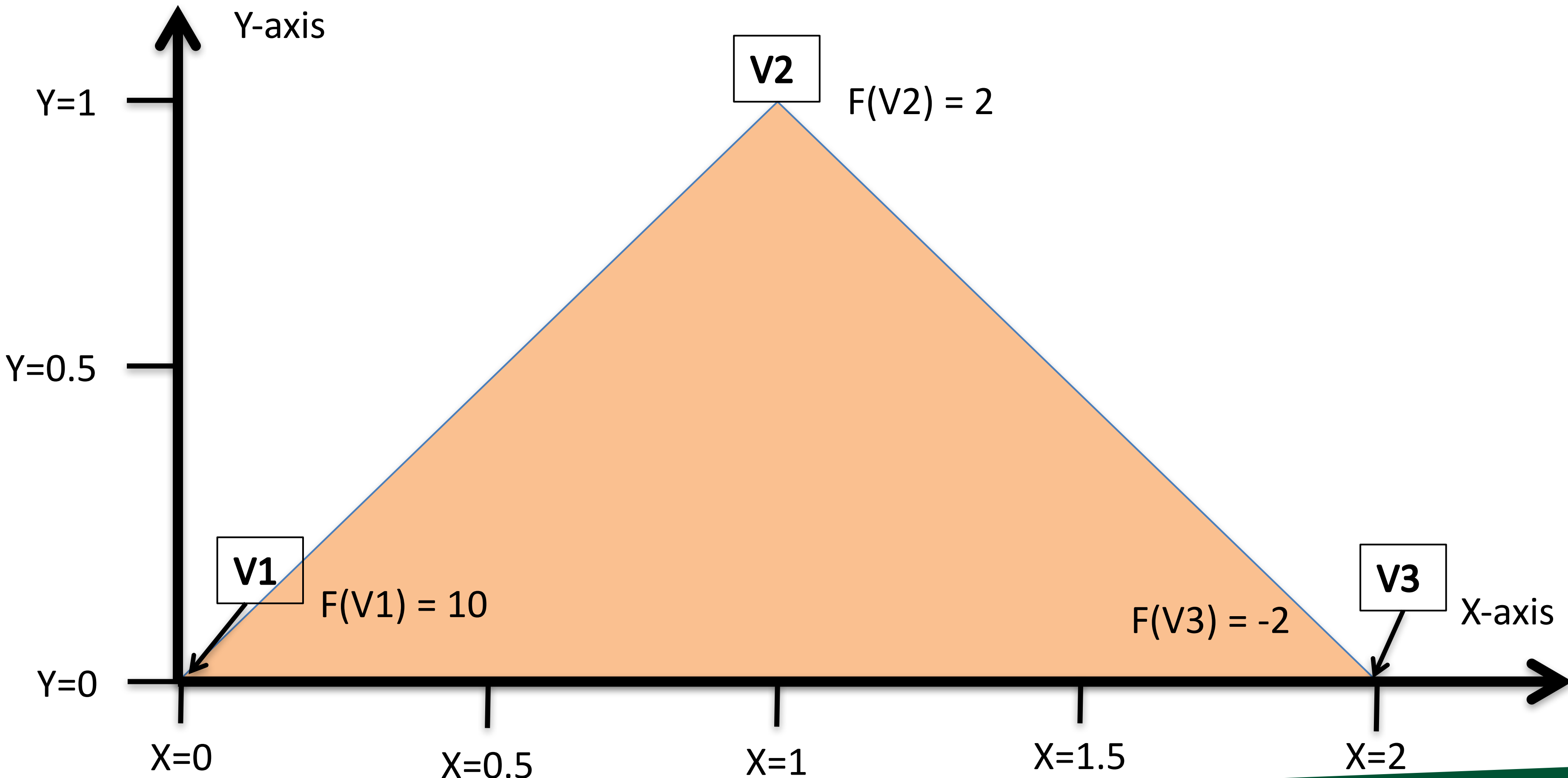
- F(3) = 5, F(6) = 11
- What is F(4)? = 5 + (4-3)/(6-3)*(11-5) = 7

- 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)

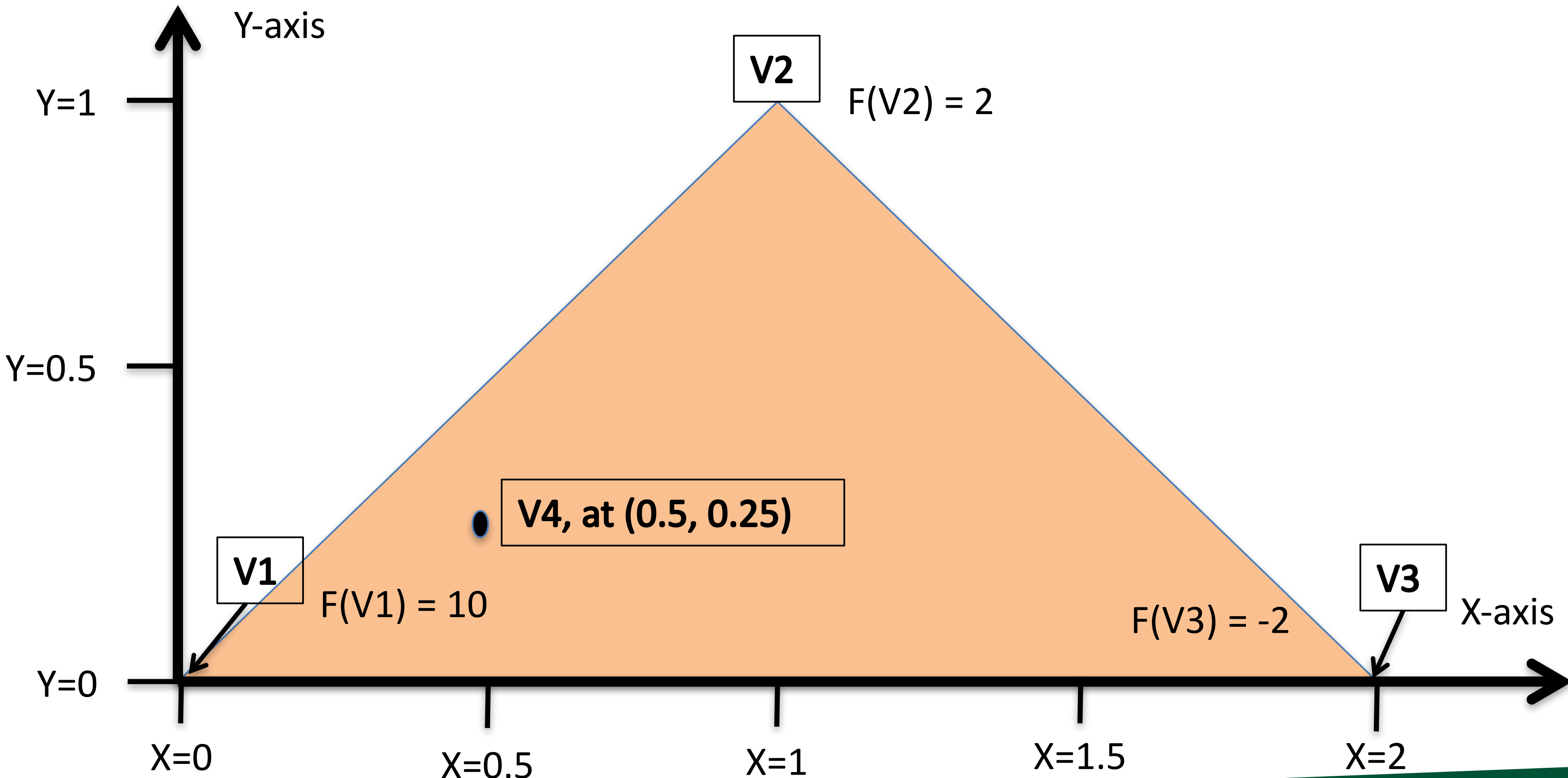
Consider a single scalar field defined on a triangle.



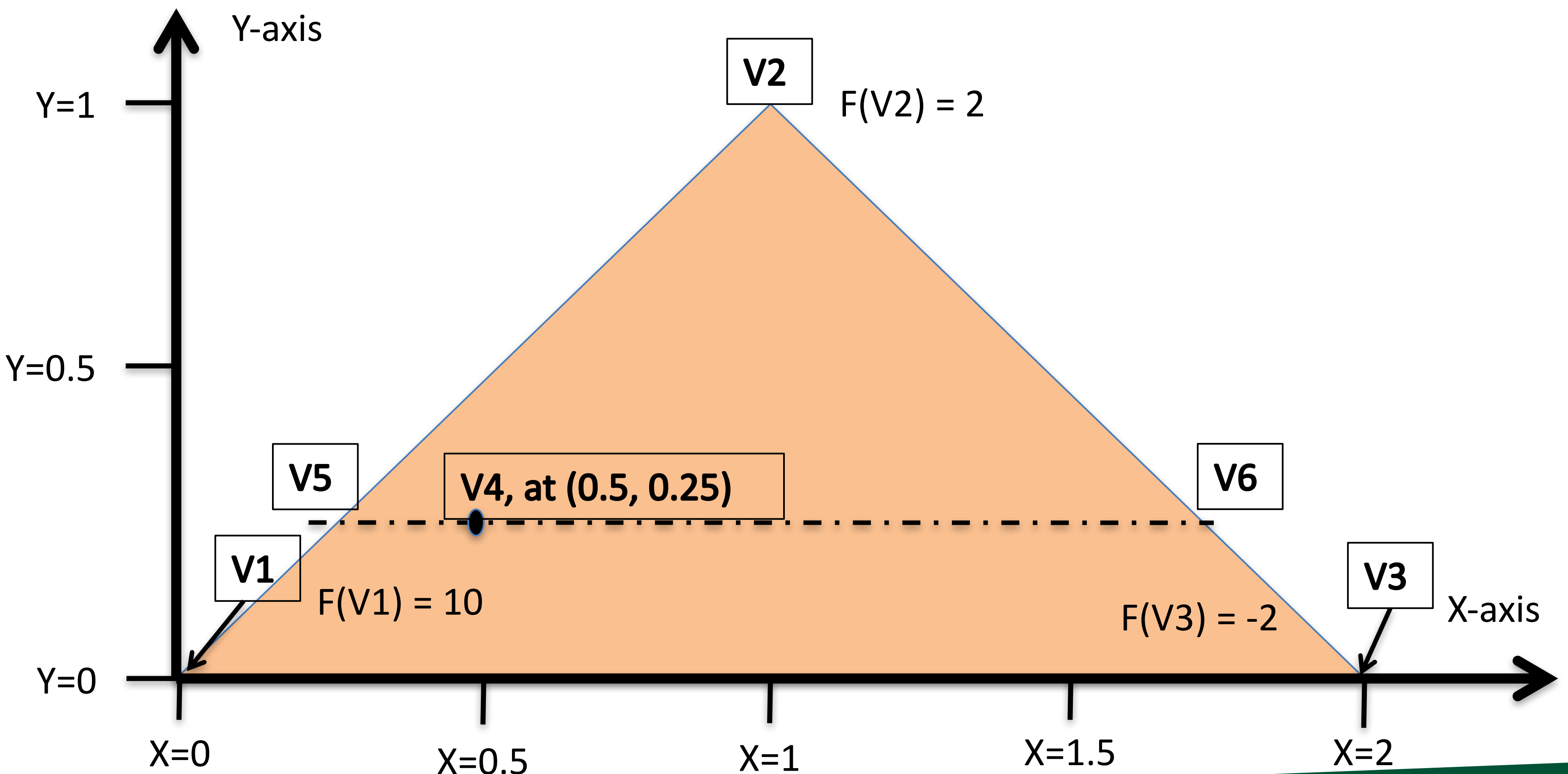
Consider a single scalar field defined on a triangle.



What is F(V4)?


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

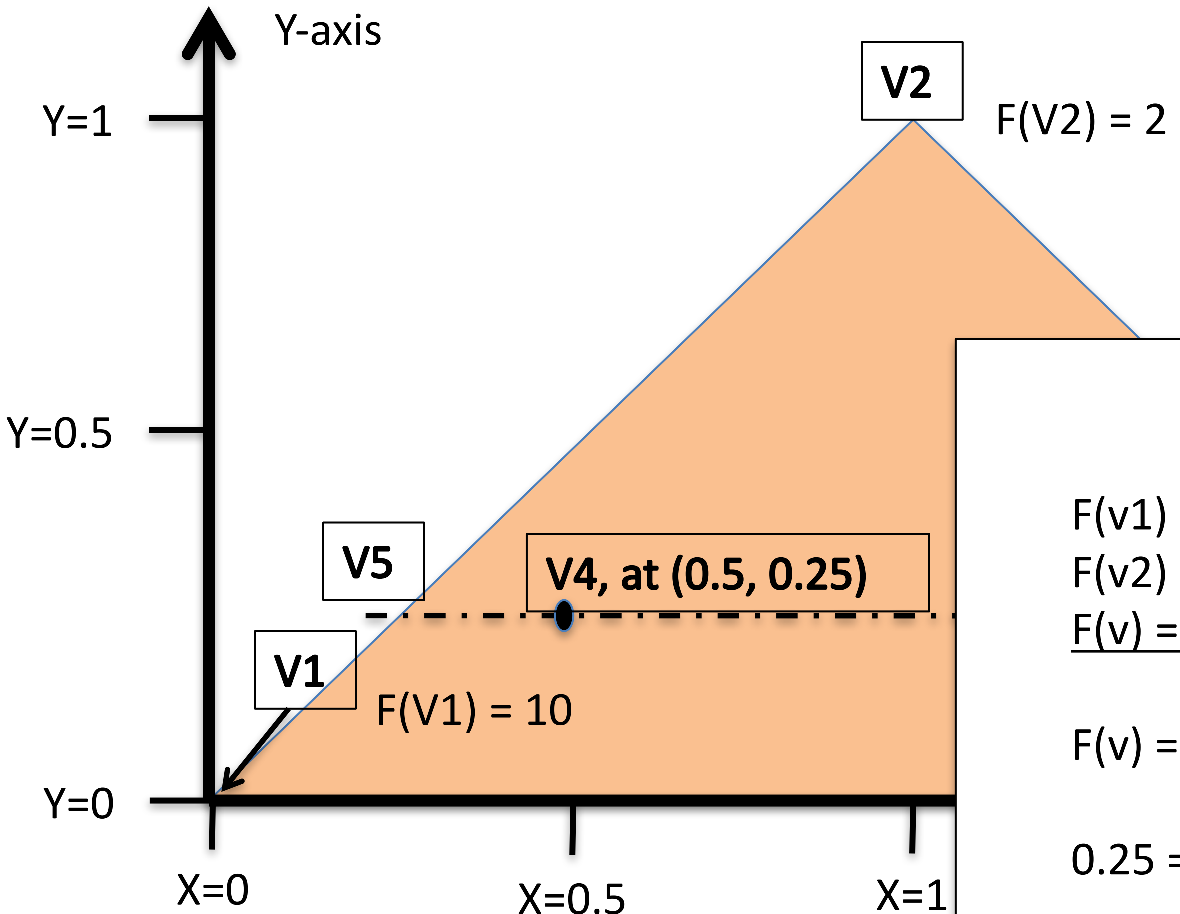
What is F(V4)?


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
F(V3) = -2
V3
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

- Note: when you implement this, you will be doing vertical scanlines, so doing it for X=0.5

What is the X-location of V5?



F(v1) = A → F(0) = 0

F(v2) = B → F(1) = 1

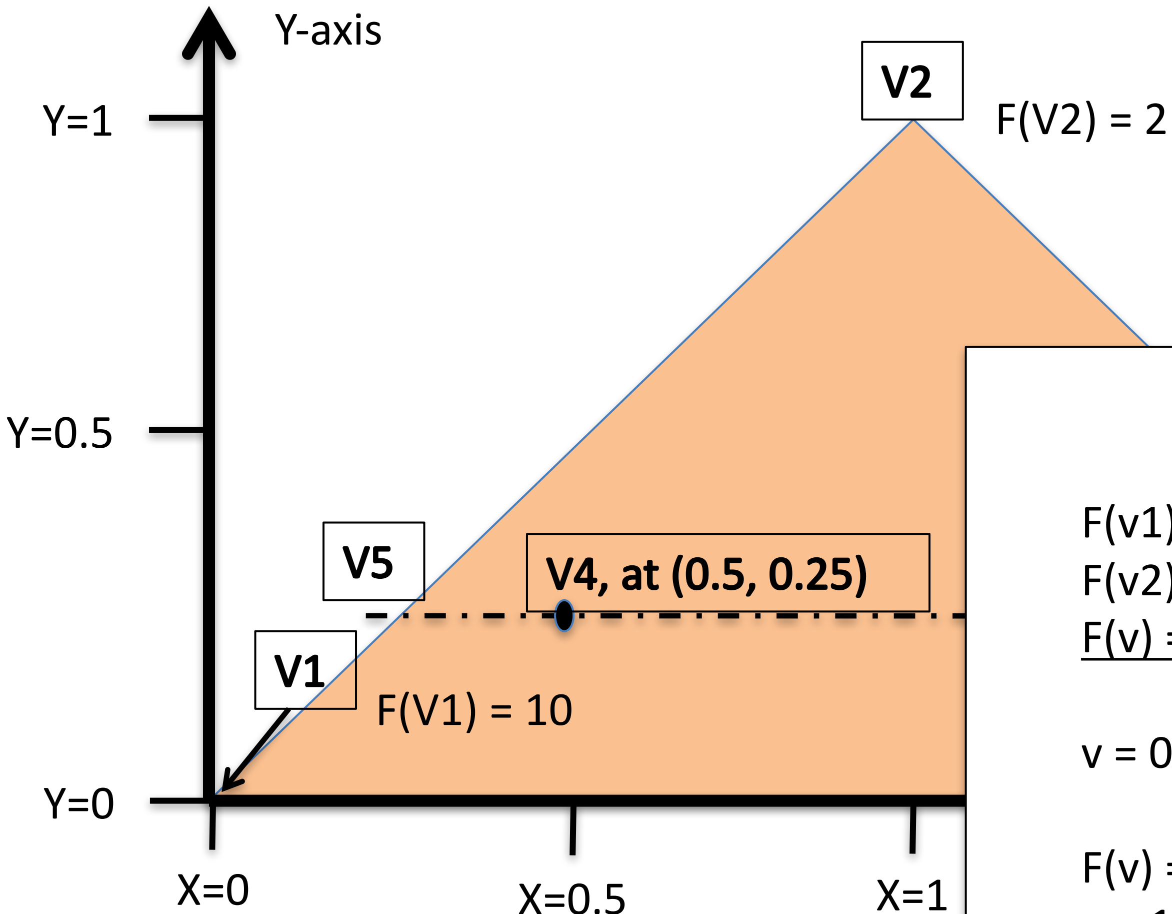
F(v) = A + ((v-v1)/(v2-v1))*(B-A):

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

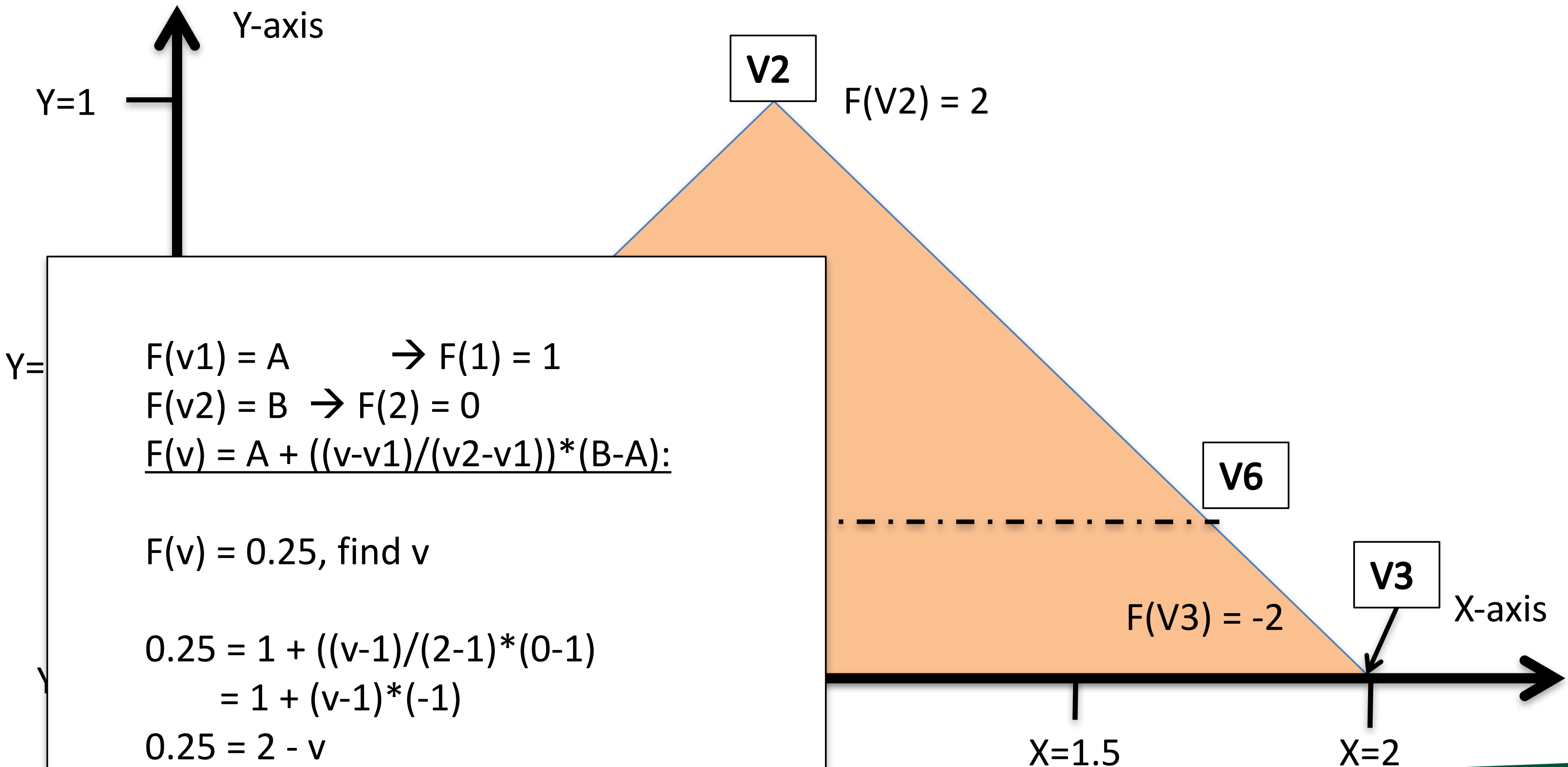
F(v2) = B → F(1) = 2

<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?

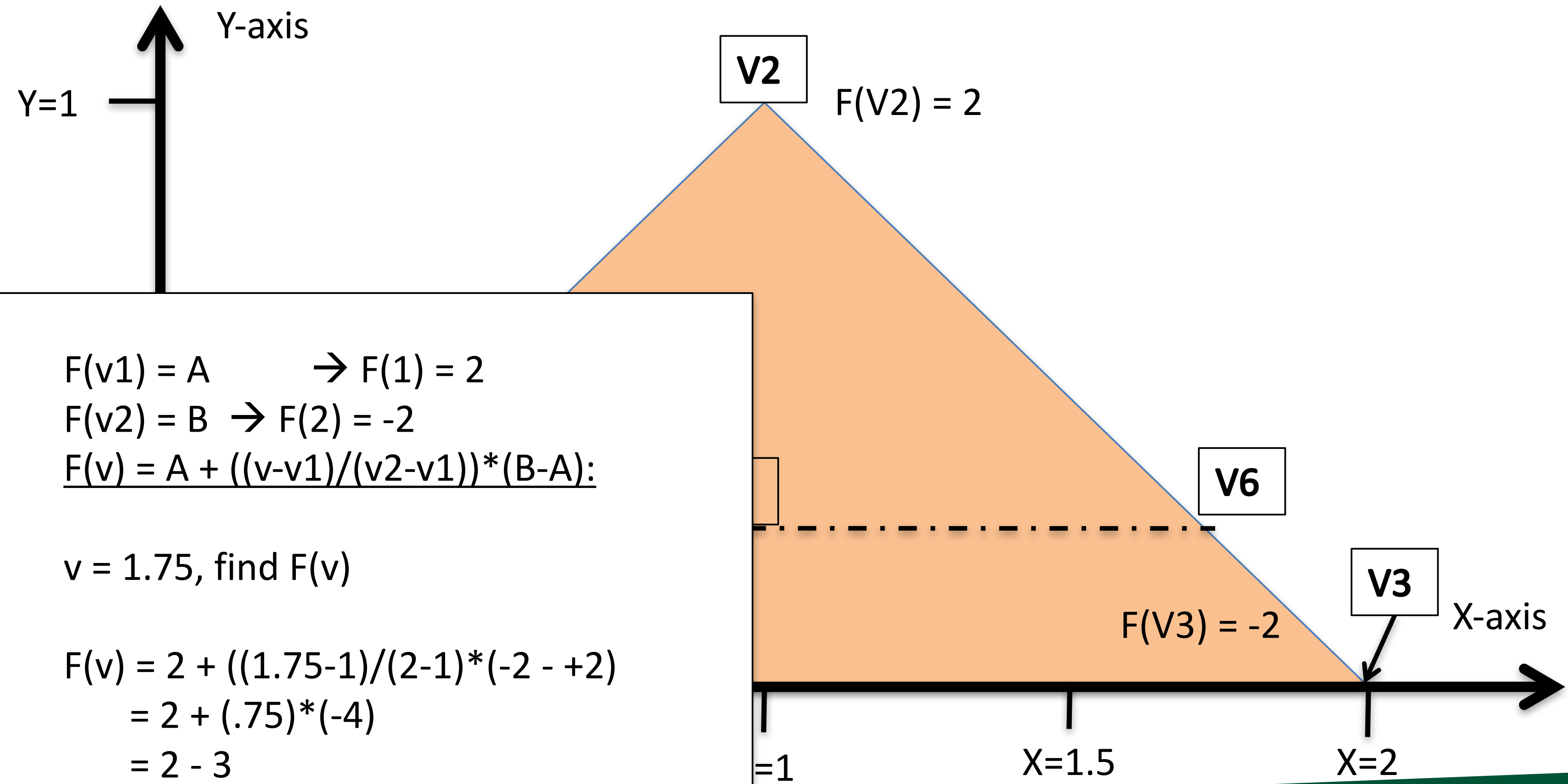


F(v1) = A → F(1) = 1
F(v2) = B → F(2) = 0
<u>F(v) = A + ((v-v1)/(v2-v1))*(B-A):</u>

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

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

v = 1.75, find F(v)

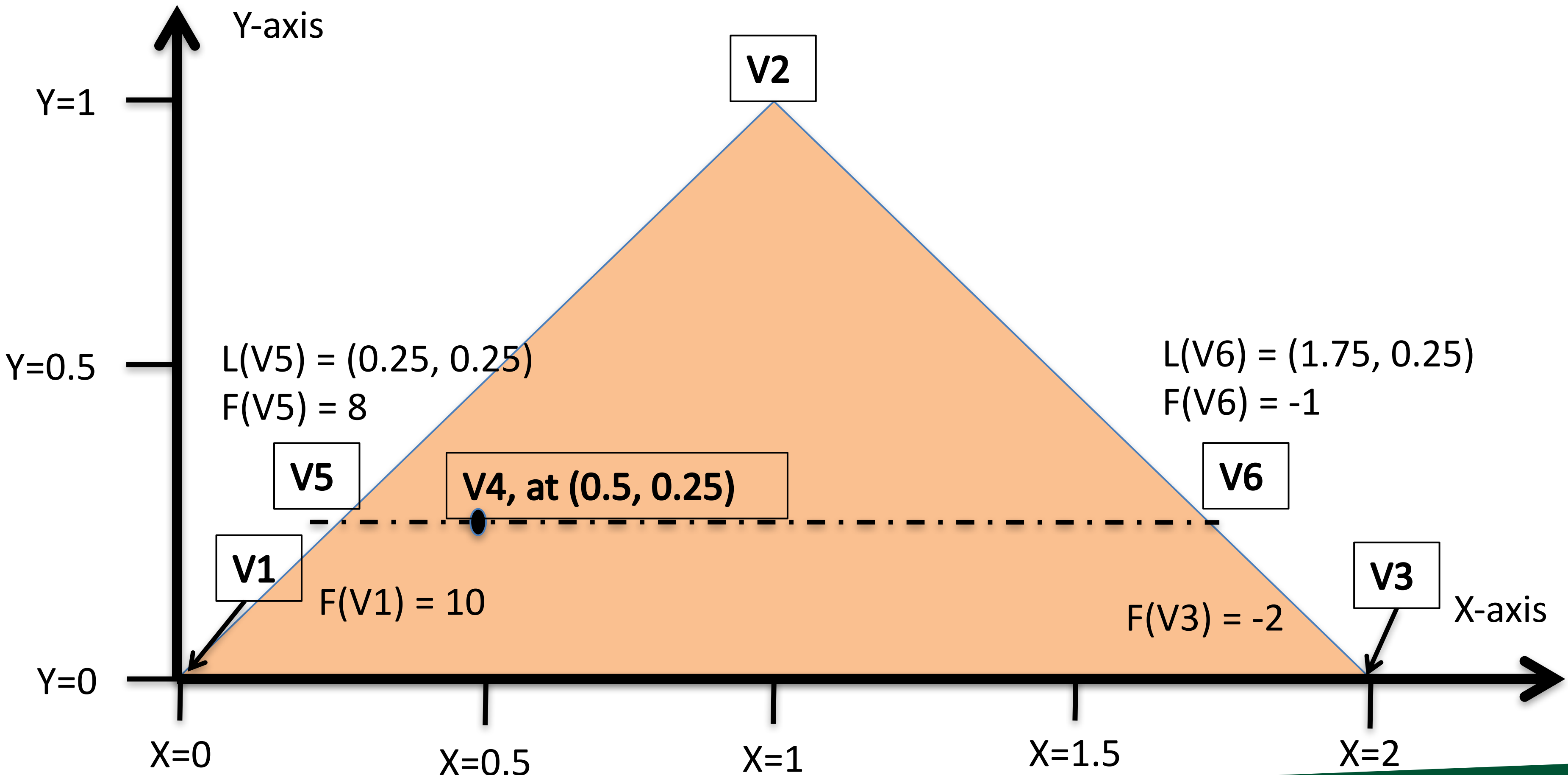
F(v) = 2 + ((1.75-1)/(2-1)*(-2 - +2)

= 2 + (.75)*(-4)

= 2 - 3

= -1

What is the F-value of V5?

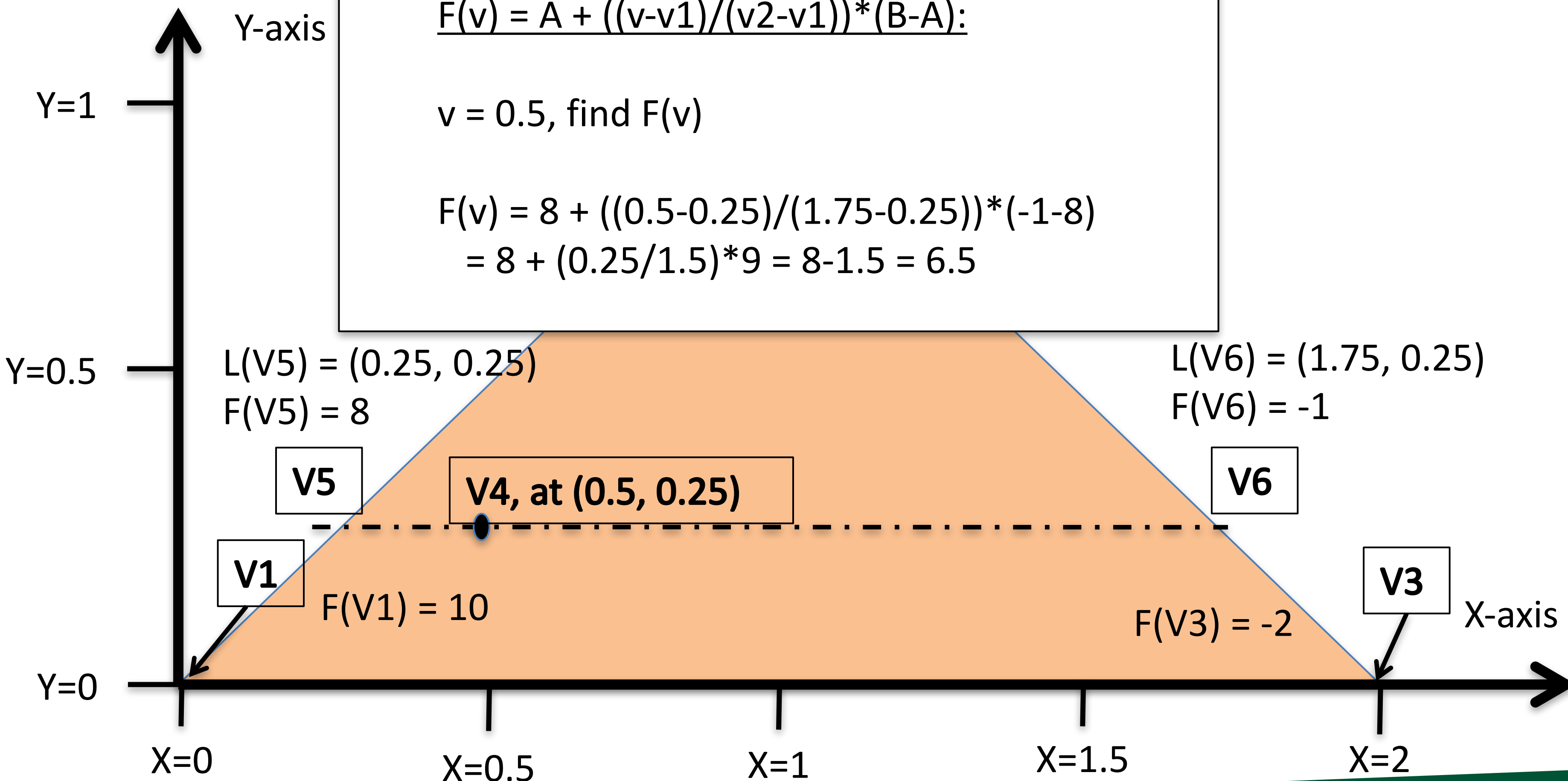


What is the F-value of V5?

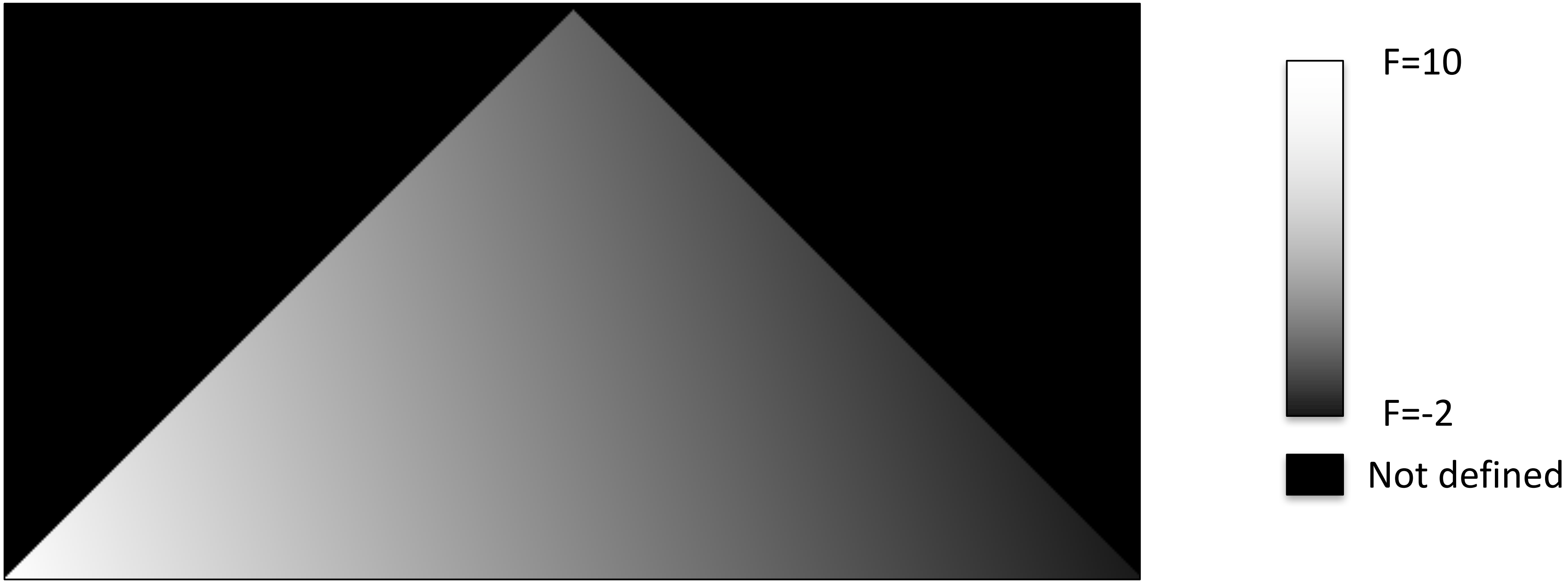
F(v1) = A → F(0.25) = 8
F(v2) = B → F(1.75) = -1
<u>F(v) = A + ((v-v1)/(v2-v1))*(B-A):</u>

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

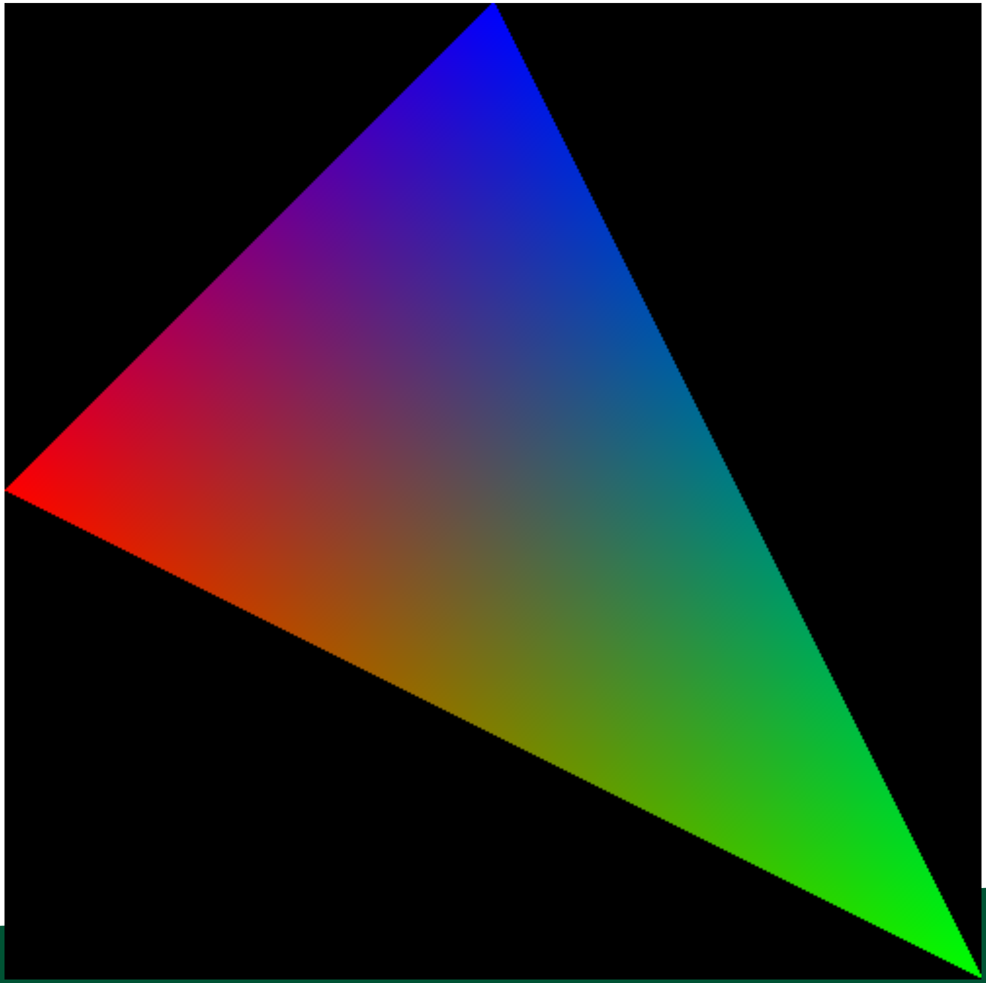


How do you think this picture was made?

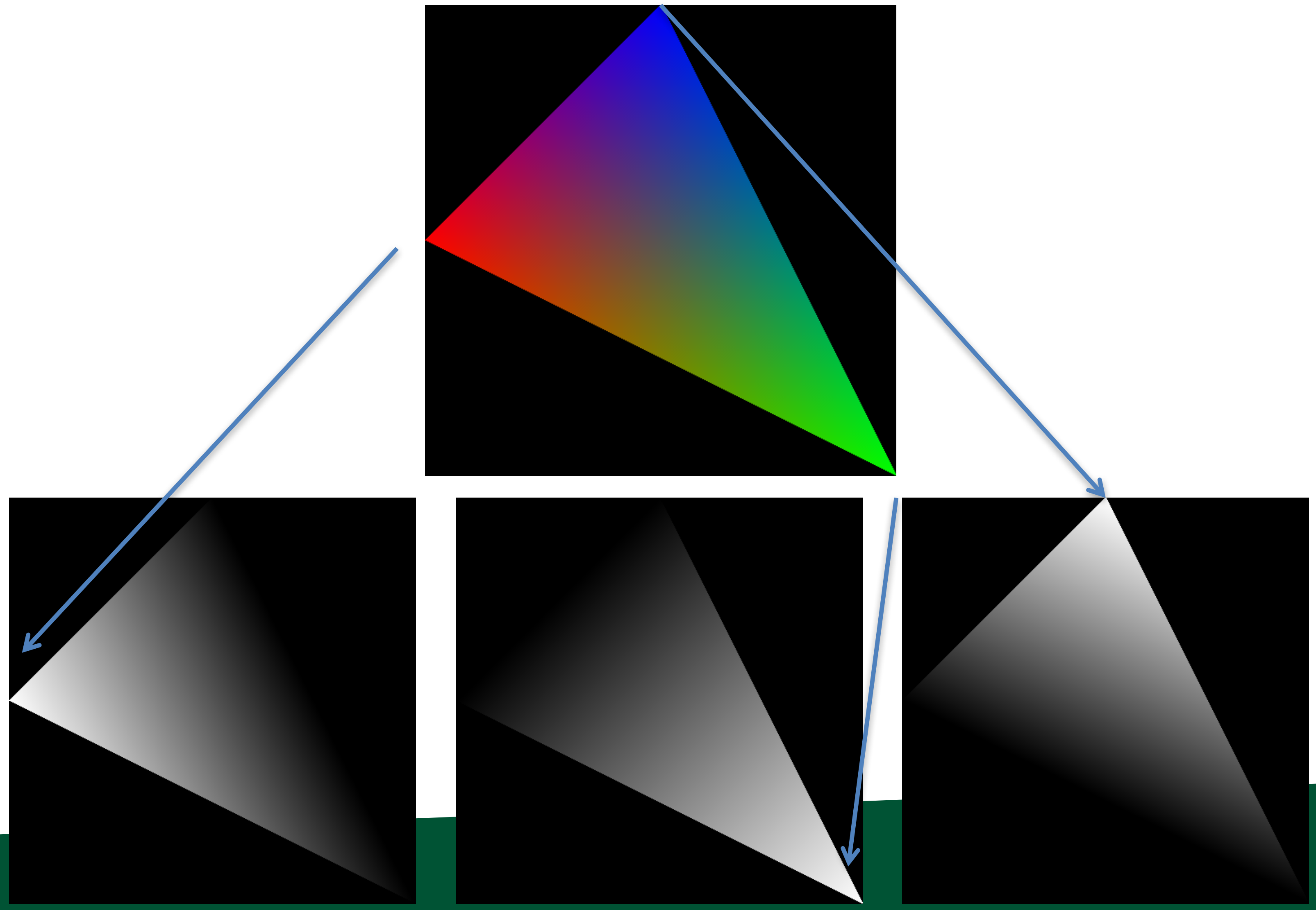
Now We Understand Interpolation
Let's Use It For Two New Ideas:
Color Interpolation
& Z-buffer Interpolation

Colors

What about triangles that have more than one color?



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



Red channel　　Green channel　　Blue channel

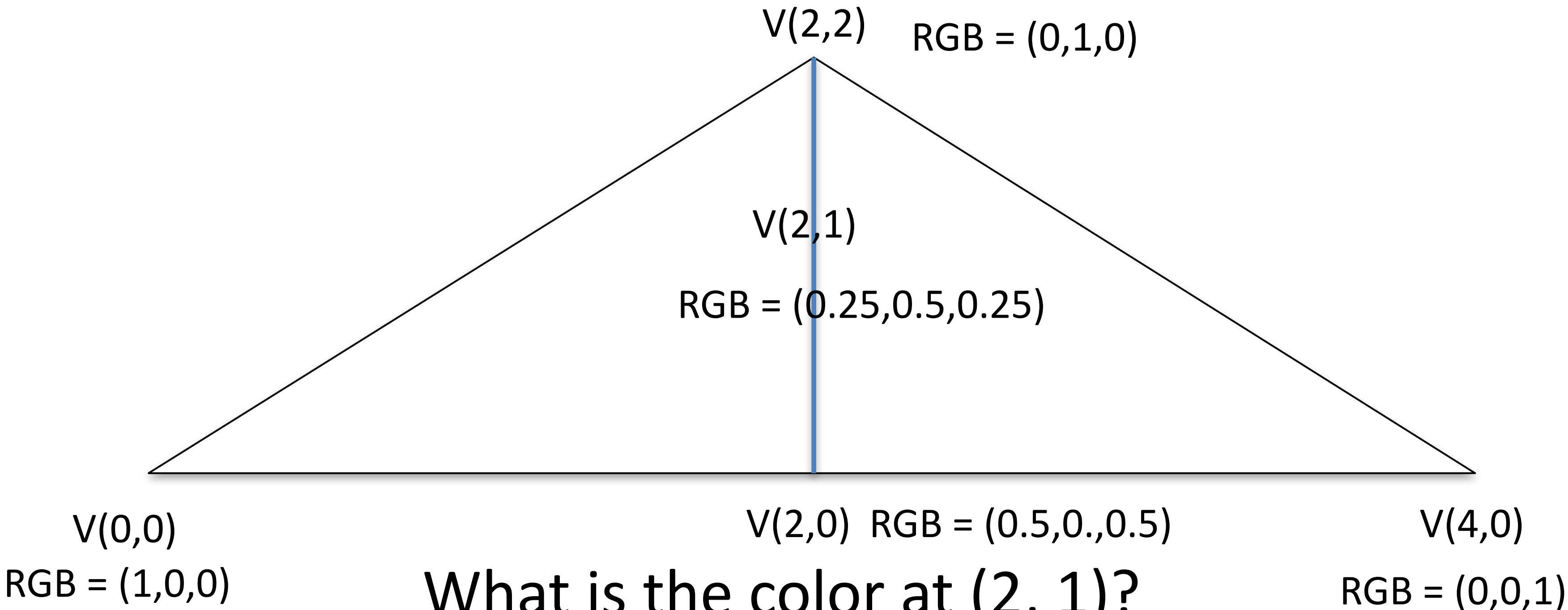
Scanline algorithm for one triangle

- 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 w/ Color

- 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
    - Calculate Color(bottomEnd) and Color(topEnd) using interpolation from triangle vertices
  - For r in [ceiling(bottomEnd) → floor(topEnd)] ; do
    - Calculate Color(r, c) using Color(bottomEnd) and Color(topEnd)
    - 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!

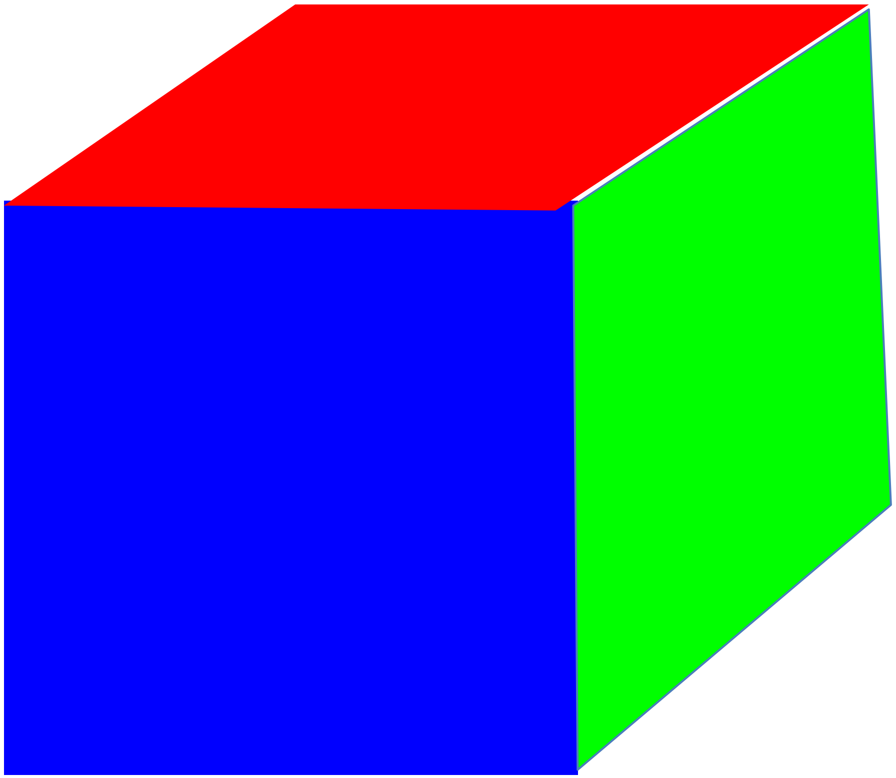
Important

- 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….

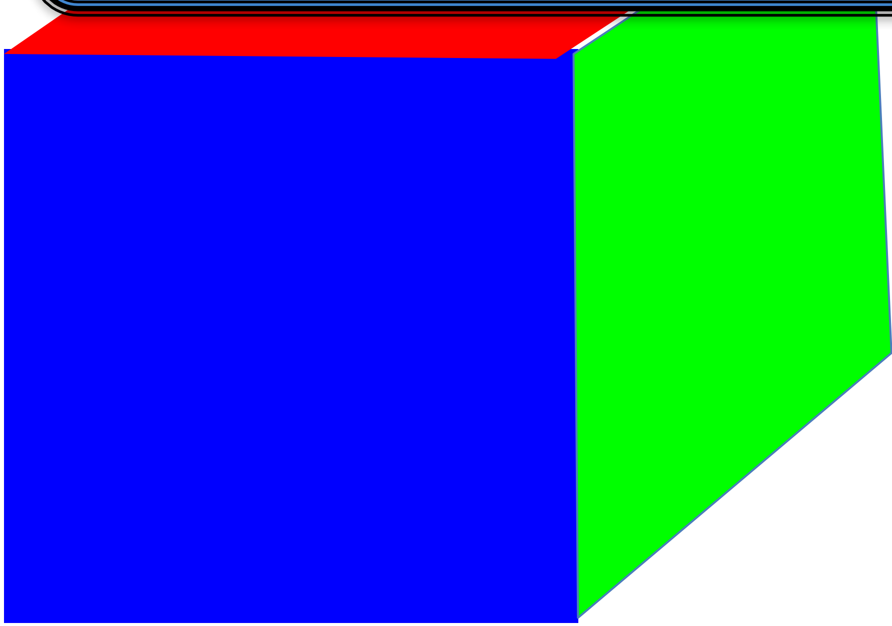


Face	Color
Front	Blue
Right	Green
Top	Red
Back	Yellow
Left	Purple
Bottom	Cyan

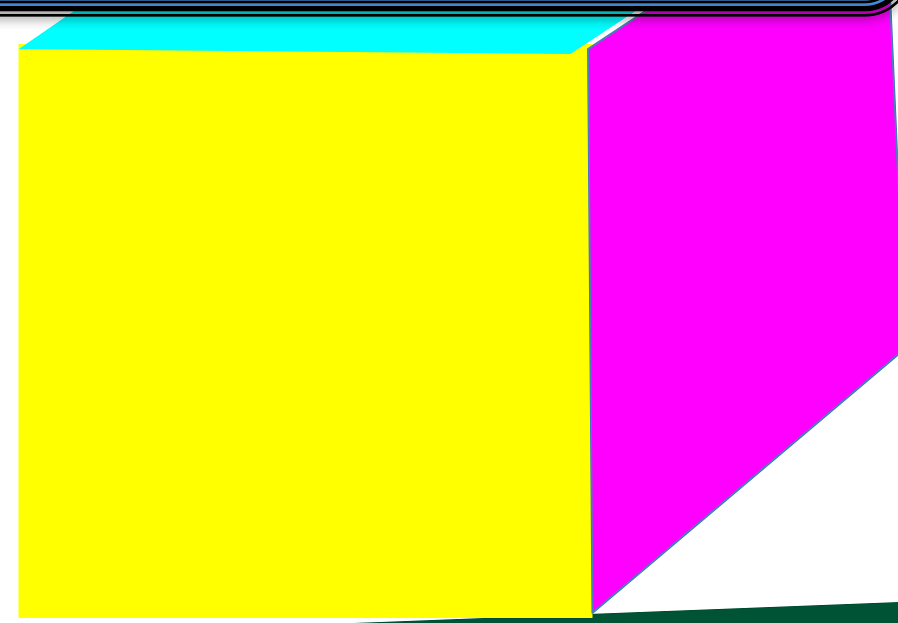
View from "front/top/right" side

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?

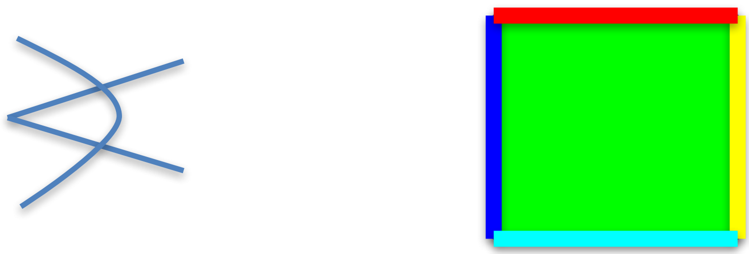


View from "front/top/right" side



View from "back/bottom/left" side

Consider a scene from the right side

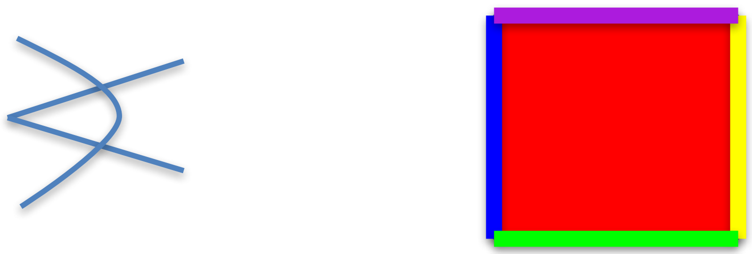


Camera/eyeball

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



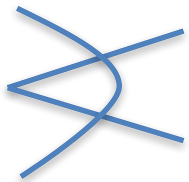
Camera/eyeball

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

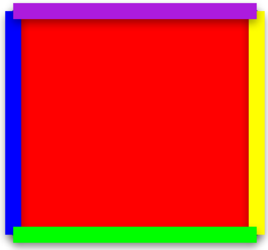
Face	Color
Front	Blue
Right	Green
Top	Red
Back	Yellow
Left	Purple
Bottom	Cyan

What do we render?

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



Camera/eyeball

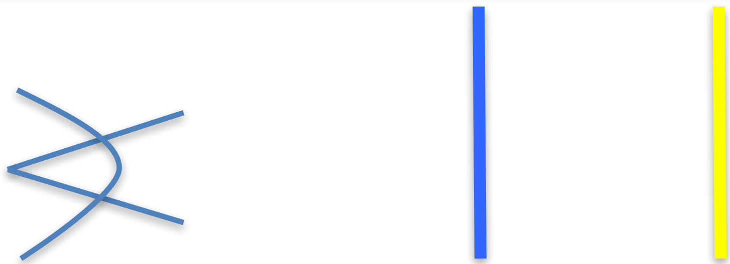


Camera oriented directly at Front face,
seen from the Top side

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

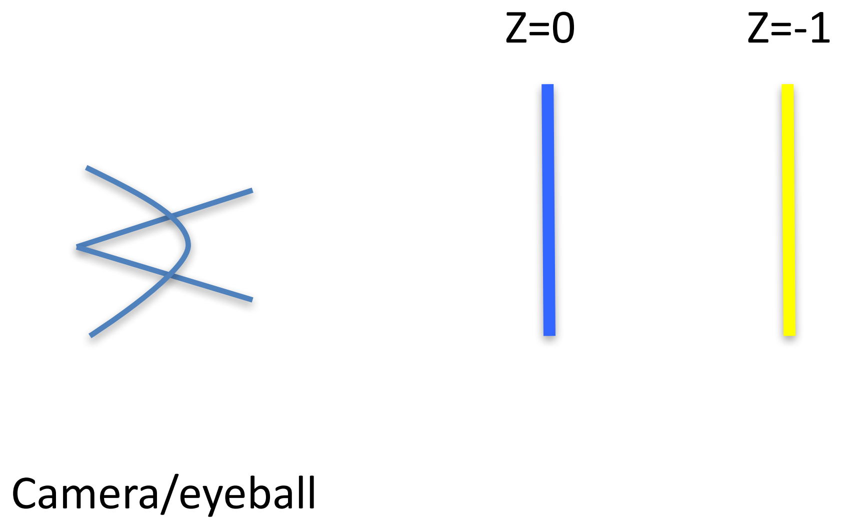
- Project 1B,1C:

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

- Now…

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

What do we render?



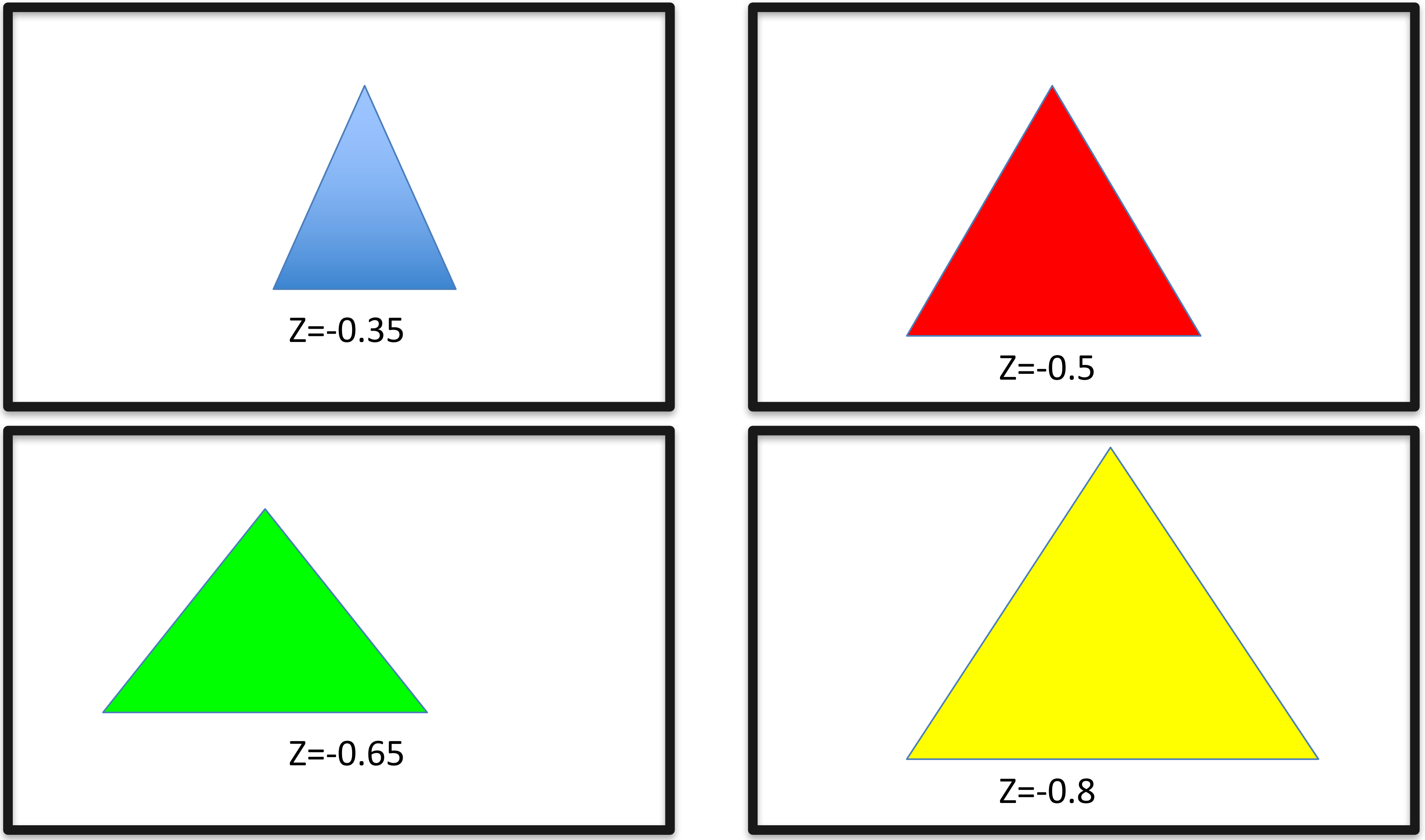
Camera oriented directly at Front face, seen from the Top side

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

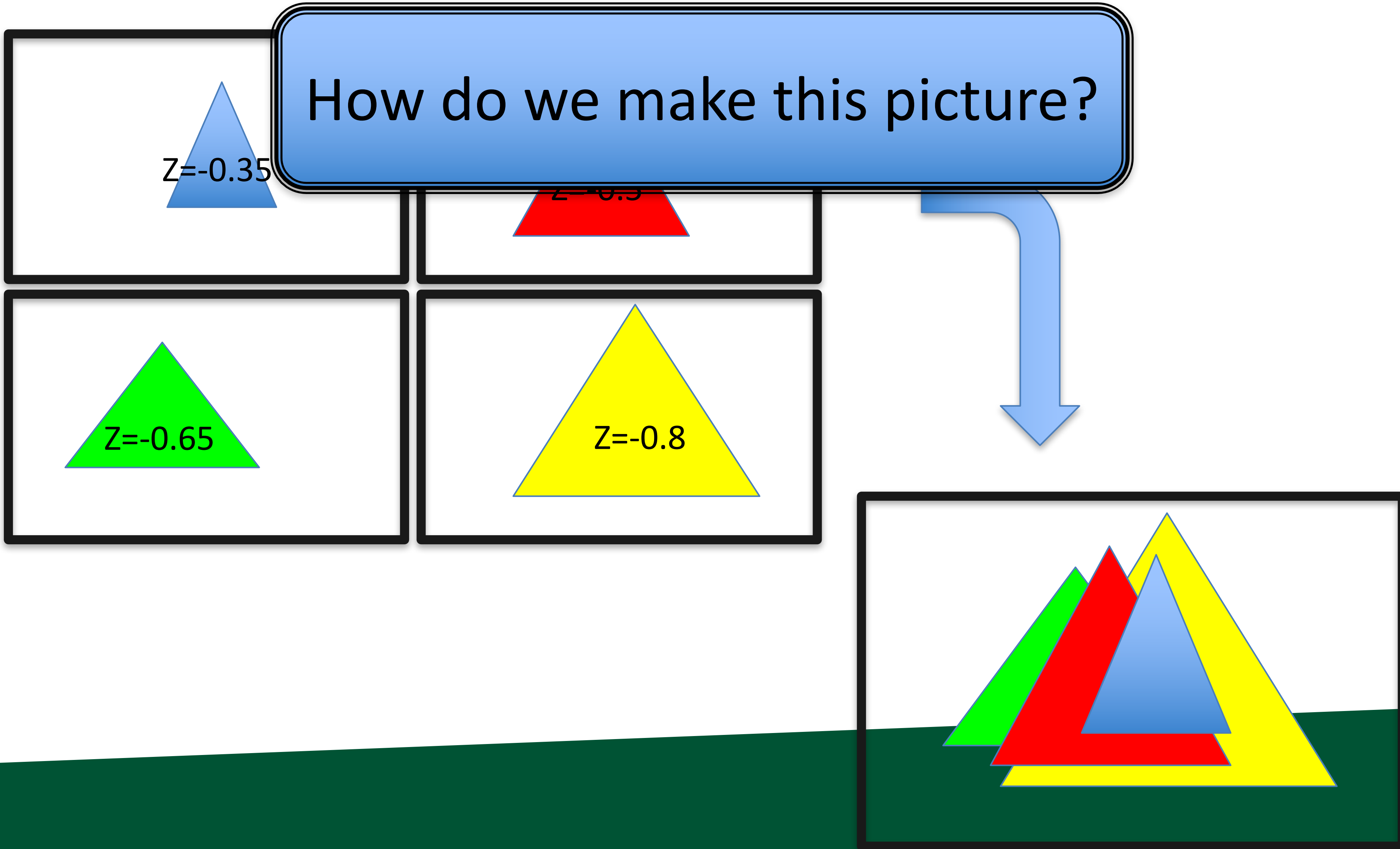
Using depth when rendering

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

Consider 4 triangles with constant Z values

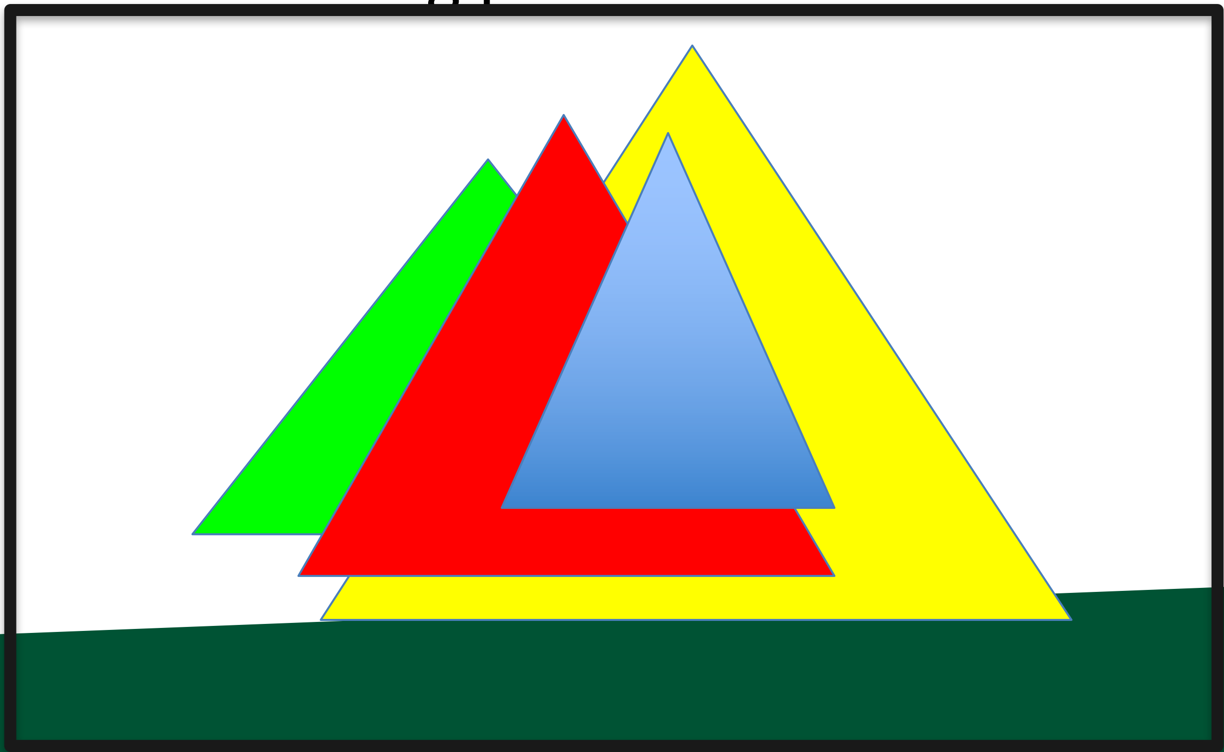


Consider 4 triangles with constant Z values



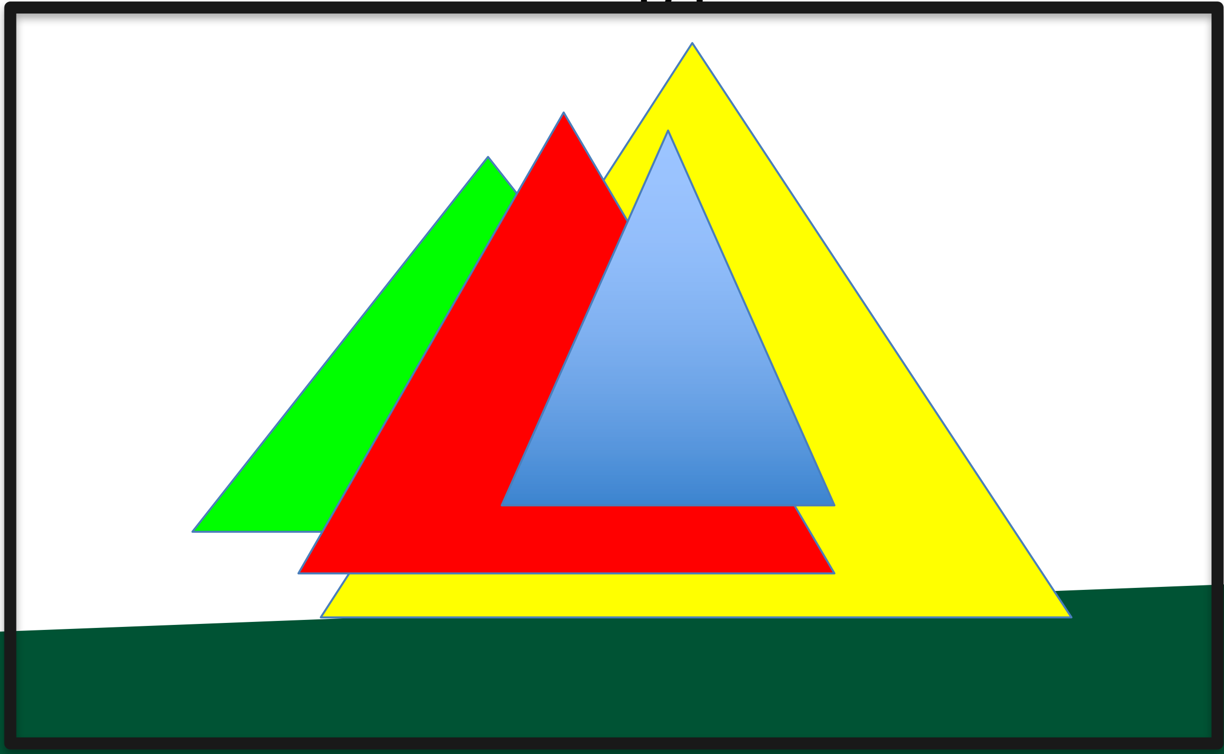
Idea #1

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

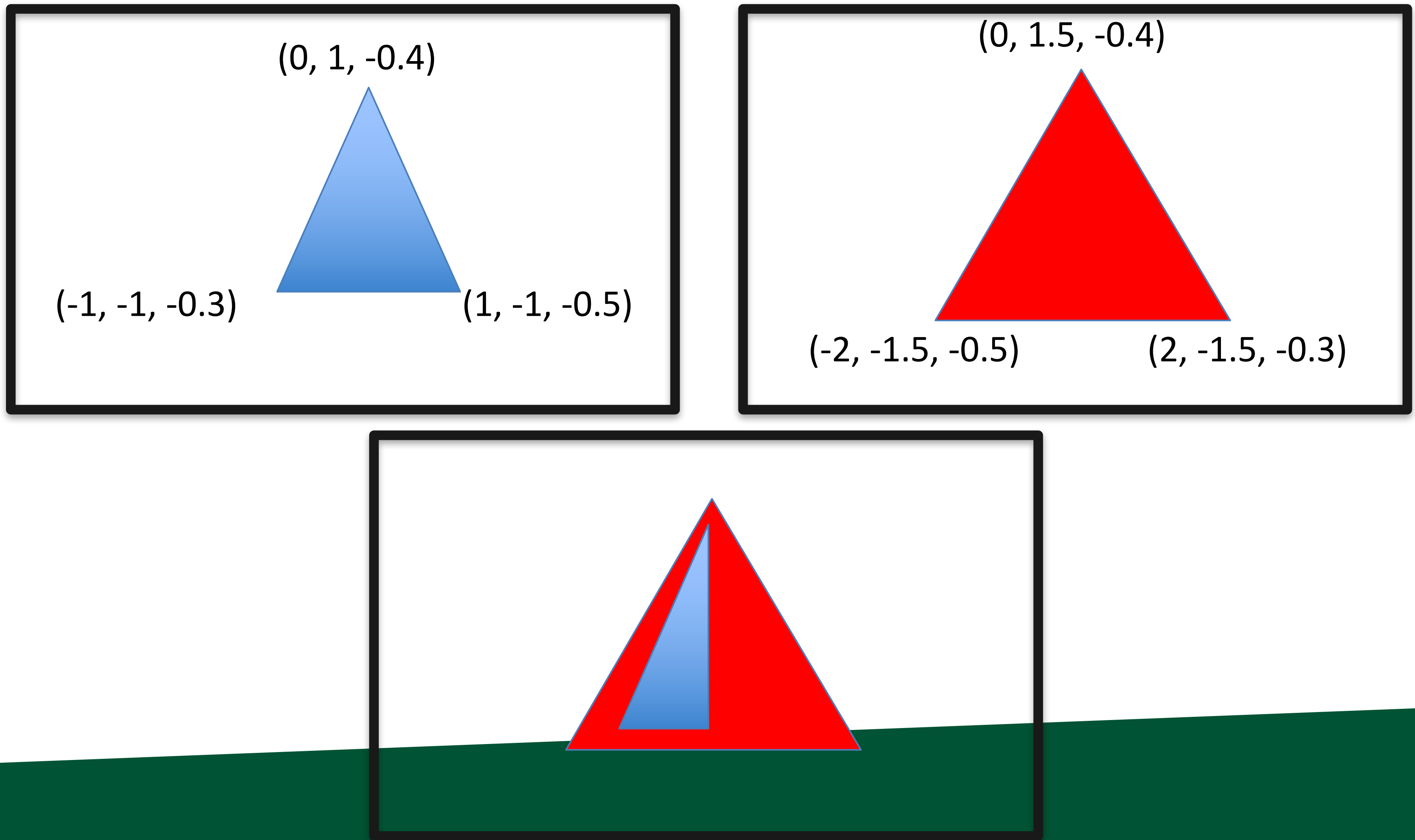


Idea #2

- 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.

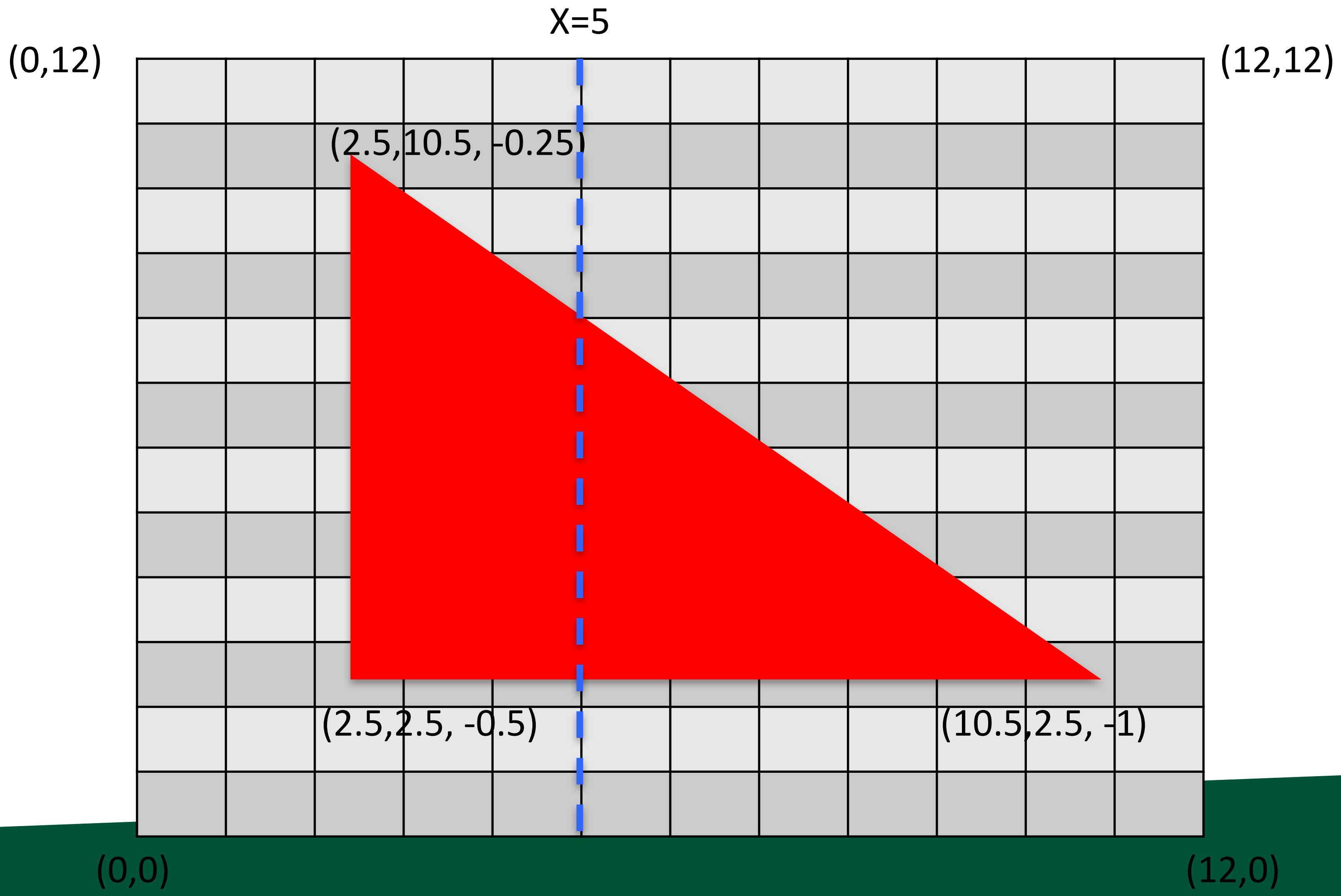
Scanline algorithm for one triangle

- 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 w/ Z-Buffer

- 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
  - Interpolate z(bottomEnd) and z(topEnd) from triangle vertices
  - For r in [ceiling(bottomEnd) → floor(topEnd)] ; do
    - Interpolate z(c,r) from z(bottomEnd) and z(topEnd)
    - If (z(c,r) > depthBuffer(c,r))
      - ImageColor(r, c) ← triangle color
      - depthBuffer(c,r) = z(c,r)

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.