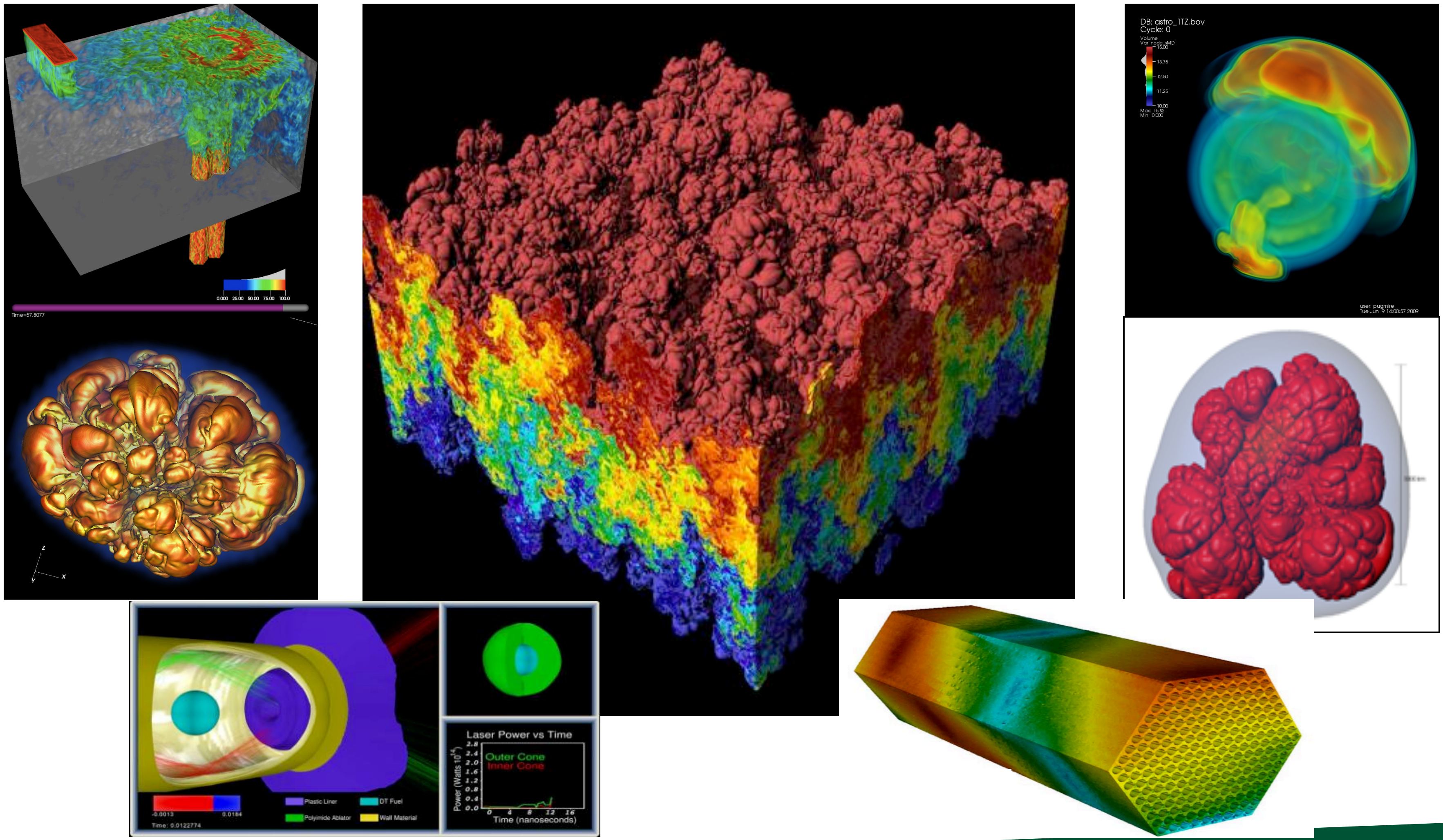
UNIVERSITY OF OREGON

CIS 441/541: Intro to Computer Graphics
Lecture 4: Z-Buffer, Project 1D, Cameras & Matrices



April 13, 2021

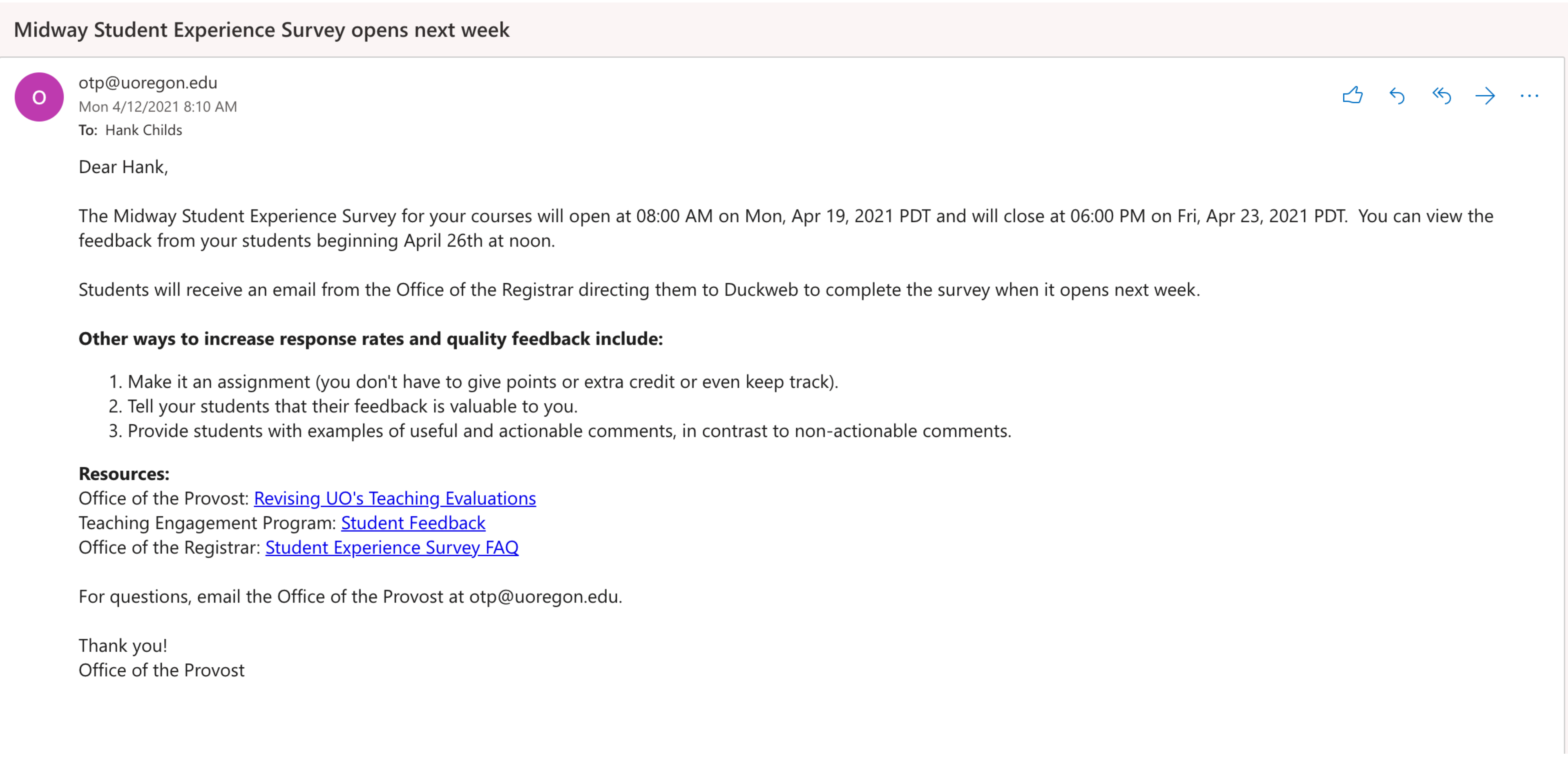
Hank Childs, University of Oregon

Class Thursday – Quiz #2

- Starts at 9am
- 9am-915am: Q&A / group OH on topics related to project 1, graphics
- 915am-945am: quiz (be here at 910)
  - You must be present for these 30 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
    - Have a friend to text for internet issues
  - Know your UO ID
  - Have a camera
- This "lecture" will not be recorded

UNIVERSITY OF OREGON

Midway Experience



Midway Student Experience Survey opens next week

O

otp@uoregon.edu
Mon 4/12/2021 8:10 AM
To: Hank Childs

Dear Hank,

The Midway Student Experience Survey for your courses will open at 08:00 AM on Mon, Apr 19, 2021 PDT and will close at 06:00 PM on Fri, Apr 23, 2021 PDT. You can view the feedback from your students beginning April 26th at noon.

Students will receive an email from the Office of the Registrar directing them to Duckweb to complete the survey when it opens next week.

Other ways to increase response rates and quality feedback include:

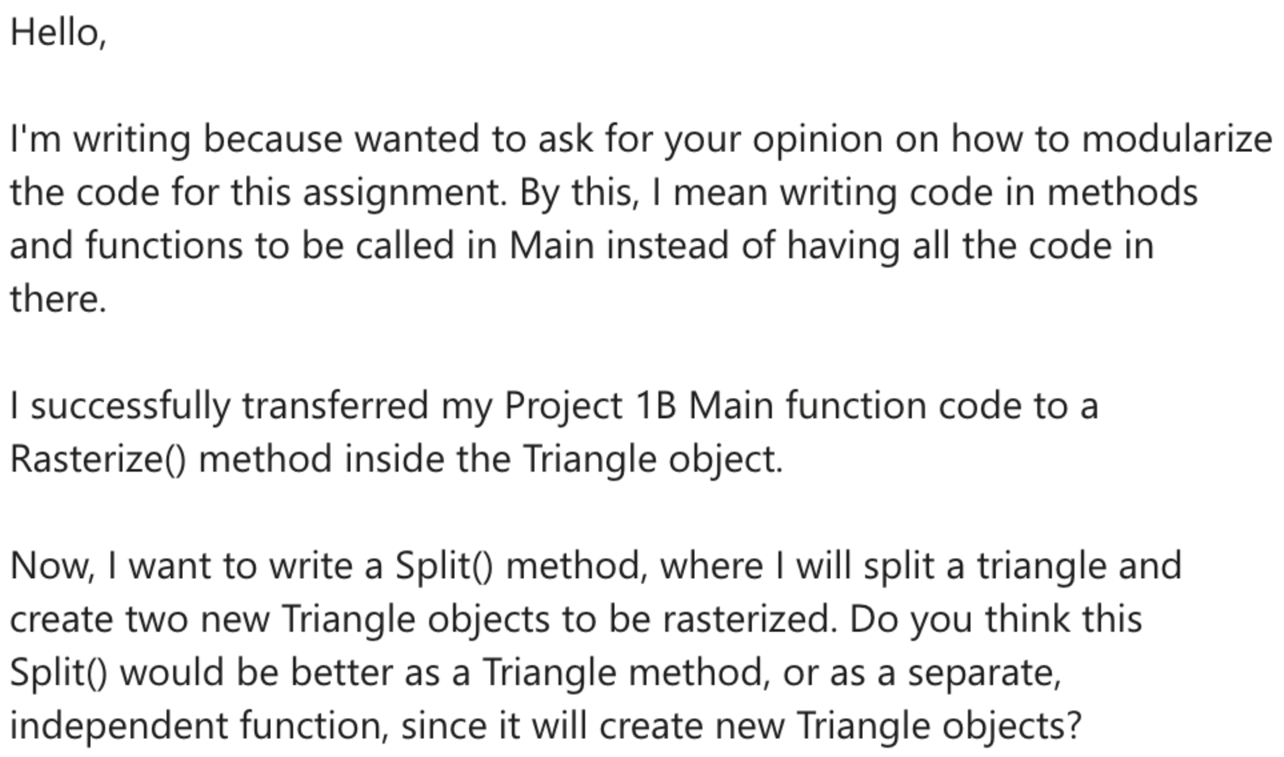
1. Make it an assignment (you don't have to give points or extra credit or even keep track).
2. Tell your students that their feedback is valuable to you.
3. Provide students with examples of useful and actionable comments, in contrast to non-actionable comments.

Resources:
Office of the Provost: Revising UO's Teaching Evaluations
Teaching Engagement Program: Student Feedback
Office of the Registrar: Student Experience Survey FAQ

For questions, email the Office of the Provost at otp@uoregon.edu.

Thank you!
Office of the Provost

Question



Hello,

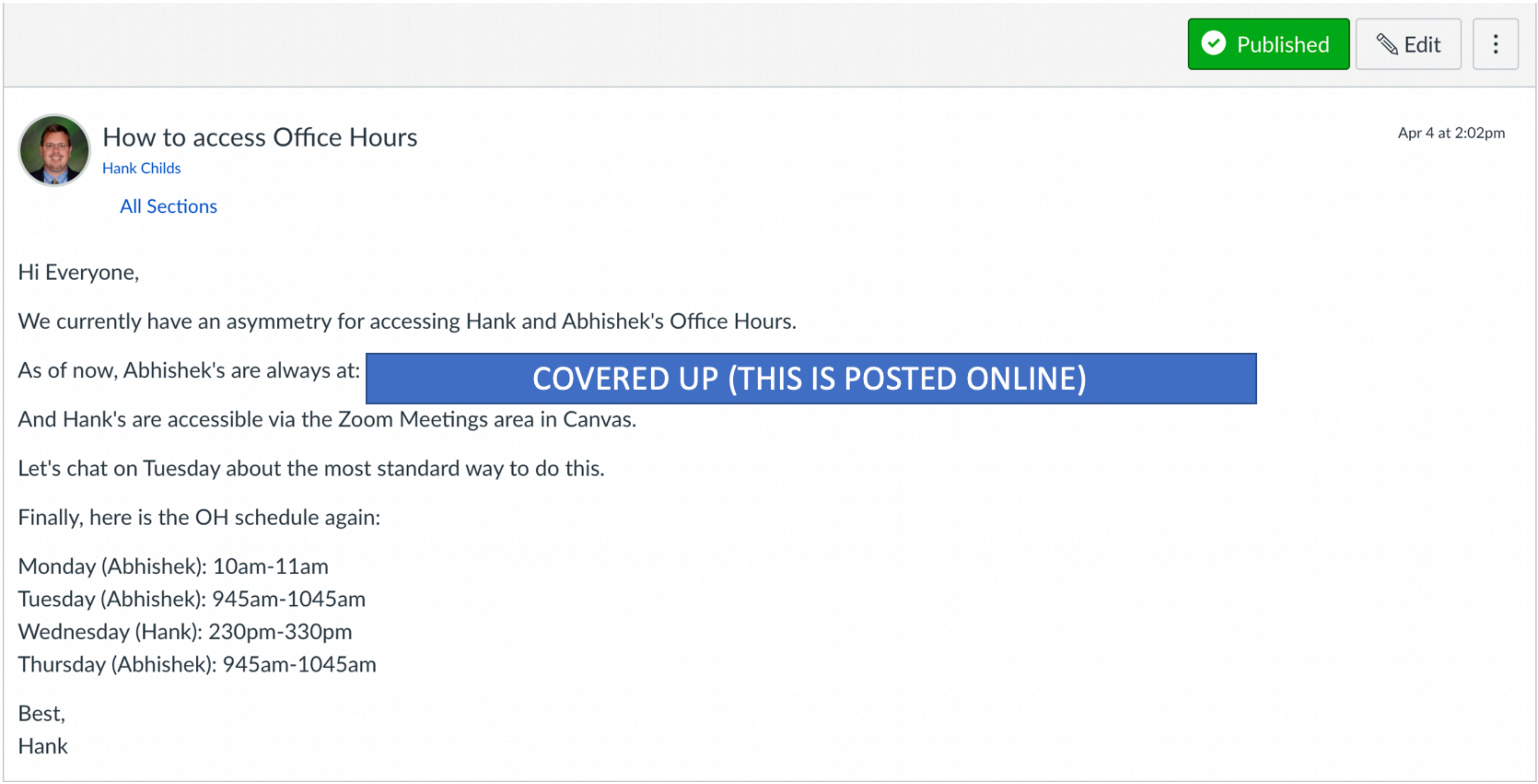
I'm writing because wanted to ask for your opinion on how to modularize the code for this assignment. By this, I mean writing code in methods and functions to be called in Main instead of having all the code in there.

I successfully transferred my Project 1B Main function code to a Rasterize() method inside the Triangle object.

Now, I want to write a Split() method, where I will split a triangle and create two new Triangle objects to be rasterized. Do you think this Split() would be better as a Triangle method, or as a separate, independent function, since it will create new Triangle objects?

UNIVERSITY OF OREGON

Week 3 Office Hours



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.

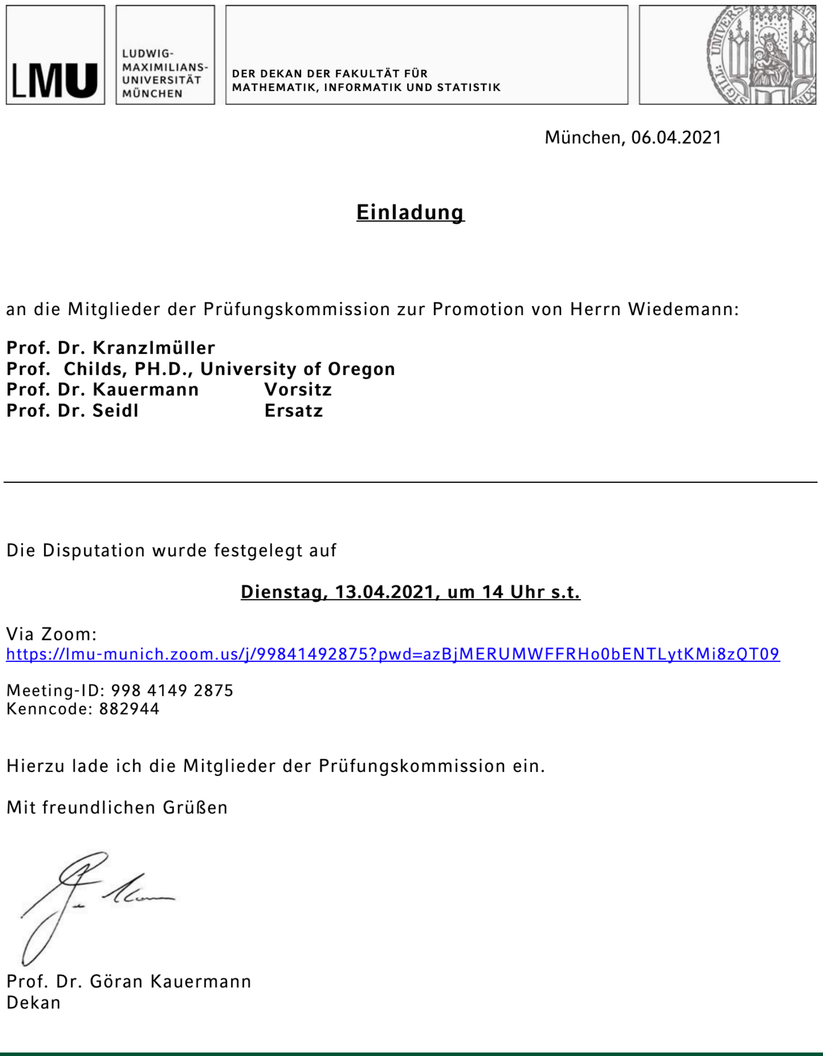
Let's chat on Tuesday about the most standard way to do this.

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

Feeling a little tired...



LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

DER DEKAN DER FAKULTÄT FÜR MATHEMATIK, INFORMATIK UND STATISTIK

München, 06.04.2021

Einladung

an die Mitglieder der Prüfungskommission zur Promotion von Herrn Wiedemann:

Prof. Dr. Kranzlmüller
Prof. Childs, PH.D., University of Oregon
Prof. Dr. Kauermann Vorsitz
Prof. Dr. Seidl Ersatz

Die Disputation wurde festgelegt auf

Dienstag, 13.04.2021, um 14 Uhr s.t.

Via Zoom:
https://lmu-munich.zoom.us/j/99841492875?pwd=azBjMERUMWFFRHo0bENTLytKMi8zQT09

Meeting-ID: 998 4149 2875
Kenncode: 882944

Hierzu lade ich die Mitglieder der Prüfungskommission ein.

Mit freundlichen Grüßen

[illegible]

Prof. Dr. Göran Kauermann
Dekan

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:

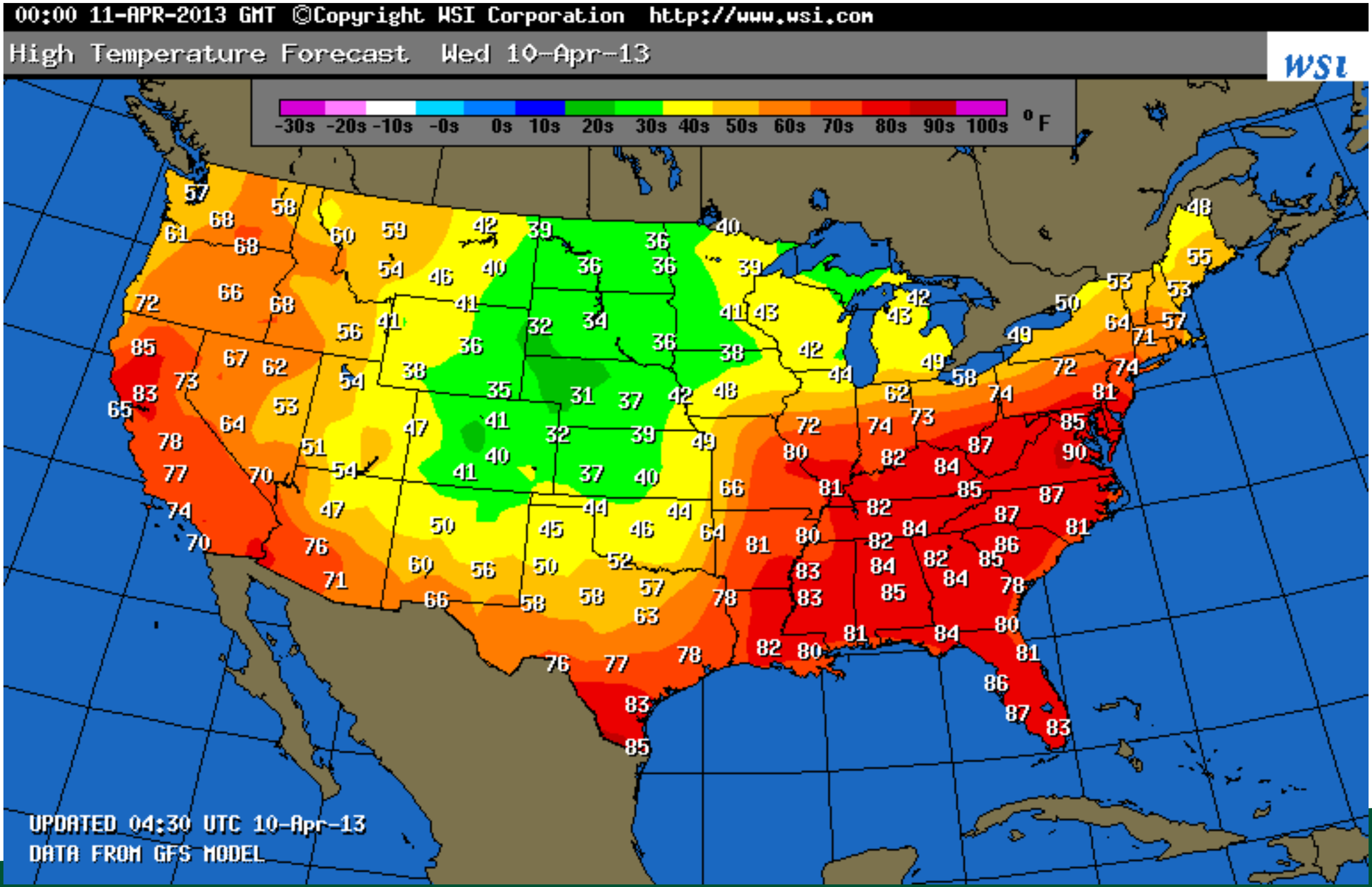
How to deposit triangle colors onto an image?

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 #2. Last week was #1.

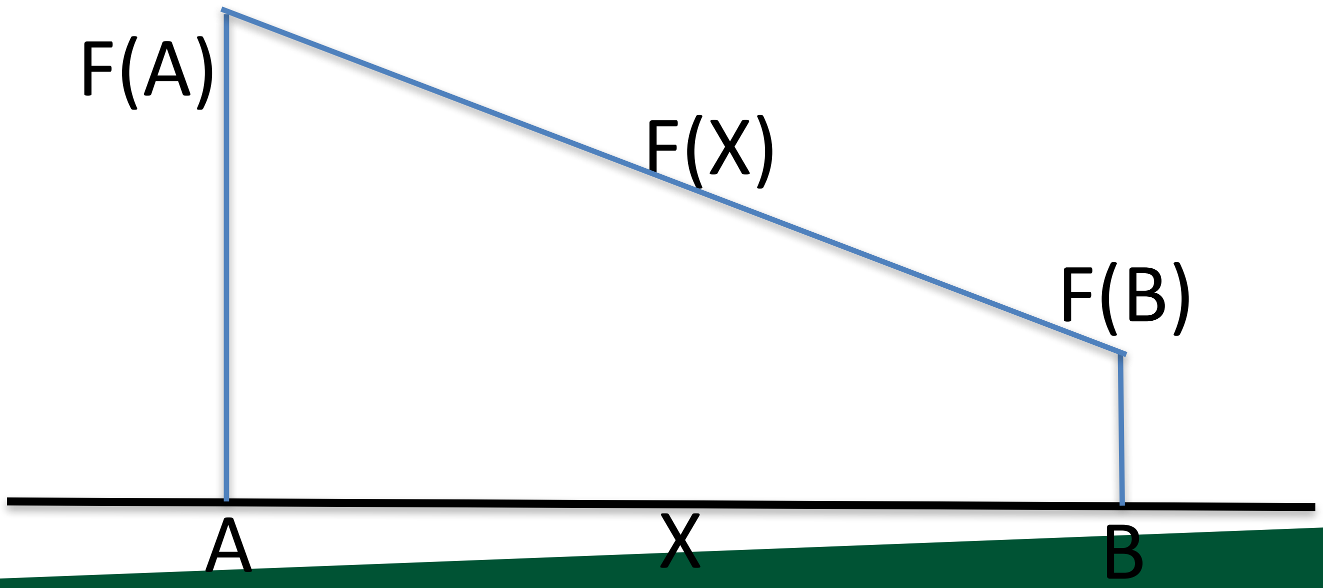
What is a field?


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

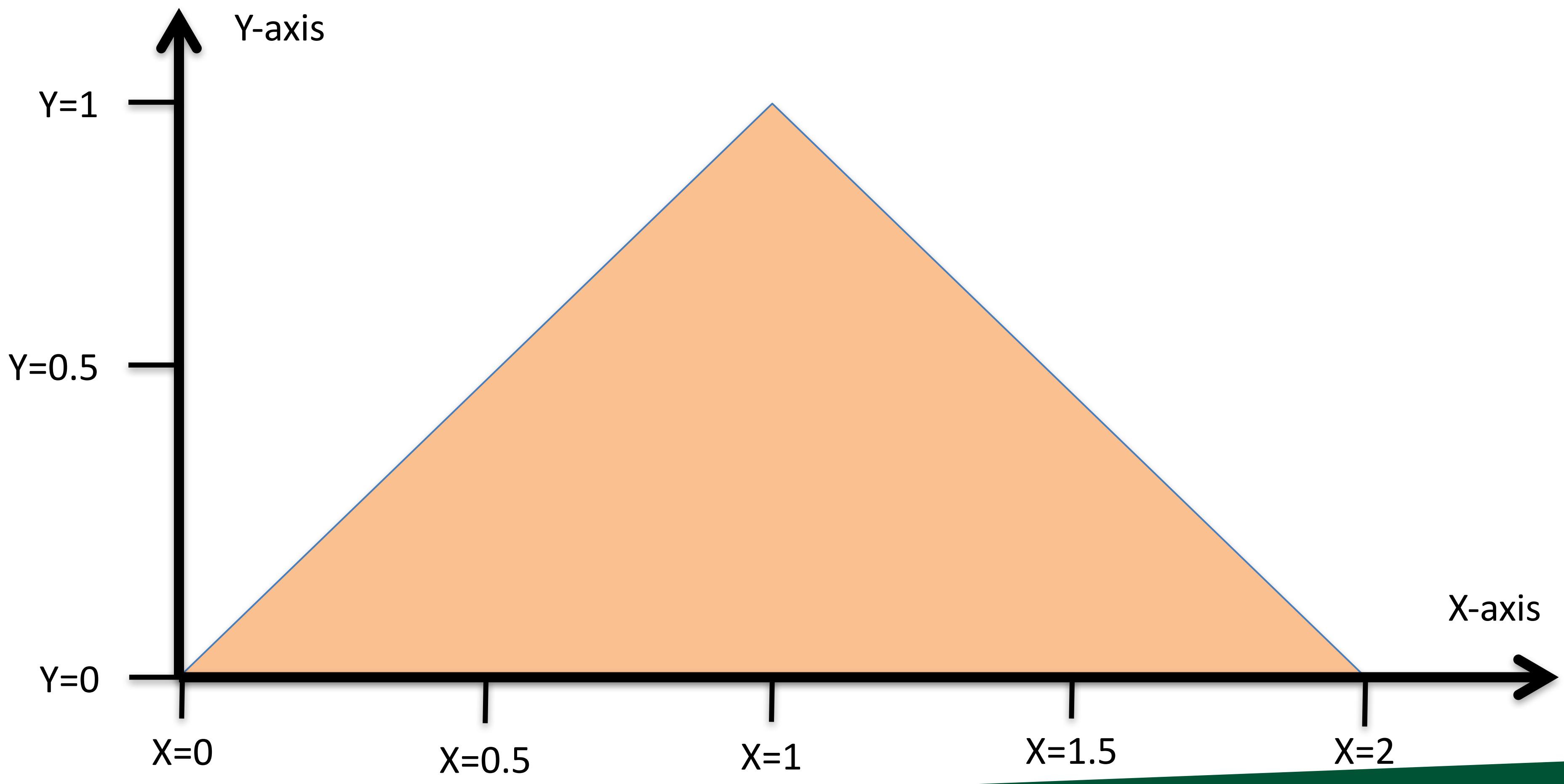
Example field (2D): temperature over the United States

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)

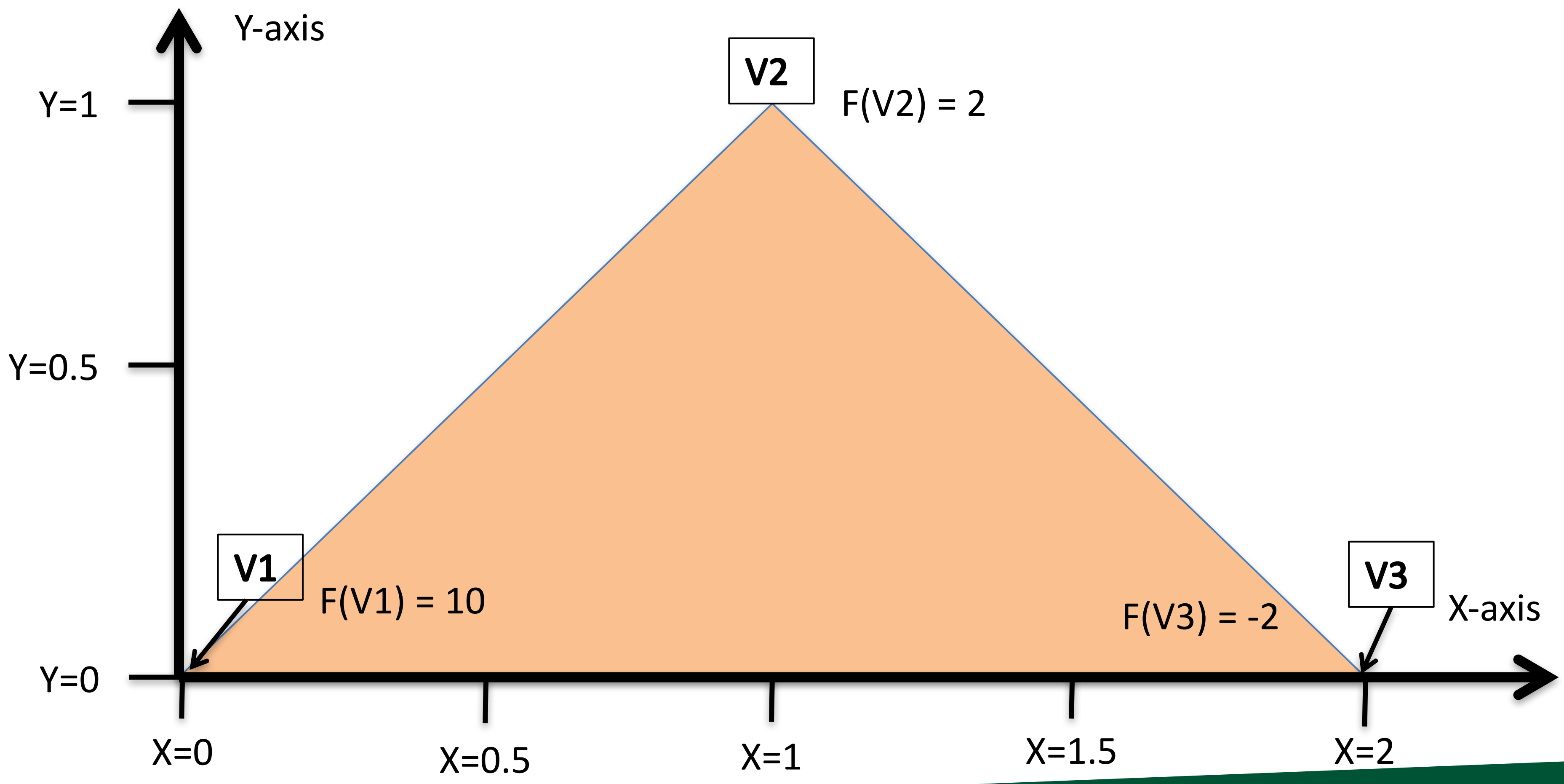


Consider a single scalar field defined on a triangle.

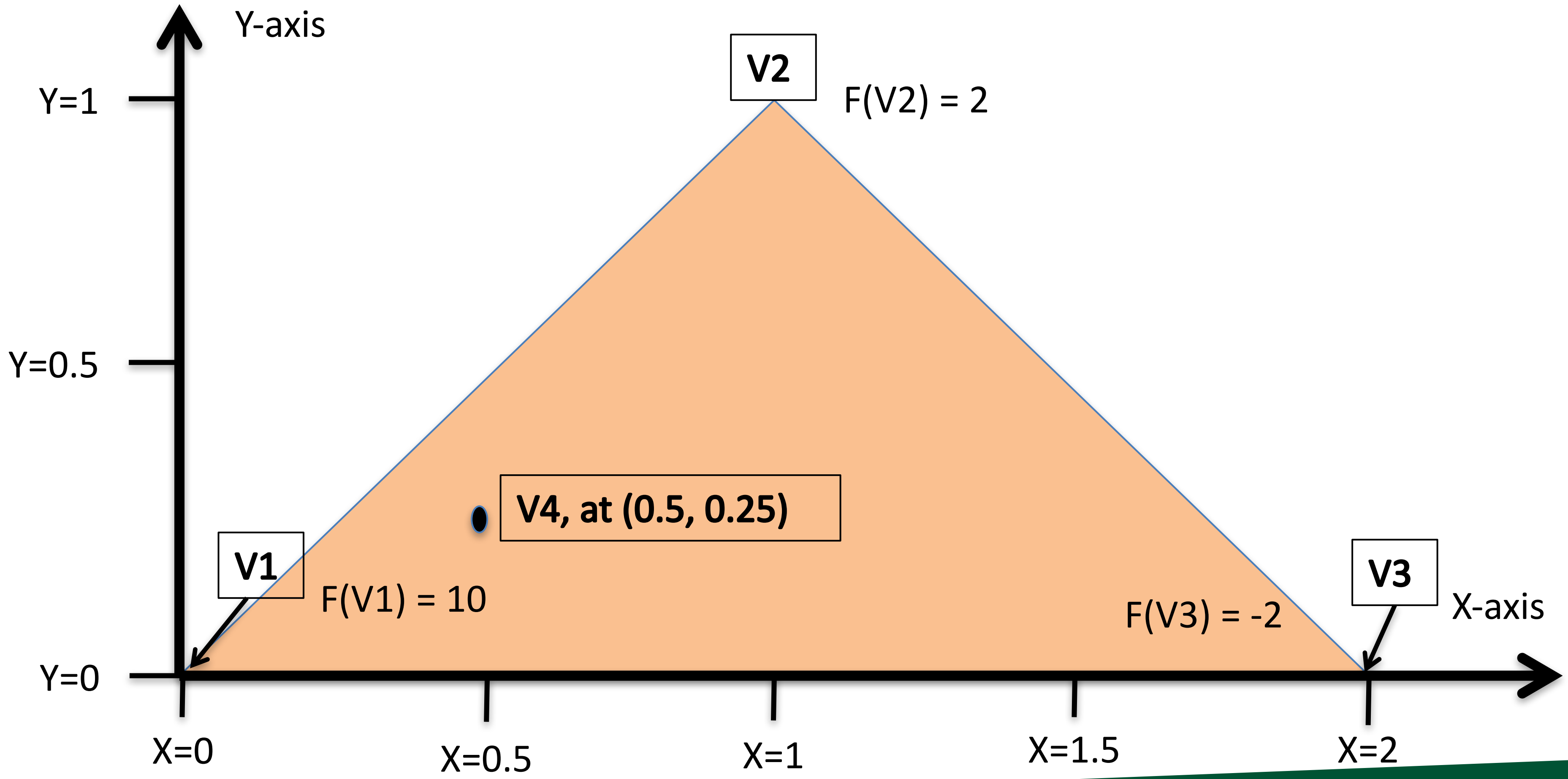


UNIVERSITY OF OREGON

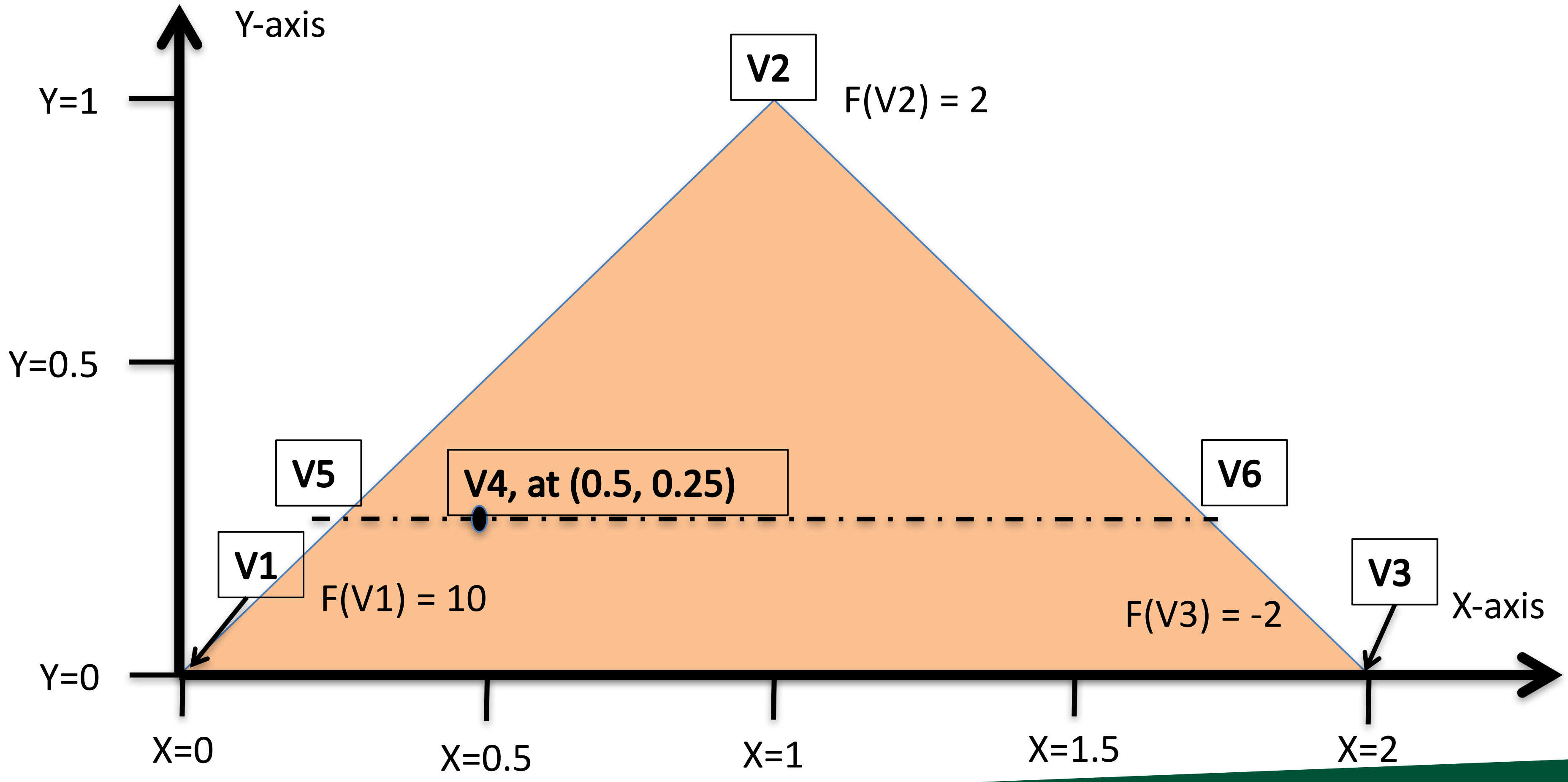
Consider a single scalar field defined on a triangle.



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

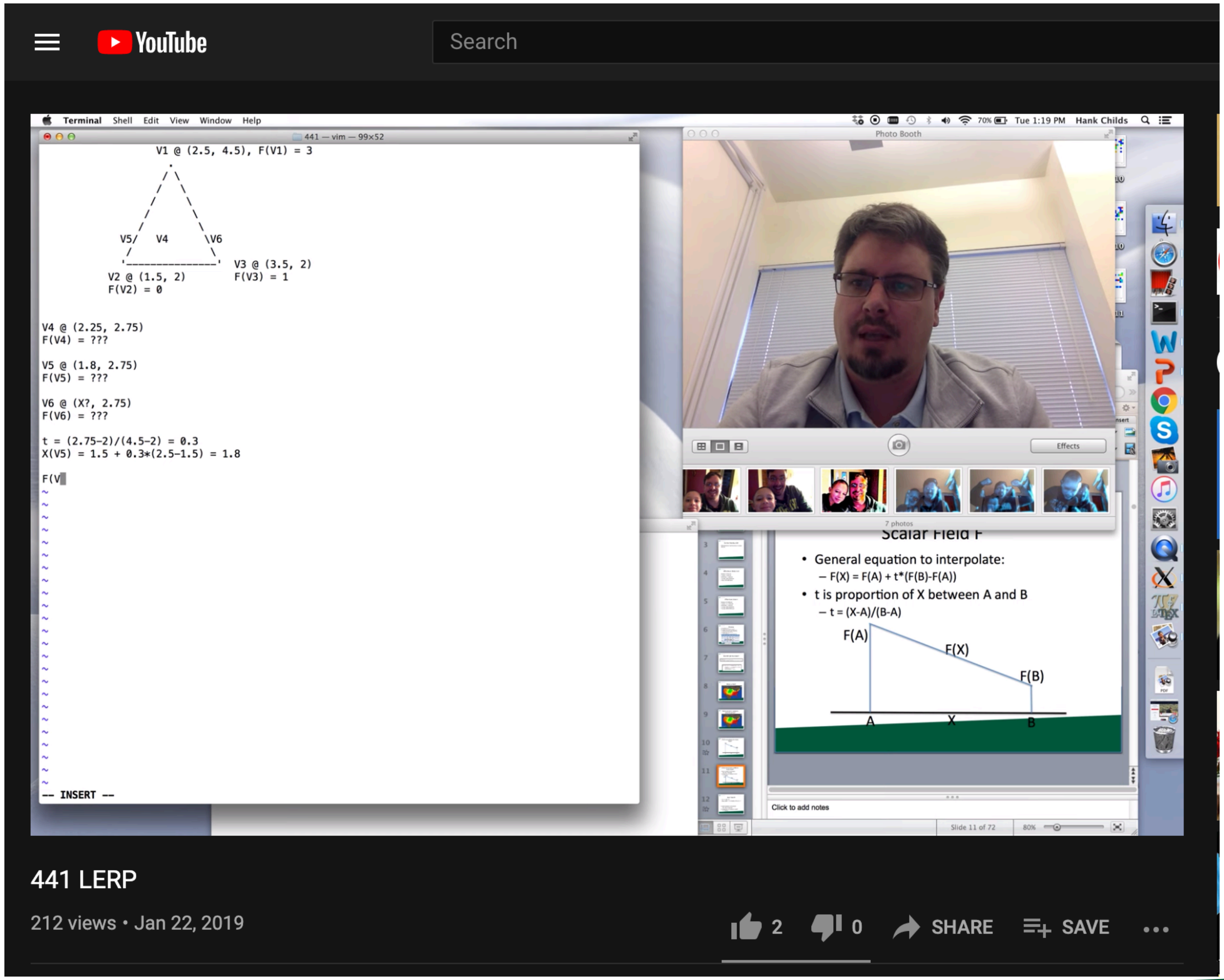
What is F(V4)?


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

UNIVERSITY OF OREGON

Online Lecture B



```
                     V1 @ (2.5, 4.5), F(V1) = 3
                      .
                     / \
                    /   \
                   /     \
                  /       \
                 /         \
          V5/   V4          \V6
           /                 \
          '-----------------'   V3 @ (3.5, 2)
        V2 @ (1.5, 2)           F(V3) = 1
        F(V2) = 0


V4 @ (2.25, 2.75)
F(V4) = ???

V5 @ (1.8, 2.75)
F(V5) = ???

V6 @ (X?, 2.75)
F(V6) = ???

t = (2.75-2)/(4.5-2) = 0.3
X(V5) = 1.5 + 0.3*(2.5-1.5) = 1.8

F(V
-- INSERT --
```

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)

Link posted on class webpage.
Also: https://www.youtube.com/watch?v=8IkioJrMiSs

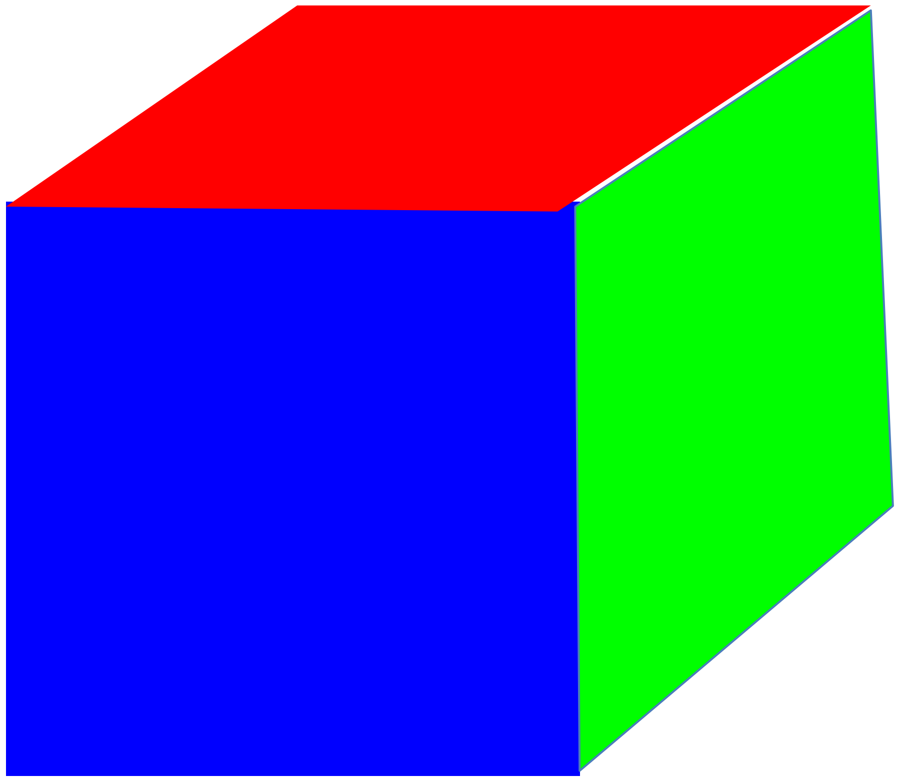
New Vocab Term: a “Fragment”

- When rasterizing a triangle
- When doing a scanline for that triangle
- When that scanline finds a pixel to deposit colors for
- → that contribution is called a fragment

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

How To Resolve When Triangles Overlap: The Z-Buffer

Imagine you have a cube where each face has its own color….

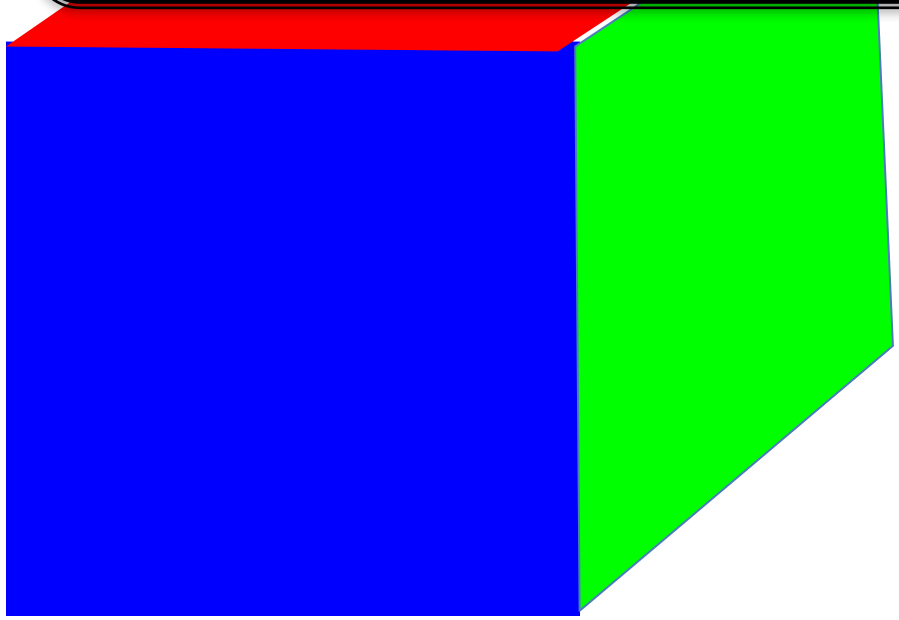


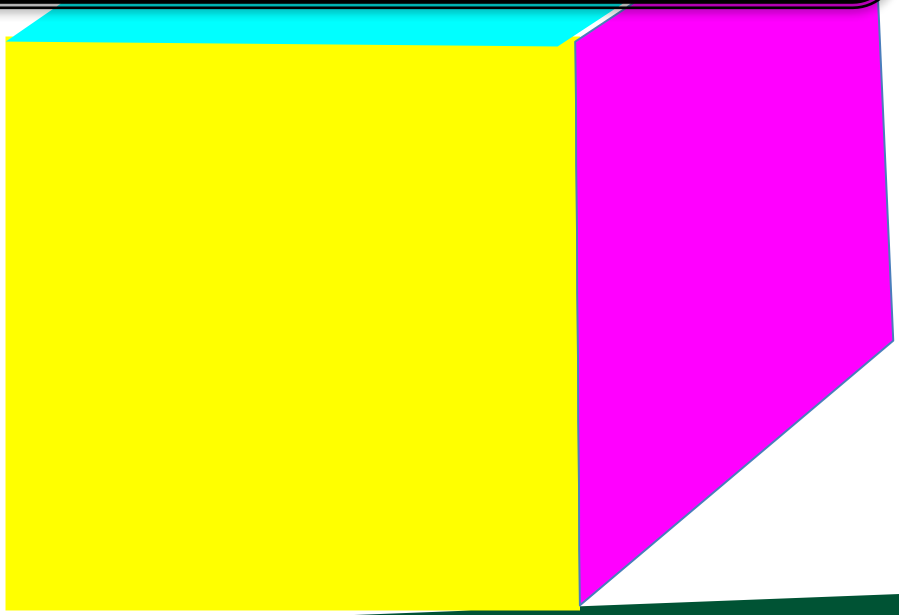
Face	Color
Front	Blue
Right	Green
Top	Red
Back	Yellow
Left	Purple
Bottom	Cyan

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?

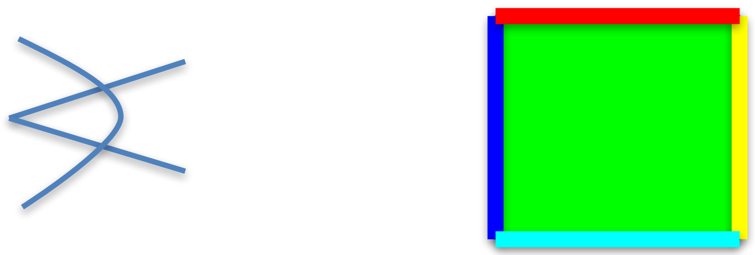




View from "front/top/right" side

View from "back/bottom/left" side

Consider a scene from the right side



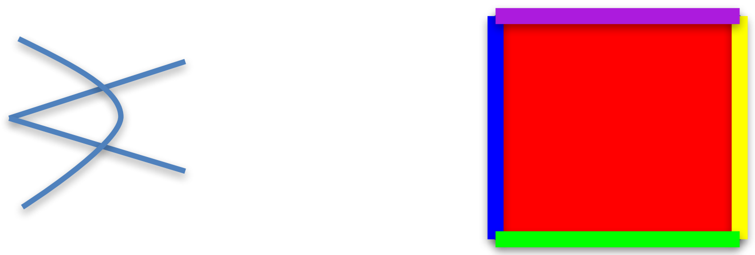
Camera/eyeball

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

UNIVERSITY OF OREGON

Consider the scene from the top side



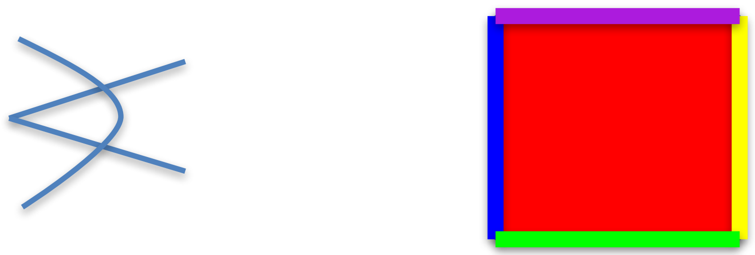
Camera/eyeball

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 (*).



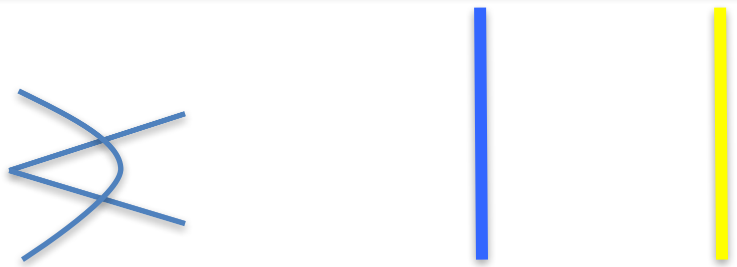
Camera/eyeball

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

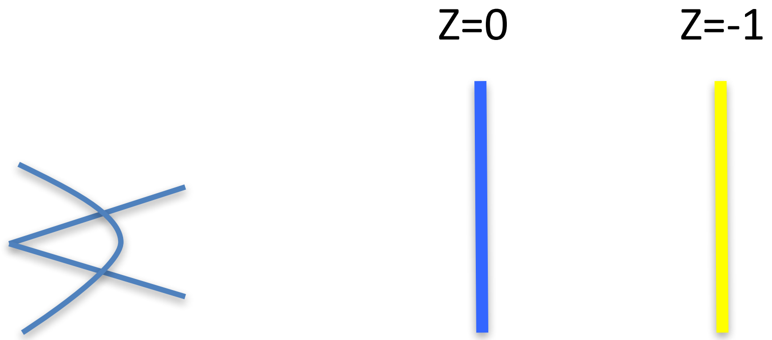
- Project 1B,1C:

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

- Now…

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

What do we render?



Camera/eyeball

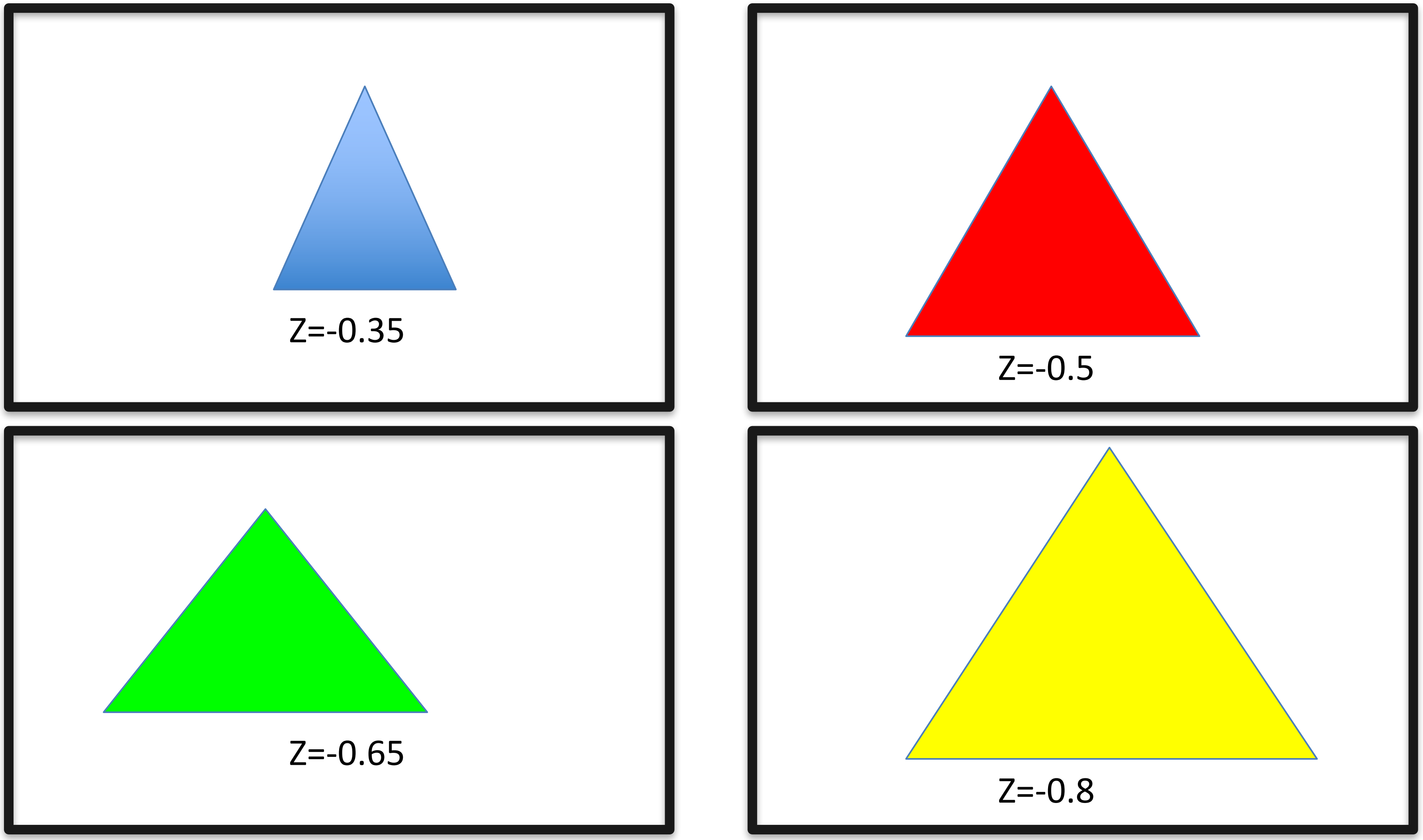
Camera oriented directly at Front face, seen from the Top side

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

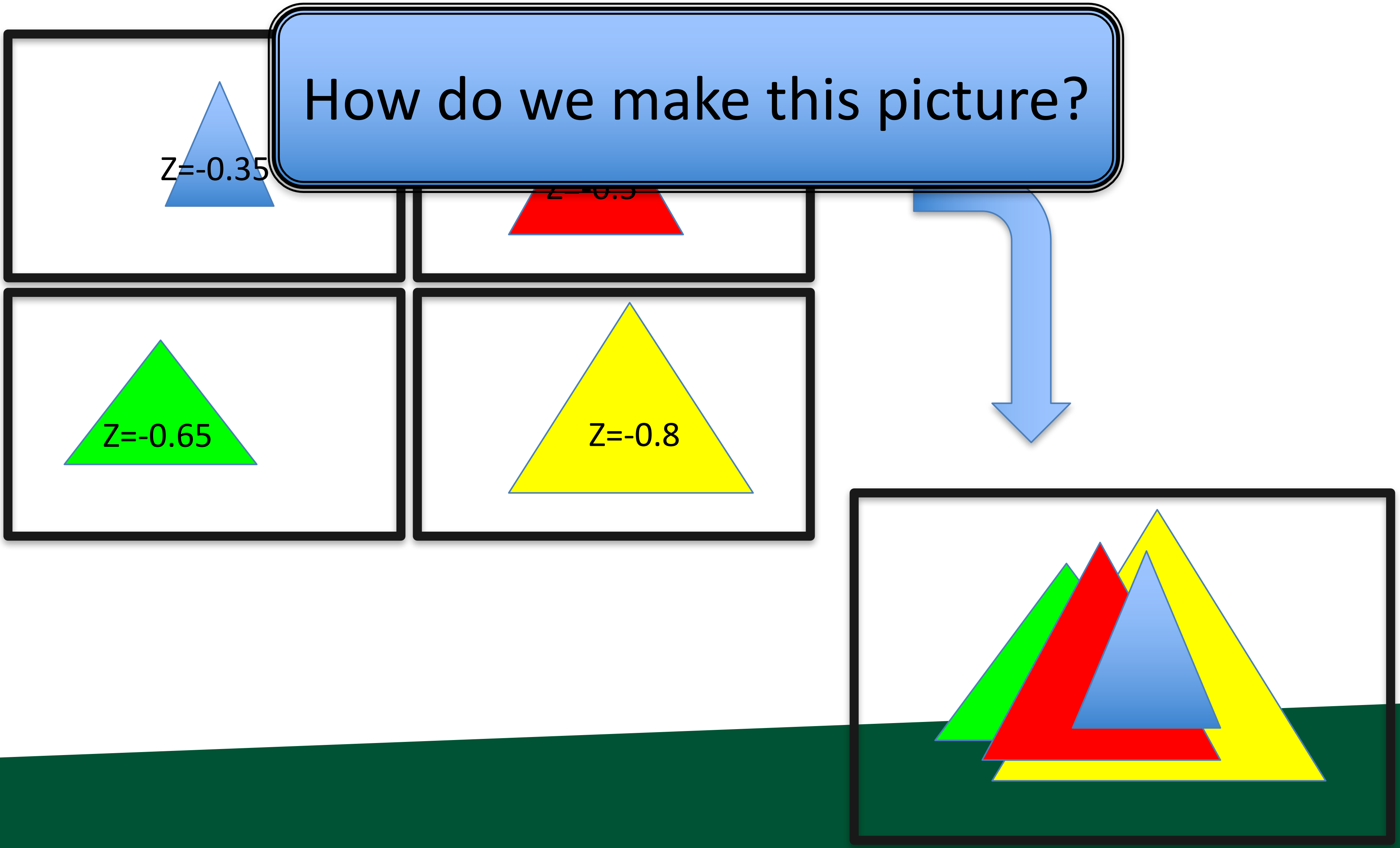
Using depth when rendering

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

Consider 4 triangles with constant Z values

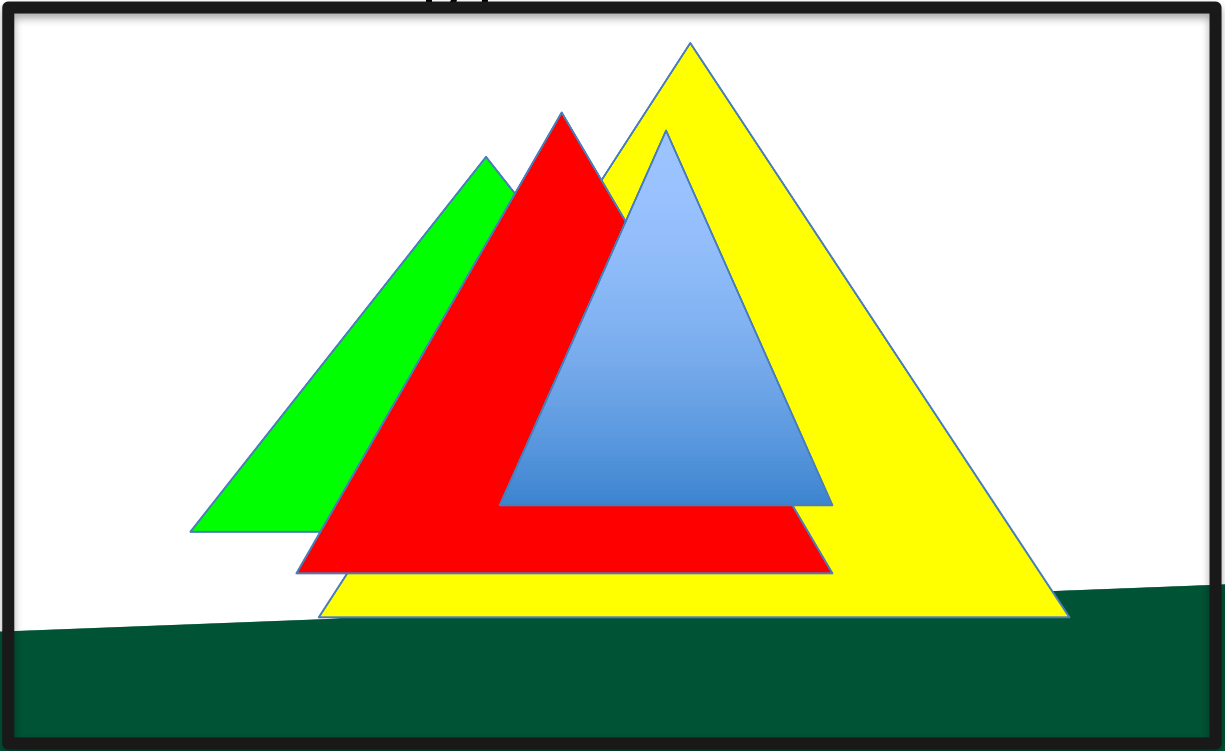


Consider 4 triangles with constant Z values



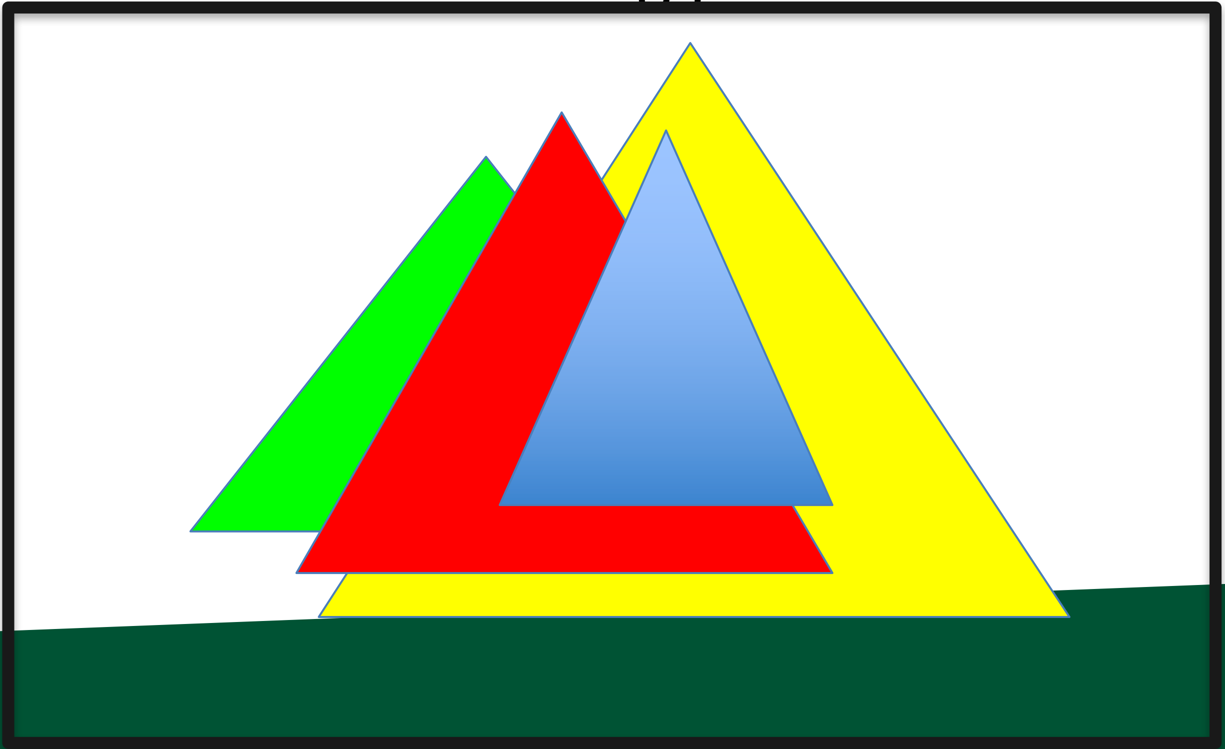
Idea #1

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

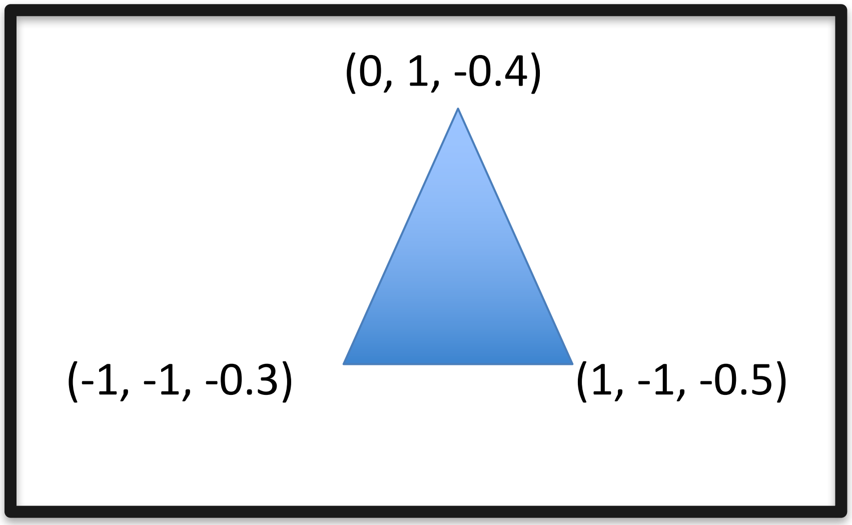


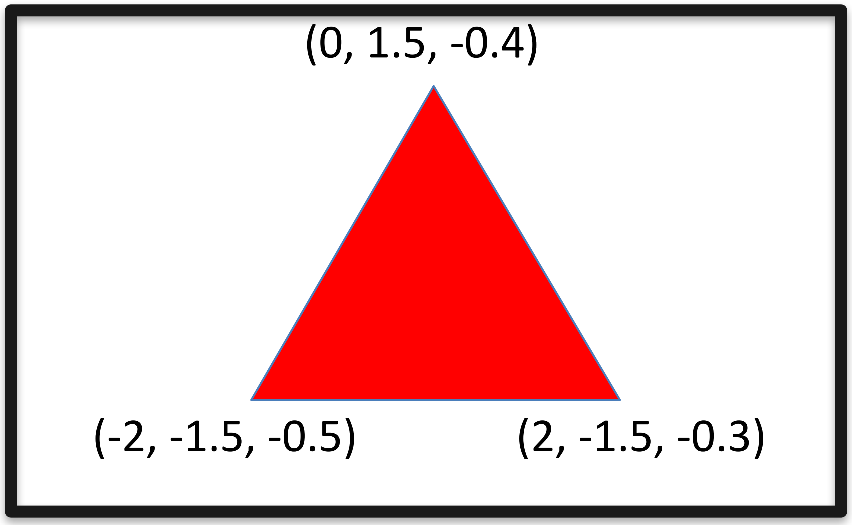
Idea #2

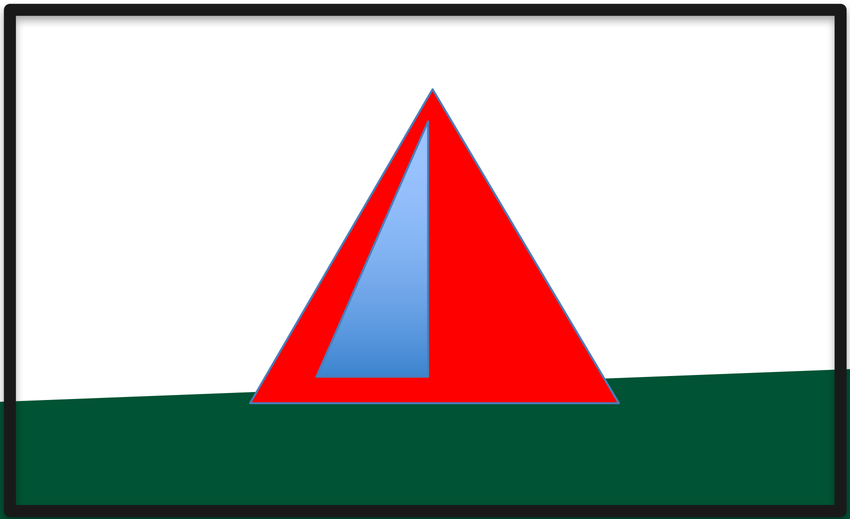
- Sort triangles “front to back” (based on Z)
- Render triangles in front to back order
  - Do not overwrite existing pixels.



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.

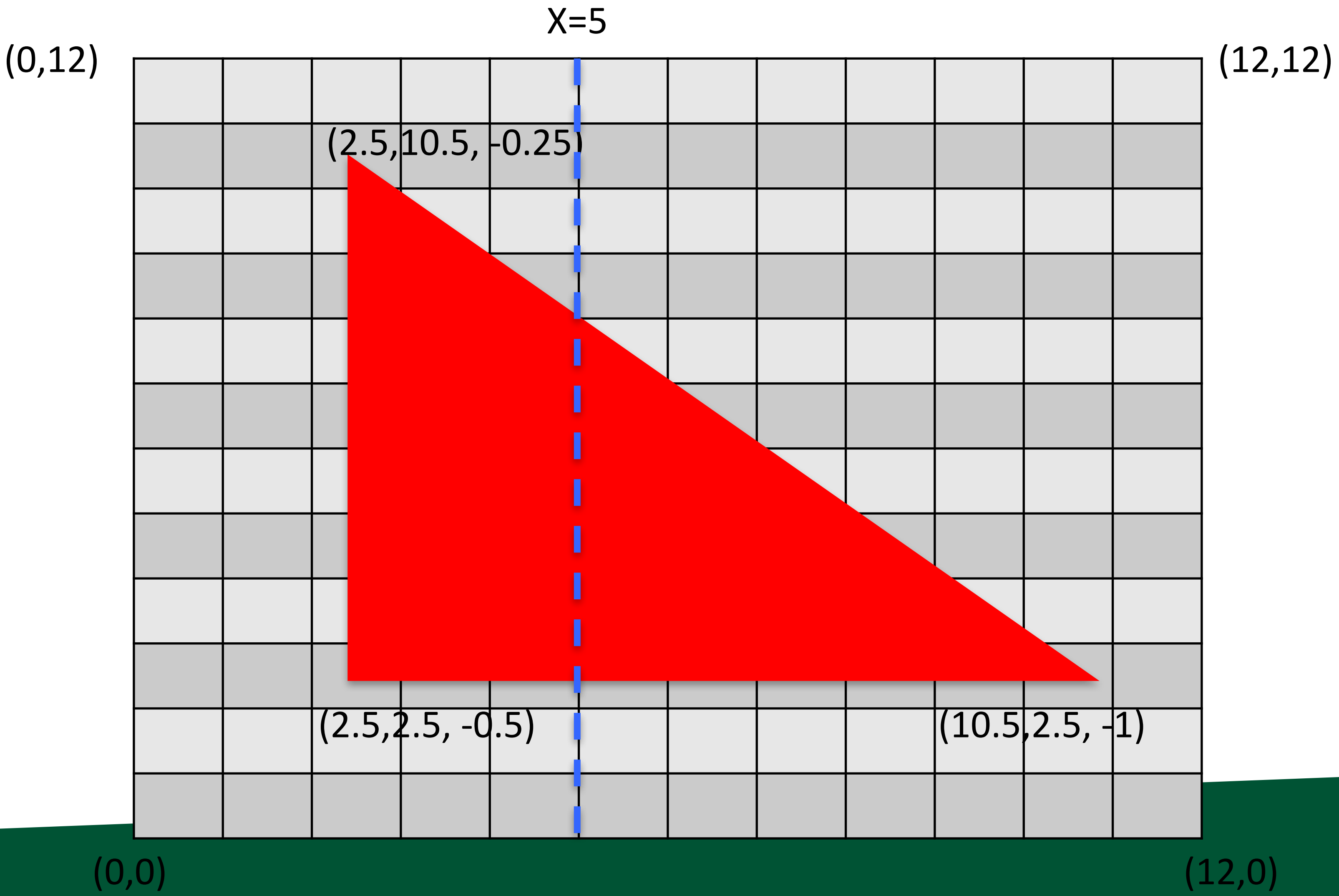
Scanline algorithm for one triangle

- 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

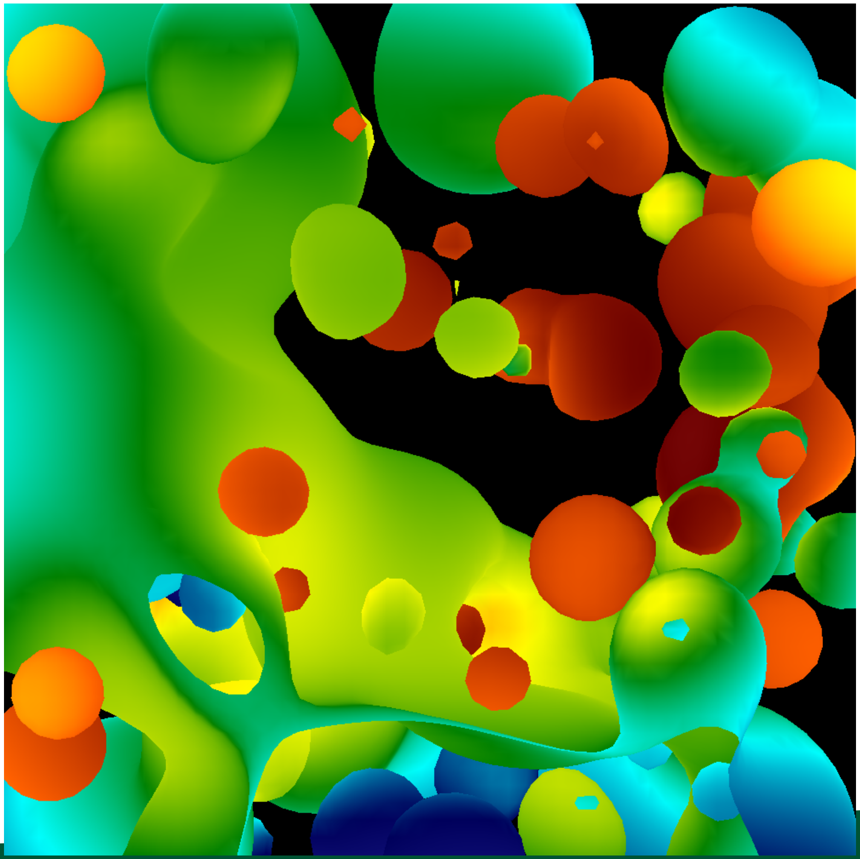


Interpolation and Triangles

- 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.

Project #1D (5%), Due Weds April 21

- 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...
  - As in: ceil__441(value*255)

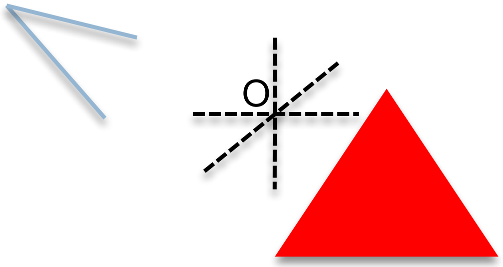
Changes to data structures

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

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

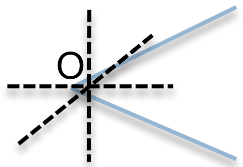
Cameras and Matrices

Our goal



World space:

Triangles in native Cartesian coordinates
Camera located anywhere



Camera space:

Camera located at origin, looking down -Z
Triangle coordinates relative to camera frame

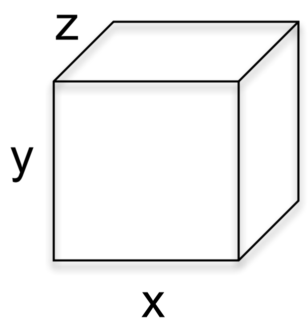
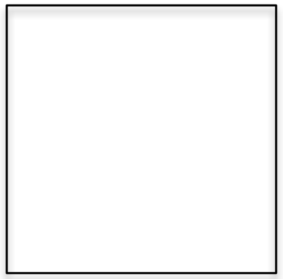


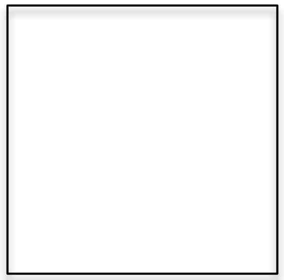
Image space:

All viewable objects within
-1 <= x,y,z <= +1



Screen space:

All viewable objects within
-1 <= x, y <= +1



Device space:

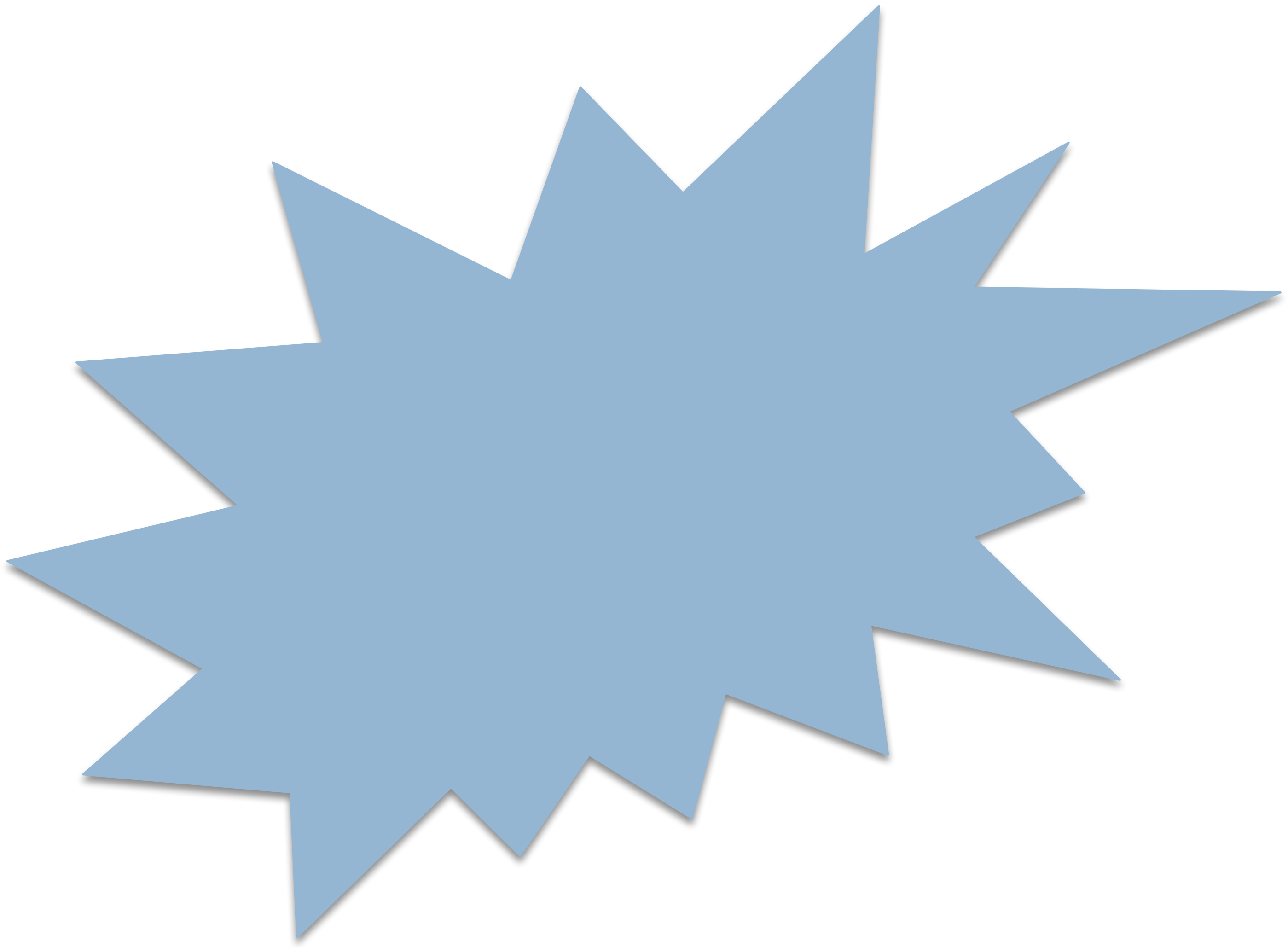
All viewable objects within
0<=x<=width, 0 <=y<=height

MATH!

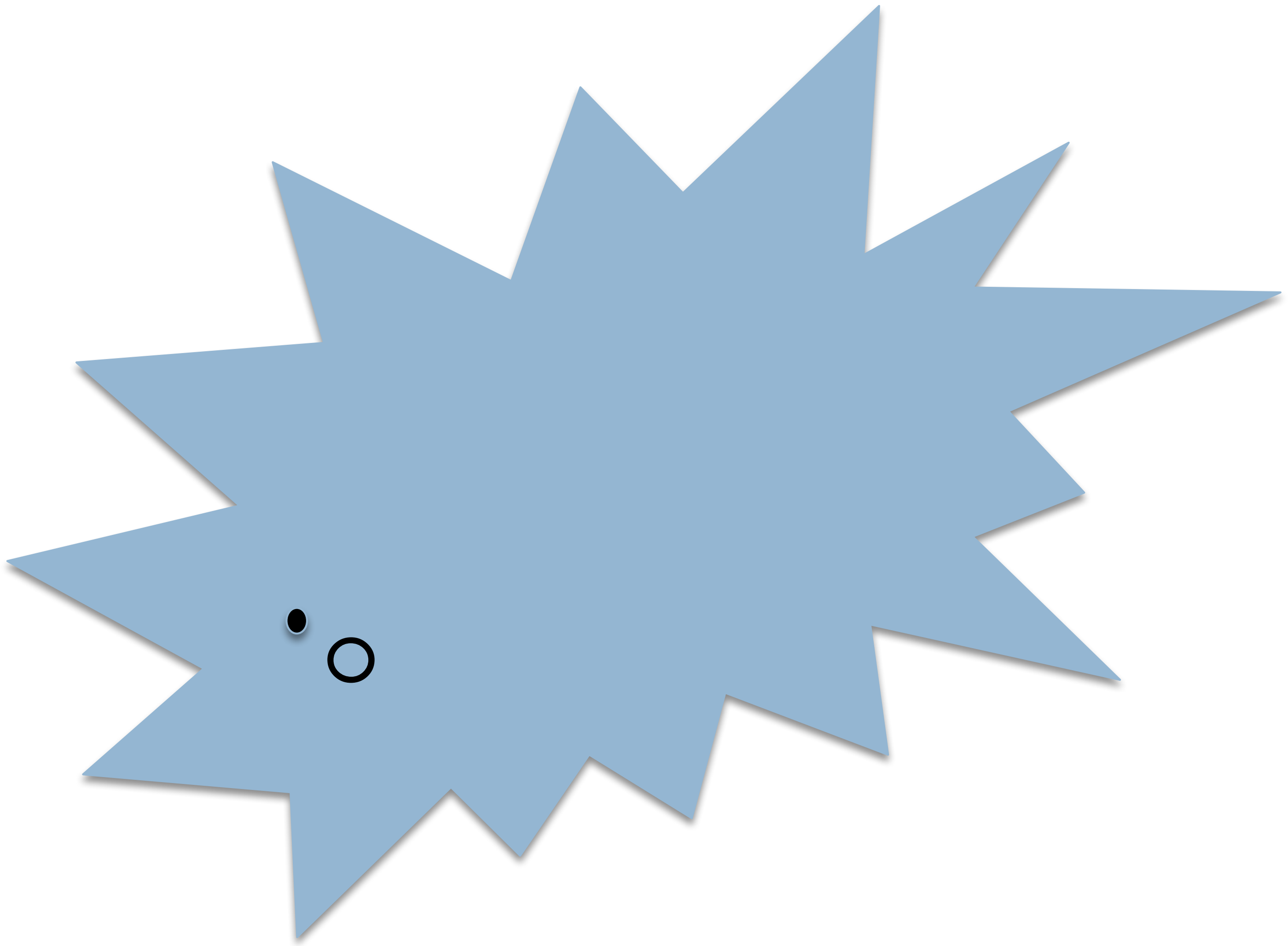
Space

- A “space” is a set of points
- Many types of spaces

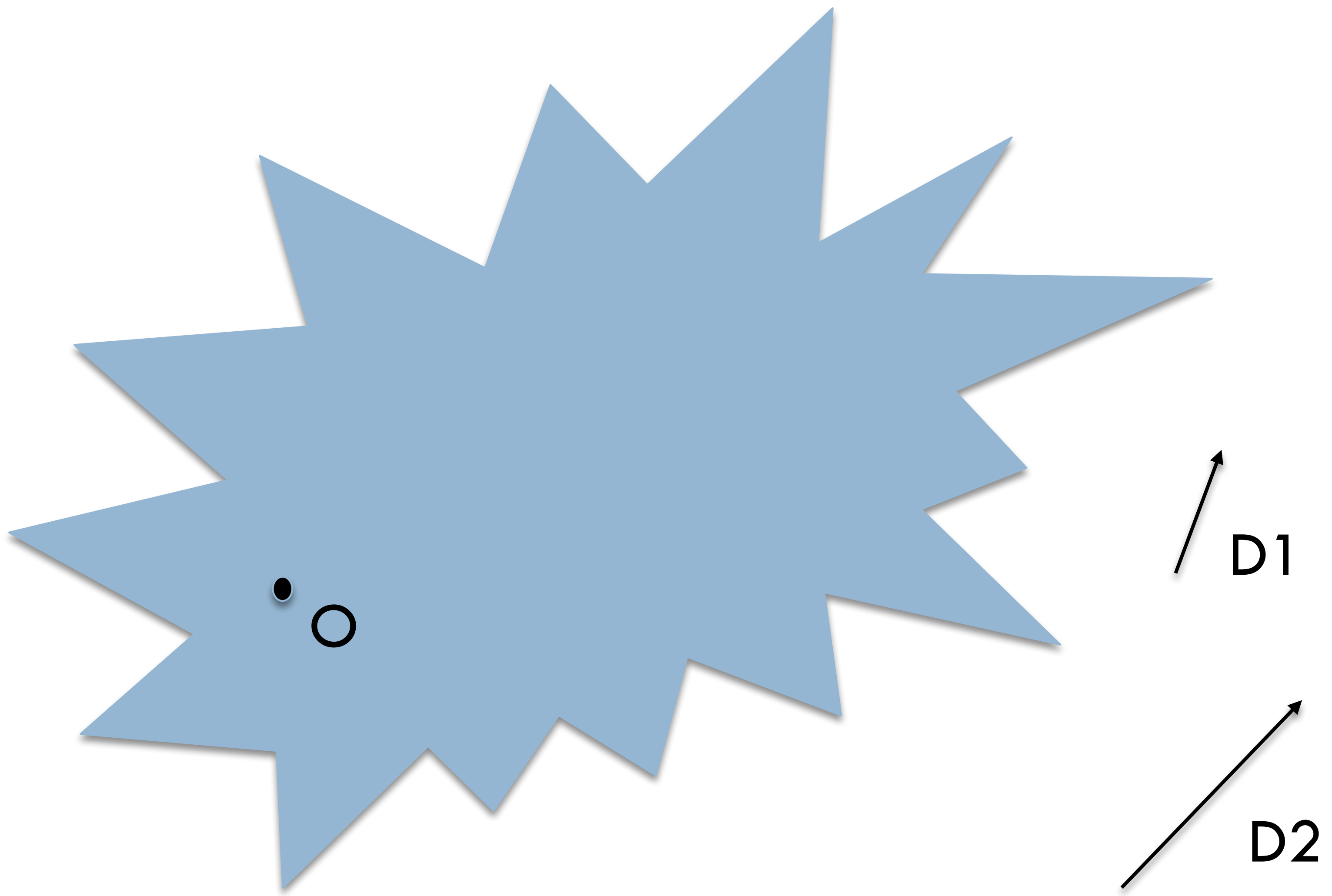
Here is a space 'S': the points in the blue shape



We can pick an arbitrary point in S and call it our "origin."



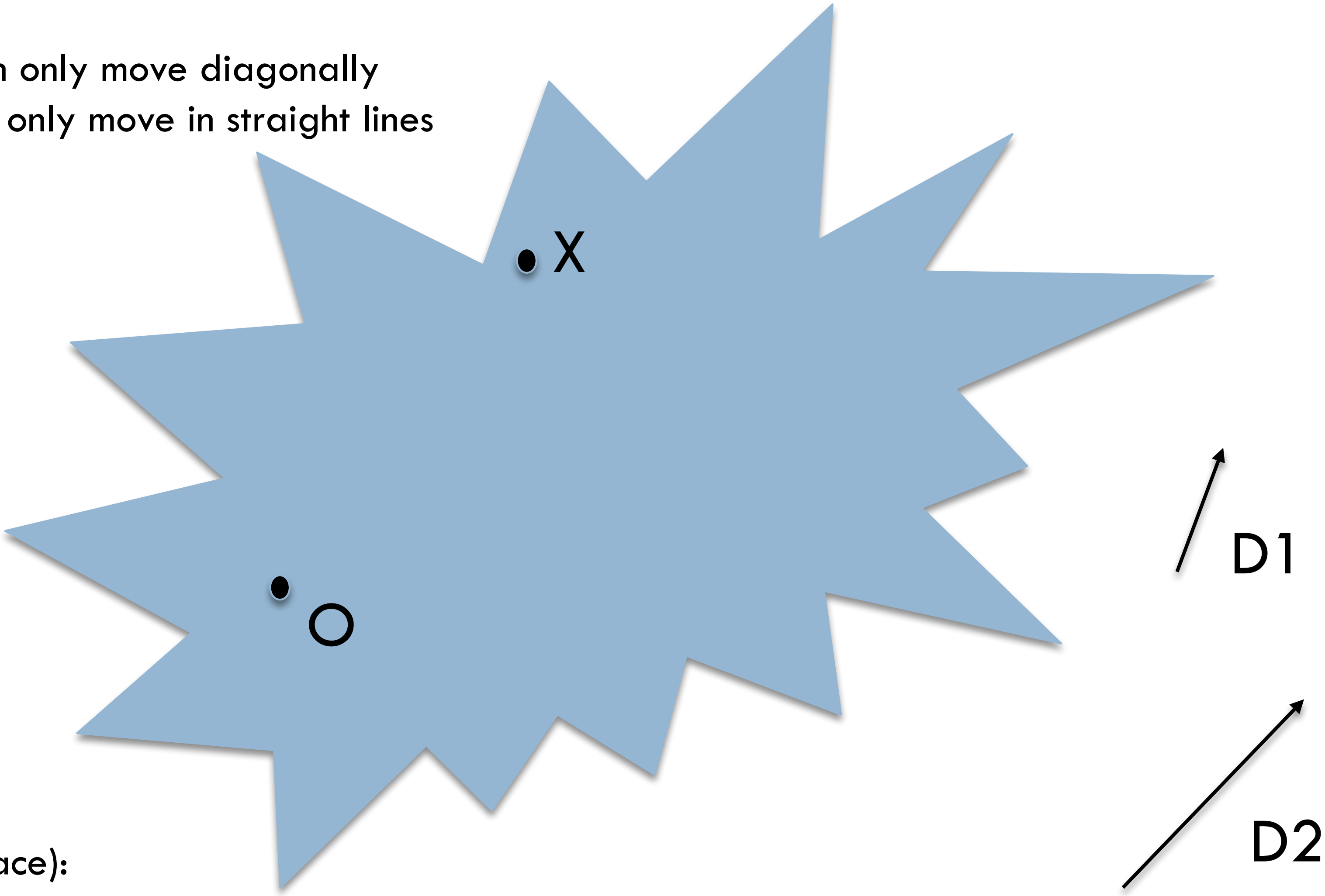
Consider two directions, D1 and D2.



Imagine you live at "O" and you want to get to "X." Can you do it?

Rules (chess):

- Bishop can only move diagonally
- Rooks can only move in straight lines



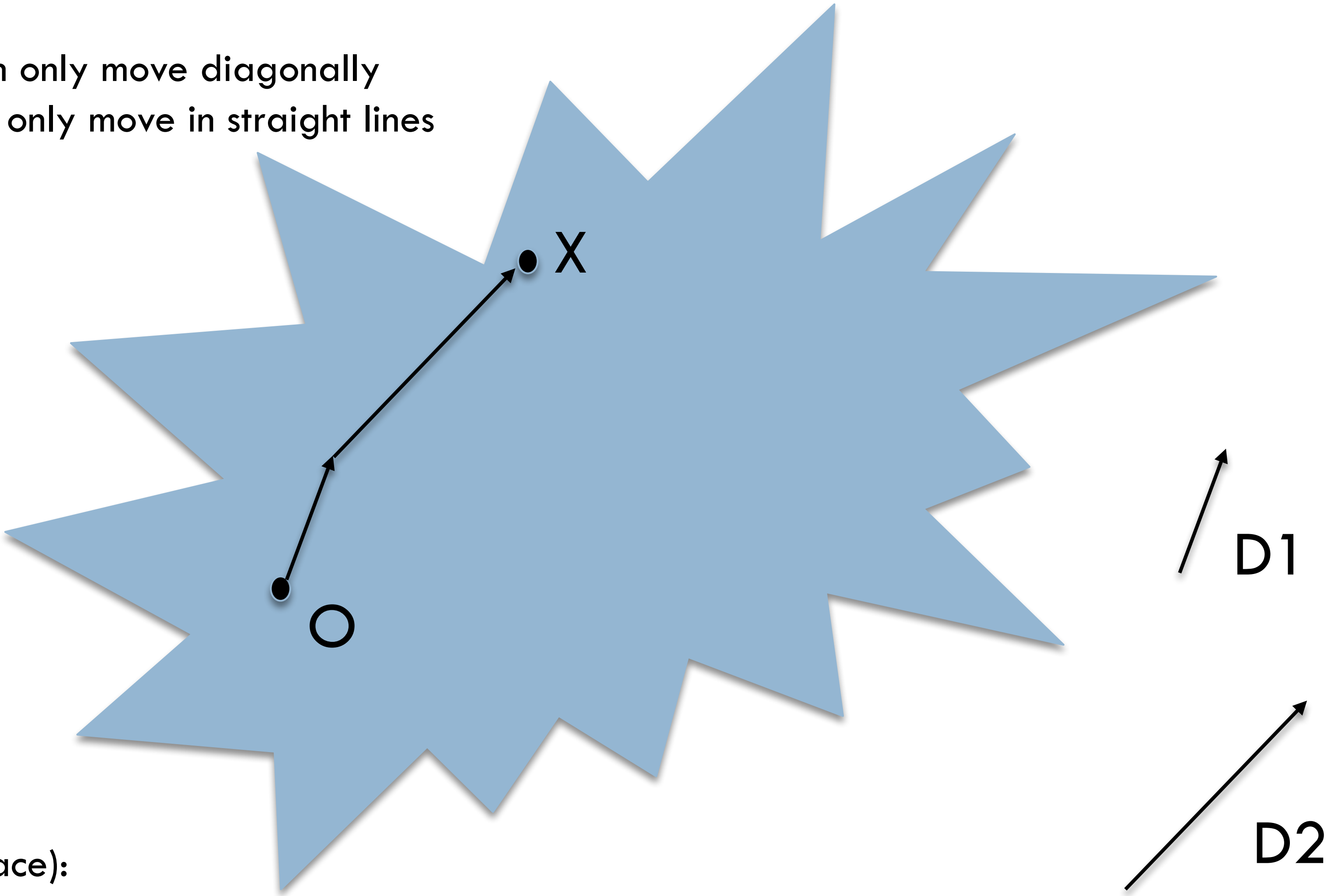
Rules (this space):

- You can only move in direction D1 or D2

Imagine you live at "O" and you want to get to "X." Can you do it?

Rules (chess):

- Bishop can only move diagonally
- Rooks can only move in straight lines



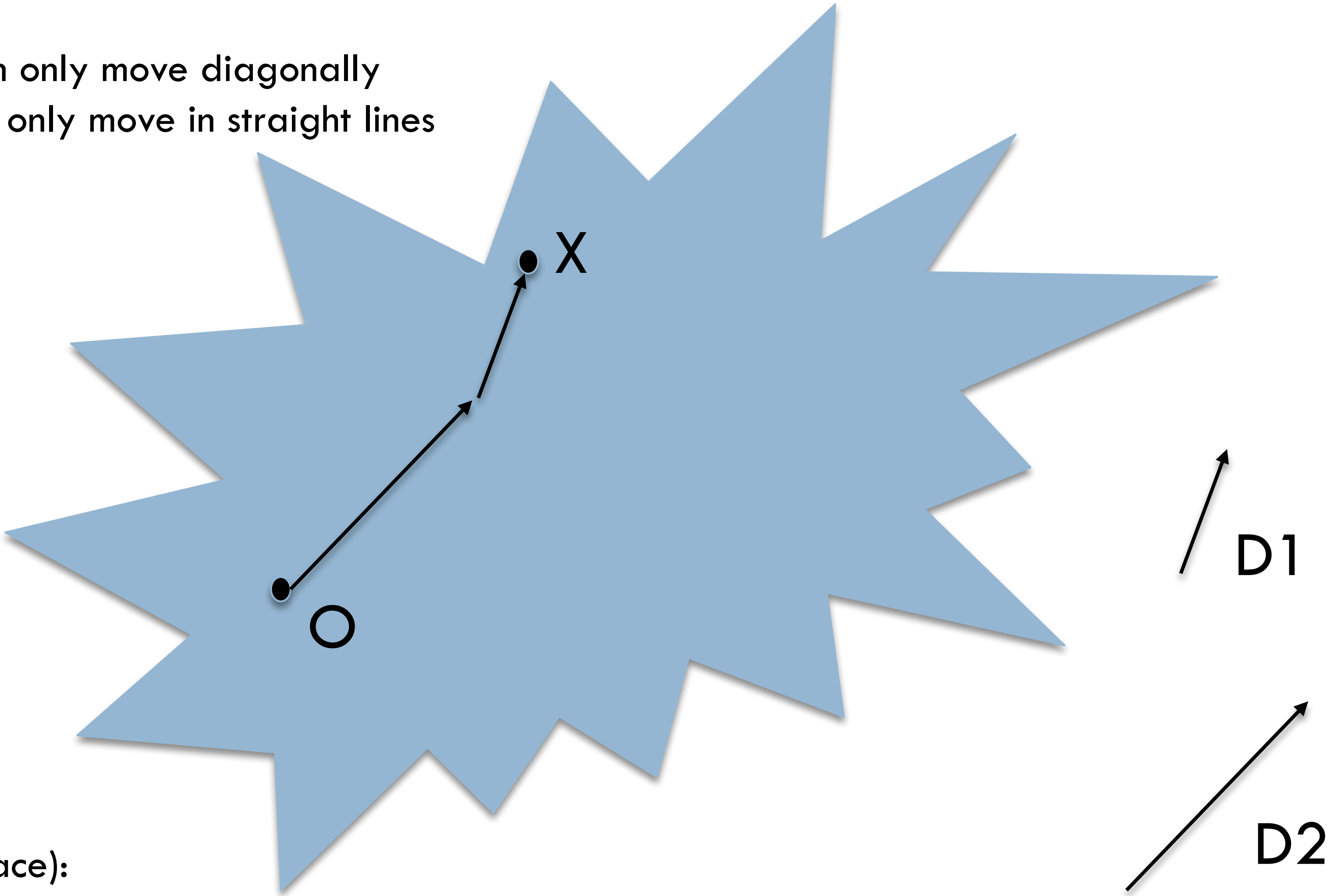
Rules (this space):

- You can only move in direction D1 or D2

Imagine you live at "O" and you want to get to "X." Can you do it?

Rules (chess):

- Bishop can only move diagonally
- Rooks can only move in straight lines



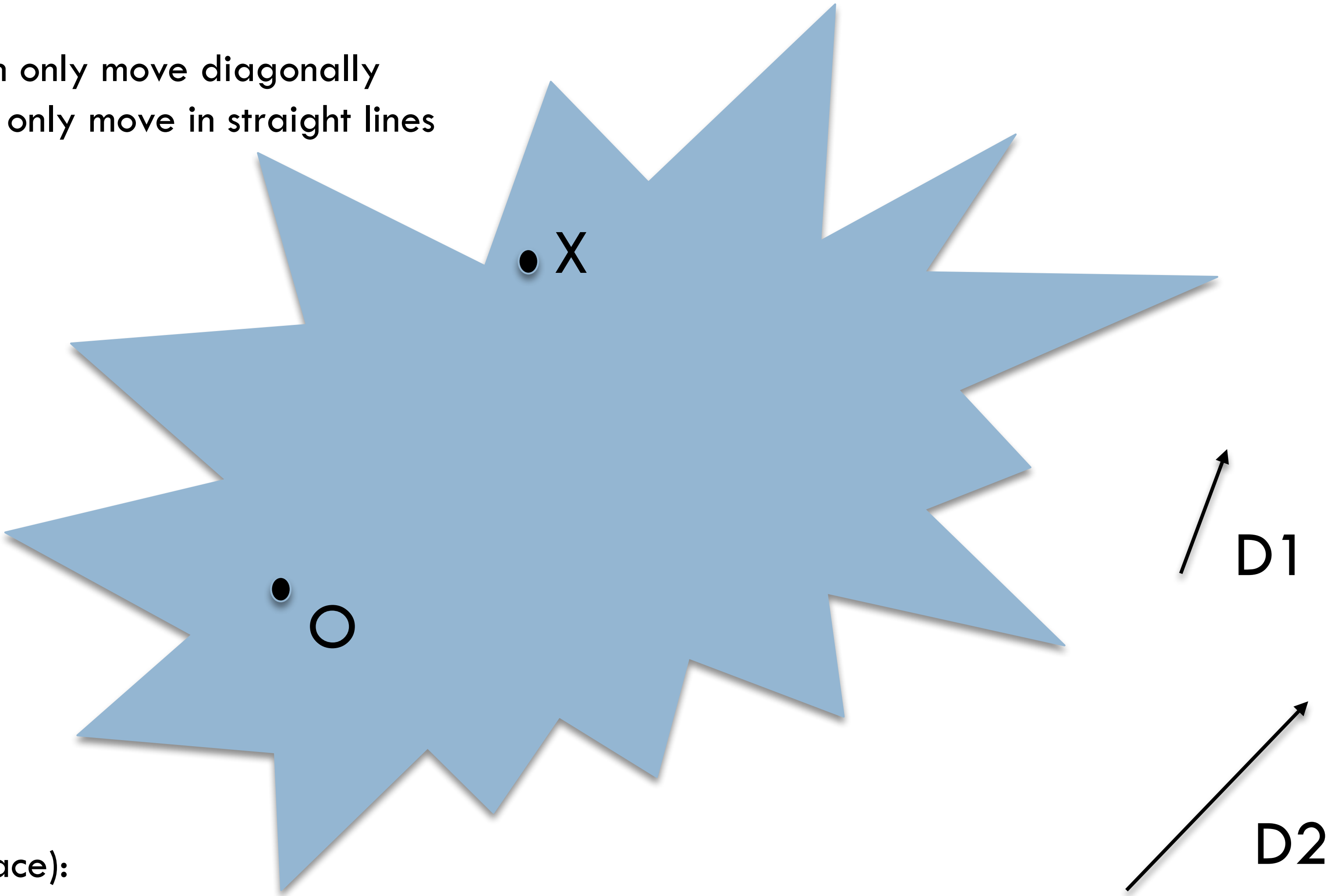
Rules (this space):

- You can only move in direction D1 or D2

Imagine you live at "O" and you want to get to "X." Can you do it?

Rules (chess):

- Bishop can only move diagonally
- Rooks can only move in straight lines

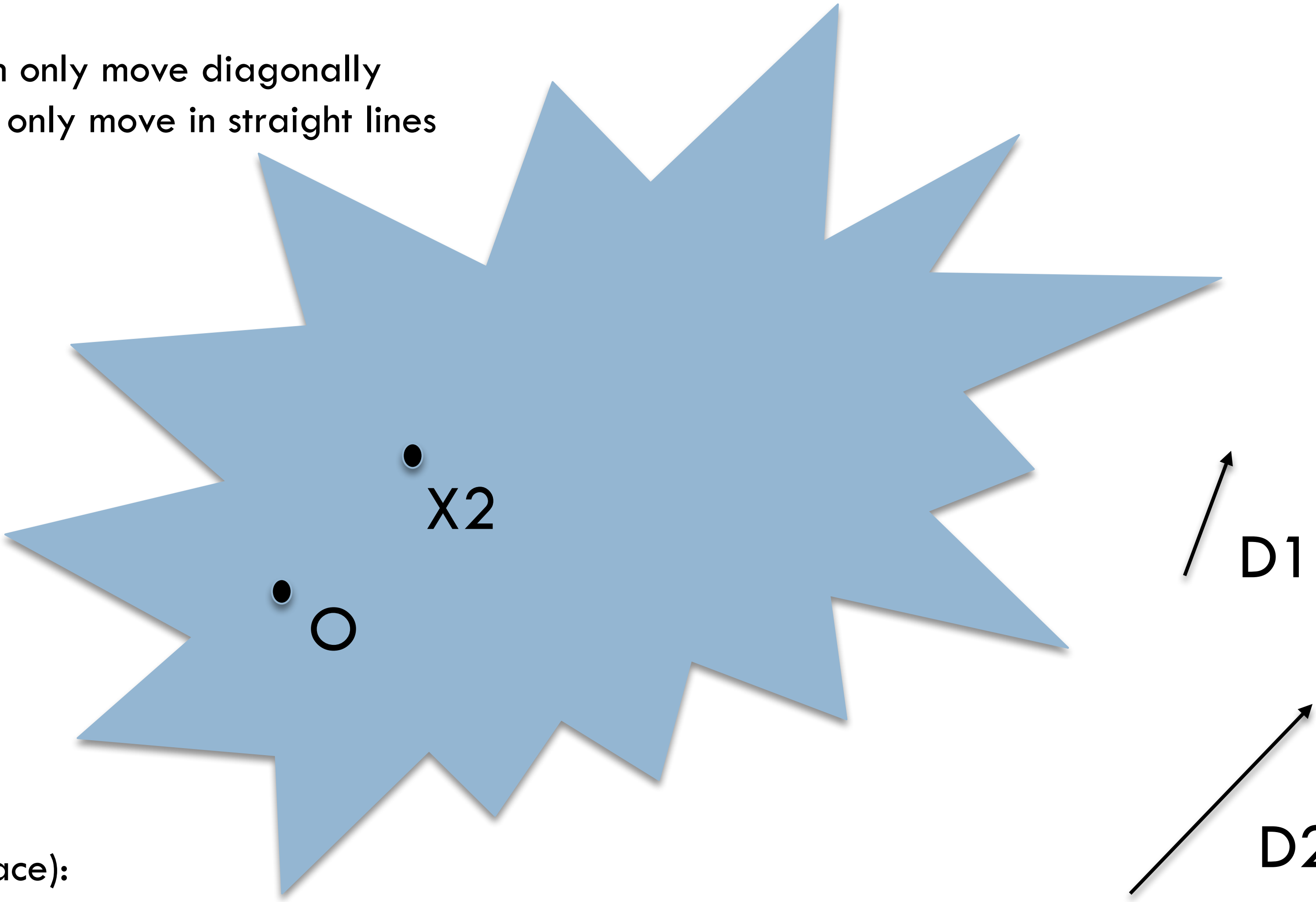


Rules (this space):

- You can only move in direction D1 or D2

Imagine you live at "O" and you want to get to "X2." Can you do it?

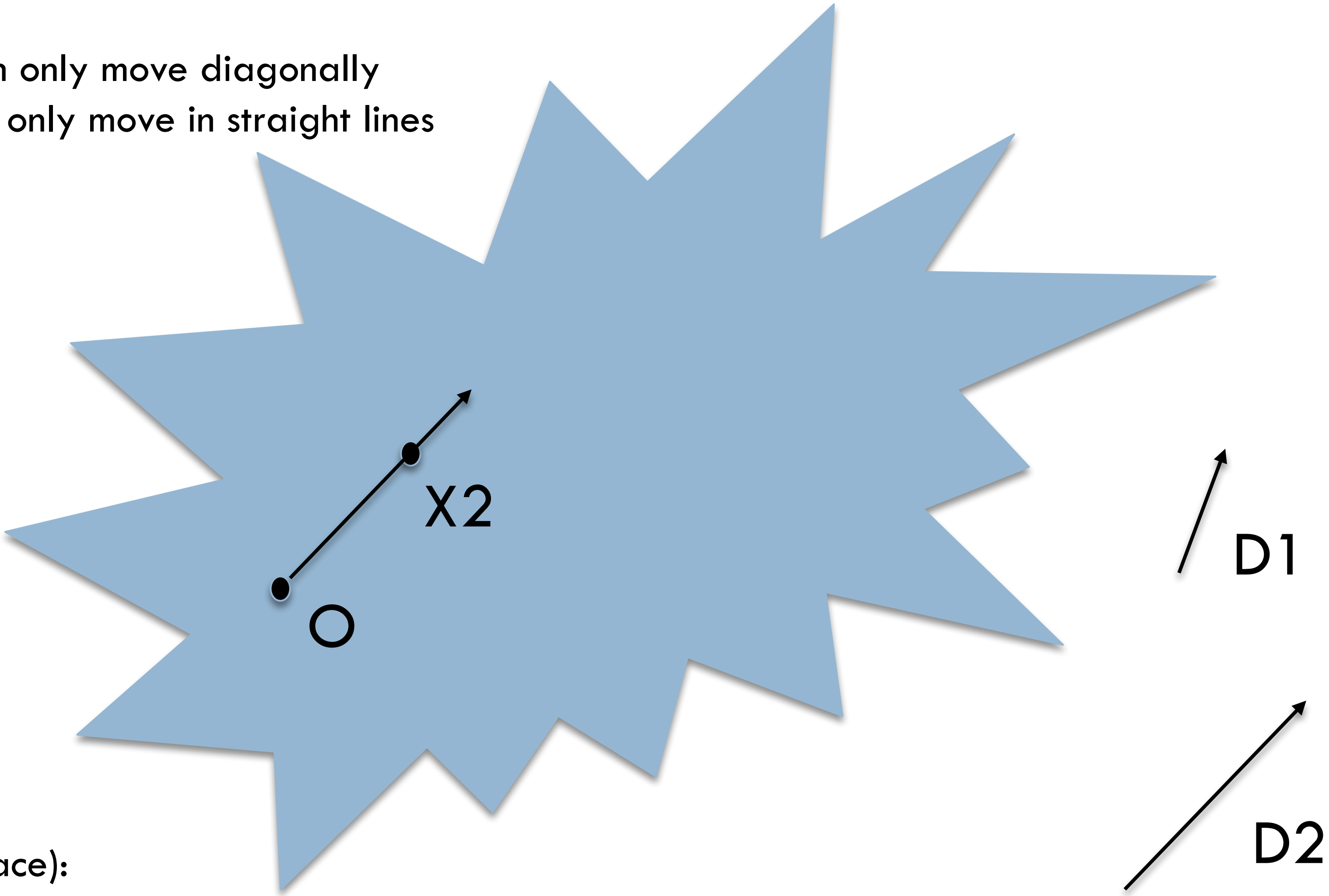
Rules (chess):
- Bishop can only move diagonally
- Rooks can only move in straight lines



Rules (this space):
- You can only move in direction D1 or D2

Imagine you live at "O" and you want to get to "X2." Can you do it?

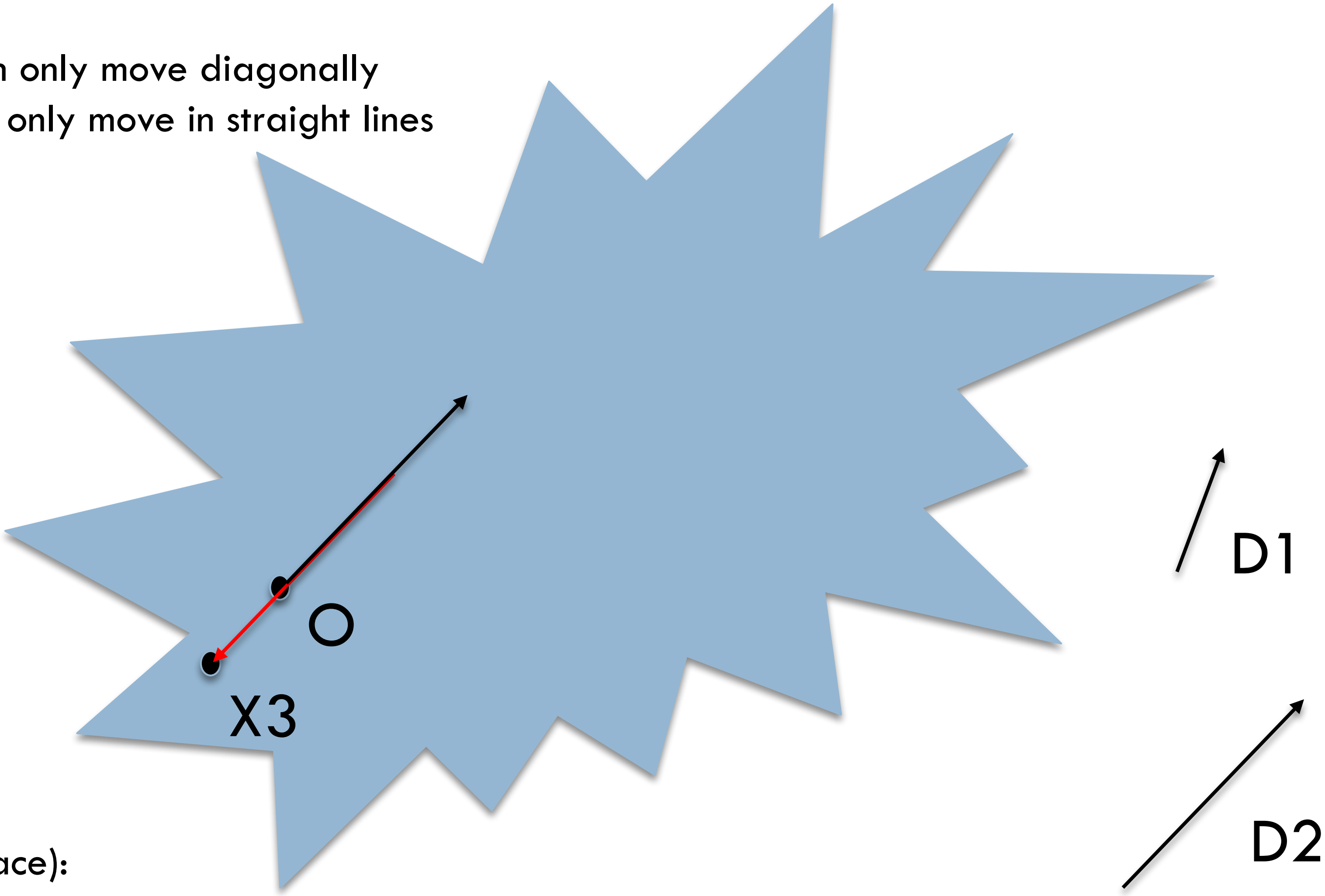
Rules (chess):
- Bishop can only move diagonally
- Rooks can only move in straight lines



Rules (this space):
- You can only move in direction D1 or D2

Imagine you live at "O" and you want to get to "X3." Can you do it?

Rules (chess):
- Bishop can only move diagonally
- Rooks can only move in straight lines

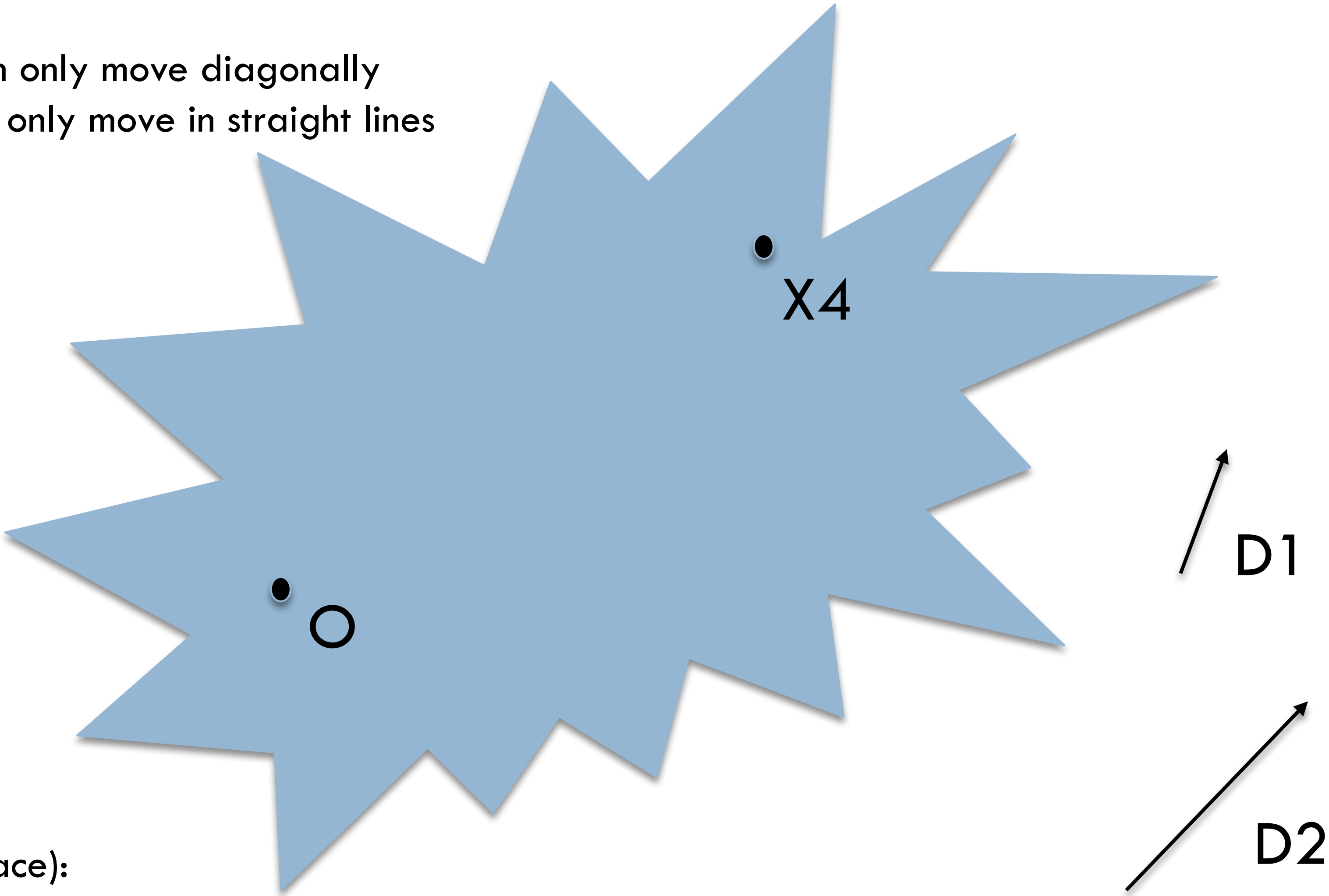


Rules (this space):
- You can only move in direction D1 or D2

Imagine you live at "O" and you want to get to "X4." Can you do it?

Rules (chess):

- Bishop can only move diagonally
- Rooks can only move in straight lines



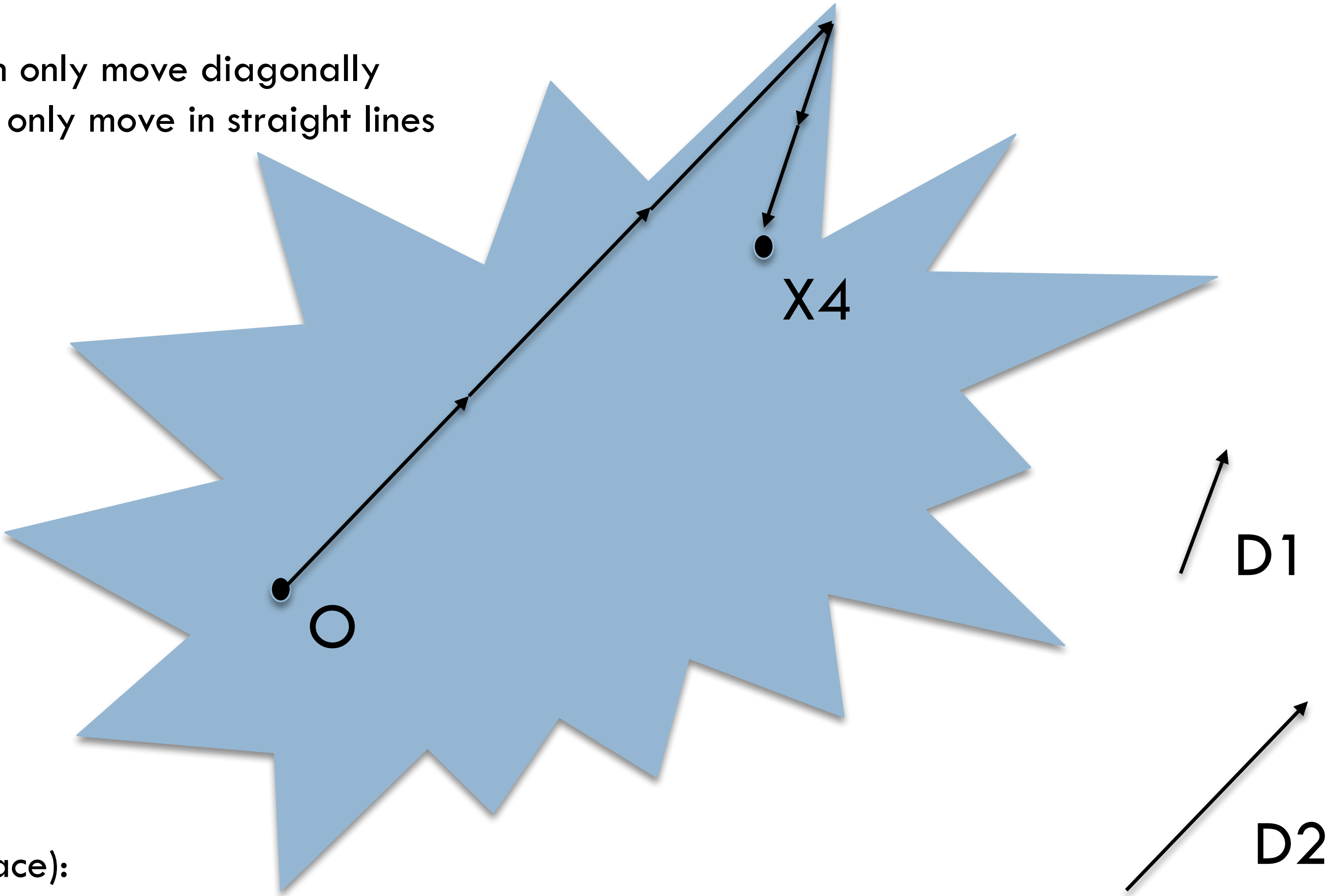
Rules (this space):

- You can only move in direction D1 or D2

Imagine you live at “O” and you want to get to “X4.” Can you do it?

Rules (chess):

- Bishop can only move diagonally
- Rooks can only move in straight lines

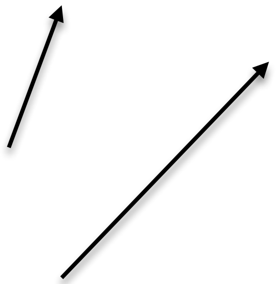


Rules (this space):

- You can only move in direction D1 or D2

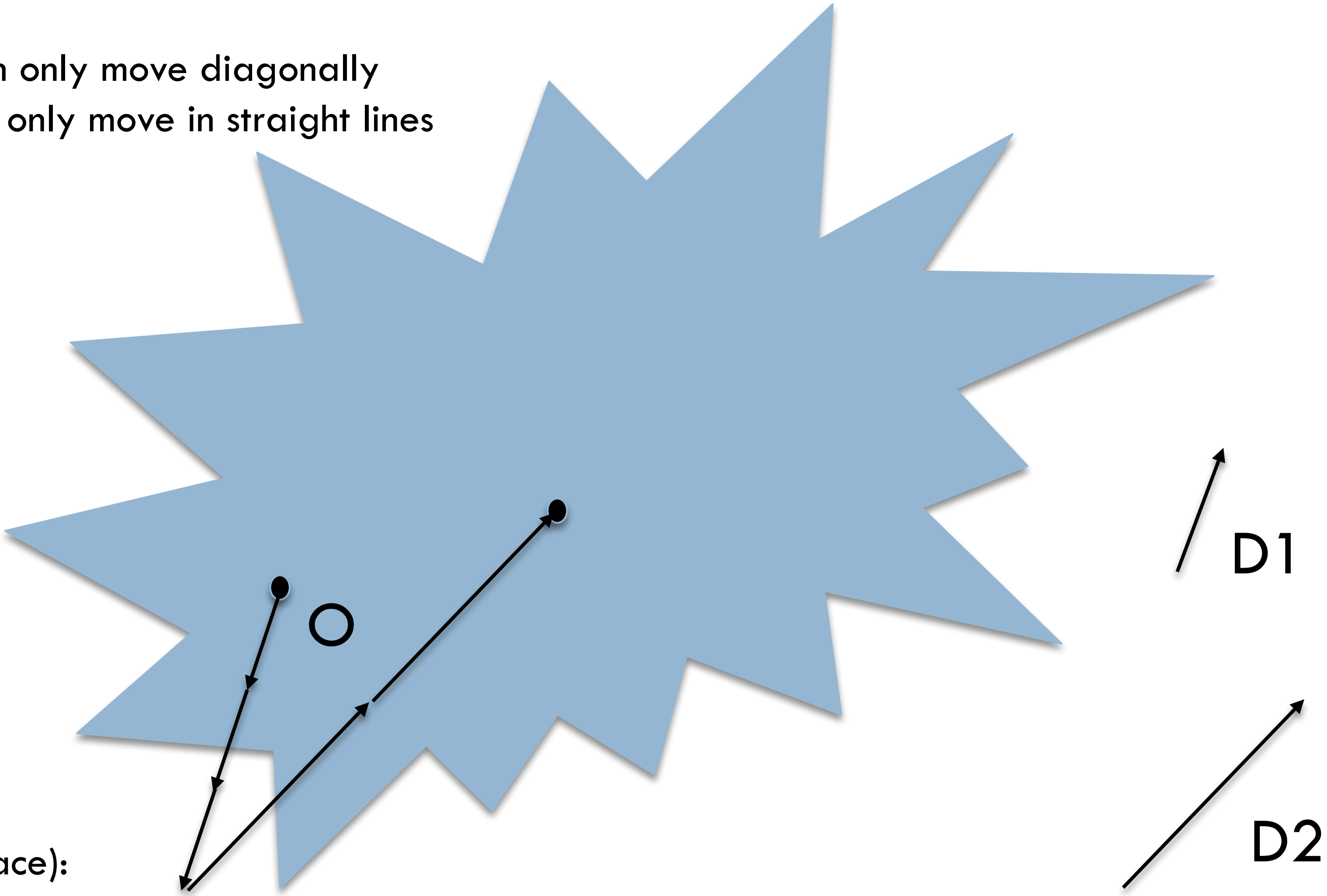
Conventions!

- Let (a, b) mean:
  - The number of steps 'a' in direction D1
  - The number of steps 'b' in direction D2



Where is (-3, 2)?

Rules (chess):
- Bishop can only move diagonally
- Rooks can only move in straight lines



Rules (this space):
- You can only move in direction D1 or D2

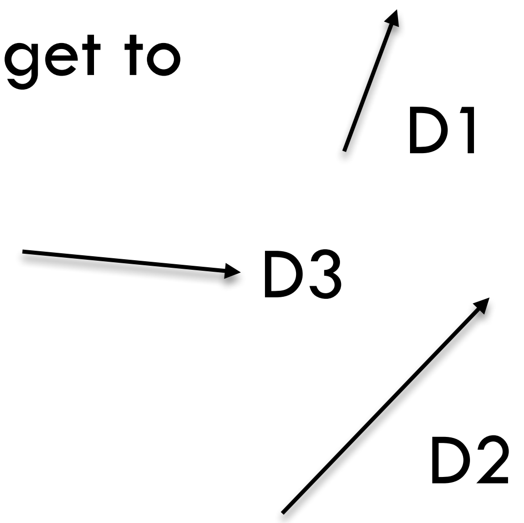
A basis

- Paraphrasing Wikipedia:
- Let B = { D1, D2 } (a set of two vectors, D1 & D2)
- Let S be our Shape
- B is a <u>basis</u> for S if every element of S can be written as a **unique** linear combination of elements of B.
- The coefficients of this linear combination are referred to as <u>components</u> or <u>coordinates</u> on *B* of the vector.
- The elements of a basis are called <u>basis vectors</u>.

Why unique?

- Let (a, b, c) mean:
  - The number of steps ‘a’ in direction D1
  - The number of steps ‘b’ in direction D2
  - The number of steps ‘c’ in direction D3

- Then there is more than one way to get to some point X in S, i.e.,
  - (a1, b1, c1) = X   and
  - (a2, b2, c2) = X



What does it mean to form a basis?

- For any vector v, there are unique coordinates (c1, …, cn) such that

  v = c1*v1 + c2*v2 + … + cn*vn
- Consider some point P.
  - The basis has an origin O
  - There is a vector v such that O+v = P
  - We know we can construct v using a combination of vi’s
  - Therefore we can represent P in our frame using the coordinates (c1, c2, …, cn)

A basis

- Paraphrasing Wikipedia:
- Let B = { D1, D2 } (a set of two vectors, D1 & D2)
- Let S be our Shape
- B is a basis for S if every element of S can be written as a **unique** linear combination of elements of B.
- The coefficients of this linear combination are referred to as components or $\boxed{\text{coordinates}}$ on *B* of the vector.
- The elements of a basis are called basis vectors.

Most common basis

- D1 = X-axis (i.e., (1,0,0)-(0,0,0))
- D2 = Y-axis (i.e., (0,1,0)-(0,0,0))
- D3 = Z-axis (i.e., (0,0,1)-(0,0,0))

- Then the coordinate (2, -3, 5) means
  - 2 units along X-axis
  - -3 units along Y-axis
  - 5 units along Z-axis

But we could have other bases

- Instead of “basis 1” (B1)
  - D1 = X-axis (i.e., (1,0,0)-(0,0,0))
  - D2 = Y-axis (i.e., (0,1,0)-(0,0,0))
  - D3 = Z-axis (i.e., (0,0,1)-(0,0,0))
- Use “basis 2” (B2)
  - D1 = Y-axis (i.e., (0,1,0)-(0,0,0))
  - D2 = X-axis (i.e., (1,0,0)-(0,0,0))
  - D3 = Z-axis (i.e., (0,0,1)-(0,0,0))
- Then (a,b,c) in B1 is the same as (b,a,c) in B2

Last vocab term for a few slides: frame

- Frame:
  - A way to place a coordinate system into a specific location in a space
  - Basis + reference coordinate (“the origin”)
- Cartesian example: (3,4,6)
  - It is assumed that we are speaking in reference to the origin location (0,0,0).

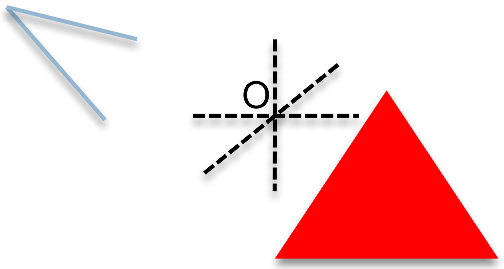
Example of Frames

- Frame F = (v1, v2, O)
  - v1 = (0, -1)
  - v2 = (1, 0)
  - O = (3, 4)
- What are F's coordinates for the point (6, 6)?

Example of Frames

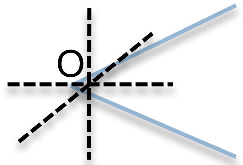
- Frame F = (v1, v2, O)
  - v1 = (0, -1)
  - v2 = (1, 0)
  - O = (3, 4)
- What are F's coordinates for the point (6, 6)?
- Answer: (-2, 3)

Each box is a frame, and each arrow converts to the next frame



World space:

Triangles in native Cartesian coordinates
Camera located anywhere



Camera space:

Camera located at origin, looking down -Z
Triangle coordinates relative to camera frame

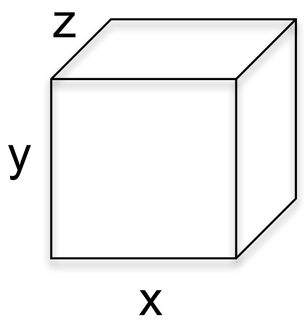
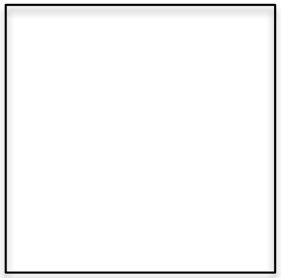


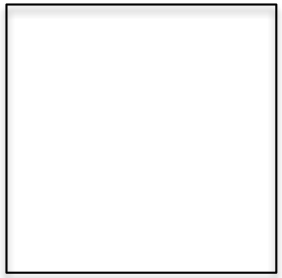
Image space:

All viewable objects within
-1 <= x,y,z <= +1



Screen space:

All viewable objects within
-1 <= x, y <= +1



Device space:

All viewable objects within
0<=x<=width, 0 <=y<=height

Context

- Models stored in “world space” frame
  - Pick an origin, store all points relative to that origin
- We have been rasterizing in “device space” frame
- Our goal: transform from world space to device space
- We will do this using matrix multiplications
  - Multiply point by matrix to convert coordinates from one frame into coordinates in another frame

But wait! There’s more…

- And matrices also useful for more than frame-to-frame conversions.

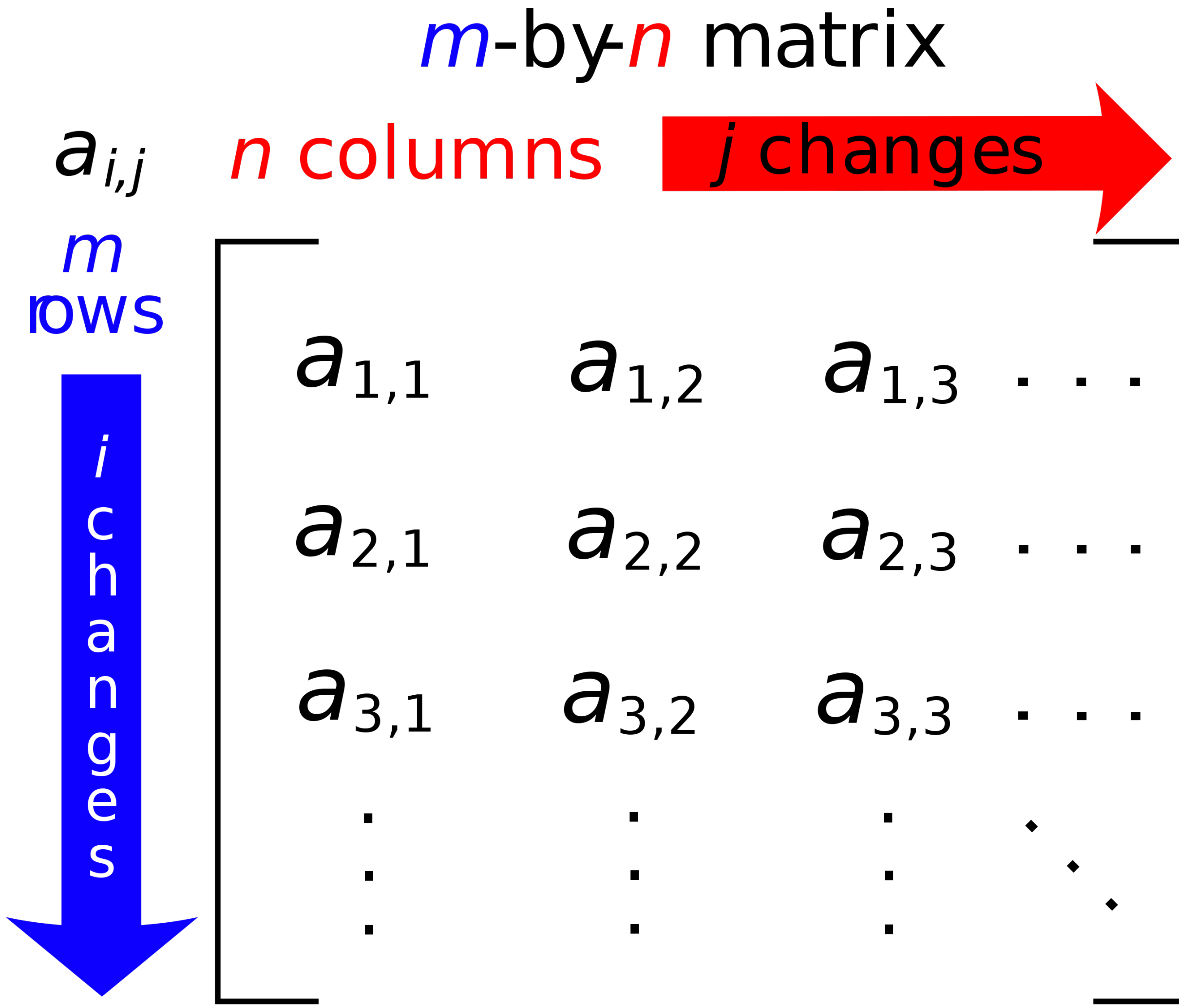
- So let’s get comfy with matrices (next time).

STOP HERE

Matrix

- Defined: a rectangular array of numbers (usually) arranged in rows and columns
- Example
  - 2D matrix
  - “two by three” (two rows, three columns)
    - [3   4   8]
    - [-1 9.2 12]

Matrix: wikipedia picture


m-by-n matrix
$a_{i,j}$
n columns
j changes
m rows
i changes
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ . . .
$a_{2,1}$ $a_{2,2}$ $a_{2,3}$ . . .
$a_{3,1}$ $a_{3,2}$ $a_{3,3}$ . . .
⋮ ⋮ ⋮ ⋱

Matrix

- What do you do with matrices?
- Lots of things
  - Transpose, invert, add, subtract
- But most of all: multiply!

Multiplying two 2x2 matrices

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} a*e+b*g & a*f+b*h \\ c*e+d*g & c*f+d*h \end{pmatrix}$$

Multiplying two 2x2 matrices

$$(a \quad b) \; X \; \begin{array}{cc}(e & f)\\ (g & h)\end{array} = (a*e+b*g \qquad a*f+b*h)$$

One usage for matrices:

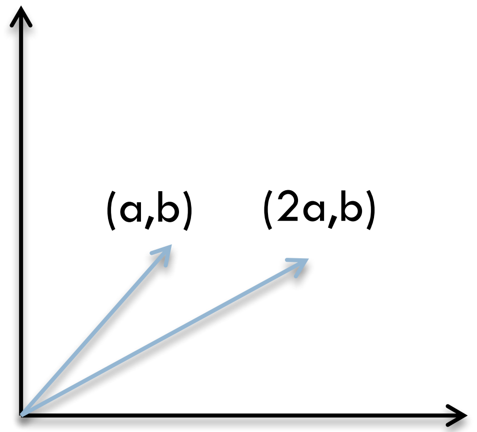
Let (a, b) be the coordinates of a point

Then the 2x2 matrix can transform (a,b) to a new location – (a*e+b*g, a*f+b*h)

Identity Matrix

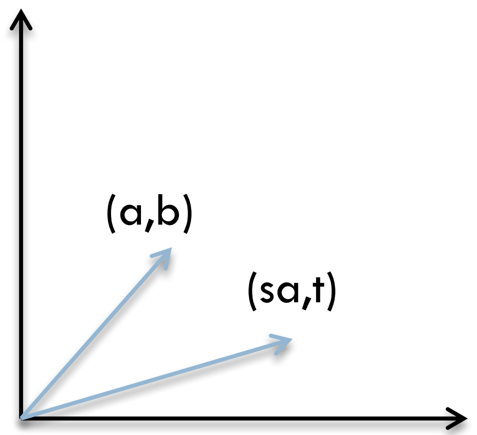
$$\begin{pmatrix} a & b \end{pmatrix} \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} a & b \end{pmatrix}$$

$$(a \quad b) \times \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} = (2a \quad b)$$



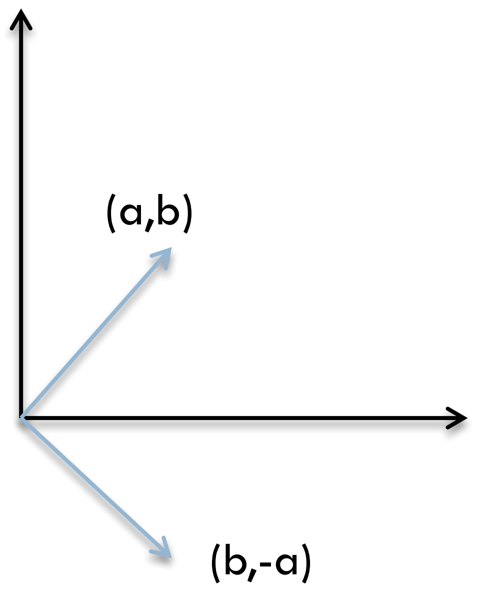
Scale in X, not in Y

$$\begin{pmatrix} a & b \end{pmatrix} \times \begin{pmatrix} s & 0 \\ 0 & t \end{pmatrix} = \begin{pmatrix} sa & tb \end{pmatrix}$$



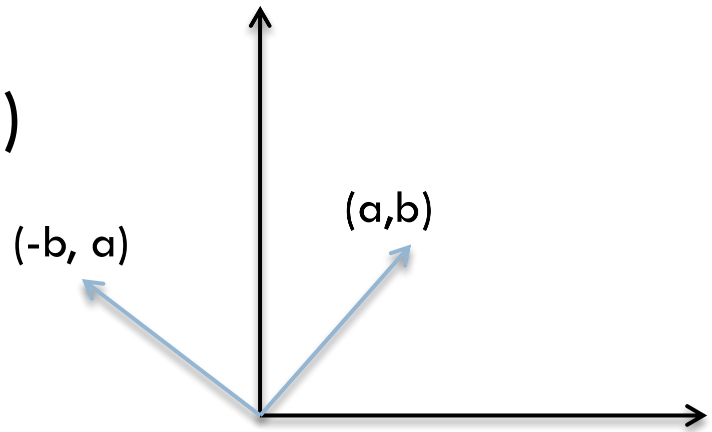
Scale in both dimensions

$$\begin{pmatrix} a & b \end{pmatrix} \times \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} b & -a \end{pmatrix}$$



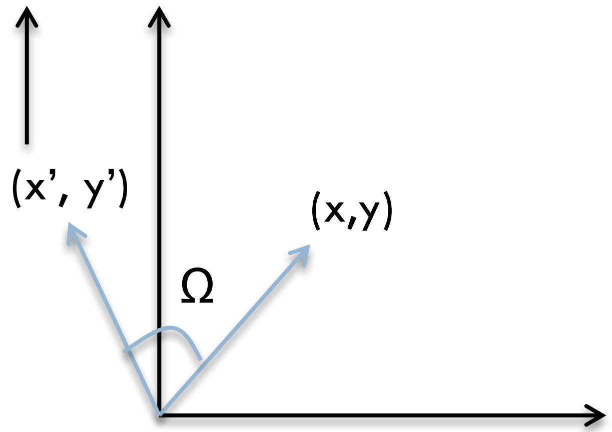
Rotate 90 degrees counter-clockwise

$$\begin{pmatrix} a & b \end{pmatrix} \times \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} = \begin{pmatrix} -b & a \end{pmatrix}$$



Rotate 90 degrees counter-clockwise

$$(a \quad b) \times \begin{pmatrix} \cos(\Omega) & -\sin(\Omega) \\ \sin(\Omega) & \cos(\Omega) \end{pmatrix} = \begin{matrix} (\cos(\Omega)*a + \sin(\Omega)*b, \\ -\sin(\Omega)*a + \cos(\Omega)*b) \end{matrix}$$



Rotate "Ω" degrees counter-clockwise

Combining transformations

- How do we rotate by 90 degrees clockwise and then scale X by 2?
  - Answer: multiply by matrix that multiplies by 90 degrees clockwise, then multiple by matrix that scales X by 2.
  - But can we do this efficiently?

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \text{X} \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 2 & 0 \end{pmatrix}$$

Combining transformations

- How do we scale X by 2 and then rotate by 90 degrees clockwise?
  - Answer: multiply by matrix that scales X by 2, then multiply by matrix that rotates 90 degrees clockwise.

$$\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \text{X} \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -2 \\ 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \text{X} \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 2 & 0 \end{pmatrix}$$

Rotate then scale
Order matters!!

Translations

- Translation is harder:

$$\begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} a+c \\ b+d \end{pmatrix}$$

But this doesn’t fit our nice matrix multiply model…
What to do??

Homogeneous Coordinates

$$\begin{pmatrix} x & y & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} x & y & 1 \end{pmatrix}$$

Add an extra dimension.
A math trick … don't overthink it.

Homogeneous Coordinates

$$\begin{pmatrix} x & y & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{pmatrix} = \begin{pmatrix} x+dx & y+dy & 1 \end{pmatrix}$$

Translation

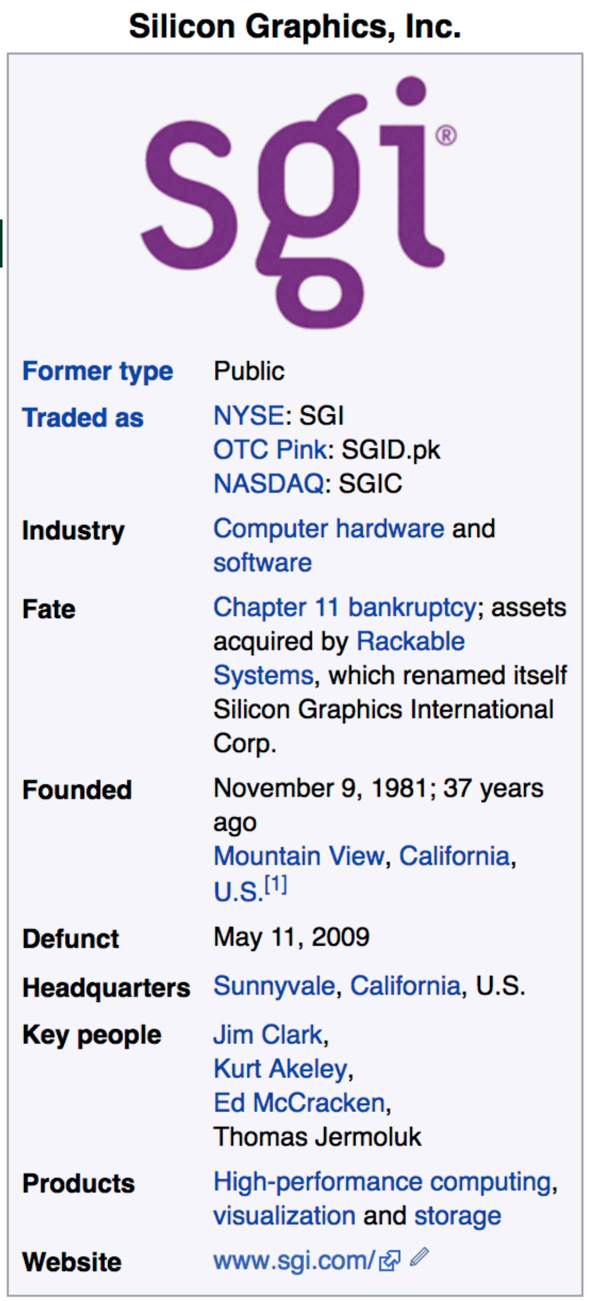
We can now fit translation into
our matrix multiplication system.

Graphics

- Two really important operations:
  - Transform from one frame to another
  - Transform geometry (rotate, translate, etc)

- Both can be done with matrix operations
- In both cases, need homogeneous coordinates

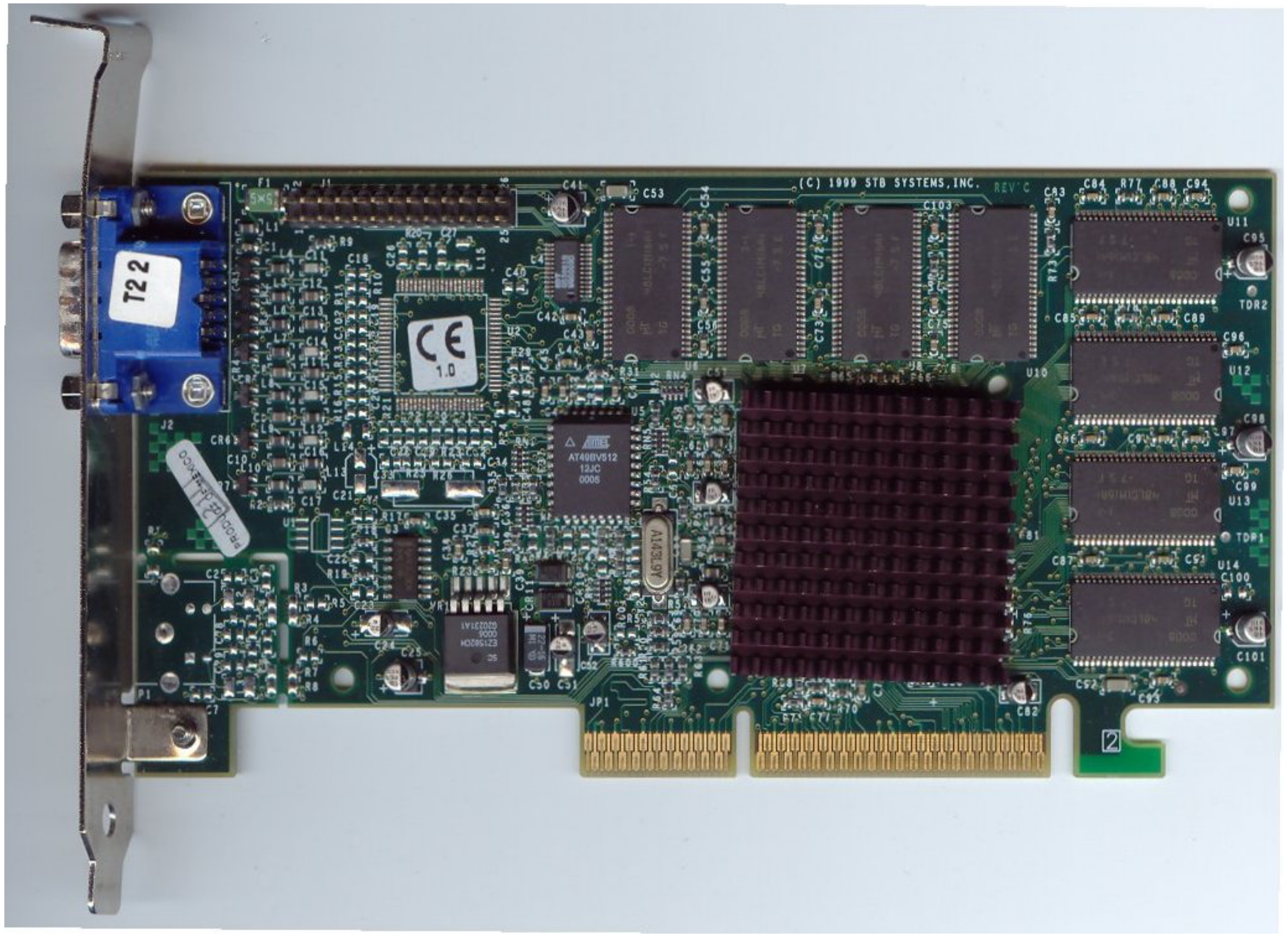
- Much of graphics is accomplished via 4x4 matrices
  - And: you can compose the matrices and do bunches of things at once (EFFICIENCY)



Silicon Graphics, Inc.

Former type	Public
Traded as	NYSE: SGI OTC Pink: SGID.pk NASDAQ: SGIC
Industry	Computer hardware and software
Fate	Chapter 11 bankruptcy; assets acquired by Rackable Systems, which renamed itself Silicon Graphics International Corp.
Founded	November 9, 1981; 37 years ago Mountain View, California, U.S.[1]
Defunct	May 11, 2009
Headquarters	Sunnyvale, California, U.S.
Key people	Jim Clark, Kurt Akeley, Ed McCracken, Thomas Jermoluk
Products	High-performance computing, visualization and storage
Website	www.sgi.com/

3dfx Voodoo (source: wikipedia)



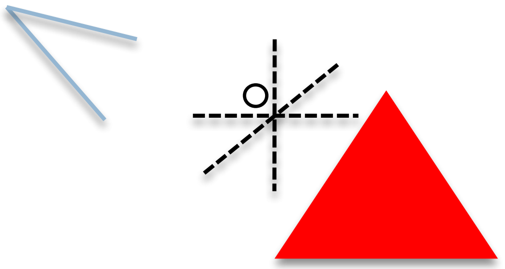
Early GPUs

- Special hardware to do 4x4 matrix operations
- A lot of them (in parallel)

GPUs now

- Many, many, many cores
- Each code less powerful than typical CPU core

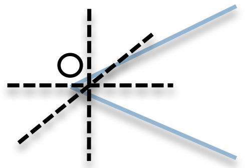
Our goal



World space:

Triangles in native Cartesian coordinates

Camera located anywhere



Camera space:

Camera located at origin, looking down -Z

Triangle coordinates relative to camera frame

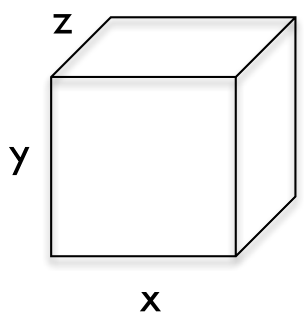
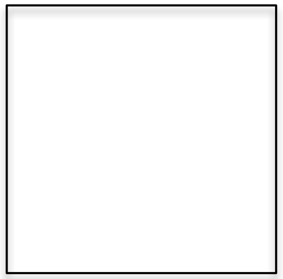


Image space:

All viewable objects within

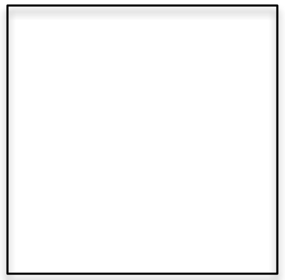
-1 <= x,y,z <= +1



Screen space:

All viewable objects within

-1 <= x, y <= +1



Device space:

All viewable objects within

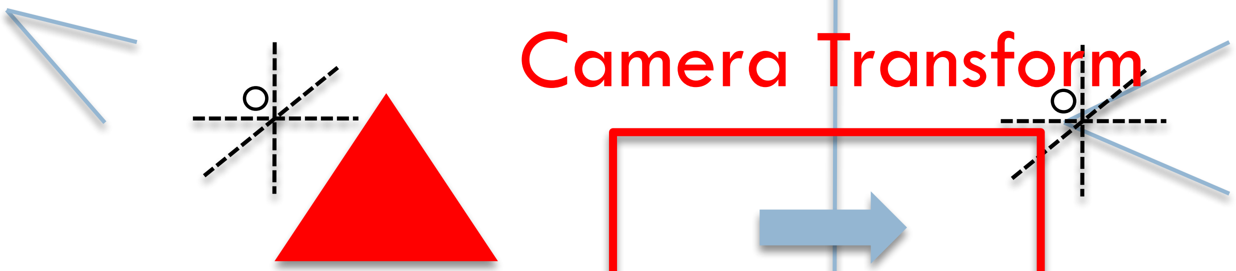
0<=x<=width, 0

<=y<=height

World Space

- World Space is the space defined by the user’s coordinate system.
- This space contains the portion of the scene that is transformed into image space by the camera transform.
- Many of the spaces have “bounds”, meaning limits on where the space is valid
- With world space 2 options:
  - No bounds
  - User specifies the bound

Our goal



World space:

Triangles in native Cartesian coordinates

Camera located anywhere

Camera space:

Camera located at origin, looking down -Z

Triangle coordinates relative to camera frame

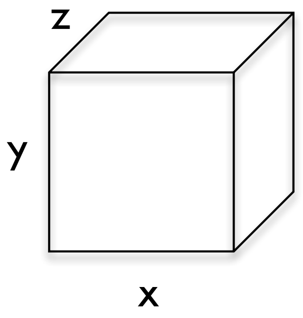
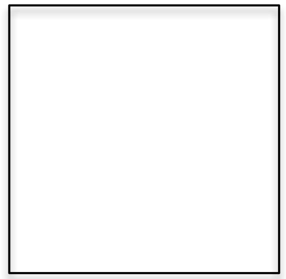


Image space:

All viewable objects within

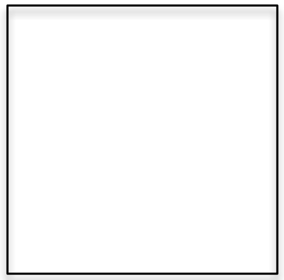
-1 <= x,y,z <= +1



Screen space:

All viewable objects within

-1 <= x, y <= +1



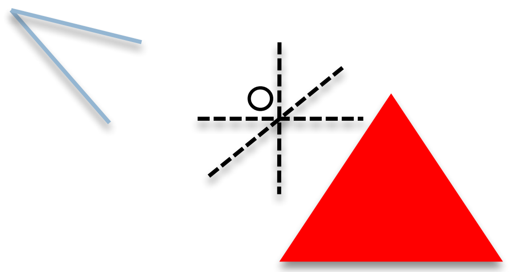
Device space:

All viewable objects within

0<=x<=width, 0

<=y<=height

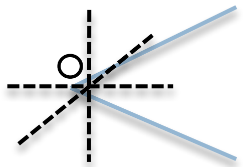
Our goal



World space:

Triangles in native Cartesian coordinates

Camera located anywhere



Camera space:

Camera located at origin, looking down -Z

Triangle coordinates relative to camera frame

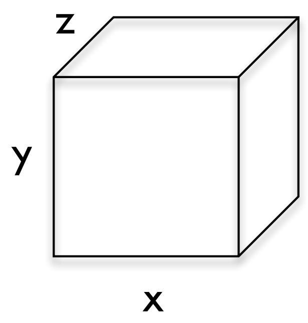
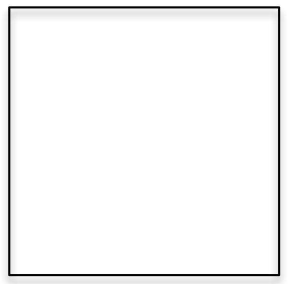


Image space:

All viewable objects within

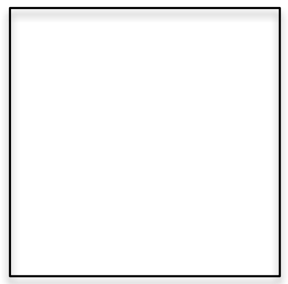
-1 <= x,y,z <= +1



Screen space:

All viewable objects within

-1 <= x, y <= +1



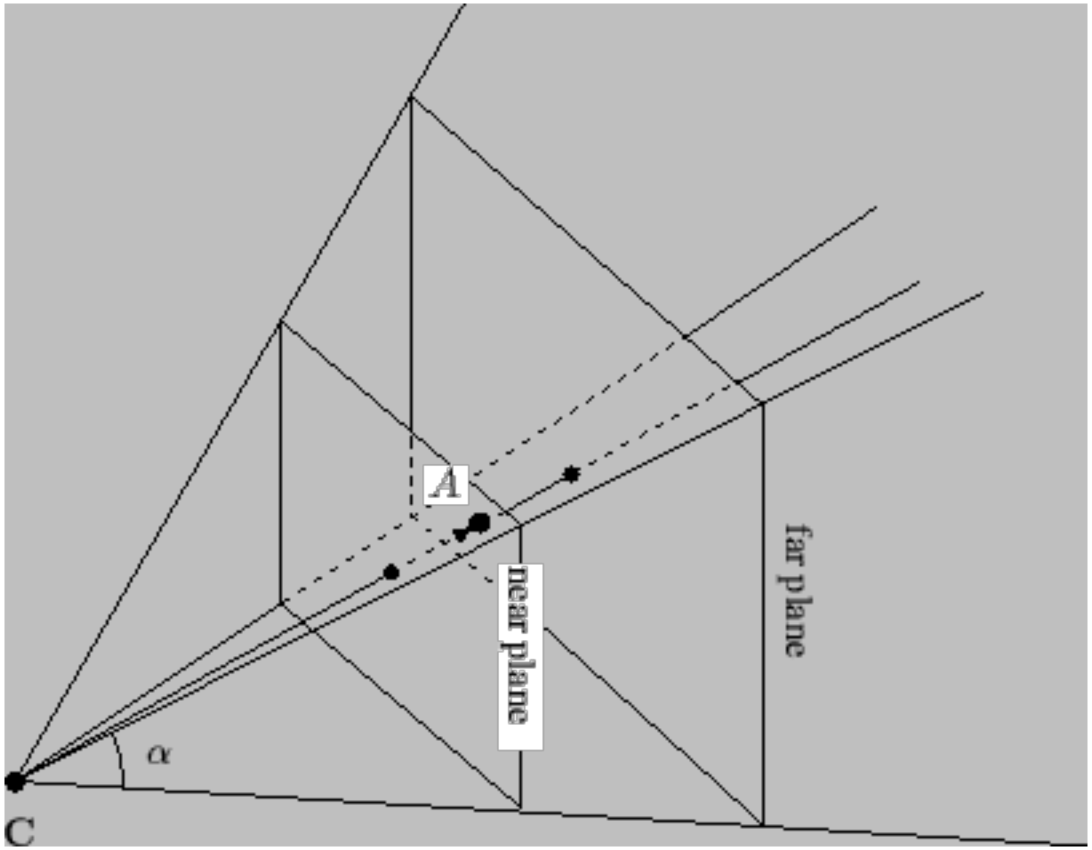
Device space:

All viewable objects within

0<=x<=width, 0

<=y<=height

How do we specify a camera?

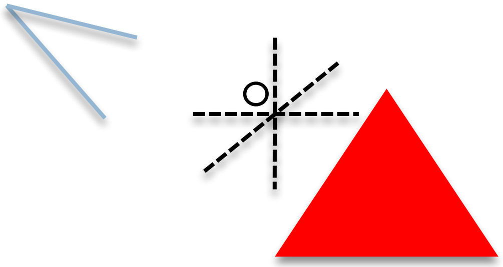


The "viewing pyramid" or "view frustum".

Frustum: In geometry, a frustum (plural: frusta or frustums) is the portion of a solid (normally a cone or pyramid) that lies between two parallel planes cutting it.

```
class Camera
{
  public:
    double          near, far;
    double          angle;
    double          position[3];
    double          focus[3];
    double          up[3];
};
```

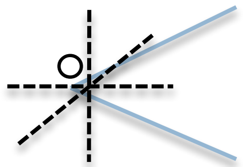
Our goal



World space:

Triangles in native Cartesian coordinates

Camera located anywhere



Camera space:

Camera located at origin, looking down -Z

Triangle coordinates relative to camera frame

View Transform

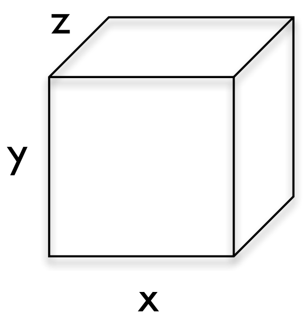
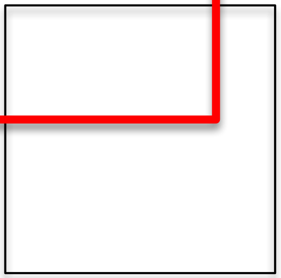


Image space:

All viewable objects within

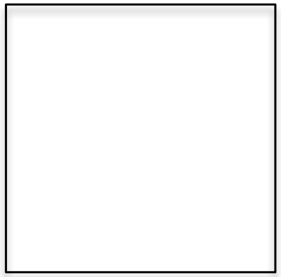
-1 <= x,y,z <= +1



Screen space:

All viewable objects within

-1 <= x, y <= +1

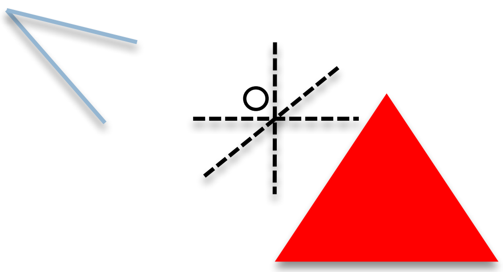


Device space:

All viewable objects within

0<=x<=width, 0 <=y<=height

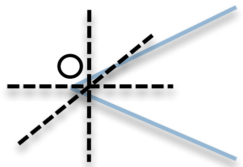
Our goal



World space:

Triangles in native Cartesian coordinates

Camera located anywhere



Camera space:

Camera located at origin, looking down -Z

Triangle coordinates relative to camera frame

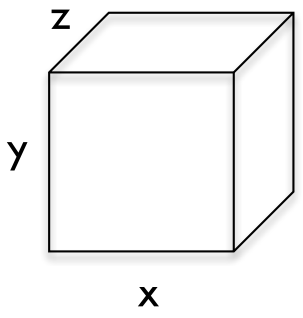
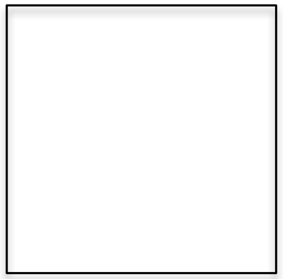


Image space:

All viewable objects within

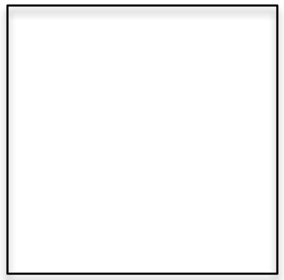
-1 <= x,y,z <= +1



Screen space:

All viewable objects within

-1 <= x, y <= +1



Device space:

All viewable objects within

0<=x<=width, 0

<=y<=height

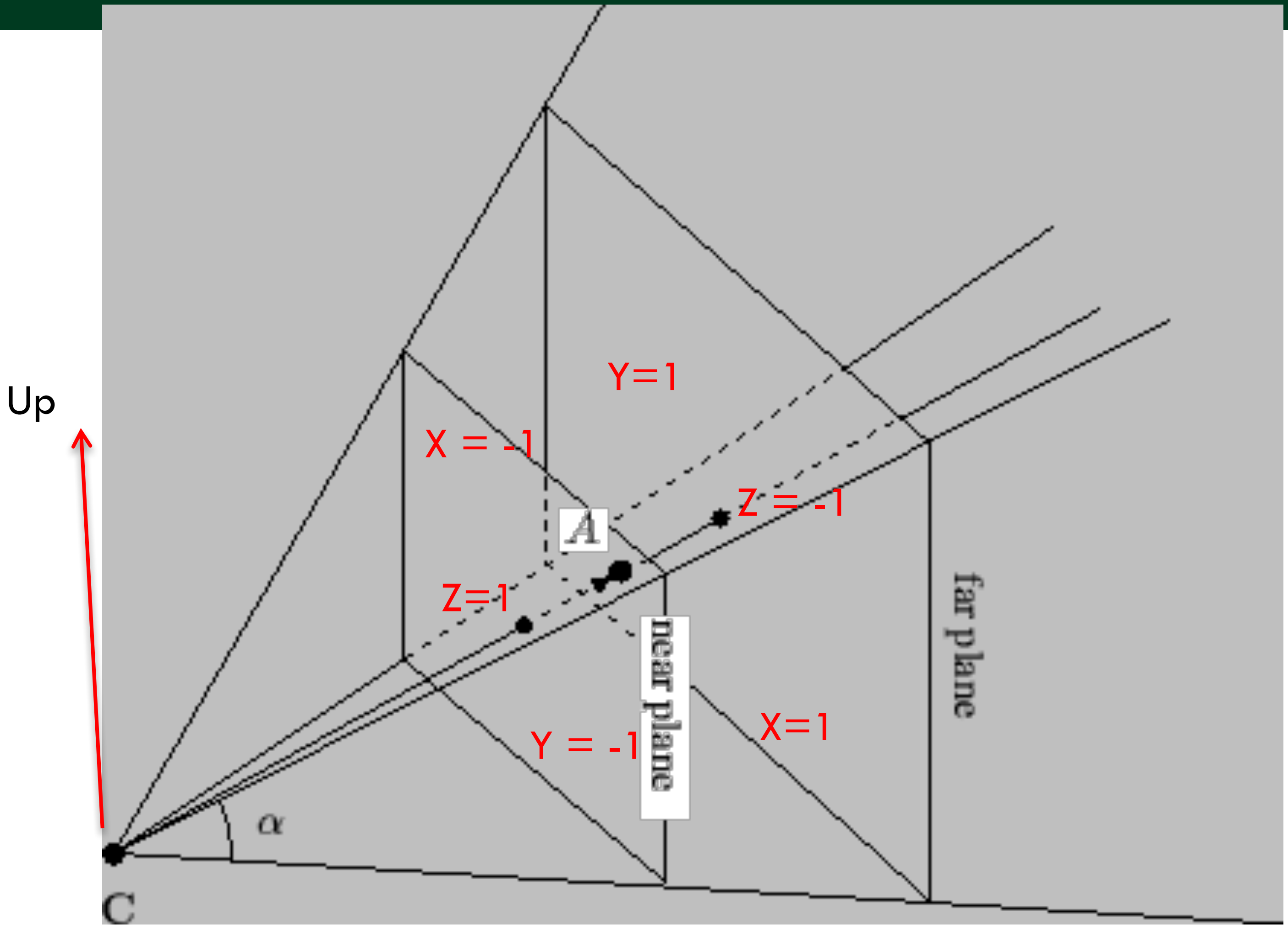
Image Space

- Image Space is the three-dimensional coordinate system that contains screen space.
- It is the space where the camera transformation directs its output.
- The bounds of *Image Space* are 3-dimensional cube.

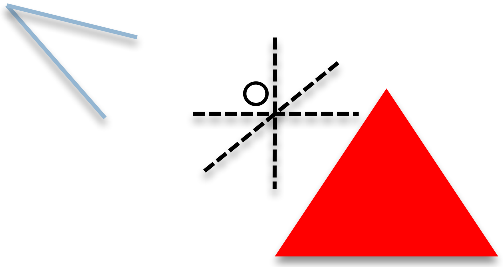
$$\{(x,y,z) : -1 \leq x \leq 1, -1 \leq y \leq 1, -1 \leq z \leq 1\}$$

(or $-1 \leq z \leq 0$)

Image Space Diagram



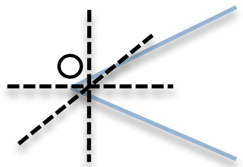
Our goal



World space:

Triangles in native Cartesian coordinates

Camera located anywhere



Camera space:

Camera located at origin, looking down -Z

Triangle coordinates relative to camera frame

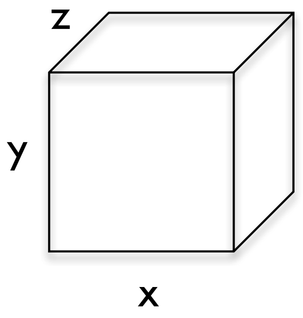
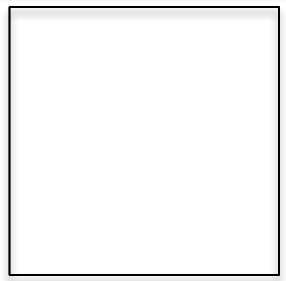


Image space:

All viewable objects within

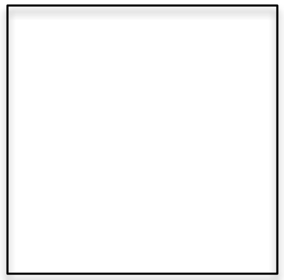
-1 <= x,y,z <= +1



Screen space:

All viewable objects within

-1 <= x, y <= +1



Device space:

All viewable objects within

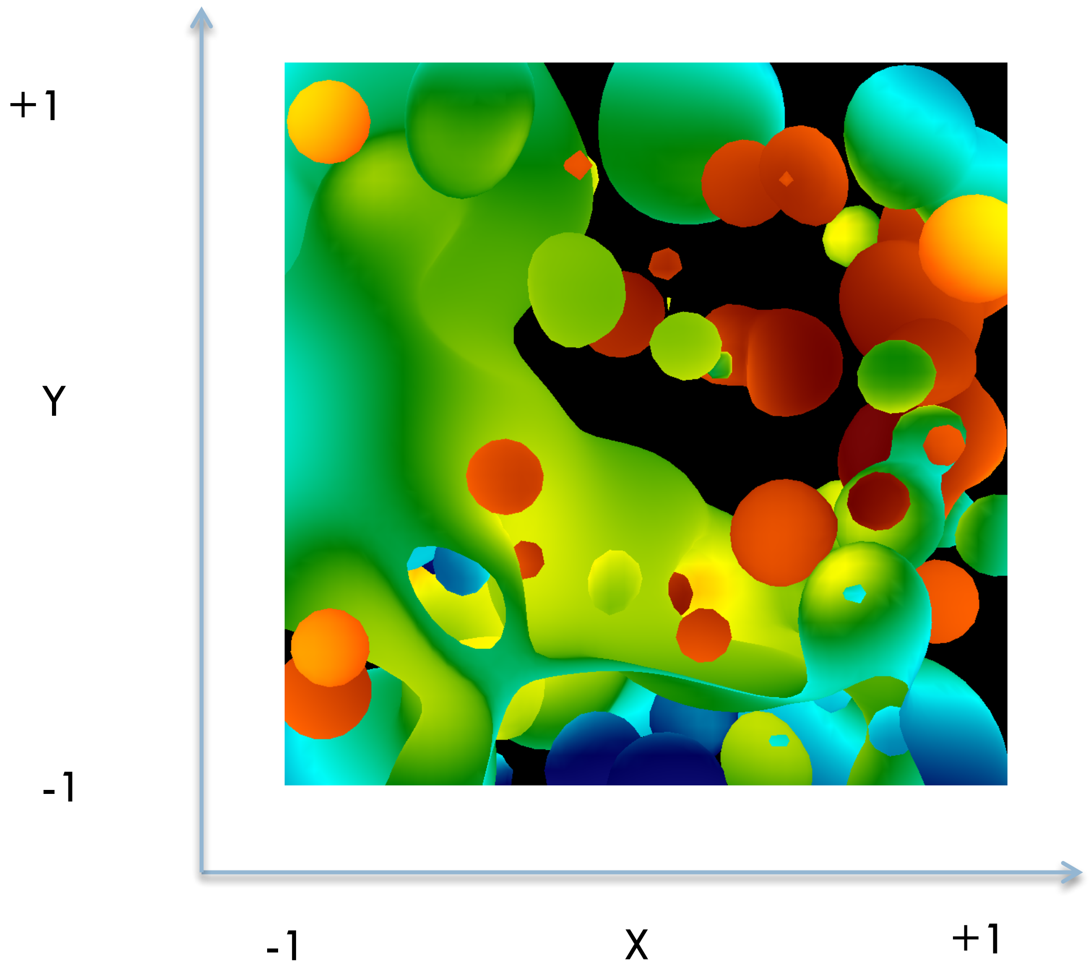
0<=x<=width, 0

<=y<=height

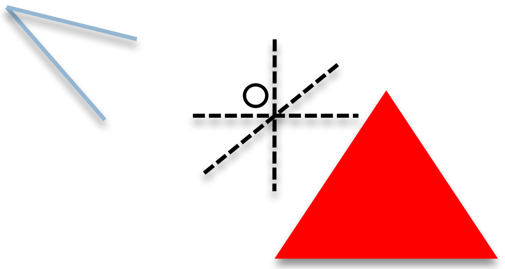
Screen Space

- Screen Space is the intersection of the xy-plane with Image Space.
- Points in image space are mapped into screen space by projecting via a parallel projection, onto the plane $z = 0$.
- Example:
  - a point $(0, 0, z)$ in image space will project to the center of the display screen

Screen Space Diagram



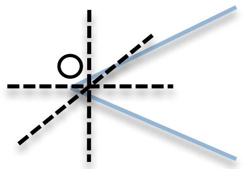
Our goal



World space:

Triangles in native Cartesian coordinates

Camera located anywhere



Camera space:

Camera located at origin, looking down -Z

Triangle coordinates relative to camera frame

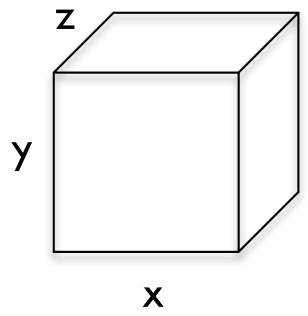
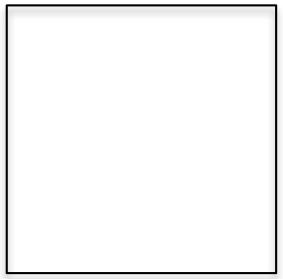


Image space:

All viewable objects within

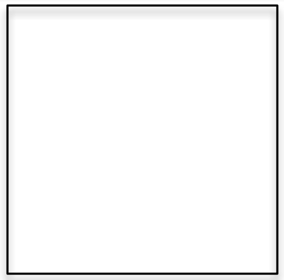
-1 <= x,y,z <= +1



Screen space:

All viewable objects within

-1 <= x, y <= +1



Device space:

All viewable objects within

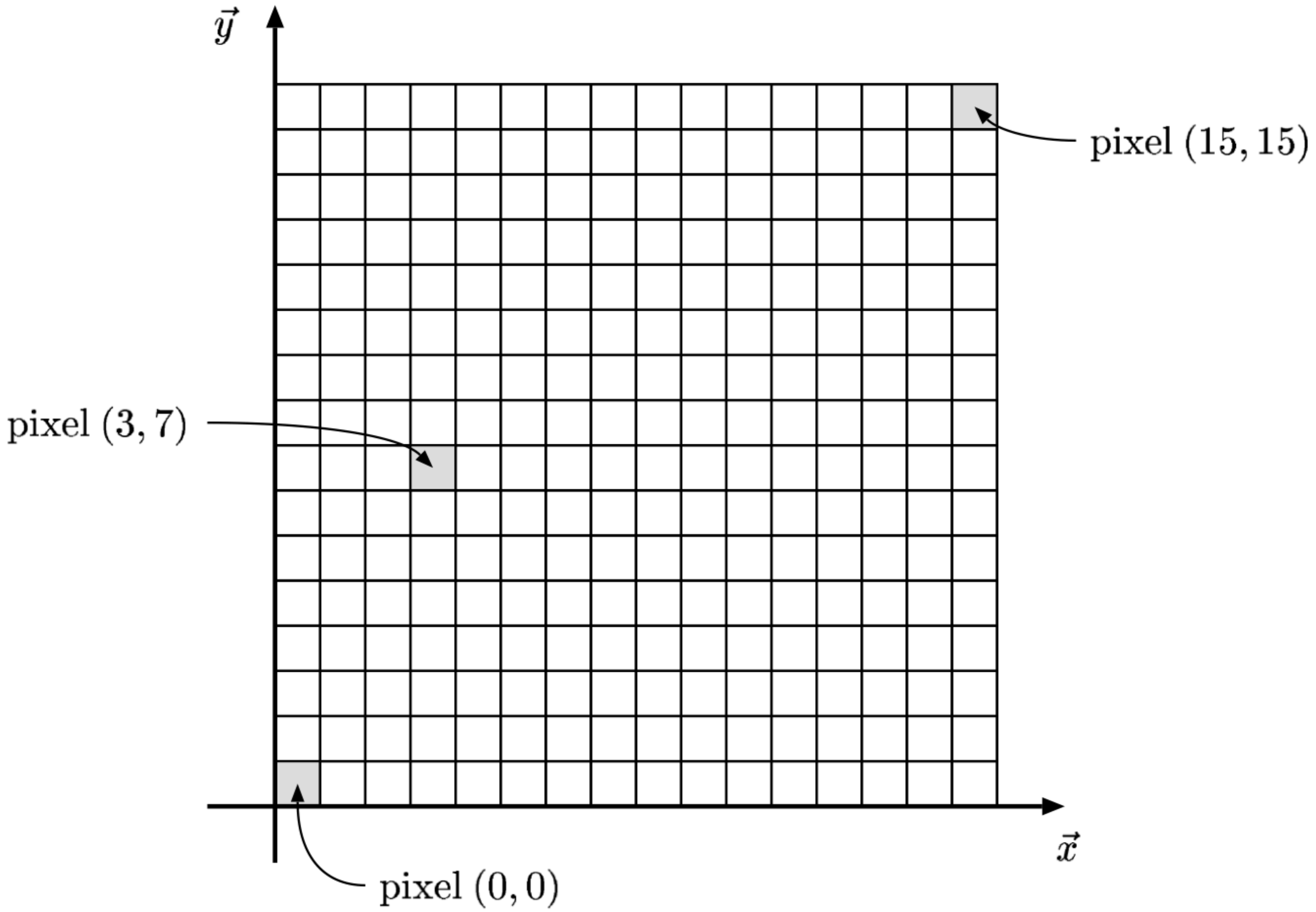
0<=x<=width, 0

<=y<=height

Device Space

- Device Space is the lowest level coordinate system and is the closest to the hardware coordinate systems of the device itself.
- Device space is usually defined to be the n × m array of pixels that represent the area of the screen.
- A coordinate system is imposed on this space by labeling the lower-left-hand corner of the array as (0,0), with each pixel having unit length and width.

Device Space Example



Device Space With Depth Information

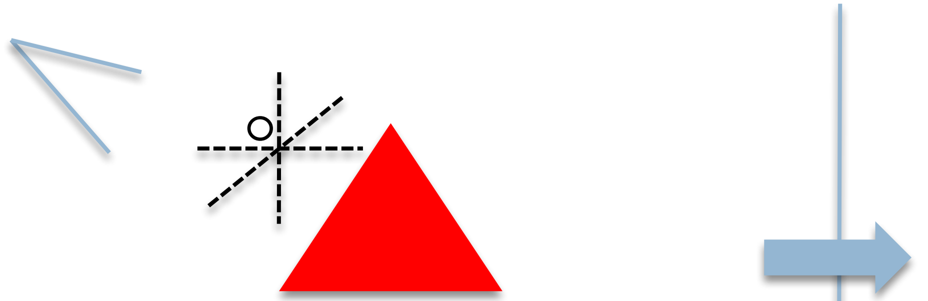
- Extends Device Space to three dimensions by adding z-coordinate of image space.
- Coordinates are (x, y, z) with

  $0 \leq x \leq n$

  $0 \leq y \leq m$

  z arbitrary (but typically $-1 \leq z \leq +1$ or $-1 \leq z \leq 0$)

Easiest Transform



World space:

Triangles in native Cartesian coordinates
Camera located anywhere

Camera space:

Camera located at origin, looking down -Z
Triangle coordinates relative to camera frame

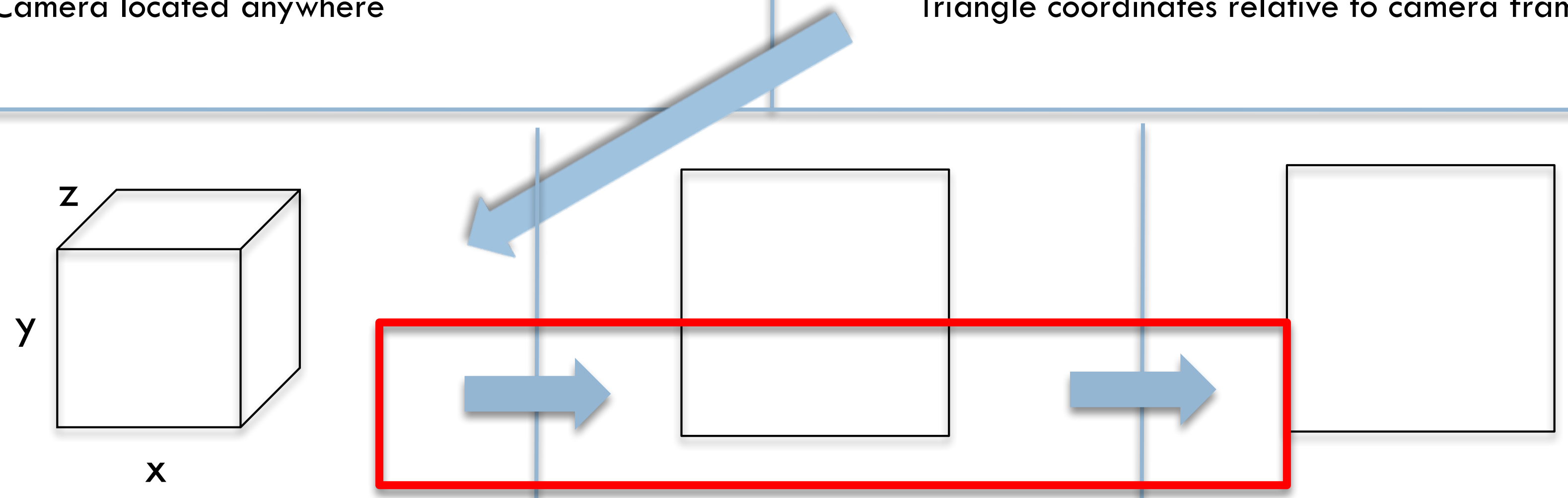


Image space:

All viewable objects within
-1 <= x,y,z <= +1

Screen space:

All viewable objects within
-1 <= x, y <= +1

Device space:

All viewable objects within
0<=x<=width, 0
<=y<=height

Image Space to Device Space

- (x, y, z) → (x’, y’, z’), where
  - x’ = n*(x+1)/2
  - y’ = m*(y+1)/2
  - z’ = z
  - (for an n x m image)
- Matrix:

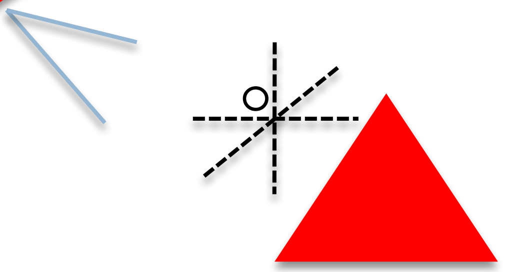
(x’ 0 0 0)

(0 y’ 0 0)

(0 0 z’ 0)

(0 0 0 1)

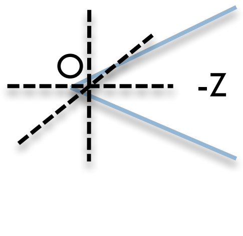
Coming Up on YouTube Lecture



World space:

Triangles in native Cartesian coordinates

Camera located anywhere



Camera space:

Camera located at origin, looking down -Z

Triangle coordinates relative to camera frame

- Need to construct a Camera Frame
- Need to construct a matrix to transform <u>points</u> from Cartesian Frame to Camera Frame
  - Transform triangle by transforming its three vertices