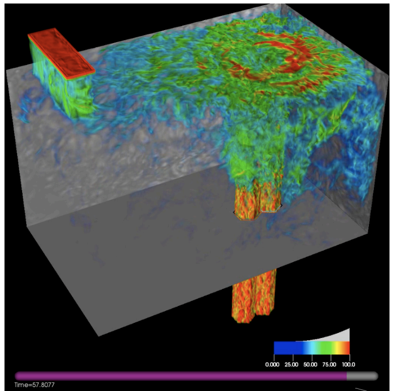
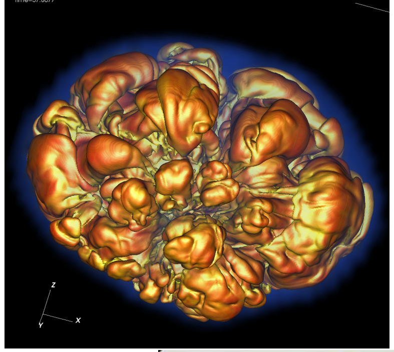
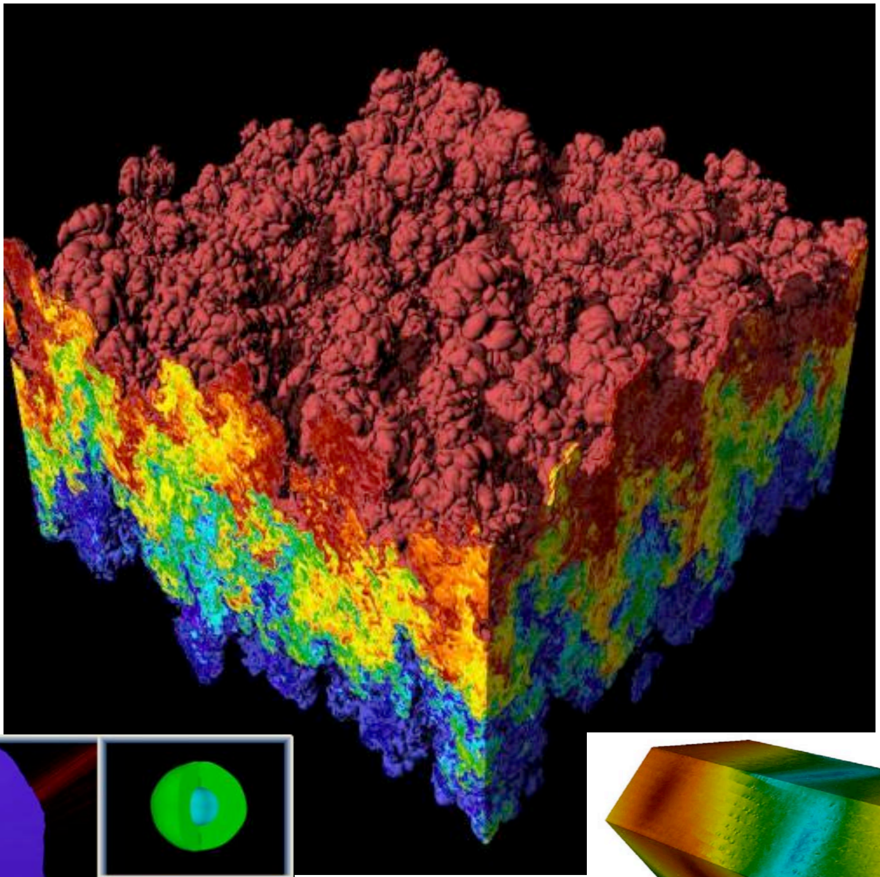
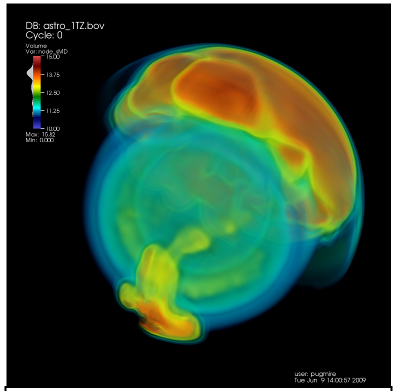
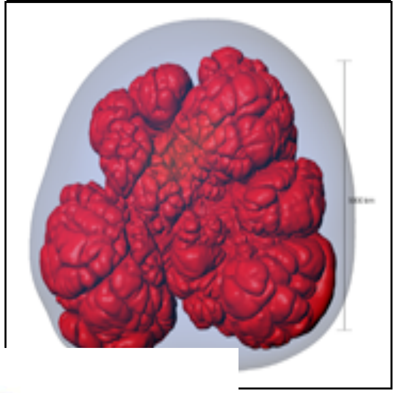
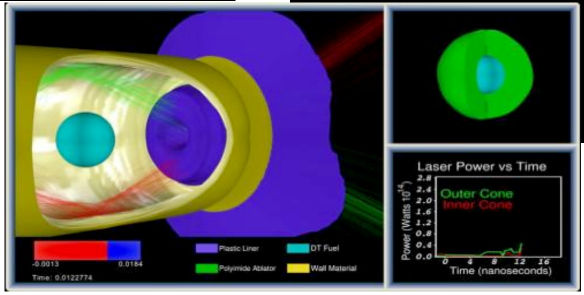
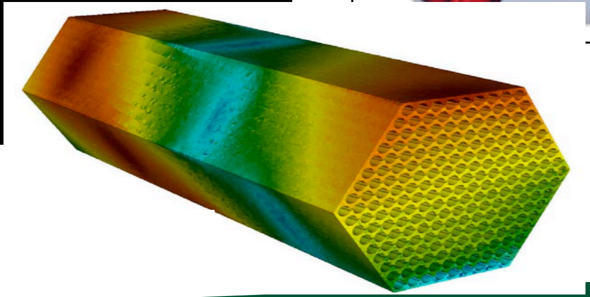
CIS 441/541: Intro to Computer Graphics
Lecture 5: Cameras & Matrices, Project 1E





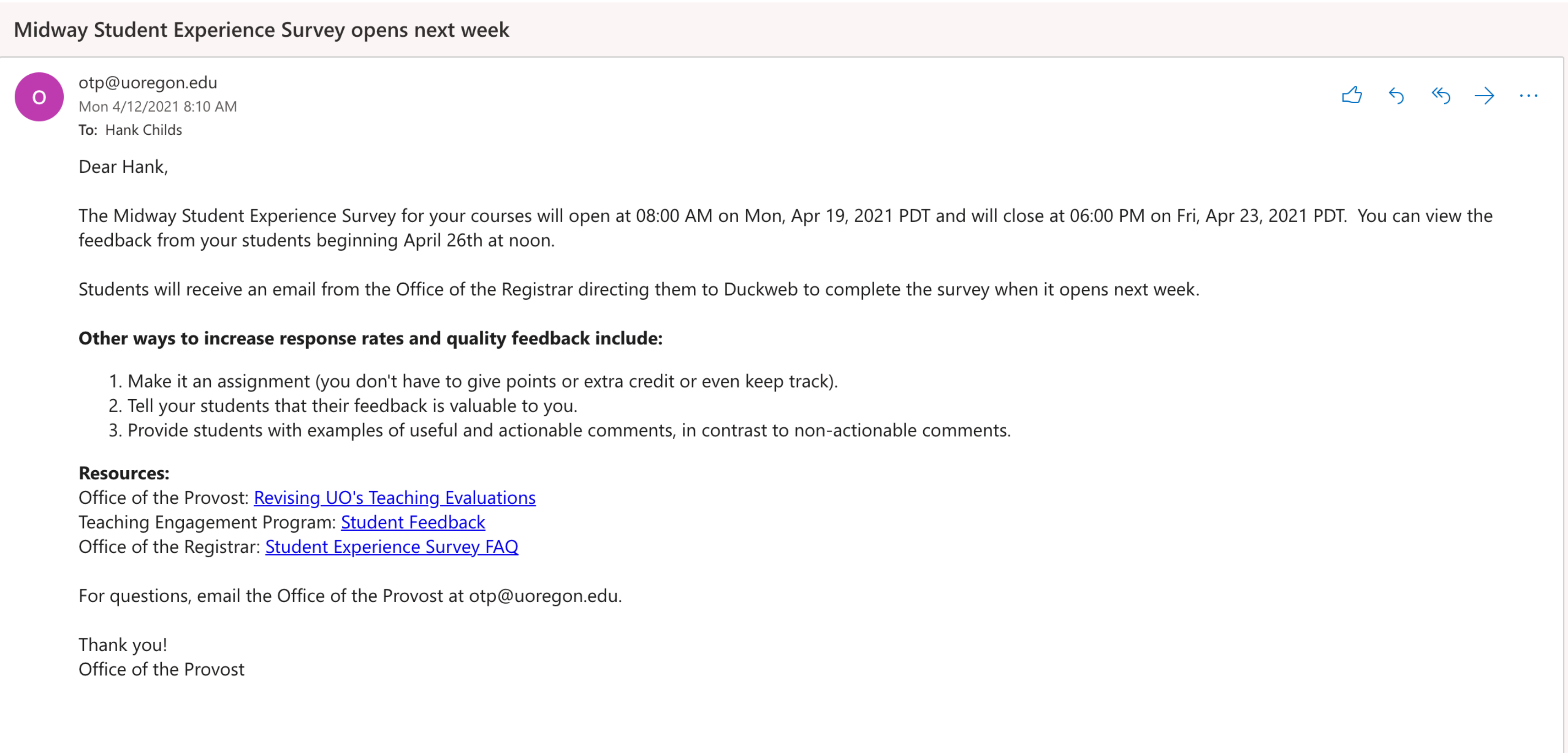


April 20, 2021

Hank Childs, University of Oregon

UNIVERSITY OF OREGON

Midway Experience



Midway Student Experience Survey opens next week

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.

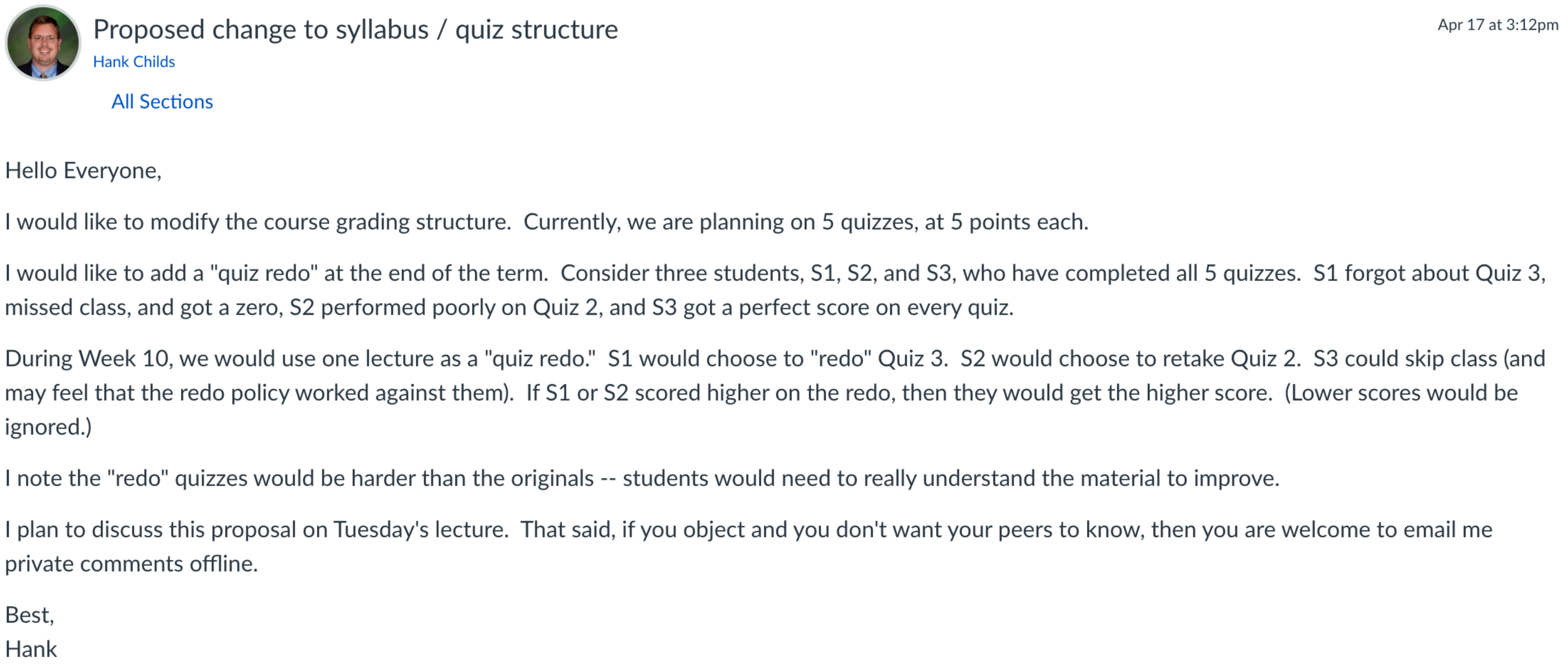
Resources:
Office of the Provost: Revising UO's Teaching Evaluations
Teaching Engagement Program: Student Feedback
Office of the Registrar: Student Experience Survey FAQ

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

Thank you!
Office of the Provost

UNIVERSITY OF OREGON

Proposed Change to Syllabus/Quiz Structure



Proposed change to syllabus / quiz structure

Hank Childs

Apr 17 at 3:12pm

All Sections

Hello Everyone,

I would like to modify the course grading structure. Currently, we are planning on 5 quizzes, at 5 points each.

I would like to add a "quiz redo" at the end of the term. Consider three students, S1, S2, and S3, who have completed all 5 quizzes. S1 forgot about Quiz 3, missed class, and got a zero, S2 performed poorly on Quiz 2, and S3 got a perfect score on every quiz.

During Week 10, we would use one lecture as a "quiz redo." S1 would choose to "redo" Quiz 3. S2 would choose to retake Quiz 2. S3 could skip class (and may feel that the redo policy worked against them). If S1 or S2 scored higher on the redo, then they would get the higher score. (Lower scores would be ignored.)

I note the "redo" quizzes would be harder than the originals -- students would need to really understand the material to improve.

I plan to discuss this proposal on Tuesday's lecture. That said, if you object and you don't want your peers to know, then you are welcome to email me private comments offline.

Best,
Hank

Thursday’s Lecture – Asynchronous (sort of)

- Links are posted on class webpages to YouTube videos
- These lectures are from last offering, but I rewatched them and they are still applicable
- WARNING: the 1E overview video has the wrong due date – April 27th, 2021, not Feb 6th, 2019.
- IMPORTANT: group discussion at 9am on Thursday

So What Do We Do On Thursday?

- Watch YouTube videos before Thursday’s class
- Call in at 9am on Thursday
- Group discussion of any questions on cameras

Class Plan

- Abhishek and I are working hard on preparing Project 2 (OpenGL)
- Projects will start coming faster
  - Want there to time to do great final projects
- 1E, 1F: simpler coding, harder concepts

Current Plan (1/2)

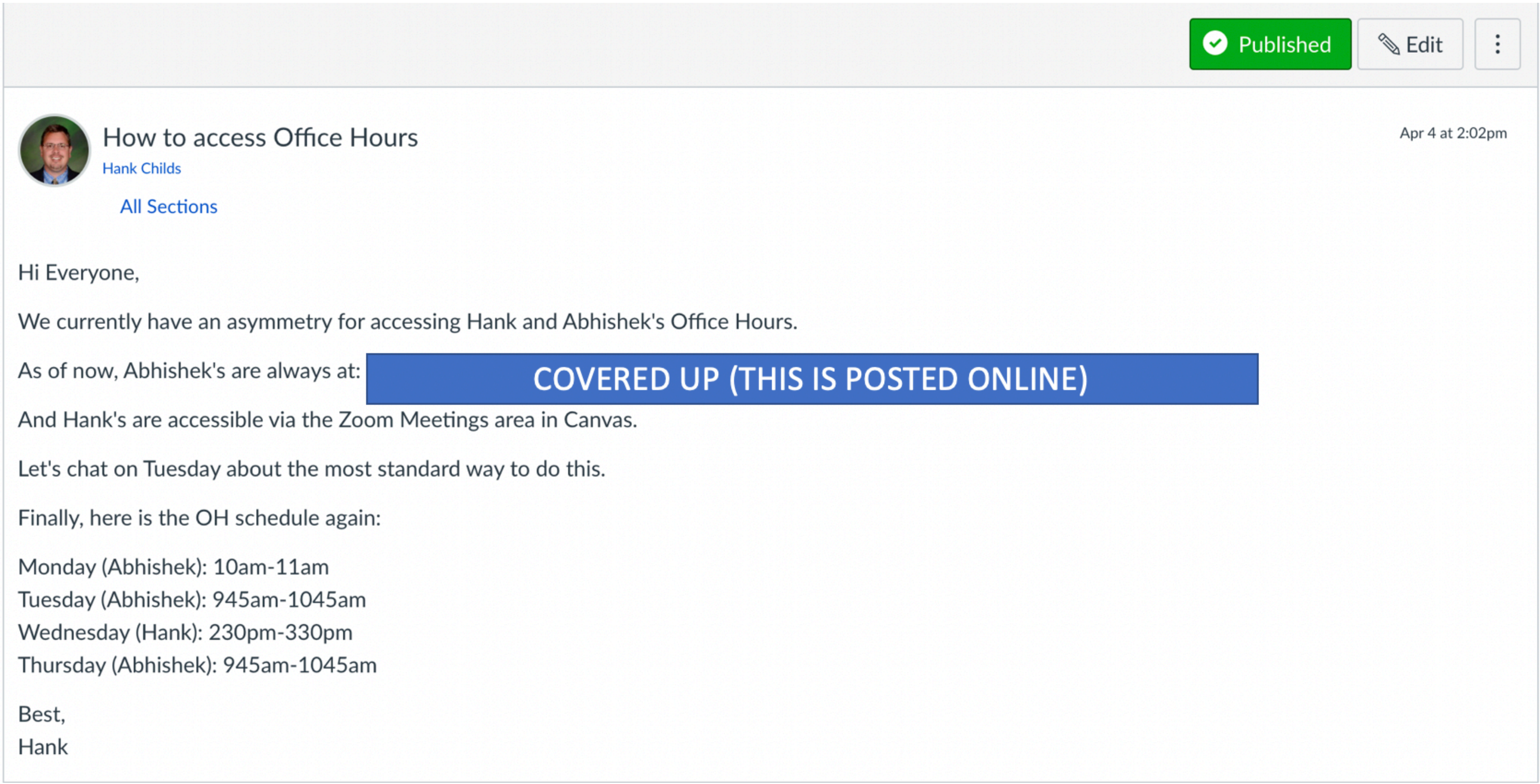
Week	Sun	Mon	Tues	Weds	Thurs	Fri	Sat
4			Lec5, 1E assigned	1D due	Lec 6 (async / group chat)		
5			Lec 7 (shading), 1F assigned, 1E due		Lec 8 (GL), 2A assigned		
6		1F due	Lec 9 (GL), 2B assigned		Discussion of final projects / Quiz 3		2A due
7			Lec 11 – ray tracing		More discussion of final projects (?)	2B due	

Current Plan (2/2)

- Weeks 8-10 → you work on final projects
- Lectures will be on misc. topics in graphics, esp. in support of final projects
- Quiz 3 (Week 6): likely on matrices
- Quiz 4 (Week 8): likely on GL
- Quiz 5 (Week 10): likely on topics in final weeks

UNIVERSITY OF OREGON

Week 4 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:

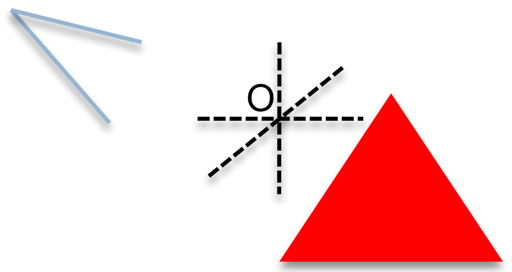
Monday (Abhishek): 10am-11am
Tuesday (Abhishek): 945am-1045am
Wednesday (Hank): 230pm-330pm
Thursday (Abhishek): 945am-1045am

Best,
Hank

Cameras and Matrices

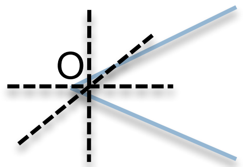
- Note: I ~~will be~~ am repeating some of this content ~~next~~ this week.

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

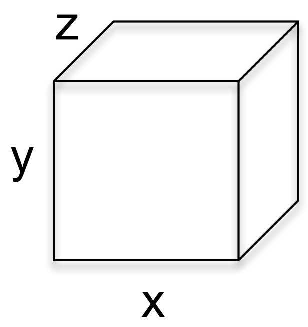
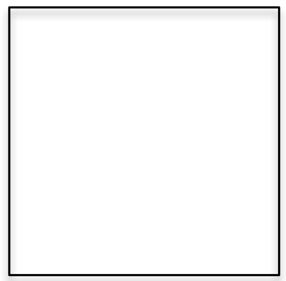


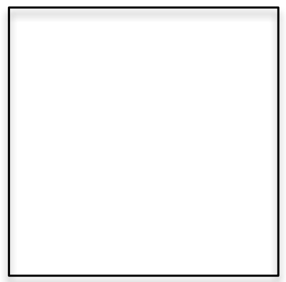
Image space:

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!

- Concepts coming:
  - Spaces
  - Basis
  - Coordinates
  - Frames
  - Matrices

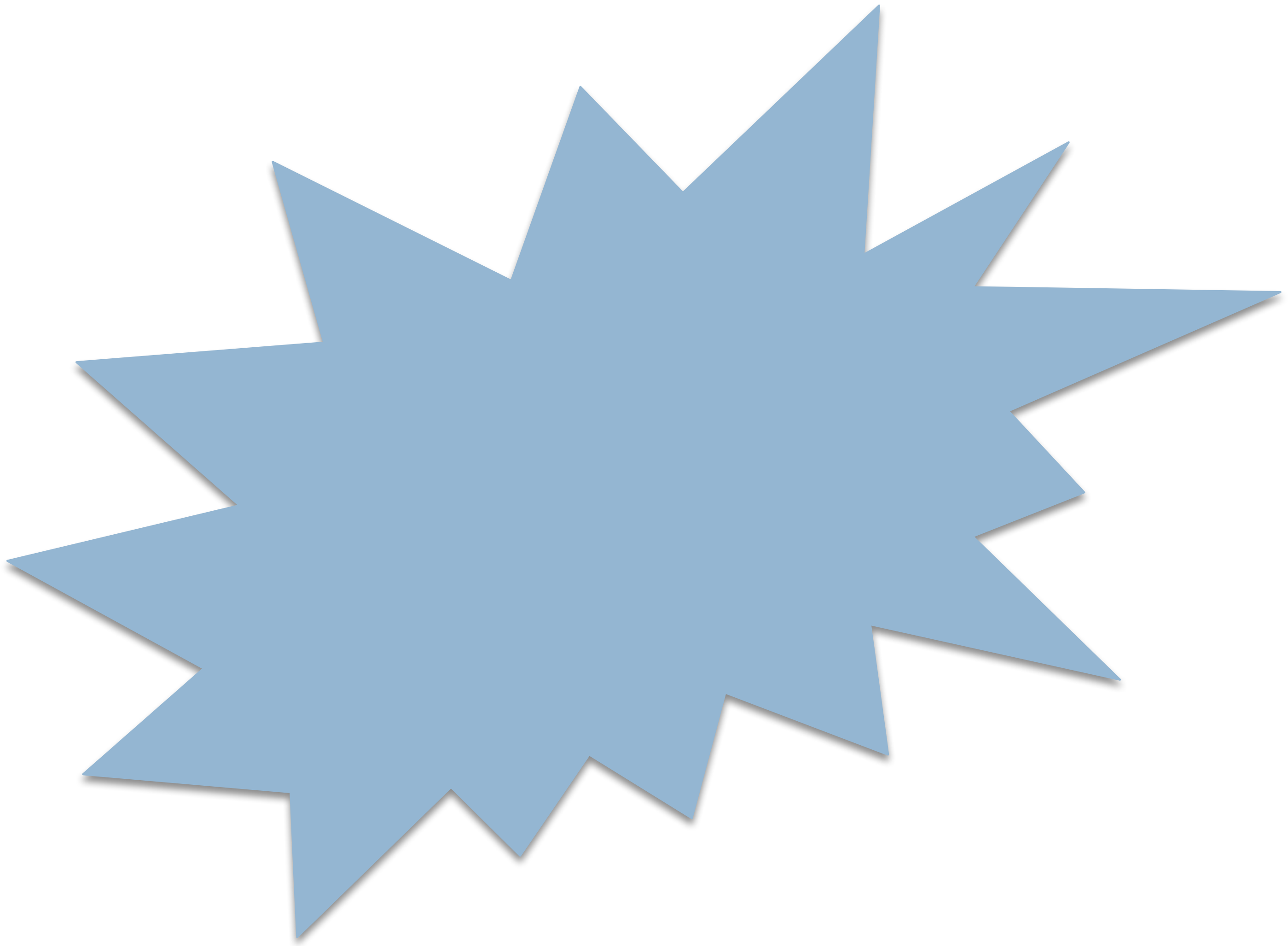
MATH!

- Concepts coming:
  - Spaces
  - Basis
  - Coordinates
  - Frames
  - Matrices

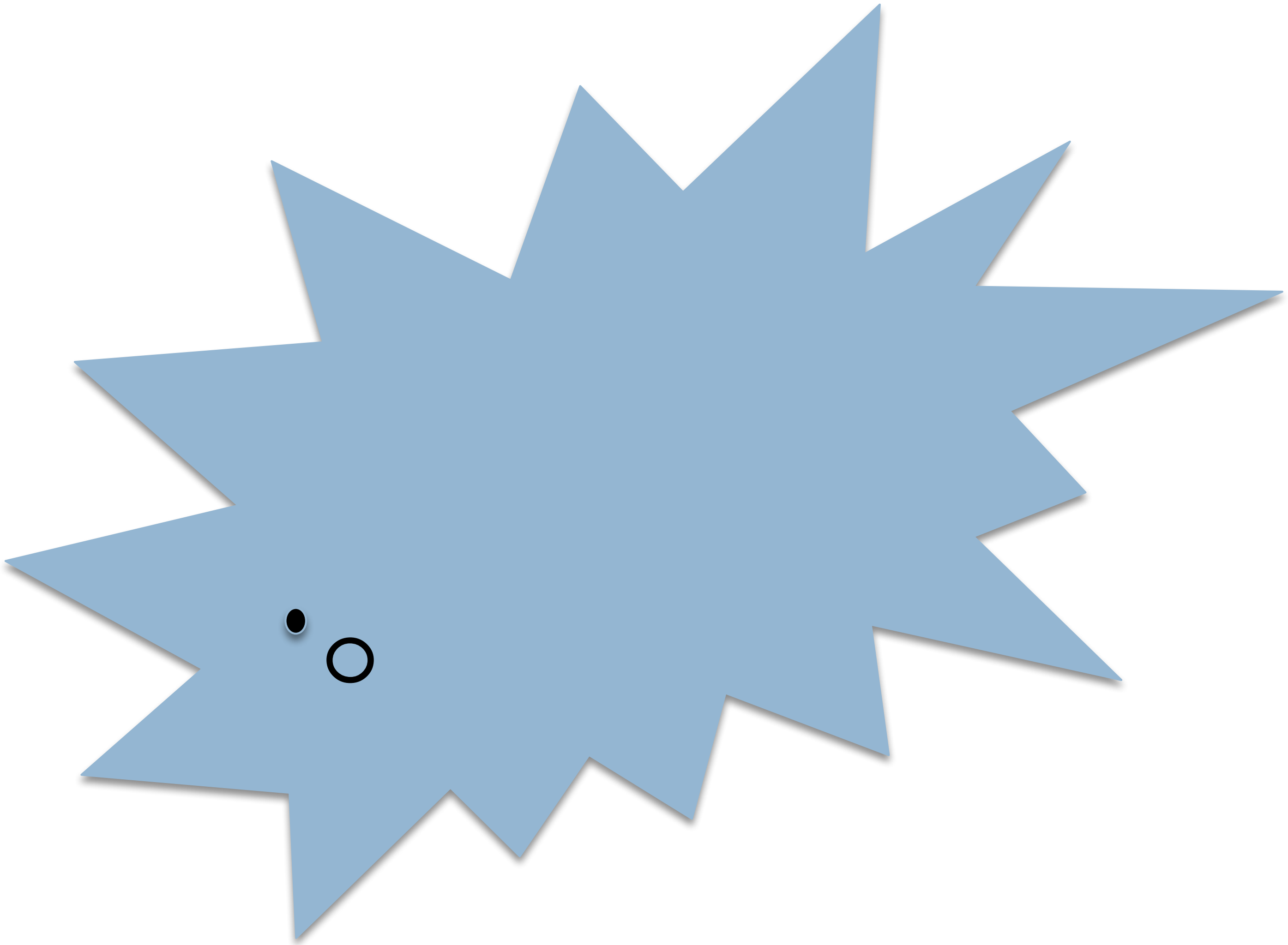
Space

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

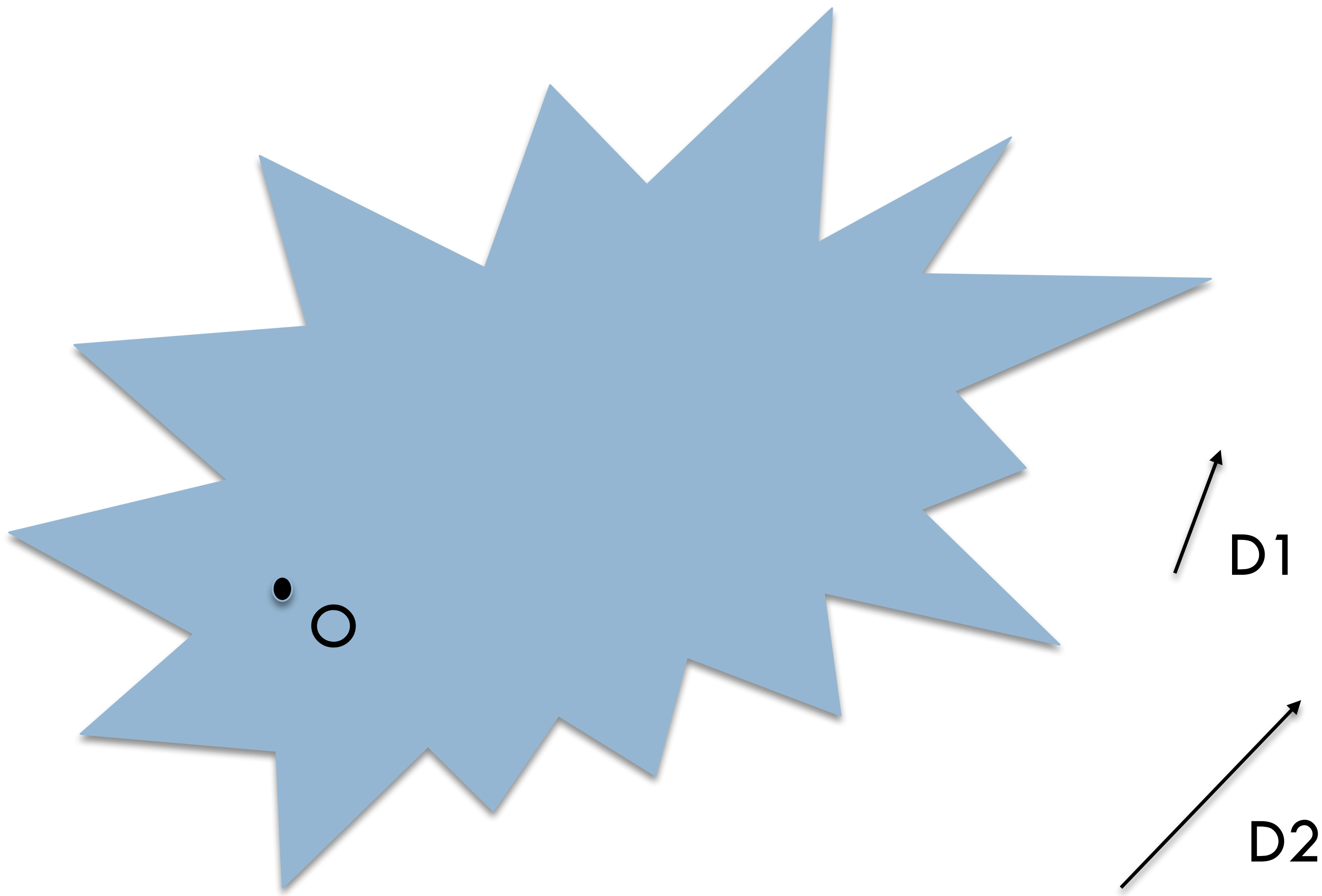
Here is a space 'S': the points in the blue shape



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

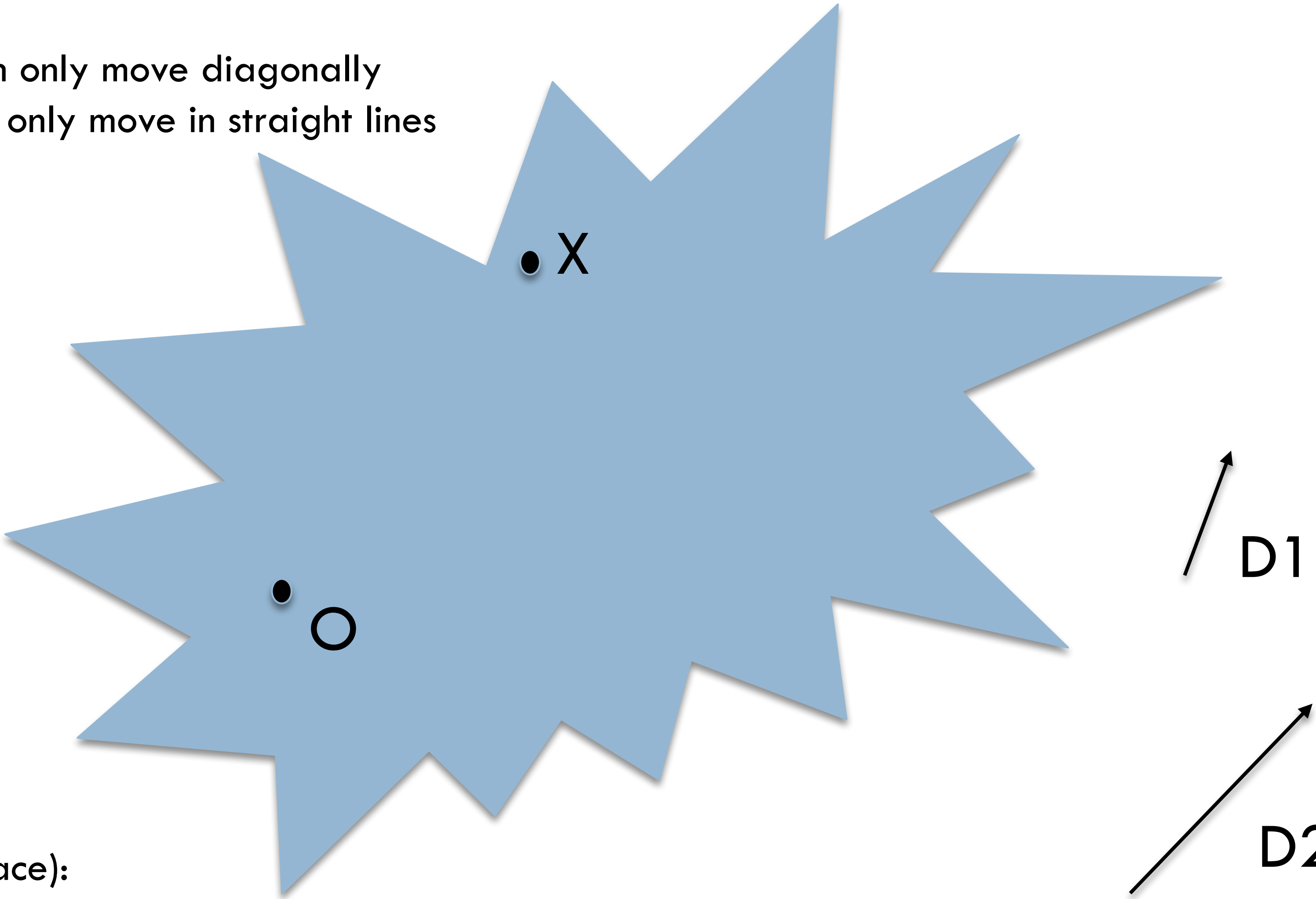


Consider two directions, D1 and D2.



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

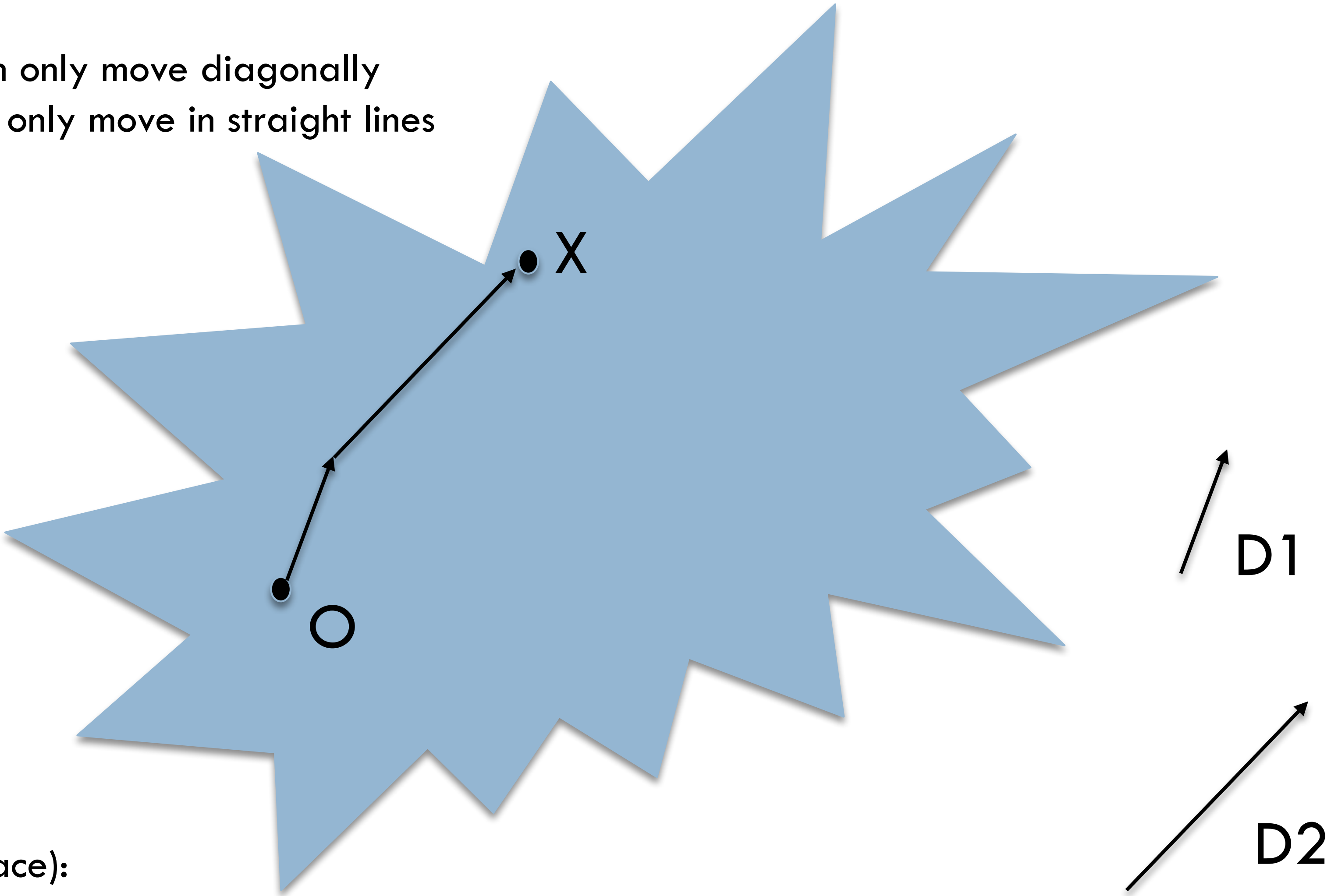


Rules (this space):
- You can only move in direction D1 or D2

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



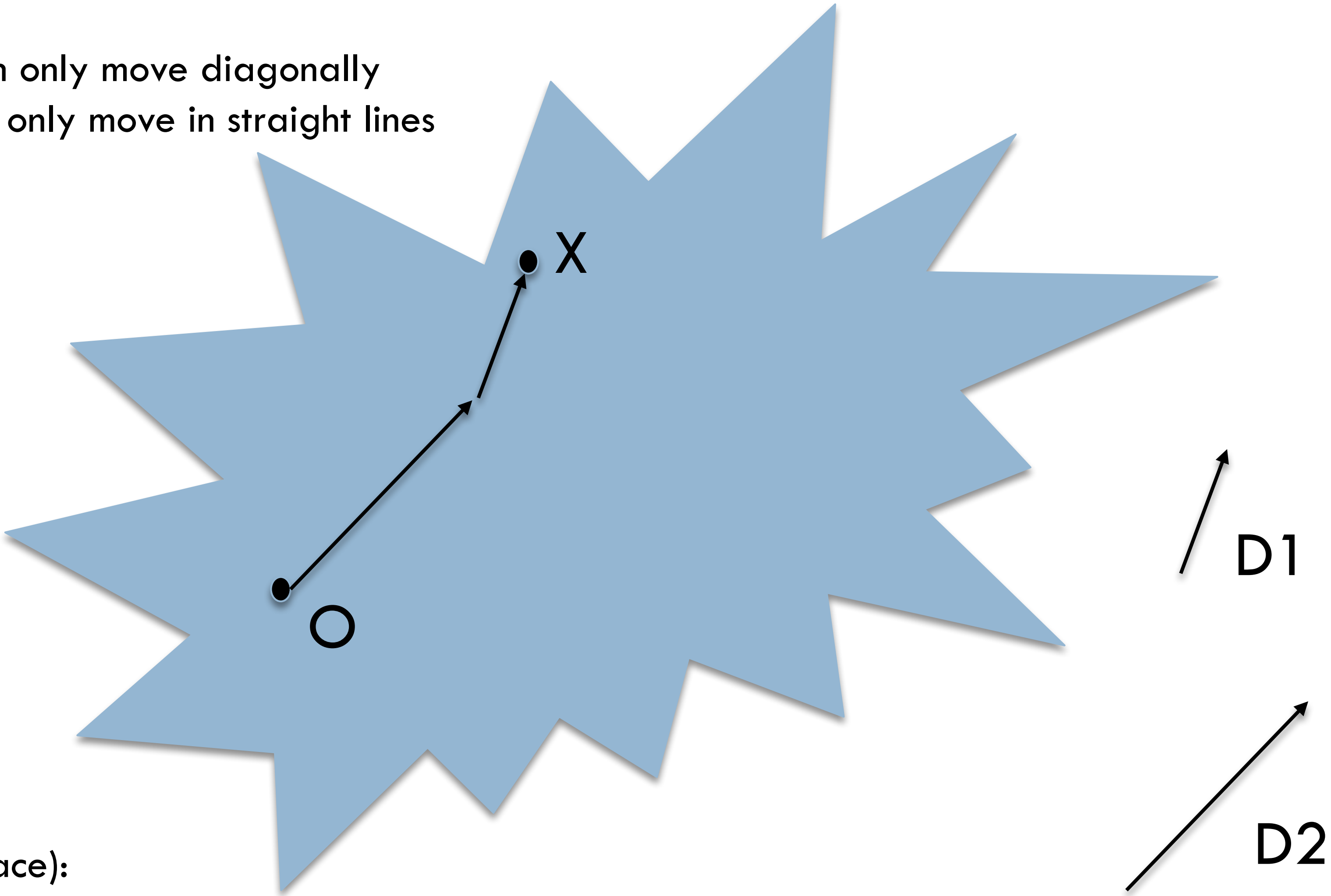
Rules (this space):

- 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



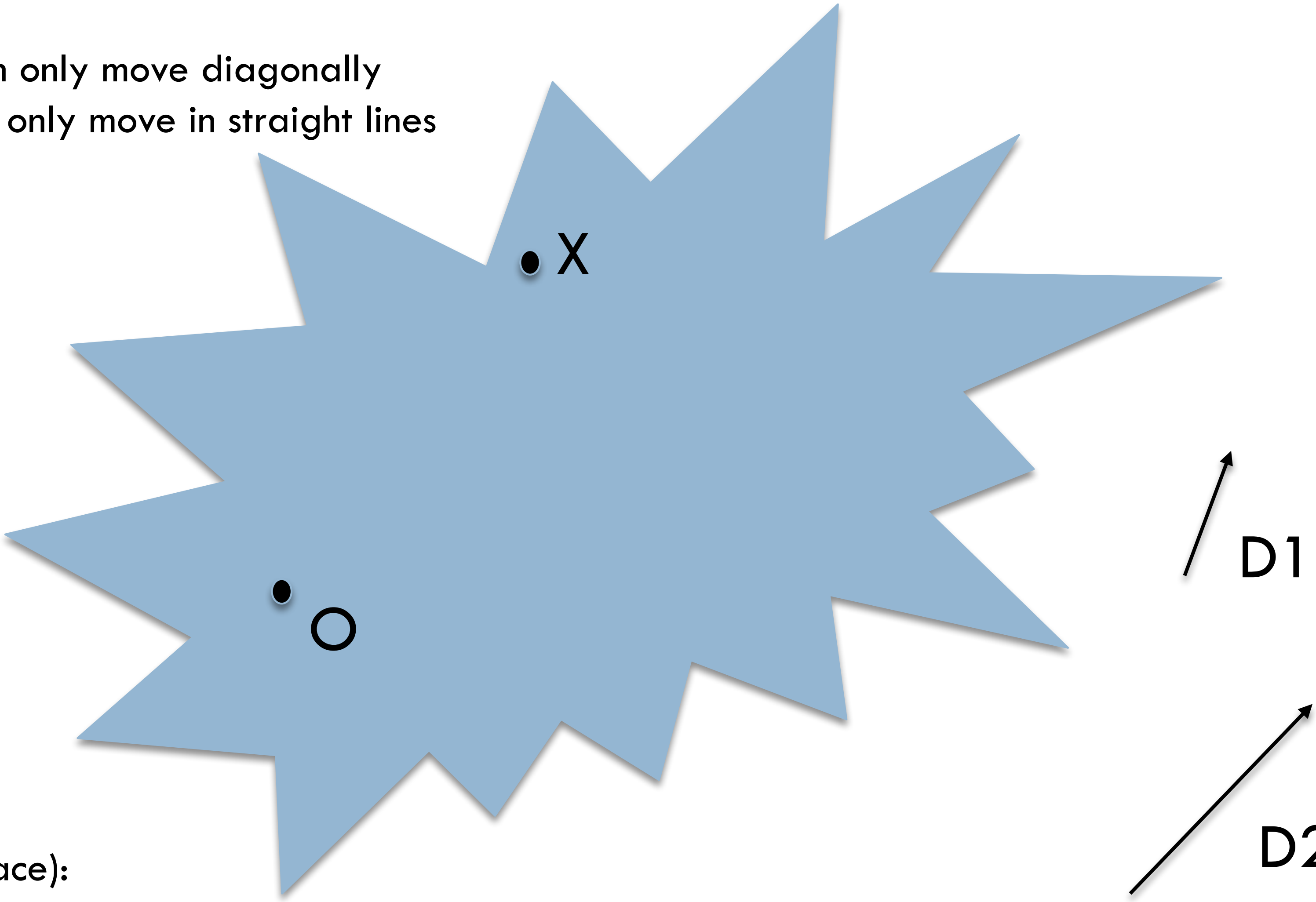
Rules (this space):

- 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



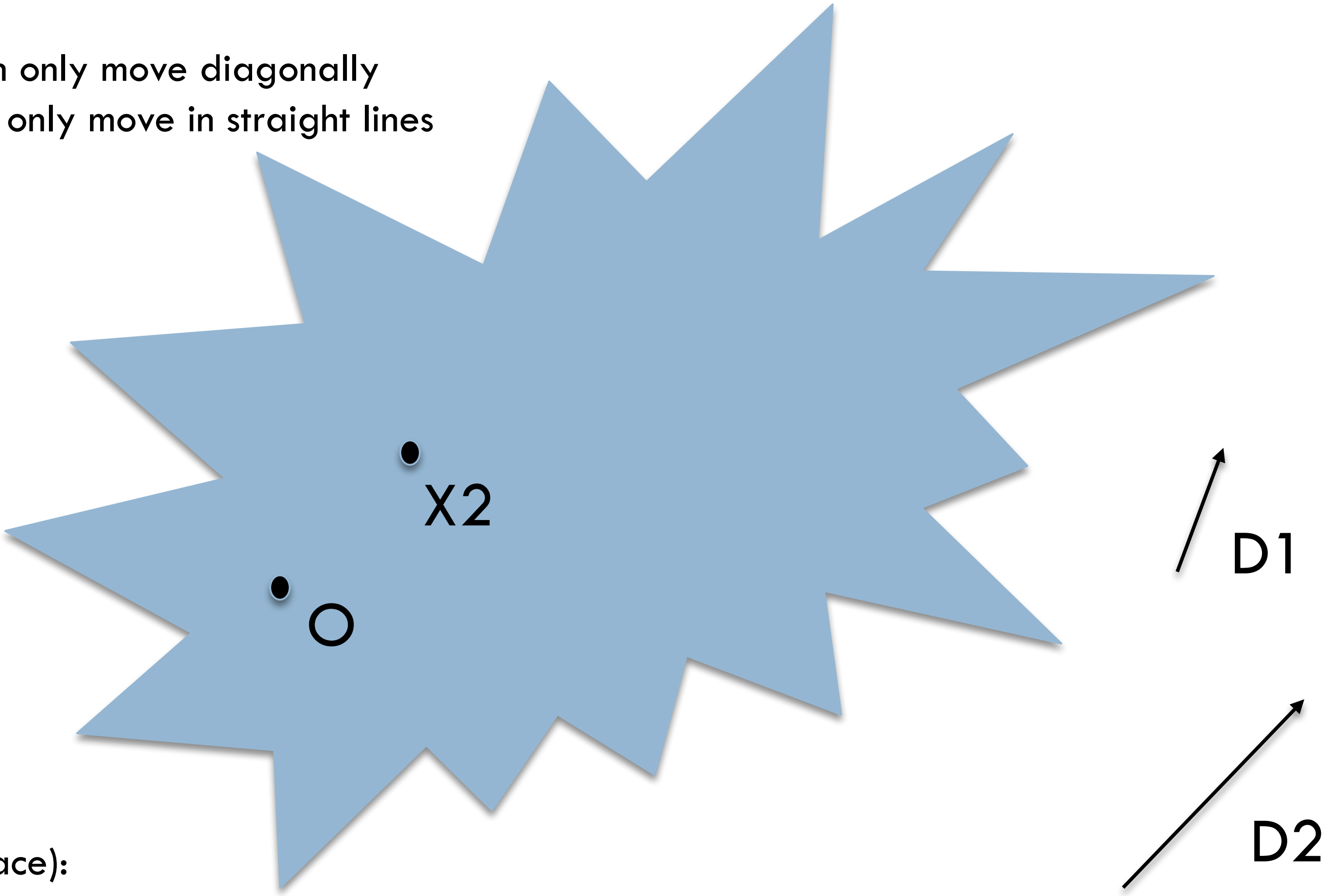
Rules (this space):

- 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



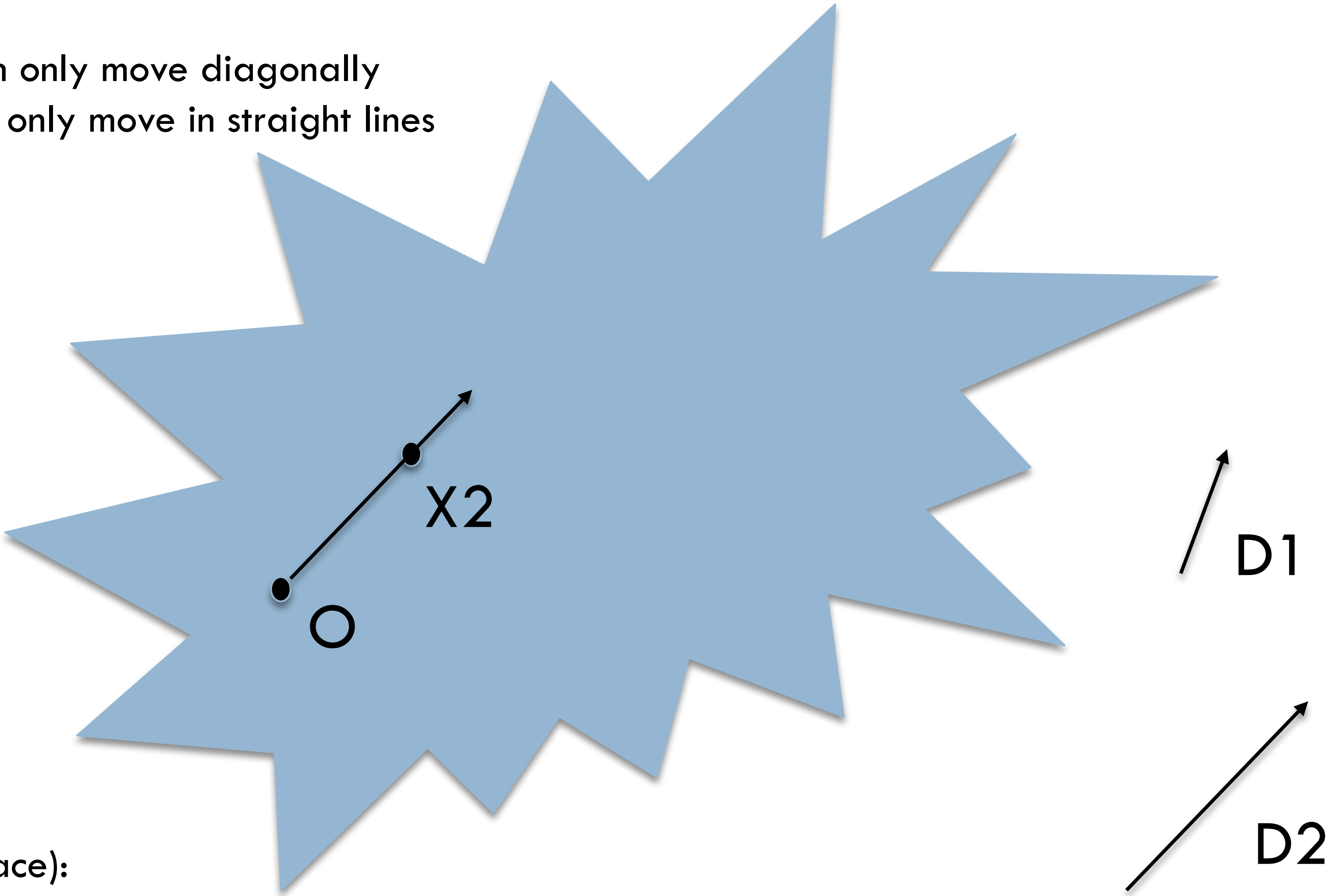
Rules (this space):

- 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

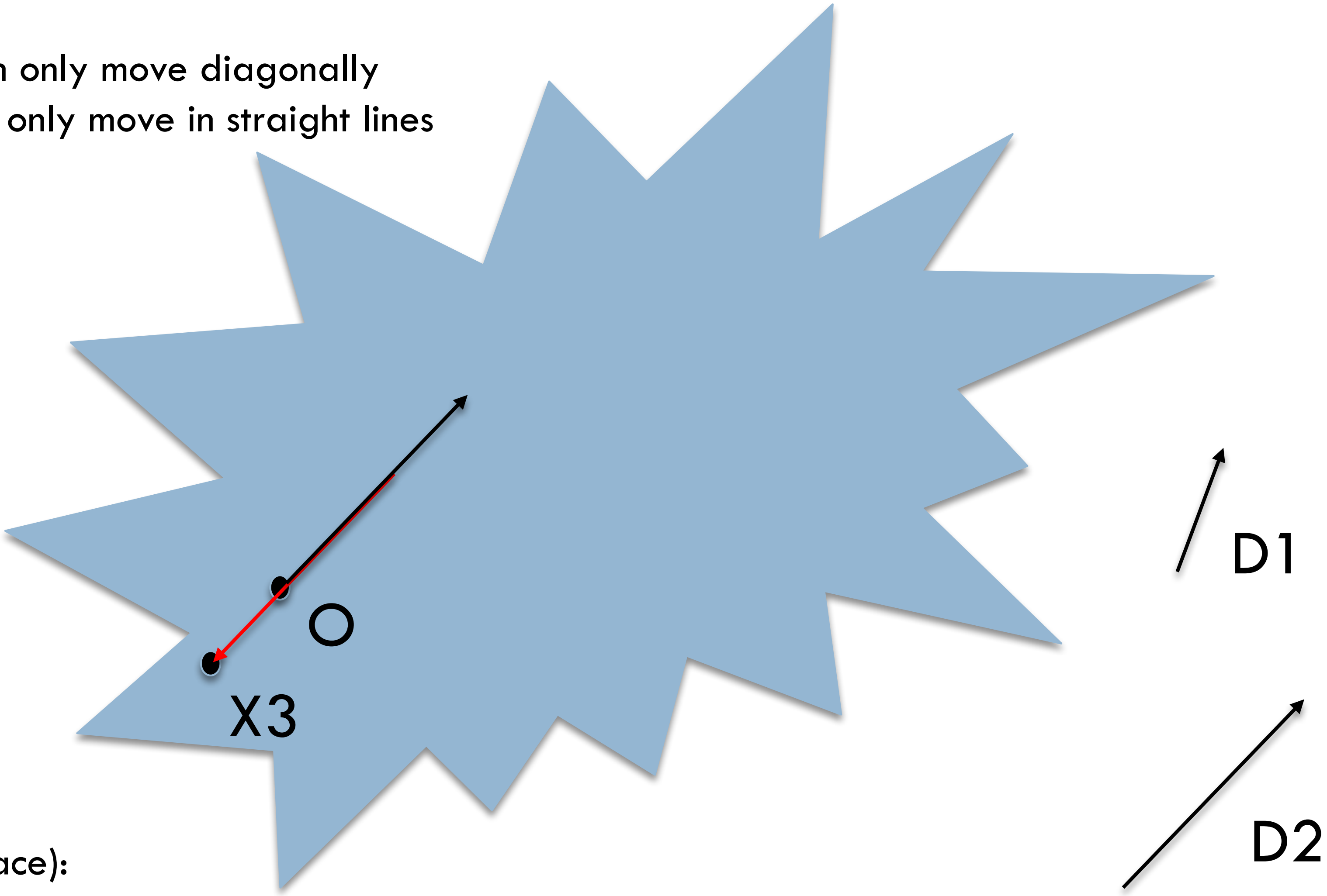


Rules (this space):

- 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

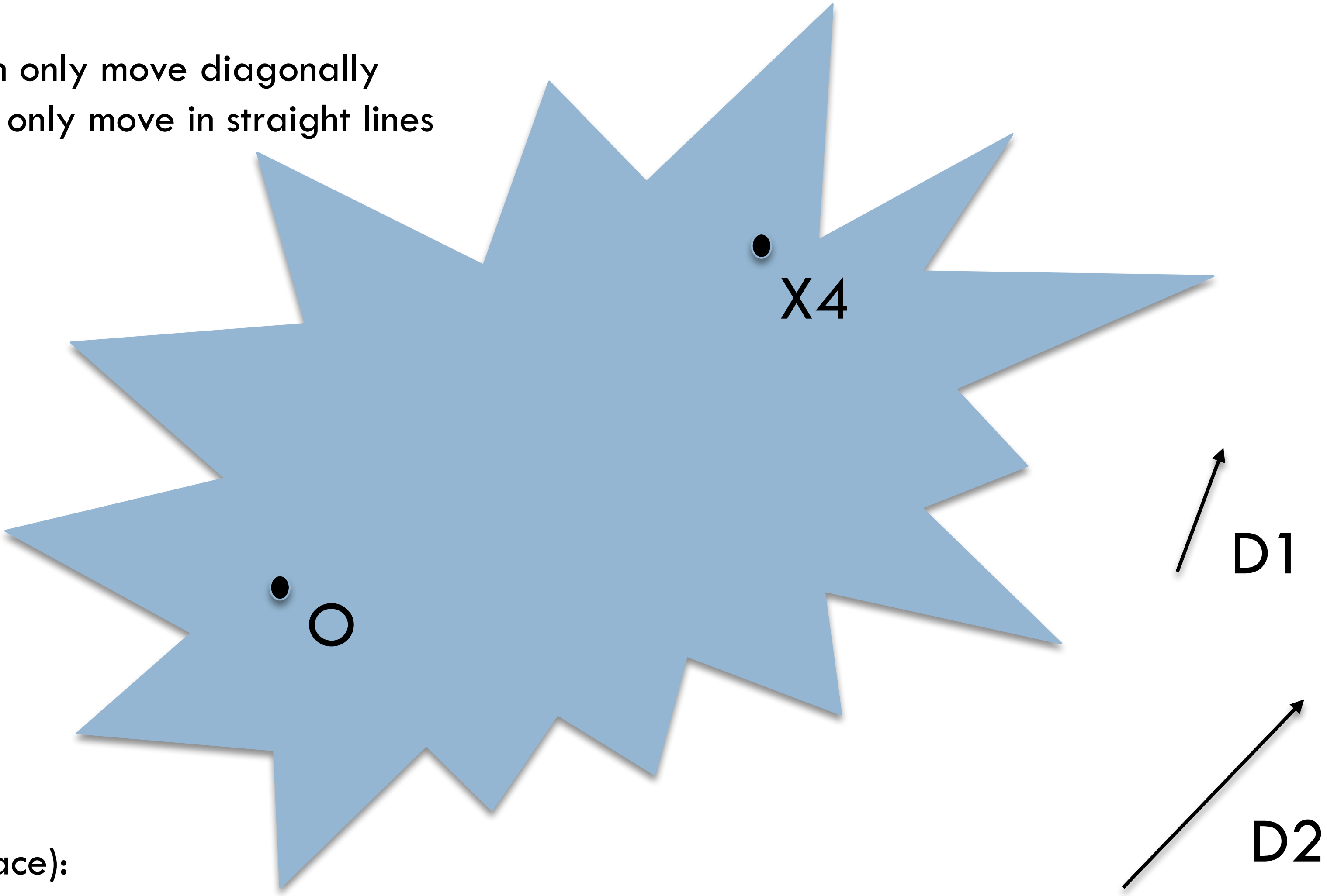


Rules (this space):
- You can only move in direction D1 or D2

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



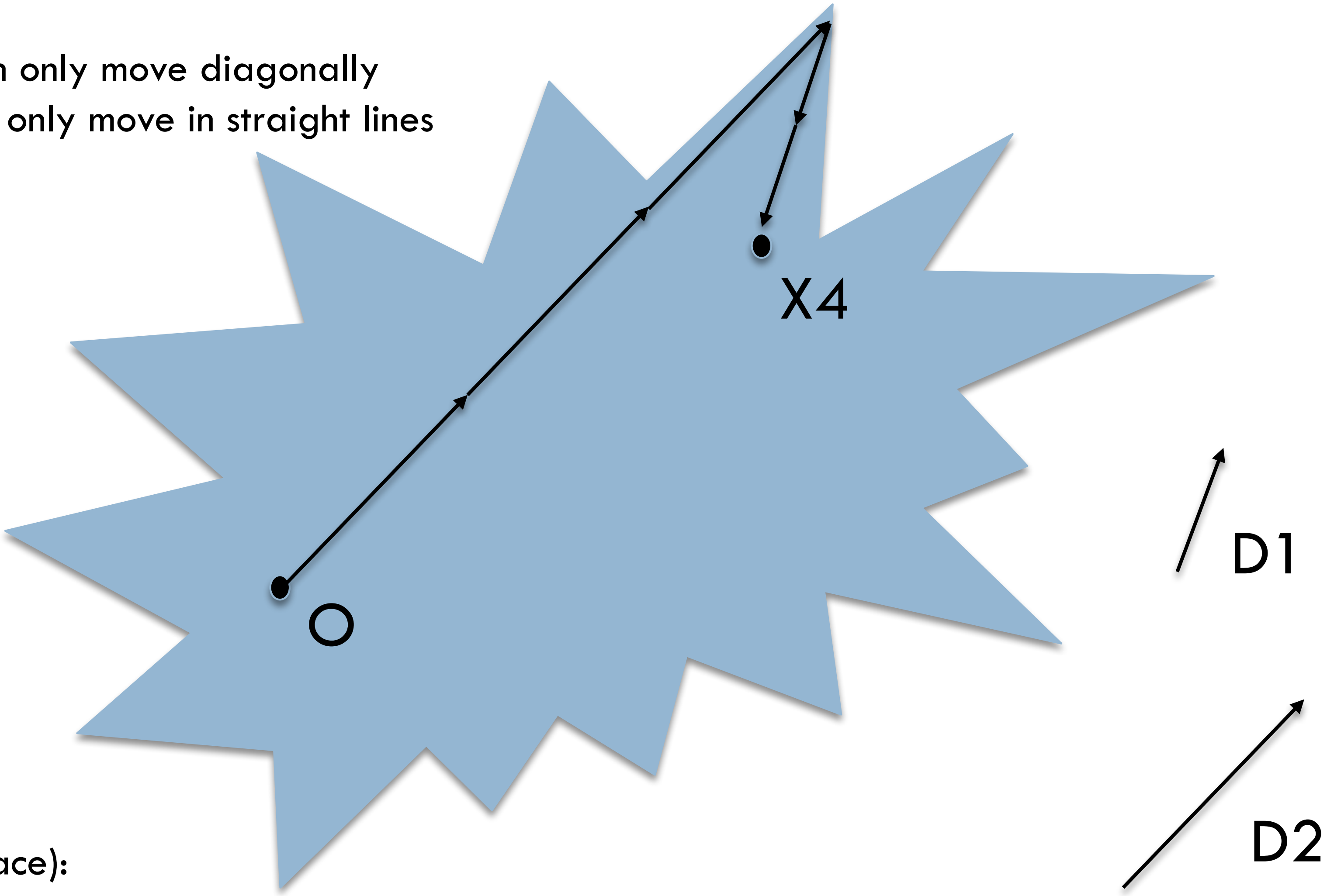
Rules (this space):

- 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

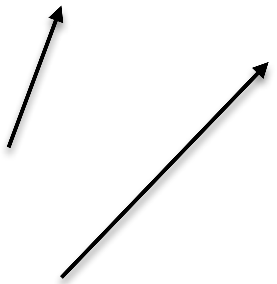


Rules (this space):

- You can only move in direction D1 or D2

Conventions!

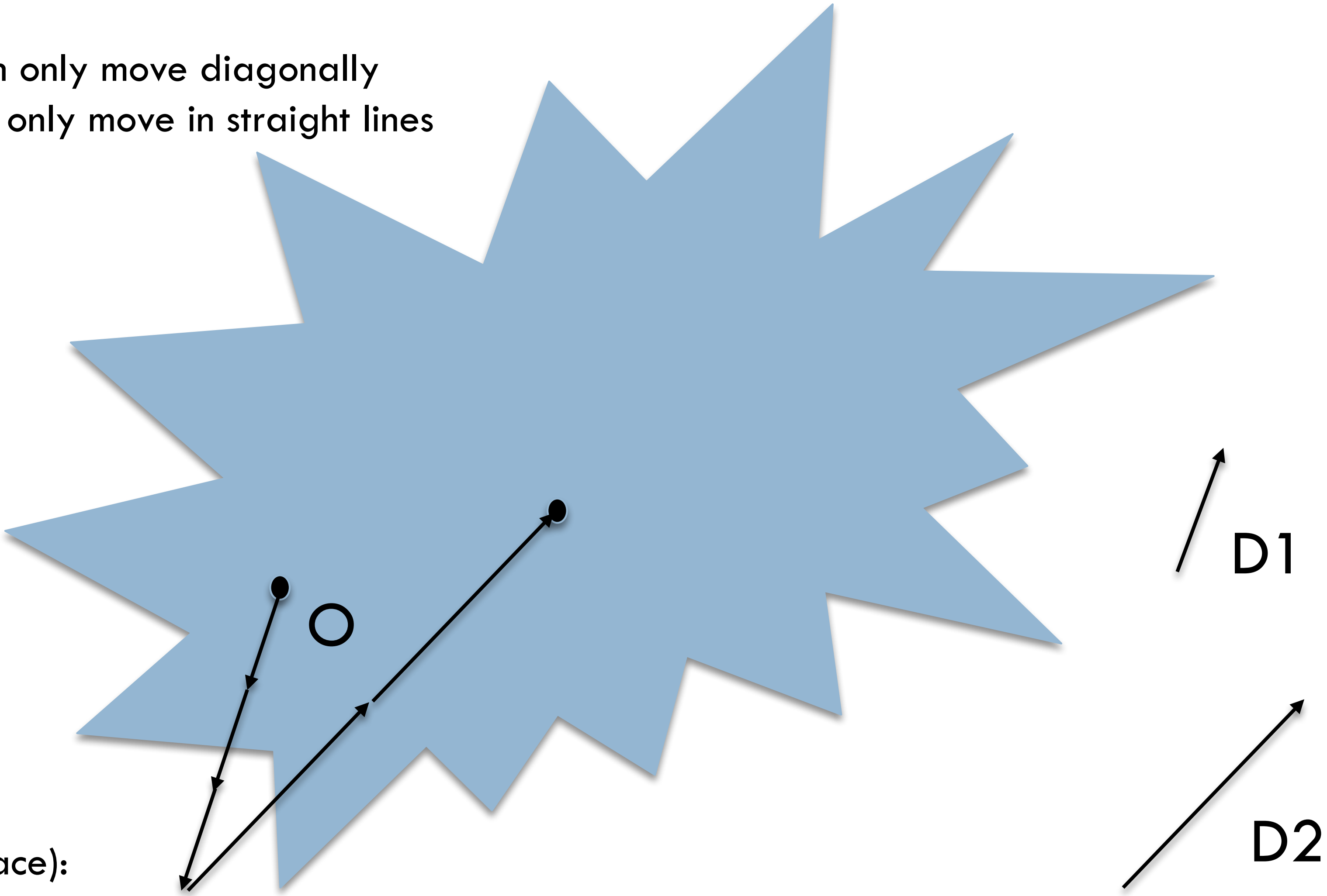
- Let (a, b) mean:
  - The number of steps ‘a’ in direction D1
  - The number of steps ‘b’ in direction D2



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

MATH!

- Concepts coming:
  - Spaces
  - Basis
  - Coordinates
  - Frames
  - Matrices

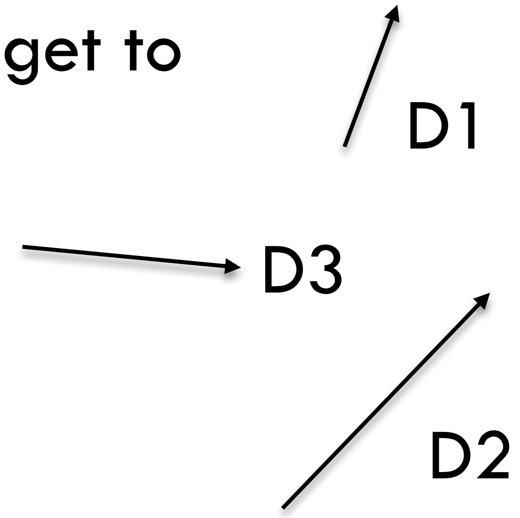
A basis

- 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 coordinates on *B* of the vector.
- The elements of a basis are called basis vectors.

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.
  - This point is relative to some 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 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 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

MATH!

- Concepts coming:
  - Spaces
  - Basis
  - Coordinates
  - Frames
  - Matrices

Frames

- 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).

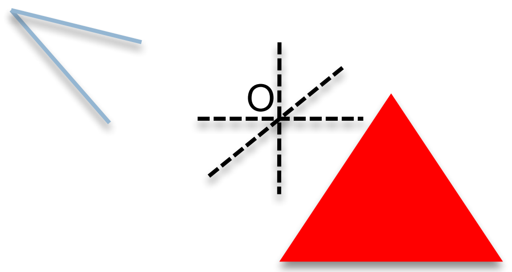
Example of Frames

- 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

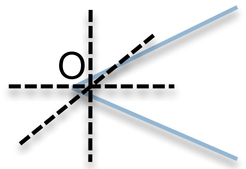
- Frame F = (v1, v2, O)
  - v1 = (0, -1)
  - v2 = (1, 0)
  - O = (3, 4)
- What are F’s coordinates for the point (6, 6)?
- Answer: (-2, 3)

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

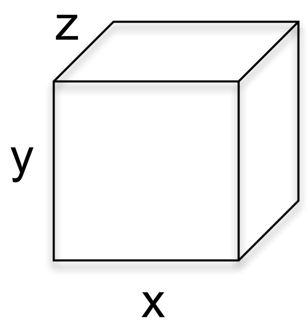
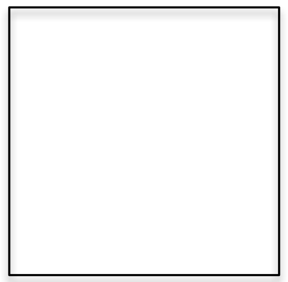


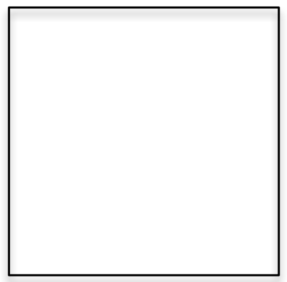
Image space:

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



Screen space:

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

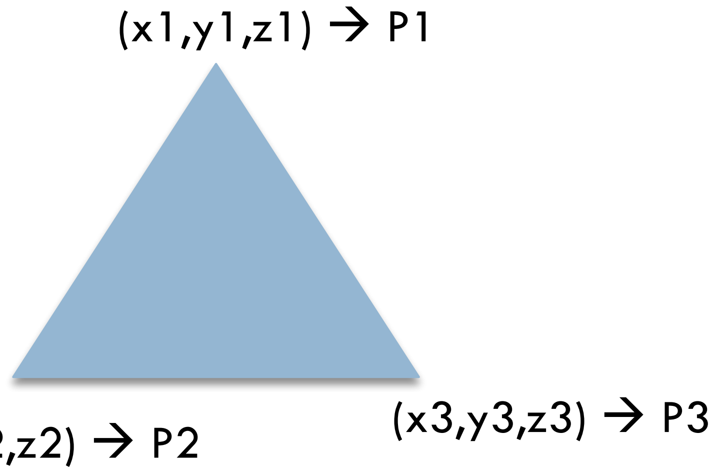


Device space:

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



$$\begin{pmatrix} x1 & y1 & z1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{pmatrix} = \begin{pmatrix} x+dx & y+dy & 1 \end{pmatrix}$$

But wait! There's more…

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

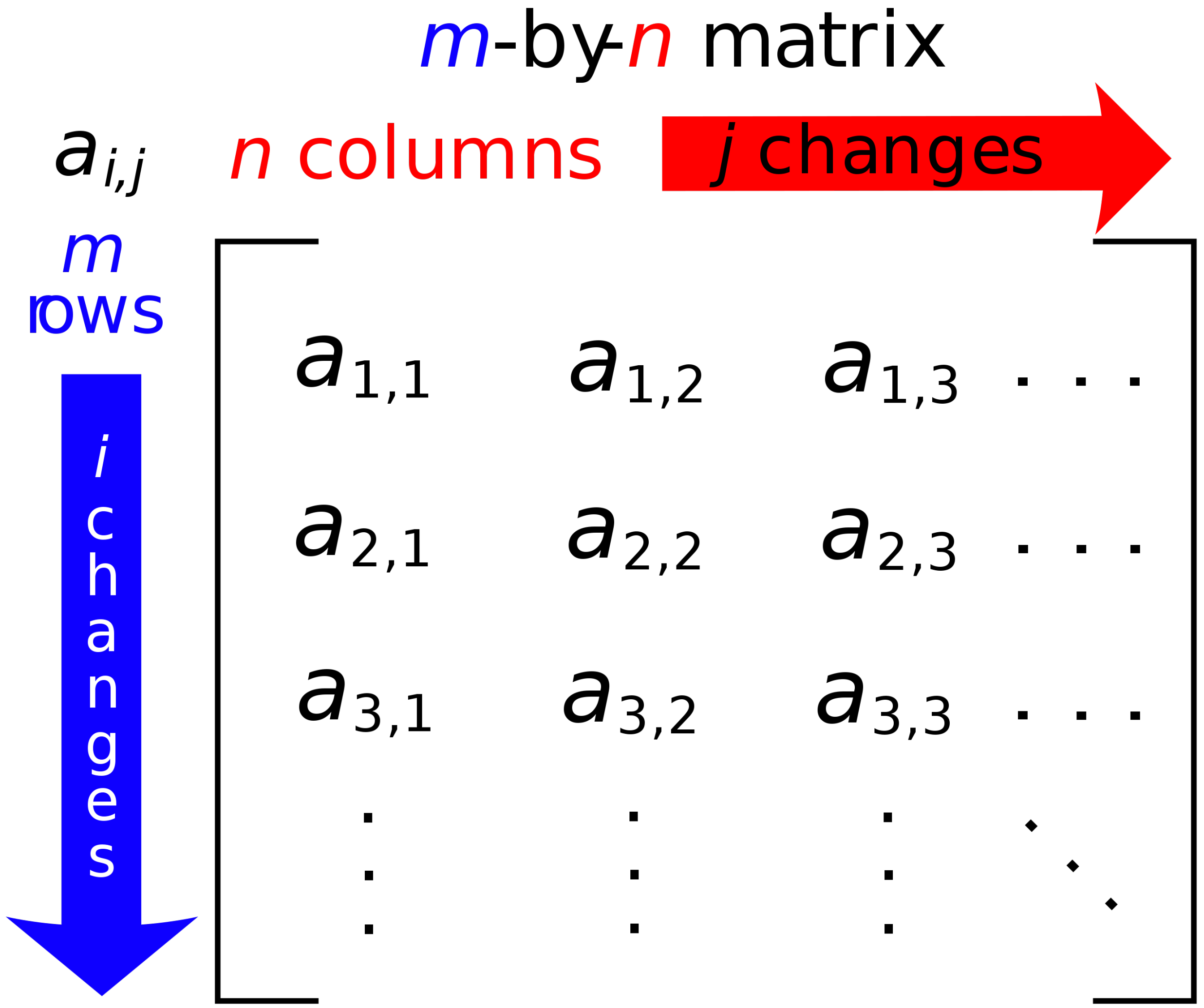
- So let's get comfy with matrices (next time).

THIS IS WHERE WE STOPPED LAST TIME

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



$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

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

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

One usage for matrices:

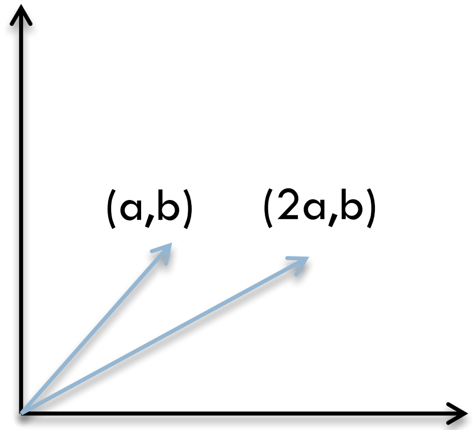
Let (a, b) be the coordinates of a point

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

Identity Matrix

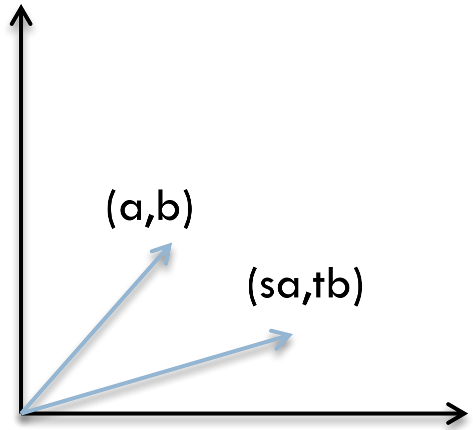
$$\begin{pmatrix} a & b \end{pmatrix} \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} a & b \end{pmatrix}$$

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



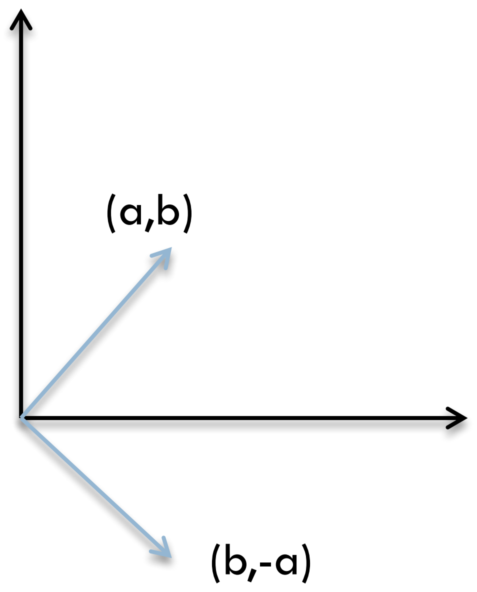
Scale in X, not in Y

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



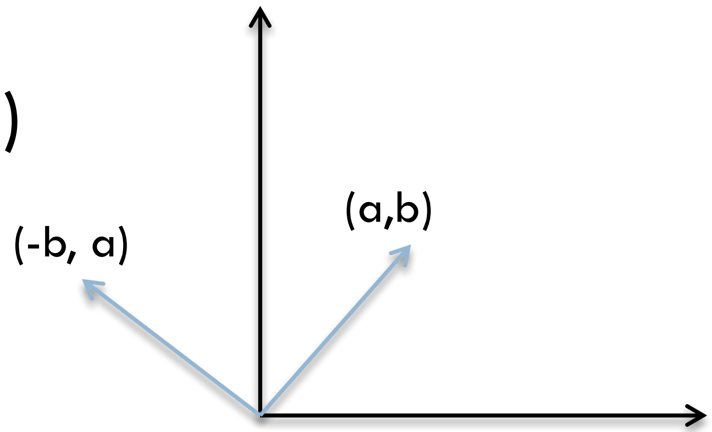
Scale in both dimensions

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



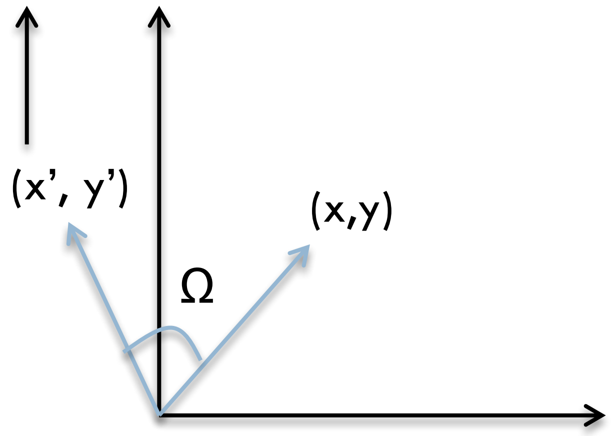
Rotate 90 degrees clockwise

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



Rotate 90 degrees counter-clockwise

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



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?

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

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

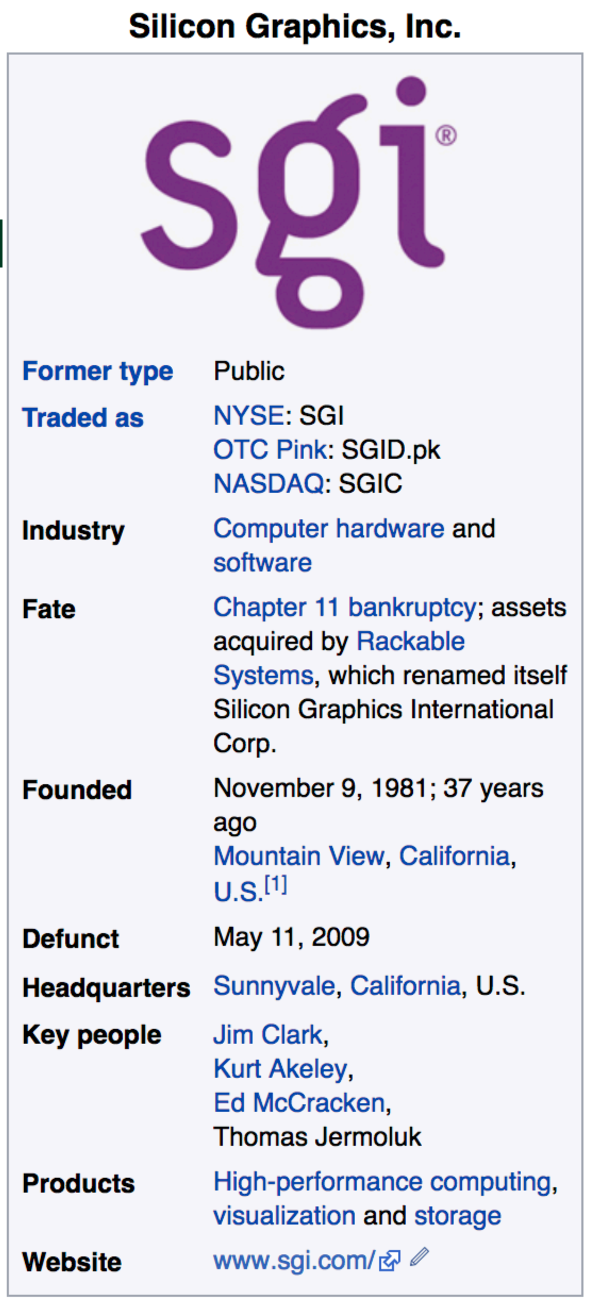
$$\begin{pmatrix} x & y & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{pmatrix} = \begin{pmatrix} x+dx & y+dy & 1 \end{pmatrix}$$

Translation

We can now fit translation into our matrix multiplication system.

Graphics

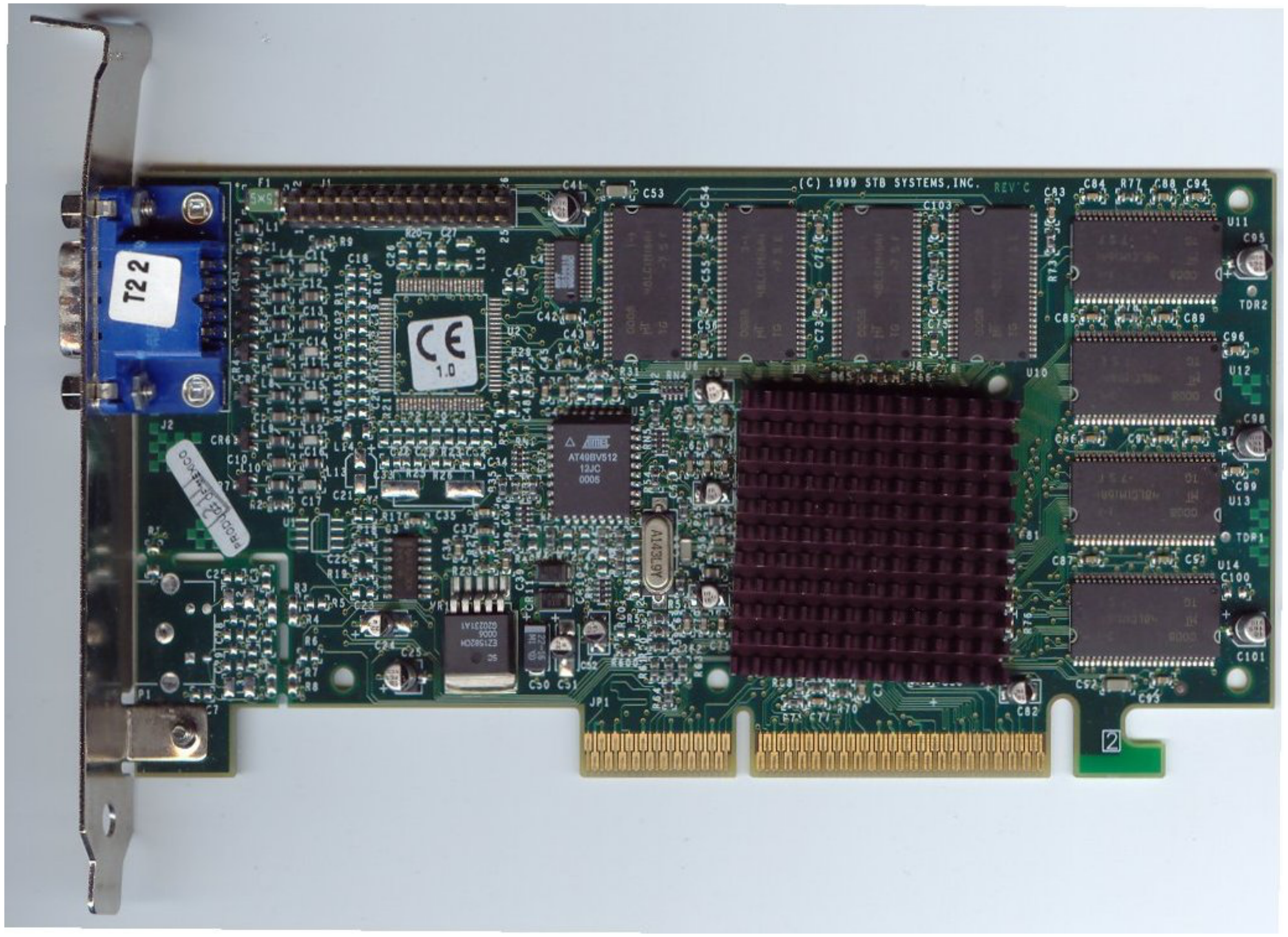
- Two really important operations:
  - Transform from one frame to another
  - Transform geometry (rotate, translate, etc)
- Both can be done with matrix operations
- In both cases, need homogeneous coordinates
- Much of graphics is accomplished via 4x4 matrices
  - And: you can compose the matrices 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)

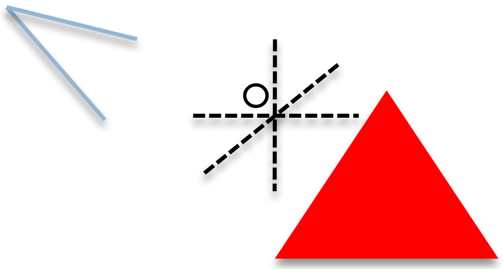
GPUs now

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

STOP HERE – rest on YouTube

- Look as a class

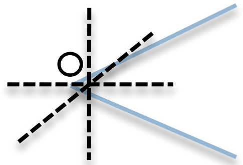
Our goal



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

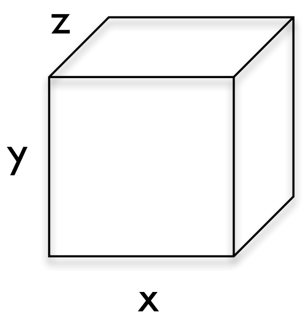
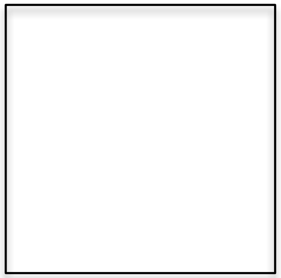


Image space:

All viewable objects within

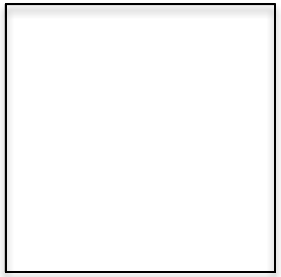
-1 <= x,y,z <= +1



Screen space:

All viewable objects within

-1 <= x, y <= +1



Device space:

All viewable objects within

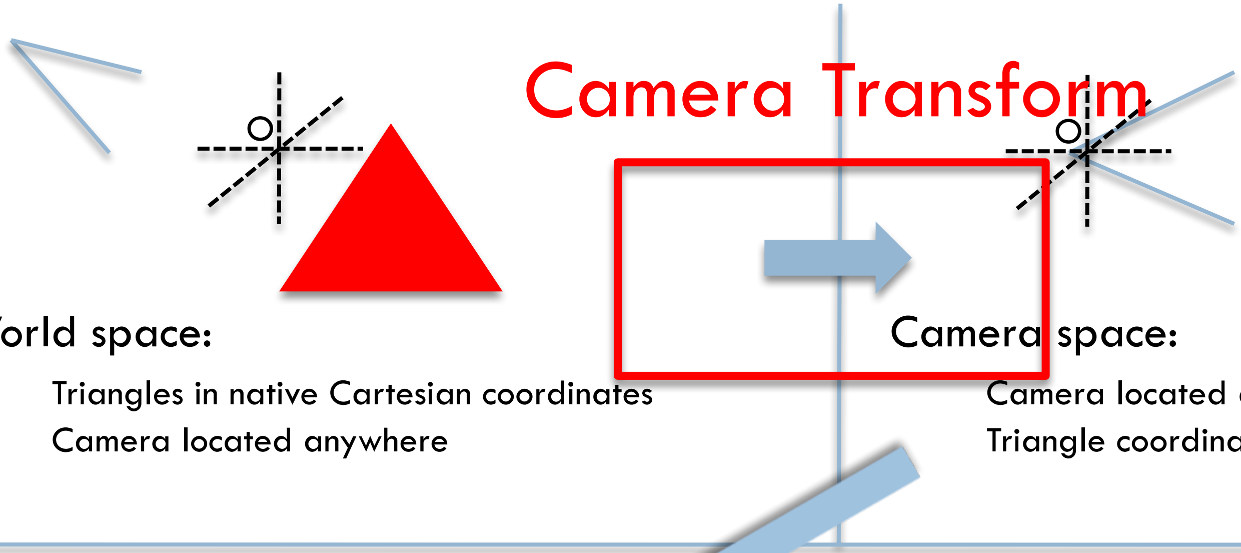
0<=x<=width, 0

<=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

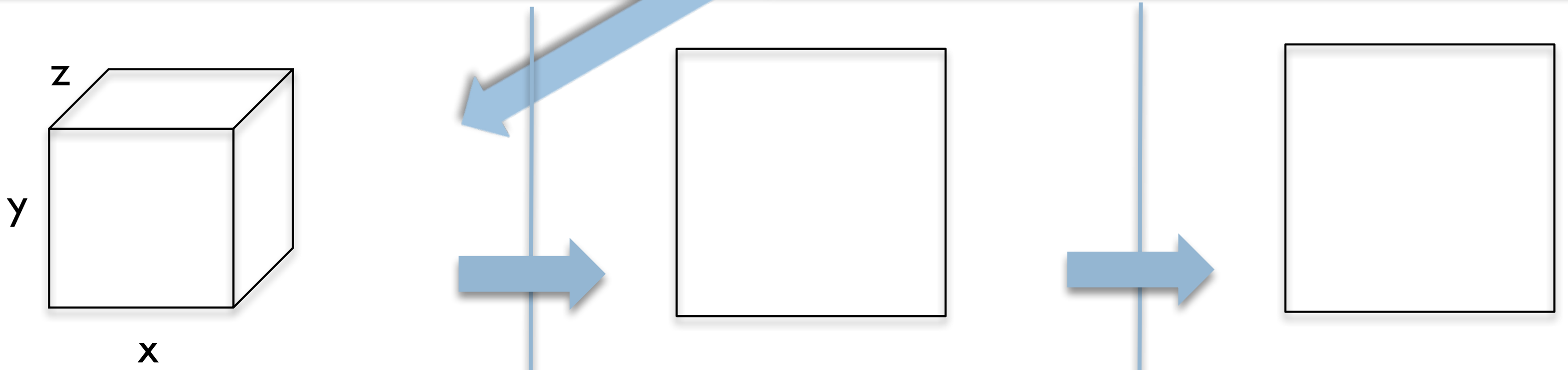


Image space:

All viewable objects within

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

Screen space:

All viewable objects within

-1 <= x, y <= +1

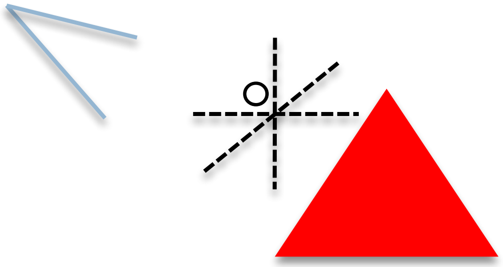
Device space:

All viewable objects within

0<=x<=width, 0

<=y<=height

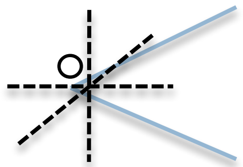
Our goal



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

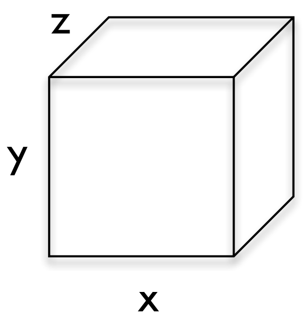
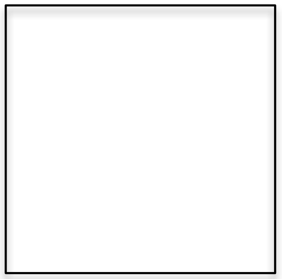


Image space:

All viewable objects within

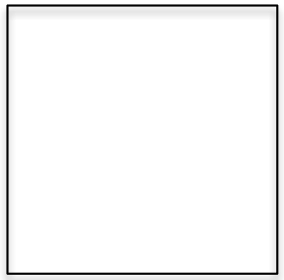
-1 <= x,y,z <= +1



Screen space:

All viewable objects within

-1 <= x, y <= +1



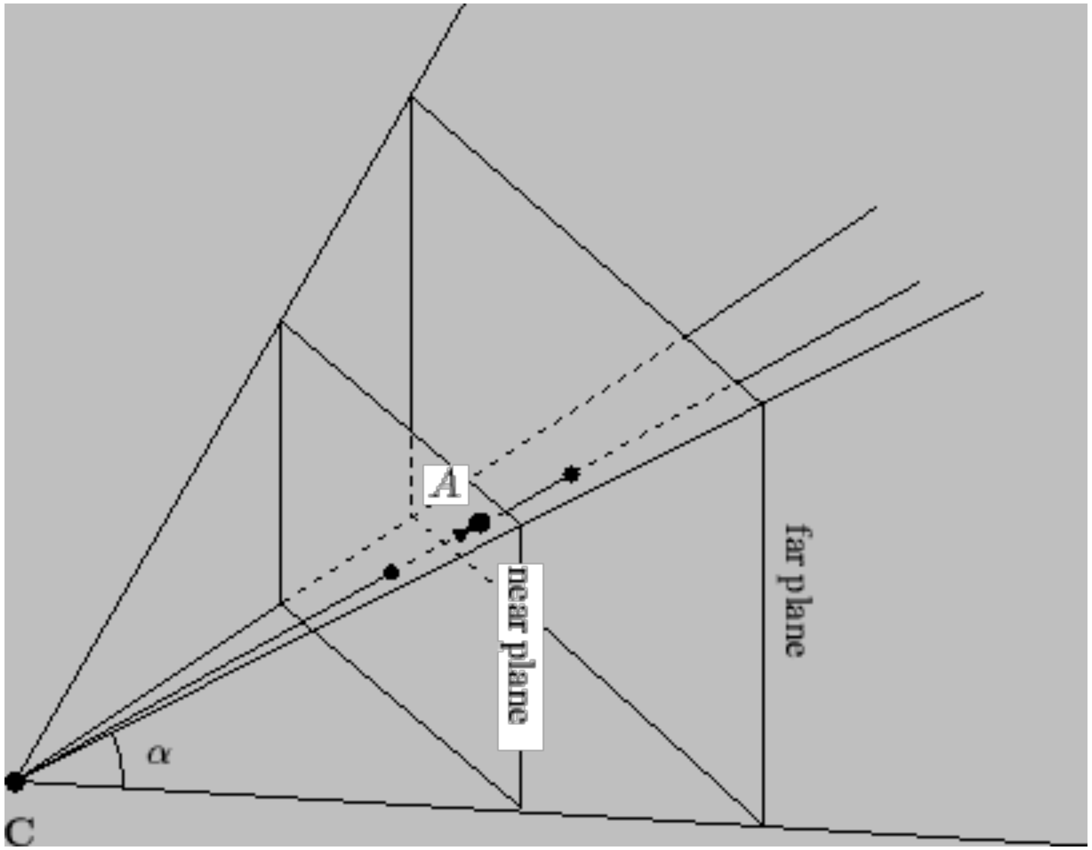
Device space:

All viewable objects within

0<=x<=width, 0

<=y<=height

How do we specify a camera?



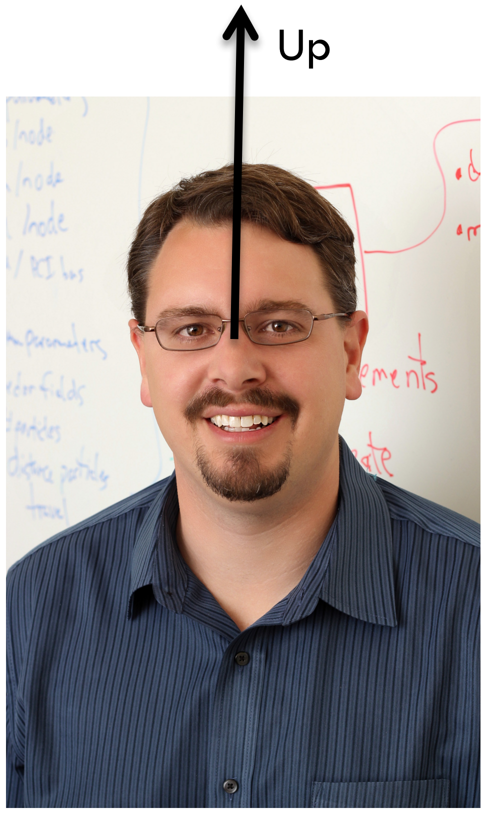
The “viewing pyramid” or “view frustum”.

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];
};
```

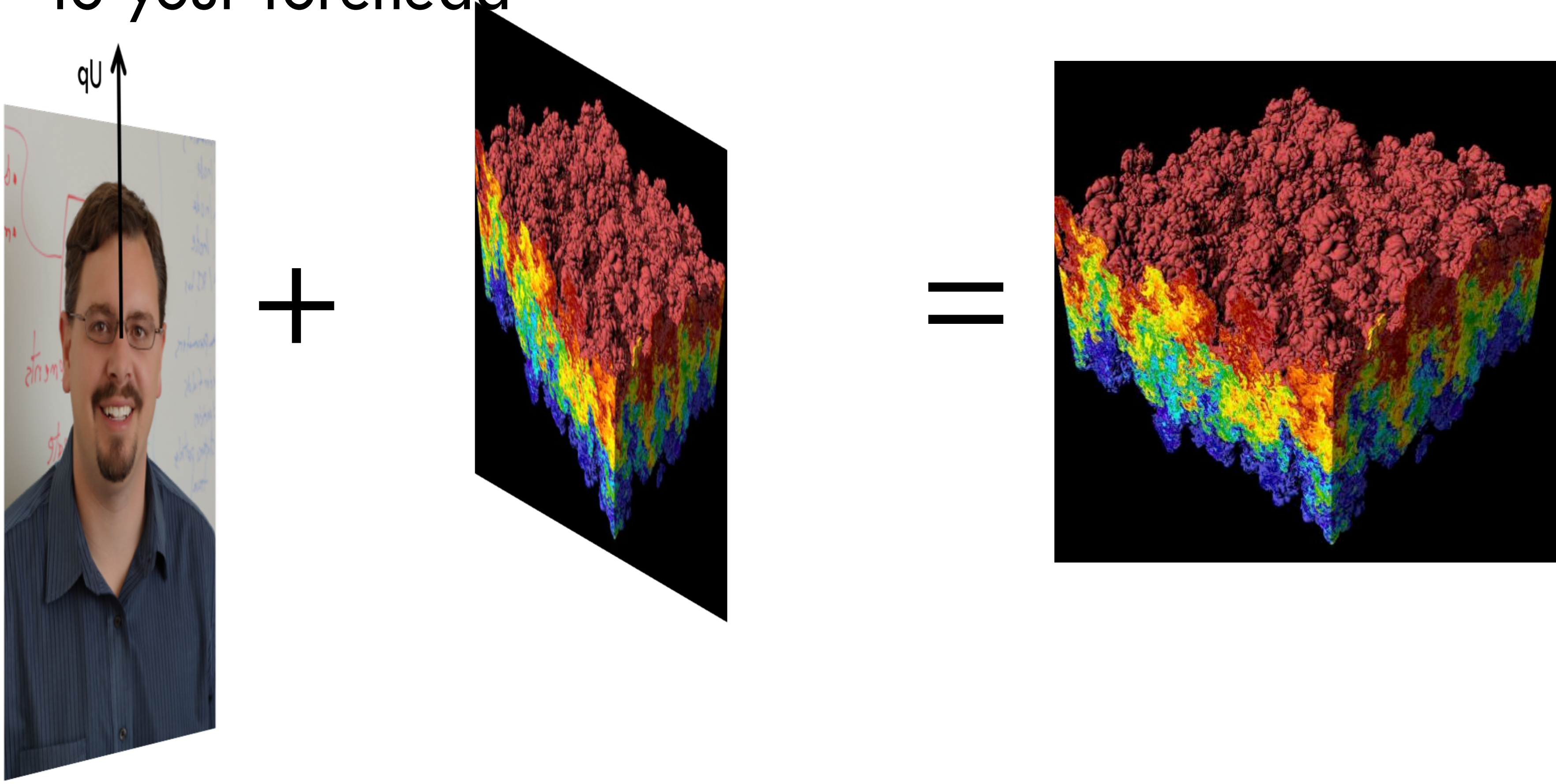
What is the up axis?

- Up axis is the direction from the base of your nose to your forehead



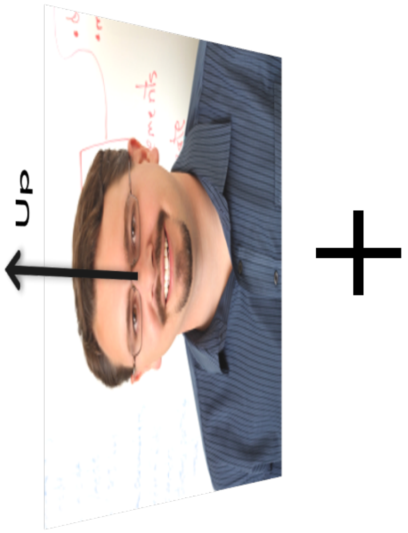
What is the up axis?

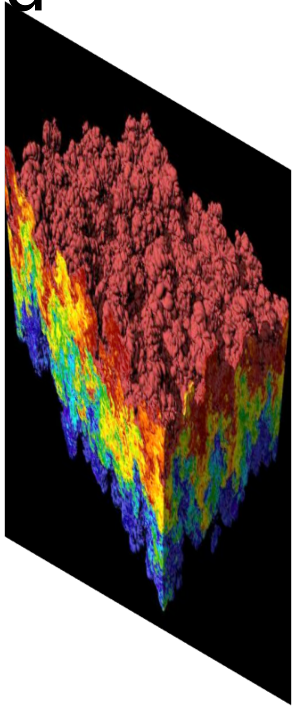
- Up axis is the direction from the base of your nose to your forehead



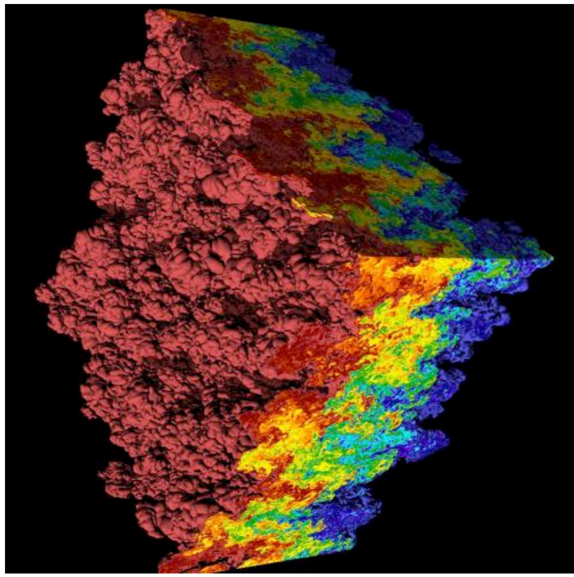
What is the up axis?

- Up axis is the direction from the base of your nose to your forehead



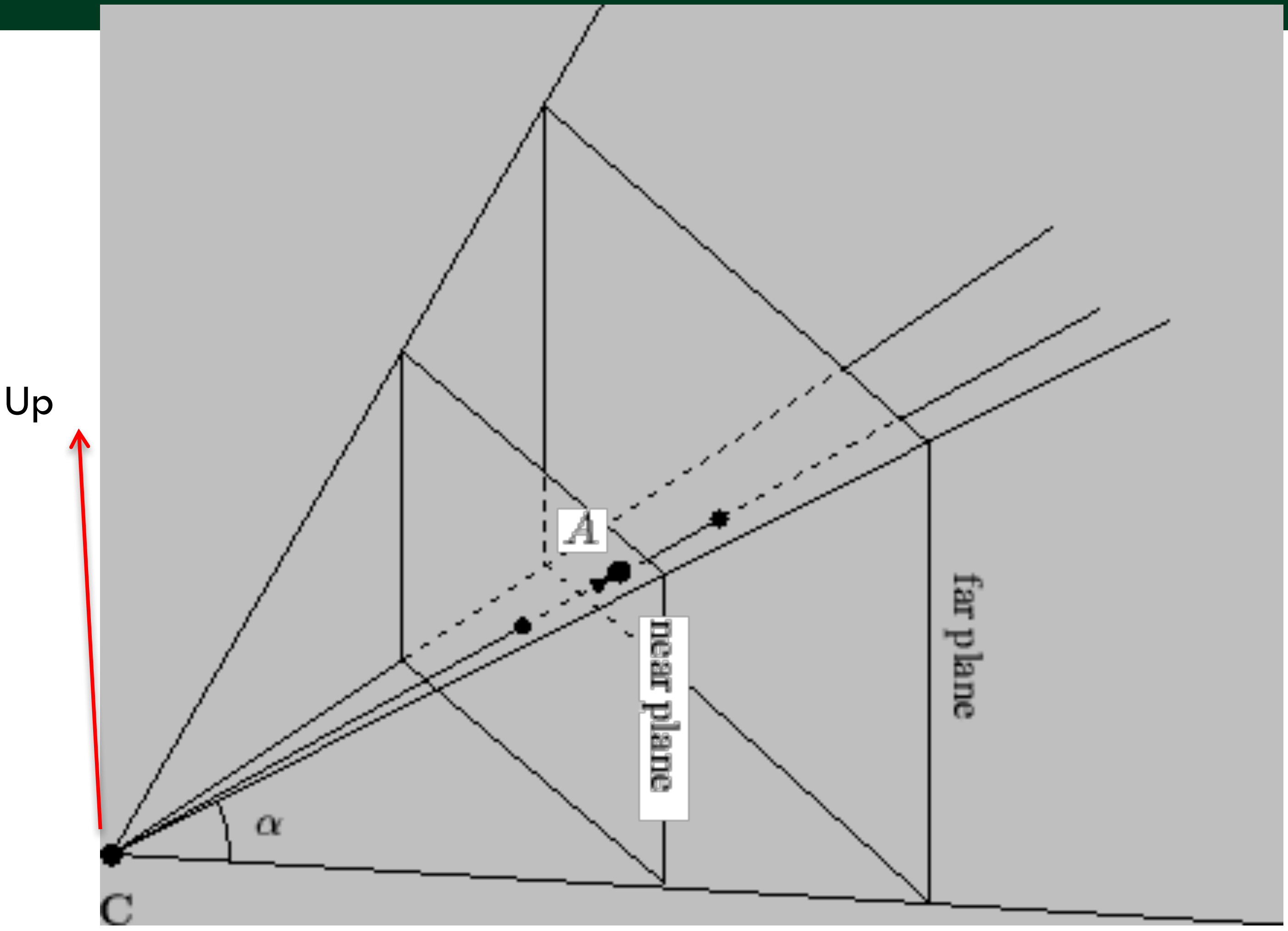


=

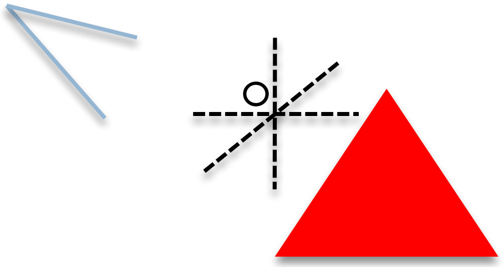


- (if you lie down while watching TV, the screen is sideways)

Image Space Diagram



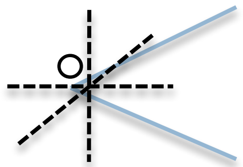
Our goal



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

View Transform

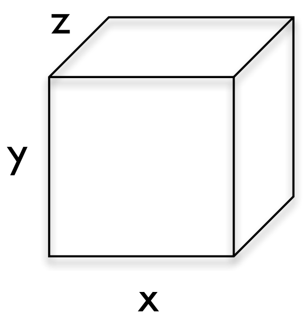
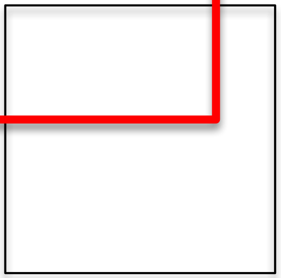


Image space:

All viewable objects within

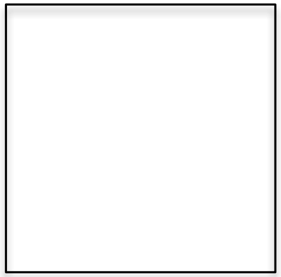
-1 <= x,y,z <= +1



Screen space:

All viewable objects within

-1 <= x, y <= +1



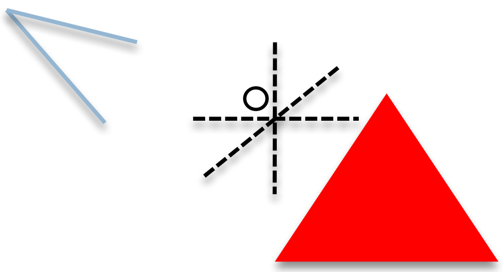
Device space:

All viewable objects within

0<=x<=width, 0

<=y<=height

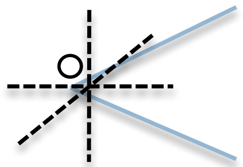
Our goal



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

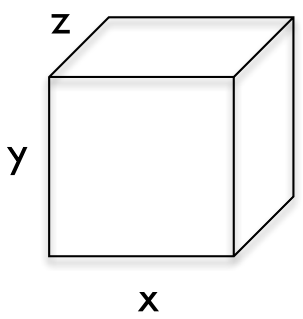
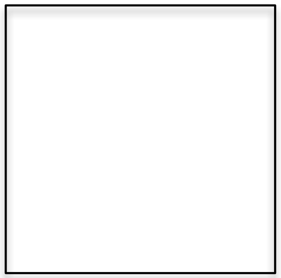


Image space:

All viewable objects within

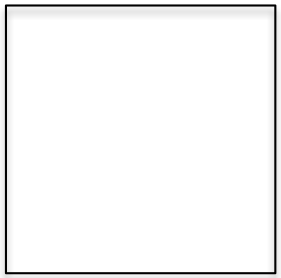
-1 <= x,y,z <= +1



Screen space:

All viewable objects within

-1 <= x, y <= +1



Device space:

All viewable objects within

0<=x<=width, 0

<=y<=height

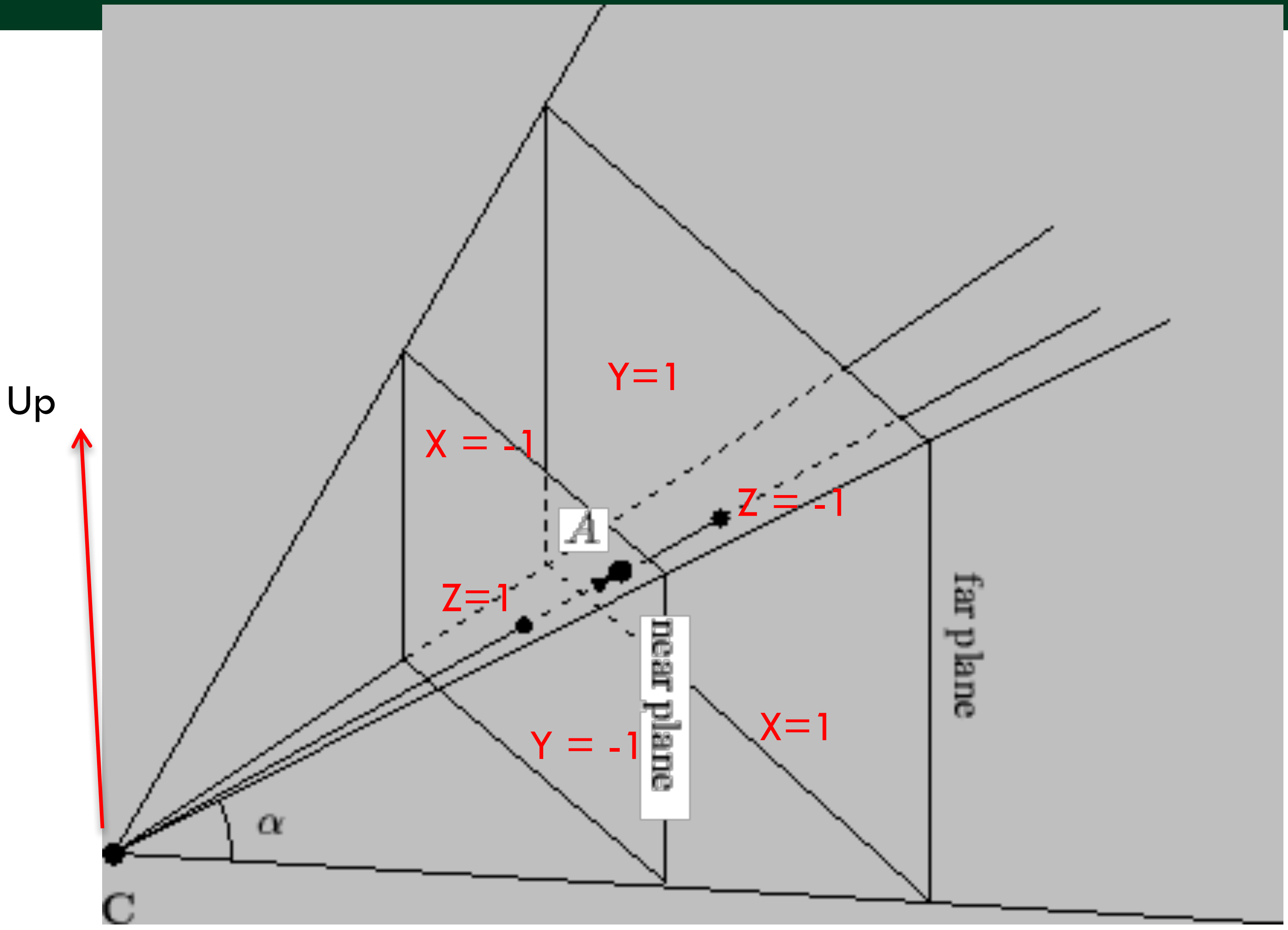
Image Space

- 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.

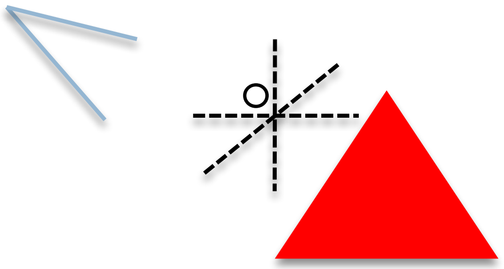
$$\{(x,y,z) : -1 \leq x \leq 1, -1 \leq y \leq 1, -1 \leq z \leq 1\}$$

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

Image Space Diagram



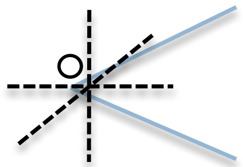
Our goal



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

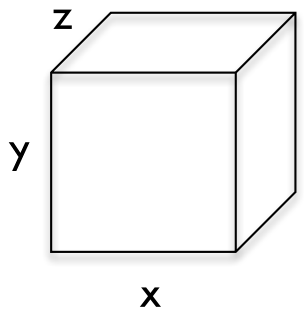
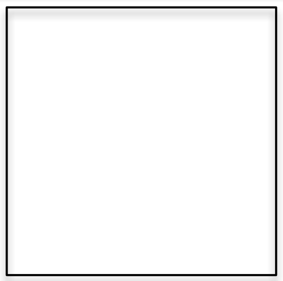


Image space:

All viewable objects within

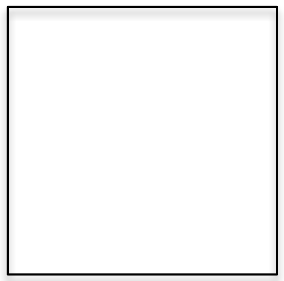
-1 <= x,y,z <= +1



Screen space:

All viewable objects within

-1 <= x, y <= +1



Device space:

All viewable objects within

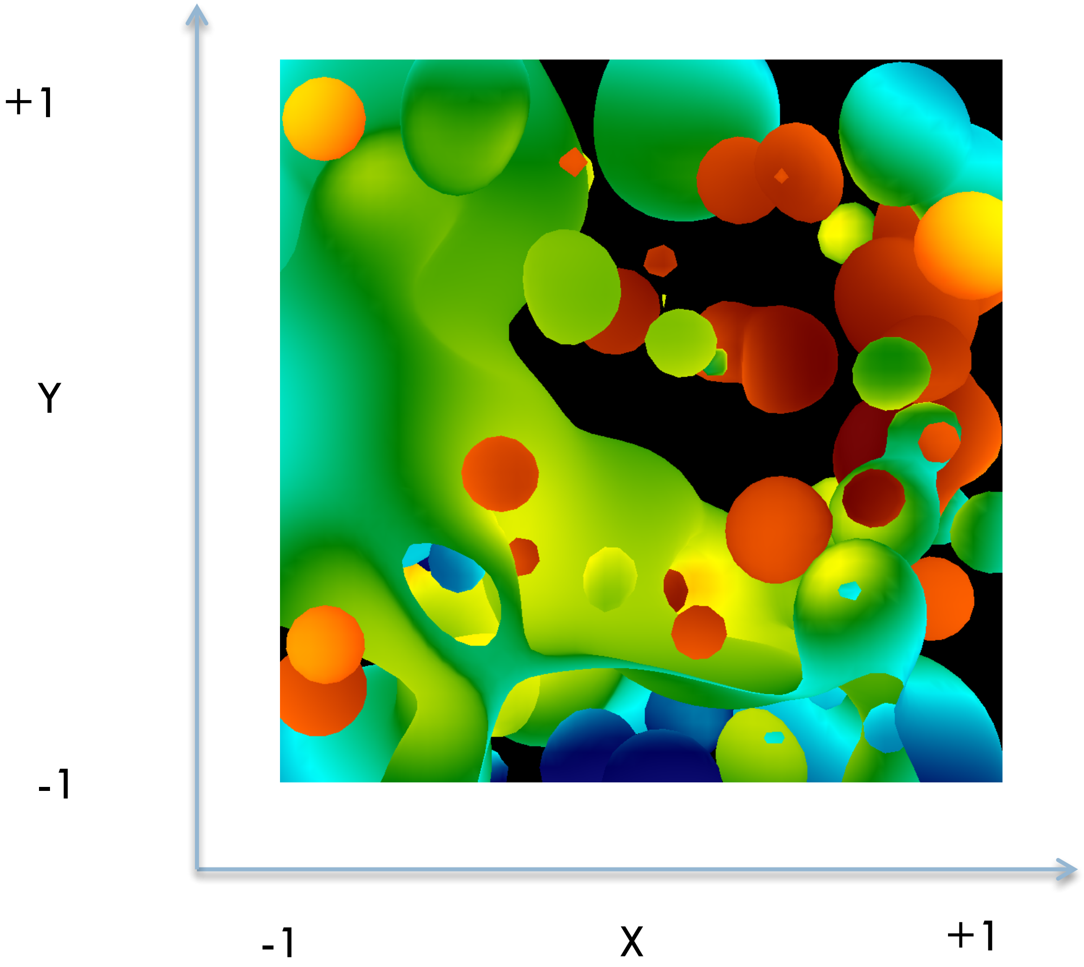
0<=x<=width, 0

<=y<=height

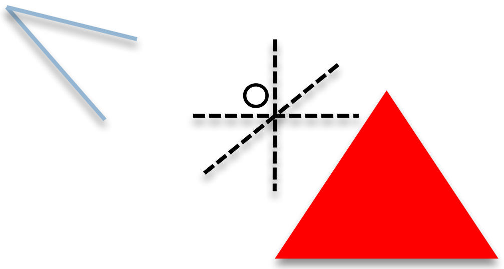
Screen Space

- 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



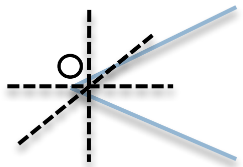
Our goal



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

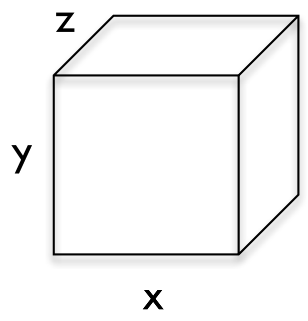
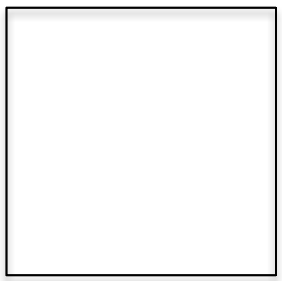


Image space:

All viewable objects within

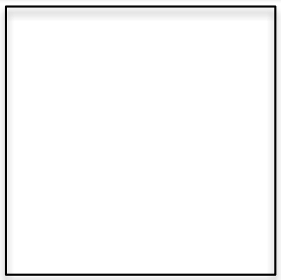
-1 <= x,y,z <= +1



Screen space:

All viewable objects within

-1 <= x, y <= +1



Device space:

All viewable objects within

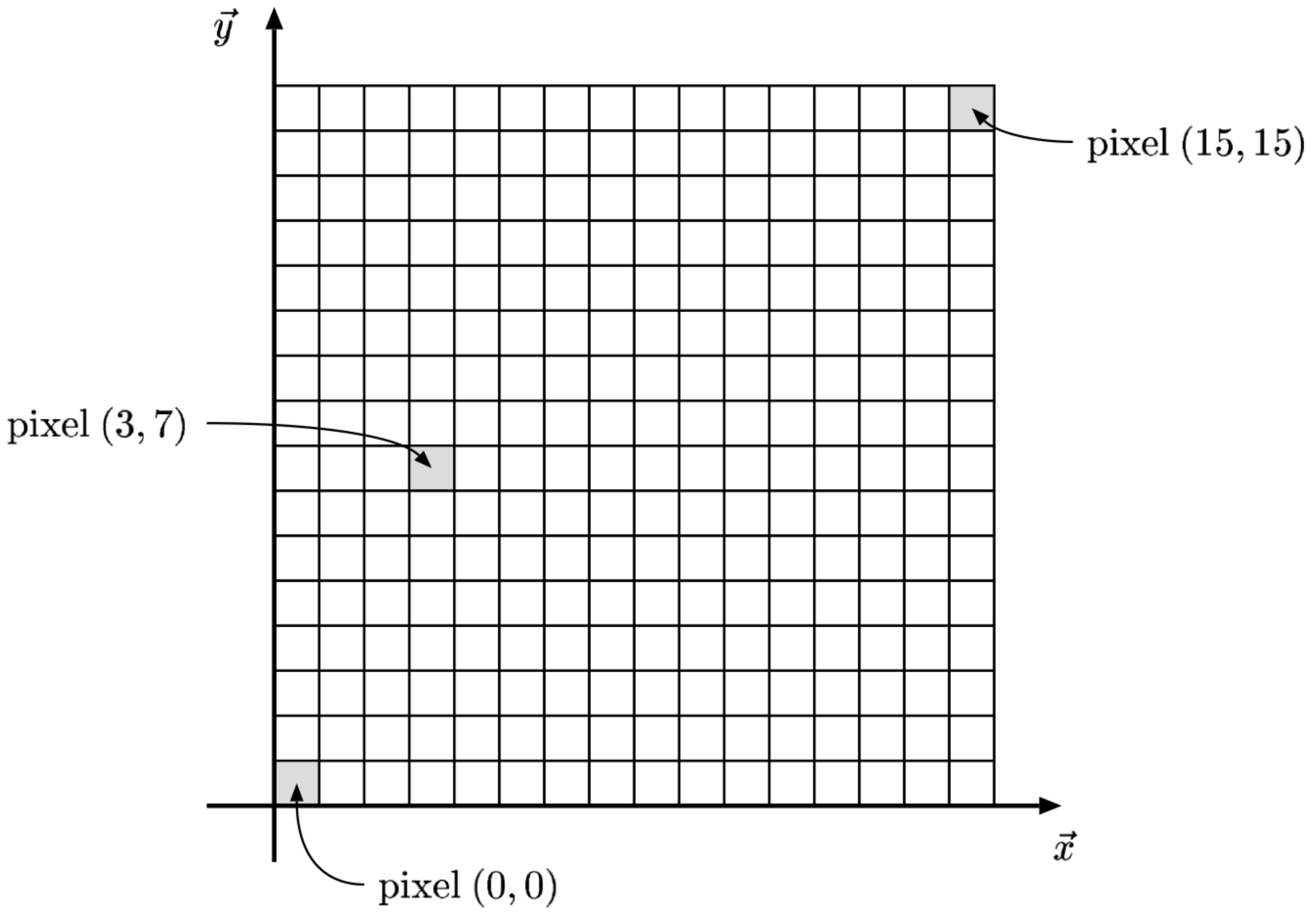
0<=x<=width, 0

<=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

$0 \leq x \leq n$

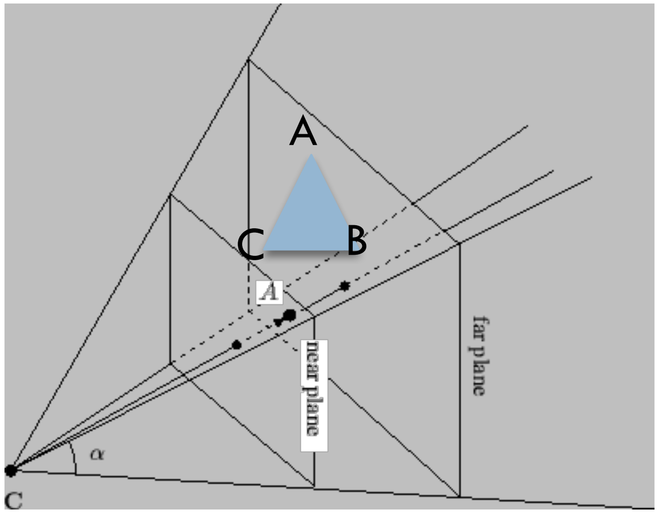
$0 \leq y \leq m$

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

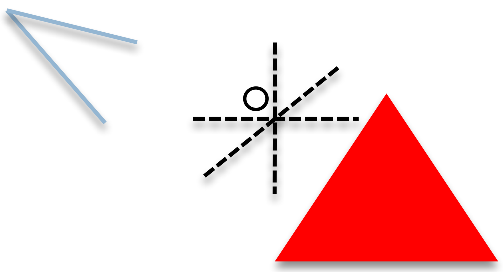
How do we transform?

- For a camera C,
  - Calculate Camera Frame
  - From Camera Frame, calculate Camera Transform
  - Calculate View Transform
  - Calculate Device Transform
  - Compose 3 Matrices into 1 Matrix (M)
- For each triangle T, apply M to each vertex of T, then apply rasterization/zbuffer

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



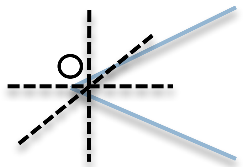
Easiest Transform



World space:

Triangles in native Cartesian coordinates

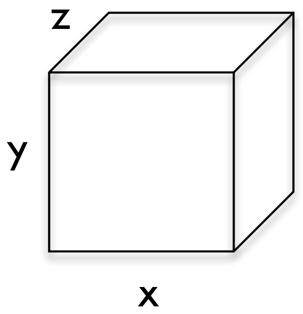
Camera located anywhere



Camera space:

Camera located at origin, looking down -Z

Triangle coordinates relative to camera frame



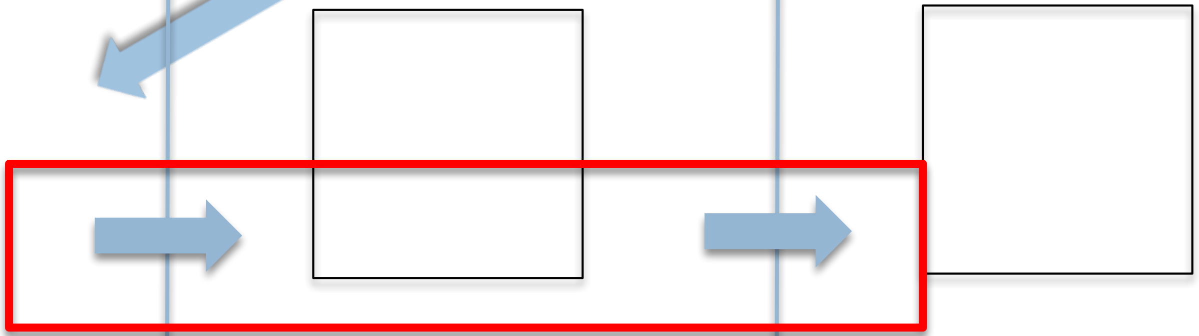


Image space:

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 = nx/2 + n/2
  - y' = m*(y+1)/2 = my/2 + m/2
  - z' = z = z
  - (for an n x m image)
- Matrix:

$$\begin{pmatrix} x & y & z & 1 \end{pmatrix} \times \begin{pmatrix} n/2 & 0 & 0 & 0 \\ 0 & m/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ n/2 & m/2 & 0 & 1 \end{pmatrix}$$

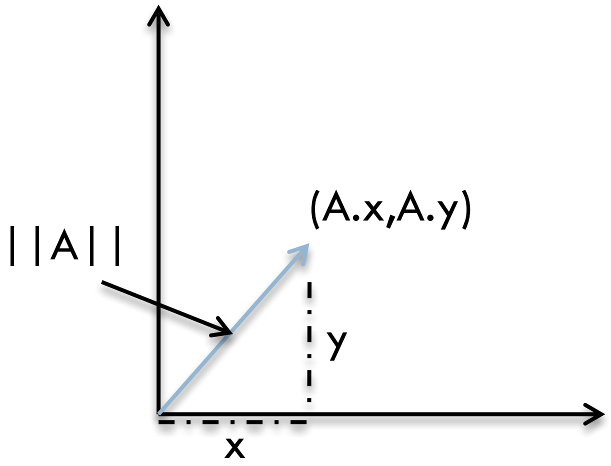
More Math Prep

Note: Ken Joy's graphics notes are fantastic http://www.idav.ucdavis.edu/education/GraphicsNotes/homepage.html

What is the norm of a vector?

- The norm of a vector is its length
  - Denoted with || · ||
- For a vector A = (A.x, A.y),

  ||A|| = sqrt(A.x*A.x+A.y*A.y)
- Physical interpretation:



- For 3D, ||A|| = sqrt(A.x*A.x+A.y*A.y+A.z*A.z)

What does it means for a vector to be normalized?

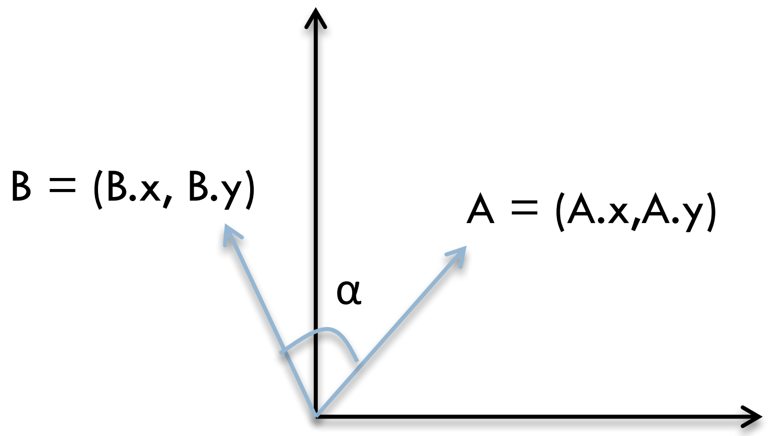
- The vector A is normalized if ||A|| = 1.
  - This is also called a unit vector.
- To obtain a normalized vector, take A/||A||

- Many of the operations we will discuss today will only work correctly with normalized vectors.

- Example: A=(3,4,0). Then:
  - ||A|| = 5
  - A/||A|| = (0.6, 0.8, 0)

What is a dot product?

- A·B = A.x*B.x + A.y*B.y
  - (or A.x*B.x + A.y*B.y + A.z*B.z)
- Physical interpretation:
  - A·B = cos(α)*(||A||*||B||)



What is the cross product?

- AxB = (A.y*B.z - A.z*B.y,
  B.x*A.z - A.x*B.z,
  A.x*B.y - A.y*B.x)

- What is the physical interpretation of a cross product?
  - Finds a vector perpendicular to both A and B.

Homogeneous Coordinates

- Defined: a system of coordinates used in projective geometry, as Cartesian coordinates are used in Euclidean geometry
- Primary uses:
  - 4×4 matrices to represent general 3-dimensional transformations
  - it allows a simplified representation of mathematical functions – the rational form – in which rational polynomial functions can be simply represented
- We only care about the first
  - I don't really even know what the second use means

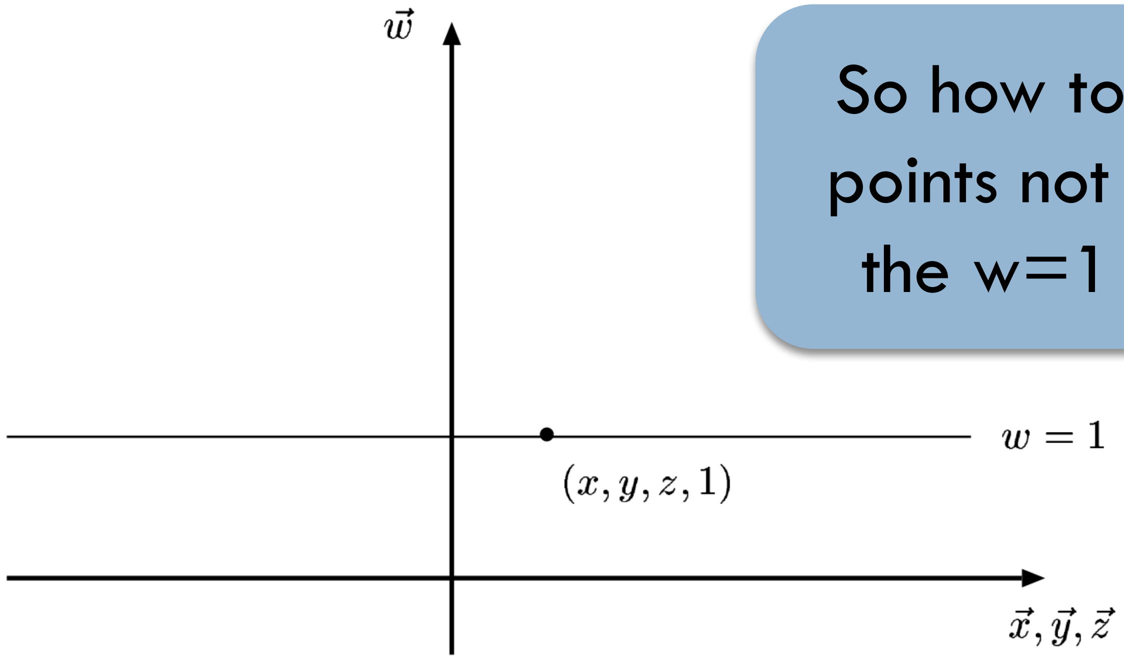
Interpretation of Homogeneous Coordinates

- 4D points: (x, y, z, w)
- Our typical frame: (x, y, z, 1)

Interpretation of Homogeneous Coordinates

- 4D points: (x, y, z, w)
- Our typical frame: (x, y, z, 1)

So how to treat points not along the w=1 line?



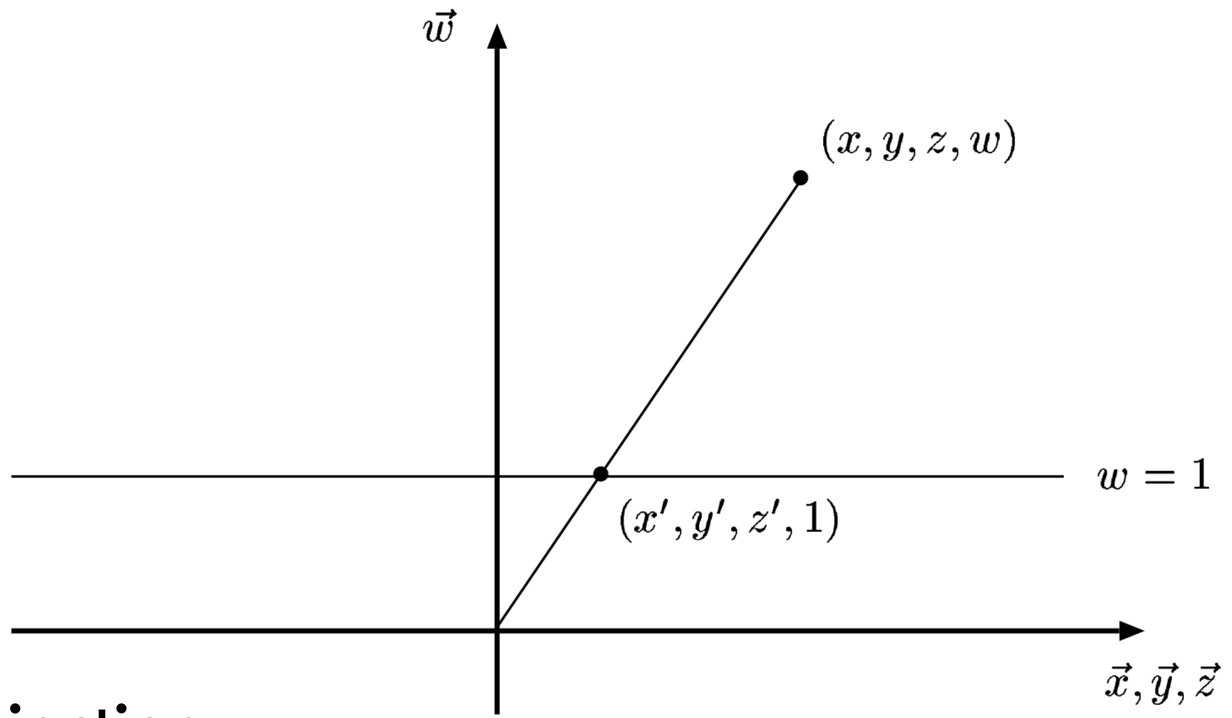
Our typical frame in the context of 4D points

Projecting back to w=1 line

- Let P = (x, y, z, w) be a 4D point with w != 1
- Goal: find P' = (x', y', z', 1) such P projects to P'
  - (We have to define what it means to project)

- Idea for projection:
  - Draw line from P to origin.
  - If Q is along that line (and Q.w == 1), then Q is a projection of P

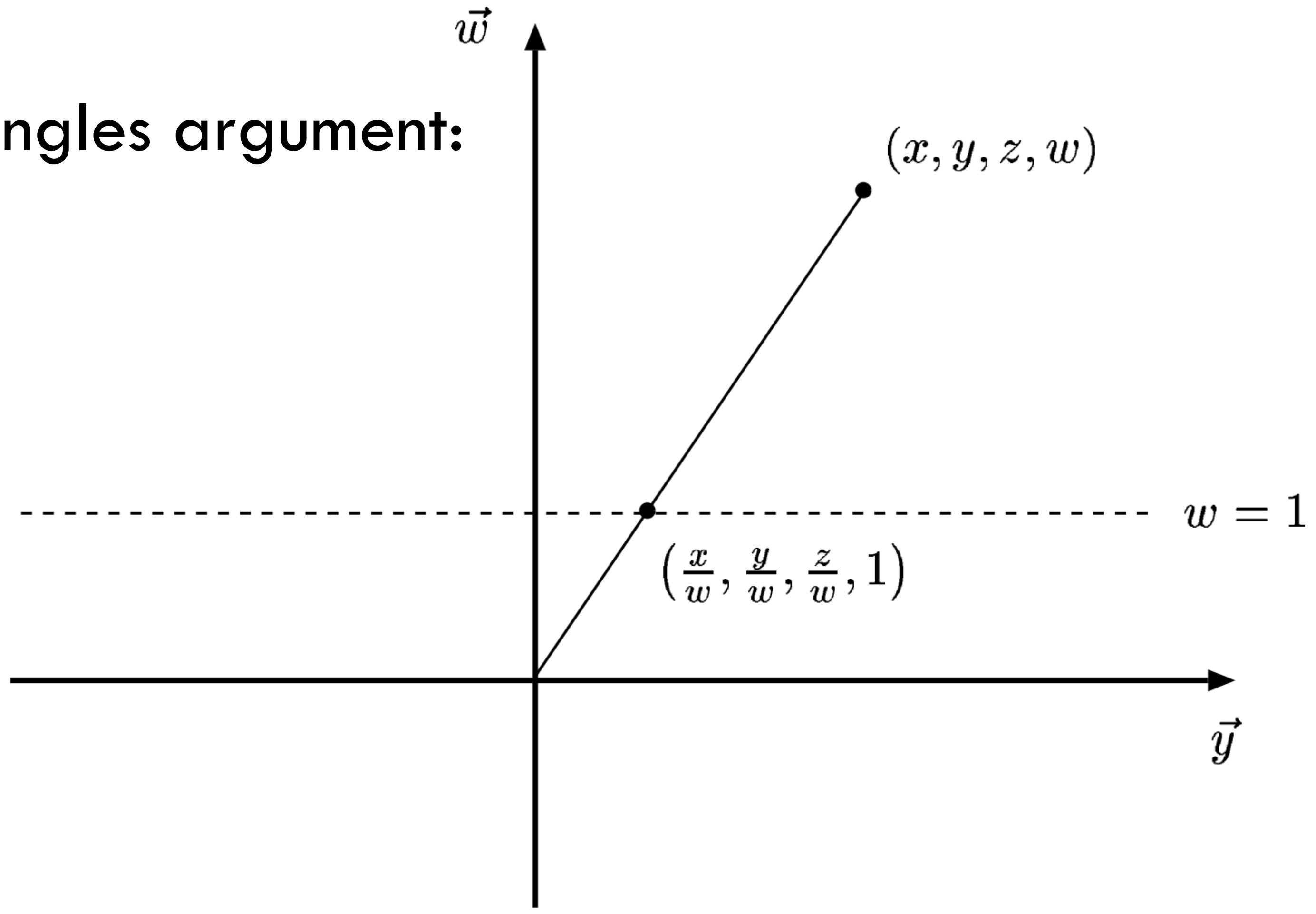
Projecting back to w==1 line



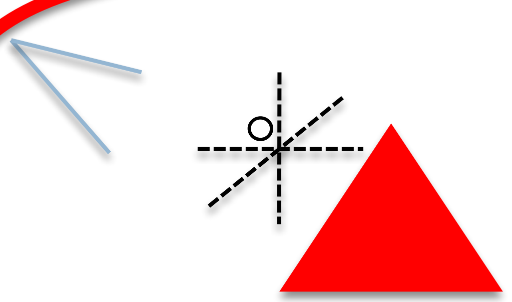
- Idea for projection:
  - Draw line from P to origin.
  - If Q is along that line (and Q.w == 1), then Q is a projection of P

So what is Q?

- Similar triangles argument:
  - x' = x/w
  - y' = y/w
  - z' = z/w

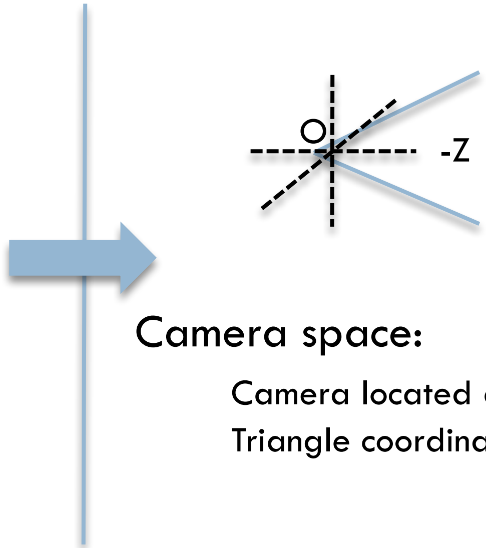


Our goal



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

- 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

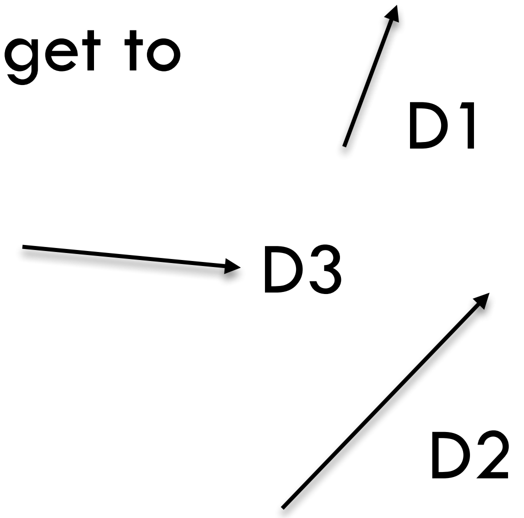
Basis pt 2
(more linear algebra-y this time)

- Camera frame must be a basis:
  - Spans space … can get any point through a linear combination of basis vectors
  - Every member must be linearly independent
    - → we didn't talk about this much on Thursday.
    - linearly independent means that no basis vector can be represented via others
    - Repeat slide (coming up) shows linearly *dependent* vectors

(REPEAT) Why unique?

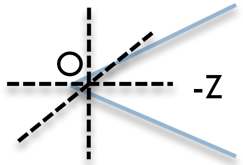
- Let (a, b, c) mean:
  - The number of steps ‘a’ in direction D1
  - The number of steps ‘b’ in direction D2
  - The number of steps ‘c’ in direction D3

- 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



Camera frame construction

- Must choose (u,v,w,O)



Camera space:

Camera located at origin, looking down -Z

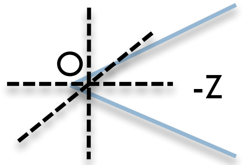
Triangle coordinates relative to camera frame

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

- O = camera position
- w = O-focus
  - Not "focus-O", since we want to look down -Z

Camera frame construction

- Must choose (u,v,w,O)



Camera space:

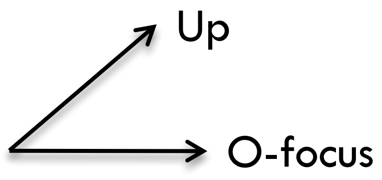
Camera located at origin, looking down -Z

Triangle coordinates relative to camera frame

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

- O = camera position
- w = O-focus
- v = up
- u = up x (O-focus)

But wait … what if dot(v2,v3) != 0?



- We can get around this with two cross products:
  - u = Up x (O-focus)
  - v = (O-focus) x u

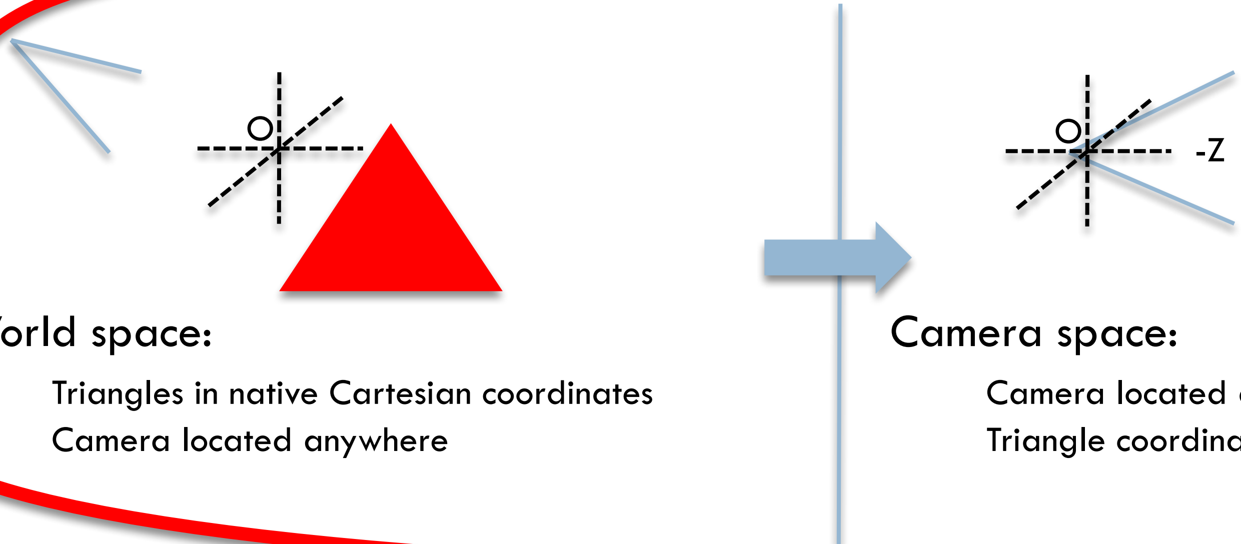
Camera frame summarized

- O = camera position
- u = Up x (O-focus)
- v = (O-focus) x u
- w = O-focus

- Important note:
  u, v, and w need to be normalized!

```
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

- Need to construct a Camera Frame ← ✔
- Need to construct a matrix to transform points from Cartesian Frame to Camera Frame
  - Transform triangle by transforming its three vertices

This Will Come Up Later

- Consider the meaning of Cartesian coordinates (x,y,z):

[x y z 1][<1,0,0>]

[<0,1,0>] = (x,y,z)

[<0,0,1>]

[(0,0,0)]

The Two Frames of the Camera Transform

- Our two frames:

- Cartesian:
  - <1,0,0>
  - <0,1,0>
  - <0,0,1>
  - (0,0,0)

- Camera:
  - u = up x (O-focus)
  - v = (O-focus) x u
  - w = (O-focus)
  - O

The Two Frames of the Camera Transform

- Our two frames:

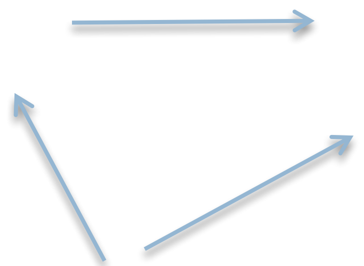
- Cartesian:
  - <1,0,0>
  - <0,1,0>
  - <0,0,1>
  - (0,0,0)

- Camera:
  - u = up x (O-focus)
  - v = (O-focus) x u
  - w = (O-focus)
  - O

The “Camera Frame” is a Frame, so we can express any Cartesian vector as a combination of u, v, w.

Converting From Cartesian Frame To Camera Frame

- The Cartesian vector <1,0,0> can be represented as some combination of the Camera Frame's basis functions u, v, w:
  - <1,0,0> = e1,1 * u + e1,2 * v + e1,3 * w
- So can the Cartesian vector <0,1,0>
  - <0,1,0> = e2,1 * u + e2,2 * v + e2,3 * w
- So can the Cartesian vector <0,0,1>
  - <0,0,1> = e3,1 * u + e3,2 * v + e3,3 * w
- So can the vector: Cartesian Frame origin – Camera Frame origin
  - (0,0,0) - O = e4,1 * u + e4,2 * v + e4,3 * w →
  - (0,0,0) = e4,1 * u + e4,2 * v + e4,3 * w + O



Putting Our Equations Into Matrix Form

- <1,0,0> = e1,1 * u + e1,2 * v + e1,3 * w
- <0,1,0> = e2,1 * u + e2,2 * v + e2,3 * w
- <0,0,1> = e3,1 * u + e3,2 * v + e3,3 * w
- (0,0,0) = e4,1 * u + e4,2 * v + e4,3 * w + O
- →
- [<1,0,0>]   [e1,1   e1,2   e1,3   0] [u]
- [<0,1,0>]   [e2,1   e2,2   e2,3   0] [v]
- [<0,0,1>]   = [e3,1   e3,2   e3,3   0] [w]
- (0,0,0)   [e4,1   e4,2   e4,3   1] [O]

Here Comes The Trick…

- Consider the meaning of Cartesian coordinates (x,y,z):

```
[x  y  z 1][<1,0,0>]
           [<0,1,0>]   = (x,y,z)
           [<0,0,1>]
           [(0,0,0)]
```

But:

```
[<1,0,0>]      [e1,1   e1,2   e1,3   0] [u]
[<0,1,0>]      [e2,1   e2,2   e2,3   0] [v]
[<0,0,1>]   =  [e3,1   e3,2   e3,3   0] [w]
(0,0,0)        [e4,1   e4,2   e4,3   1] [O]
```

Here Comes The Trick…

But:

$$\boxed{[x\ y\ z\ 1]}\begin{bmatrix}\langle 1,0,0\rangle\\ \langle 0,1,0\rangle\\ \langle 0,0,1\rangle\\ (0,0,0)\end{bmatrix} = \boxed{[x\ y\ z\ 1]\begin{bmatrix} e_{1,1} & e_{1,2} & e_{1,3} & 0\\ e_{2,1} & e_{2,2} & e_{2,3} & 0\\ e_{3,1} & e_{3,2} & e_{3,3} & 0\\ e_{4,1} & e_{4,2} & e_{4,3} & 1\end{bmatrix}}\begin{bmatrix}u\\ v\\ w\\ O\end{bmatrix}$$

Coordinates of (x,y,z) with respect to Cartesian frame.

Coordinates of (x,y,z) with respect to Camera frame.
<u>So this matrix is the camera transform!</u>!

And Cramer's Rule Can Solve This, For Example…

$$e_{1,1} = \frac{(<1,0,0> \times \vec{v}) \cdot \vec{w}}{(\vec{u} \times \vec{v}) \cdot \vec{w}} \quad ,$$

$$e_{1,2} = \frac{(\vec{u} \times <1,0,0>) \cdot \vec{w}}{(\vec{u} \times \vec{v}) \cdot \vec{w}} \quad , \text{and}$$

$$e_{1,3} = \frac{(\vec{u} \times \vec{v}) \cdot <1,0,0>}{(\vec{u} \times \vec{v}) \cdot \vec{w}}$$

(u == v1, v == v2, w == v3 from previous slide)

Solving the Camera Transform

$$\begin{bmatrix} e1,1 & e1,2 & e1,3 & 0 \\ e2,1 & e2,2 & e2,3 & 0 \\ e3,1 & e3,2 & e3,3 & 0 \\ e4,1 & e4,2 & e4,3 & 1 \end{bmatrix} = \begin{bmatrix} u.x & v.x & w.x & 0 \\ u.y & v.y & w.y & 0 \\ u.z & v.z & w.z & 0 \\ u \cdot t & v \cdot t & w \cdot t & 1 \end{bmatrix}$$

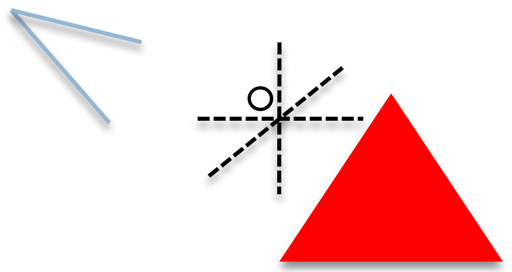
Where t = (0,0,0)-O

How do we know?: Cramer's Rule + simplifications

Want to derive?:

http://www.idav.ucdavis.edu/education/GraphicsNotes/Camera-Transform/Camera-Transform.html

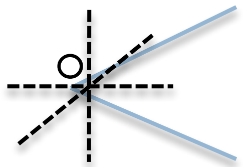
Our goal



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

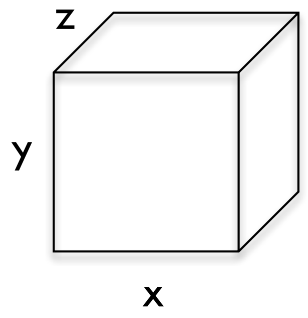
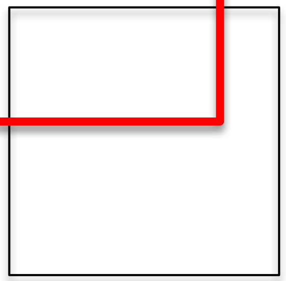


Image space:

All viewable objects within

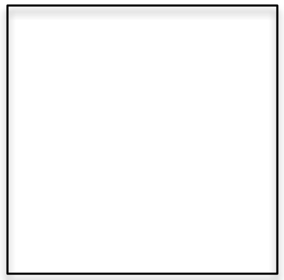
-1 <= x,y,z <= +1



Screen space:

All viewable objects within

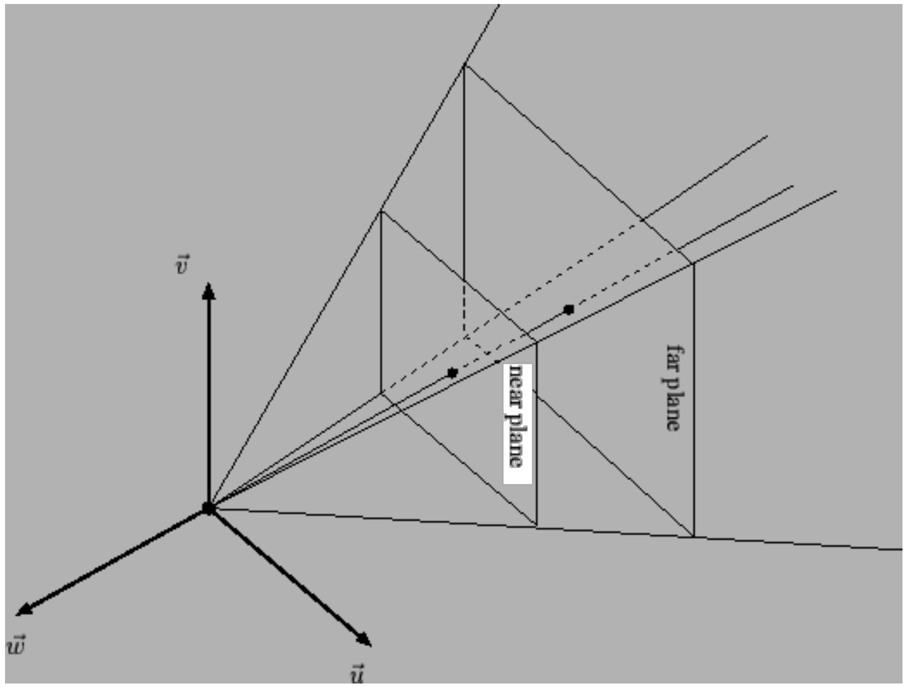
-1 <= x, y <= +1



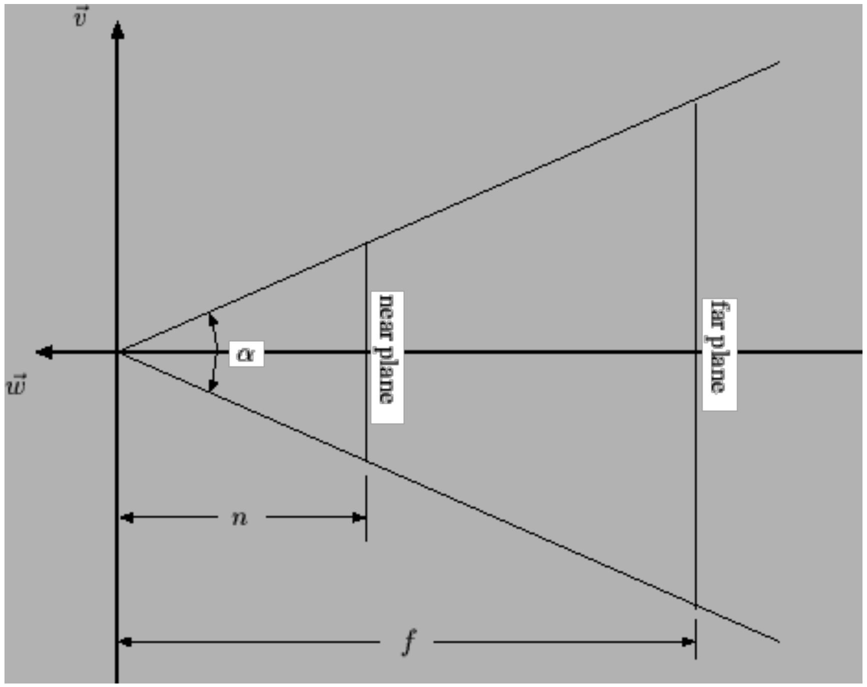
Device space:

All viewable objects within

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

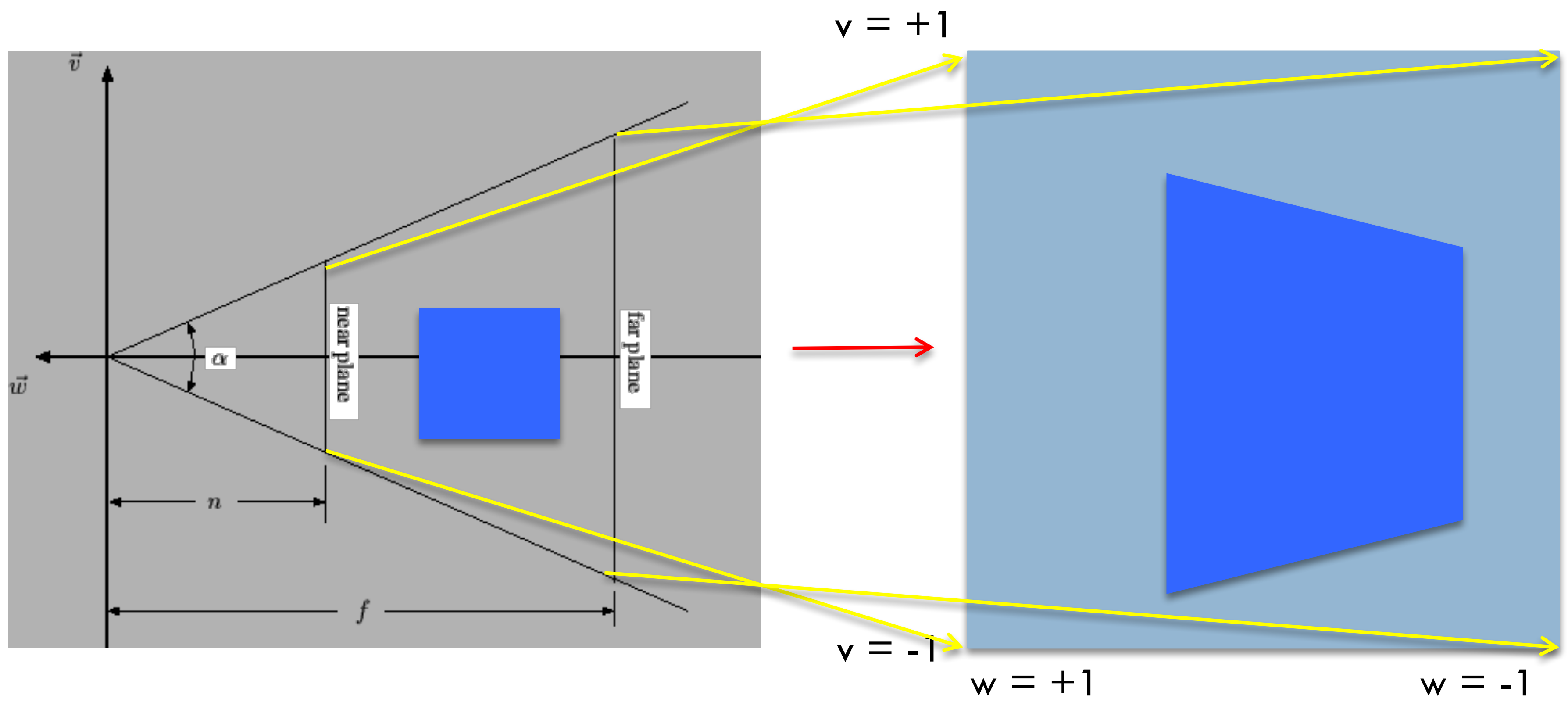


$\vec{v}$
near plane
far plane
$\vec{w}$
$\vec{u}$



$\vec{v}$
$\vec{w}$
α
near plane
far plane
n
f

View Transformation



The viewing transformation is not a combination of simple translations, rotations, scales or shears: it is more complex.

Derivation of Viewing Transformation

- I personally don't think it is a good use of class time to derive this matrix.
- Well derived at:
  - http://www.idav.ucdavis.edu/education/GraphicsNotes/Viewing-Transformation/Viewing-Transformation.html

The View Transformation

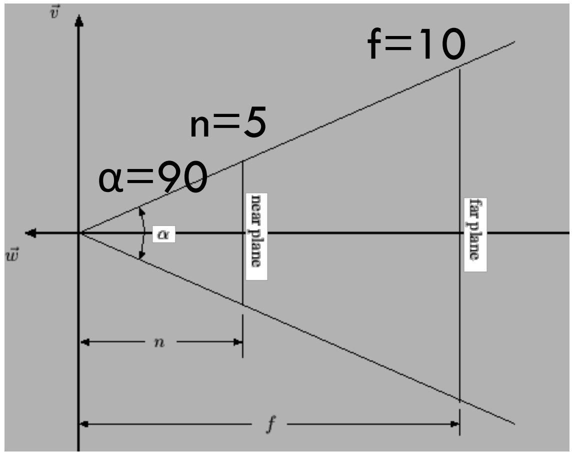
- Input parameters: (α, n, f)
- Transforms view frustum to image space cube
  - View frustum: bounded by viewing pyramid and planes z=-n and z=-f
  - Image space cube: -1 <= u,v,w <= 1

$$\begin{array}{llll}
[\cot(\alpha/2) & 0 & 0 & 0] \\
[0 & \cot(\alpha/2) & 0 & 0] \\
[0 & 0 & (f+n)/(f-n) & -1] \\
[0 & 0 & 2fn/(f-n) & 0]
\end{array}$$

  - Cotangent = 1/tangent

Let's do an example

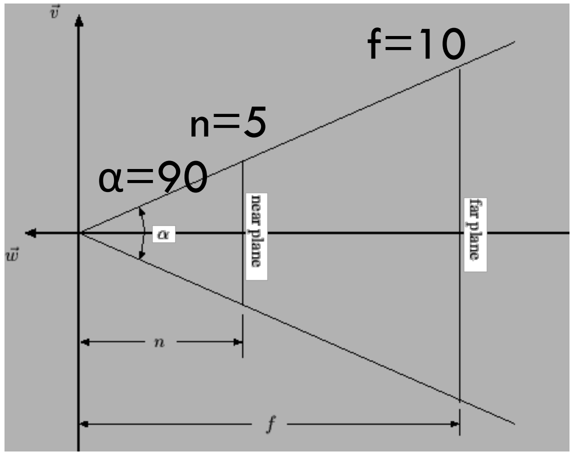
- Input parameters: (α, n, f) = (90, 5, 10)



$$\begin{bmatrix} \cot(\alpha/2) & 0 & 0 & 0 \\ 0 & \cot(\alpha/2) & 0 & 0 \\ 0 & 0 & (f+n)/(f-n) & -1 \\ 0 & 0 & 2fn/(f-n) & 0 \end{bmatrix}$$

Let's do an example

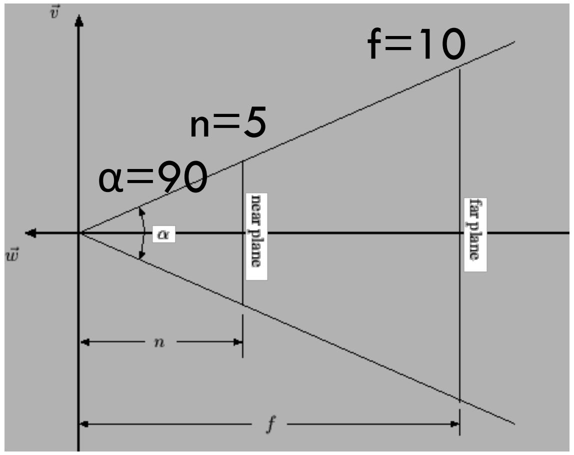
- Input parameters: (α, n, f) = (90, 5, 10)



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & -1 \\ 0 & 0 & 20 & 0 \end{bmatrix}$$

Let's do an example

- Input parameters: (α, n, f) = (90, 5, 10)

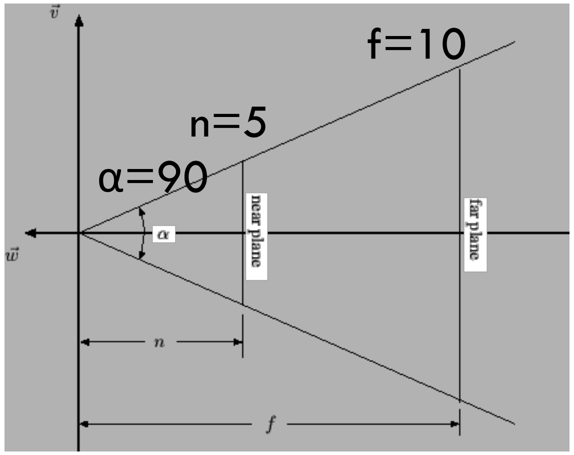


$$\begin{matrix}[1 & 0 & 0 & 0] \\ [0 & 1 & 0 & 0] \\ [0 & 0 & 3 & -1] \\ [0 & 0 & 20 & 0]\end{matrix}$$

Let's multiply some points:
(0,7,-6,1)
(0,7,-8,1)

Let's do an example

- Input parameters: (α, n, f) = (90, 5, 10)

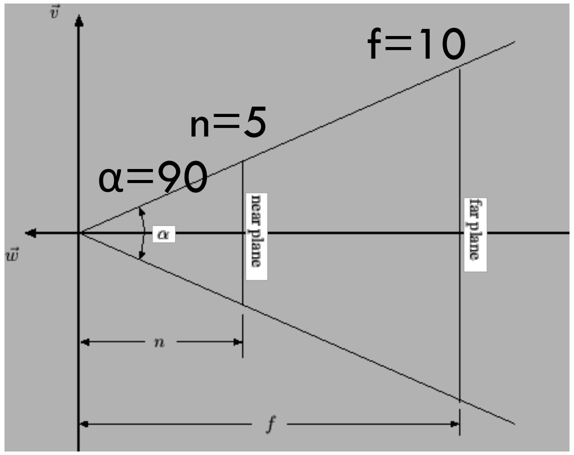


$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & -1 \\ 0 & 0 & 20 & 0 \end{bmatrix}$$

Let's multiply some points:
(0,7,-6,1) = (0,7,2,6) = (0, 1.16, 0.33)
(0,7,-8,1) = (0,7,-4,8) = (0, 0.88, -0.5)

Let’s do an example

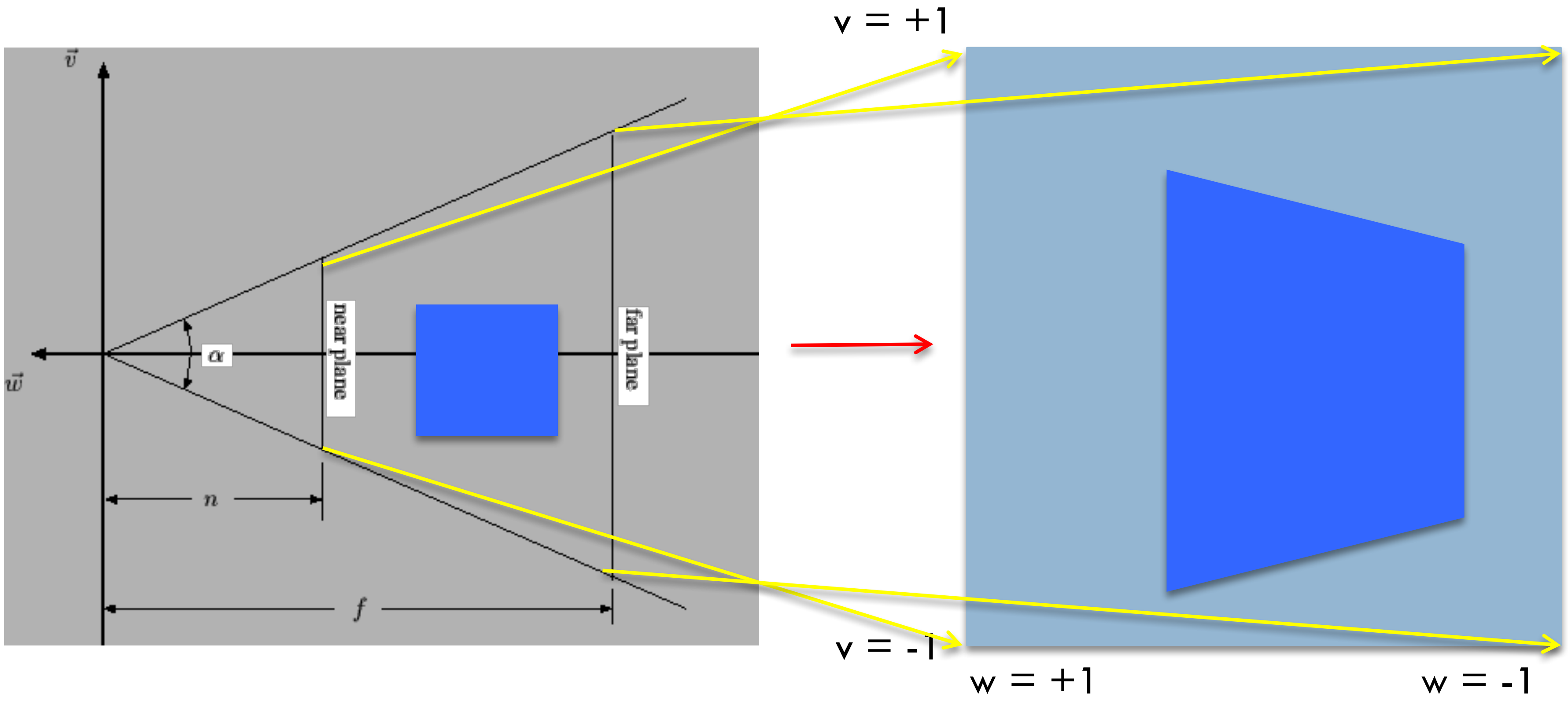
- Input parameters: (α, n, f) = (90, 5, 10)



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & -1 \\ 0 & 0 & 20 & 0 \end{bmatrix}$$

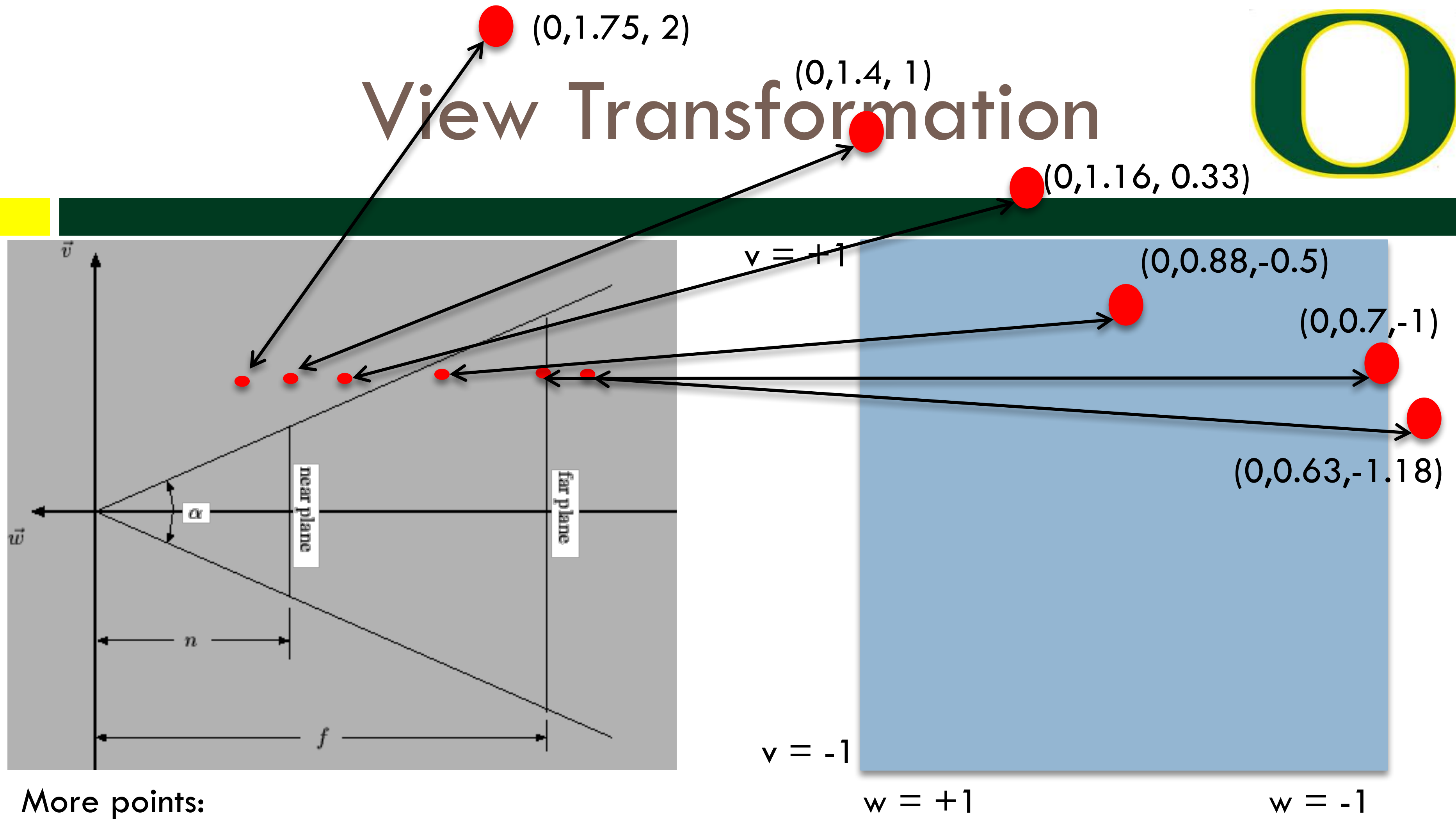
More points:
(0,7,-4,1) = (0,7,8,4) = (0, 1.75, 2)
(0,7,-5,1) = (0,7,5,5) = (0, 1.4, 1)
(0,7,-6,1) = (0,7,2,6) = (0, 1.16, 0.33)
(0,7,-8,1) = (0,7,-4,8) = (0, 0.88, -0.5)
(0,7,-10,1) = (0,7,-10,10) = (0, 0.7, -1)
(0,7,-11,1) = (0,7,-13,11) = (0, .63, -1.18)

View Transformation



The viewing transformation is not a combination of simple translations, rotations, scales or shears: it is more complex.

View Transformation



More points:

(0,7,-4,1) = (0,7,8,4) = (0, 1.75, 2)

(0,7,-5,1) = (0,7,5,5) = (0, 1.4, 1)

(0,7,-6,1) = (0,7,2,6) = (0, 1.16, 0.33)

(0,7,-8,1) = (0,7,-4,8) = (0, 0.88, -0.5)

(0,7,-10,1) = (0,7,-10,10) = (0, 0.7, -1)

(0,7,-11,1) = (0,7,-13,11) = (0, .63, -1.18)

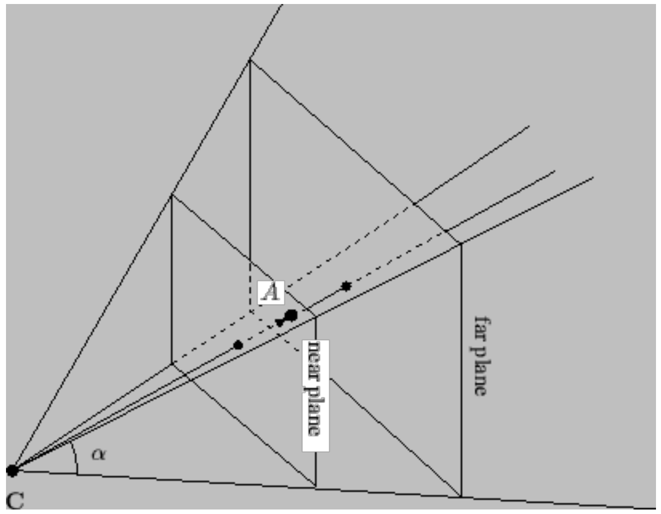
Note there is a non-linear relationship in W ("Z").

Putting It All Together

How do we transform?

- For a camera C,
  - Calculate Camera Frame
  - From Camera Frame, calculate Camera Transform
  - Calculate View Transform
  - Calculate Device Transform
  - Compose 3 Matrices into 1 Matrix (M)
- For each triangle T, apply M to each vertex of T, then apply rasterization/zbuffer

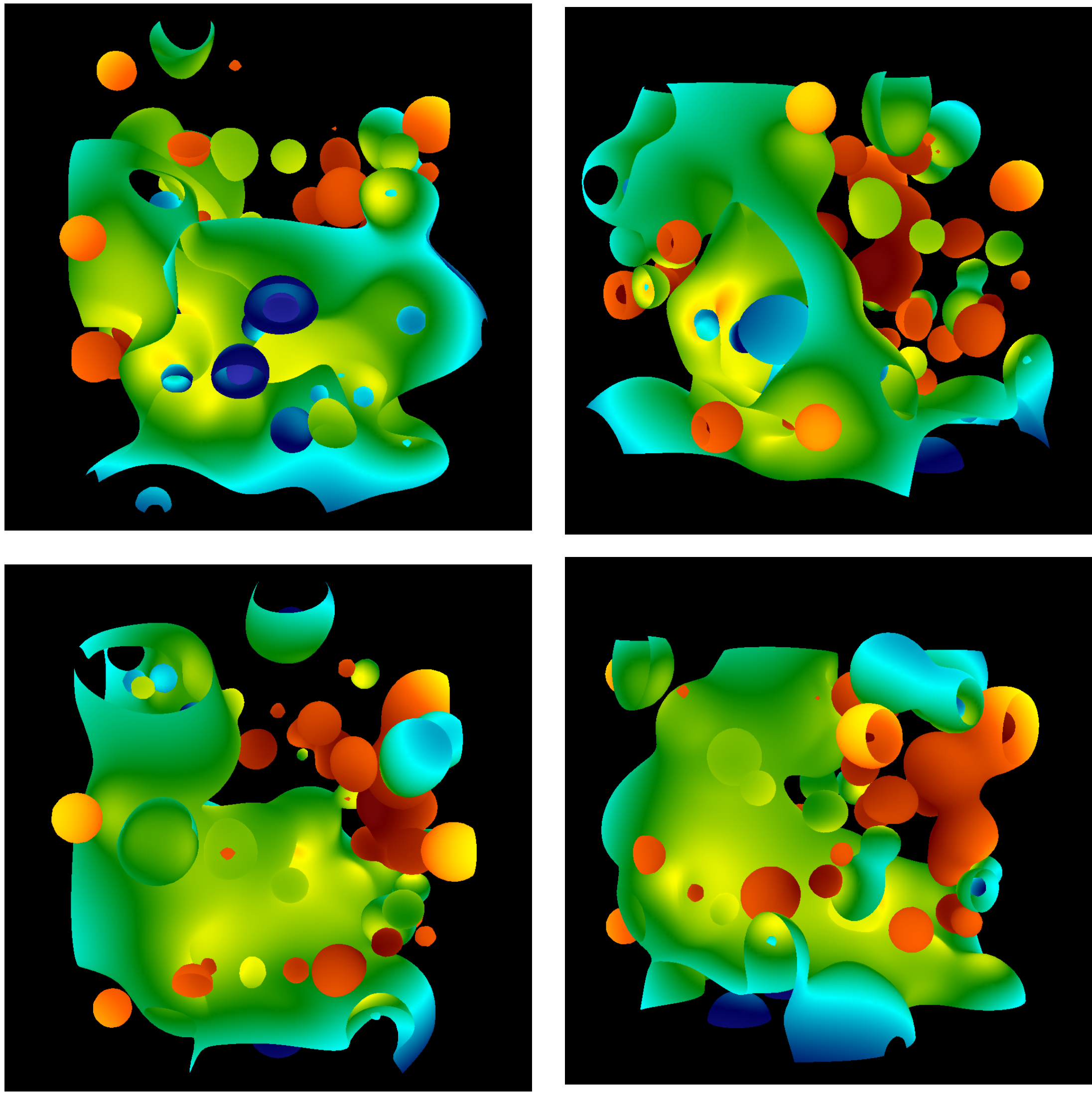
```
class Camera
{
  public:
    double          near, far;
    double          angle;
    double          position[3];
    double          focus[3];
    double          up[3];
};
```



Project 1E

Project #1E (6%), Due Tues April 27th

- Goal: add arbitrary camera positions
- Extend your project1D code
- New: proj1e_geometry.vtk available on web (9MB), “reader1e.cxx”.
- New: Matrix.cxx, Camera.cxx
- No Cmake, project1E.cxx



Project #1E, expanded

- Matrix.cxx: complete
- Methods:

```
class Matrix
{
  public:
    double          A[4][4];

    void            TransformPoint(const double *ptIn, double *ptOut);
    static Matrix   ComposeMatrices(const Matrix &, const Matrix &);
    void            Print(ostream &o);
};
```

Project #1E, expanded

- Camera.cxx: you work on this

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

    Matrix          ViewTransform(void) {;};
    Matrix          CameraTransform(void) {;};
    Matrix          DeviceTransform(void) {;};
    // Will probably need something for calculating Camera Frame as well
};
```

Also: GetCamera(int frame, int nFrames)

Project #1E, deliverables

- Same as usual, but times 4
  - 4 images, corresponding to
    - GetCamera(0, 1000)
    - GetCamera(250,1000)
    - GetCamera(500,1000)
    - GetCamera(750,1000)
- If you want:
  - Generate all thousand images, make a movie
    - Then you should wait for 1F. Then we will have shading too.

Project #1E, game plan

```
vector<Triangle> t = GetTriangles();
AllocateScreen();
for (int i = 0 ; i < 4 ; i++)
{   int f = 250*i;
    InitializeScreen();
    Camera c = GetCamera(f, 1000);
    TransformTrianglesToDeviceSpace(); // involves setting up and applying matrices
                                       //… if you modify vector<Triangle> t,
                                       // remember to undo it later
    RenderTriangles();
    SaveImage();
}
```

Correct answers given for GetCamera(0, 1000)

```
Camera Frame: U = 0, 0.707107, -0.707107
Camera Frame: V = -0.816497, 0.408248, 0.408248
Camera Frame: W = 0.57735, 0.57735, 0.57735
Camera Frame: O = 40, 40, 40
Camera Transform
(0.0000000 -0.8164966 0.5773503 0.0000000)
(0.7071068 0.4082483 0.5773503 0.0000000)
(-0.7071068 0.4082483 0.5773503 0.0000000)
(0.0000000 0.0000000 -69.2820323 1.0000000)
View Transform
(3.7320508 0.0000000 0.0000000 0.0000000)
(0.0000000 3.7320508 0.0000000 0.0000000)
(0.0000000 0.0000000 1.0512821 -1.0000000)
(0.0000000 0.0000000 10.2564103 0.0000000)
Transformed 37.1132, 37.1132,37.1132, 1 to 0, 0,1
Transformed -75.4701, -75.4701,-75.4701, 1 to 0, 0,-1
```

Project #1E pitfalls

- All vertex multiplications use 4D points. Make sure you send in 4D points for input and output, or you will get weird memory errors.
  - Make sure you divide by w.

Project #1E, pitfalls

- People often get a matrix confused with its transpose. Use the method Matrix::Print() to make sure the matrix you are setting up is what you think it should be. Also, remember the points are left multiplied, not right multiplied.
- Regarding multiple renderings:
  - Don't forget to initialize the screen between each render
  - If you modify the triangle in place to render, don't forget to switch it back at the end of the render

Project #1F (8%), Due Monday May 3rd

- Goal: add shading, movie