# LECTURE 3: MATH BACKGROUND, UNITY INSTALLATION

Ehsan Aryafar
earyafar@pdx.edu
http://web.cecs.pdx.edu/~aryafare/VR.html

ROLLING

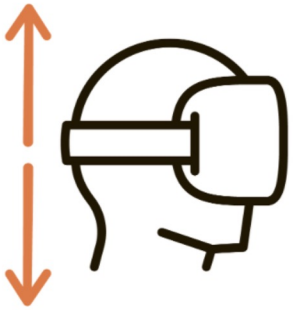
**Roll** is where the head **pivots side to side** (i.e. when peeking around a corner)

PITCHING

**Pitch** is where the head **tilts along a vertical axis** (i.e. when looking up or down).

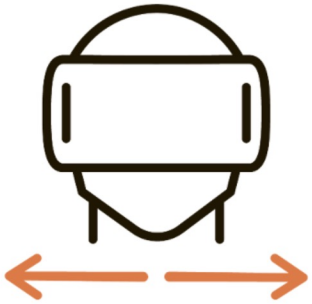
YAWING

**Yaw** is where the head **swivels along a horizontal axis** (i.e. when looking left or right)



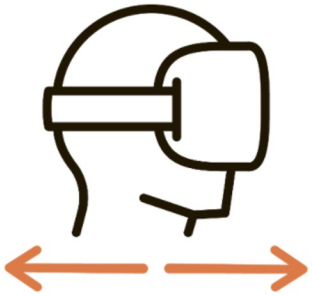
ELEVATING

**Elevation** is where a person **moves up or down** (i.e. when bending down or standing up)



STRAFING

**Strafe** is where a person **moves left or right** (i.e. when sidestepping)



SURGING

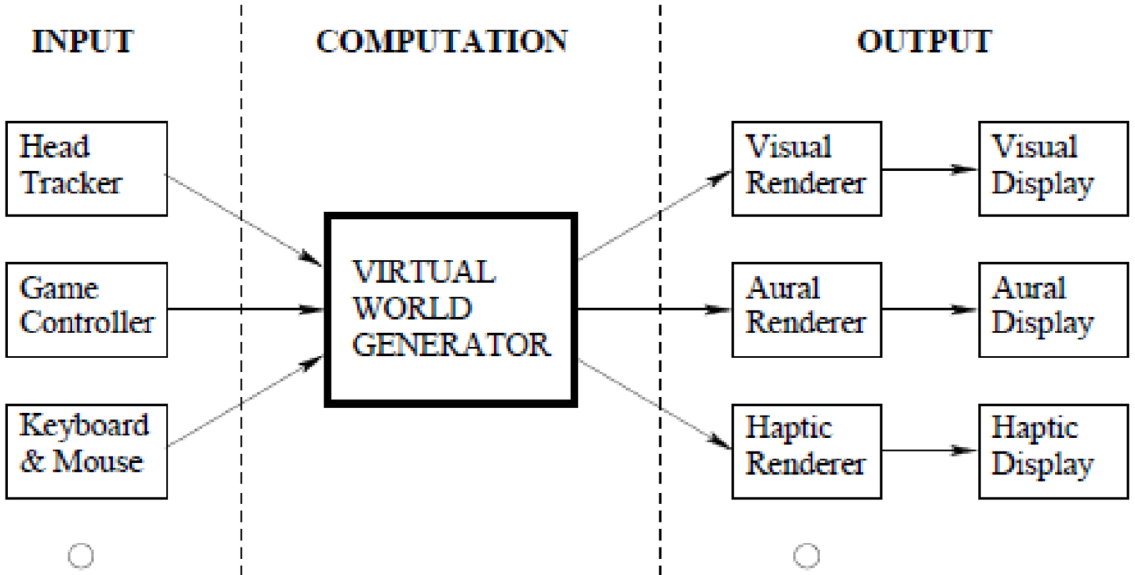
**Surge** is where a person moves **forwards or backwards** (i.e. when walking)

# Recall: Components of VR Hardware

- Displays (outputs)
  - Devices that each stimulate a sense organ
- Sensors (input)
  - Devices that extract information from the real world
- Computers
  - Devices that process input and output

# Recall: VR Software

- Industry is moving towards full-fledged VR engines
  - Similar to game engines
    - Game engine: a software-development environment designed for people to create video games, which provides the following functionalities: 2D/3D rendering and management of memory, sound, networking, AI, threading, etc.
  - SDKs are being developed for particular headsets
    - Handle low level operations, e.g., device derivers, head tracking, and display

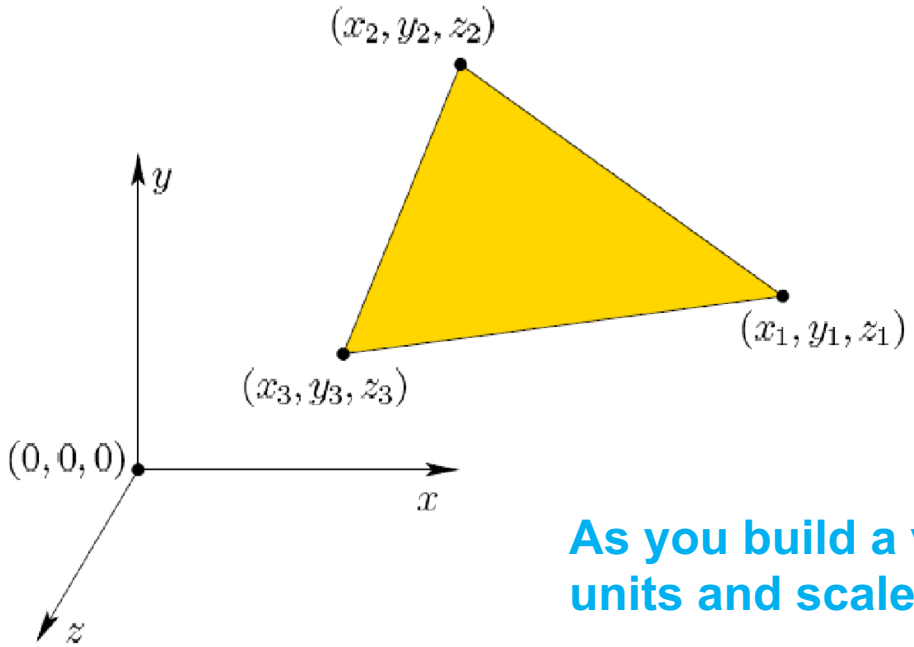


# Outline

- How to represent and store geometric models
- Linear algebra: working with vectors and matrices
  - Addition
  - Multiplication
  - Transpose
  - Dot product
  - Cross product
  - Matrix Inversion
- Unity Installation
- Discussion information

# Geometric Model of a Virtual World

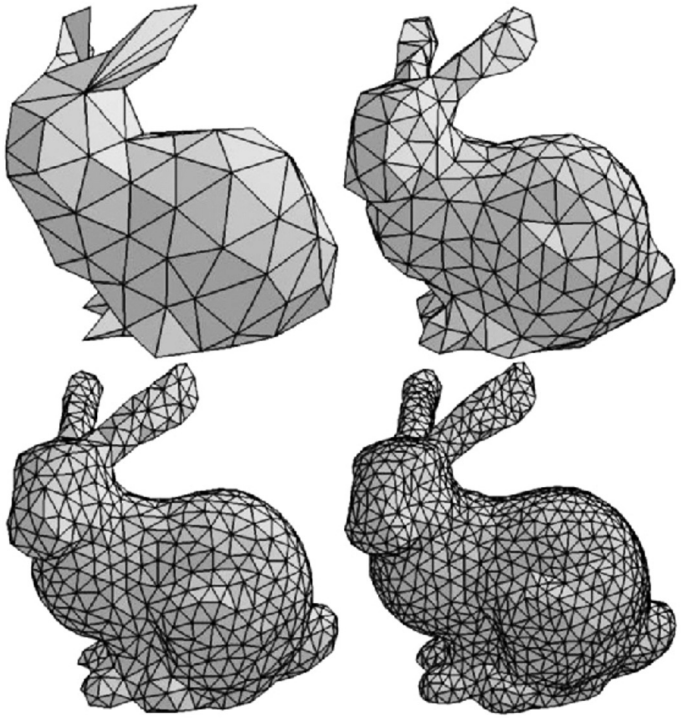
- We assume a 3D virtual world in Euclidean space
  - Each point is represented as a triple of real-valued coordinates (x, y, z)
  - We consider a right-hand coordinate system
  - Models are objects placed in the virtual world
  - Models can be fixed (walls) or movable (bullet)




**As you build a virtual world, try to keep the units and scale like the real world!**

# How To Represent Geometric Models

- Geometric models are solid regions in 3D space
  - Geometric models are represented in terms of primitives, the simplest form is a 3D triangle
    - Triangles are heavily used, e.g., for ease of manipulation on GPUs



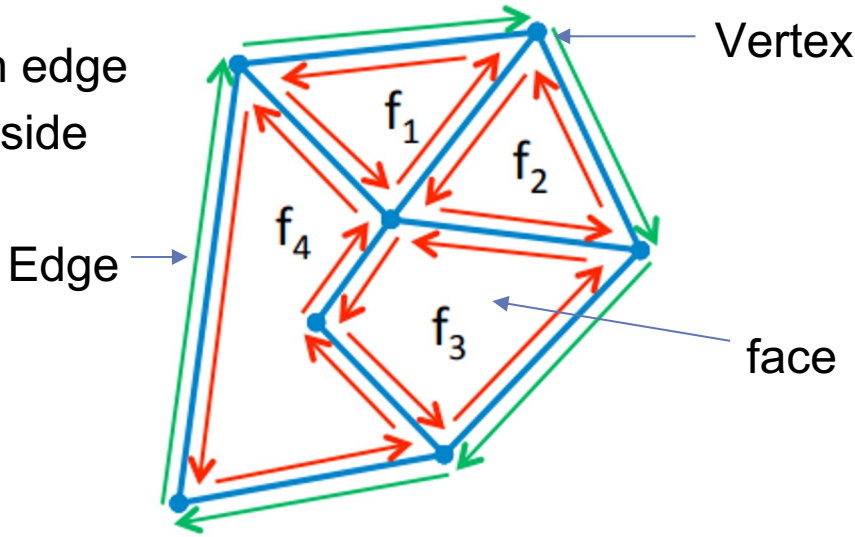
3D mesh-triangles with different resolutions!

# How to Store Models

**Not in Exam!**

- Many adjacent triangles share common edges and vertices
  - **We can cleverly store all the needed data (instead of storing an infinite amount of points corresponding to each triangle)**
- A very common data structure used is “Doubly Connected Edge List”
  - Records geometric + other information for each vertex, edge, and face
    - Since an edge borders two faces, each edge is replaced by two half edges, one for each face
      - Two directional half-edges for each edge
    - Edges are oriented counterclockwise inside each face

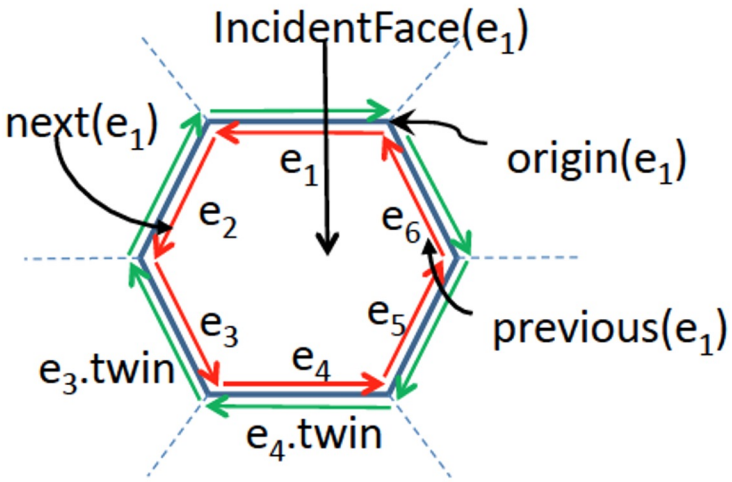
Storage space requirement: linear in the number of vertices, edges, and faces



# Doubly Connected Edge List (DCEL)

- The vertex record of a vertex v stores the coordinates of v. It also stores a pointer IncidentEdge(v) to an arbitrary half-edge that has v as its origin

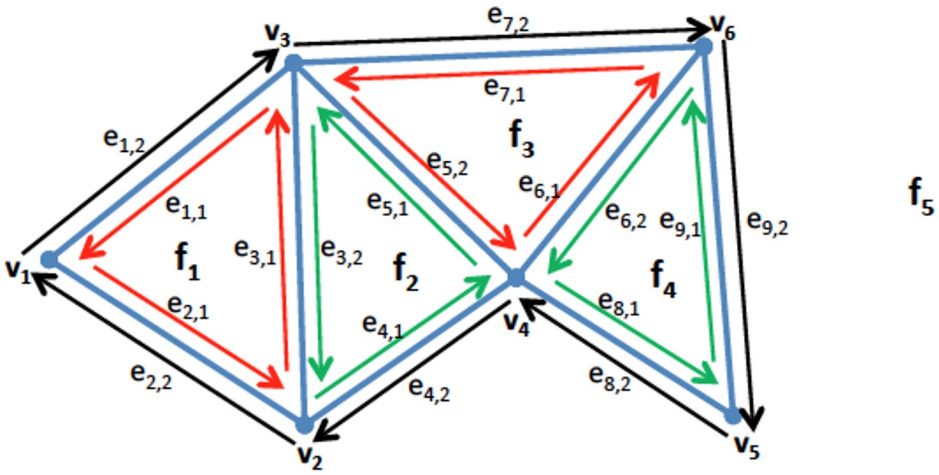
- The face record of a face f stores a pointer to some half-edge on its boundary which can be used as a starting point to traverse f in counterclockwise order




- The half-edge record of a half-edge e stores pointer to:
  - Origin (e)
  - Twin of e, e.twin or twin(e)
  - The face to its left ( IncidentFace(e) )
  - Next(e) : next half-edge on the boundary of IncidentFace(e)
  - Previous(e) : previous half-edge

**Not in Exam!**

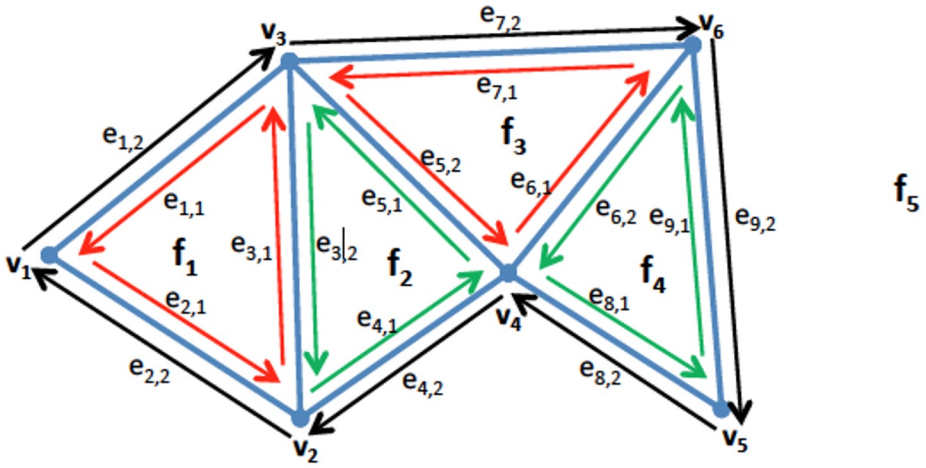
# Doubly Connected Edge List (DCEL)




**Not in Exam!**

| Vertex | Coordinates | IncidentEdge |
|---|---|---|
| $v_1$ | $(x_1, y_1)$ | $e_{2,1}$ |
| $v_2$ | $(x_2, y_2)$ | $e_{4,1}$ |
| $v_3$ | $(x_3, y_3)$ | $e_{3,2}$ |
| $v_4$ | $(x_4, y_4)$ | $e_{6,1}$ |
| $v_5$ | $(x_5, y_5)$ | $e_{9,1}$ |
| $v_6$ | $(x_6, y_6)$ | $e_{7,1}$ |

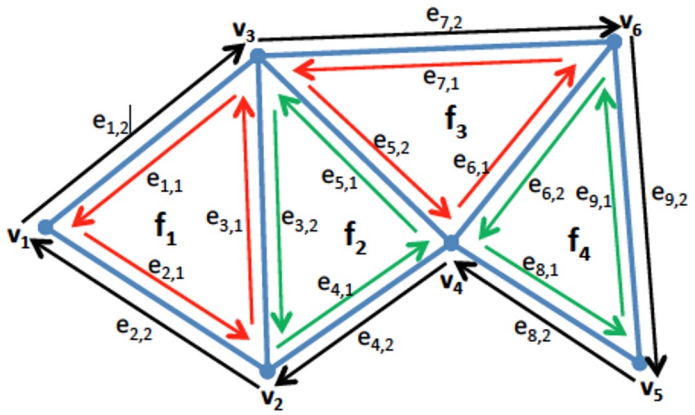
# Doubly Connected Edge List (DCEL)




| Face | Edge |
|---|---|
| $f_1$ | $e_{1,1}$ |
| $f_2$ | $e_{5,1}$ |
| $f_3$ | $e_{5,2}$ |
| $f_4$ | $e_{8,1}$ |
| $f_5$ | $e_{9,2}$ |

**Not in Exam!**

# Doubly Connected Edge List (DCEL)




**Not in Exam!**

| Half-edge | Origin | Twin | IncidentFace | Next | Previous |
|---|---|---|---|---|---|
| $e_{3,1}$ | $v_2$ | $e_{3,2}$ | $f_1$ | $e_{1,1}$ | $e_{2,1}$ |
| $e_{3,2}$ | $v_3$ | $e_{3,1}$ | $f_2$ | $e_{4,1}$ | $e_{5,1}$ |
| $e_{4,1}$ | $v_2$ | $e_{4,2}$ | $f_2$ | $e_{5,1}$ | $e_{3,2}$ |
| $e_{4,2}$ | $v_4$ | $e_{4,1}$ | $f_5$ | $e_{2,2}$ | $e_{8,2}$ |
| ... | ... | ... | ... | ... | ... |

# Viewing the Models

- One of the most important aspects of VR is how the models are going to look when viewed on a display, which has two parts:
  - **Where** the points in the virtual world should appear on the display
    - Topic of next lecture
  - How should each part of the model appear after lighting and other impacts
    - **This is called rendering and will be covered later**

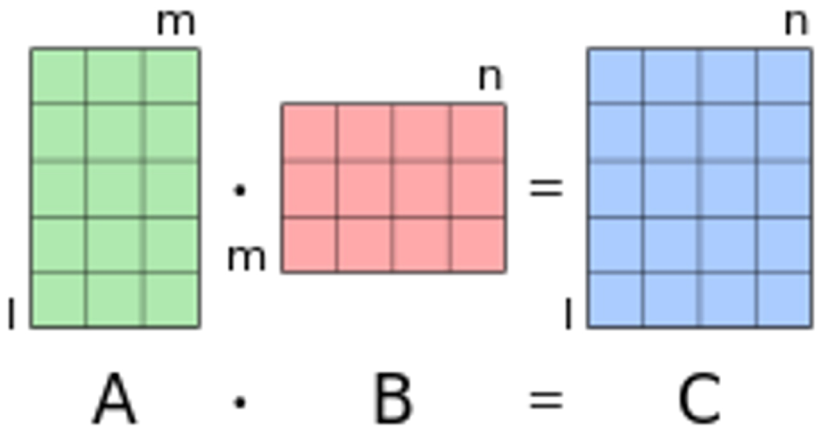
# Background: Matrix Addition

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots & a_{2n} + b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \cdots & a_{mn} + b_{mn} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 \\ 1+7 & 0+5 \\ 1+2 & 2+1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 8 & 5 \\ 3 & 3 \end{bmatrix}$$

# Background: Matrix Multiplication



# Background: Matrix Multiplication

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41}$$

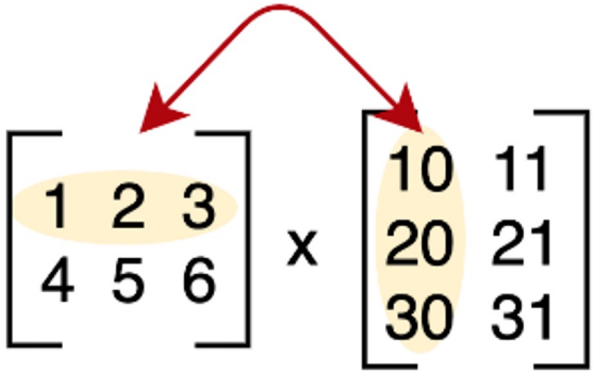
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

2 x 4 4 x 3 2 x 3

$$c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

# Example Matrix Multiplication



$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 10 & 11 \\ 20 & 21 \\ 30 & 31 \end{bmatrix}$$

$$= \begin{bmatrix} 1x10 + 2x20 + 3x30 & 1x11 + 2x21 + 3x31 \\ 4x10 + 5x20 + 6x30 & 4x11 + 5x21 + 6x31 \end{bmatrix}$$

$$= \begin{bmatrix} 10+40+90 & 11+42+93 \\ 40+100+180 & 44+105+186 \end{bmatrix} = \begin{bmatrix} 140 & 146 \\ 320 & 335 \end{bmatrix}$$

# Background: Matrix Transpose

- Transpose of a matrix A is shown as $A^T$
- Formally, the i-th row, j-th column element of $A^T$ is the j-th row, i-th column element of A. Below are some examples:

$$\begin{bmatrix} 1 & 2 \end{bmatrix}^T = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

# Background: Dot Product

- Algebraic definition

The dot product of two vectors **a** = [$a_1$, $a_2$, …, $a_n$] and **b** = [$b_1$, $b_2$, …, $b_n$] is defined as:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

$$[1, 3, -5] \cdot [4, -2, -1] = (1 \times 4) + (3 \times -2) + (-5 \times -1) = 3$$

- Geometric definition (the same value as algebraic)

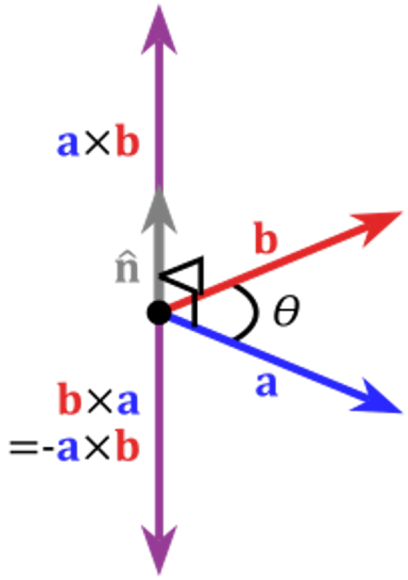
$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta$$

$$\|\mathbf{a}\| = \sqrt{\mathbf{a} \cdot \mathbf{a}}$$

# Background: Cross Product

- Cross product of two vectors a and b, is a vector that is perpendicular to both a and b and its size is given by:

$$\mathbf{a} \times \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \sin(\theta) \, \mathbf{n}$$



# Cross Product Example

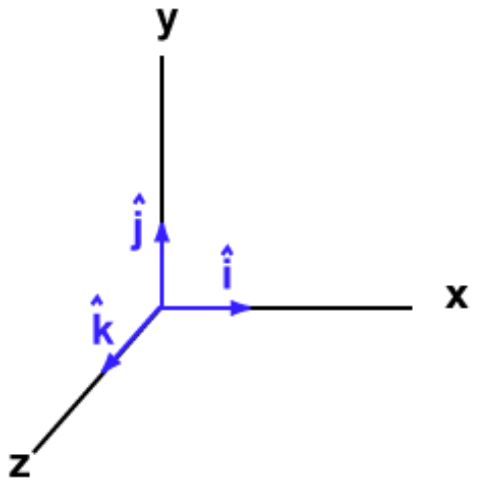
If $\mathbf{a} = \langle 1, 3, 4 \rangle$ and $\mathbf{b} = \langle 2, 7, -5 \rangle$, then

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ 1 & 3 & 4 \\ 2 & 7 & -5 \end{vmatrix}$$

$$= \begin{vmatrix} 3 & 4 \\ 7 & -5 \end{vmatrix} \mathbf{i} - \begin{vmatrix} 1 & 4 \\ 2 & -5 \end{vmatrix} \mathbf{j} + \begin{vmatrix} 1 & 3 \\ 2 & 7 \end{vmatrix} \mathbf{k}$$

$$= (-15 - 28)\mathbf{i} - (-5 - 8)\mathbf{j} + (7 - 6)\mathbf{k}$$

$$= -43\mathbf{i} + 13\mathbf{j} + \mathbf{k}$$



# Background: Matrix Inversion

- In linear algebra, an n-by-n square matrix is called invertible, if there exists an n-by-n square matrix B such that

$$\mathbf{AB} = \mathbf{BA} = \mathbf{I}_n$$

$$I_n = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

$I_n$ is the identity matrix

- Inverse of **A** is typically denoted by $\mathbf{A}^{-1}$

# Unity

- Unity is an all-in-one game engine/editor that you can use to build VR/AR games.
- Installation can take a significant amount of time so probably start now.

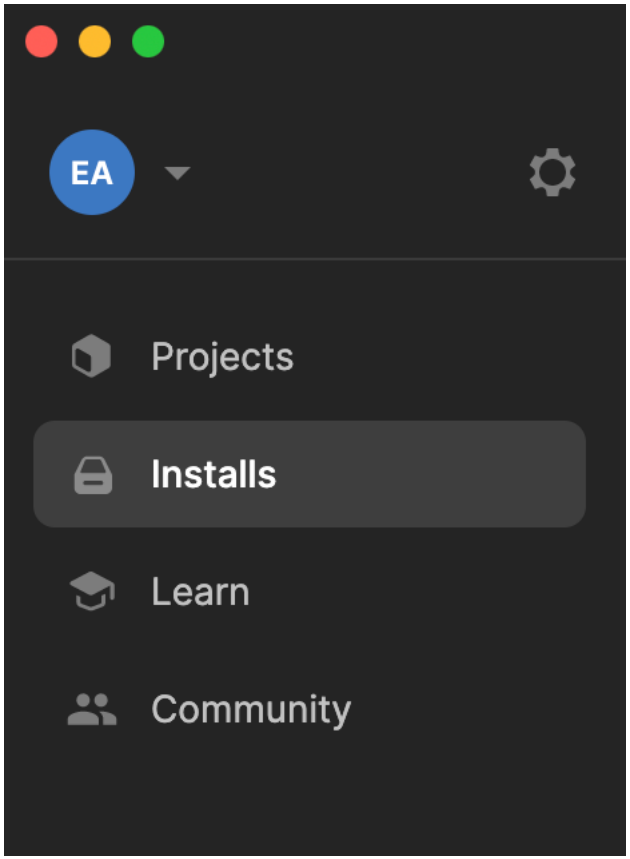
# UnityHub

- Unity has a lot of versions of its editor and many of them aren't compatible with each other.
- UnityHub is an installer and manager for your editors and projects.
- Download it here: https://store.unity.com/download
- This shouldn't take too long (It's ~80Mb)

# Editors

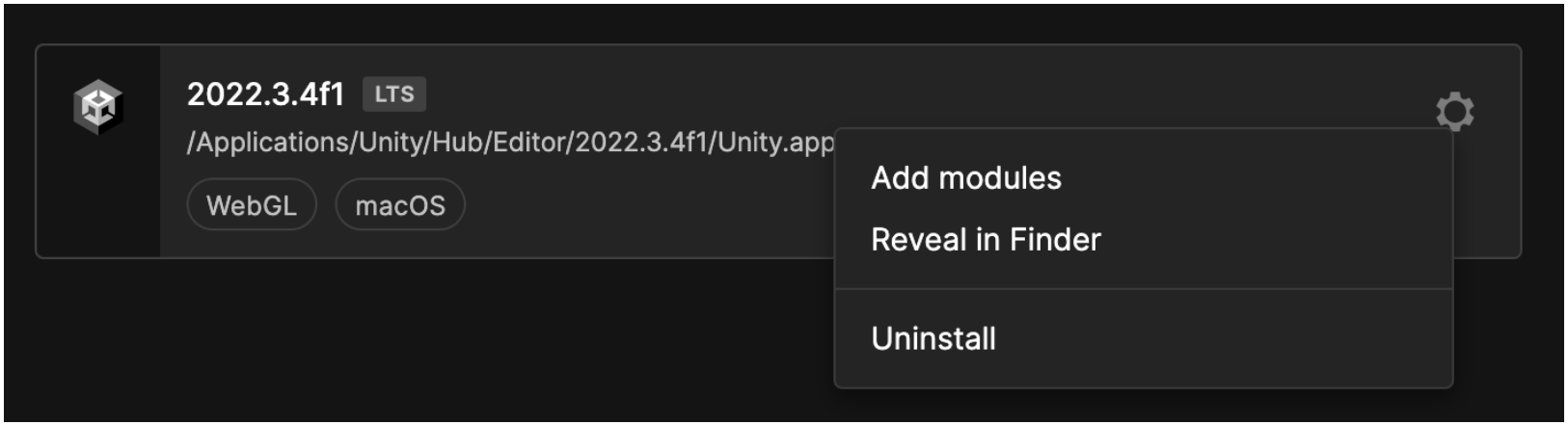
- Go to Installs in the UnityHub

**Unity most of the times automatically installs the latest LTS version for you!**



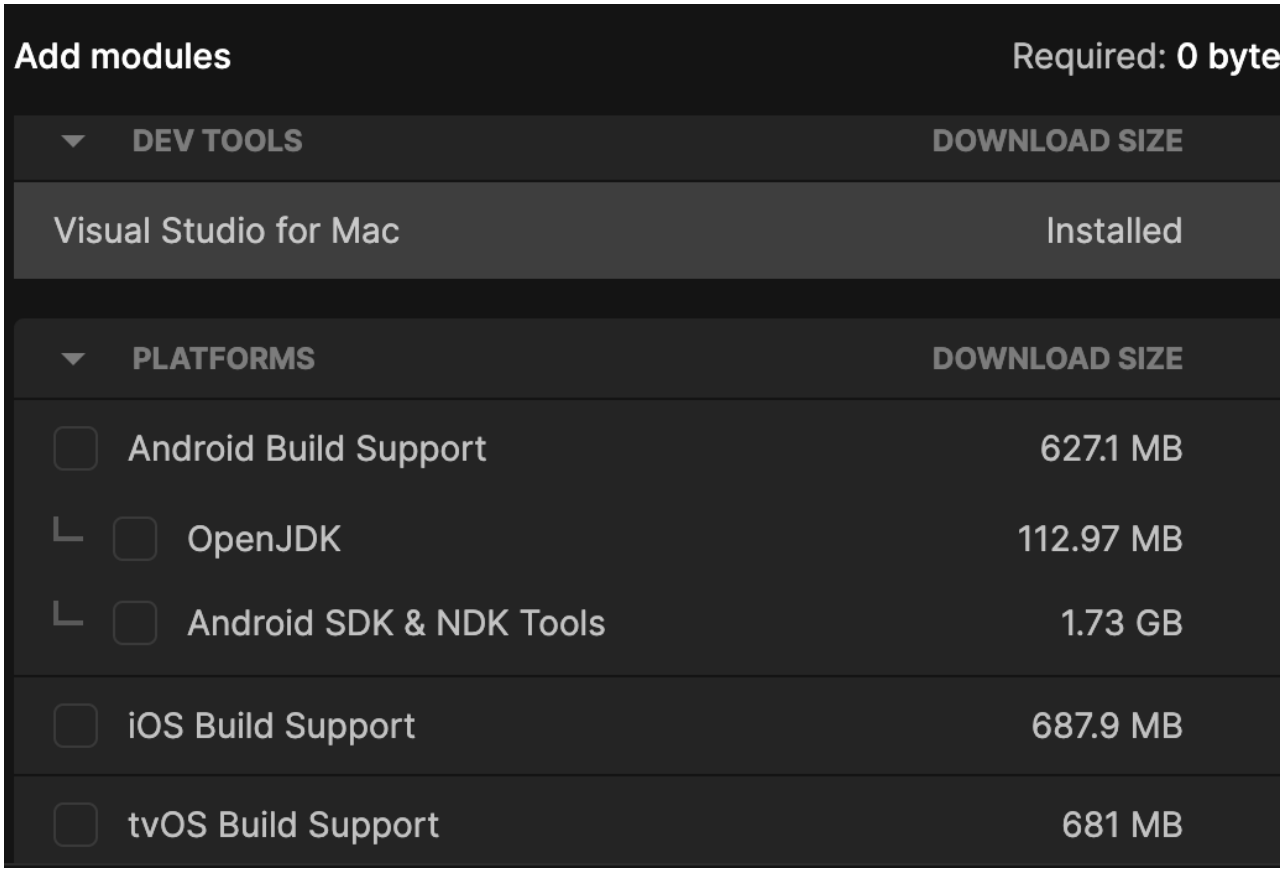
# Editors

- You can add the modules you need.
  - Android is needed for Oculus Quests



# Editors

- You can add the modules you need.
  - Android is needed for Oculus Quests



# Unity

- Done! Wait for it to install.
- It is possible there will be errors at some point in the process- This is normal, don’t panic.
- We’ll start creating projects next week.

# Canvas Discussions

- I am not using a Slack channel for the class this term
  - Rely on Canvas Discussions
- Good place to trade tips on Unity errors