CMPS 327: Introduction to Video Game Design and Development

Dr. Arun K. Kulshreshth

Fall 2020

Lecture 9: Midterm 1 Review

Reminder

- Midterm Exam 1
  - Wednesday, September 23, 2020
  - ODS Students: Schedule your exam with ODS
  - Location: OLVR 112 and OLVR 113
  - Mask is required

Making games fun

*Adams & Rollings

- 50% Avoiding errors--bad programming, bad music and sound, bad art, bad user-interfaces, bad game design. "Basic competence will get you up to average."
- 35% Tuning and polishing--attention to detail
- 10% Imaginative variations--level design
- 4% True design innovation--the game's original idea and subsequent creative decisions
- 1% An unpredictable, unananalyzable, unnamable quality--"luck, magic, or stardust"

Making games fun

*Adams & Rollings

- 50% Avoiding errors
- 35% Tuning and polishing
- 10% Imaginative variations
- 4% True design innovation
- 1% Luck, magic, or stardust

Implications:

- A well-tuned game with no major problems and interesting levels but no new ideas could be **95%** fun.
- A novel game idea that is (very) poorly executed could be only **4%** fun.

Game Creation

- Game engine
  - a software system within which games can be created
  - E.g. Unity 3D, Unreal engine, CryEngine, etc.

- Three main components of a game engine:
  - A rendering engine for graphics
  - A physics engine to provide a collision detection system and dynamics simulations
  - A sound-generating component or sound engine

Game Engine Functionality

- A scripting language apart from the code driving the game
- Animation
- Artificial intelligence algorithms (e.g., path-finding algorithms)
- A scene graph that holds the spatial representation in a graphical sense

Game Architecture

- The code for modern games is highly complex
- With code bases exceeding a million lines of code, a well-defined architecture is essential
- A video game has three important steps
  - Initialization/startup/loading
  - Game play
  - Shutdown/unloading

Initialization Step

- Prepares everything that is necessary to start a part of the game
- This step deals with resource and device acquisition, memory allocation, initialization of the game’s GUI, and loading of art assets such as an intro video from file.

Shutdown Step

- Undoes everything the initialization step did, but in reverse order
- Unloads game data from memory, free resources

- Minimalize mismatch errors in the initialization and shutdown steps
- Creating an object acquires and initializes all the necessary resources, and destroying it destroys and shuts down all those resources

Main Game Loop

- Games are driven by a game loop that performs a series of tasks every frame
- Some games have separate loops for the frontend and the game itself
- Other games have a unified main loop

Main Game Loop

- Tasks
  - Handling time
    - Timers
    - Must be precise to at least 1 ms or less
    - 30fps = 1/30 second = 33.333… ms
  - Gathering player input
  - Networking
  - Simulation
  - Collision detection and response
  - Object updates
  - Rendering
  - Other miscellaneous tasks

Frame Rate

- Expressed in frames per second or fps
- In the game loop each task takes time
- Frames/second
  - How many times the loop can be executed in a second?
  - E.g. 30 fps, 60 fps,
- Display also has a certain refresh rate
  - Frame rate is independent of this

Frame Rate

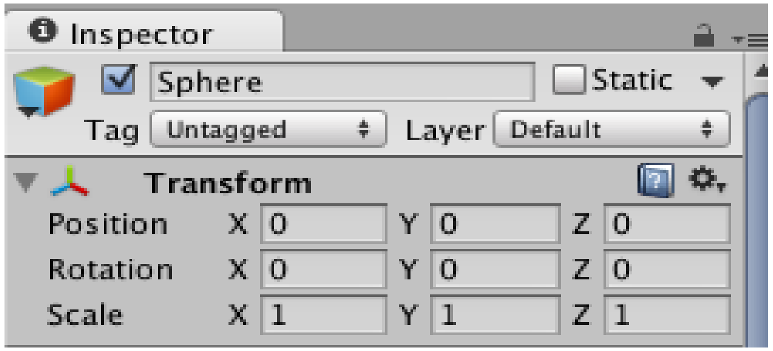
- Slow frame rate
  - motion looks jagged
- Too fast
  - Soap-opera effect
  - Jerky motion
- Soap-opera effect
  - motion smoothing/bluring
  - a feature of many modern televisions
  - E.g. sports content on televisions

Unity 3D

- What data is stored in a transform component?
- What are important methods of a MonoBehavior class?
- What is the difference between Update, LateUpdate, and FixedUpdate?
- What is a game object?
- What is a prefab? Why is a prefab useful?
- What happens when you set a collider as a Trigger?
- What happens when you set a rigid body as Kinematic?

Transform Component

- Three parts
  - Position, Rotation and Scale
- Position along X, Y, Z axis
- Rotation along X, Y, Z axis
  - Euler angles
- Scale along X, Y, Z axis



Position	X	0	Y	0	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

Monobehavior Methods

- Awake is called when the script instance is being loaded.
- Start is called on the frame when a script is enabled just before any of the Update methods is called the first time.
- Update is called every frame, if the MonoBehaviour is enabled.

More Methods: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

Monobehaviour Methods

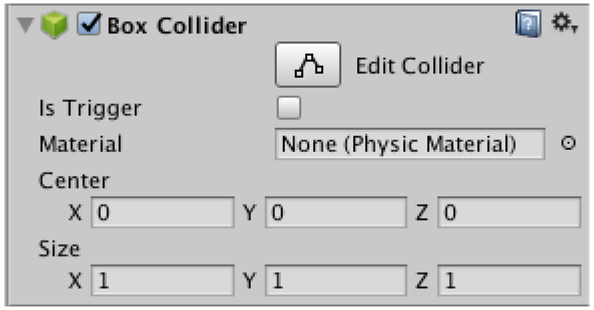
- FixedUpdate
  - Called every fixed time intervals, if the MonoBehaviour is enabled.
  - Independent of the frame rate
  - FixedUpdate should be used instead of Update when dealing with physics. For example when adding a force to a rigidbody, you have to apply the force every fixed frame inside FixedUpdate instead of every frame inside Update.

Monobehaviour Methods

- LateUpdate
  - Called every frame, if the Behaviour is enabled.
  - LateUpdate is called after all Update functions have been called.
  - This is useful to order script execution.
  - For example a follow camera should always be implemented in LateUpdate because it tracks objects that might have moved inside Update.

A Trigger Collider

- A collider could be set as a trigger
  - Check "Is Trigger"
- When a collider is a trigger
  - Objects don't bump into it
  - Objects pass through it
- Physics engine ignores such colliders
- We can detect the events using trigger callback functions



Box Collider

Edit Collider

Is Trigger	☐				
Material	None (Physic Material)				
Center	X 0	Y 0	Z 0		
Size	X 1	Y 1	Z 1		

Smooth Motion Between Two Positions

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public Transform startMarker;
    public Transform endMarker;
    public float speed = 1.0F;
    private float startTime;
    private float journeyLength;
    void Start() {
        startTime = Time.time;
        journeyLength = Vector3.Distance(startMarker.position, endMarker.position);
    }
    void Update() {
        float distCovered = (Time.time - startTime) * speed;
        float fracJourney = distCovered / journeyLength;
        transform.position = Vector3.Lerp(startMarker.position, endMarker.position, fracJourney);
    }
}
```

https://docs.unity3d.com/ScriptReference/Vector3.Lerp.html

Smooth Motion Between Two Positions

- What changes would you make to the code to make the moving object stop if it gets closer than half of the distance between the starting position and ending position?

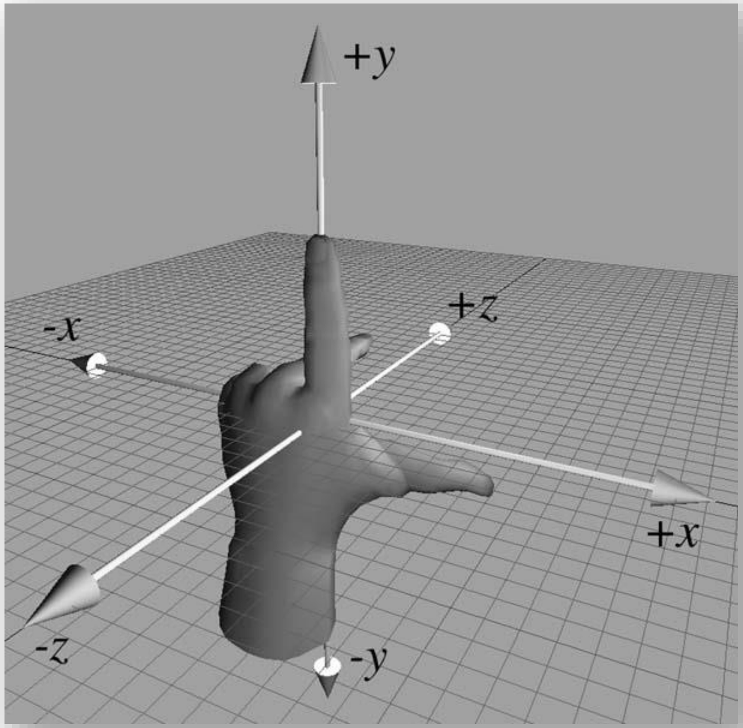
Smooth Motion Between Two Positions

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public Transform startMarker;
    public Transform endMarker;
    public float speed = 1.0F;
    private float startTime;
    private float journeyLength;
    void Start() {
        startTime = Time.time;
        journeyLength = Vector3.Distance(startMarker.position, endMarker.position);
    }
    void Update() {
        float distCovered = (Time.time - startTime) * speed;
        float fracJourney = distCovered / journeyLength;
        if(fracJourney < 0.5f)
            transform.position = Vector3.Lerp(startMarker.position, endMarker.position, fracJourney);
    }
}
```

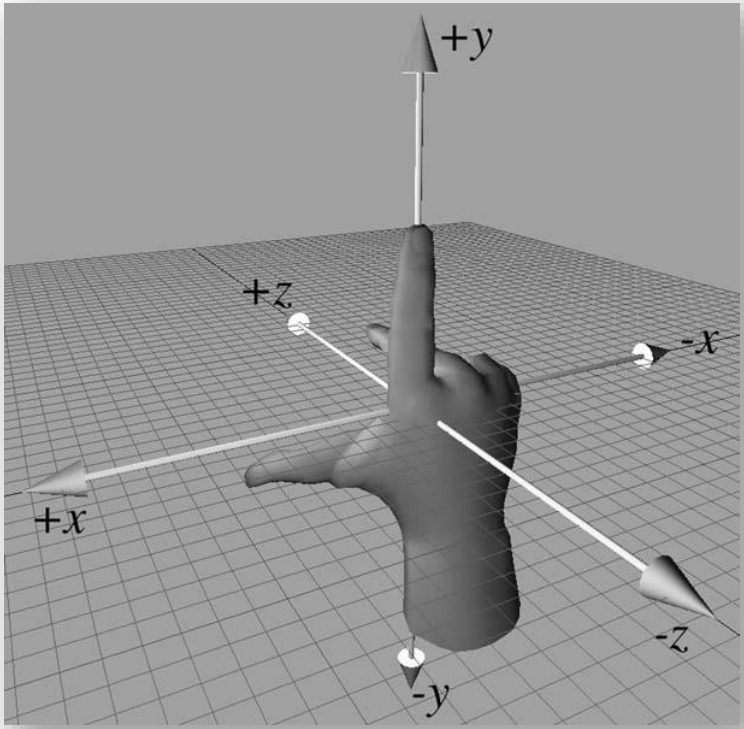
https://docs.unity3d.com/ScriptReference/Vector3.Lerp.html

Left-handed Coordinates



- *+z* goes “into” screen
- Use your left hand
- Thumb is *+x*
- Index finger is *+y*
- Second finger is *+z*

Right-handed Coordinates



- + *z* goes "out from" screen
- Use your right hand
- Thumb is +*x*
- Index finger is +*y*
- Second finger is +*z*
- (Same fingers, different hand)

Vector Multiplication by a Scalar

- Can multiply a vector by a scalar.
- Result is a vector of the same dimension.
- To multiply a vector by a scalar, multiply each component by the scalar.

$$k \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} k = \begin{bmatrix} ka_1 \\ ka_2 \\ \vdots \\ ka_{n-1} \\ ka_n \end{bmatrix}$$

Vector Addition: Algebra

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} = \begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \\ \vdots \\ a_{n-1} + b_{n-1} \\ a_n + b_n \end{bmatrix}$$

Vector Subtraction: Algebra

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} - \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} + \left(- \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} \right) = \begin{bmatrix} a_1 - b_1 \\ a_2 - b_2 \\ \vdots \\ a_{n-1} - b_{n-1} \\ a_n - b_n \end{bmatrix}$$

Vector Magnitude

- The magnitude of a vector is a scalar.
- Also called the “norm”.
- It is always positive
- Magnitude of a vector is its length.

$$\|\mathbf{v}\| = \sqrt{\sum_{i=1}^{n} {v_i}^2} = \sqrt{{v_1}^2 + {v_2}^2 + \cdots + {v_{n-1}}^2 + {v_n}^2}$$

Normalized Vector

- A *normalized* vector always has unit length.
- To normalize a nonzero vector, divide by its magnitude.

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$$

Example

Normalize [12, -5]:

$$\frac{\begin{bmatrix}12 & -5\end{bmatrix}}{\left\|\begin{bmatrix}12 & -5\end{bmatrix}\right\|} = \frac{\begin{bmatrix}12 & -5\end{bmatrix}}{\sqrt{12^2+5^2}} = \frac{\begin{bmatrix}12 & -5\end{bmatrix}}{\sqrt{169}} = \frac{\begin{bmatrix}12 & -5\end{bmatrix}}{13} = \begin{bmatrix}\frac{12}{13} & \frac{-5}{13}\end{bmatrix}$$

$$\approx \begin{bmatrix}0.923 & -0.385\end{bmatrix}$$

Computing Distance

- To find the geometric distance between two points *a* and *b*.
- Compute the vector **d** from **a** to **b**.
  - **d** = **b** – **a**
- Compute the magnitude of **d**.

Problem

- Assuming a unit of your coordinate axis is one meter, if the player at location (13, 23, 9) shot a bullet at an enemy at location (43, 23, 49). If the range of the bullet is only 30 meters, would it hit the enemy? Explain your answer with all the vector calculations. No geometric explanations are allowed.

Problem

- The distance between the player and enemy is the length of the vector between the two.

- $v = (43 - 13, 23 - 23, 49 - 9) = (30, 0, 40)$

- The length of this vector is its magnitude which is $||v|| = \sqrt{30 * 30 + 40 * 40} = 50$ meters.

- The enemy is farther than 30 meters and thus the bullet is not going to hit it.

Dot Product

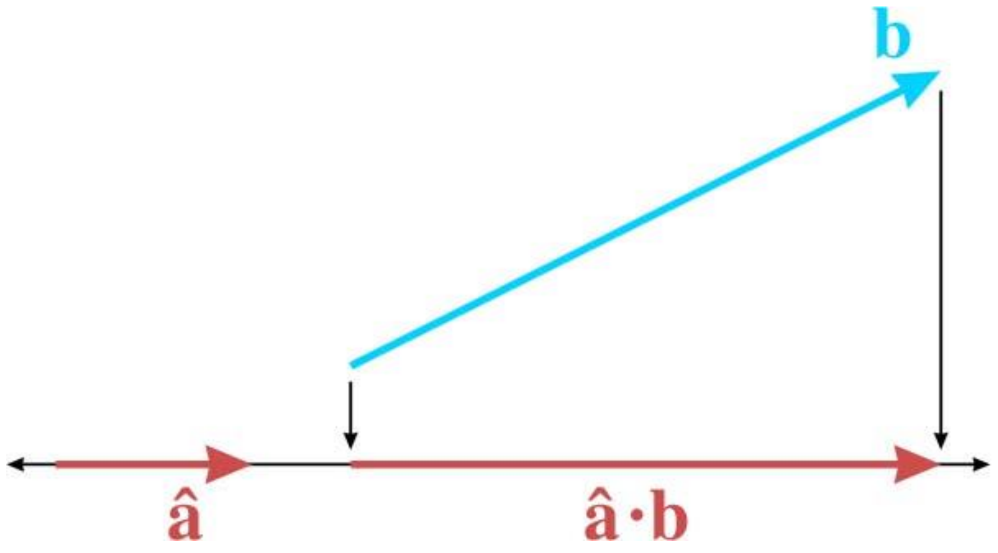
Can take the dot product of two vectors of the same dimension. The result is a scalar.

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$$

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} = a_1 b_1 + a_2 b_2 + \cdots + a_{n-1} b_{n-1} + a_n b_n$$

Dot Product: Geometry

Dot product is the magnitude of the projection of one vector onto another.



Cross Product

- Can take the cross product of two vectors of the same dimension.
- Result is a vector of the same dimension.

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \times \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} y_1 z_2 - z_1 y_2 \\ z_1 x_2 - x_1 z_2 \\ x_1 y_2 - y_1 x_2 \end{bmatrix}$$

Transpose of a Vector

If **v** is a row vector, **v**^{T} is a column vector and vice-versa

$$\begin{bmatrix} x & y & z \end{bmatrix}^T = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \begin{bmatrix} x \\ y \\ z \end{bmatrix}^T = \begin{bmatrix} x & y & z \end{bmatrix}$$

Transpose of a Matrix

- The transpose of an *r* x *c* matrix **M** is a *c* x *r* matrix called **M**^{T}.
- Take every row and rewrite it as a column.
- Equivalently, flip about the diagonal

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}^T = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$$

Multiplying By a Scalar

- Can multiply a matrix by a scalar.
- Result is a matrix of the same dimension.
- To multiply a matrix by a scalar, multiply each component by the scalar.

$$k\mathbf{M} = k\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \\ m_{41} & m_{42} & m_{43} \end{bmatrix} = \begin{bmatrix} km_{11} & km_{12} & km_{13} \\ km_{21} & km_{22} & km_{23} \\ km_{31} & km_{32} & km_{33} \\ km_{41} & km_{42} & km_{43} \end{bmatrix}$$

Matrix Multiplication

Multiplying an *r* x *n* matrix **A** by an *n* x *c* matrix **B** gives an *r* x *c* result **AB**.

$$
\underset{\substack{r \times n \\ \boxed{4} \times 2}}{\overset{\mathbf{A}}{\begin{bmatrix} ? & ? \\ ? & ? \\ ? & ? \\ ? & ? \end{bmatrix}}}
\quad
\underset{\substack{n \times c \\ 2 \times \boxed{5}}}{\overset{\mathbf{B}}{\begin{bmatrix} ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \end{bmatrix}}}
=
\underset{\substack{r \times c \\ \boxed{4 \times 5}}}{\overset{\mathbf{AB}}{\begin{bmatrix} ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \end{bmatrix}}}
$$

2 x 2 Case

$$\mathbf{AB} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

3 x 3 Case

$$\mathbf{AB} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} & a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} & a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} & a_{31}b_{13} + a_{32}b_{23} + a_{33}b_{33} \end{bmatrix}$$

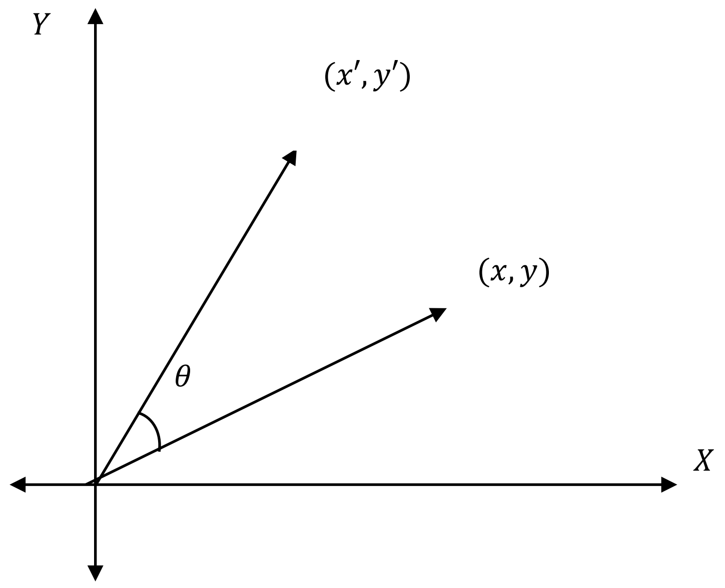
Matrix Times Column Vector Multiplication

Can multiply a matrix times a column vector.

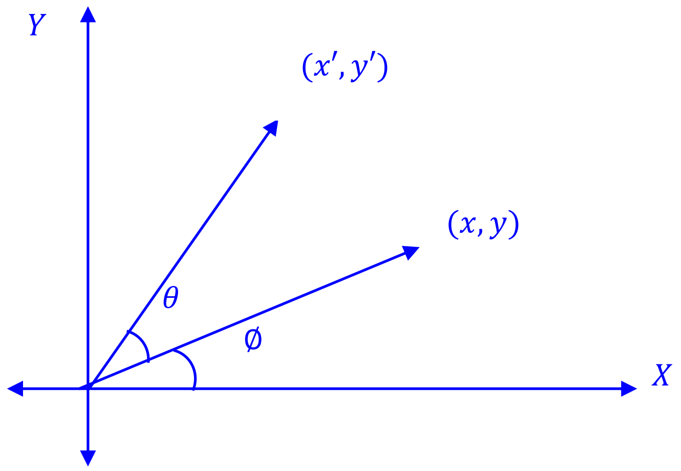
$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} xm_{11} + ym_{12} + zm_{13} \\ xm_{21} + ym_{22} + zm_{23} \\ xm_{31} + ym_{32} + zm_{33} \end{bmatrix}$$

Problem

- Given a point (x,y) in 2D coordinate system, find the new coordinate position (x’, y’) of this point if it is rotated by an angle θ around the origin. Your answer should be in terms of the original position and the rotation angle θ.



Problem



- Let the vector from origin to point (x, y) be of length r and makes an $\emptyset$ angle from the X-axis.
- Then, $x = r\ cos(\emptyset)$ and $y = r\ sin(\emptyset)$. Now let's calculate x' and y' in terms of x, y and θ.
- $x' = rcos(\theta + \emptyset) = r[\cos(\theta)\cos(\emptyset) - \sin(\theta)\sin(\emptyset)]$
  $= [rcos(\emptyset)]\cos(\theta) - [rsin(\emptyset)]\sin(\theta)$
  $= xcos(\theta) - ysin(\theta)$
- $y' = rsin(\theta + \emptyset) = r[\sin(\theta)\cos(\emptyset) + \cos(\theta)\sin(\emptyset)]$
  $= [rcos(\emptyset)]\sin(\theta) + [rsin(\emptyset)]\cos(\theta)$
  $= xsin(\theta) + ycos(\theta)$

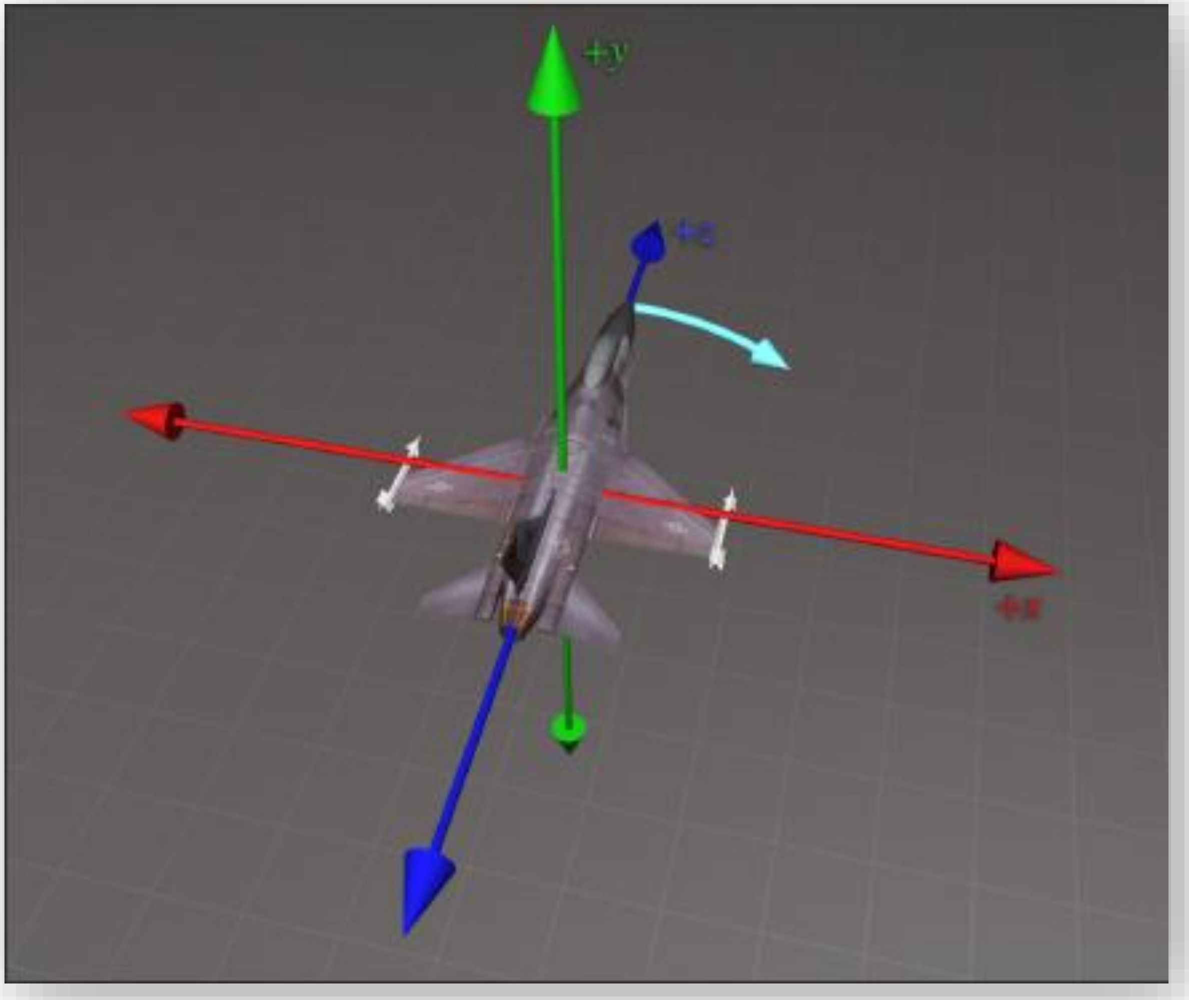
How to Represent Orientation

1. Matrices
2. Euler angles
3. Quaternions

Euler Angles

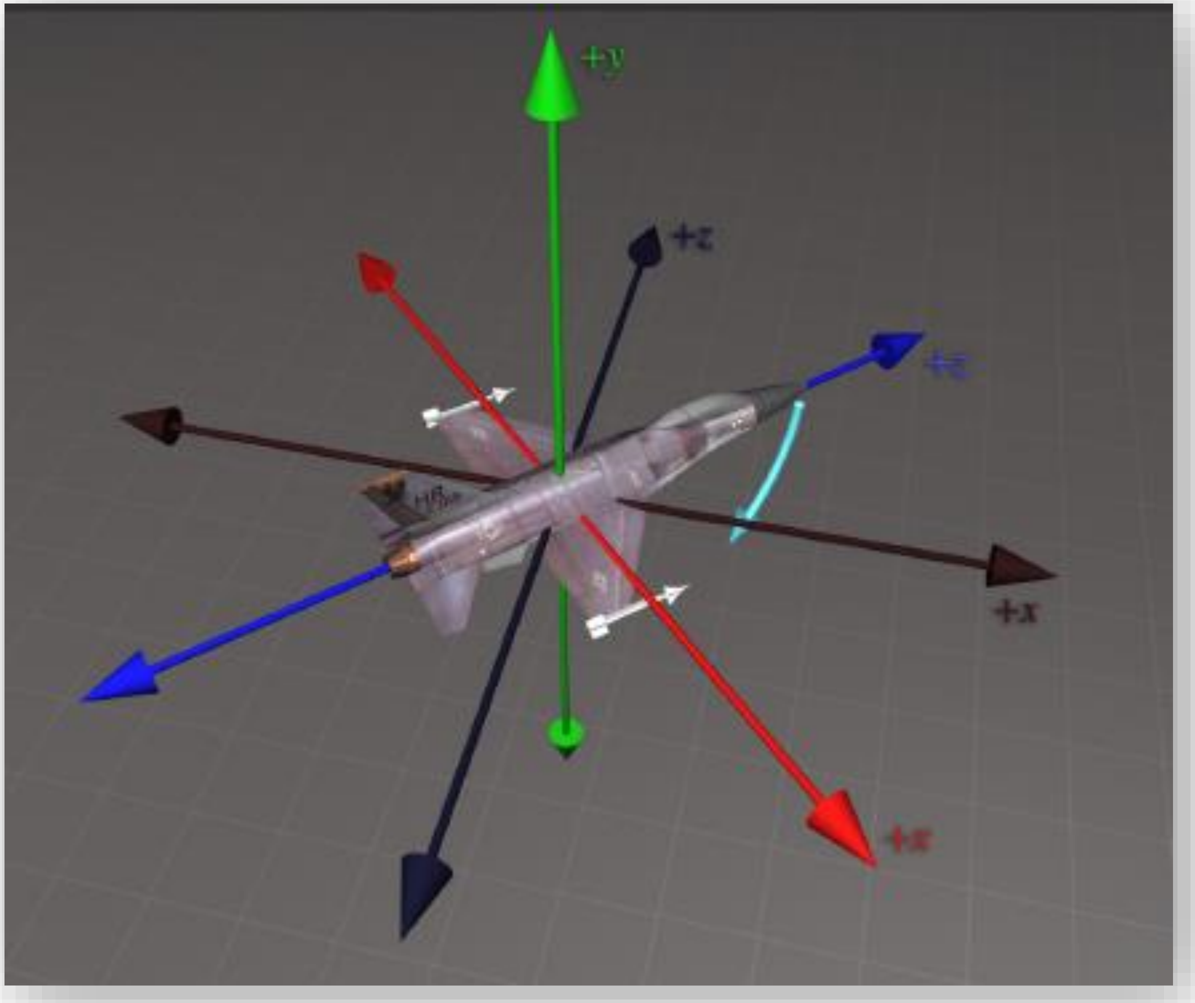
- Specify orientation as a series of 3 angular displacements from upright space to object space.
- Which axes? Which order?
- Need a convention.
- Heading-pitch-bank
  - Heading: rotation about *y* axis (aka “yaw”)
  - Pitch: rotation about *x* axis
  - Bank: rotation about *z* axis (aka “roll”)

Heading



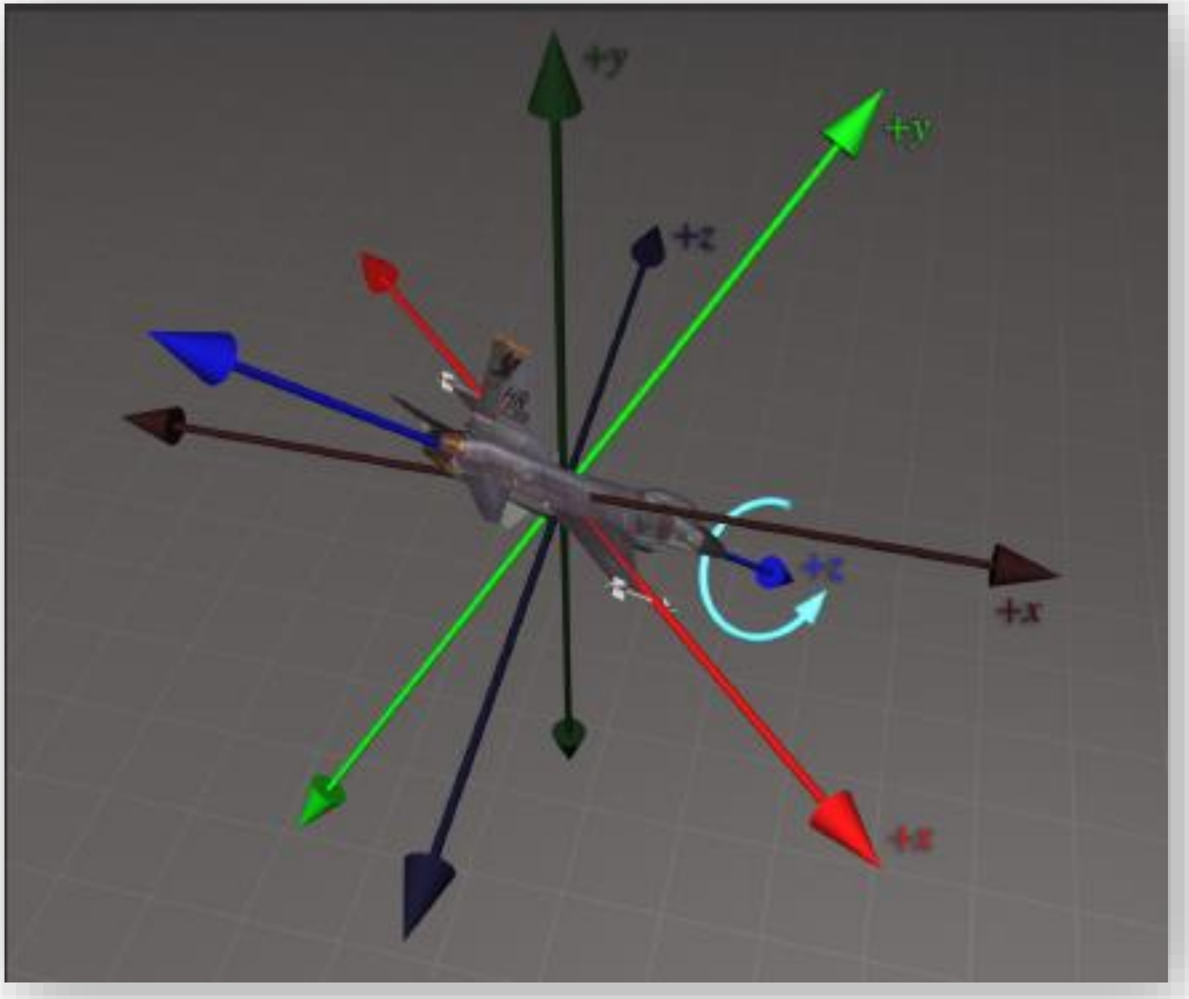
Rotation about y-axis

Pitch



Rotation about x-axis

Bank



Rotation about z-axis

Advantages of Euler Angles

- Easy for humans to use.
- Really the only option if you want to enter an orientation by hand.

- Minimal space – 3 numbers per orientation.
    - Bottom line: if you need to store a lot of 3D rotational data in as little memory as possible, as is very common when handling animation data, Euler angles are the best choices.

Good Luck for the exam !!