BCA614 Bilgisayar Animasyonu II

Ders 2 : Animasyona Giriş

Zümra Kavafoğlu

Computer Animation

Computer Animation: Any computer based computation used in producing images intended to create the perception of motion.

- In general any value that can be changed can be animated.
- An object's position and orientation are obvious candidates for animation, but other attributes in the scene can also be animated like the object's shape, its texture coordinates, the light source or camera parameters.

https://ohmy.disney.com/movies/2016/07/20/twelve-principles-animation-disney/

Motion Perception

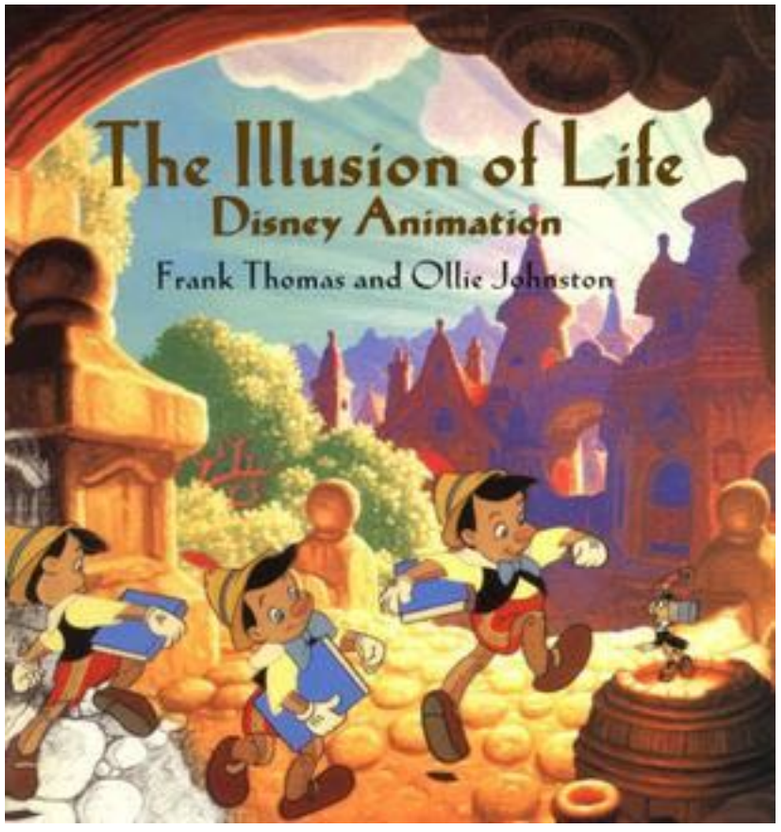
A series of images, when displayed in rapid succession are perceived by an observer as a single moving image.

Hand-drawn animation (Traditional Animation)

Traditional animation (or **classical animation**, **cel animation** or **hand-drawn animation**) is an animation technique where each frame is drawn by hand. The technique was the dominant form of animation in cinema until the advent of computer animation.

Hand-drawn animation (Traditional Animation)

Before studying animation made by computer, it's useful to understand the basic principles of animation made by hand.

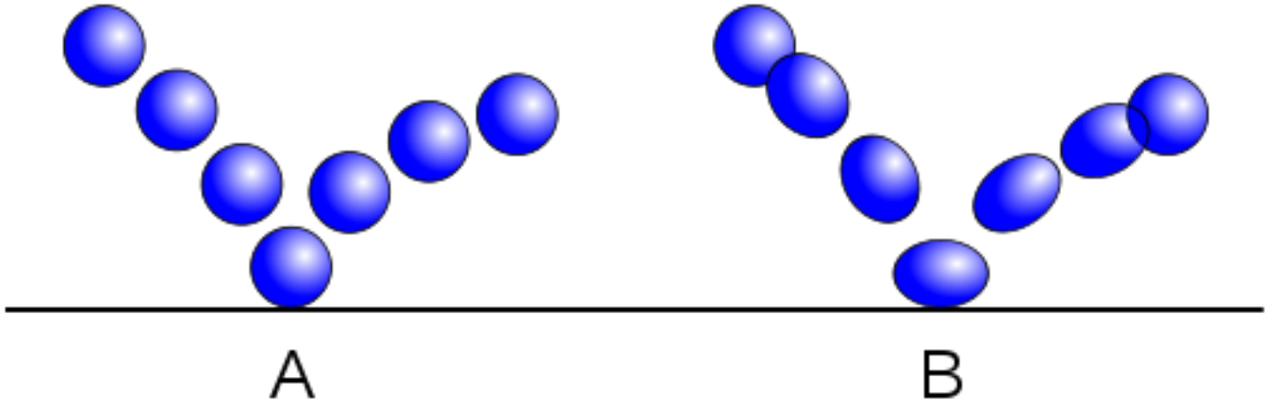


The 1981 book "The Illusion of Life: Disney Animation" by Disney animators Ollie Johnston and Frank Thomas introduces the "12 basic principles of animation" for guiding the hand-drawn animation

Basic principles of animation

1. Squash and Strech

The most important principle is "squash and stretch", the purpose of which is to give a sense of weight and flexibility to drawn objects.



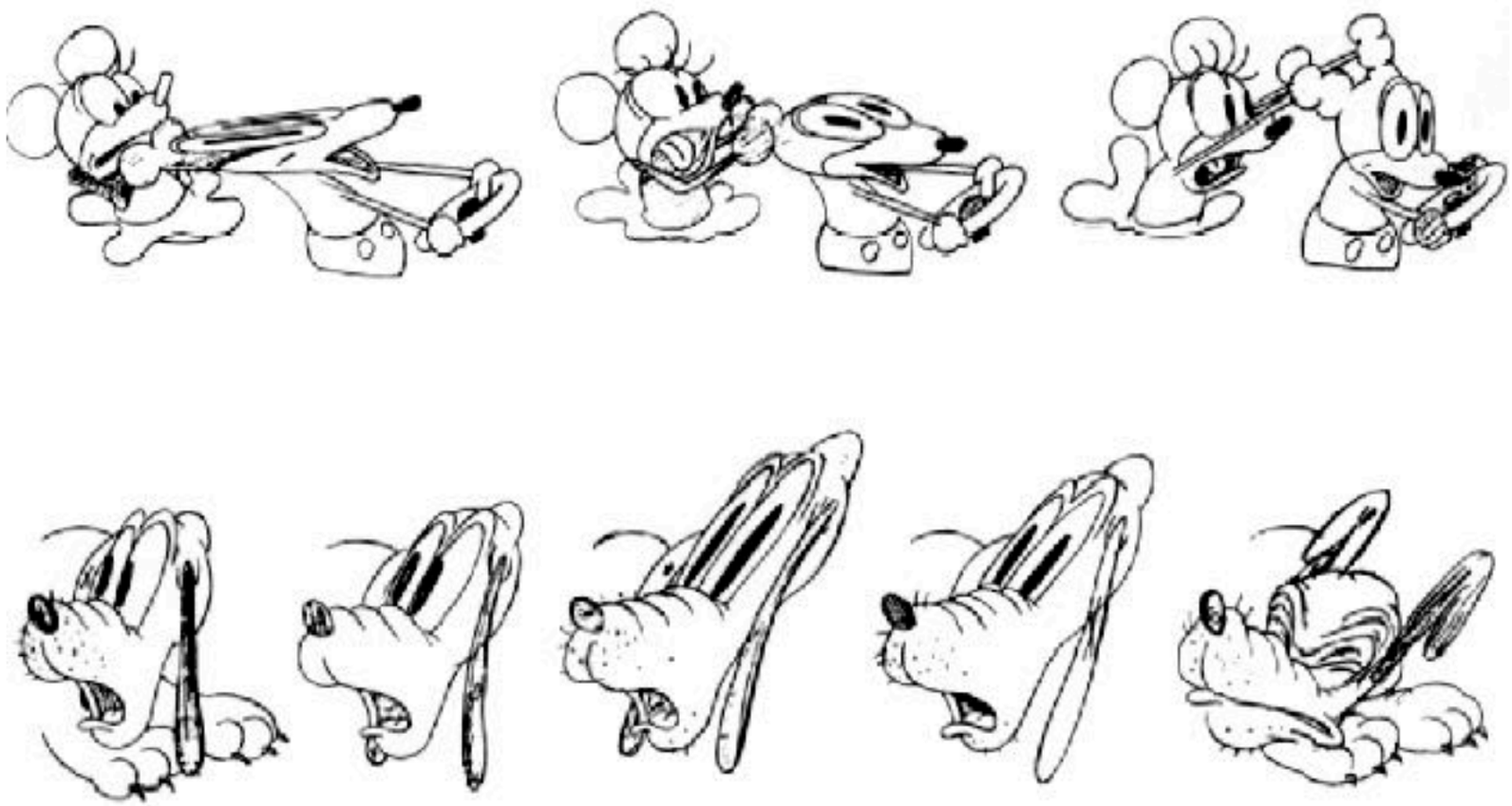
What's the difference between A and B?
Which one seems more realistic?

- With "squash and strech" applied, objects get longer or flatter to emphasize their speed, momentum and weight.

https://en.wikipedia.org/wiki/12_basic_principles_of_animation

Basic principles of animation

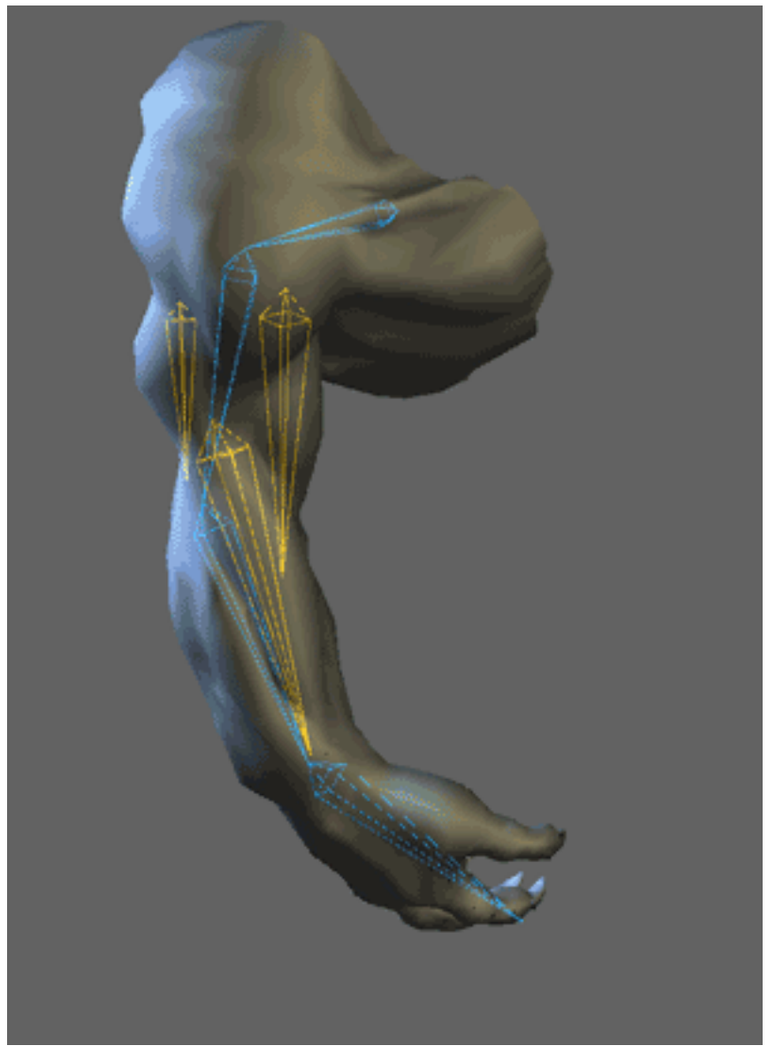
- Taken to an extreme point, a figure stretched or squashed to an exaggerated degree can have a comical effect.



- http://facweb.cs.depaul.edu/sgrais/squash_and_stretch.htm

Basic principles of animation

- In realistic animation, however, the most important aspect of this principle is the fact that an object's volume *does not* change when squashed or stretched.

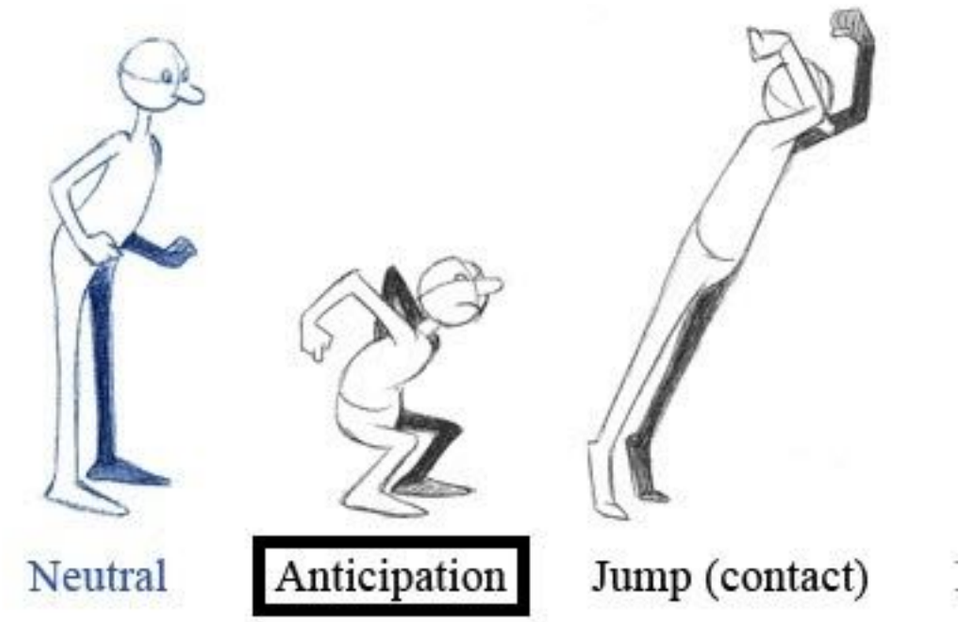


- http://facweb.cs.depaul.edu/sgrais/squash_and_stretch.htm

Basic principles of animation

2. Anticipation

Anticipation is used to prepare the audience for an action, and to make the action appear more realistic.



https://en.wikipedia.org/wiki/12_basic_principles_of_animation

Basic principles of animation

3. Follow through and overlapping action

Following through means that separate parts of a body will continue moving after the character has stopped.

Overlapping action is the tendency for parts of the body to move at different rates (an arm will move on different timing of the head and so on).

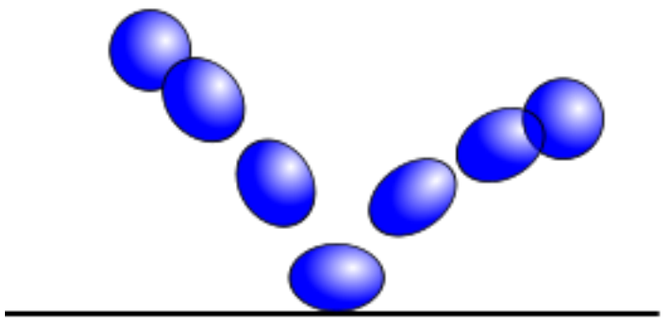
A third related technique is “drag”, where a character starts to move and parts of him take a few frames to catch up. These parts can be inanimate objects like clothing or the antenna on a car, or parts of the body, such as arms or hair.

https://danterinaldidesign.com/principles-animation-follow-overlap/#!prettyPhoto
https://ohmy.disney.com/movies/2016/07/20/twelve-principles-animation-disney/

Basic principles of animation

4. Ease in and Ease out (Slow in and slow out)

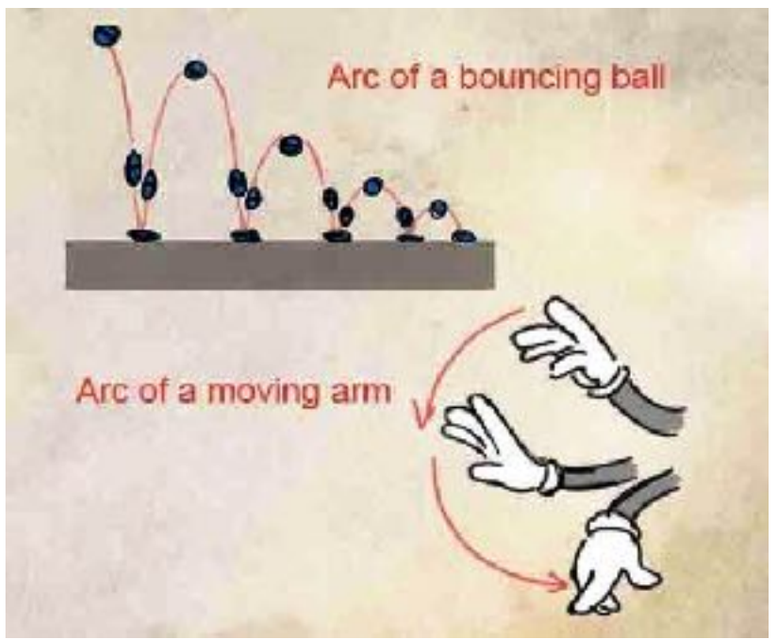
The movement of the human body, and most other objects, needs time to accelerate and slow down. For this reason, animation looks more realistic if it has more drawings near the beginning and end of an action, emphasizing the extreme poses, and fewer in the middle.



Basic principles of animation

5. Arc

Most natural action tends to follow an arched trajectory, and animation should adhere to this principle by following implied "arcs" for greater realism. This technique can be applied to a moving limb by rotating a joint, or a thrown object moving along a parabolic trajectory. The exception is mechanical movement, which typically moves in straight lines



http://blog.animationmentor.com/arc-the-12-basic-principles-of-animation/

Basic principles of animation

For more principles see

https://ohmy.disney.com/movies/2016/07/20/twelve-principles-animation-disney/

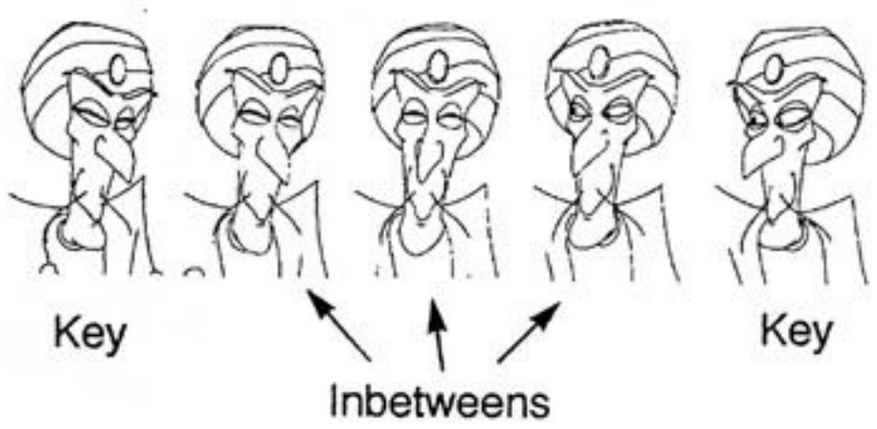
Technical Background

Spaces and transformations

- Remember your Computer Graphics Lecture
- Read Section 2.1 of Computer Animation book by Rick Parent

Keyframe animation (Traditional animation)

- The *lead animator* draws the keyframes and sometimes the breakdowns and the inbetween animators draw the inbetween frames using the keyframes and breakdowns.
- A **key** (also called "**extreme**") is a key moment in an animated sequence, where the motion is at its extreme. The number of keys in an animated sequence depends on how complicated the movement is.
- A **keyframe** is a drawing that corresponds to a key moment.
- A **breakdown** comes between keys to help the keyframe animator describe the action to inbetween animators.
- **Inbetweens** fill the gaps between keyframes.



https://www.utdallas.edu/atec/midori/Handouts/keyframing.htm

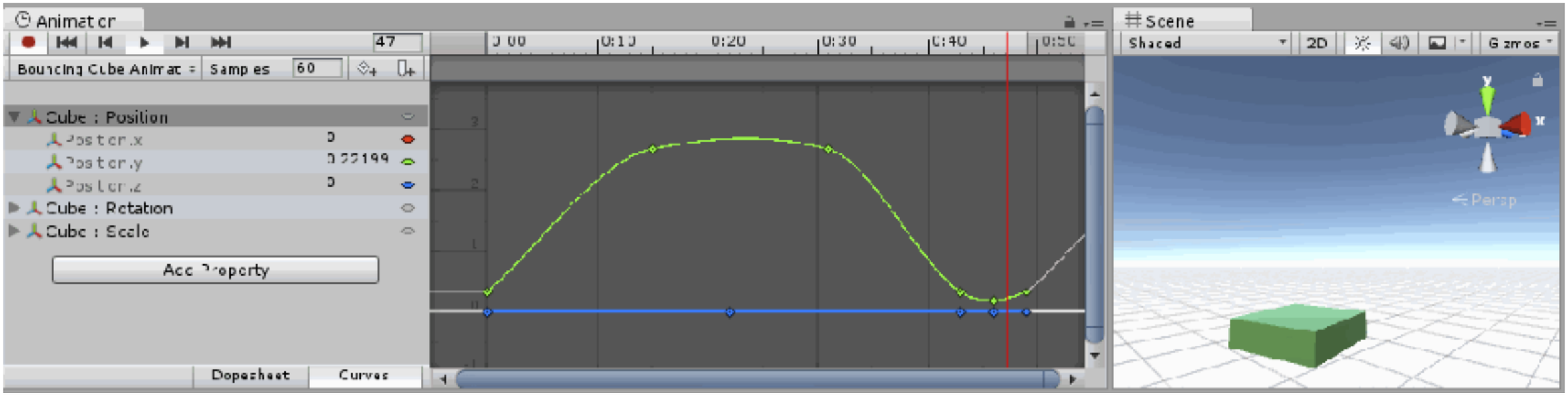
Keyframe animation (Computer animation)

- Animators specify keyframes
- Computers do inbetweening with interpolation.
- Type of interpolation is important:
  - linear, cubic, parametric curves
- Speed characteristics of interpolation should also be specified
  - constant velocity, ease in ease out, vs.

Keyframe animation (Computer animation)

- Specify the properties of the entity to animate
  - position
  - orientation
  - scale
  - color
  - etc.
- Specify the keys for each property.
  - A key is a pair of frame number and property value.
- Specify the interpolation type for inbetweening
  - linear, cubic, parametric curves
- Specify the speed characteristics of interpolation
  - constant velocity, ease in ease out, vs.

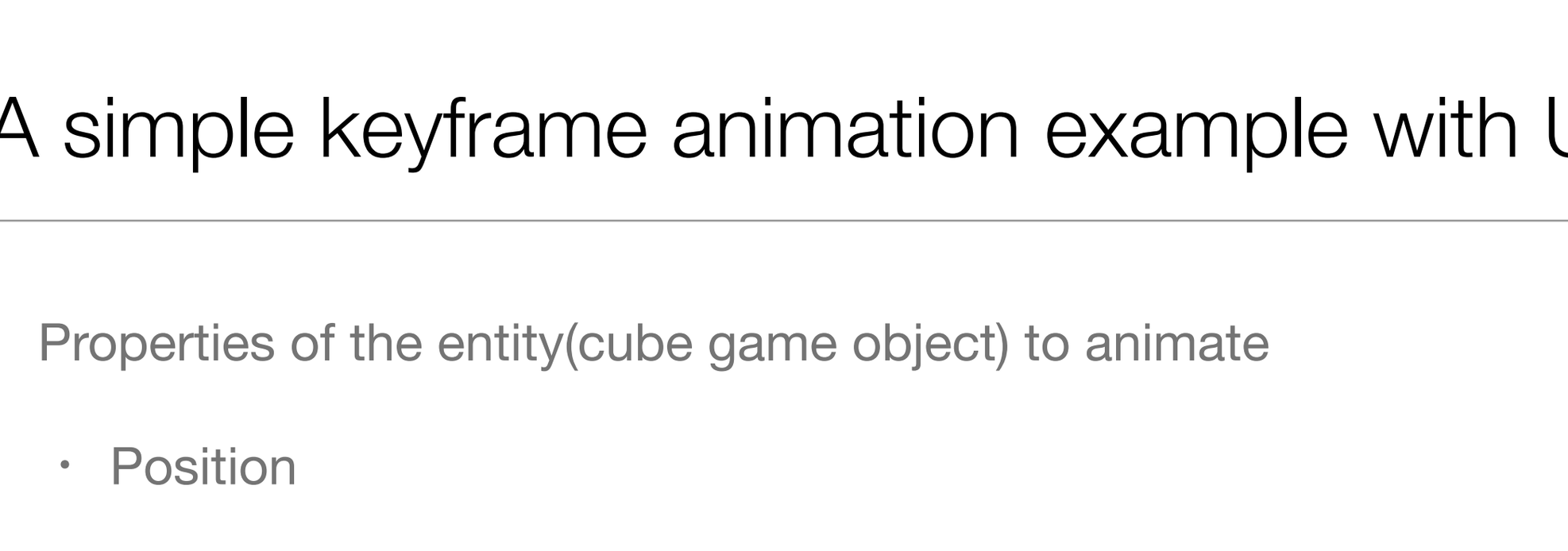
A simple keyframe animation example with Unity



https://docs.unity3d.com/560/Documentation/Manual/animeditor-AnimationCurves.html

A simple keyframe animation example with Unity

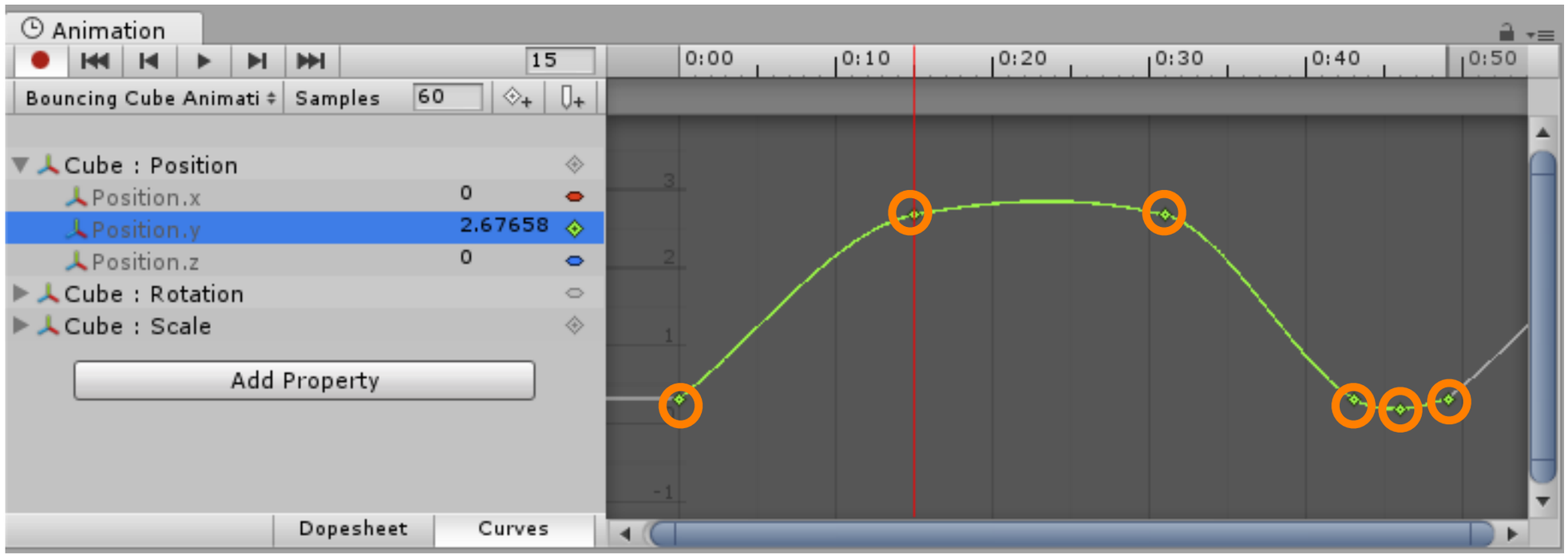
- Properties of the entity(cube game object) to animate
  - Position
  - Rotation
  - Scale



https://docs.unity3d.com/560/Documentation/Manual/animeditor-AnimationCurves.html

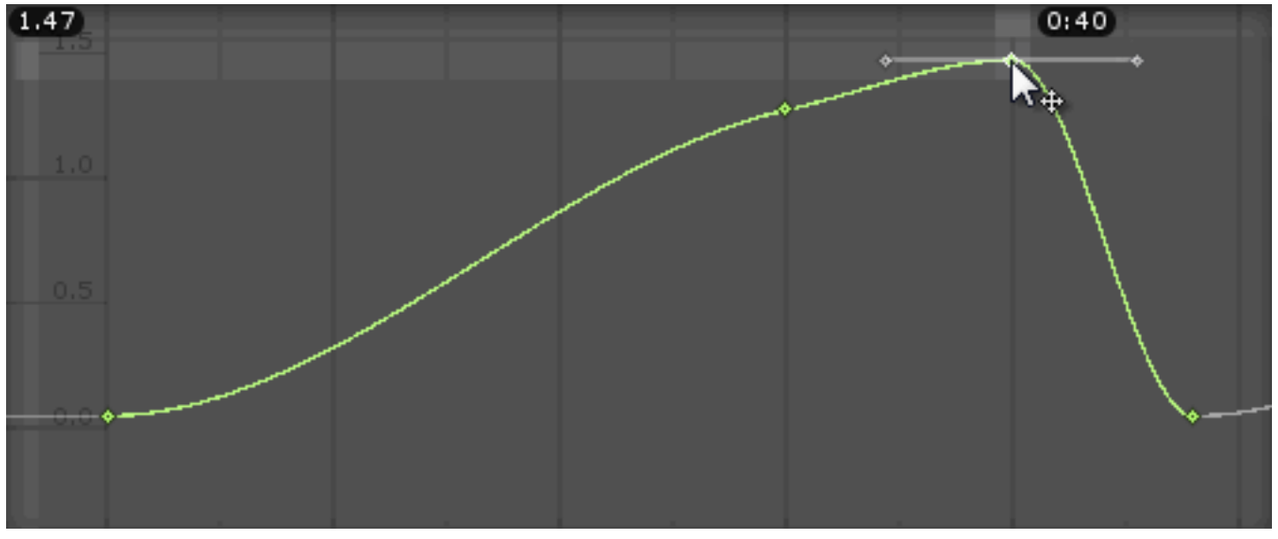
A simple keyframe animation example with Unity

- 6 keyframes : (time, value) pairs for y component of position



A simple keyframe animation example with Unity

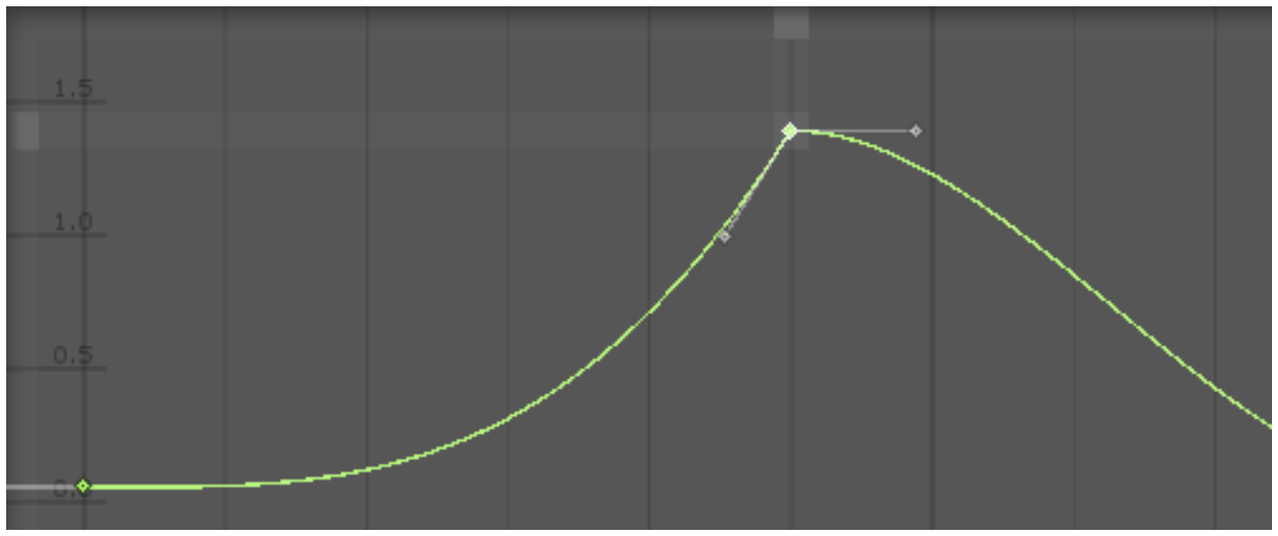
- Interpolation curve between keyframes is created automatically. Different tangent modes are available for specifying the shape of the interpolation curve. For example with the automatic tangent mode shown below, the tangents are automatically set to make the curve pass smoothly through the keys.

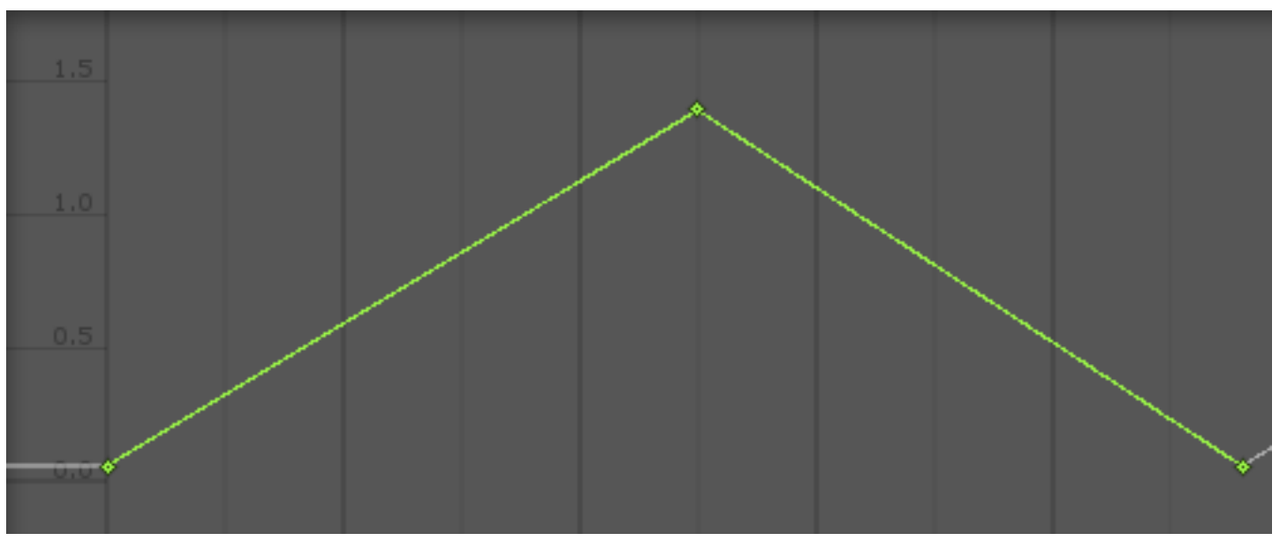


https://docs.unity3d.com/560/Documentation/Manual/EditingCurves.html

A simple keyframe animation example with Unity

- With different tangent modes different interpolation curves can be created

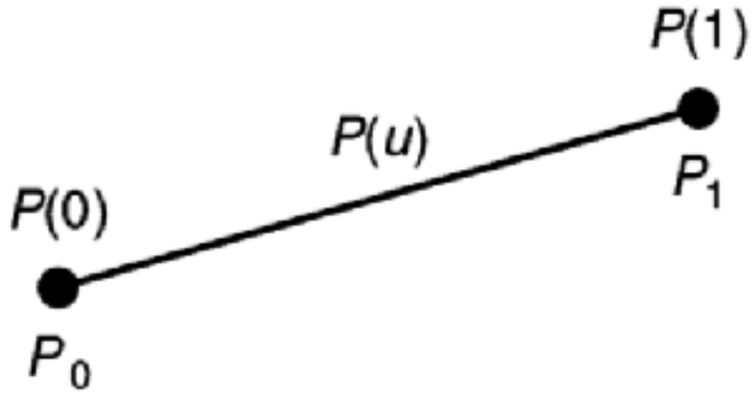




https://docs.unity3d.com/560/Documentation/Manual/EditingCurves.html

How to choose the appropriate interpolation function?

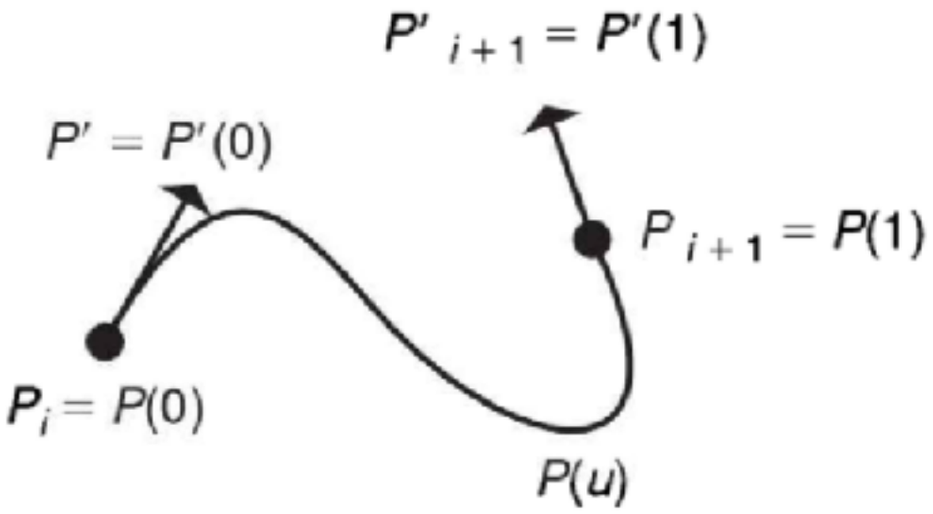
- Simple linear interpolation:
  - Given beginning and end points P(0) and P(1), inbetween points are calculated with P(u).



$$P(u) = (1 - u)P_0 + uP_1$$

How to choose the appropriate interpolation function?

- Hermite interpolation
  - generates a cubic polynomial from one point to another
  - in addition to specifying beginning and end points (P_i, P_{i+1}), beginning and ending tangent vectors (P'_i, P'_{i+1}) should also be specified.



$$P(u) = U^T M B$$

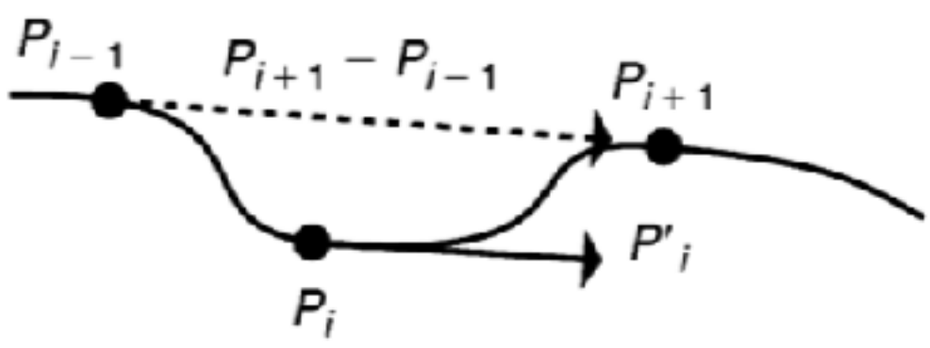
$$U^T = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} P_i \\ P_{i+1} \\ P_{i+1} \\ P'_i \\ P'_{i+1} \end{bmatrix}$$

How to choose the appropriate interpolation function?

- Catmull-Rom spline
  - can be viewed as a Hermite curve in which the tangents at the interior control points are automatically generated according to a relatively simple geometric procedure
  - For each interior point P_i the tangent vector is calculated as the average of the adjacent control points



$$P'_i = (1/2)(P_{i+1} - P_{i-1})$$

$$P(u) = U^T M B$$

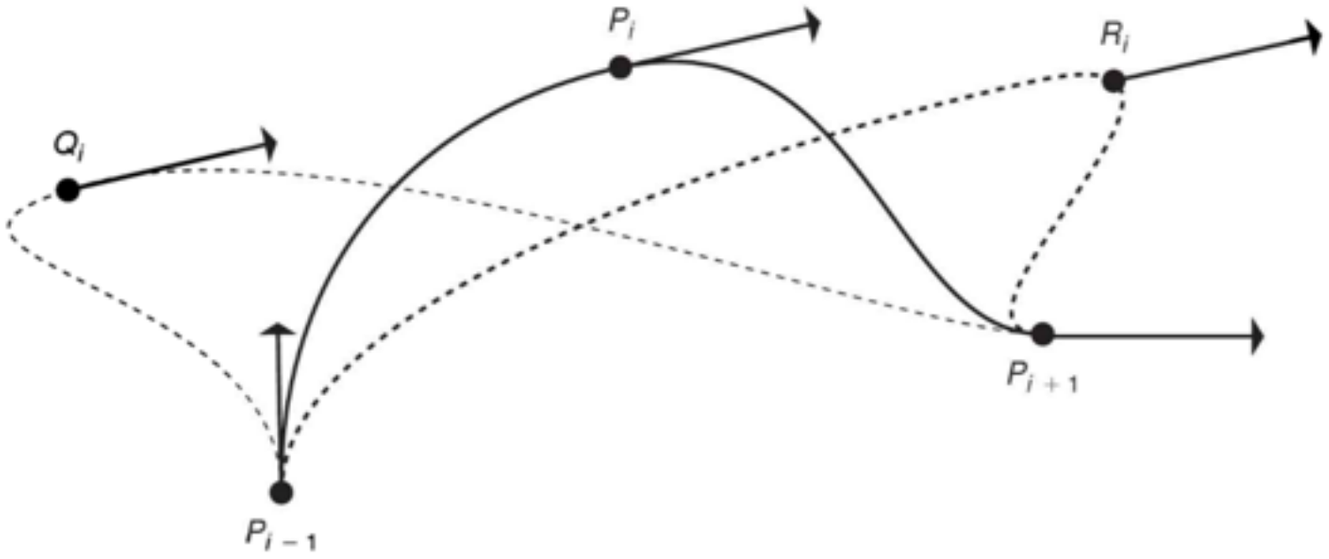
$$U^T = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}$$

$$M = \frac{1}{2}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

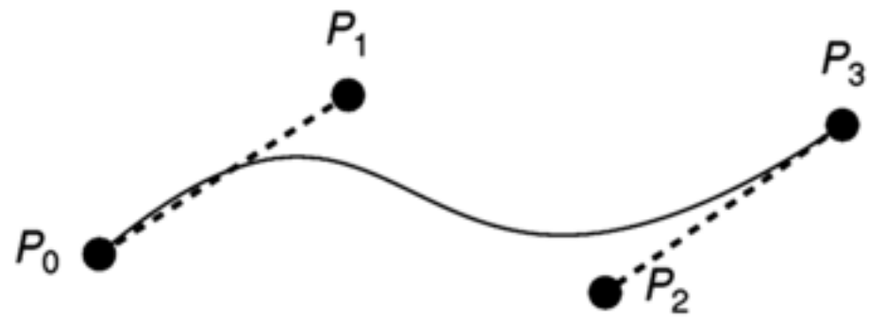
How to choose the appropriate interpolation function?

- A drawback of Catmull-Rom calculation is that an internal tangent vector is not dependent on the position of the internal point relative to its two neighbours. In the figure below all three points for the i^{th} point would have the same tangent vector.



How to choose the appropriate interpolation function?

- Bezier interpolation / approximation
  - A cubic Bezier curve is defined by the beginning and ending point which are interpolated and two interior points which control the shape of the curve
  - Similar to Hermite interpolation but Bezier uses interior points for shape control, instead of beginning and ending tangent vectors.



- With Bezier interpolation curves of arbitrary order can be defined: if three interior control points are defined then the resulting curve becomes quartic, if four interior control points are defined then the resulting curve becomes quintic.

How to choose the appropriate interpolation function?

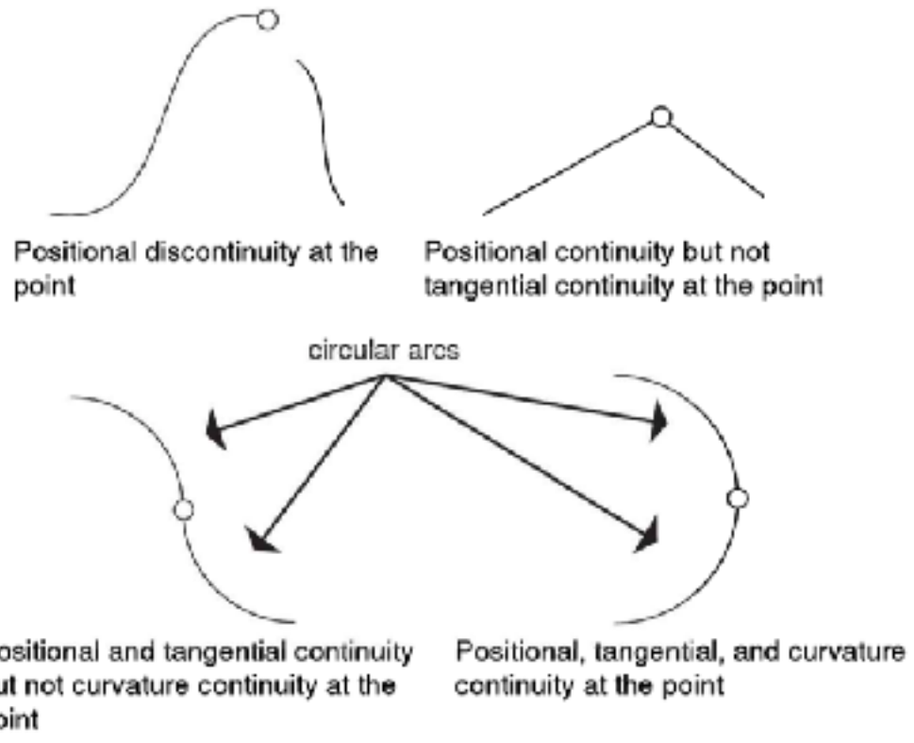
- **Interpolation vs. Approximation**
  - Interpolation: All of the given values represent actual positions that the curve should pass through
    - ex: Linear interpolation, Hermite interpolation, Catmull-Rom spline
  - Approximation: Some of the values are given to control the shape of the curve and do not represent actual positions that the curve will intersect
    - ex: Bezier and B-spline curves

How to choose the appropriate interpolation function?

- **Complexity**
    - Computational efficiency is effected by the complexity of the underlying interpolation equation. The simpler the underlying equations of the interpolating function, the faster its evaluation.
    - A compromise between smoothness and efficiency is needed
        - Polynomials are easy to compute
        - Piecewise cubic polynomials are the lowest degree polynomials that provide sufficient smoothness
        - A polynomial whose degree is lower than cubic does not provide a point of inflection between two endpoints, therefore it might not fit smoothly to certain data points.
        - Using a polynomial whose degree is higher than cubic doesn’t provide any significant advantages and is more costly to evaluate

How to choose the appropriate interpolation function?

- **Continuity (Smoothness)**
  - The smoothness of the resulting curve is important
  - Mathematically, smoothness is determined by how many of the derivatives of the curve equation are continuous
    - Zero-order continuity is the continuity of the curve points
    - First-order continuity is the continuity of the tangents
    - Second-order continuity is the continuity of the curvature
      - In animation applications first order continuity suffices for the spatial curves



How to choose the appropriate interpolation function?

- **Continuity of piecewise-defined curves**

- In most applications a curve interpolating or approximating more than a few points is defined piecewise; the curve is defined by a sequence of segments where each segment is defined by a single parametric function and shares its endpoints with the functions of adjacent segments.

- Curve segments are mostly designed as cubic polynomials, therefore the issue is not the continuity within a curve segment but the continuity at the junction between the curve segments

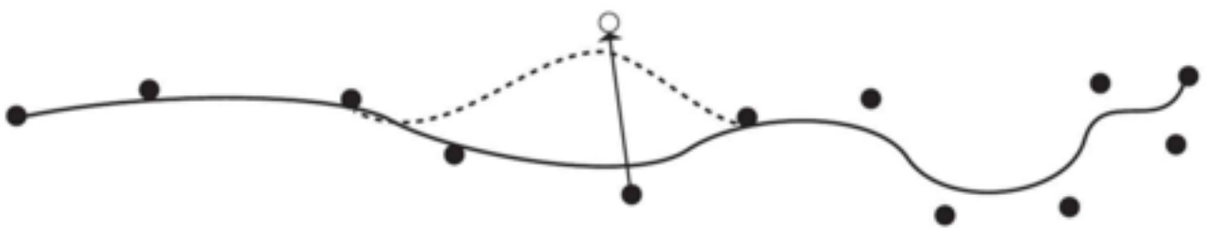
    - Hermite, Catmull-Rom, cubic Bezier curves can all produce first-order continuity between curve segments

    - A cubic B-spline is second order continuous everywhere

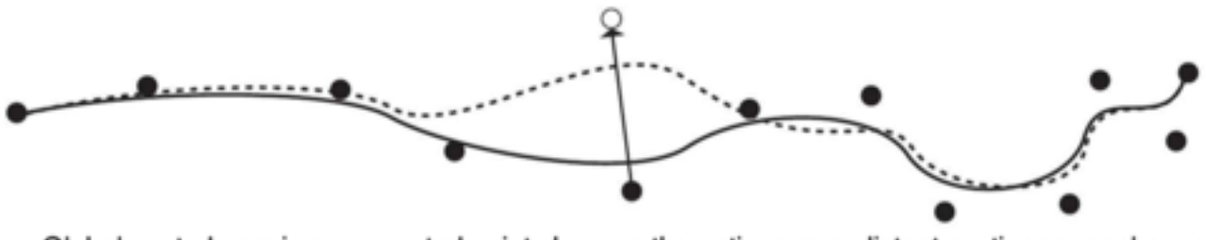
    - All of these curves provide sufficient continuity for many animation applications.

How to choose the appropriate interpolation function?

- **Local vs. global control**
- When designing a curve a user often repositions a point in order to tweak just the part of the curve close to the control point.
- It is usually considered an advantage if a change in a single control point has an effect on a limited region of the curve as opposed to effecting the entire curve.
- A formulation in which control points have a limited effect on the curve is referred to as providing local control.
- If repositioning one control point changes the shape of the entire curve, then the formulation provides only global control
- Almost all of the composite curves provide local control: parabolic blending, Catmull-Rom splines, composite cubic Bezier and cubic B-splines.



Local control: moving one control point only changes the curve over a finite bounded region



Global control: moving one control point changes the entire curve; distant sections may change only slightly

How to choose the appropriate interpolation function?

- **For more on interpolation functions, arc length parametrization and speed control see Sections 3.1 and 3.2 and Appendix B.5 of Computer Animation book by Rick Parent**

Interpolating Orientation

- Interpolating orientations of 3d entities is not as straightforward as interpolating positions

- Orientation representation is quite important for achieving geometrically correct and realistic animations.

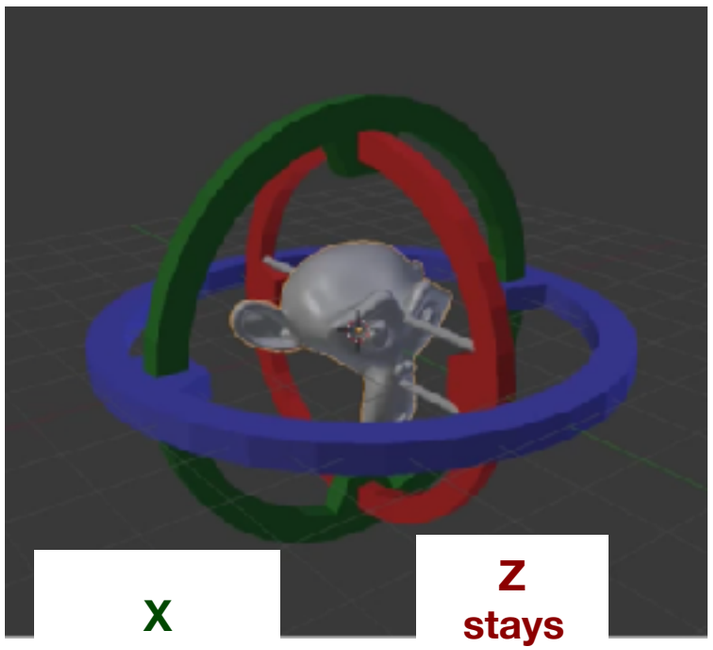
Euler angles representation

- Euler angles representation
  - represents the orientation with 3 angles, one for x axis, one for y axis and one for z axis
  - compact for orientation representation
- BUT
  - although it seems very clear to understand, it’s not such straightforward to understand and use, since rotation axes are effected from each other’s rotation
  - the order of rotation is very important and completely changes the resulting orientation.
  - it causes gimbal lock(explained in the next slide)
  - interpolation between orientations can sometimes create unnatural motions.

Euler angles representation

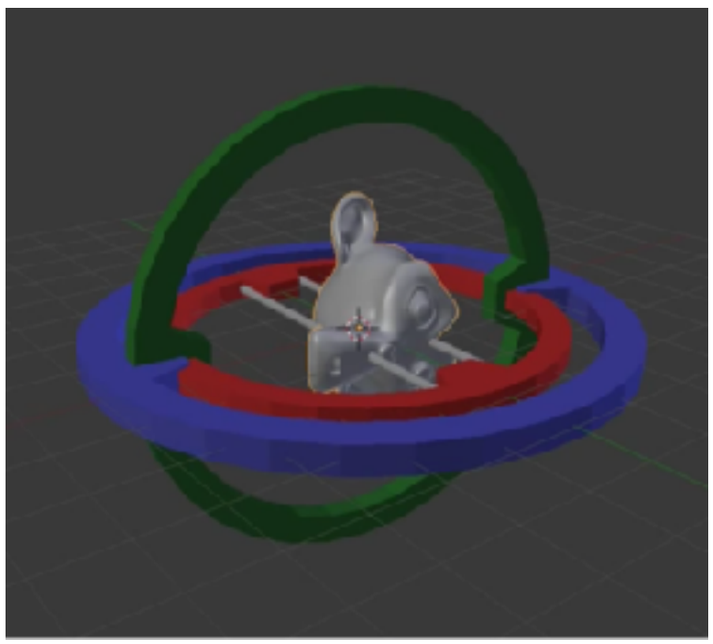
- Let's assume that rotation order is ZXY and We can think of the rotation axes as three nested rings connected to each other. The inner most ring represents the first rotation axis Z, the middle ring represents the second rotation axis X and the outer most represents the last rotation axis Y.
- So why is rotation order important?
    - Rotation of inner most ring doesn't effect other rings, but it's effected from rotation of all other rings. So if the inner most ring represents Z axis, then rotation around z axis is equivalent to rotation around local z axis
    - Rotation of the middle ring effects the innermost ring and it's effected by the outermost ring. Therefore, it's quite complicated to understand its rotation behaviour under different scenarios. But when it rotates 90 degrees, it makes the outer and inner rings aligned, which is called gimbal lock. Gimbal lock causes loss of one rotation axis.
    - Rotation of the outer ring effects all the other rings but it's not effected by others. So if the outer ring represent Y axis then rotation around x axis is equivalent to rotation around global x axis.

Y
stays
worldly



X
causes
gimbal lock

Z
stays
local

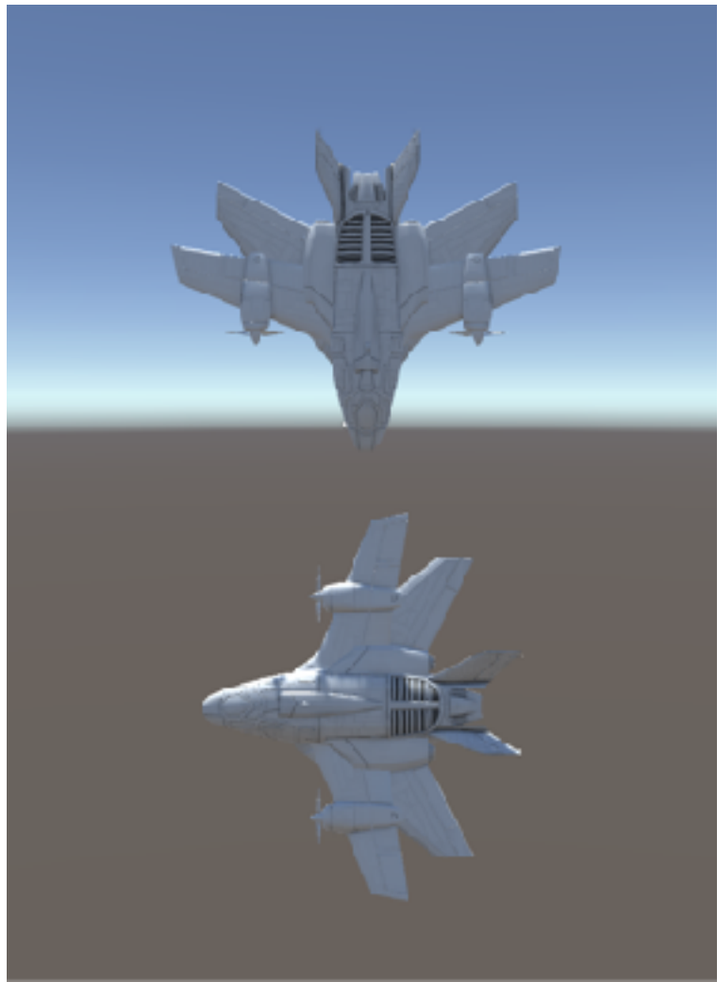


Gimbal lock: Z and Y axes get aligned

https://www.youtube.com/watch?v=Mm8tzzfy1Uw

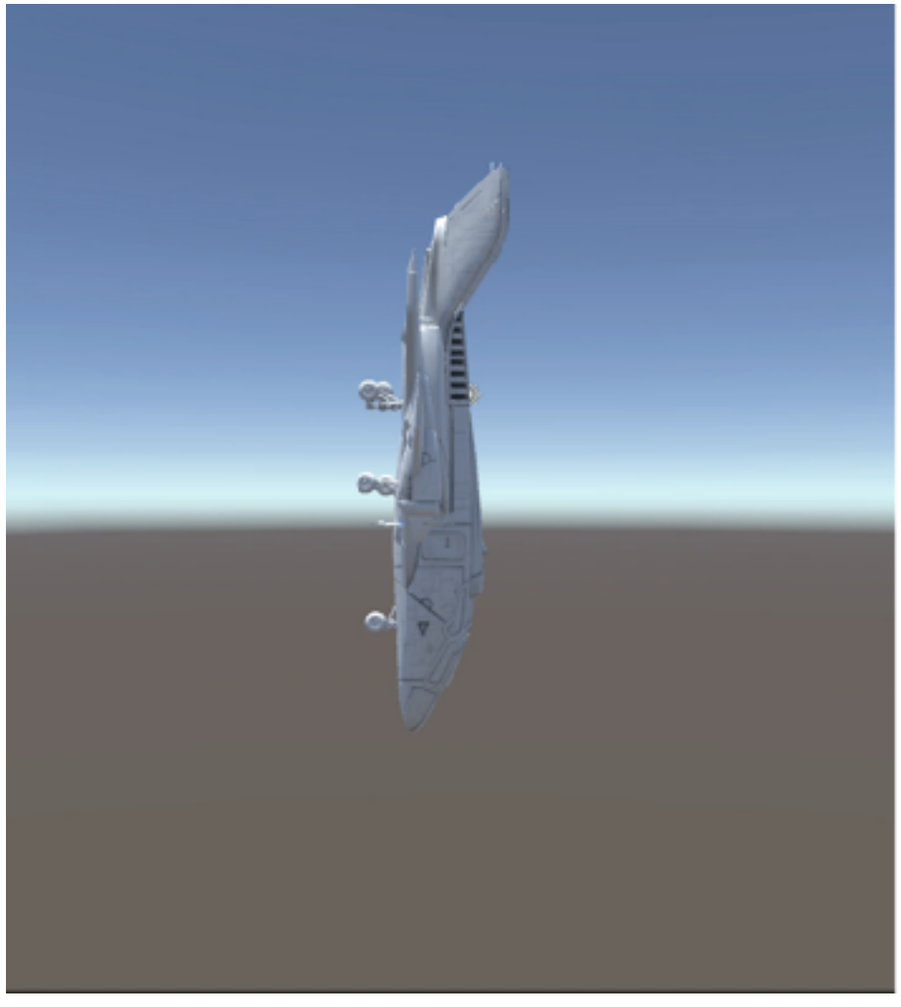
An example with Unity3D

- Let’s consider a 3D aircraft with start orientation (90,0,0) (the one at the top) and ending orientation (0,90,90) (the one at the bottom). The shortest, thus most natural interpolation from start to end would be by rotating the aircraft 90 degrees around world z axis or local y axis, right?
- If we change the z component of the euler angles, the aircraft rotates around local z axis which is aligned with world y axis.
- If we change the y component of the euler angles, the aircraft rotates around world y axis which is aligned with local z axis.
- There is no way of rotating the aircraft around local y-axis or world z-axis? WHY?



An example with Unity3D

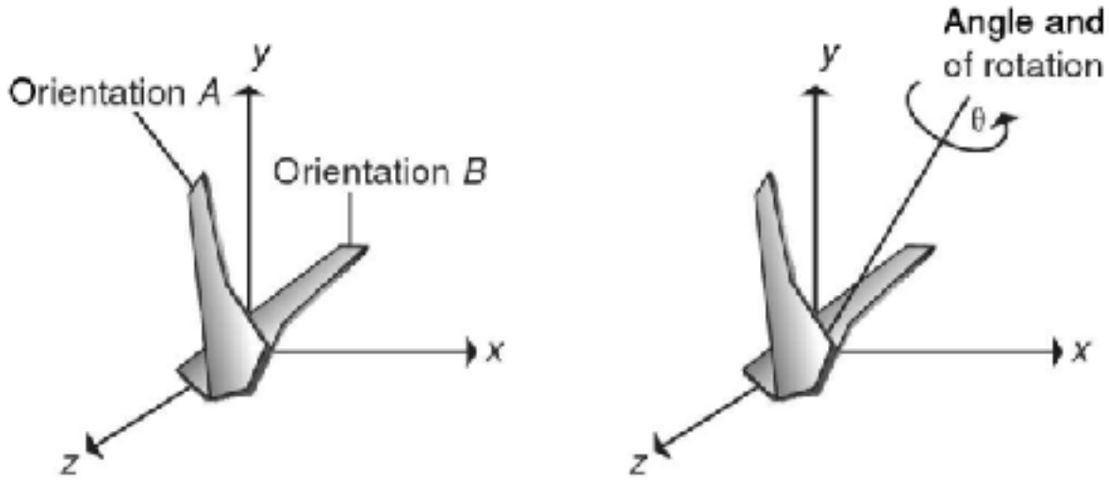
- If we linearly interpolate the components of the start and ending key orientations, an unnatural rotation occurs



- Did you notice the weird rotation around local x? To see why it happens, just calculate the half-way rotation with component-wise linear interpolation.

Angle axis representation

- In the mid 1700s Leonard Euler showed that one orientation can be derived from another by a single rotation about an axis. This is known as Euler rotation theorem.



- Therefore any orientation can be represented by a three dimensional axis A and an angle α.
- However interpolation between two orientations (A_1,α_1) and (A_2,α_2) is not straightforward:
  - You cannot linearly interpolate axes A_1 and A_2 and angles α_1 and α_2(Don't forget, rotation is NOT something linear!)
  - An intermediate axis can be determined by rotating one axis partway toward the other. The axis for this rotation is formed by taking the cross product of two axes

Angle axis representation

- Therefore any orientation can be represented by a three dimensional axis A and an angle α.
- However interpolation between two orientations (A_1,α_1) and (A_2,α_2) is not straightforward:
  - You cannot linearly interpolate axes A_1 and A_2 and angles α_1 and α_2(Don't forget, rotation is NOT something linear!)
  - An intermediate axis can be determined by rotating one axis partway toward the other. The axis for this rotation is formed by taking the cross product of two axes and the angle between the two axes is determined with the inverse cosine of the dot product of the normalized versions of the axes.
  - Note that it's not easy to concatenate a series of rotations with axis angle representation. However, the information contained in this representation can be put in a form in which these operations are easily implemented: quaternions

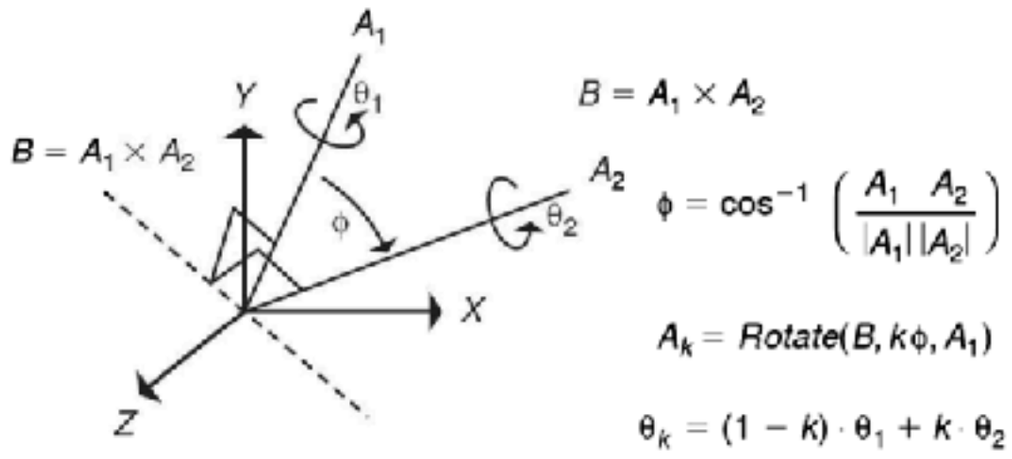


FIGURE 2.22

Interpolating axis-angle representations of (A_1, θ_1) and (A_2, θ_2) by k to get (A_k, θ_k), where 'Rotate(a,b,c)' rotates 'c' around 'a' by 'b' degrees.

Lovely quaternions :)

- Axis angle representation of an orientation can be easily put into an algebraic form called quaternions:
- A quaternion is a four-tuple vector, which is mostly denoted with (x,y,z,w) or [v,w] where v denotes the vector part (x,y,z) and w denotes the scalar part.
- Suppose the orientation of a 3D entity is represented by the 3D unit axis u = (u1,u2,u3) and the angle α. Then the quaternion representation of this orientation is found as below:
  - (x,y,z) = sin(α/2)(u1,u2,u3)
  - w = cos(α/2)
- Then the quaternion q = (u1sin(α/2),u2sin(α/2),u3sin(α/2),cos(α/2)) represents the orientation of the 3D entity.

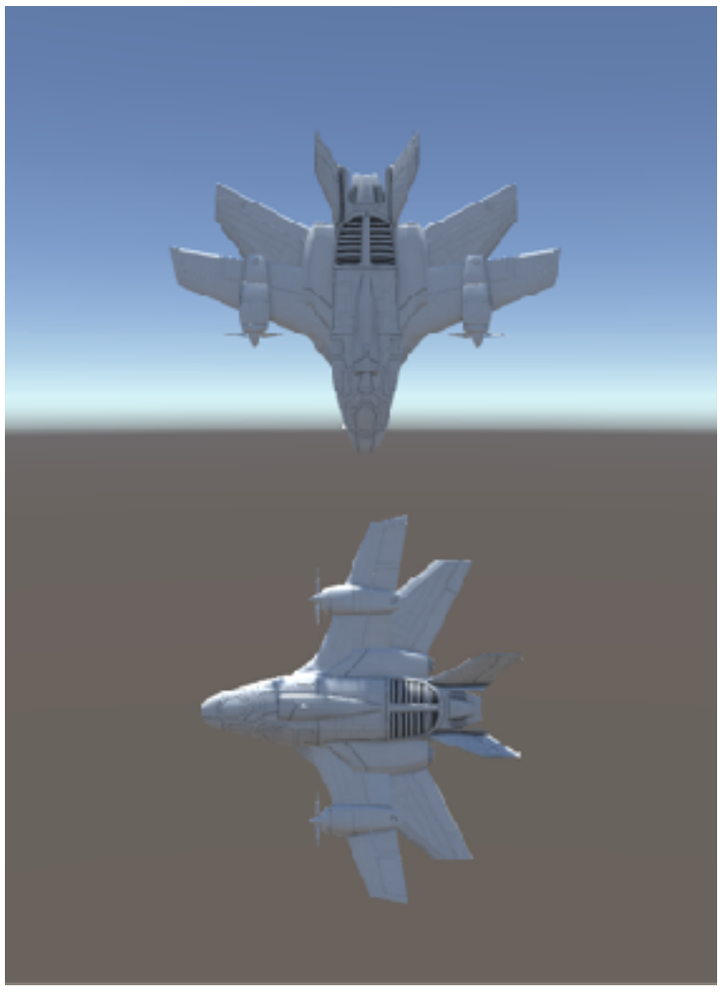
Why use quaternions?

- Series of rotations are easily concatenated into a single representation by using quaternion multiplication:
  - For rotating with first q then p, you can rotate directly with pq
  - if p = [v1,w1] and q = [v2,w2], where v1 and v2 denote the vector parts and w1, w2 denote the scalar parts, then
    - pq = [s1v2 + s2v1 + v1 x v2 , s1s2 - v1·v2]
- No gimbal lock (because it's axis angle based)
- Offers the best interpolation technique for rotations: slerp (spherical linear interpolation)

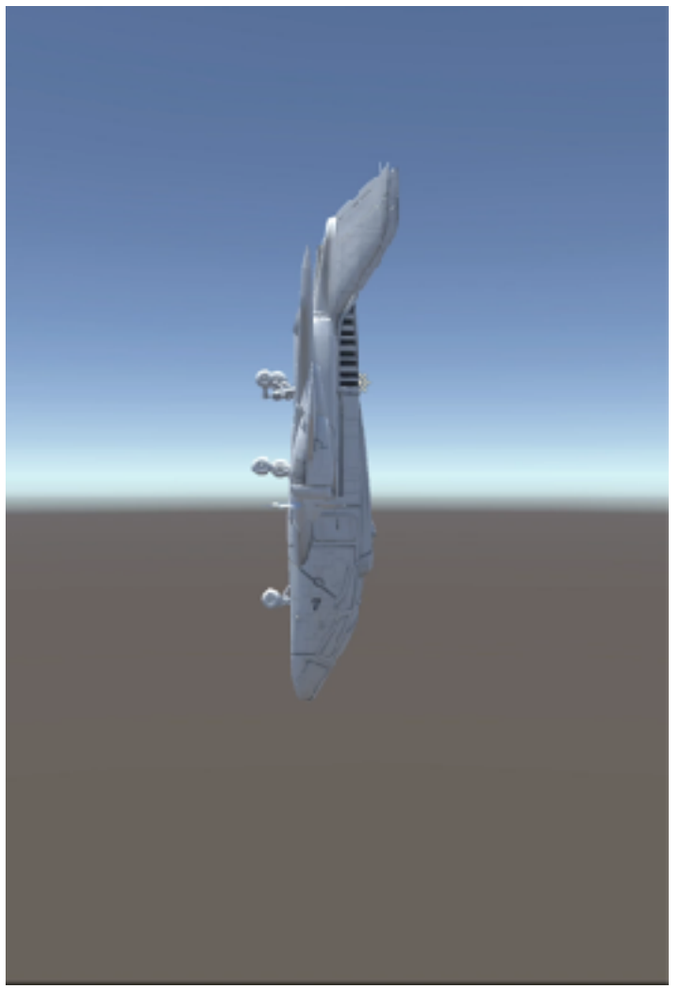
$$slerp(q_1, q_2, u) = \frac{\sin((1-u)\theta)}{sin(\theta)} q_1 + \frac{\sin(u\theta)}{sin(\theta)} q_2$$

SLERP (Spherical Linear Interpolation)

- Remember our example Unity scene where we aim to interpolate between two rotations of a 3D aircraft



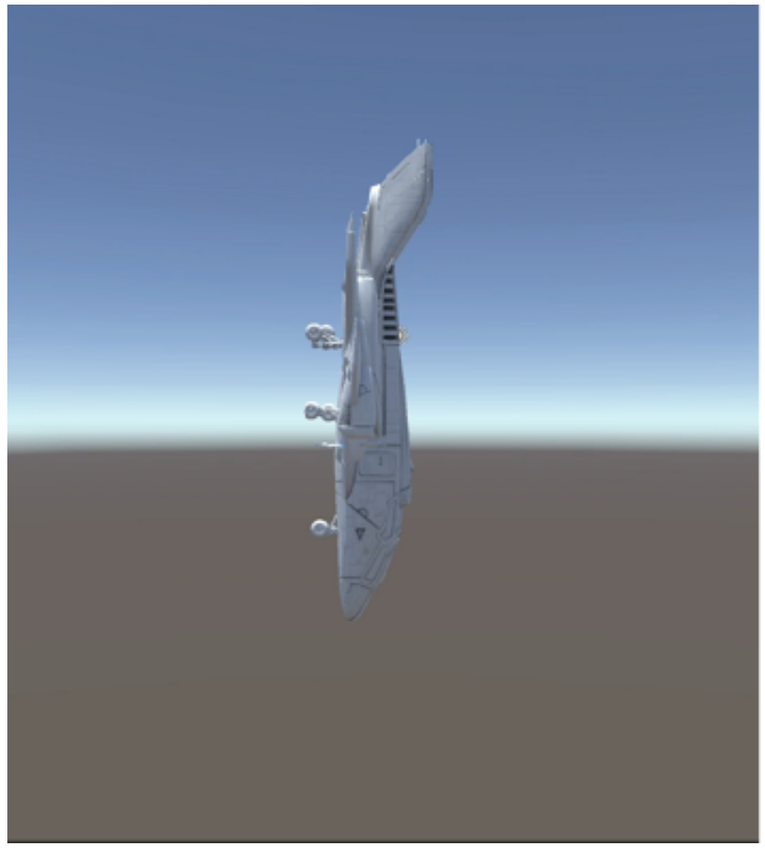
Start and end orientations



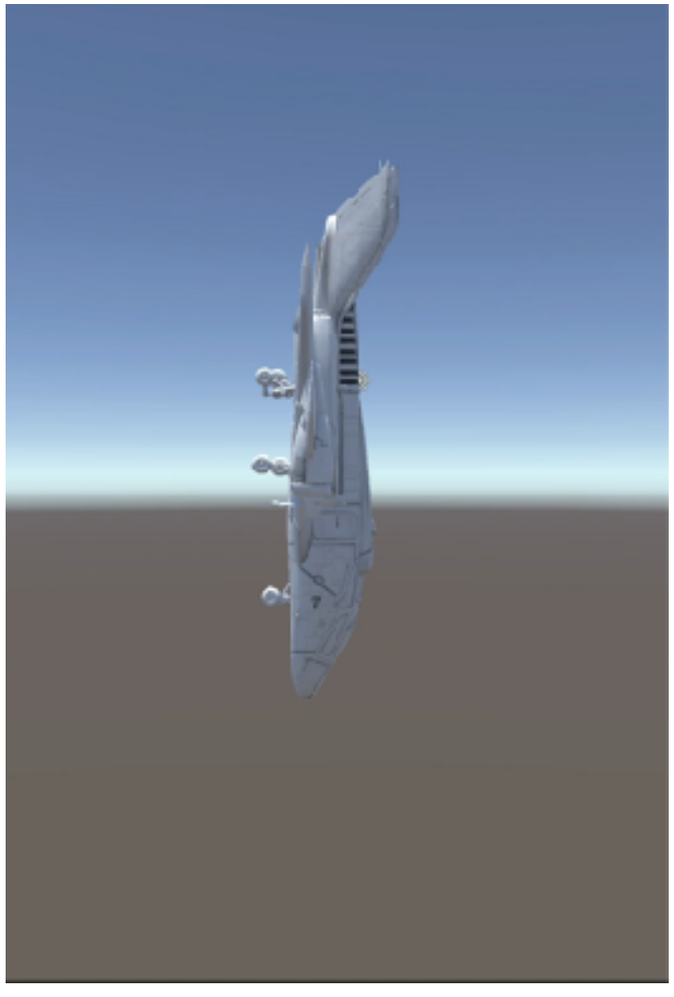
Result of representing the orientations with quaternions and using slerp for interpolation

SLERP (Spherical Linear Interpolation)

- Do you notice the difference?



Result of representing the orientations with euler angles and using linear interpolation



Result of representing the orientations with quaternions and using slerp

SLERP (Spherical Linear Interpolation)

- For more on quaternions, gimbal lock and slerp
  - **Section 2.2.4 and 3.3 of Computer Animation book by Rick Parent**
  - **https://www.youtube.com/watch?v=5YJwszR246I&t=1389s**
  - **https://www.youtube.com/watch?v=Mm8tzzfy1Uw**