Introduction to Computer Animation

Animation in production

Animation in Computer Graphics

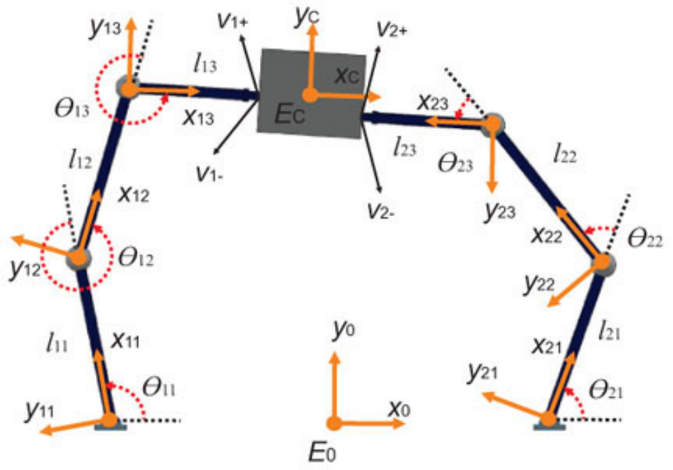
Two main ways to describe animation

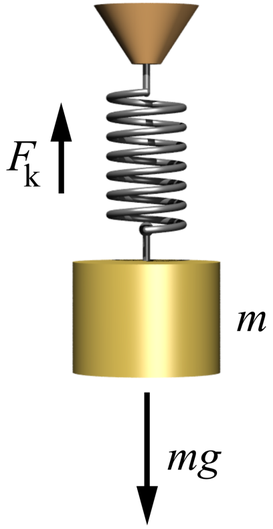
1- Kinematics

Shape deformation/motion without knowledge of the physical cause of the motion

2- Dynamics

Motion through forces acting on shapes (ex. physically based simulation)





Animation in Computer Graphics

Four ways to generate animation

1- Descriptive animation

2- Acquired motion

3- Procedural generation

4- Physically based simulation

Descriptive Animation

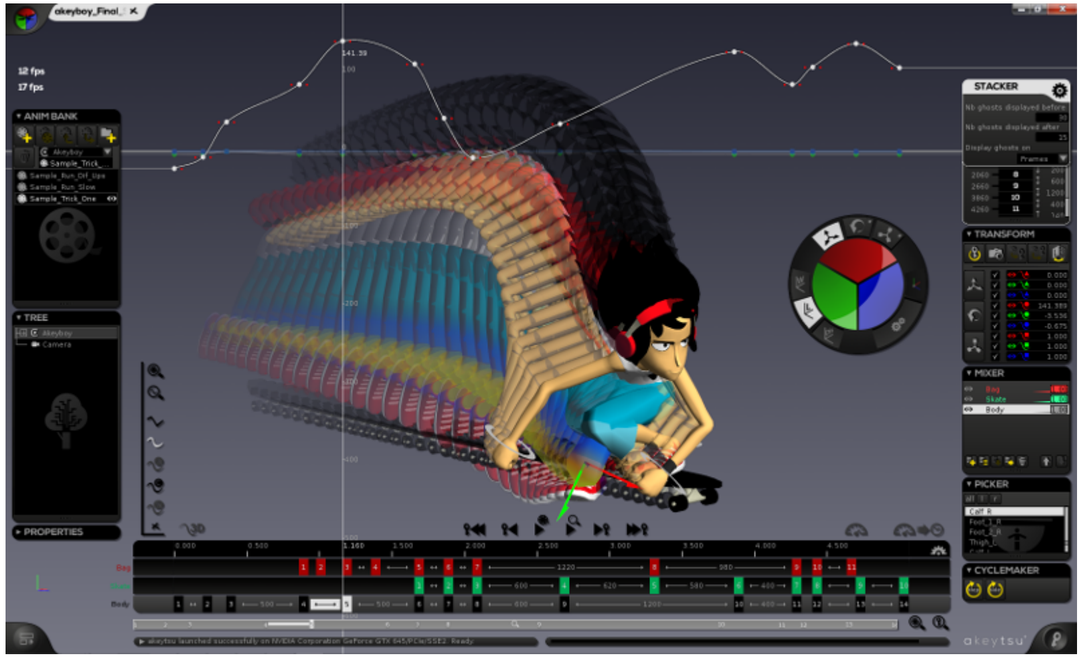
The artist/an algorithm fully describes the motion and deformation.

pro

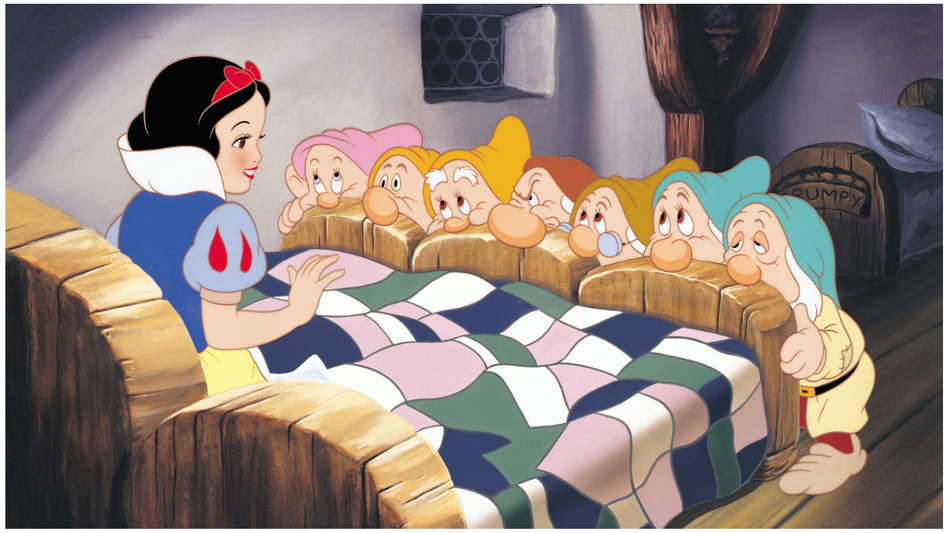
+ Full control on the result

cons

- May introduce un-physical artifacts



History of CG Animation



First production of animated films (Disney)

- 30 drawings / sec

Principle

- *Animator in chief*: Create **Key Frames**
- *Assistants*: Fill the *in between* (secondary drawings)

=> Principle of **Key Framing**

(c) Walt Disney Company, from "The Illusion of Life"

Key Framing in CG

- Create *manually* a set of **Key Frames** at specific time steps
- Compute automatically intermediate frames (30 fps) using **interpolation**

Animator Terminology

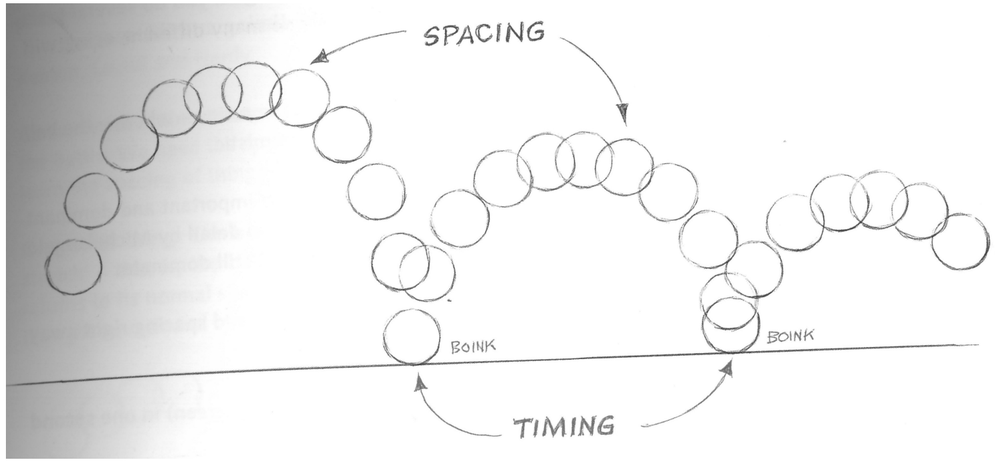
1- **Posing** the *key frames*

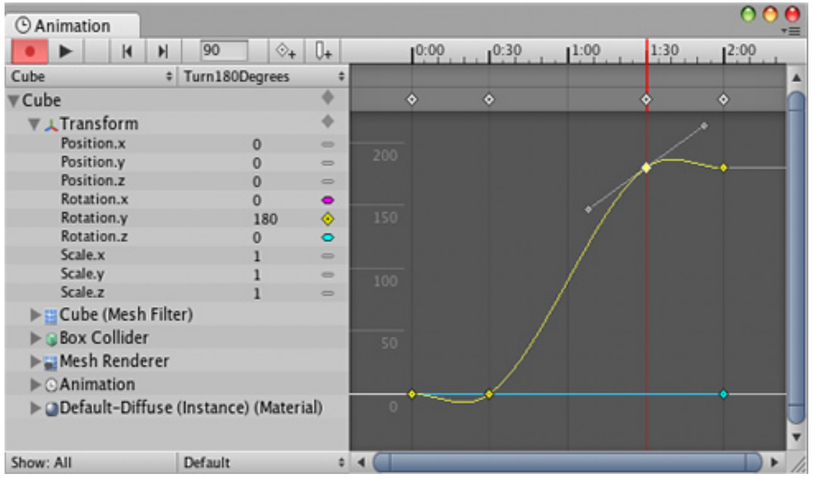
Set the main general posture of the character

*Linked to geometric **character deformation**, time is not involved*

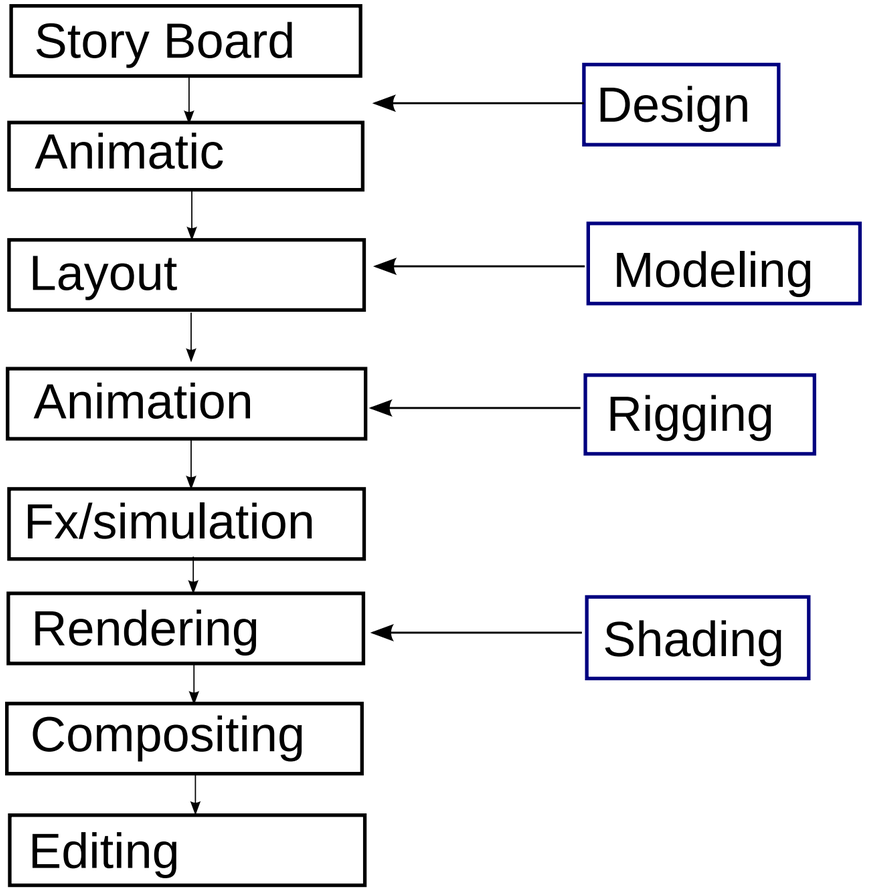
2- **Animating** the in-betweens

- **Timing :** Place key frames at specific times (global length of an action)
- **Spacing :** Speed of the interpolation (dynamic of the action)





Animation in production



- Animators: about 3/4 of production artists team.

- One professional animator can create 1s-10s animation per day.

"MONSTERS UNIVERSITY" Progression Image 1 of 6: STORY
©2013 Disney•Pixar. All Rights Reserved.

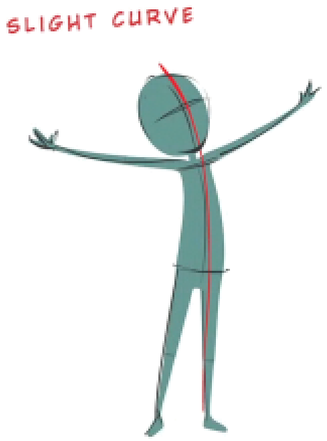
"MONSTERS UNIVERSITY" Progression Image 4 of 6: LAYOUT
©2013 Disney•Pixar. All Rights Reserved.

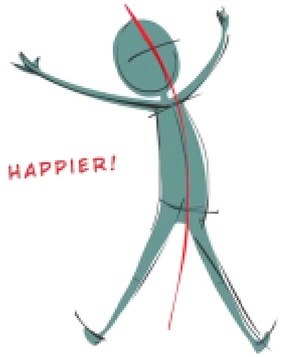
Animation in production - example

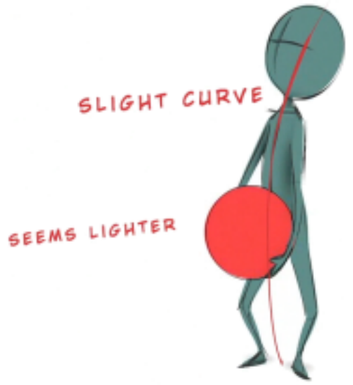
- In production *Animation* refers only to rigged character animation
- All other effects (dynamic hair, cloth, explosions, etc) are called *special effects*.

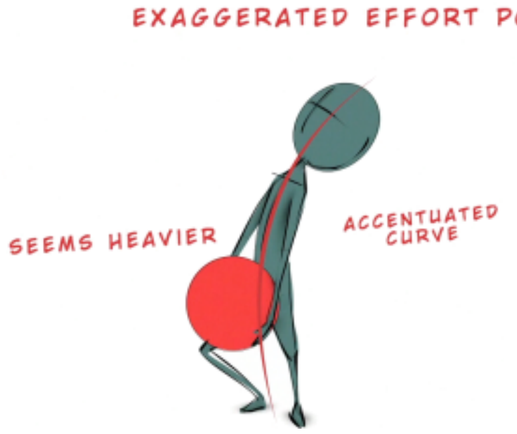
Character Animation Posing - Line of Action

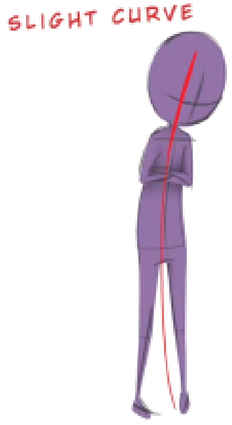
- Line of action: *Medial axis* expressing the character pose
- Express *statically* the dynamic of the action
  - Unstable pose ⇒ Dynamic action/motion

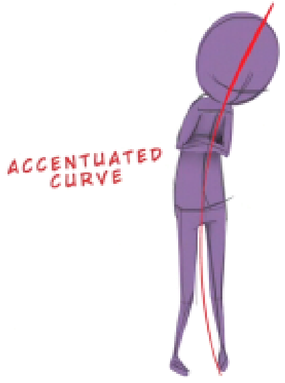












Expressive animation

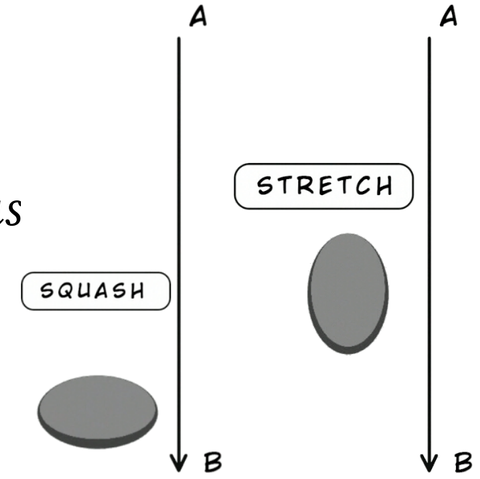
- Interpolation between realistic poses isn't enough for expressive animation
- *12 principles of animation* by Disney *Illustion of Life, 1981*

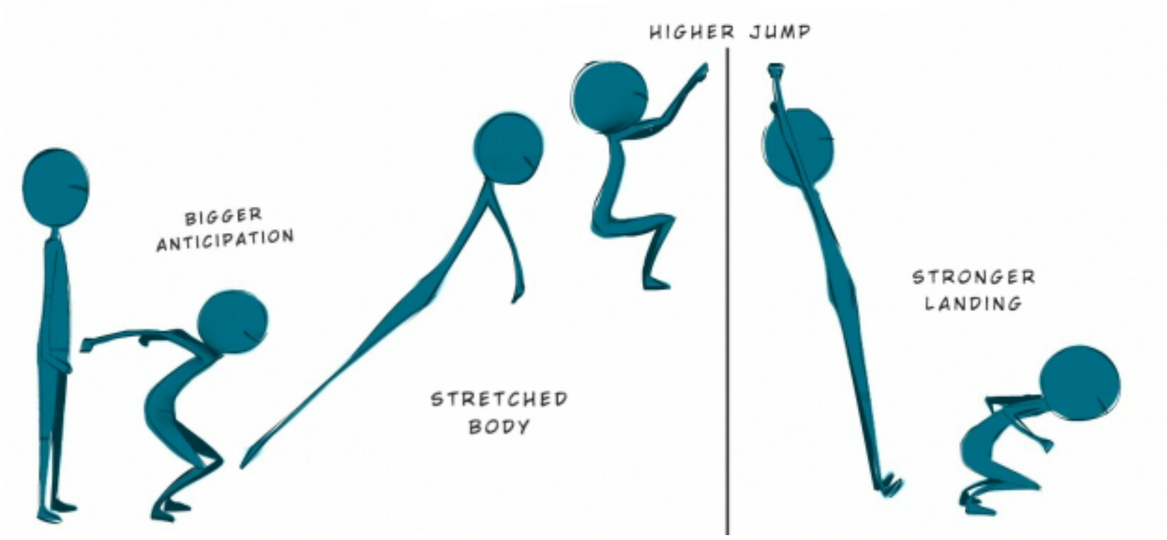
1. Timing
2. Spacing
3. Slow-in, Slow-out
4. Squash & Stretch
5. Anticipation
6. Follow Through
7. Secondary Action
8. Exaggeration
9. Appeal
10. Arcs
11. Staging
12. Straigh Ahead/Pose to Pose

Expressive animation

Squash & Stretch

- Very common in cartoon
- Unrealistic, but surprizingly *plaus*

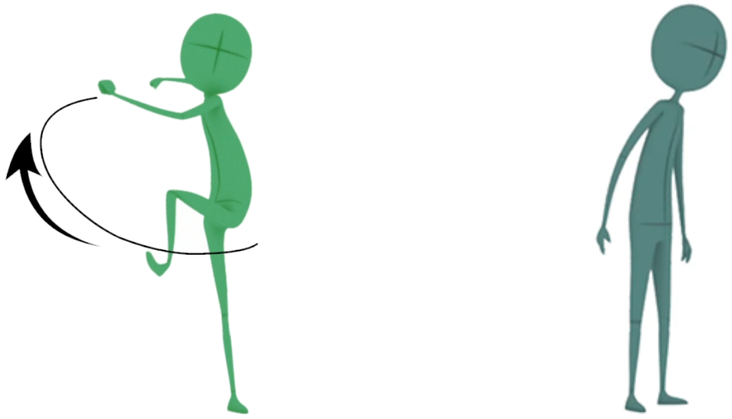


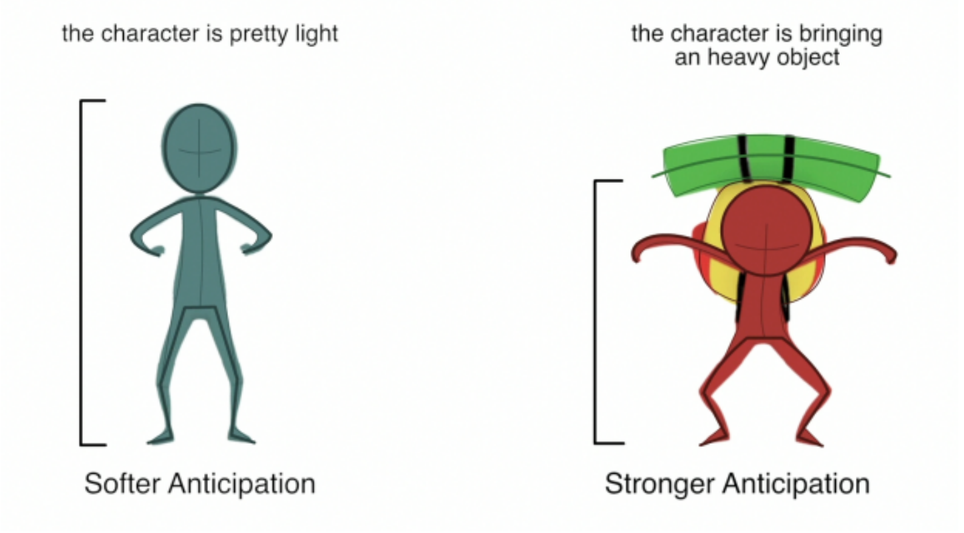


(C) Chiara Porri

Expressive animation

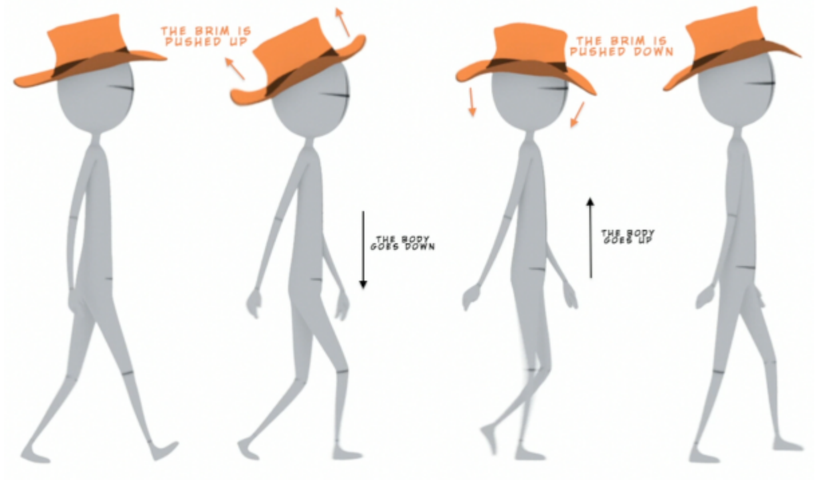
Anticipation

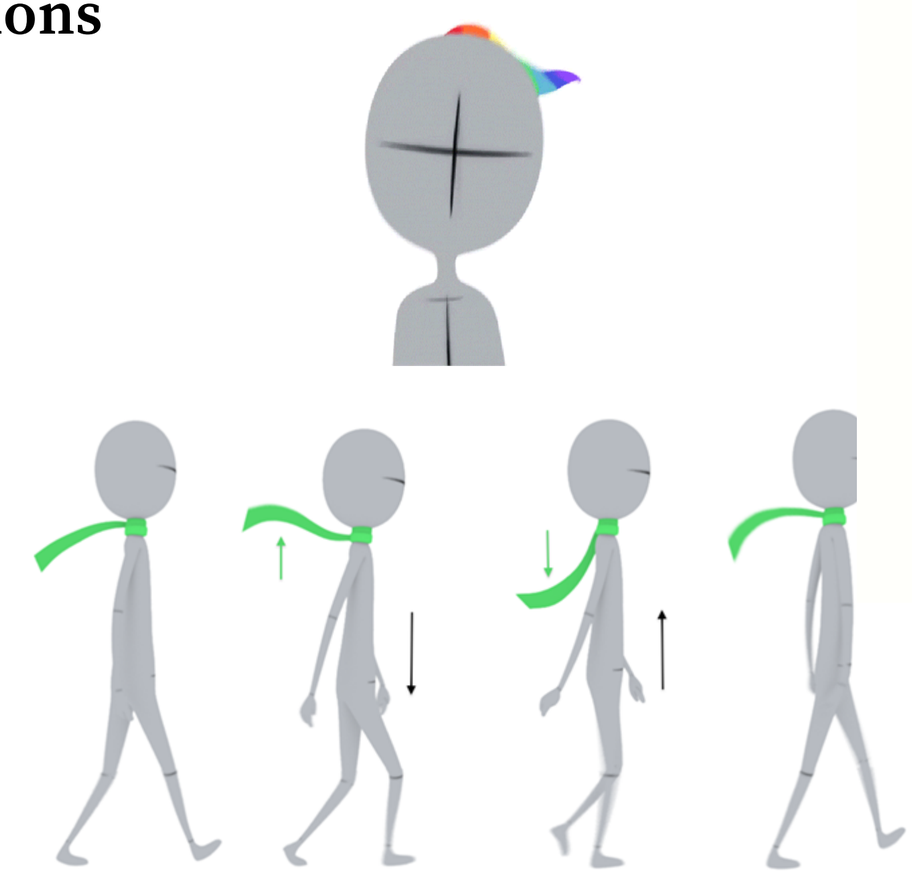


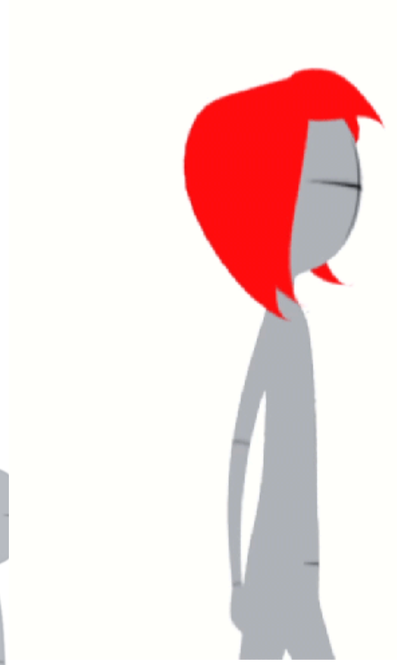


Expressive animation

Follow Through / Secondary motions







(C) Chiara Porri

Particle system

Particle system

Particle : element at a given position + extra parameters (mass, life time, etc)

Called **particles systems** in opposition to

- Rigid bodies - Solid objects with static shape
- Deformable bodies - Continuum material that can deforms

Pro

(+) Lightweight representation (both CPU+memory)

(+) Generic flexible model (spatial deformation, no connectivity, etc)

$\Rightarrow$ highly used in CG

Cons

(-) Simple model from physics point of view

Particles systems History

One of the first animated model in CG

Star Trek II

[Particle systems - A Technique for Modeling a Class of Fuzzy Objects, William T. Reeves, Lucasfilm, 1983] [pdf]

Karl Sims, Particle dreams, 1988 [link]

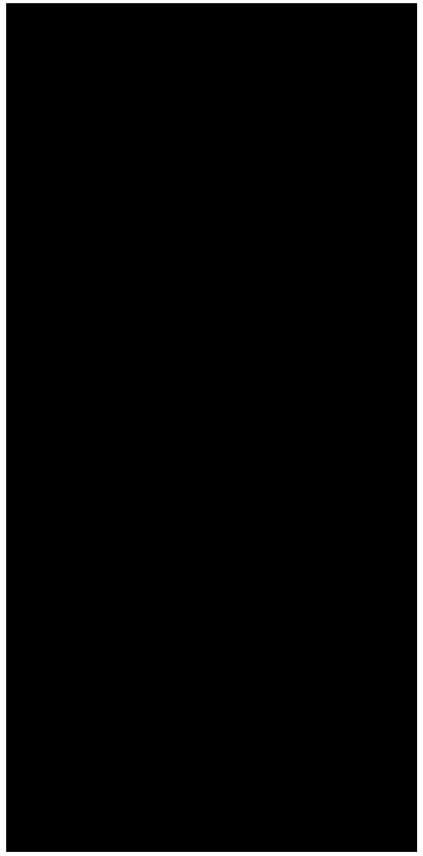
Example of Particle system

Free fall of spheres under gravity

- Geometrical representation of each particle: sphere
- Equation of motion $p(t) = \frac{1}{2}gt^2 + v_0 t + p_0$

- Initial position and speed may be placed at random positior
- Each particle may have a different life time

- *What are the parameters used for p_0 and v_0 in this example ?*



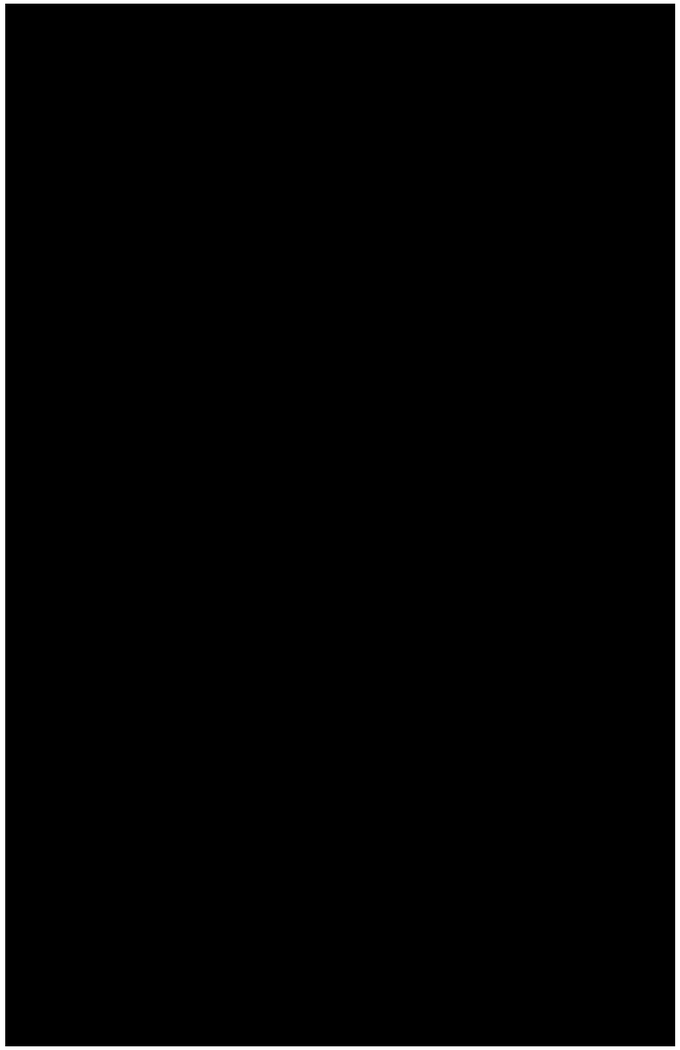
Bouncing spheres

- What is the equation of motion with boun

  - *Consider a particle emited at time* $t = 0$
  - *At what time* t_i*, the particle touch the floor ?*
  - *What is the new speed after impact ?*

General motions

Motion equation is not restricted to physics-based equations



- *What are the corresponding equations of motion ?*

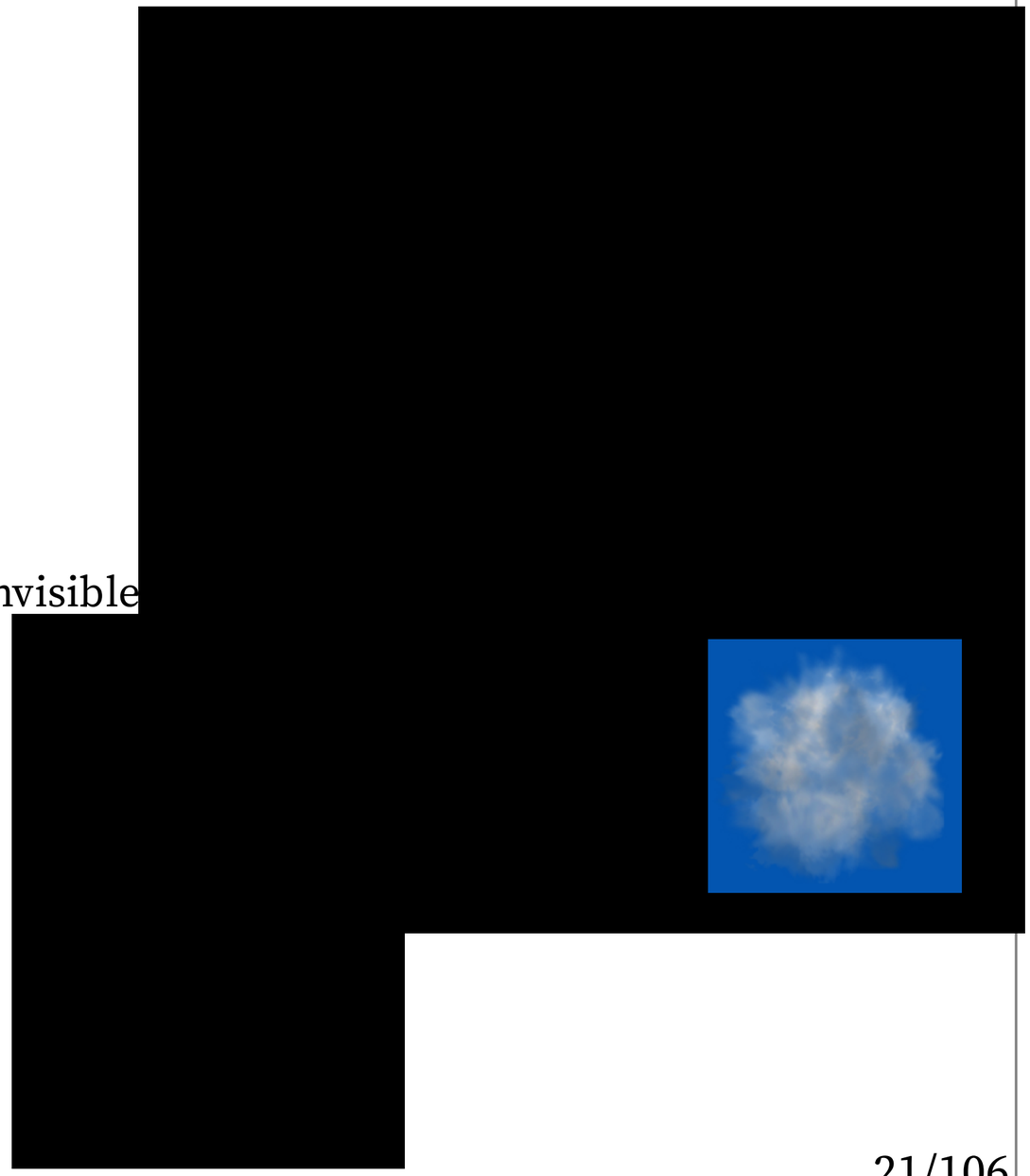
Billboards, Impostors, Sprites

Particles can be displayed as small images/thumbnails

In practice

- Each particle is displayed as a quadrangle
- A texture is mapped on the quad

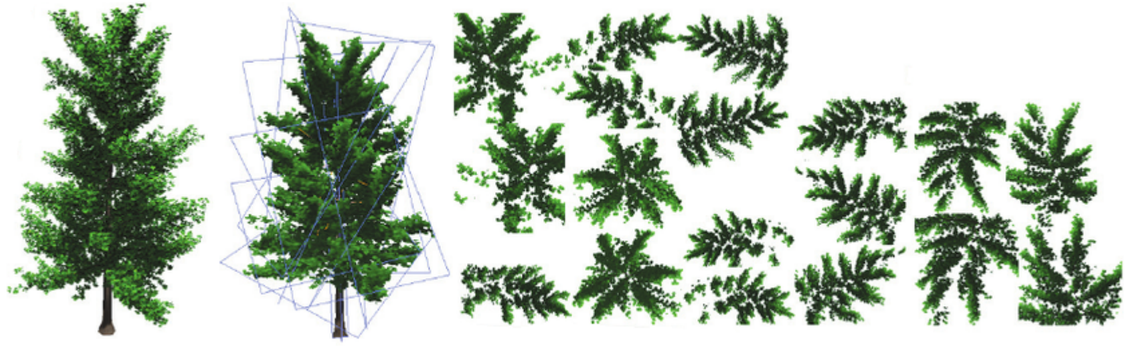
- The texture can contains transparency : quad geometry is invisible
- The quad can be facing the camera at all time (billboard)
- The texture can be animated (sprite)
- Texture can be adapted to the point of view (impostors)

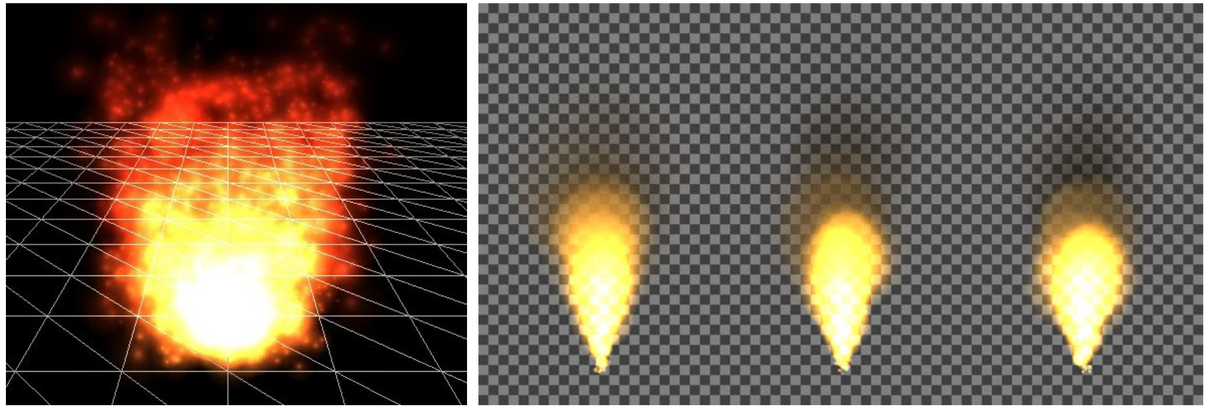


Usage of billboards

Large use of billboards for complex models

Vegetation, fire, smoke, etc.





Use case in production

How to model the *horse heads made of water* ?

The Lord of the Rings

Use case in production

How to model the *horse heads made of water* ?

The Lord of the Rings

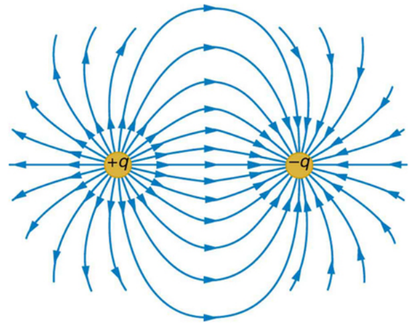
Use case in production

Full Making-Of

The Lord of the Rings

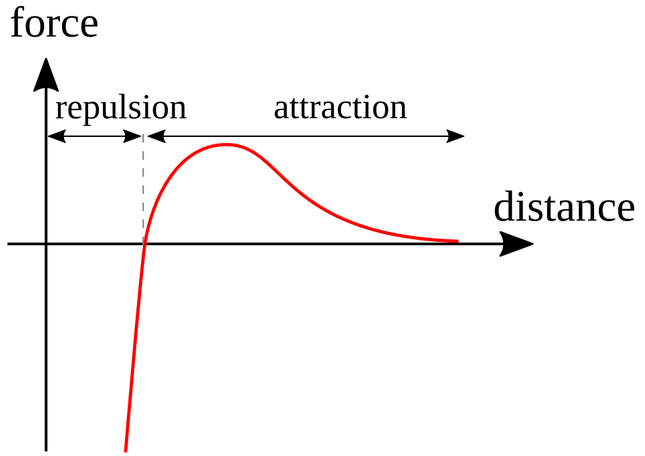
Interaction between particles

Interaction as **force field** (interact at distance)

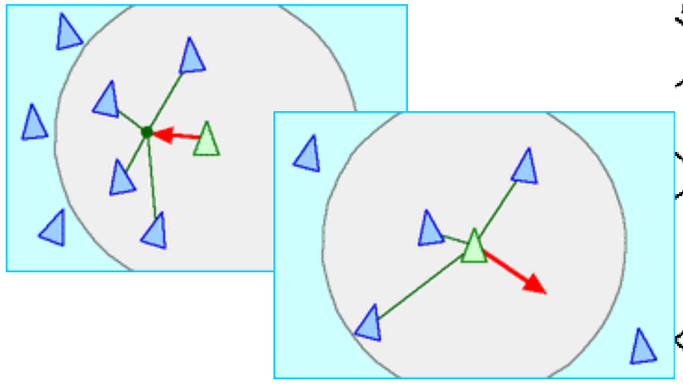


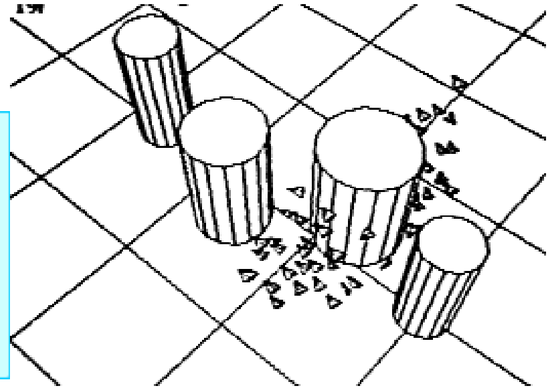
Example of usage

- Models crowd of life-like characters at large scale
  Inspired from physics particles forces (ex. Lennard-Jones potential)
  Attraction at *long-range*
  Repulsion at *short-range*



- First model: **Boids** Craig Reynolds 1987
- Extended later to human crowd modeling





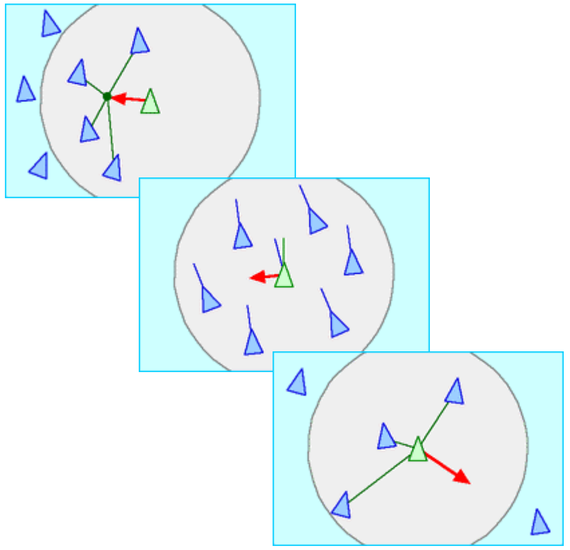
Boids Model

Introduced by
- [*Craig Reynolds. Flocks, Herds, and Schools: A Distributed Behavioral Model, SIGGRAPH 1987*] [link]
- [*Craig Reynolds. Steering Behaviors For Autonomous Characters. Proceedings of Game Developers, 1999*] [link]

A boid is defined by its
- Position
- Speed
- Forces acting on it

Three basic local *steering behaviors* to model
- **Cohesion** between local particles
- **Alignment** between local particles
- **Separation** between too close particles

=> Leads to emerging global behaviors.



Original video in 1986 (C. Reynolds)

Boids Model - Basic model.

- Set random initial position/speed to N particles.

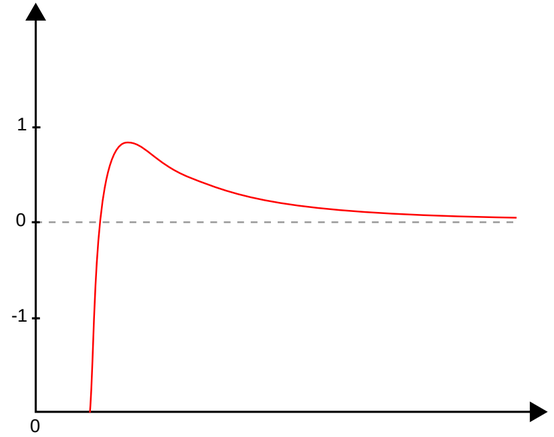
- Set attraction/repulsion force depending on pairwise distances

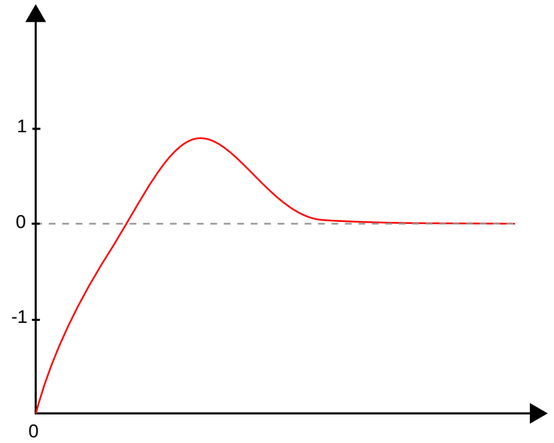
$$F(p_i) = \sum_j f(\|p_j - p_i\|) \frac{p_j - p_i}{\|p_j - p_i\|}$$

Example

- Inverse of distance $f(x) = \frac{\alpha_1}{x^2} - \frac{\alpha_2}{x^4}$

- Exponential/Gaussian $f(x) = \alpha_1 \exp\left(-k\left(\frac{x - x_0}{x_0}\right)^2\right) - \alpha_2 \exp\left(\frac{x}{x_0}\right)$





- Integrate position and speed through time

$$v^{t+\Delta t}(p_i) = v^t(p_i) + \Delta t \; F(p_i^t)$$
$$p^{t+\Delta t}(p_i) = p^t(p_i) + \Delta t \; v^{t+\Delta t}(p_i)$$

Boids Model - Complexity.

Simple implementation:

```
struct particle { vec3 p, v, f; };
std::vector<particle> boids;
// Initialize N boids ...
// ...



// compute pairwise force
for(int i=0; i<N; ++i)
{
    for(int j=0; j<N; ++j)
    {
        if( i!=j )
        {
            const vec3& pi = boids[i].p;
            const vec3& pj = boids[j].p;
            boids[i].f = force( norm(pi,pj)) / (pi-pj)/norm(pi-pj);
        }
    }
}
// integration
for(int i=0; i<N; ++i)
{
    boids[i].v = boids[i].v + dt * boids[i].f;
    boids[i].p = boids[i].p + dt * boids[i].v;
}
```

- What is the complexity (wrt. N) of this algorithm ?
- Can you think of a way to be more efficient for large N ?

Boids Model - Usage and limitations

- Well adapted to flocks (birds, fishes - looking behavior)
- Display particles using 3D animated model

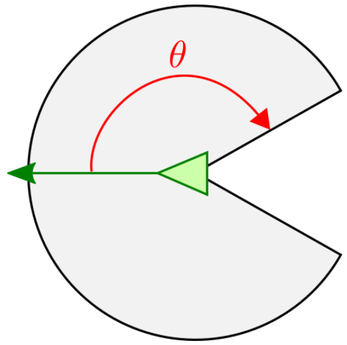
Additional behaviors

- Objective position/speed value
- Constraints: Obstacle avoidance, limited velocity
- Pursue and evade target/other particles - follow the leader, predators, etc.

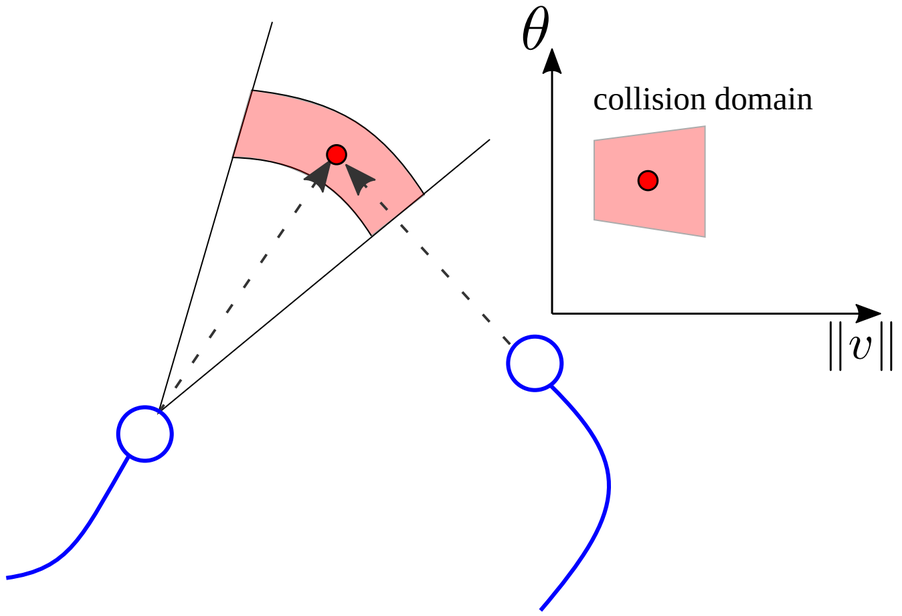
- *Is collision between particles possible ?*
- *Human displacement are mostly guided by vision, what key element is missing in the basic boids force-based model ?*

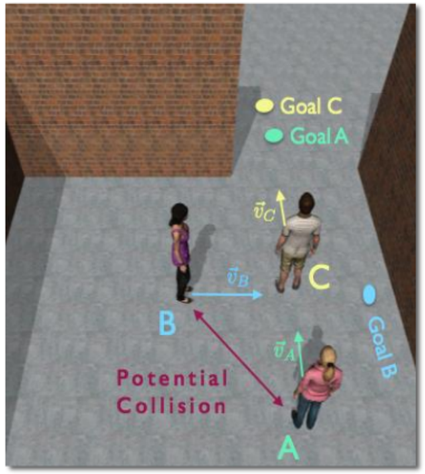
Boids Model - Extensions

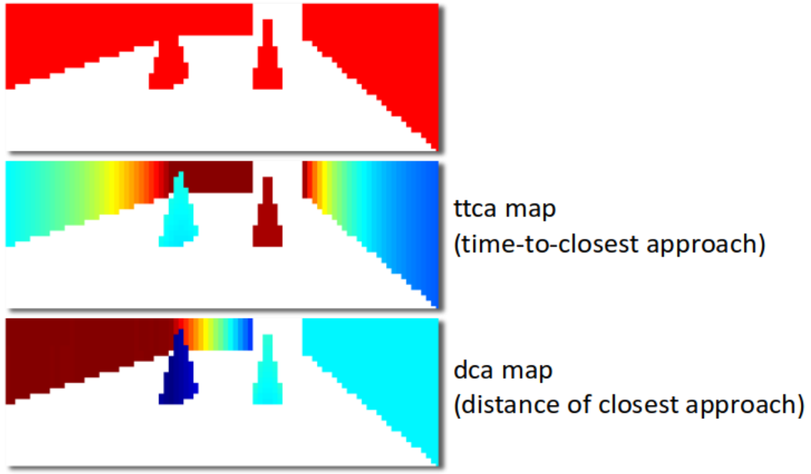
Improving force computation
- Vision-based force computation (angle, speed) : limited view angle



Improving collision avoidance
- Velocity-based
- Vision-based





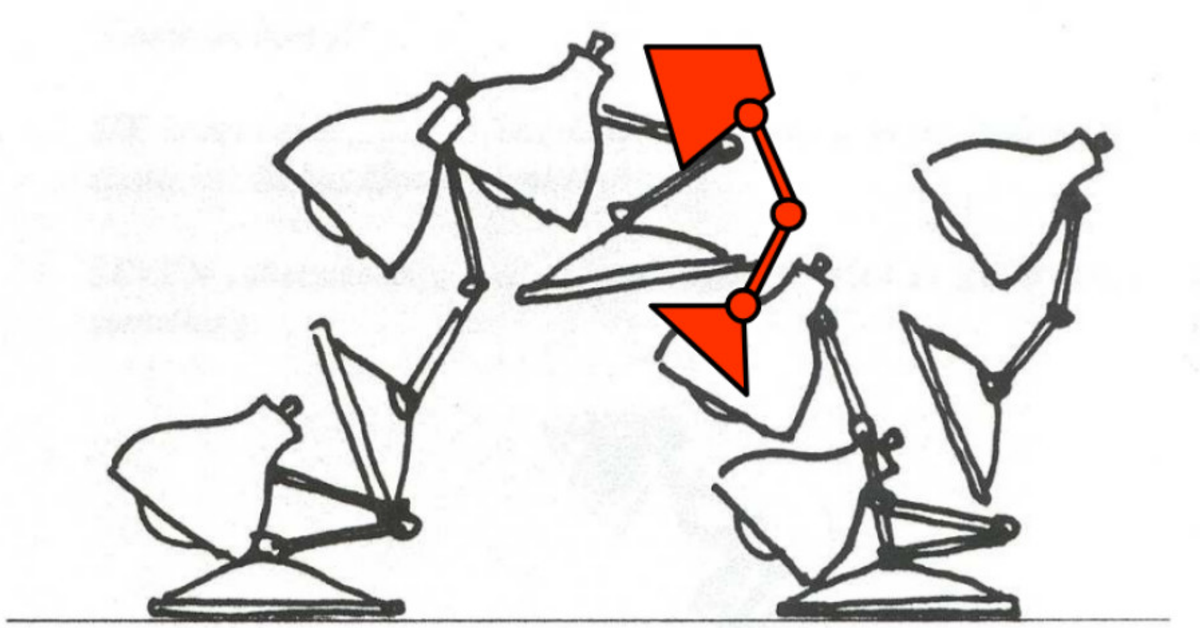


Julien Pettré, Inria Rennes

[J. Ondrej et al. A synthetic-vision based steering approach for crowd simulation, SIGGRAPH 2010]

Keyframe interpolation

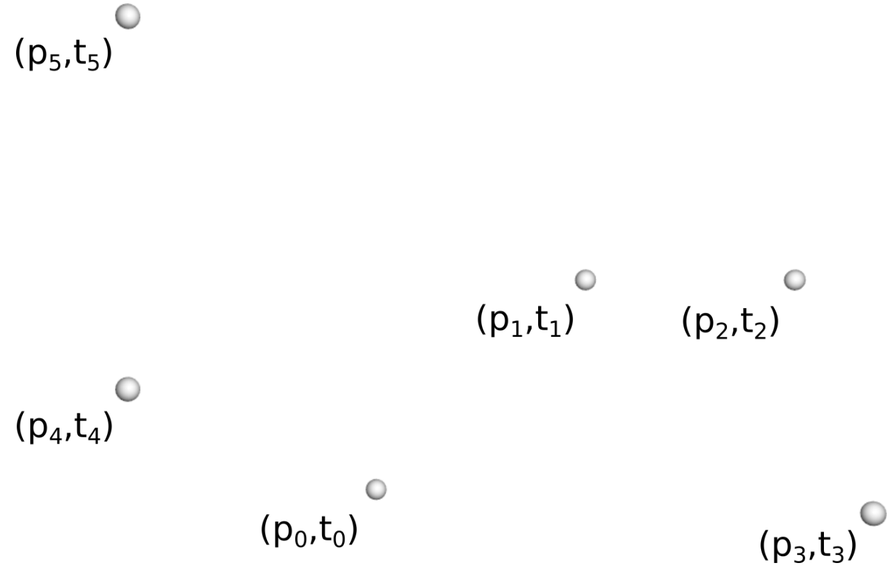
- **Interpolate positions**
- Interpolate rotations



Interpolate positions

Objective

- Given a set of key-positions (positions+time) we want to find an interpolating space-time curve



- **Input** $(p_i, t_i), i \in [0, N-1]$
- **Output**
  - Space time curve $p(t), t \in [t_0, t_{N-1}]$
  - Space time curve $p(t_i) = p_i$

Linear Interpolation

- Simplest solution: linear interpolation between each sample pairs

- $\forall t \in [t_i, t_{i+1}]$, $\begin{cases} p(t) = (1 - \alpha(t))\, p_i + \alpha(t)\, p_{i+1} \\ \alpha(t) = \dfrac{t - t_i}{t_{i+1} - t_i} \end{cases}$

Pros

- Simple
- Constant speed between keyframes

  Easy to ajust

Cons

- Non smooth trajectory
- Generates straight segments

  Artists prefer non straight trajectories for "real" looking motion

Smooth curve

Objective: Generate a **smooth** interpolating space-time curve

Classical Idea Use polynomials curves

$$\begin{cases} p(t) = \sum_{i=0}^{N-1} \alpha_i(t)\, p_i \\ \alpha_i(t) = \sum_{j=0}^{d} c_i^j\, t^i \end{cases}$$

(α_i) polynomial basis function of degree d

Which polynomials/degree choose ?

Lagrange polynomial interpolation

Naive idea: Interpolate all points at once

$$\forall t \in [t_0, t_{N-1}]\ ,\ \ p(t) = \sum_{i=0}^{N-1} \alpha_i(t)\ p_i \qquad \forall i \in [0, N-1]\ \ p(t_i) = p_i$$

Degree of polynomial : $N-1$

- Known solution: **Lagrange polynomial**

$$p(t) = \sum_{i=0}^{N-1} \alpha_i(t)\ p_i \qquad \alpha_i(t) = \prod_{k=0, k\neq i}^{N-1} \frac{t - t_k}{t_i - t_k}$$

- Explanation: By construction $\alpha_i(t_i) = 1$ and $\alpha_i(t_k) = 0$

(+) Interpolate all points

(-) Large oscillations between samples for large degree.

(-) Non local influence

=> Not used in practice

Lagrange polynomial interpolation : Comparison

Ex. Global effect on curve when two samples are added.

10 samples

12 samples

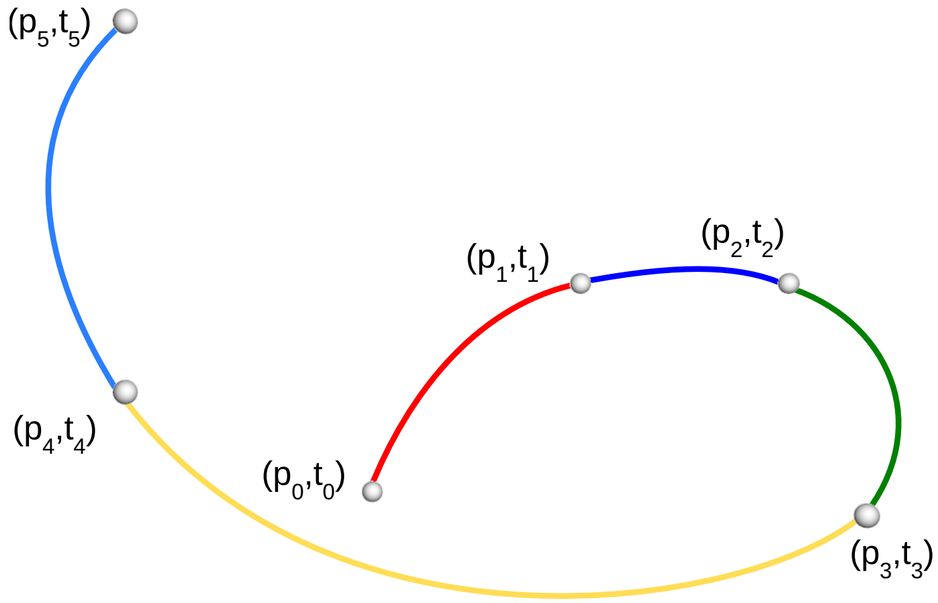
Spline

Idea

- Define on each part a polynomial
- Smooth junctions between them.

How to choose the polynomial

- Sufficiently high degree to be smooth
- Sufficiently low degree to avoid oscillations



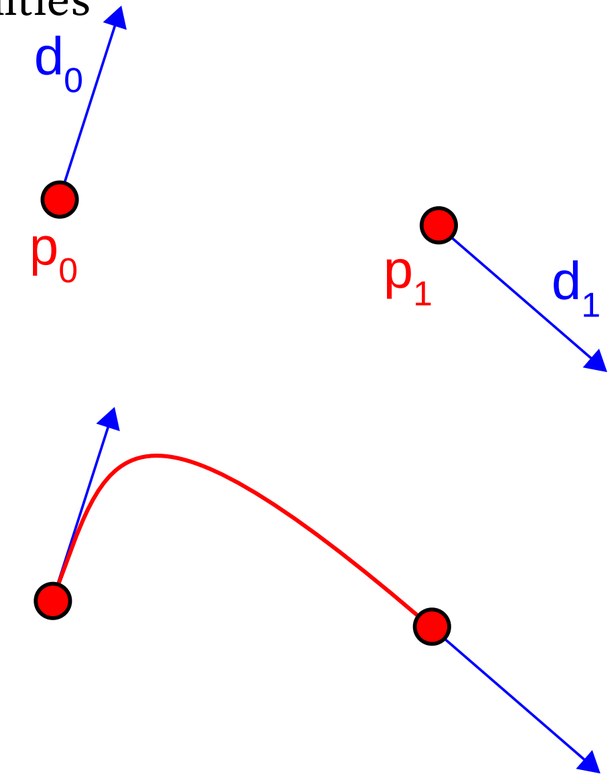
Each color is another polynomial

=> In Graphics cubic polynomials are often used

Allows up to C^2 junctions

Hermite interpolation

Hermite interpolation : cubic curve interpolating points and derivatives at extremities



Consider the following constraints

$$\begin{cases} p(s) = c_3 s^3 + c_2 s^2 + c_1 s + c_0 \\ p(0) = p_0,\ p(1) = p_1,\ p'(0) = d_0,\ p'(1) = d_1 \end{cases}$$

$\Rightarrow$ System of equations

$$\begin{cases} \,c_3 + \,c_2 + \,c_1 + \,c_0 = p_0 \\ \,c_3 + \,c_2 + \,c_1 + \,c_0 = p_1 \\ \,c_3 + \,c_2 + \,c_1 + \,c_0 = d_0 \\ 3\,c_3 + 2\,c_2 + \,c_1 + \,c_0 = d_1 \end{cases} \Rightarrow \begin{cases} c_0 = p_0 \\ c_1 = d_0 \\ c_2 = -3p_0 + 3p_1 - 2d_0 - d_1 \\ c_3 = 2p_0 - 2p_1 + d_0 + d_1 \end{cases}$$

$$\forall s \in [0,1]\ ,\quad p(s) = (2s^3 - 3s^2 + 1)\, p_0 + (s^3 - 2s^2 + s)\, d_0 + (-2s^3 + 3s^2)\, p_1 + (s^3 - s^2)\, d_1$$

For arbitrary $t \in [t_i, t_{i+1}]$, we set $s = \dfrac{t - t_i}{t_{i+1} - t_i}$, $\tilde{d}_{0/1} = (t_{i+1} - t_i)\, d_{0/1}$

Interpolating curve

Our initial problem: set of multiple keyframes position+time

Two solutions

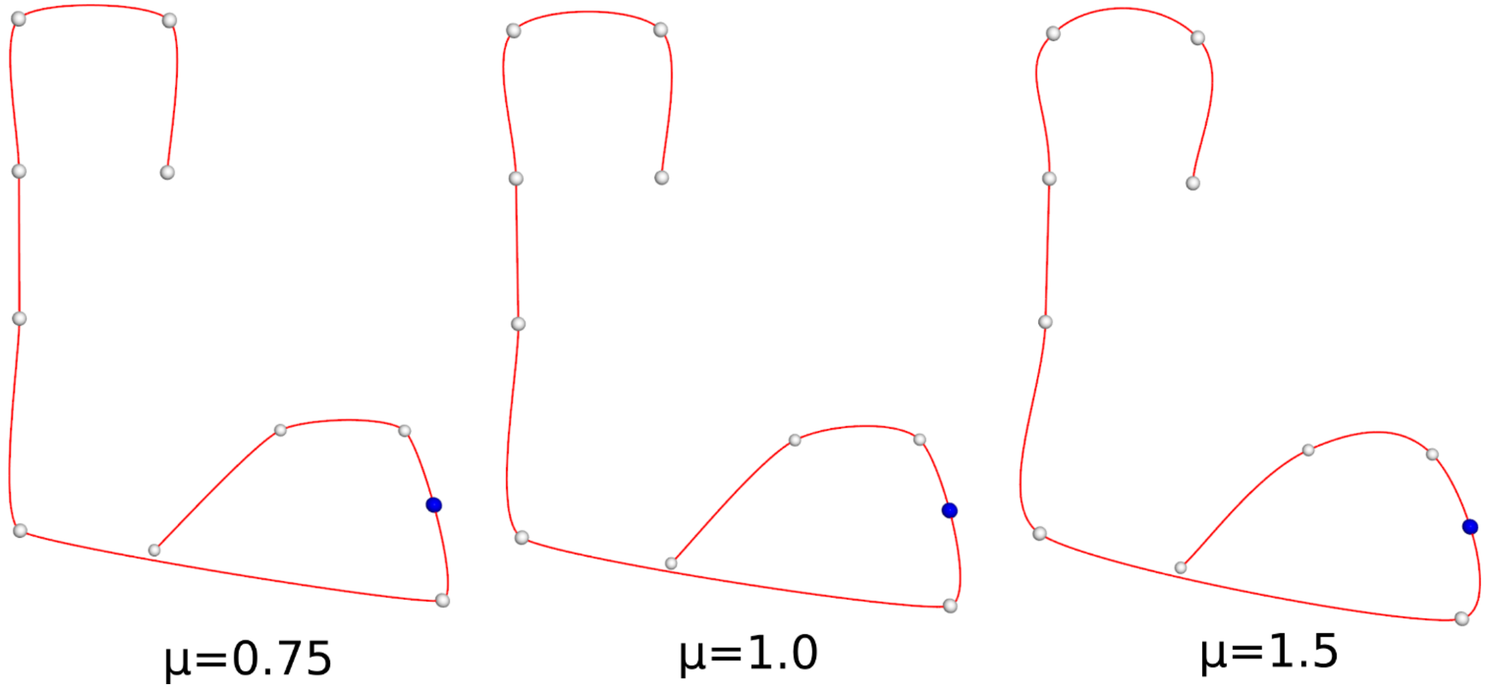
- Set explicitely derivatives for each keyframe - *teadious*
- Compute automatically plausible derivatives from surrounding samples - *often used*

Cardinal spline

Set $d_i = \mu \dfrac{p_{i+1} - p_{i-1}}{t_{i+1} - t_{i-1}}$

- μ curve tension $\in [0, 2]$
- $\mu = 1$ is commonly used

Catmull Rom Spline



Wrap-up Algorithm

Compute $p(t)$ as a cubic spline interpolation

- Given keyframes $(p_i, t_i)_{i \in [0, N-1]}$
- Given time $t \in [t_1, t_{N-2}]$

1. Find i such that $t \in [t_i, t_{i+1}]$
2. Compute $d_i = \mu \, \sigma \, \dfrac{p_{i+1} - p_{i-1}}{t_{i+1} - t_{i-1}}$, and $d_{i+1} = \mu \, \sigma \, \dfrac{p_{i+2} - p_i}{t_{i+2} - t_i}$

   $\sigma = t_{i+1} - t_i$
3. Compute $p(t) = (2s^3 - 3s^2 + 1)\, p_i + (s^3 - 2s^2 + s)\, d_i + (-2s^3 + 3s^2)\, p_{i+1} + (s^3 - s^2)\, d_{i+1}$

   with $s = \dfrac{t - t_i}{\sigma}$.

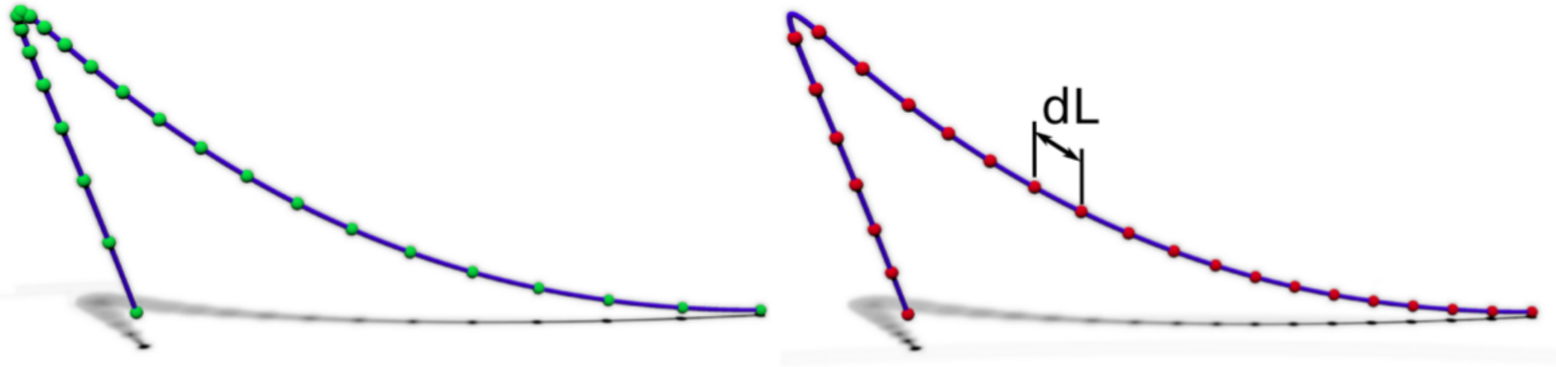
Limitation of cubic curve interpolation

- Only C^1, but not C^2 at junctions : Curvature/acceleration discontinuity

  - *Force C^2 continuity for cubic polynomial (global linear system to solve - loose local structure)*
  - *Consider higher degree polynomial*

- Non constant speed along each polynomial

  - *Reparametrization with curvilinear length*



Usage of keyframes interpolation

Interpolate every vertex of multiple meshes

Multi-target blending and blend shapes

The standard for facial animation

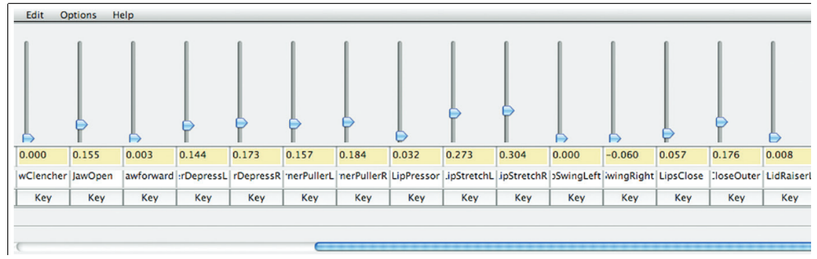
Full blend shape

- Per-vertex formulation $p^i(t) = \sum_k \omega_k(t)\, b_k^i$
- Matrix formulation $\mathbf{p}(t) = \mathrm{B}\,\omega(t)$

Differential representation

- Per-vertex formulation $p^i(t) = b_0^i + \sum_k \omega_k(t)\, (\underbrace{b_k^i - b_0^i}_{d_k^i})$
- Matrix formulation $\mathbf{p}(t) = \mathbf{b_0} + \mathrm{D}\,\omega(t)$

$\Rightarrow$ Allows expression transfert (keep same D for different $\mathbf{b_0}$).



Problems:

- D is not an orthogonal matrix (non unique combination, redundancies) $\Rightarrow$ decomposition PCA, etc.
- How to directly manipulate blend shapes

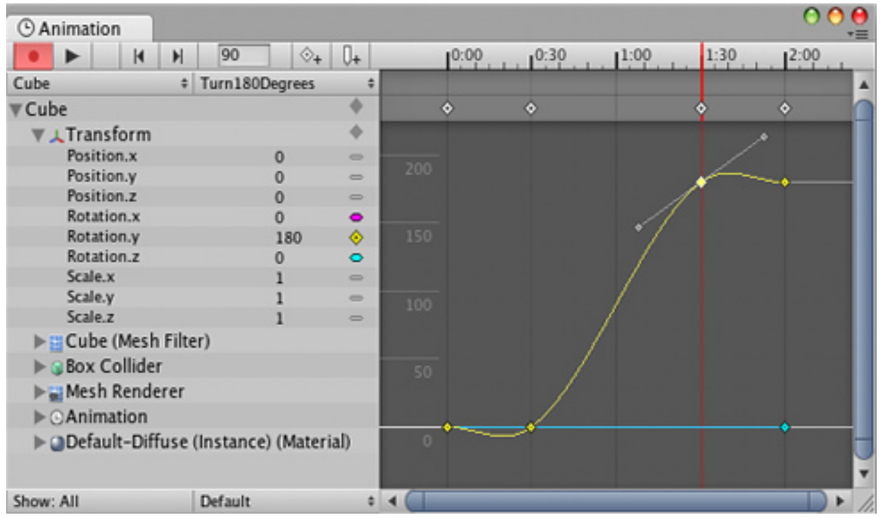
[J.P. Lewis et al. Practice and Theory of Blendshape Facial Models. EG STAR 2014]

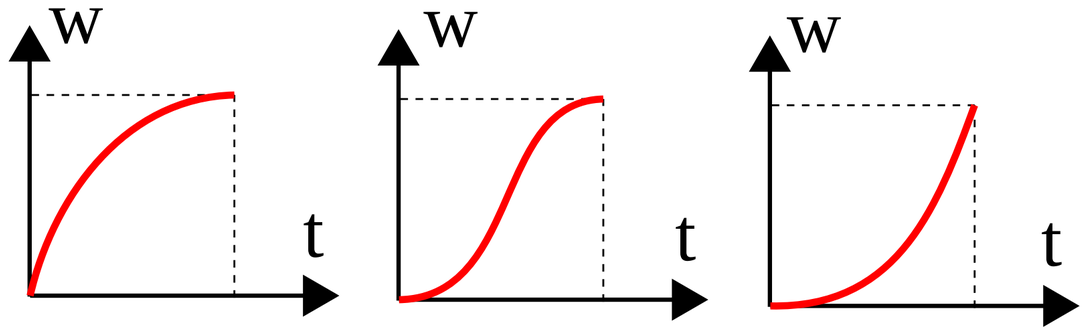
Curve editing

- Animation software (Maya, 3DSMax, Blender, etc) always come with a **curve editor**.

- Artists can manually ajust their position, time, and **derivatives** on curve editor.
  - One curve for each scalar parameter
    - position (x, y, z)
    - scaling (s_x, s_y, s_z)
    - rotation/quaternion (q_x, q_y, q_z, q_w)

- Can also use a wrapper function w to change time $p(t) = f(w(t))$



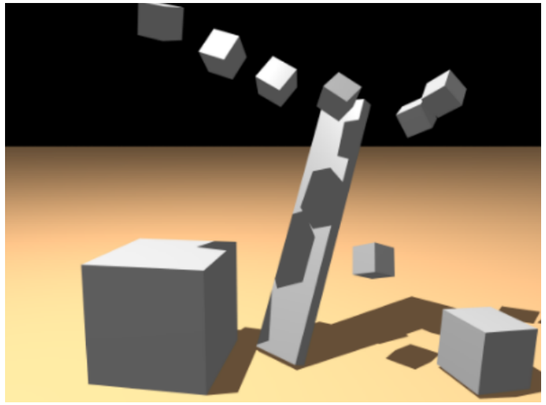


Interpolate rotations

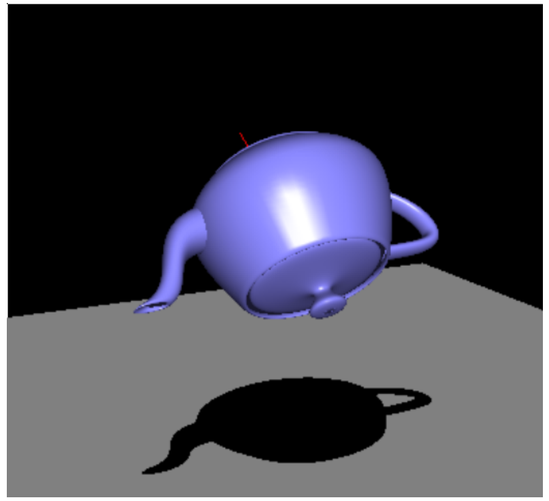
and rigid transformations

Rotation

When dealing with rigid objects (ex. Character skeleton, oriented rigid objects, etc) orientations (thus rotations) are involved.







- How to **encode and interpolate rotations** ?

Rotation in 2D: Easy

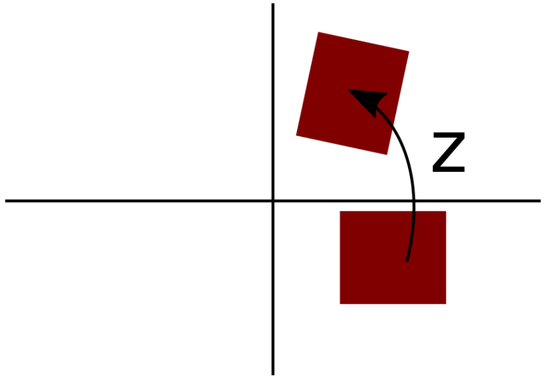
1 - Model rotation by an angle

2 - Interpolate the angle

Can be formalized using complex numbers representation

- $R_1 \rightarrow z_1 = e^{i\,\theta_1}$
- $R_2 \rightarrow z_2 = e^{i\,\theta_2}$

- $R^{\alpha}_{1\rightarrow 2} \rightarrow z = e^{i\,(\alpha\theta_1 + (1-\alpha)\theta_2)}$



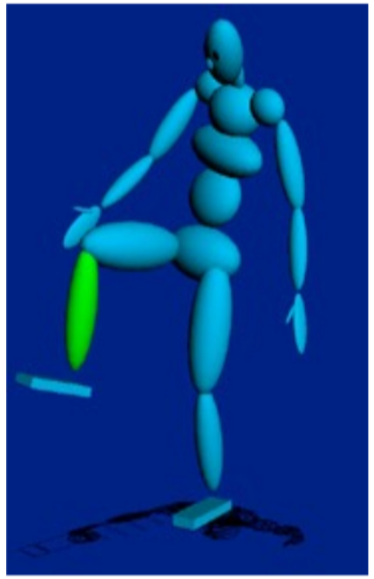
In 3D: not so easy ...

How many angles do I need to represent a 3D rotation?

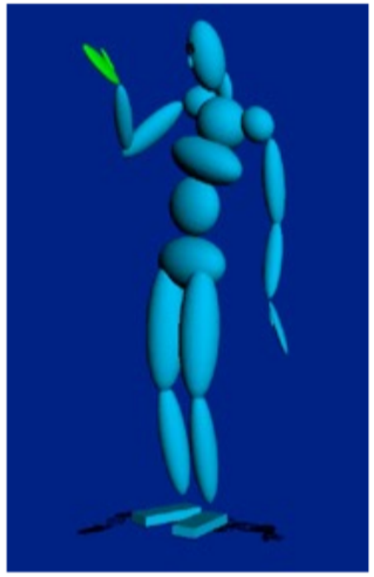
Rotation in 3D - DOF

- 3 Degrees of Freedom (dof)

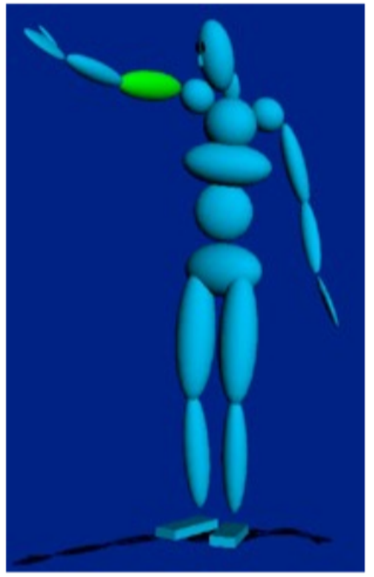
1 DOF: knee



2 DOF: wrist



3 DOF: arm



Representing 3D Rotations

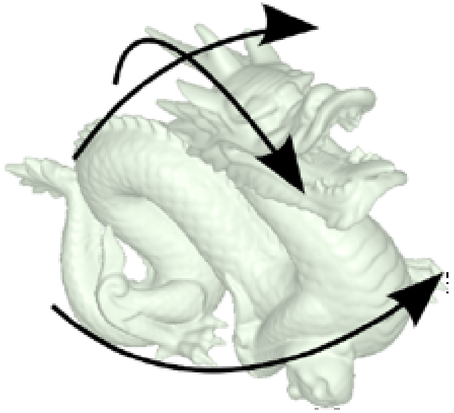
Multiple ways to represent rotations

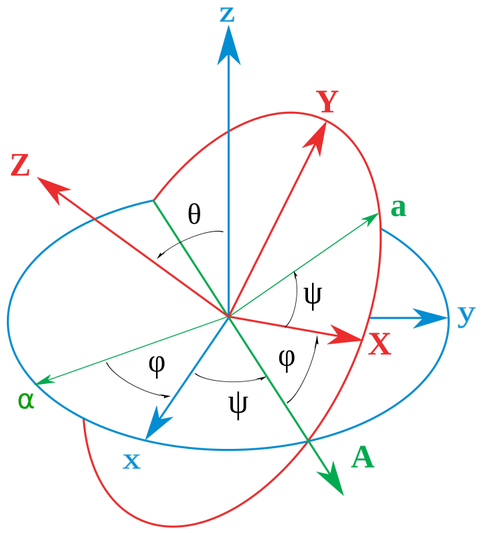
1 - Matrices

2 - Euler (/Trait-Bryan) angles

3 - Axis-Angle

4 - Quaternion





Rotation representation: Matrix

$$\begin{cases} \mathrm{R} = \begin{pmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{pmatrix} \\ \\ \mathrm{R}^T \, \mathrm{R} = \mathrm{Id} \end{cases}$$

Pro.

- Standard structure (storage,manipulate)
- Simple application to vector: $v' = \mathrm{R}v$
- Composition b/w rotations: $\mathrm{R} = \mathrm{R}_1 \, \mathrm{R}_2$

Cons.

- Large redundancy (*9 coefficients for 3 dof*).
- Rotation parameters don't appear explicitely
- Unclear interpolation
  - Linear interpolation doesn't work well for large angles

  ex. $\mathrm{M} = (1 - \alpha) \, \mathrm{R}_1 + \alpha \mathrm{R}_2$

  => M is not a rotation anymore (not a vectorial space)

Representing 3D Rotations

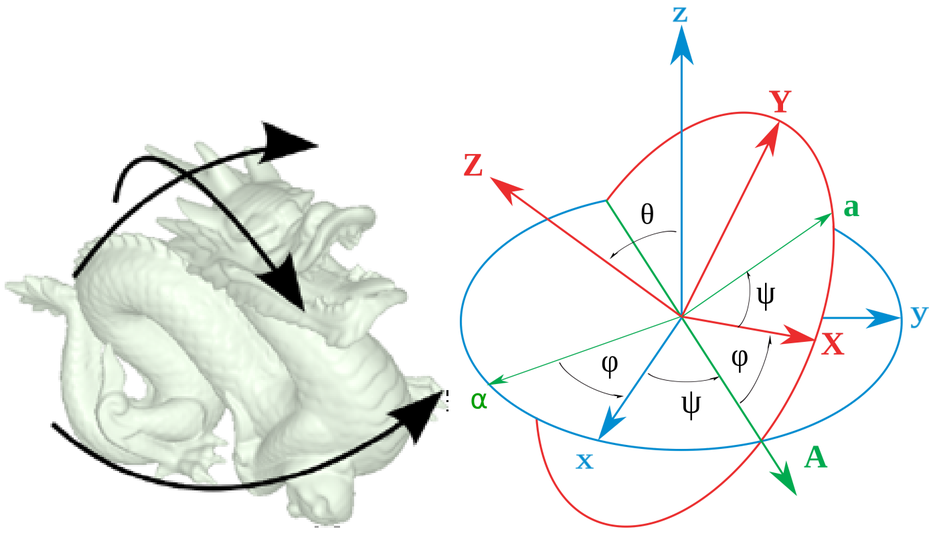
Multiple ways to represent rotations

1 - Matrices

2 - Euler (/Trait-Bryan) angles

3 - Axis-Angle

4 - Quaternion



Rotation representation: Euler angles

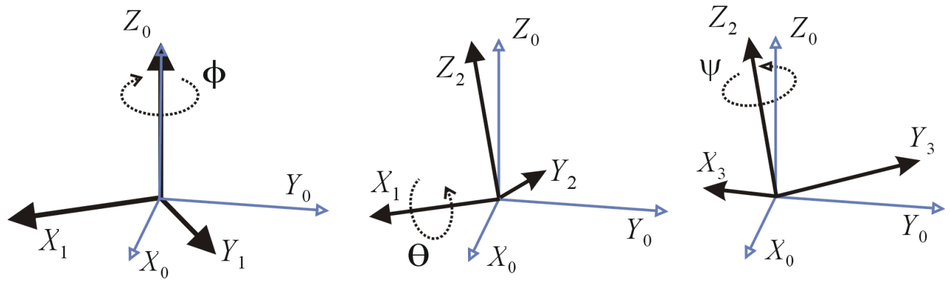
Three consecutive rotations along (x, y, z) coordinates.

Can use basic rotation matrices

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin\theta & \cos(\theta) \end{pmatrix} \quad R_y = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \quad R_z = \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

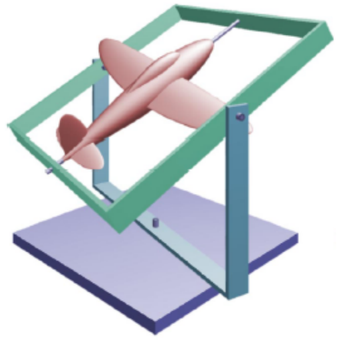
Multiple Euler angles conventions

- *Proper Euler : z-x-z', x-y-x', y-z-y', ...*
- *Trait-Bryan : x-y-z (global), ...*



Pro

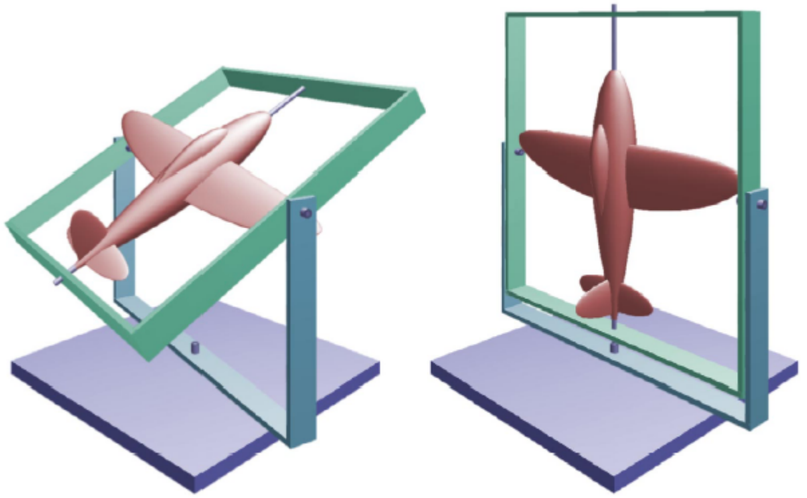
- Combination of rotation around known axis
- Comprehensive parameters (3 dof)
- Animators can interact with angular curves
- Easy conversion to matrix
- *Widely used in robotics*



Rotation representation: Euler angles

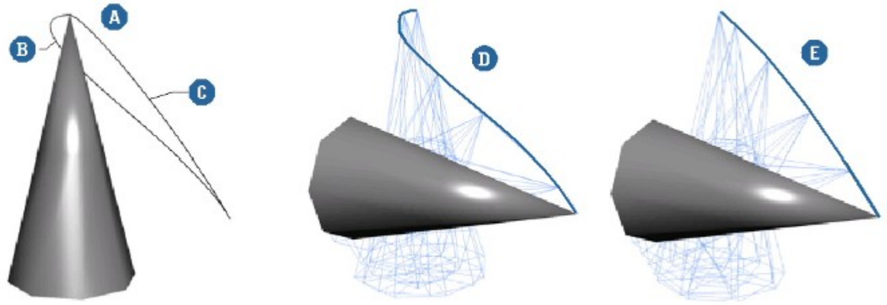
Limitations of Euler Angle

- *Gimbal Lock* when composing b/w some rotations



http://www.fho-emden.de/~hoffmann/gimbal09082002.pdf

- Interpolation of 3 angles leads to non-intuitive trajectory.



Representing 3D Rotations

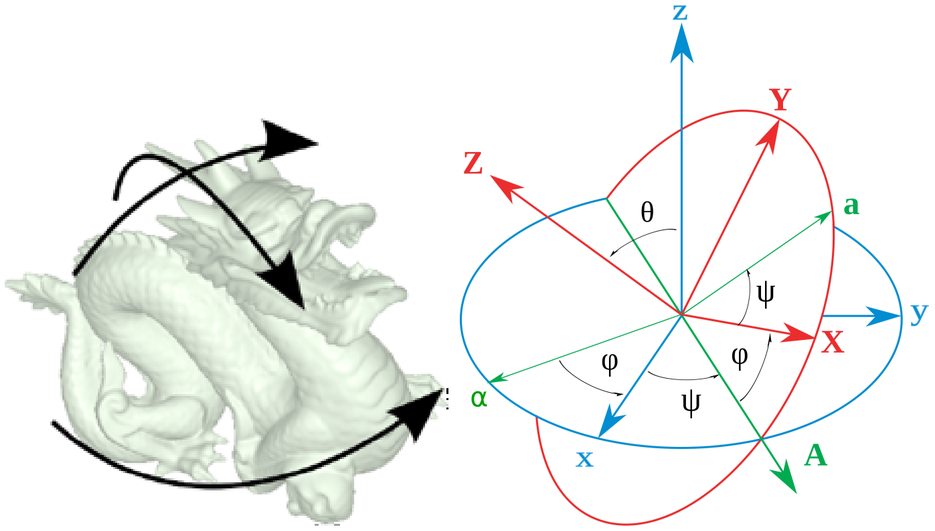
Multiple ways to represent rotations

1 - Matrices

2 - Euler (/Trait-Bryan) angles

3 - Axis-Angle

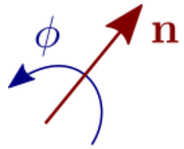
4 - Quaternion



Rotation representation: Axis-Angle

Any 3D rotation can be represented by

- A unit axis n
- An angle θ

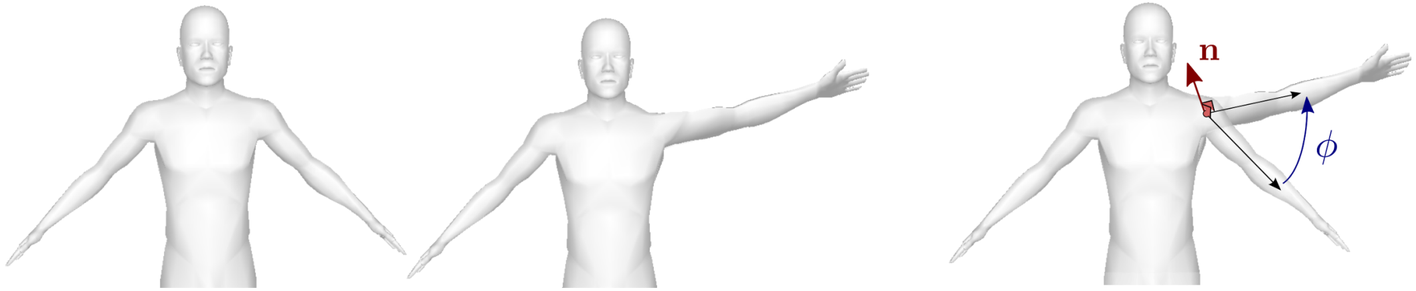


Pro.

- Concise representation: 3 DOF
- Meaningfull parameters
- Describes well the rotation between two vectors

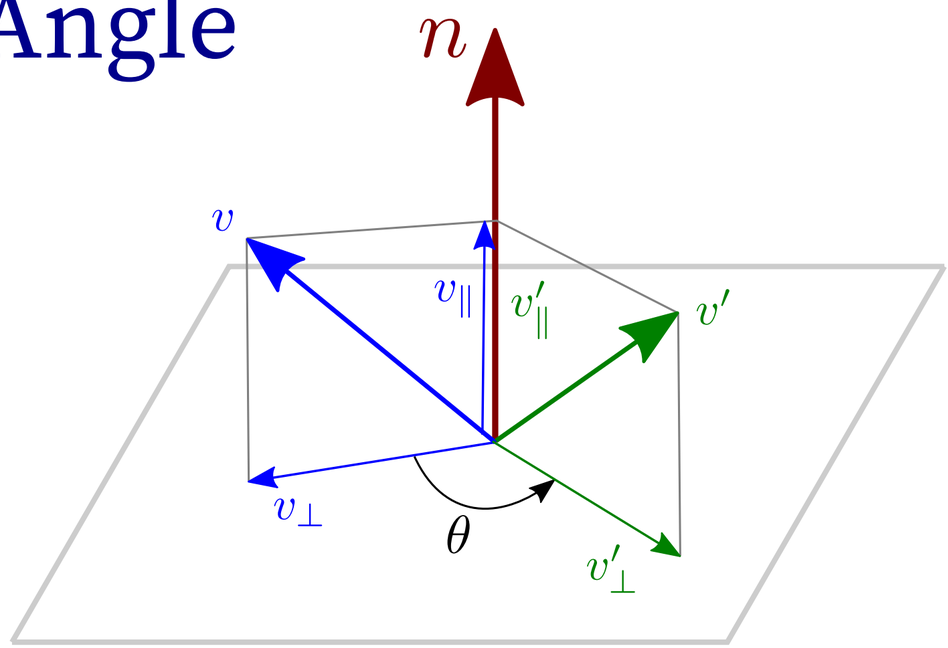
Ex. Rotation between u_1 and u_2

- Axis of rotation $n = (u_1 \times u_2)/\|u_1 \times u_2\|$
- Angle of rotation $\theta = \text{acos}(u_1 \cdot u_2)$
- Twist around u_2 can be arbitrarily chosen



Rotation representation: Axis-Angle

Applying a rotation (n, θ) to a vector v



$$v = v_\parallel + v_\perp$$
$$v' = v'_\parallel + v'_\perp$$
$$\Rightarrow v' = v_\parallel + (\cos(\theta)\, v_\perp + \sin(\theta)\, n \times v_\perp)$$

$$v_\parallel = (v \cdot n)\, n$$
$$v_\perp = v - (v \cdot n)\, n$$

$$v' = (v \cdot n)\, n + \cos(\theta)(v - (v \cdot n)\, n) + \sin(\theta)\, n \times (v - (v \cdot n)\, n)$$

$$\Rightarrow v' = \cos(\theta)\, v + \sin(\theta)\, n \times v + (1 - \cos(\theta))\, (v \cdot n)\, n$$

Rodrigues' rotation Formula

Axis-Angle as matrix

$$v' = \cos(\theta)\, v + \sin(\theta)\, n \times v + (1 - \cos(\theta))\, (v \cdot n)\, n = R(n, \theta)\, v$$

$$v' = \cos(\theta) v + \sin(\theta) \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \times v + (1 - \cos(\theta)) \begin{pmatrix} n_x & n_y & n_z \end{pmatrix} v \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}$$

$$v' = \cos(\theta)\, v + \sin(\theta) \underbrace{\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}}_{\mathrm{K}} v + (1 - \cos(\theta)) \underbrace{\begin{pmatrix} n_x^2 & n_x\, n_y & n_x\, n_z \\ n_x\, n_y & n_y^2 & n_y\, n_z \\ n_x\, n_z & n_y\, n_z & n_z^2 \end{pmatrix}}_{\mathrm{K}^2} v$$

$$v' = \begin{pmatrix} \cos(\theta) + n_x^2(1 - \cos(\theta)) & n_x n_y (1 - \cos(\theta)) - n_z \sin(\theta) & n_x n_z (1 - \cos(\theta)) + n_y \sin(\theta) \\ n_x n_y (1 - \cos(\theta)) + n_z \sin(\theta) & \cos(\theta) + n_y^2 (1 - \cos(\theta)) & n_y n_z (1 - \cos(\theta)) - n_x \sin(\theta) \\ n_x n_z (1 - \cos(\theta)) - n_y \sin(\theta) & n_y n_z (1 - \cos(\theta)) + n_x \sin(\theta) & \cos(\theta) + n_z^2 (1 - \cos(\theta)) \end{pmatrix} v$$

Rotation representation: Axis-Angle

Given a rotation (n, θ) - Corresponding rotation matrix

$$R(n,\theta) = \mathrm{I} + \sin(\theta)\,\mathrm{K} + (1-\cos(\theta))\,\mathrm{K}^2$$

$$R(n,\theta) = \begin{pmatrix} \cos(\theta) + n_x^2(1-\cos(\theta)) & n_x n_y(1-\cos(\theta)) - n_z\sin(\theta) & n_x n_z(1-\cos(\theta)) + n_y\sin(\theta) \\ n_x n_y(1-\cos(\theta)) + n_z\sin(\theta) & \cos(\theta) + n_y^2(1-\cos(\theta)) & n_y n_z(1-\cos(\theta)) - n_x\sin(\theta) \\ n_x n_z(1-\cos(\theta)) - n_y\sin(\theta) & n_y n_z(1-\cos(\theta)) + n_x\sin(\theta) & \cos(\theta) + n_z^2(1-\cos(\theta)) \end{pmatrix}$$

Pro

- Concise and general representation
- Expressive parameters in 3D space (axe, angles)
- Efficient rotation and correspondance with matrix

Cons

- No simple composition expression between two rotations.
- No direct interpolation

Composition between axis angle representation

Given two rotations (n_1, θ_1), (n_2, θ_2)

The composition $(n_3, \theta_3) = (n_1, \theta_1) \circ (n_2, \theta_2)$ is expressed by

- $\cos\left(\frac{\theta_3}{2}\right) = \cos\left(\frac{\theta_1}{2}\right)\cos\left(\frac{\theta_2}{2}\right) - \sin\left(\frac{\theta_1}{2}\right)\sin\left(\frac{\theta_2}{2}\right) n_1 \cdot n_2$

- $n_3 = \frac{1}{\tan\left(\frac{\theta_3}{2}\right)} \frac{\tan\left(\frac{\theta_2}{2}\right) n_2 + \tan\left(\frac{\theta_1}{2}\right) n_1 - \tan\left(\frac{\theta_1}{2}\right)\tan\left(\frac{\theta_2}{2}\right) n_1 \times n_2}{1 - \tan\left(\frac{\theta_1}{2}\right)\tan\left(\frac{\theta_2}{2}\right) n_1 \cdot n_2}$

- Can be derived using quaternions
- Rare to use the formula as it is

Representing 3D Rotations

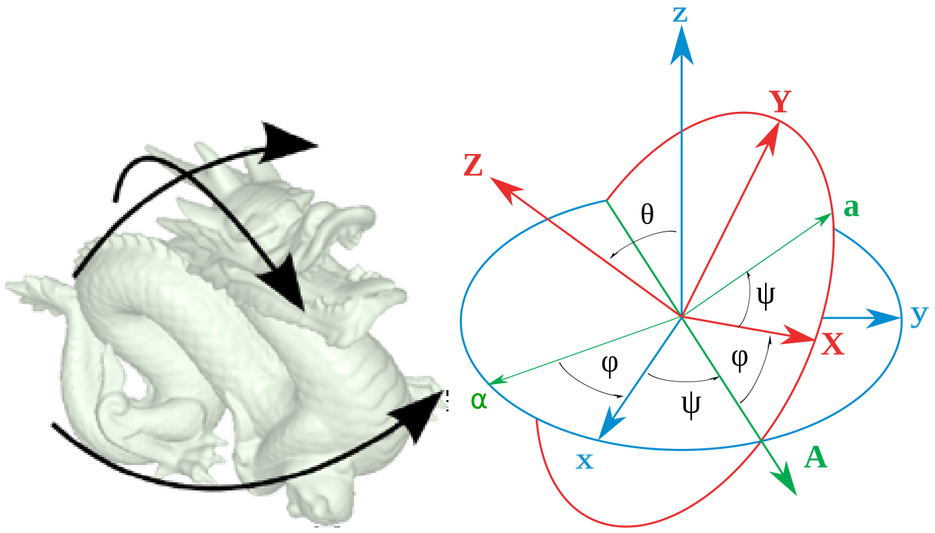
Multiple ways to represent rotations

1 - Matrices

2 - Euler (/Trait-Bryan) angles

3 - Axis-Angle

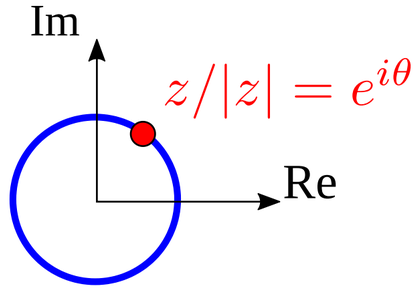
4 - Quaternion



Intuition for quaternion

Going back to 2D and complex space

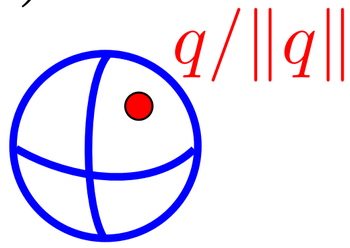
- 2D rotations have 1 dof. (rotation angle)
- 2D rotation represented as a point on a unit circle (S^1).
  - *1D structure embedded in a 2D space.*
  - *Complex space has algebraic structure adapted to model 2D rotation.*



rotations
sphere S^2

Moving to 3D

- 3D rotations have 3 dof.
  - *(point on a unit 3D sphere allows only 2 dof : not sufficient ex. orientation that can twist)*
  - point on a unit 4D sphere allows 3 dof
- 3D rotation are represented as **point on the unit sphere in 4D** (S^3).
  - *3D structure embedded in a 4D space.*
  - *Complex quaternion space has algebraic structure adapted to model 3D rotation.*



rotations
sphere S^4

Big picture of quaternions

Share similarities with 2D complex

- Complex value object with 4 components $q = (x, y, z, w)$
- Unit quaternions represent rotations $q/\|q\|$.
- Quaternion product represent rotation composition $q_1\, q_2$

Some differences

- 3D vectors are points in pure quaternion subspace $q_v = (x, y, z, 0)$
- Applying quaternion to points $q_{v'} = q\, q_v\, q^\star$

Advantage of quaternion

- Interpolation works well in quaternion space (interpolation of points on sphere).

Rotation representation: Quaternions

Quaternions: generalization of complex numbers.

$q = x\,\mathbf{i} + y\,\mathbf{j} + z\,\mathbf{k} + w$ w real part, (x, y, z) imaginary (or pure quaternion) part.

We write in short $q = (x, y, z, w)$

(don't confound with 4D vectors in homogeneous coordinates)

Properties of imaginary basis vectors

$$\begin{cases} \mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1 \\ \mathbf{ij} = -\mathbf{ji} = \mathbf{k} \\ \mathbf{jk} = -\mathbf{kj} = \mathbf{i} \\ \mathbf{ki} = -\mathbf{ik} = \mathbf{j} \\ \mathbf{ijk} = -1 \end{cases}$$

- Provides the algebraic properties

Basics operations on quaternions

- Conjugated quaternion $q^\star = (-x, -y, -z, w)$

- Quaternion norm $\|q\| = \sqrt{q\,q^\star} = \sqrt{x^2 + y^2 + z^2 + w^2}$. Unit quaternion satisfies $\|q\| = 1$.

- Quaternion product

$$q_1\ q_2 = (x_1\,\mathbf{i} + y_1\,\mathbf{j} + z_1\,\mathbf{k} + w_1)\,(x_2\,\mathbf{i} + y_2\,\mathbf{j} + z_2\,\mathbf{k} + w_2) = \ldots$$

$$q_1\ q_2 = \begin{pmatrix} x_1 w_2 + w_1 x_2 + y_1 z_2 - z_1 y_2 \\ y_1 w_2 + w_1 y_2 + z_1 x_2 - x_1 z_2 \\ z_1 w_2 + w_1 z_2 + x_1 y_2 - y_1 x_2 \\ w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2 \end{pmatrix}$$

Note: Quaternion product is
- associative: $(q_1 q_2) q_3 = q_1 (q_2 q_3) = q_1\ q_2\ q_3$
- **non-commutative**: $q_1\ q_2 \neq q_2\ q_1$

Sometimes interesting to separate *real part* w from *pure quaternion* part $s = (x, y, z)$.
- Shorthand vector form $q = (s, w)$
- Quaternion product in vector form $q_1\ q_2 = (s_1\,w_2 + s_2\,w_1 + s_1 \times s_2\ ,\ w_1\,w_2 - s_1 \cdot s_2)$

Relation between quaternion and rotation

Consider

- a quaternion $q = (s, w)$ of unit norm $\|q\| = 1$.
- a vector $v = (v_x, v_y, v_z)$ assimilated to the pure quaternion $q_v = (v, 0) = (v_x, v_y, v_z, 0)$.

Then

- $q_{v'} = \mathcal{R}_q(v) = q\, q_v\, q^\star$ is a pure quaternion $q_{v'} = (v'_x, v'_y, v'_z, 0)$
- And $v' = (v'_x, v'_y, v'_z)$ is the rotation of vector v around the axis $n = s/\|s\|$, with angle $2\,\mathrm{acos}(w)$.

Demonstration

$\mathcal{R}_q(v) = (s, w)\ (v, 0)\ (-s, w) = \ldots = ((w^2 - s^2)\, v + 2(s \cdot v)\, s + 2w\, (s \times v), 0)$

As $\|q\| = 1$, we can write $q = (s, w) = (n \sin(\phi), \cos(\phi))$, where $\|n\| = 1$

Then $\mathcal{R}_q(v) = (\underbrace{(\cos^2(\phi) - \sin^2(\phi))}_{\cos(2\phi)}\, v + \underbrace{2 \sin^2(\phi)}_{1-\cos(2\phi)} (n \cdot v)\, n + \underbrace{2\, \cos(\phi) \sin(\phi)}_{\sin(2\phi)}\, n \times v\,,\ 0)$

$\Rightarrow$ Rodrigues formula for axis n and angle 2ϕ.

The unit quaternion $q = (n\, \sin(\theta/2), \cos(\theta/2))$ represents the rotation of angle θ around the axis n.

Composition of rotations

Consider two rotation (R_1, R_2) associated to their unit quaternions (q_1, q_2).

The product $q_1\, q_2$ represents the composition $R_1 \circ R_2$.

Demonstration

We show that $\mathcal{R}_{q_1\, q_2}(v) = \mathcal{R}_{q_1} \circ \mathcal{R}_{q_2}(v)$.

$$\mathcal{R}_{q_1\, q_2}(v) = (q_1 q_2)\; v\; (q_1 q_2)^\star$$

$$\mathcal{R}_{q_1\, q_2}(v) = (q_1\; q_2)\; v\; (q_2^\star\; q_1^\star), \text{ as } (q_1 q_2)^\star = q_2^\star\; q_1^\star$$

$$\mathcal{R}_{q_1\, q_2}(v) = q_1\; (q_2\; v\; q_2^\star)\; q_1^\star$$

$$\mathcal{R}_{q_1\, q_2}(v) = q_1\; \mathcal{R}_{q_2}(v)\; q_1^\star = \mathcal{R}_{q_1} \circ \mathcal{R}_{q_2}(v)$$

Correspondance quaternion to rotation matrix

The unit quaternion $q = (x, y, z, w)$ represents the rotation given by the matrix

$$R = \begin{pmatrix} 1 - 2(y^2 + z^2) & 2(xy - wz) & 2(xz + wy) \\ 2(xy + wz) & 1 - 2(x^2 + z^2) & 2(yz - wx) \\ 2(xz - wy) & 2(yz + wx) & 1 - 2(x^2 + y^2) \end{pmatrix}$$

Demonstration

$$v' = \mathcal{R}_q(v) = q\, q_v\, q^\star = ((w^2 - s^2)\, v + 2(s \cdot v)\, s + 2w\, (s \times v), 0) \quad \text{with } s = (x, y, z)$$

$$v' = (w^2 - x^2 - y^2 - z^2)v + 2 \begin{pmatrix} x & y & z \end{pmatrix} v \begin{pmatrix} x & y & z \end{pmatrix}^T + 2w \begin{pmatrix} x & y & z \end{pmatrix} \times v$$

$$v' = \left((w^2 - x^2 - y^2 - z^2)\, \mathrm{I} + 2 \begin{pmatrix} x^2 & xy & xz \\ xy & y^2 & yz \\ xz & yz & z^2 \end{pmatrix} + 2w \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix} \right) v$$

$$v' = R\, v, \text{ with } x^2 + y^2 + z^2 + w^2 = 1$$

Summary - Correspondance quaternion / matrix-vector

Representation and Operations

	3D space	**Quaternion space**
Vector	$v = (v_x, v_y, v_z)$	$q_v = (v, 0) = (v_x, v_y, v_z, 0)$
Rotation	R (3×3 matrix)	$q = (x, y, z, w)$, $\|q\| = 1$
Apply rotation to vector	Rv	$q\ q_v\ q^\star$
Rotation composition	$R_1\ R_2$	$q_1\ q_2$

Rotation to Quaternion

Rotation of axis n and angle $\theta \Rightarrow q = (n \sin(\theta/2), \cos(\theta/2))$

Quaternion to Rotation

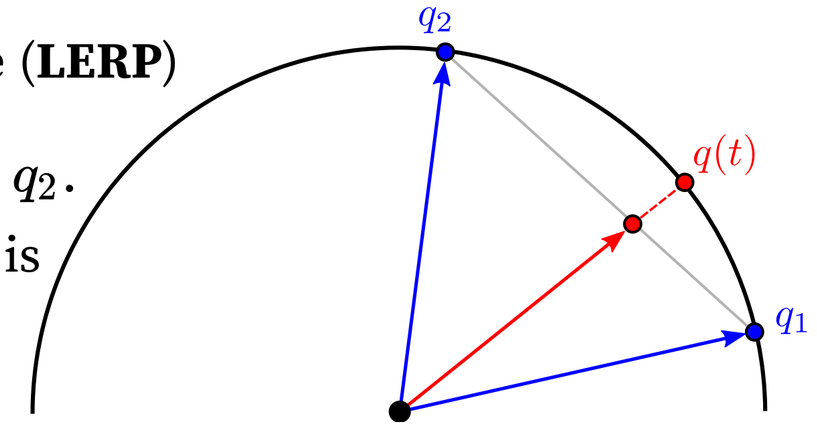
$$\text{Unit quaternion } q = (x, y, z, w) \Rightarrow R = \begin{pmatrix} 1 - 2(y^2 + z^2) & 2(xy - wz) & 2(xz + wy) \\ 2(xy + wz) & 1 - 2(x^2 + z^2) & 2(yz - wx) \\ 2(xz - wy) & 2(yz + wx) & 1 - 2(x^2 + y^2) \end{pmatrix}$$

Interpolation of Quaternion - LERP

Linear interpolation (with normalization) in quaternion space (**LERP**)

- Consider two rotations with corresponding quaternion q_1, q_2.
- The *linearly interpolated* quaternion at parameter $t \in [0, 1]$ is

$$q(t) = \frac{(1-t)\,q_1 + t\,q_2}{\|(1-t)\,q_1 + t\,q_2\|}$$



(+) Follows a shortest path (great circle on 4D sphere)

(+) Can be generalized to more general parametric curves (Splines, etc)

(-) Angular speed of orientation is not-constant

ex. extreme case of two opposite quaternions

Interpolation of Quaternion - SLERP

Spherical Linear interpolation (**SLERP**)

- Consider two rotations with corresponding quaternion q_1, q_2.
- The *spherical linear interpolated* quaternion at parameter $t \in [0, 1]$ is

$$q(t) = \frac{\sin((1-t)\Omega)}{\sin(\Omega)} q_1 + \frac{\sin(t\Omega)}{\sin(\Omega)} q_2 \quad \text{with } \cos(\Omega) = q_1 \cdot q_2$$

Demonstration

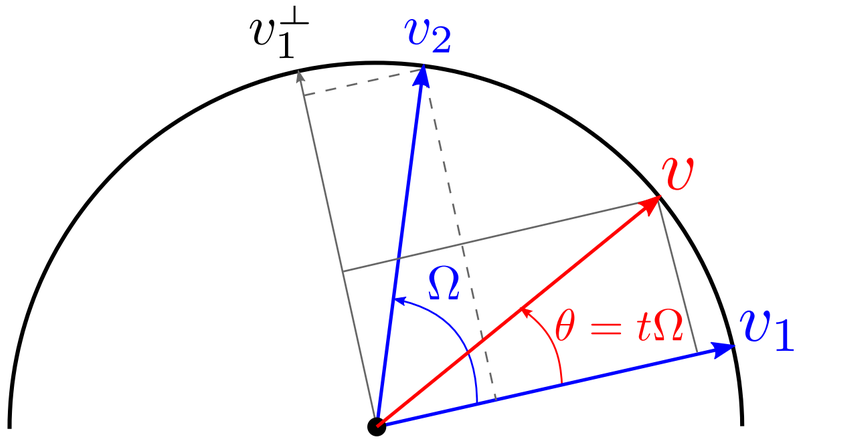
Consider

- Two unit vectors (arbitrary dimensions) v_1, v_2.
- Ω: angle b/w v_1 and v_2
- The interpolated vector v at angle $\theta = \Omega t, t \in [0, 1]$.

$v = v_1 \cos(\theta) + v_1^{\perp} \sin(\theta)$, and $v_1^{\perp} = \frac{v_2 - \cos(\Omega)\, v_1}{\sin(\Omega)}$

$\Rightarrow v = \left(\frac{\sin(\Omega)\ \cos(\theta) - \cos(\Omega)\ \sin(\theta)}{\sin(\Omega)} \right) v_1 + \frac{\sin(\theta)}{\sin(\Omega)} v_2$

$\Rightarrow v = \frac{\sin(\Omega - \theta)}{\sin(\Omega)} v_1 + \frac{\sin(\theta)}{\sin(\Omega)}$

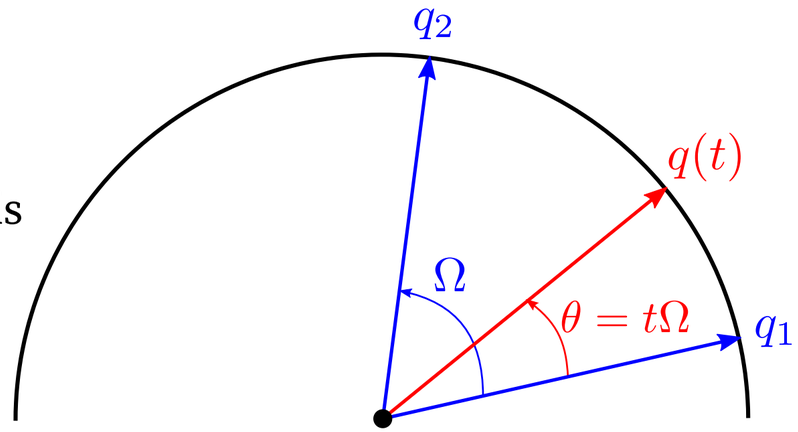


Interpolation of Quaternion - SLERP

Spherical Linear interpolation (**SLERP**)

- Consider two rotations with corresponding quaternion q_1, q_2.
- The *spherical linear interpolated* quaternion at parameter $t \in [0, 1]$ is

$$q(t) = \frac{\sin((1-t)\Omega)}{\sin(\Omega)} q_1 + \frac{\sin(t\Omega)}{\sin(\Omega)} q_2 \quad \text{with } \cos(\Omega) = q_1 \cdot q_2$$



(+) Follows a shortest path (great circle on 4D sphere)

(+) Constant angular speed

(-) Cannot be generalized to more general curves.

Cannot interpolates between more than two quaternions

Care with quaternion negation

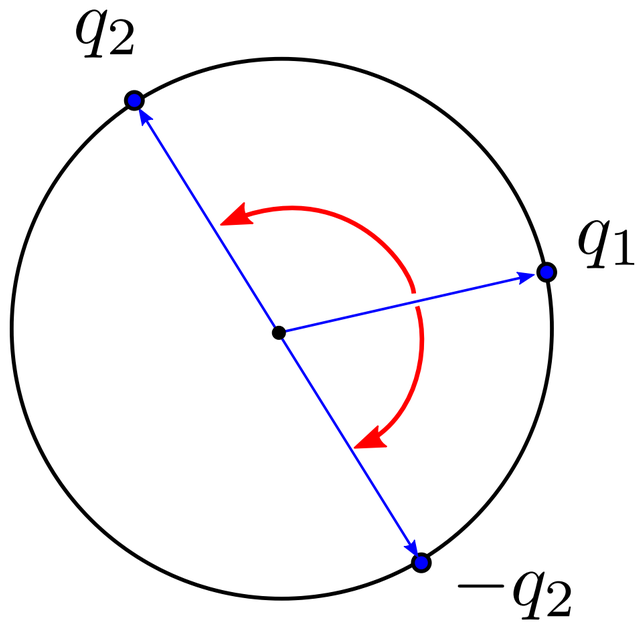
$+q$ and $-q$ correspond to the same rotation $(n \to -n, \theta \to 2\pi - \theta)$

Warning $-q$ ***does not*** *corresponds to the rotation matrix $-R$.*

But to a **different path** when interpolated in the 4D quaternion space.

$\Rightarrow$ path $q_1 \to -q_2$ is shorter than $q_1 \to q_2$ when $q_1 \cdot q_2 < 0$.

In practice we check for the shorter path before applying SLERP.



Algorithm

```
if( dot(q1,q2)<0 )
    q2 = -q2
q(t) = SLERP(q1,q2,t)
```

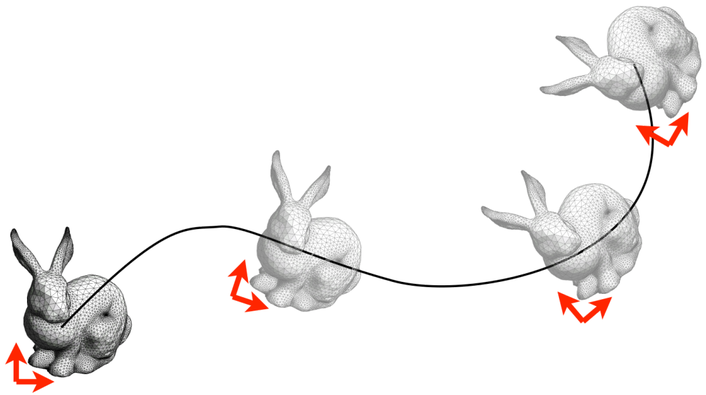
Interpolating rigid motion

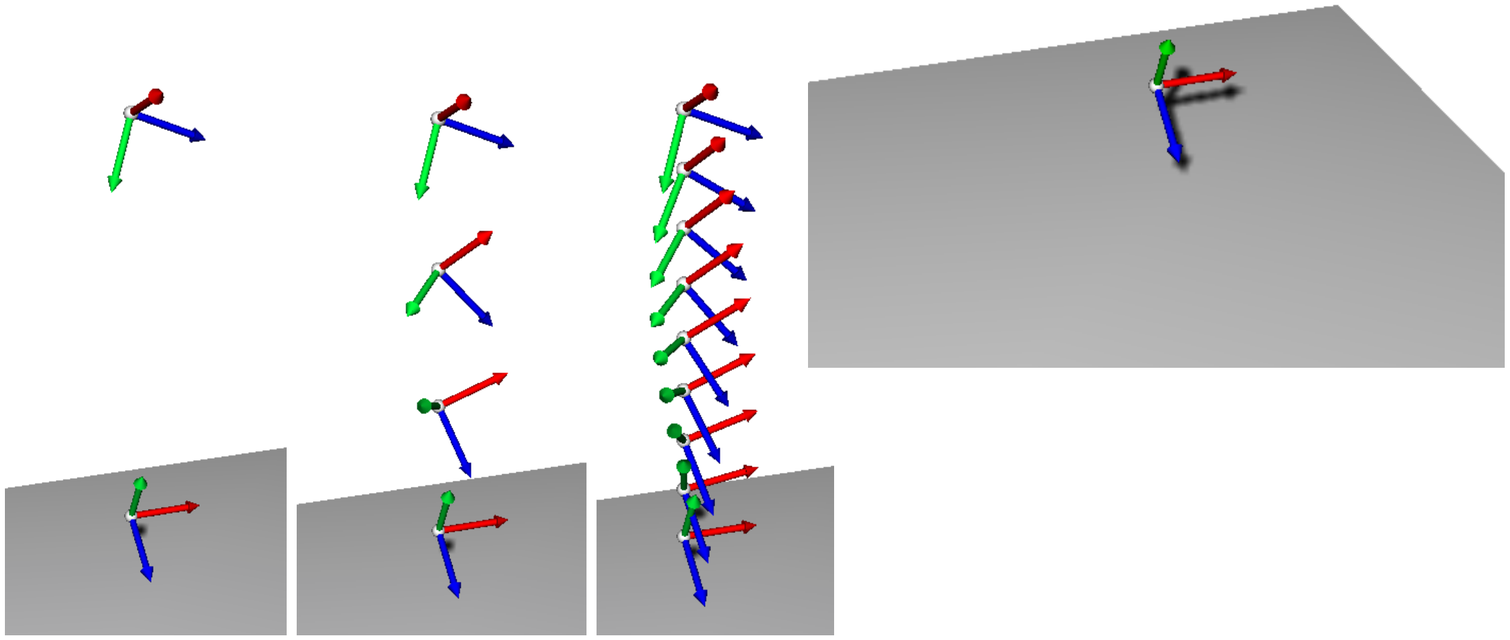
3D objects have orientation r and position p.

$\Rightarrow$ Need to interpolate both, usually handled separately.

Given two key-positions (p_1, r_1) and (p_2, r_2) *in-betweens* computed at time $t \in [0, 1]$ as

- Linear interpolation of positions $p(t) = (1 - t)\, p_1 + t\, p_2$
- Interpolate rotation with SLERP on quaternions
  - Convert $(r_1, r_2) \rightarrow (q_1, q_2)$
  - Compute $q(t) = SLERP(q_1, q_2, t)$
  - Convert back $q(t) \rightarrow r(t)$





Interpolating rigid motion - Comparison

Matrix interpolation

Euler angle interpolation

Quaternion interpolation

Handling affine transformation : Polar Decomposition

Key-pose can be given as matrix M containing rotation, but also scaling and shearing.

$\Rightarrow$ Cannot convert M to quaternion (not a rotation).

$\Rightarrow$ Separate *rotation* component from shearing and scaling component using **polar decomposition**.

[K. Shoemake and T. Duff, Matrix Animation and Polar Decomposition. Graphics Interface 92.]

- Polar decomposition: $\mathrm{M} = \mathrm{R}\,\mathrm{D}$
  - R: Rotation matrix
  - D: Positive semi-definite matrix

- Polar decomposition is obtained from SVD
  $\mathrm{SVD}(\mathrm{M}) = W\Sigma V^T$ with $\mathrm{R} = \mathrm{W}\mathrm{V}^T$, $\mathrm{D} = \mathrm{V}\Sigma\mathrm{V}^T$

- Numerically, R can be computed using the following iterative scheme
  $R_0 = \mathrm{M}$, $R_{i+1} = 0.5\,(R_i + R_i^{-T})$

Handling affine transformation

Algorithm to interpolate between two general 4×4 matrix M_1, M_2

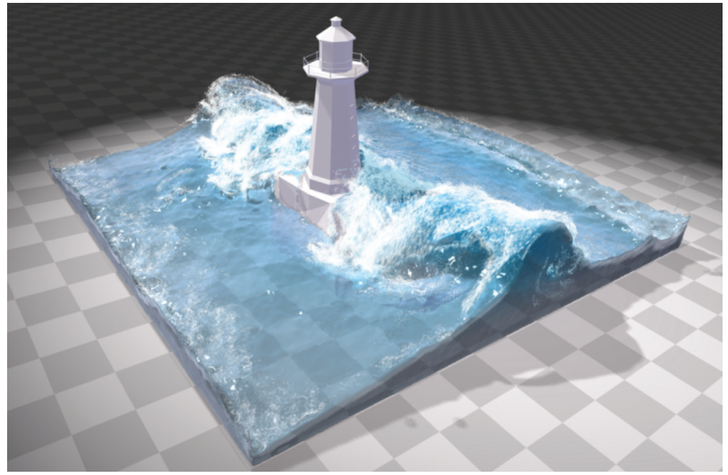
1. Extract translation p_1, p_2 from M_1, M_2.
2. Compute R_1, R_2 (3×3 rotation matrices), and D_1, D_2 (3×3 scaling/shearing matrices) from M_1, M_2
3. Interpolate linearly position and scaling/shearing $p(t) = (1-t)\, p_1 + t\, p_2$, $D(t) = (1-t)\, D_1 + t\, D_2$
4. Compute quaternions q_1, q_2 from R_1, R_2

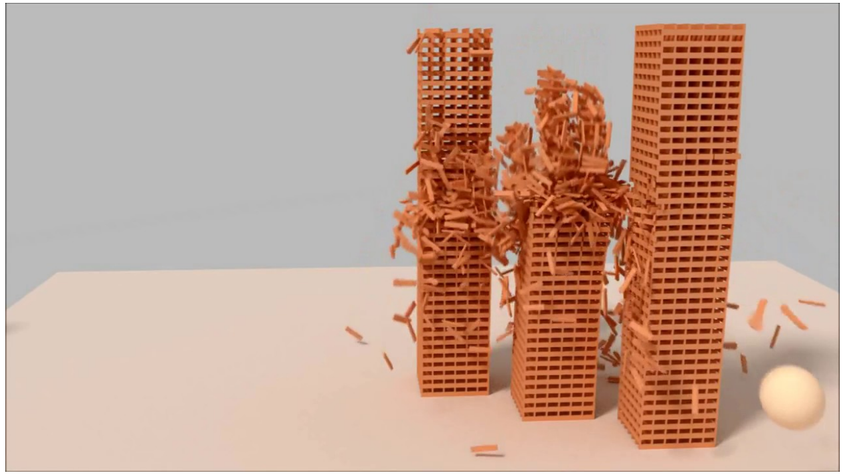
   Note $M \to q$ with $q = \left(\frac{M_{zy} - M_{yz}}{2r}, \frac{M_{xz} - M_{zx}}{2r}, \frac{M_{yx} - M_{xy}}{2r}, \frac{r}{2} \right), r = \sqrt{1 + M_{xx} + M_{yy} + M_{zz}}$
5. Compute $q(t) = \mathrm{SLERP}(q_1, q_2, t)$
6. Convert back to matrix $q(t) \to R(t)$
7. Compute final matrix $M(t) = R(t)\, D(t)$ with translation $p(t)$.

Physically based simulation

When physically based simulation is needed

- Accurate dynamics
- Teadious to model by hand or procedurally
  - Multiple interacting elements: ex. Multiple collisions: rigid bodies, hairs, etc.
  - Complex animated geometry: Cloths, fluids





General metohdology

1. Description of the system - Describe system by some parameters (positions, speed, orientation, etc).

- State of the system is known at time $t = 0$ - *Initial value problem in time*
- State of the system may be constrained in space - *Boundary value problem in space*

2. Evolution Link the evolution of the system to forces or constraints using dynamic principles and conservation laws

$\Rightarrow$ Differential equation

3. Numerical Solution Solve the differential equation using numerical iterative approaches.

Note: Fundamentally different that direct approach controling the trajectories at key-frames

- The system is set at an initial step
- We let the numerical solution build the space-time trajectory for us

(+) Allows to model complex behavior

(-) Lack of control on the result

Physically based models

1- Particles

2- Rigid bodies

3- Continuum models

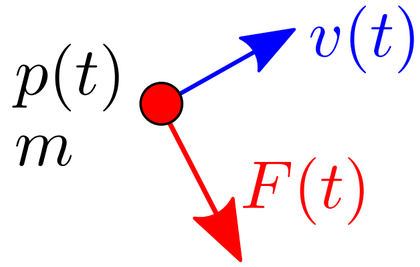
Physically-based particle system

1. Description

Particle is fully described by: Position p, Speed v, Mass m

fundamental quantities: position and linear momentum $P = m\,v$

Momentum preserved through time when no external forces are applied



2. Evolution

- Fundamental principle of the dynamics

  Force applied on particle $F(p, v, t)$

$$\begin{cases} p'(t) = v(t) \\ P'(t) = m\,v'(t) = F(p, v, t) \end{cases}$$

- Conservation of energy (ex. kinetic energy $(1/2\,m\,v^2)$+potential energy = const, etc.)

- Lagragian, or Hamiltonian

Physically-based particle system

3. Numerical Solution

ODE (Ordinary Differential Equation) formulated as an **Initial Value Problem**

$$\text{ex. } \begin{cases} p'(t) = v(t) \\ m v'(t) = F(p, v, t) \end{cases}, \quad \text{with } v(0) = v_0,\ p(0) = p_0$$

- Discretize in time $t^k = k\,h, \quad h = \Delta t =$ time step.

  $\Rightarrow$ Build a discrete numerical solution $p^k = p(t^k), v^k = v(t^k)$.

- We can consider initially the following iterative scheme

$$\begin{cases} v^{k+1} = v^k + h\,F(p^k, v^k, t^k) \\ p^{k+1} = p^k + h\,v^{k+1} \end{cases}$$

Simple to implement, reasonably OK for simple examples (more details later).

Physically-based particle system

Pro

(+) Simple to implement, and control.

(+) Efficient to compute, scalable

(+) Highly adaptable from simple particle to rigid and deformable models

Cons

(-) Limited accuracy - highly simplified model from physical point of view.

$\Rightarrow$ Dominant model in CG production for general purpose deformable model animation.

Common use: Lots of sparsely interacting particles

Physically based models

1- Particles

2- Rigid bodies

3- Continuum models

Orientation and angular speed

In addition of position and speed, rigid body is defined by its **orientation** r and **angular speed** ω.

- In matrix form $r \in \mathbb{R}^{3\times 3}$

- Angular speed $\omega = (\omega_x, \omega_y, \omega_z) \in \mathbb{R}^3$

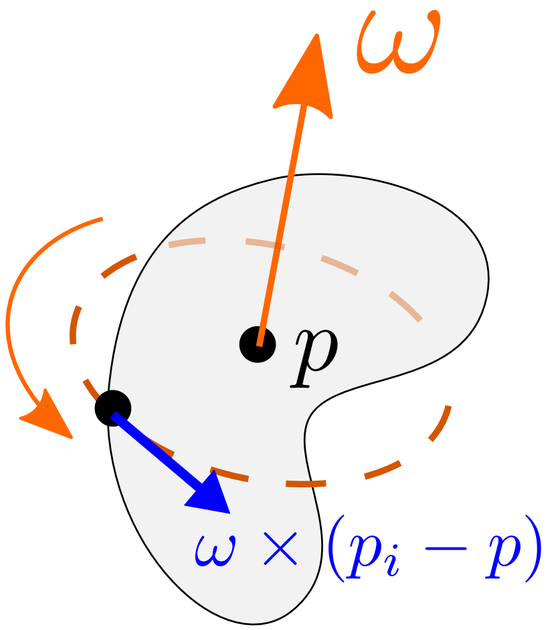
  Every vector $p_i(t)$ of the rigid body has a speed $p_i'(t) = v(t) + \omega(t) \times (p_i(t) - p(t))$

- Matrix expression of the angular speed

$$\mathrm{M}_\omega = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}$$

- Derivative of orientation matrix

  $r'(t) = \mathrm{M}_\omega(t)\, r(t)$



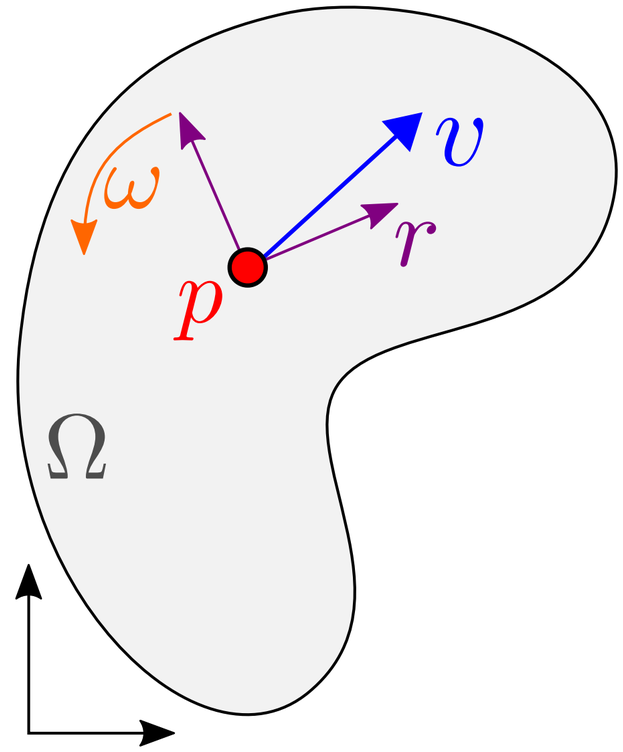
Rigid bodies system description

Similar to particle: Position $p \in \mathbb{R}^3$ and speed $v \in \mathbb{R}^3$ of the center of mass. Mass of the solid $m \in \mathbb{R}$.

- $p'(t) = v(t)$
- Fundamental quantity: position p and linear momentum $P = m\,v$.

Addition of rigid body: **Orientation** $r \in \mathbb{R}^{3\times 3}$ (in its matrix formulation), and **angular speed** $w \in \mathbb{R}^3$ of the solid.

- $r'(t) = \mathrm{M}_\omega(t)\, r(t)$
- Fundamental quantity: orientation r and **angular momentum** $L = I\,\omega \in \mathbb{R}^3$, with $I \in \mathbb{R}^{3\times 3}$ **inertia tensor** of the solid.



Intuition

- Solid acts like a particle at its center of mass + orientation and angular speed
- Mass $\rightarrow$ resistance to change of linear speed
- Inertia tensor ($\simeq$ angular mass) $\rightarrow$ resistance to change of angular speed

Definition of quantities

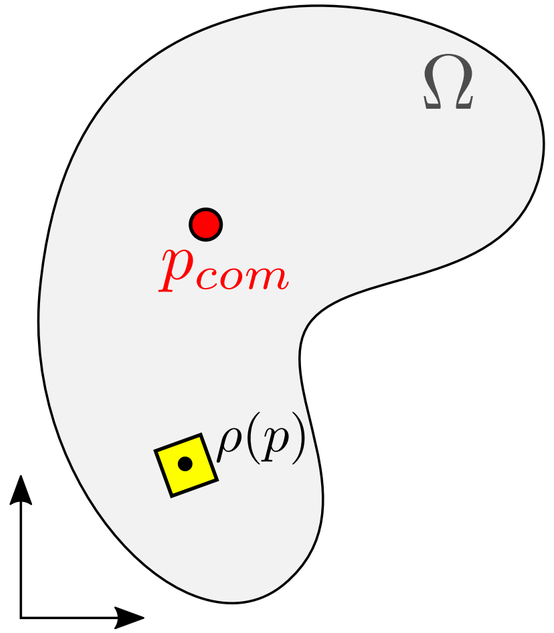
- Solid defined within a domain $\Omega \subset \mathbb{R}^3$
- With a density of mass $\rho(p)$ at each point $p \in \Omega$

- Total mass of the solid m

$$m = \int_{p \in \Omega} \rho(p) \, \mathrm{d}\Omega$$

- Position of the center of mass p_{com}

$$p_{com} = \frac{1}{m} \int_{p \in \Omega} \rho(p) \, p \, \mathrm{d}\Omega$$



Definition of inertia

Inertia tensor expressed at point q

Calling r the local coordinate $r = p - q$

$$I = \int_{r \in \Omega} \rho(r) \begin{pmatrix} I_{xx}(r) & I_{xy}(r) & I_{xz}(r) \\ I_{xy}(r) & I_{yy}(r) & I_{yz}(r) \\ I_{xz}(r) & I_{yz}(r) & I_{zz}(r) \end{pmatrix} \mathrm{d}\Omega = \int_{r \in \Omega} \rho(r) \begin{pmatrix} r_y^2 + r_z^2 & -r_x r_y & -r_x r_z \\ -r_x r_y & r_x^2 + r_z^2 & -r_y r_z \\ -r_x r_z & -r_y r_z & r_x^2 + r_y^2 \end{pmatrix} \mathrm{d}\Omega$$

$$I = \int_{r \in \Omega} \rho(r) \left(r^T r \, \mathrm{Id} - r \, r^T \right) \mathrm{d}\Omega$$

Rem.

- I is usually expressed at the center of mass $q = p_{com}$
- I depends on the body orientation. Given a rotation r: $I = r \, I_0 \, r^T$

  $\Rightarrow$ compute once I_0 in a rest position, then update it using r
- There exist a frame in which I is diagonal (principle axes of inertia). Corresponds to eigenvectors of matrix I.

Forces and torques on a solid

- Given a local force $f(p)$ acting on a position $p \in \Omega$
- Contribute to 2 global components applied on the body:

  - Total **external force** F applied on the center of mass

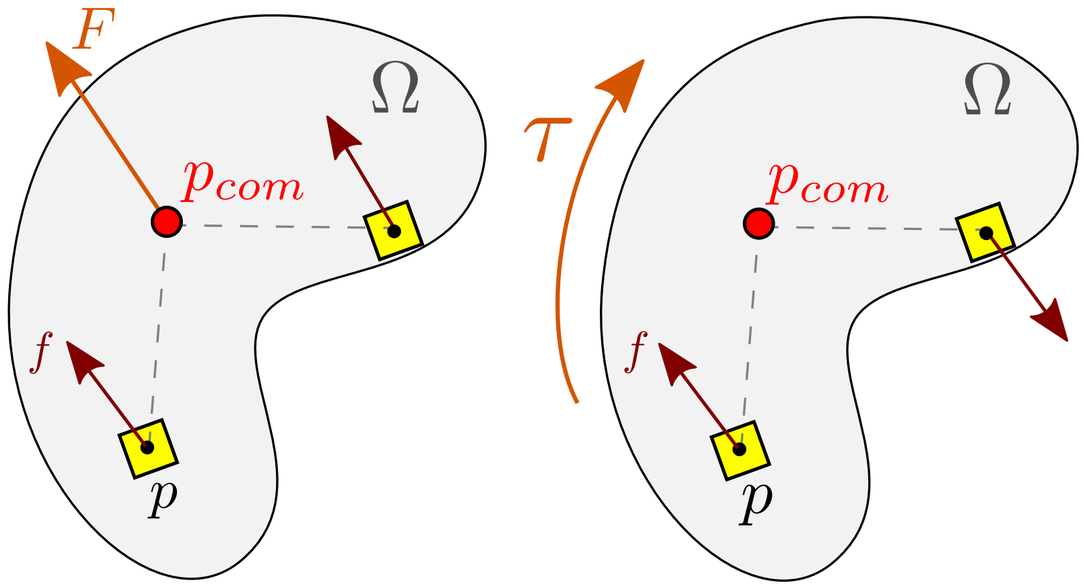
    Induce linear displacement of COM

    $$F = \int_{p \in \Omega} f(p)\, \mathrm{d}p$$

  - **Torque** τ applied on the body

    Induce spinning of the solid around its COM

    $$\tau = \int_{p \in \Omega} (p - p_{com}) \times f(p)\, \mathrm{d}\Omega$$



Dynamics

Similarly to particles

Force F is related to the change of linear momentum $F(t) = P'(t) = (m\,v)'(t)$

Specific to rigid bodies

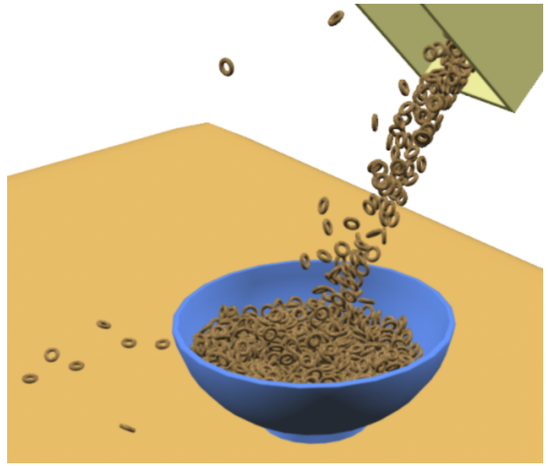
Torque τ is related to the change of angular momentum $\tau(t) = L'(t) = (I\,\omega)'(t)$

Equation of Motion

Fundamental principle of dynamics for rigid body

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} p \\ P \\ r \\ L \end{pmatrix}(t) = \begin{pmatrix} v(t) \\ F(t) \\ \omega(t)\,r(t) \\ \tau(t) \end{pmatrix}$$

Summary: Rigid solid dynamics



1. Description

State vector $u(t) = (p(t), P(t) = m\,v(t), r(t), L(t) = I(t)\,\omega(t))$

Initial condition $u(0) = (p_0, m\,v_0, r_0, I_0\,\omega_0)$

2. Temporal Evolution

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} p \\ P \\ r \\ L \end{pmatrix}(t) = \begin{pmatrix} v(t) \\ F(t) \\ \omega(t)\,r(t) \\ \tau(t) \end{pmatrix}$$

- $I(t) = r(t)\,I_0\,r(t)^T$
- $\omega(t) = I^{-1}(t)\,L(t)$

In practice:

- Shape and density ρ are known
- Compute I_0 for the given shape
- p_0, v_0, r_0, ω_0 are Initial Conditions

Iterate over t^k

- Compute $F(t^k)$ and $\tau(t^k)$ given external forces
- Compute $I(t^k) = r(t^k)\,I_0\,r(t^k)^T$
- Compute $\omega(t^k) = I^{-1}(t^k)\,L(t^k)$
- Update state vector $p^{k+1}, P^{k+1}, r^{k+1}, L^{k+1}$ at t^{k+1}

Use of quaternion

Relation $r'(t) = \omega(t)\, r(t)$ leads in numerical drift from rotation matrix

ex. $r^{k+1} = (\mathrm{Id} + h\,\omega^k)\, r^k$ (explicit scheme)

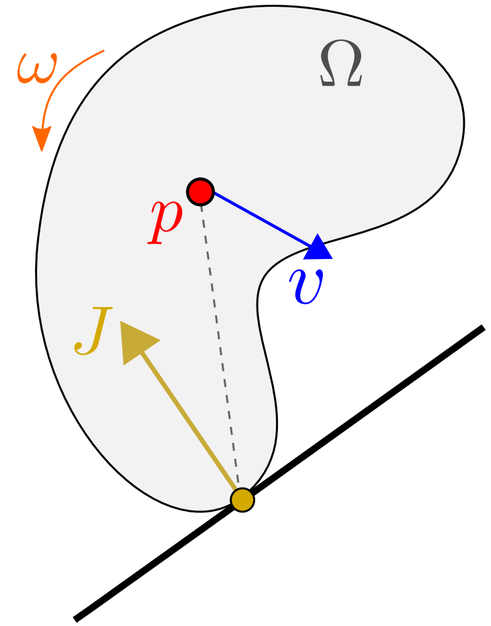
Using quaternion leads to more robust behavior

- Quaternion expression: $q'(t) = \frac{1}{2}\, q_\omega(t)\, q(t)$, with $q_\omega(t) = (\omega(t)\,,\ \ 0))$

- Quaternion is forced to keep a unit norm

ex. $\begin{cases} q^{k+1} = q^k + \frac{1}{2}\, q_\omega^k\, q^k \\ q^{k+1} = q^{k+1} / \|q^{k+1}\| \end{cases}$

Collisions in rigid bodies

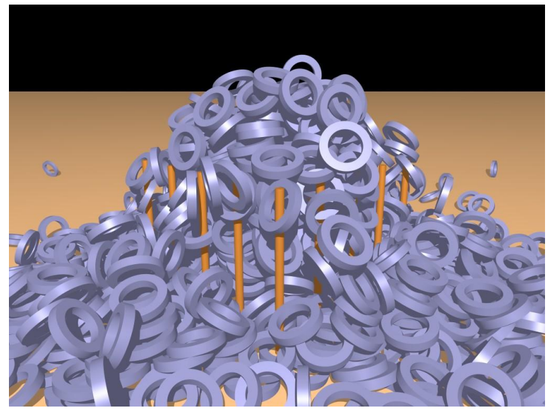


Collisions at position p change both linear and angular velocity

Similarly to particle: use of impulse (sudden change of speed) J.

Impulse split into

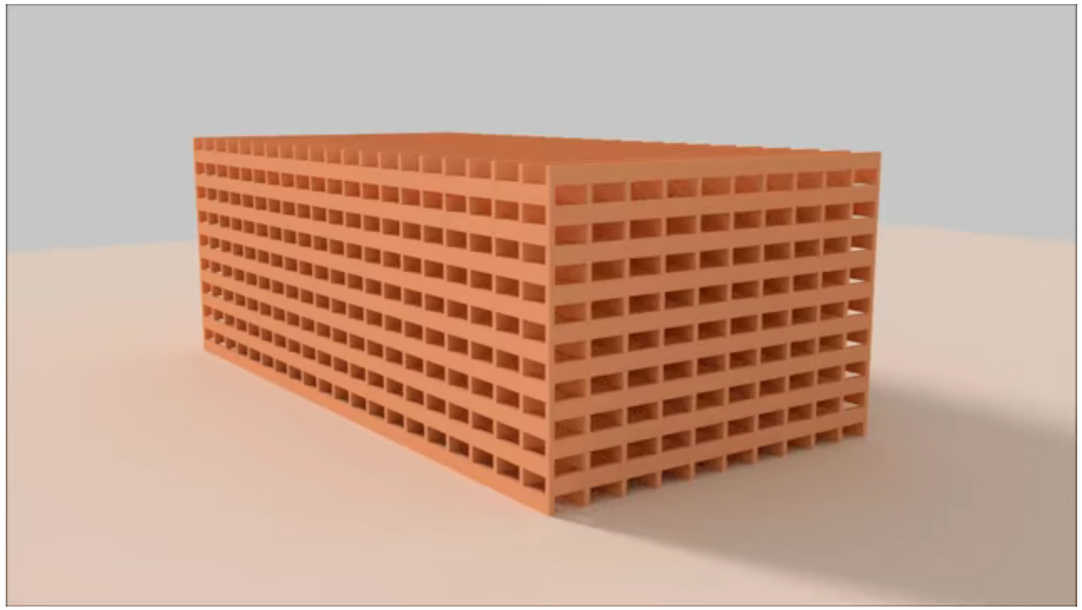
- *Force impulse* $\Delta P = m\,\Delta v = J$
- *Torque impulse* $\Delta L = I\,\Delta\omega = (p - p_{com}) \times J$



More details [D. Barff. Physically Based Modeling. SIGGRAPH Course Notes 1999]

Rigid bodies usage

- Standard usage for rigid bodies motions
  - Limited to non-deformable shapes
- Common in VFX (explosions), and *simulation games* (cars, airplanes, etc).
- Standard library: Bullet physics (ex. used in Blender).



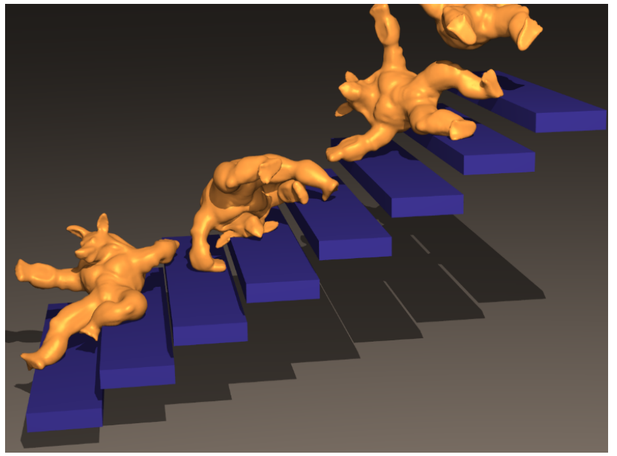
Physically based models

1- Particles

2- Rigid bodies

3- Continuum models

Deformation of a continuous shape



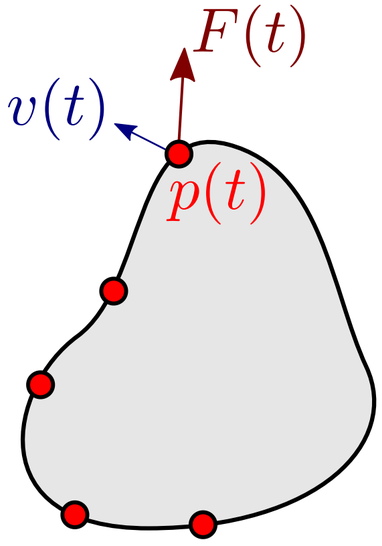
Every part of the shape can be deformed

ex. Describing elastic shapes, visco-elastic shapes, fluids, etc.

Two ways to describe the deforming object

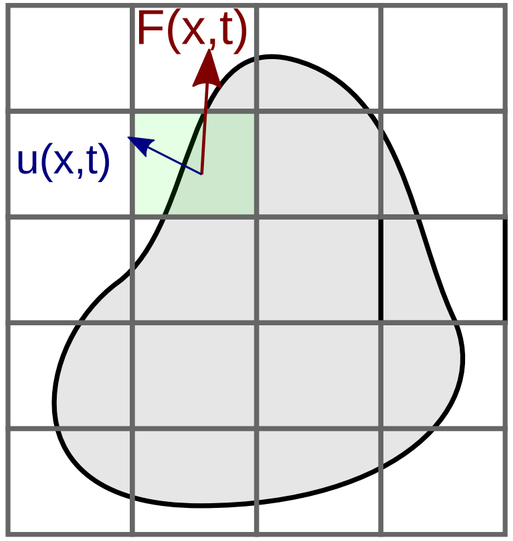
1. Lagrange representation

Positions follow the object deformation



2. Euler representation

Positions are fixed in 3D space

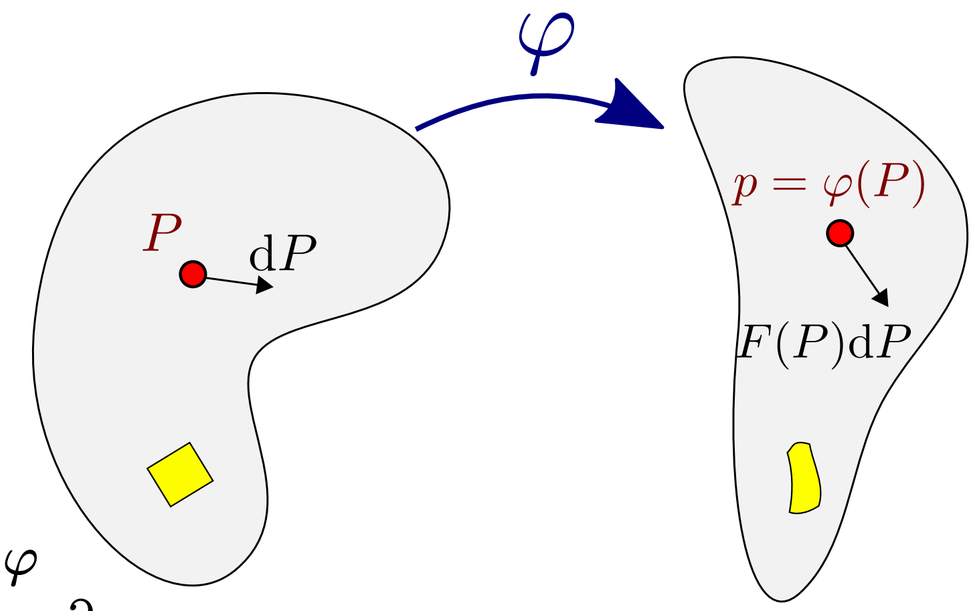


Deformation the Lagrangian description

Deformation map $\varphi : \mathbb{R}^3 \to \mathbb{R}^3$ such that $p = \varphi(P)$

P position in the reference undeformed shape

p position in the deformed configuration.



Deformation Gradient F

- $F(P) = \dfrac{\partial \varphi}{\partial P}(P) = \dfrac{\partial p}{\partial P} \in \mathbb{R}^{3\times 3}$
- Characterizes the local deformation associated to φ

  Position $P + \mathrm{d}P$ is mapped into $\varphi(P + \mathrm{d}P) \simeq p + \dfrac{\partial \varphi}{\partial P}\,\mathrm{d}P$

- $F(P) = \begin{pmatrix} \frac{\partial \varphi_x}{\partial X} & \frac{\partial \varphi_x}{\partial Y} & \frac{\partial \varphi_x}{\partial Z} \\ \frac{\partial \varphi_y}{\partial X} & \frac{\partial \varphi_y}{\partial Y} & \frac{\partial \varphi_y}{\partial Z} \\ \frac{\partial \varphi_z}{\partial X} & \frac{\partial \varphi_z}{\partial Y} & \frac{\partial \varphi_z}{\partial Z} \end{pmatrix}$

Strain

Deformation gradient F describe both

- Rigid transformation (translation, rotation) - not related to material effort
- Any other deformation inducing local length change - related to material effort

Strain ϵ is a measure of deformation ignoring rigid transformation.

Several possible measure of strain

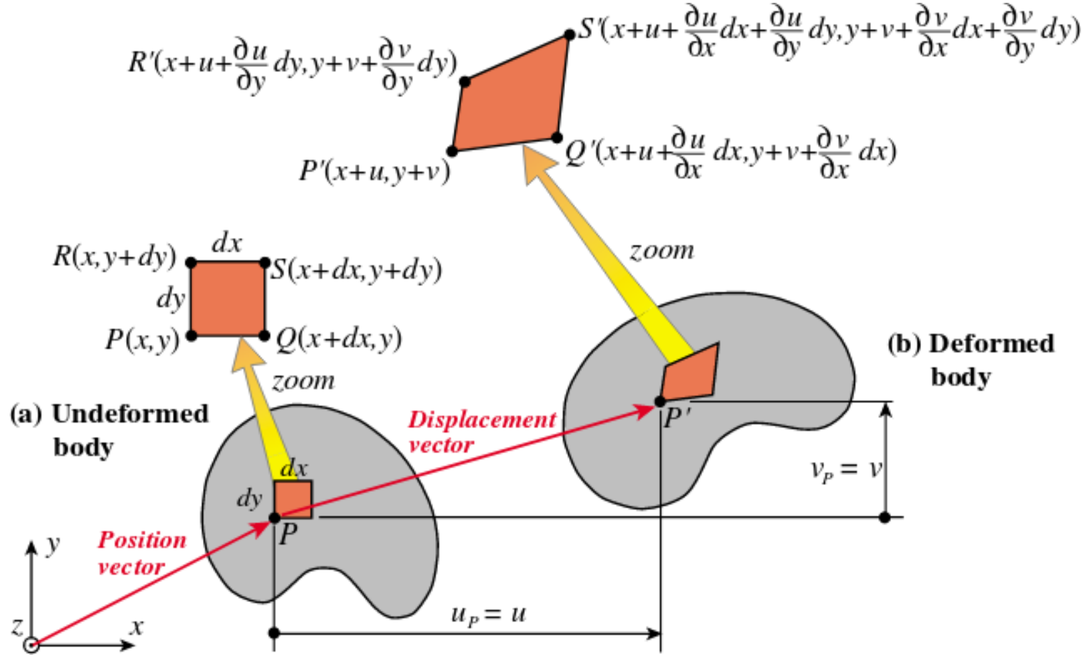
- Green strain tensor $\epsilon = \frac{1}{2}\left(F\,F^T - \mathrm{Id}\right)$

  (+) If φ is a rotation $F = R \Rightarrow \epsilon = 0$

  (-) Non linear in p

- Linearized Cauchy strain $\epsilon = \frac{1}{2}(F^T + F) - \mathrm{Id}$

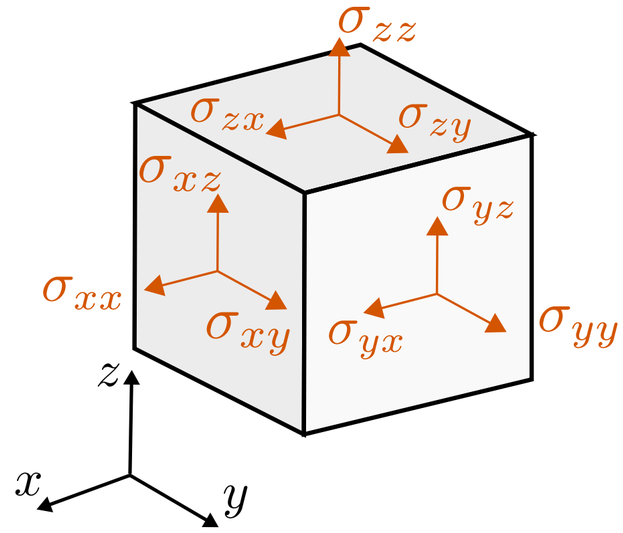
  Used for small deformations



Stress

Stress $\sigma \in \mathbb{R}^{3\times 3}$ describes internal forces (per area unit) induced by the local deformation (strain) in any direction

Constitutive Relation: Relation between stress and strain, characterize a type of material.



For linear constitutive relation:

$\sigma_{ij} = \sum_{k,l} C_{ijkl}\,\epsilon_{kl}$, C: stiffness tensor (81 coefficients)

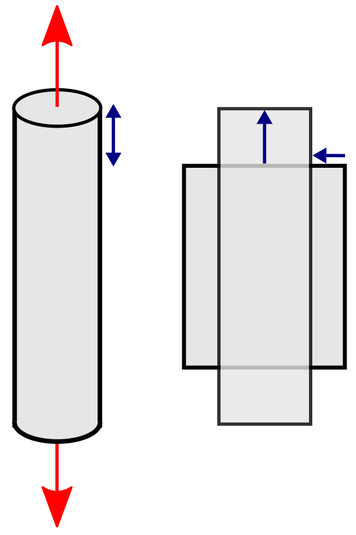
- Strain energy $U(\epsilon) = \sum_{i,j,k,l} C_{ijkl}\,\epsilon_{ij}\,\epsilon_{kl}/2$

For homogeneous isotropic elastic material, constitutive relation simplifies to

$\sigma = 2\mu\,\epsilon + \lambda \mathrm{tr}(\epsilon)\,\mathrm{Id}$, (μ, λ): Lamé parameters

Related to common mechanical modulus : Young' modulus Y and Poisson's ratio ν

$\mu = \frac{Y}{2(1+\nu)}, \lambda = \frac{Y\nu}{(1+\nu)(1-2\nu)}$



Evolution equation

Fundamental principle of dynamics in the entire volume Ω

Change of momentum = External forces (in volume) + Traction (stress applied on exterior surface normals)

$$\Rightarrow \underbrace{\int_\Omega \rho\, p''(t)\,\mathrm{d}\Omega}_{\text{Change of momentum}} = \underbrace{\int_\Omega F(t)\,\mathrm{d}\Omega}_{\text{External forces}} + \underbrace{\int_{\partial\Omega} \sigma\, n\,\mathrm{d}S}_{\text{Traction force on the boundary}}$$

Using divergence theorem $\displaystyle\int_{\partial\Omega} \sigma\, n\,\mathrm{d}S = \int_\Omega \mathrm{div}(\sigma)\,\mathrm{d}\Omega$

Equation in volume satisfied at each position $p \in \Omega$

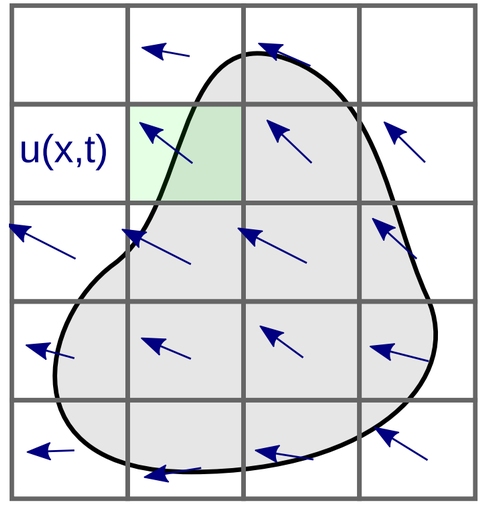
$$\boxed{\rho\, p''(t) = F(t) + \mathrm{div}(\sigma(t))} \qquad \sigma = \begin{pmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{pmatrix} \qquad \mathrm{div}(\sigma) = \begin{pmatrix} \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{yx}}{\partial y} + \frac{\partial \sigma_{zx}}{\partial z} \\ \frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \sigma_{zy}}{\partial z} \\ \frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{yz}}{\partial y} + \frac{\partial \sigma_{zz}}{\partial z} \end{pmatrix}$$

Euler formulation

In Euler formulation quantities are expressed at fixed position in 3D space.

Deformation described by velocity $u(p,t)$ at a given 3D fixed point $p=(x,y,z)$ at time t.

- Do not require anymore a reference shape
- Usefull for heavily deforming shapes (ex. fluids, gaz).



- Change of speed during dt

$$\frac{\mathrm{d}u}{\mathrm{d}t}(p,t)=\frac{\partial u}{\partial t}+\frac{\partial p}{\partial t}\cdot\frac{\partial u}{\partial p}=\frac{\partial u}{\partial t}+u\cdot\nabla u\,.$$ Called *material derivative*.

- Similarily to Lagrangian derivation:
  - Stress-rate tensor ϵ (rate of change of deformation in a neighborhood of a point) expressed with respect to u $\epsilon=\frac{1}{2}\left(\nabla u+\nabla u^T\right)$
  - Strain-rate tensor σ (rate of change of deformation in a neighborhood of a point).

Equation of motion for a fluid

- Fundamental principle of dynamics on linear momentum

$$\rho \frac{\mathrm{d}u}{\mathrm{d}t} = F + \mathrm{div}(\sigma)$$

$$\Rightarrow \rho \frac{\partial u}{\partial t} = F + \mathrm{div}(\sigma) - \rho\, u \cdot \nabla u\,. \qquad \text{The term } u \cdot \nabla u \text{ is called } \textit{convection}.$$

- External force: weight $F = \rho\, g$

- Stress decomposed into

$$\sigma = \sigma_{viscous} + \sigma_{pressure}$$

$$\sigma_{pressure} = -p\,\mathrm{Id} \text{ } \textit{(pressure acts along normal of surface elements)}$$

$$\rho \frac{\partial u}{\partial t} = \rho\, g - \rho\, u \cdot \nabla u + \mathrm{div}\left(\sigma_{viscous} - p\,\mathrm{Id}\right)$$

$$\Rightarrow \rho \frac{\partial u}{\partial t} = \rho\, g - \rho\, u \cdot \nabla u - \nabla p + \mathrm{div}\left(\sigma_{viscous}\right)$$

Navier-Stokes equation

- Newtonian fluid $\Rightarrow$ Linear relation between strain-rate ϵ and stress-rate $\sigma_{viscous}$
  - $\sigma_{viscous} = 2\mu\,\epsilon = \mu\left(\nabla u + \nabla u^T\right)$, μ constant viscosity parameter
- Incompressible fluid $\Rightarrow \operatorname{div}(u) = 0$

Equation of motion

$$\Rightarrow \rho\,\frac{\partial u}{\partial t} = \rho\,g - \rho\,u\cdot\nabla u - \nabla p + \operatorname{div}\left(\mu\left(\nabla u + \nabla u^T\right)\right)$$

- Noting that $\operatorname{div}(\nabla u^T) = \nabla \operatorname{div}(u) = 0$
- And $\operatorname{div}(\nabla u) = \triangle u$
- Set $\nu = \mu/\rho$

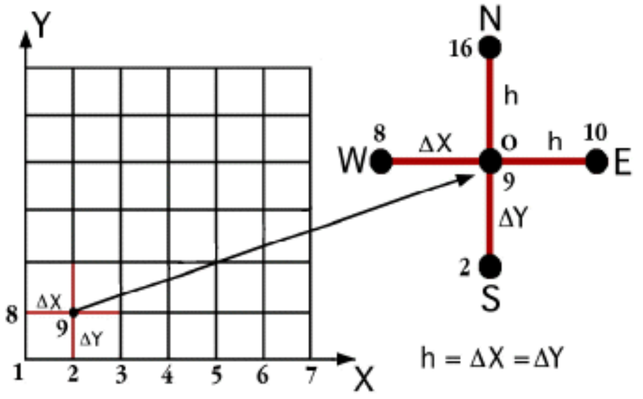
$$\Rightarrow \frac{\partial u}{\partial t} = g - u\cdot\nabla u - \frac{1}{\rho}\nabla p + \nu\triangle u$$

Navier-Stokes equation for incompressible Newtonian fluid.

Numerical solutions

Lagrangian and Euler description leads to PDE (Partial Differential Equations)

In general: no explicit solutions $\Rightarrow$ approximate numerical solution



Finite Differences (FD)

- Discretize in space on a grid Δx and time Δt
- Use numerical approximation of derivatives using masks in space and time
  - (+) Very general, simple to setup
  - (+) Works well with rectangular grid (ex. Euler description)
  - (-) Difficult to handle shape boundaries
  - (-) Instabilities

Finite elements method (FEM)

- Discretize the shape into simple elements. Build continuous function on each element.
  In CG: Elements are tetrahedron (in volume). Continuous function are barycentric coordinates (linear interpolation functions).
- Integrate PDE over each element (weak formulation), leads to a linear syst
  - (+) Handle boundaries (ex. Deforming solid)
  - (+) Guarantee on accuracy
  - (-) Complex to set up, and computationally heavy
  - (-) Requires good quality meshing