Introduction to Computer Animation

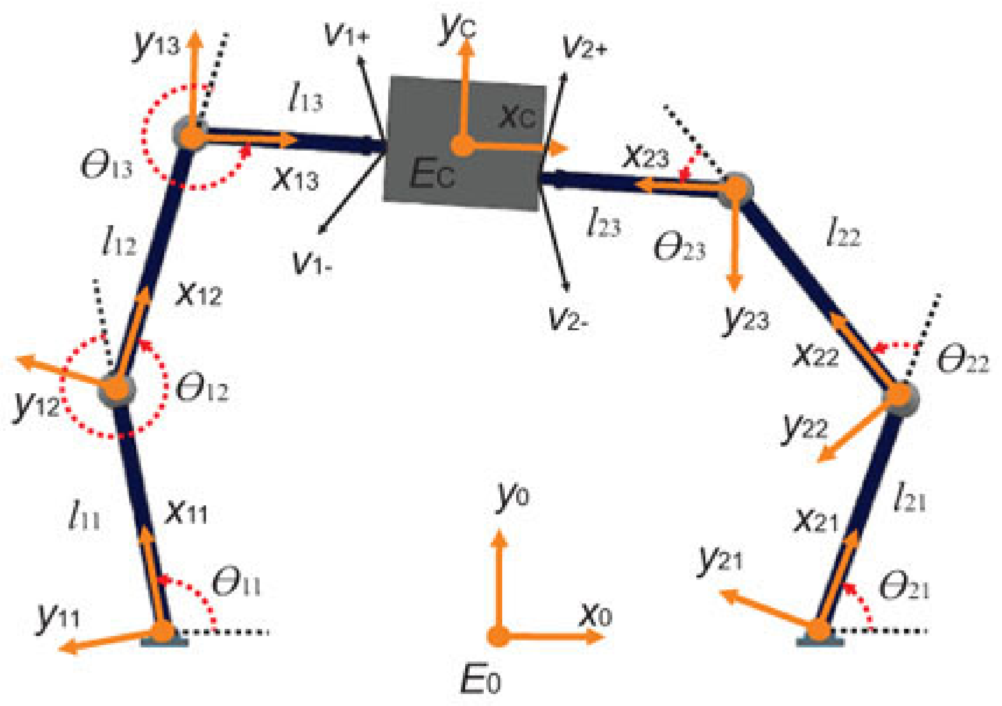
Animation in production

Animation in Computer Graphics

Two main ways to describe animation

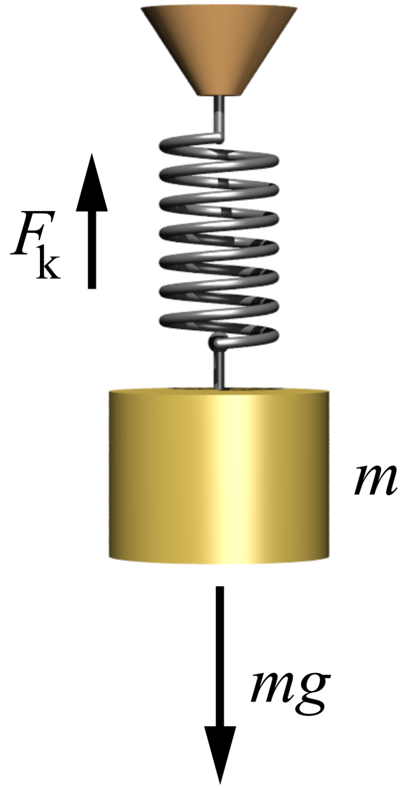
1- Kinematics

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

Ways to generate animation

1- Descriptive animation

Fully designed trajectory

2- Motion tracking

Real data aquisition

3- Procedural generation

Automatic synthesis

4- Physically based simulation

5- Learning-based synthesis

Descriptive Animation

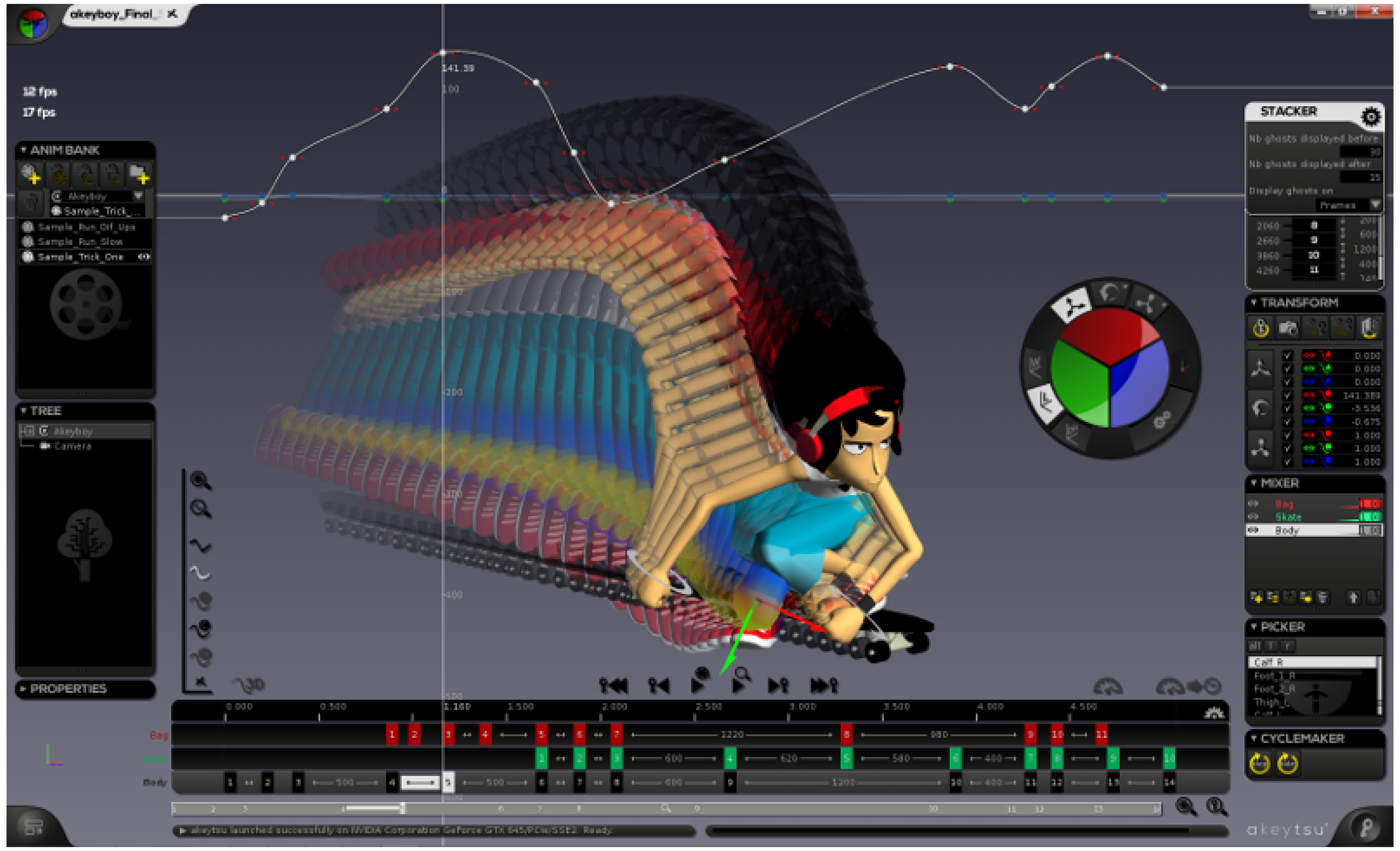
The artist/an algorithm fully describes the motion and deformation.

pro

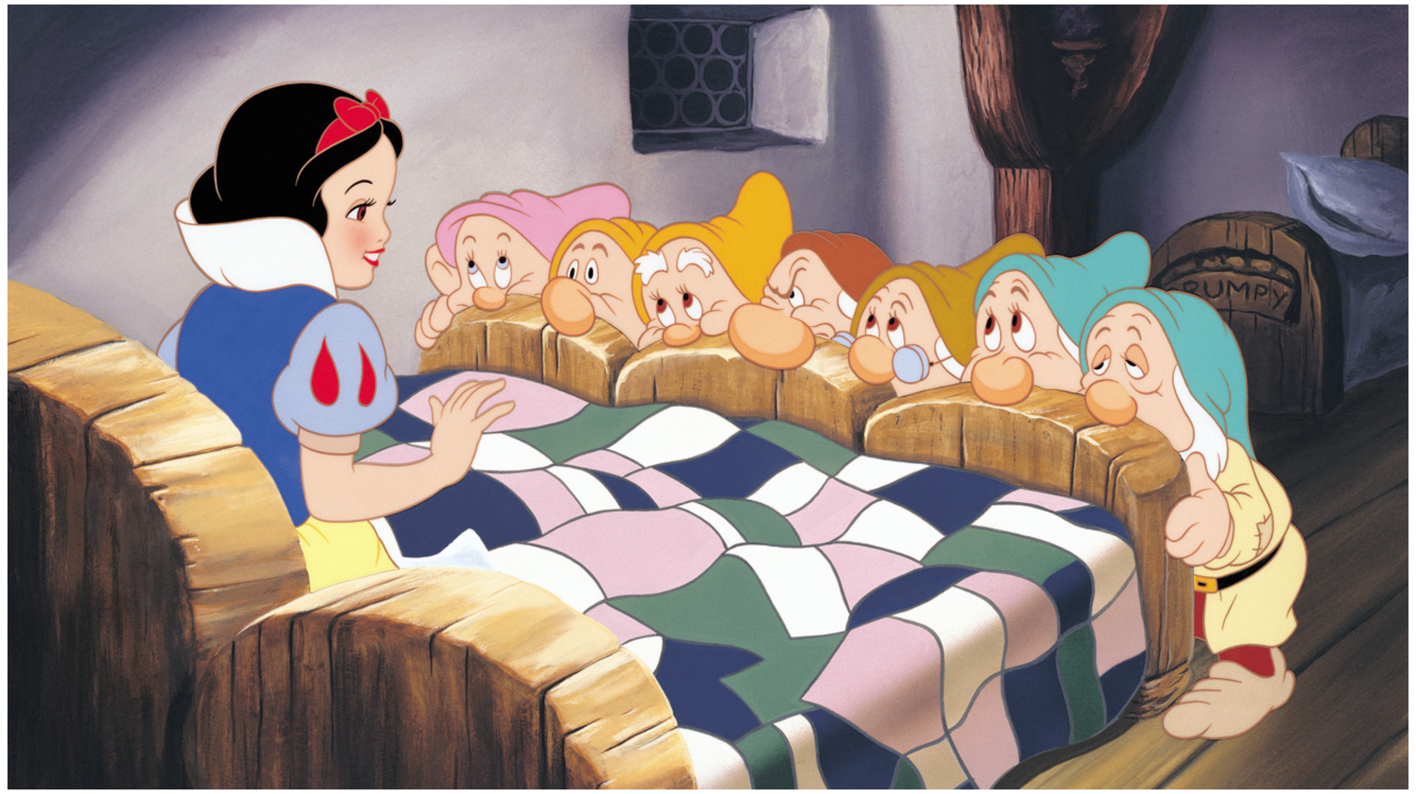
+ Full control on the result

cons

- May introduce un-physical artifacts



History of CG Animation



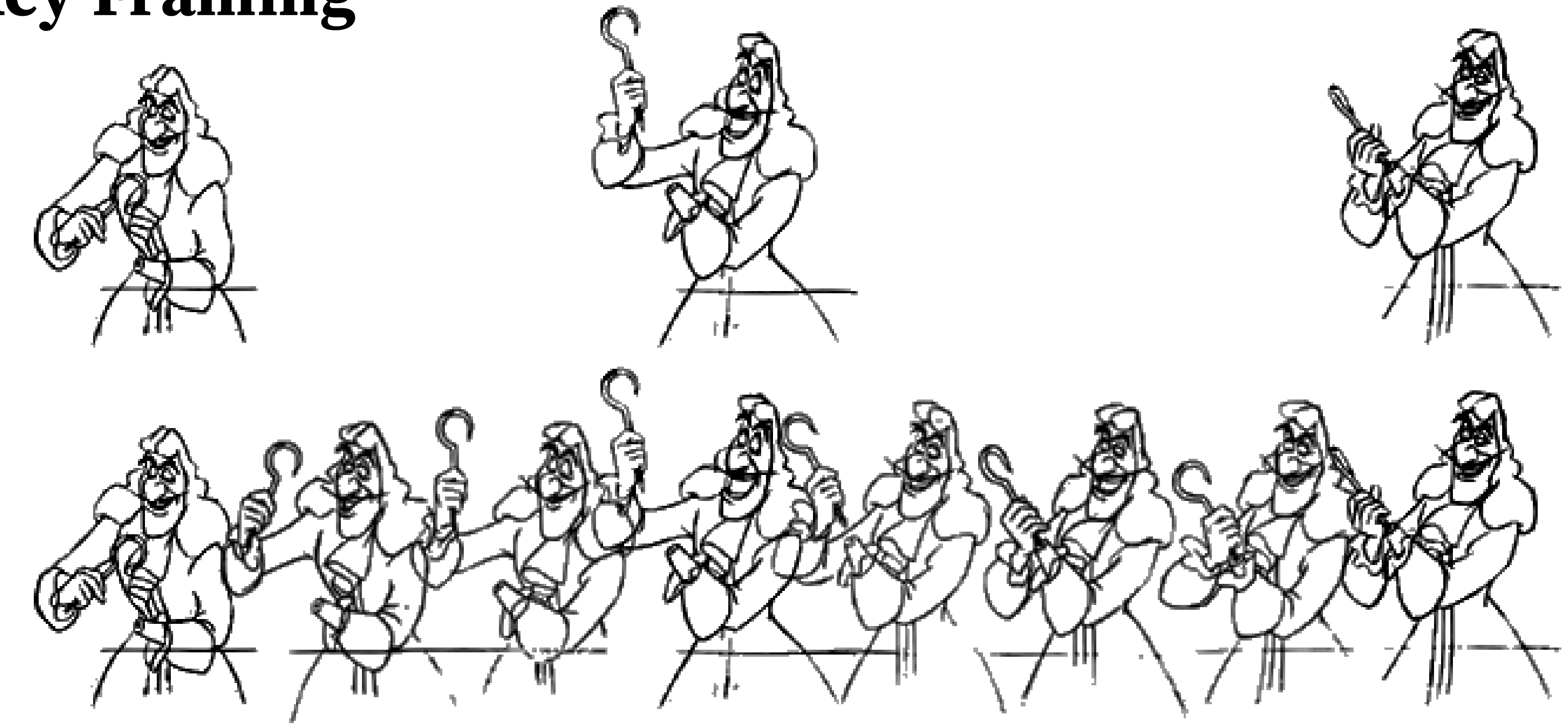
First production of animated films (Disney)

- 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"

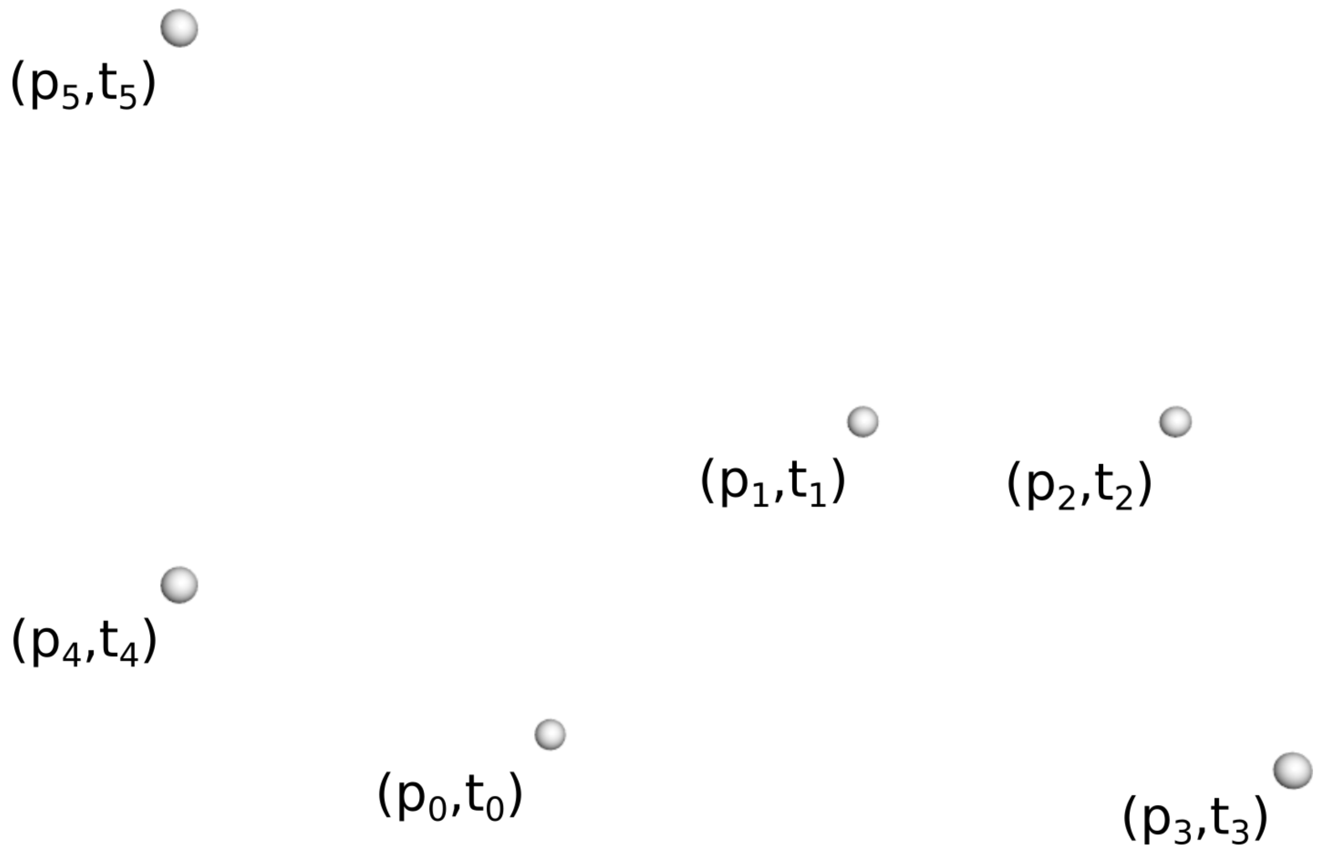
Key Framing in CG

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

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

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

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

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

(α_i) polynomial basis function of degree d

Which polynomials/degree choose ?

Lagrange polynomial interpolation

Naive idea: Interpolate all points at once

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

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

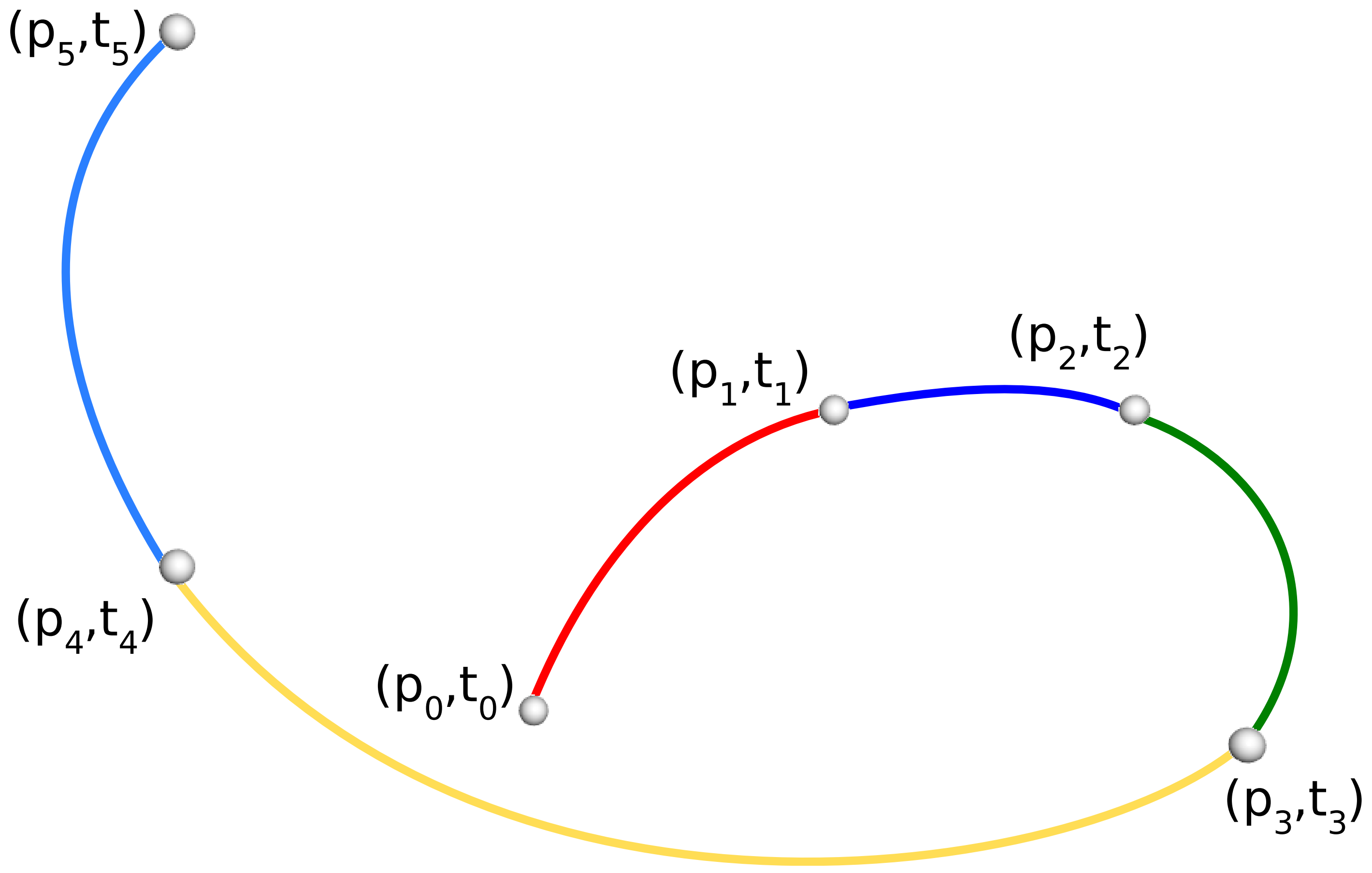
- Define on each part a polynomial
- Smooth junctions between them.

How to choose the polynomial

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

=> In Graphics cubic polynomials are often used

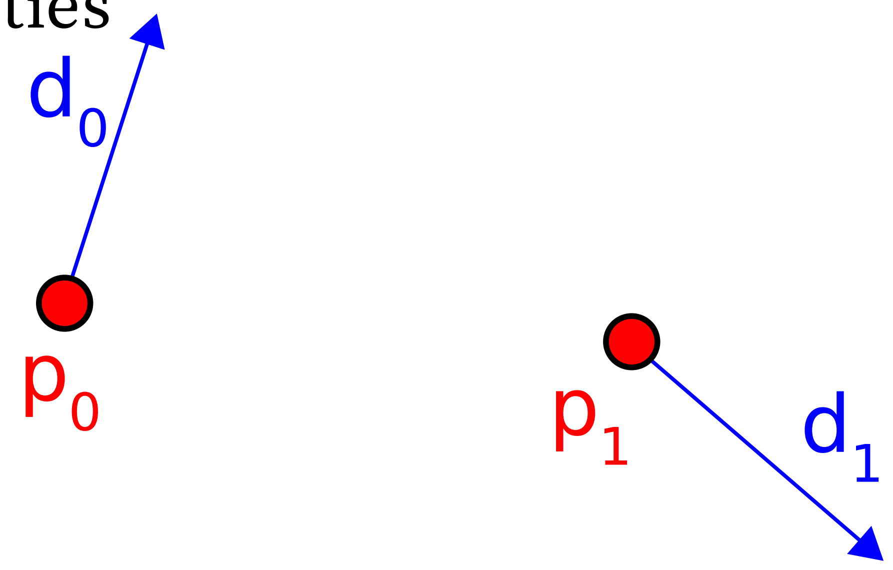
Allows up to C^2 junctions



Each color is another polynomial

Hermite interpolation

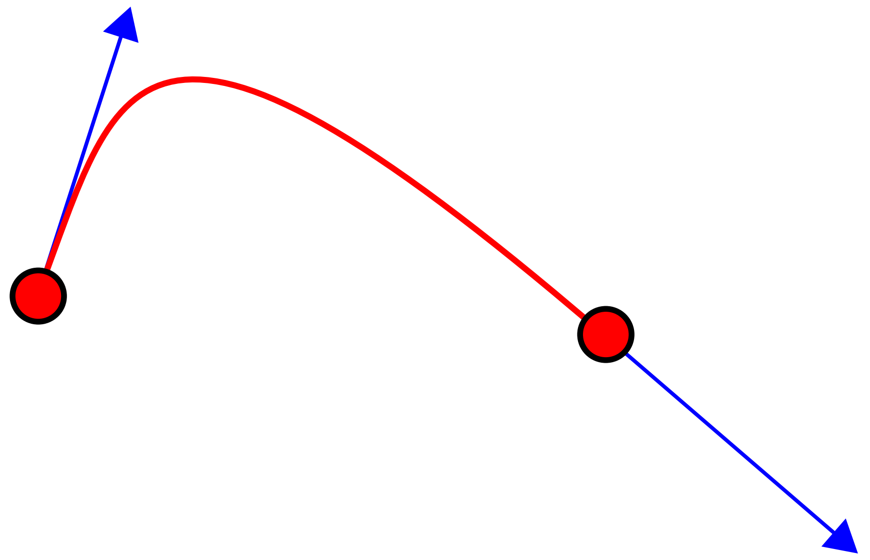
Hermite interpolation : cubic curve interpolating points and derivatives at extremities



Consider the following constraints

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

$\Rightarrow$ System of equations



$$\left\{\begin{array}{rrrrl} & & & c_0 & = p_0 \\ c_3 + & c_2 + & c_1 + & c_0 & = p_1 \\ & & c_1 & & = d_0 \\ 3\,c_3 + & 2\,c_2 + & c_1 & & = d_1 \end{array}\right. \Rightarrow \left\{\begin{array}{l} c_0 = p_0 \\ c_1 = d_0 \\ c_2 = -3p_0 + 3p_1 - 2d_0 - d_1 \\ c_3 = 2p_0 - 2p_1 + d_0 + d_1 \end{array}\right.$$

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

For arbitrary $t \in [t_i, t_{i+1}]$, we set $s = \dfrac{t - t_i}{t_{i+1} - t_i}$

Interpolating curve

Our initial problem: set of multiple keyframes position+time

Two solutions

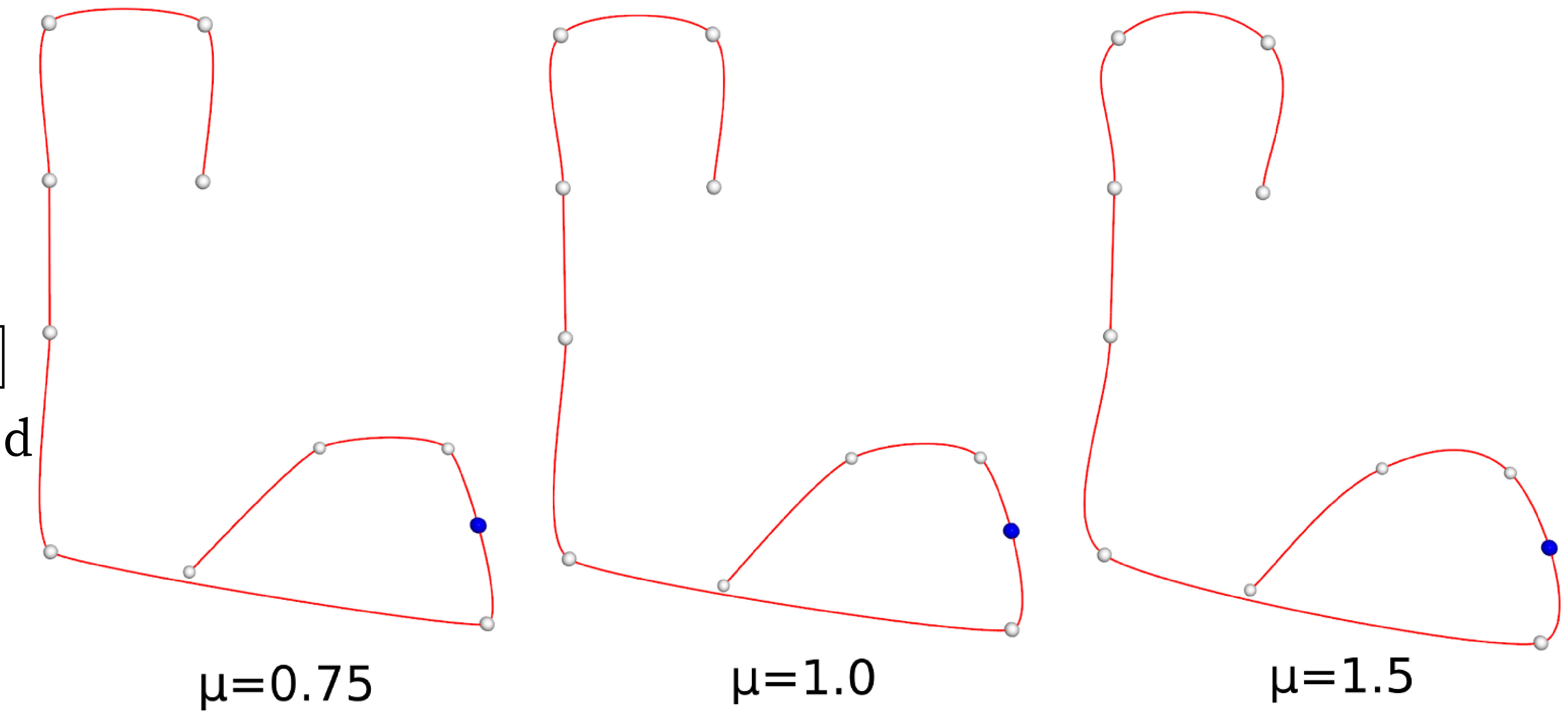
- Set explicitely derivatives for each keyframe - *teadious*
- Compute automatically plausible derivatives from surrounding samples - *often used*

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 $t \in [t_1, t_{N-2}]$

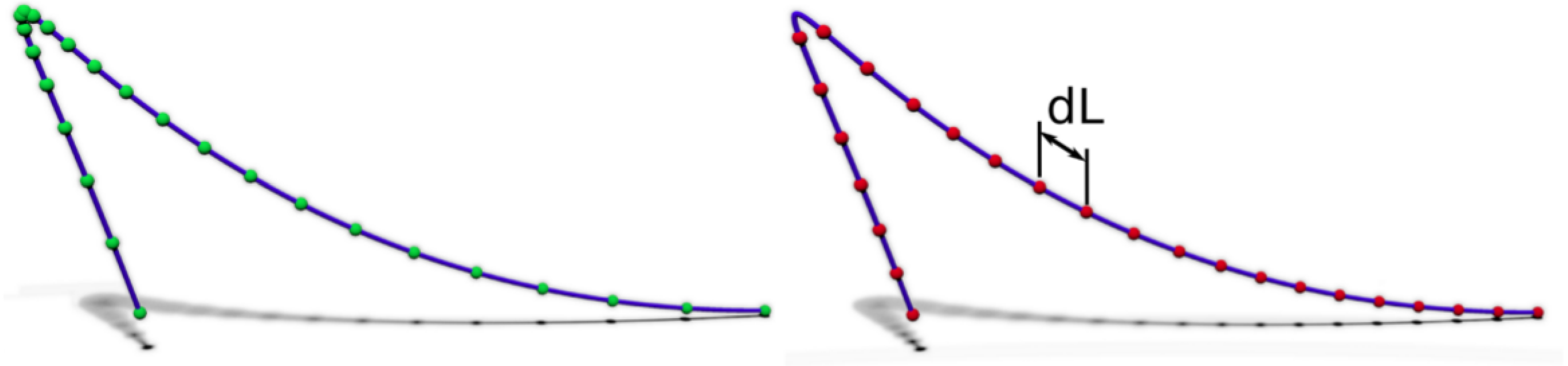
1. Find i such that $t \in [t_i, t_{i+1}]$
2. Compute $d_i = \mu \dfrac{p_{i+1} - p_{i-1}}{t_{i+1} - t_{i-1}}$, and $d_{i+1} = \mu \dfrac{p_{i+2} - p_i}{t_{i+2} - t_i}$
3. Compute $p(t) = (2s^3 - 3s^2 + 1)\, p_i + (s^3 - 2s^2 + s)\, d_i + (-2s^3 + 3s^2)\, p_{i+1} + (s^3 - s^2)\, d_{i+1}$

   with $s = \dfrac{t - t_i}{t_{i+1} - t_i}$.

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*

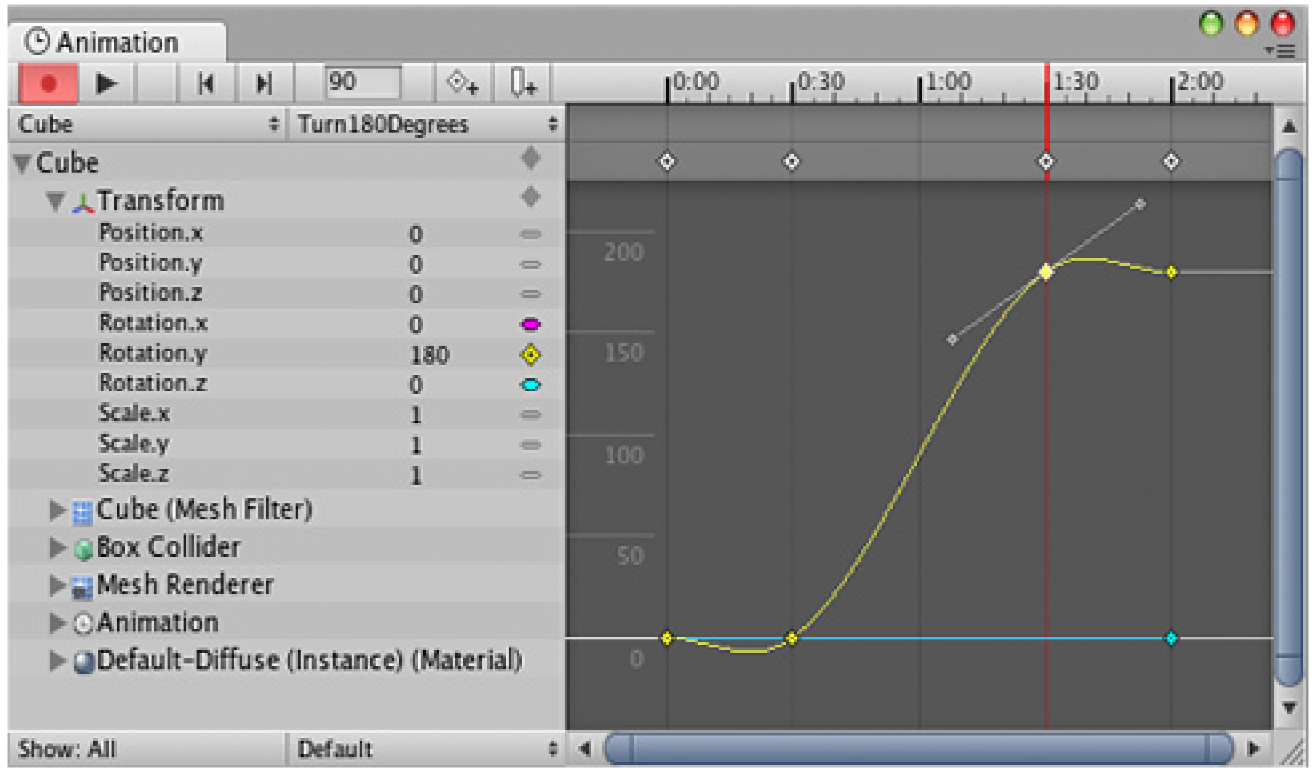


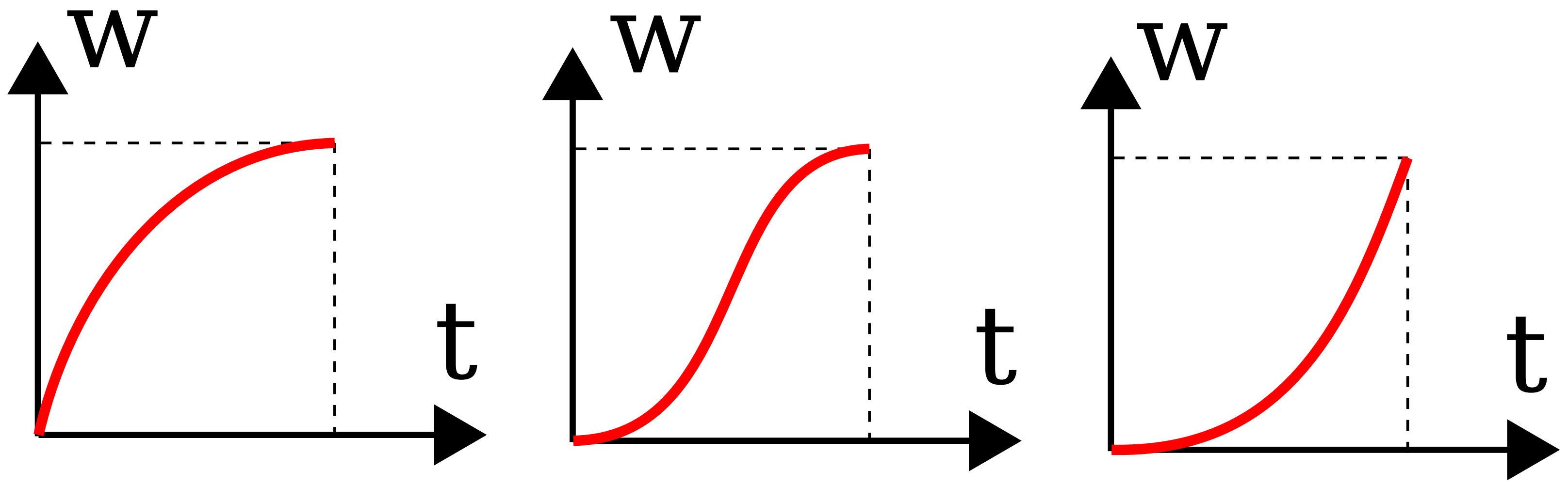
Curve editing

- 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 $p(t) = f(w(t))$





Usage of keyframes interpolation

Interpolate every vertex of multiple meshes

Multi-target blending

Interpolate between multiple key poses

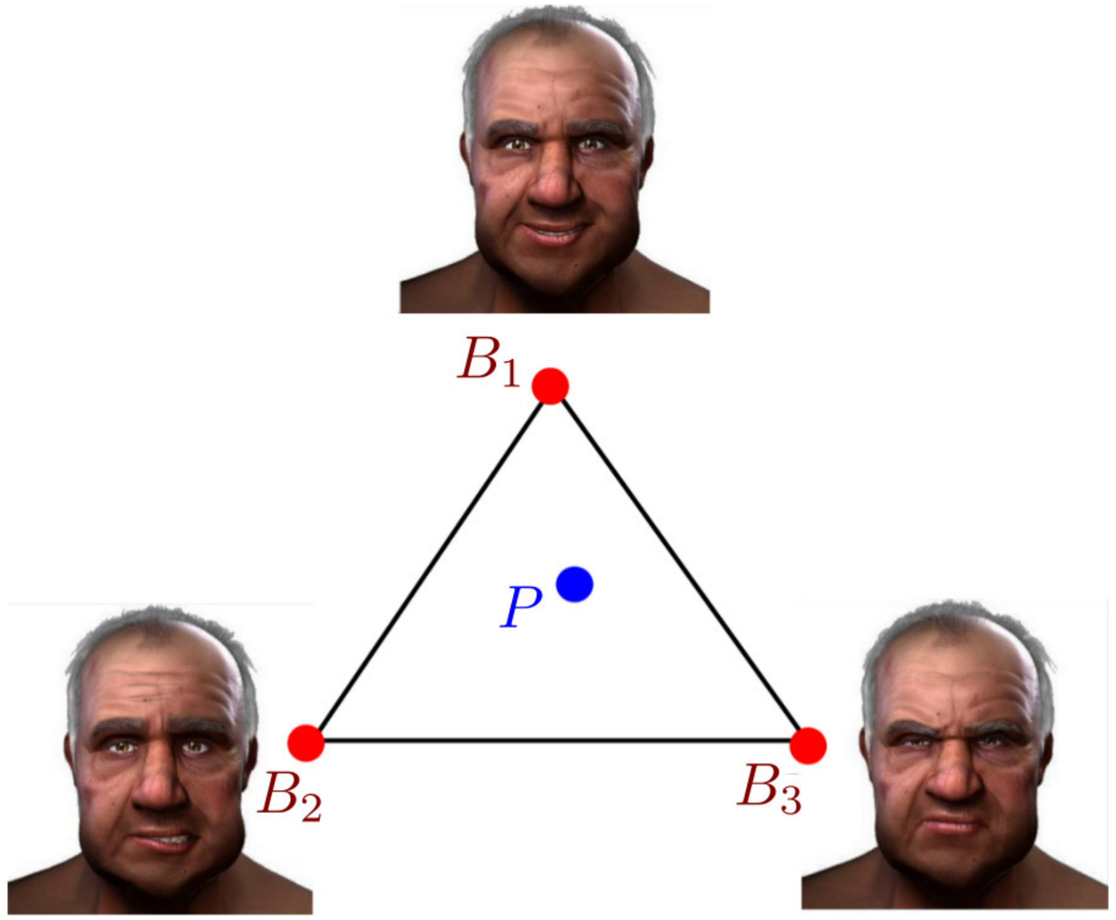
Interesting for facial animation

ex. 30% happy, 50% suprised, 20% tired

- Per-vertex formulation $p_i(t) = \sum_{k}^{N_{poses}} \omega_k(t)\, b_{ik}$
  - p_i: ith deformed vertex coordinates
  - b_{ik}: Initial ith vertex coordinates at pose k
  - w_k: Interpolation weight for pose k.
- Matrix formulation $\mathbf{p}(t) = \mathrm{B}\,\omega(t)$

Sometimes easier to control in adding "change of expression" instead of interpolating.

ex. Neutral + 20% happy smile + 30% surprised eyes



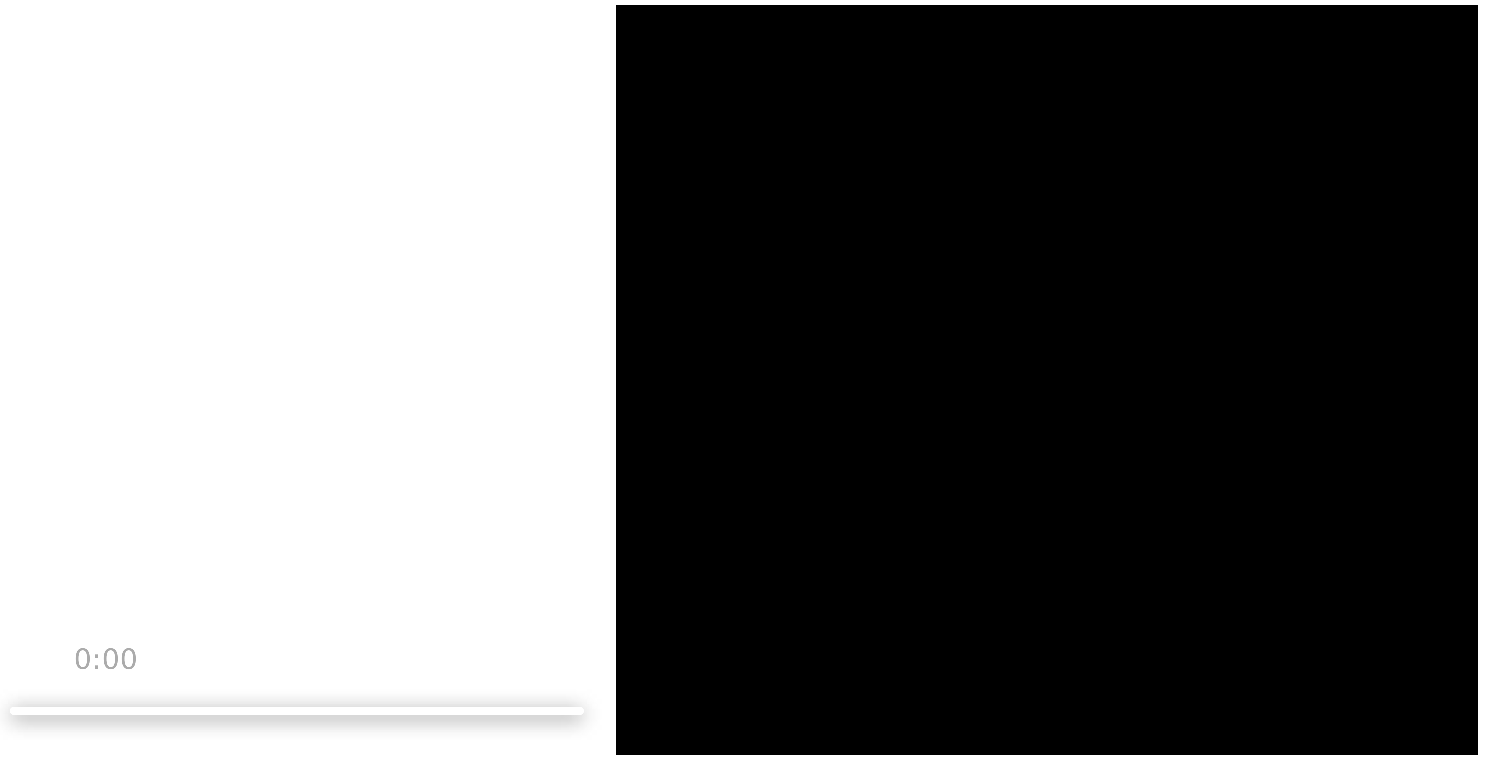
Blend shapes

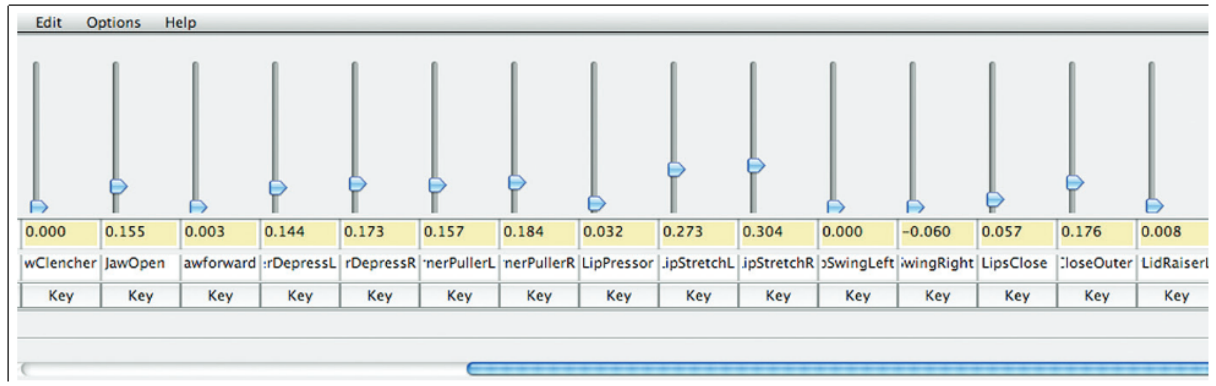
Represent deformation as a difference of coordinates with respect to neutral pose

- Per-vertex formulation

$$p_i(t) = b_i^0 + \sum_k^{N_{poses}} \omega_k(t)\, \underbrace{(b_{ki} - b_i^0)}_{d_{ki}}$$

- Matrix formulation $\mathbf{p}(t) = \mathbf{b^0} + \mathrm{D}\,\omega(t)$





Can add/substract local expressions (even extrapolate)

as long as they are not interfering/redundant

Allows expression transfert (keep same D for different faces b^0).

same connectivity required

Note:

- 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