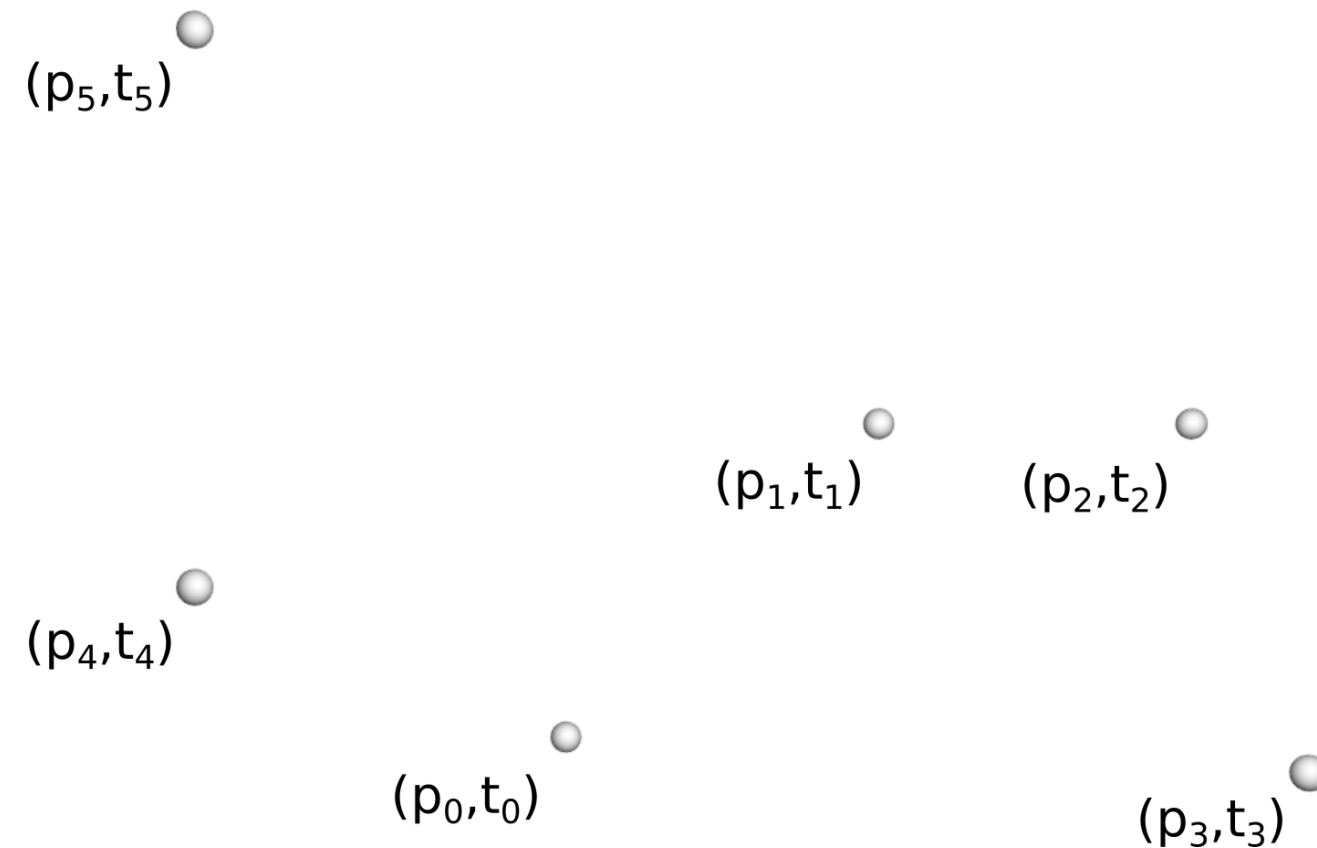


Interpolating positions

Spline trajectory

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 (p_i, p_{i+1})

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

Pros

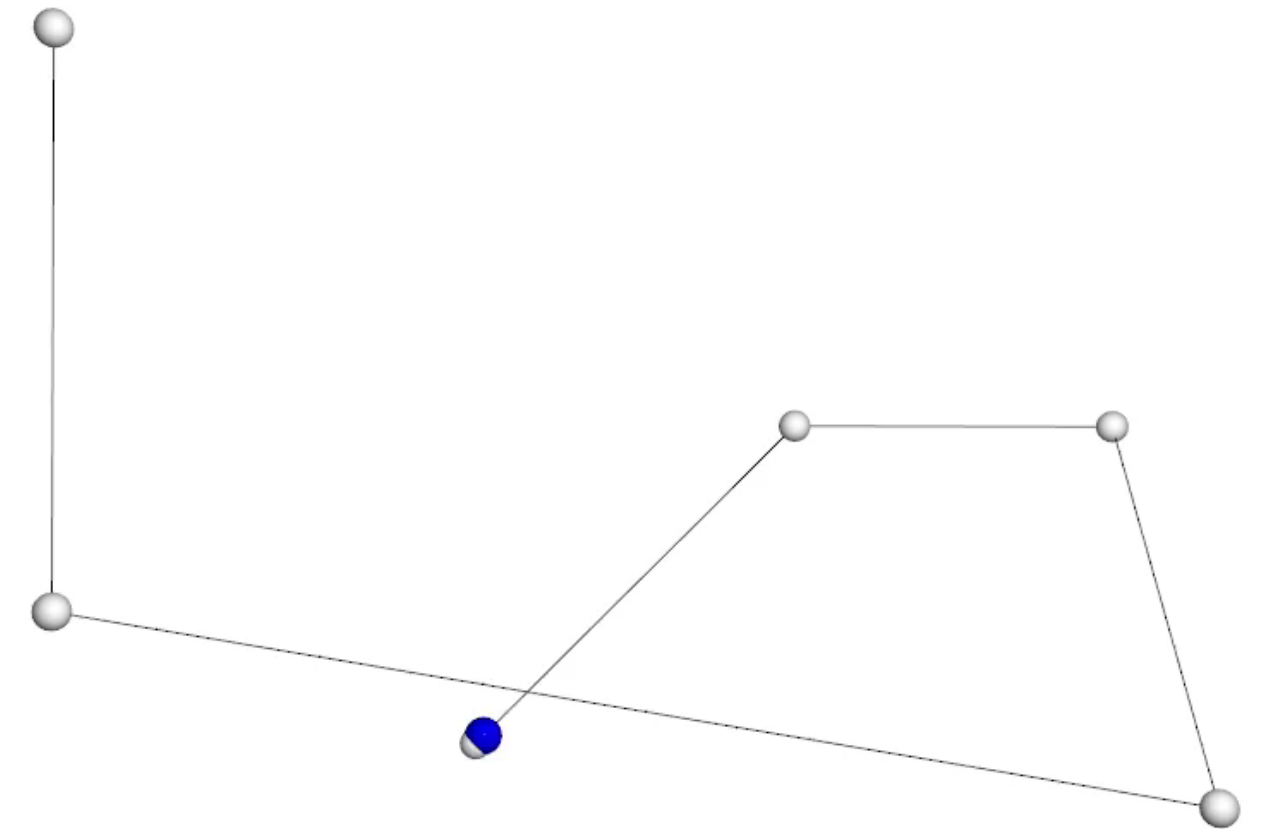
- Simple
- Constant speed between keyframes

Easy to adjust

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

$$\left\{ \begin{array}{l} p(t) = \sum_{i=0}^{N-1} \alpha_i(t) p_i \\ \alpha_i(t) = \sum_{j=0}^d c_{ij} t^j \end{array} \right.$$

(α_i) polynomial basis function of degree d

Which polynomials/degree choose ?

Lagrange polynomial interpolation

Naive idea: Interpolate all points at once

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

Degree of polynomial : $N - 1$

- Known solution: **Lagrange polynomial**

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

- 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

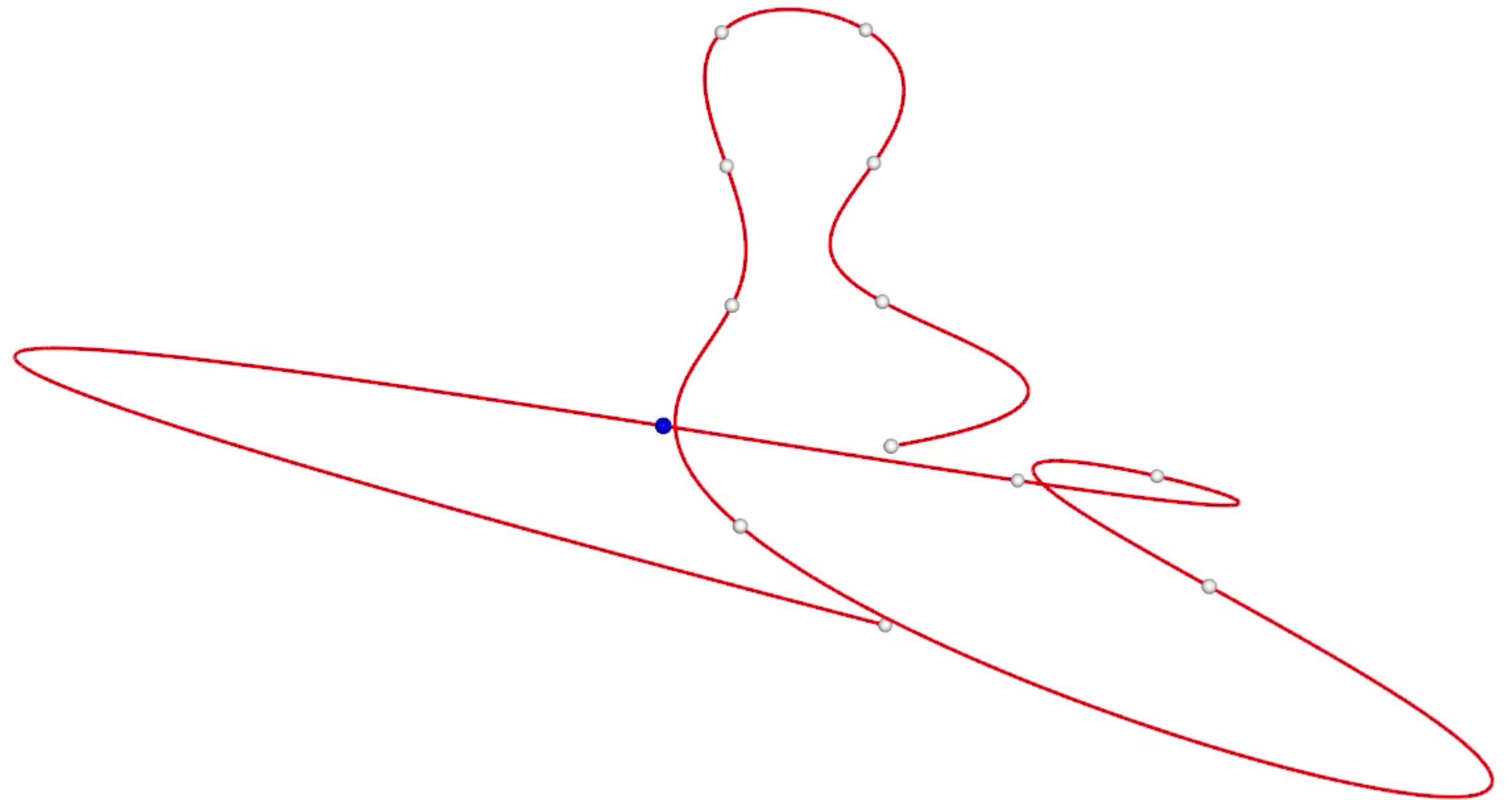
Lagrange polynomial interpolation : Comparison

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

10 samples



12 samples



Spline

Idea

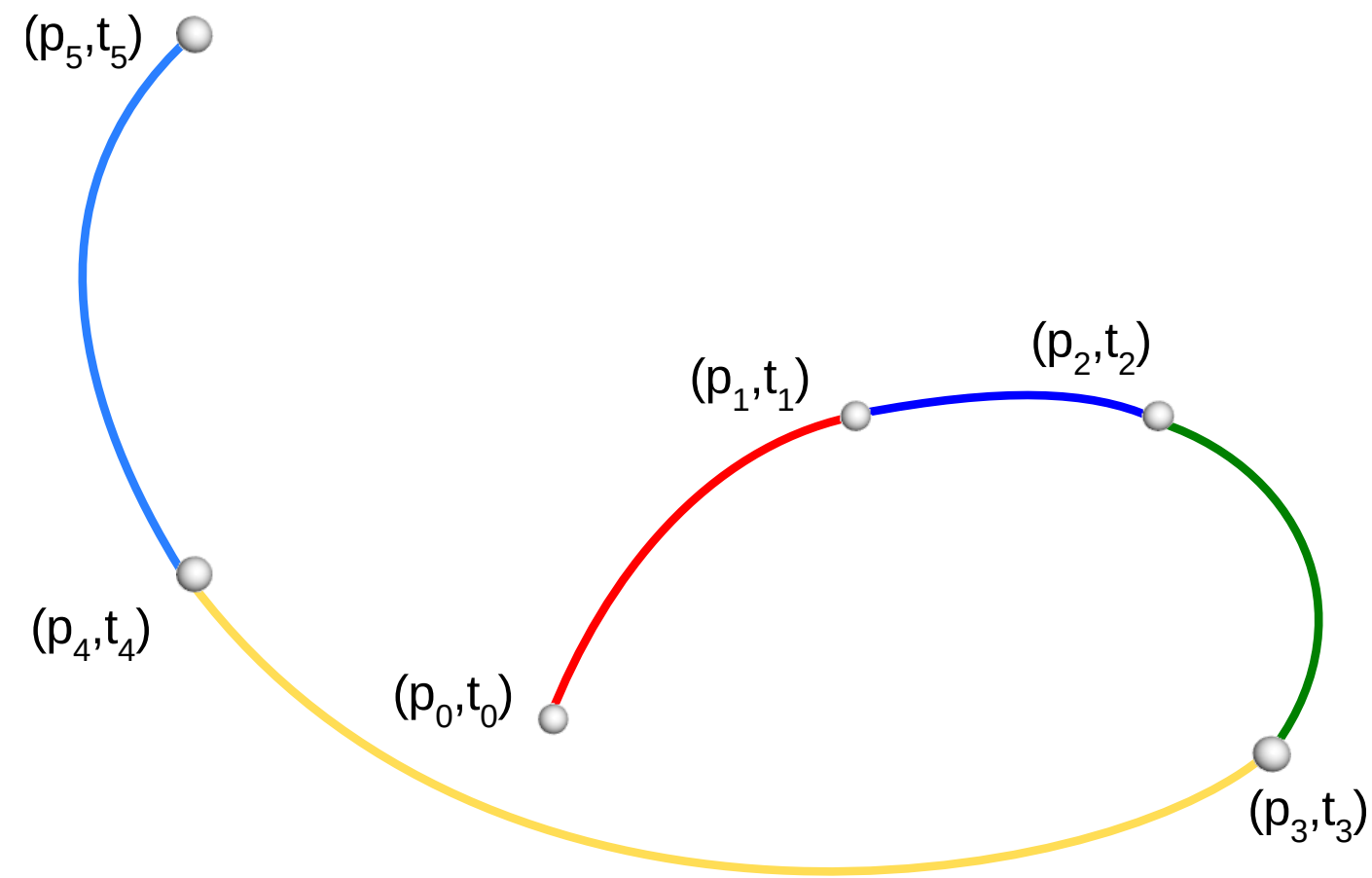
- Define on each part a polynomial
- Smooth junctions between them.

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

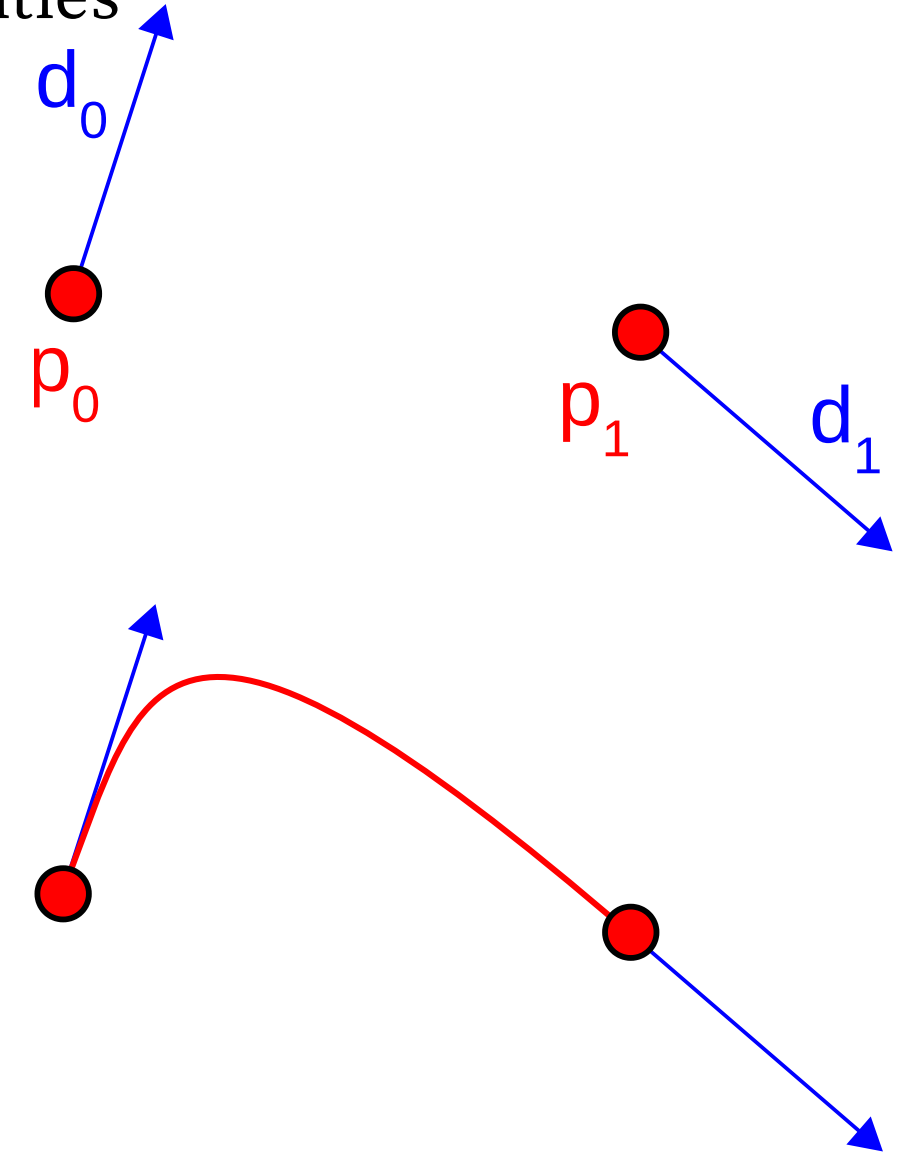
$$\begin{cases} p(s) = c_3 s^3 + c_2 s^2 + c_1 s + c_0 \\ p(0) = p_0, p(1) = p_1, p'(0) = d_0, p'(1) = d_1 \end{cases}$$

$\Rightarrow$ System of equations

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

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

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



Cardinal Spline

Our initial problem: set of multiple keyframes position+time

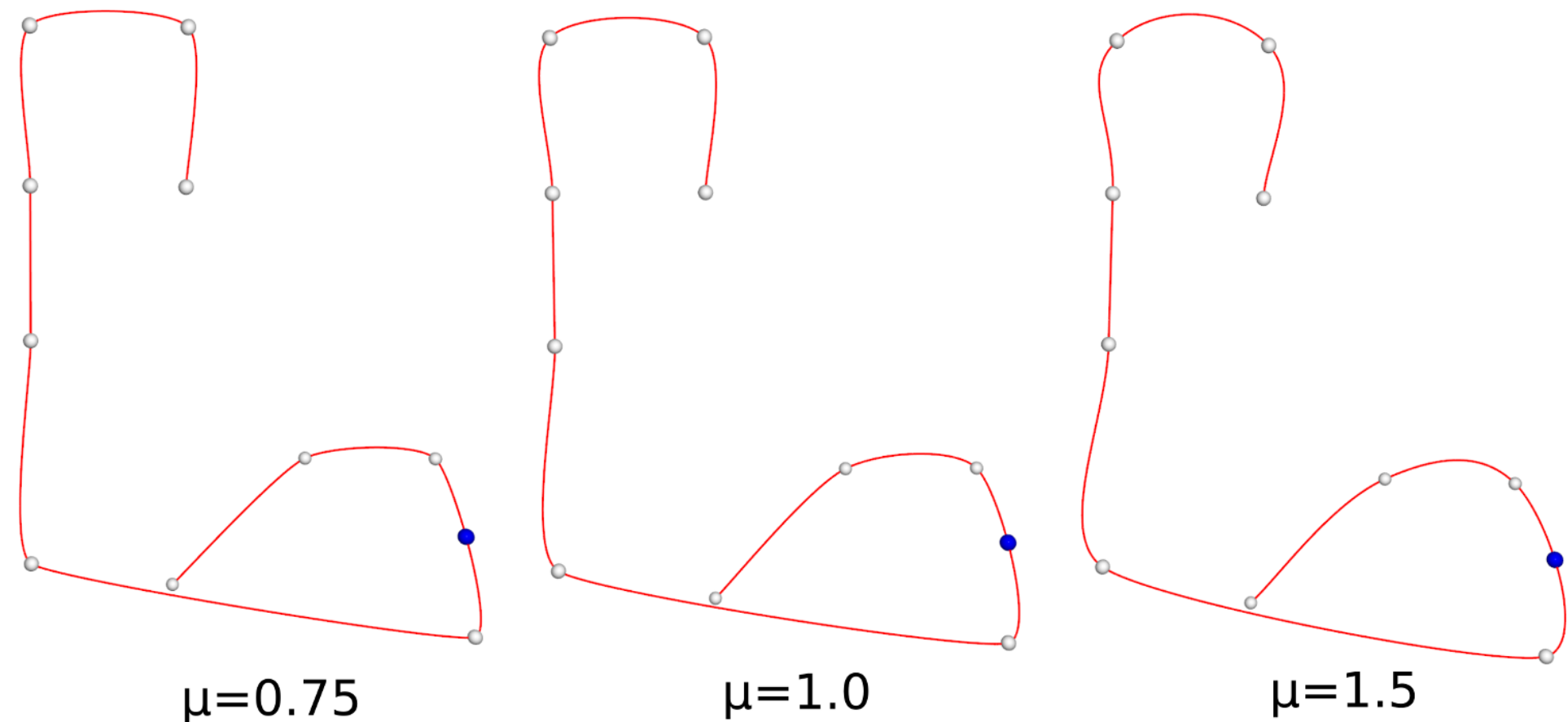
Two solutions

- **Hermite representation:** Set explicitly derivatives for each keyframe - can be *teadious*.
- **Cardinal Spline:** Compute automatically plausible derivatives from surrounding samples (finite differences)

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

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

Called: Catmull Rom Spline



Wrap-up Algorithm

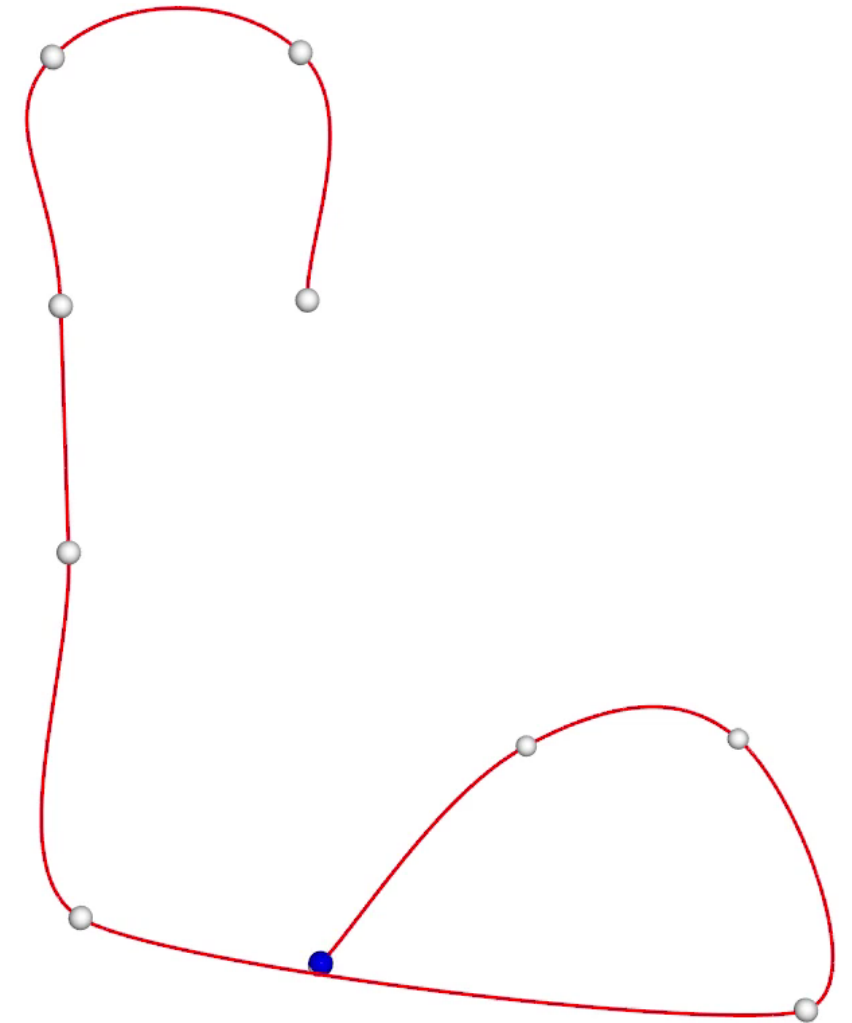
Compute $p(t)$ as a cubic spline interpolation

- 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 $t \in [t_i, t_{i+1}]$

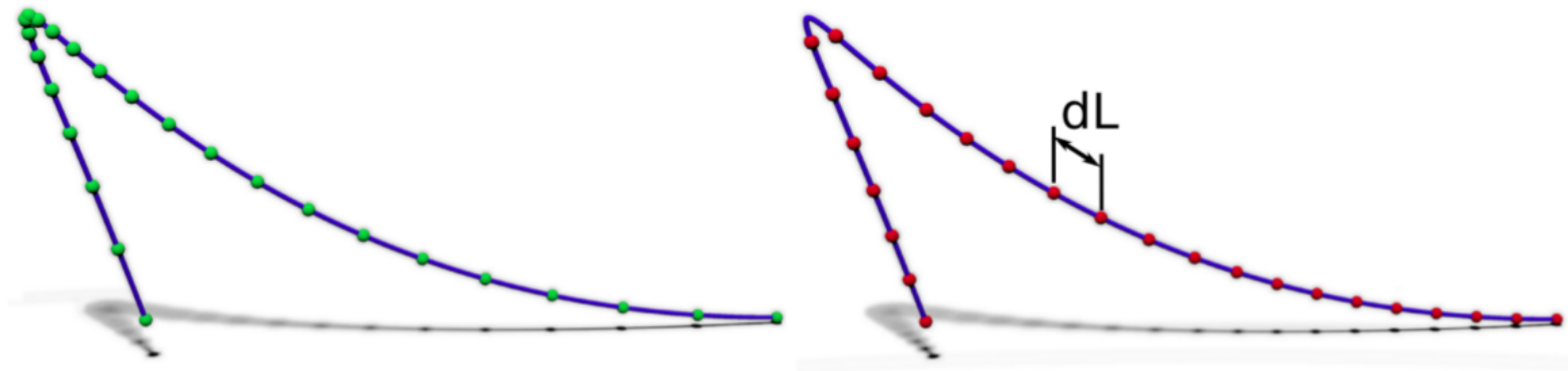
2. Compute $d_i = \mu \frac{p_{i+1} - p_{i-1}}{t_{i+1} - t_{i-1}}$, and $d_{i+1} = \mu \frac{p_{i+2} - p_i}{t_{i+2} - t_i}$

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



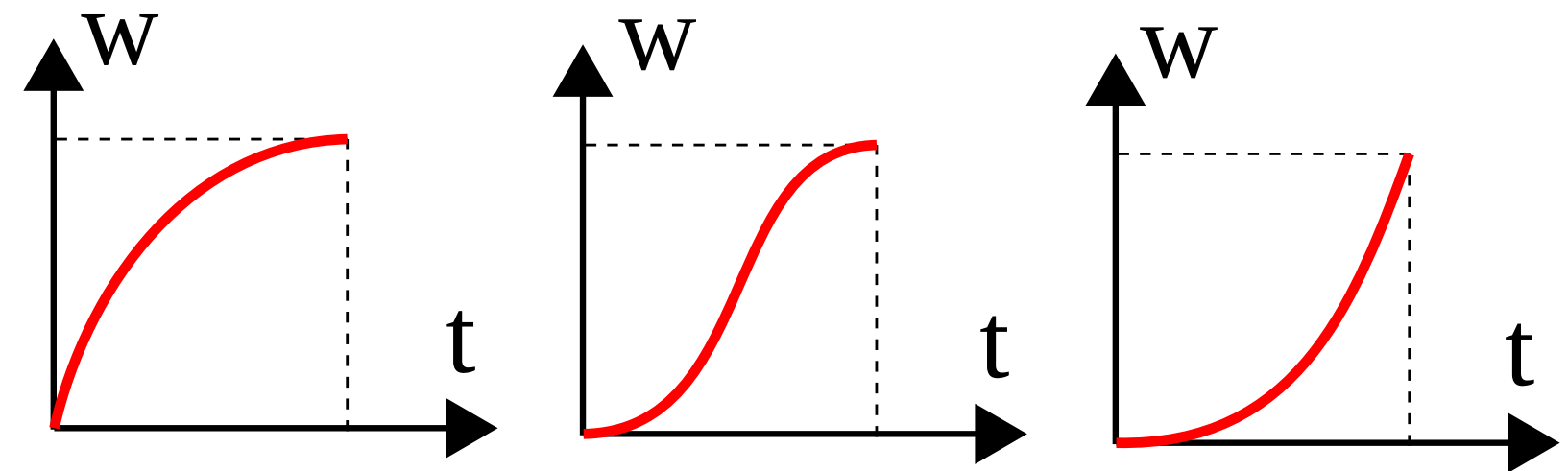
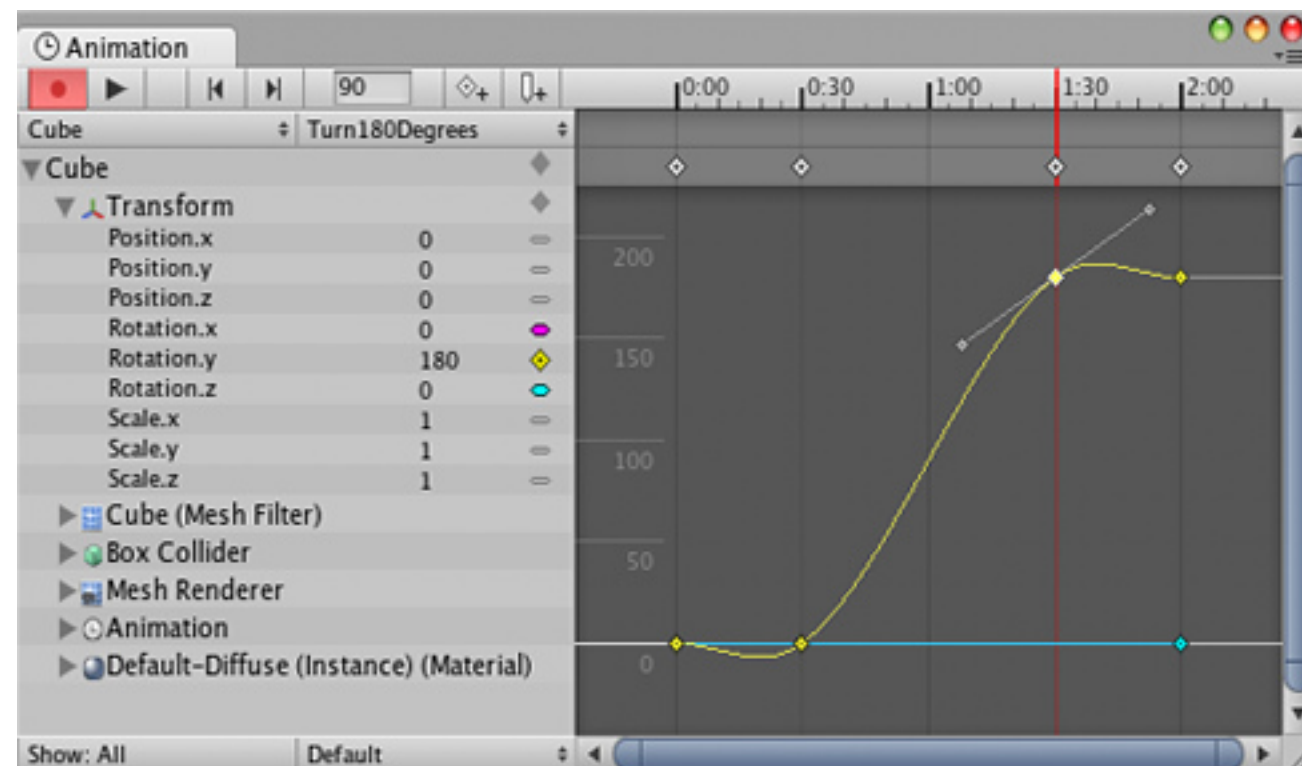
Limitation of cubic curve interpolation

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



Interpolating position

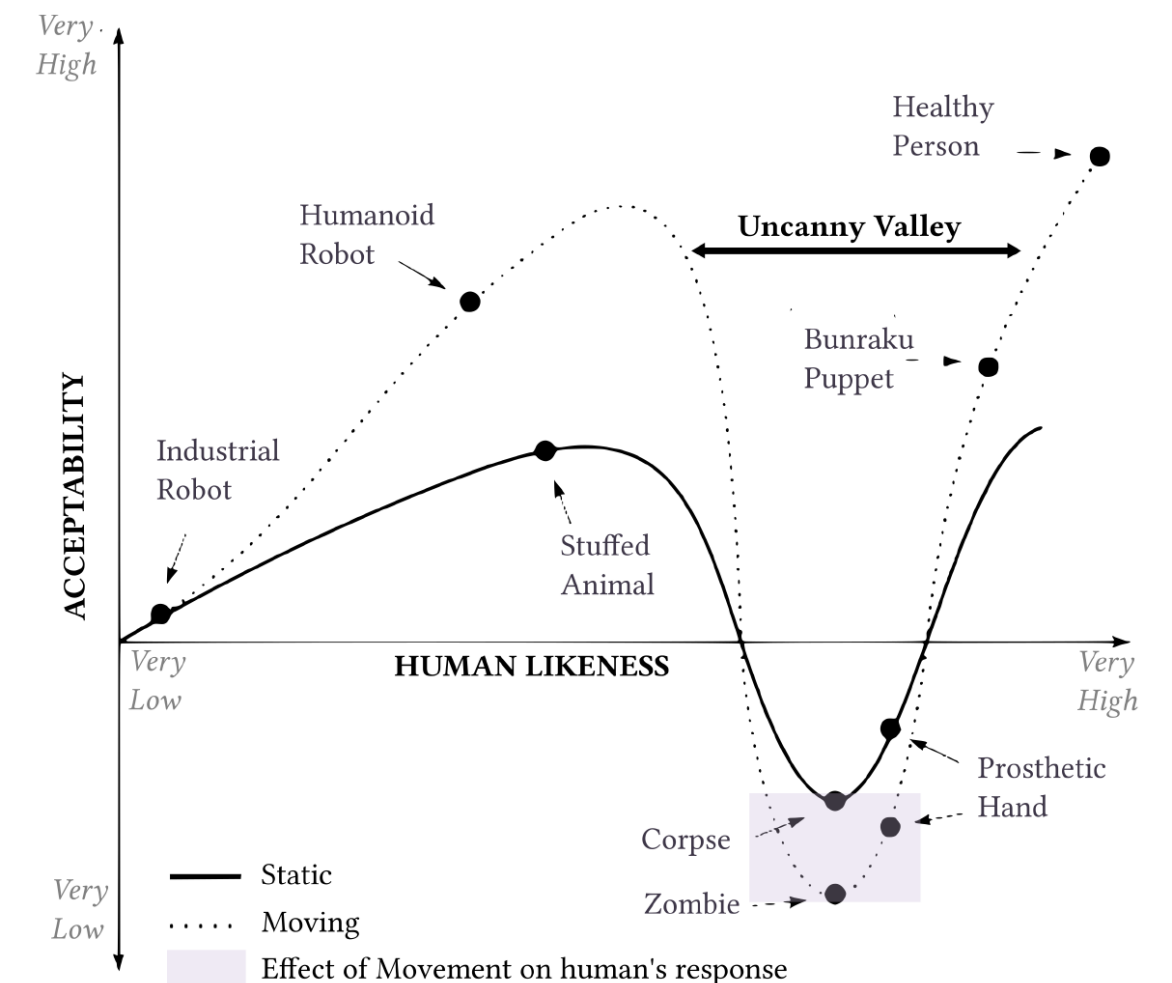
Facial Animation: Blend Shapes

Facial animation

Facial expression are particularly hard to automatically model

- Lots of details (large number of muscles)
- Human perception is very sensible on face

Risk of *Uncanny Valley*



Facial expression are categorized



Usage of keyframes interpolation

Interpolate every vertex between multiple meshes (key-poses)



Pose b_1

Interpolation $p(t)$

Pose b_2

$$p_i(t) = \omega_1(t)b_{i1} + \omega_2(t)b_{i2}$$

Blend Shapes: Multi-target blending

Interpolate between multiple key poses

Interesting for facial animation

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

- Per-vertex formulation $p_i(t) = \sum_k^{N_{poses}} \omega_k(t) b_{ik}$

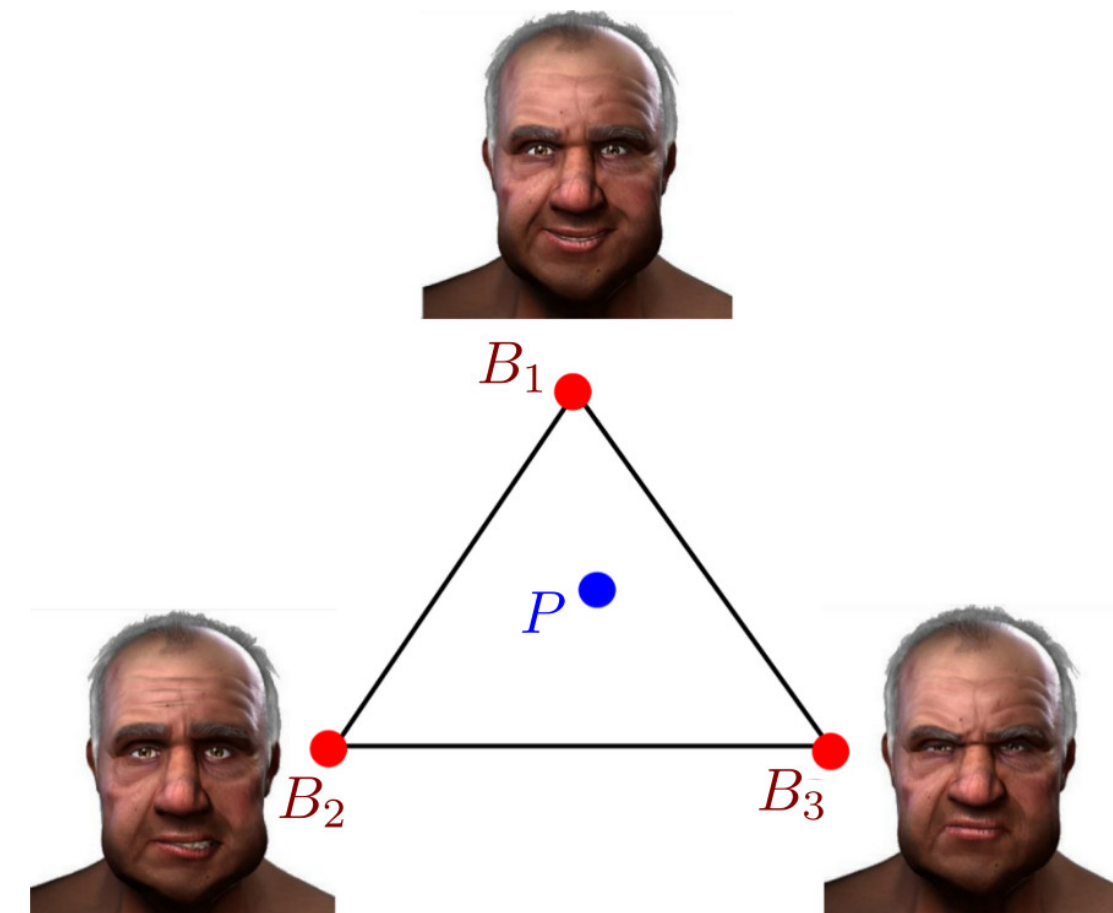
- p_i : i th deformed vertex coordinates

- b_{ik} : Initial i th vertex coordinates at pose k

- w_k : Interpolation weight for pose k .

- Matrix formulation $\mathbf{p}(t) = \mathbf{B} \omega(t)$

$$\begin{pmatrix} p_1(t) \\ p_2(t) \\ \vdots \\ p_N(t) \end{pmatrix} = \underbrace{\begin{pmatrix} b_{11} & \dots & b_{1N_{poses}} \\ b_{21} & \dots & b_{2N_{poses}} \\ \vdots & & \vdots \\ b_{N1} & \dots & b_{NN_{poses}} \end{pmatrix}}_{\text{all vertices from all poses}} \underbrace{\begin{pmatrix} \omega_1(t) \\ \vdots \\ \omega_{N_{poses}}(t) \end{pmatrix}}_{\text{weights per pose}}$$



Coordinate differences

Often simpler to add/remove some expression w/r neutral pose

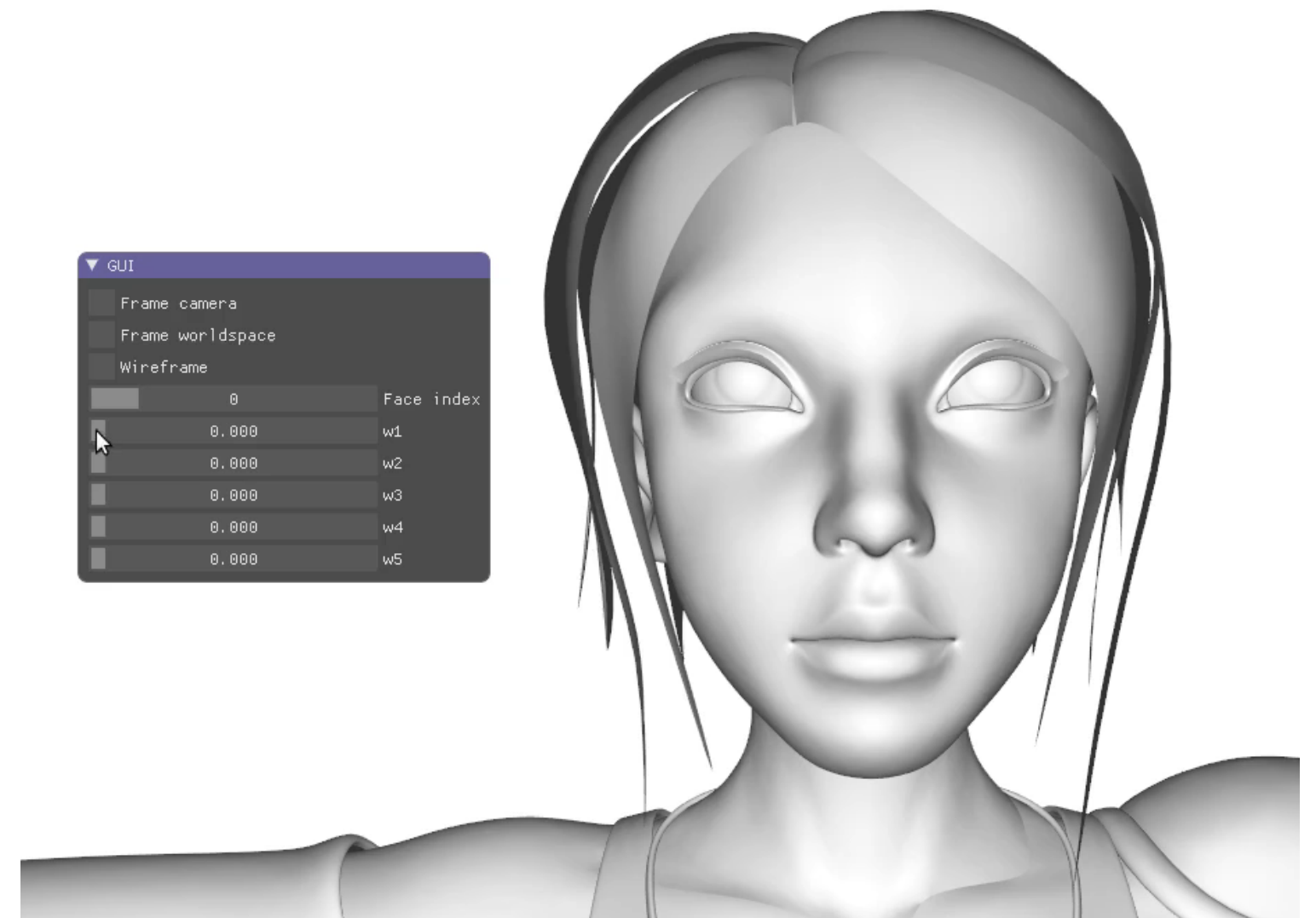
Add smile, open more the left eye, etc.

Idea:

- Set b^0 to be neutral pose
- Precompute *poses differences* $d_k = b_k - b^0$
- Compute $p_i(t) = b_i^0 + \sum_k^{N_{poses}} \omega_k(t) d_{ik}$

In matrix formulation $\mathbf{p}(t) = \mathbf{b}^0 + \mathbf{D} \omega(t)$

D : matrix of vertex differences between poses.



Blend shapes - Summary

Standard for facial/complex shapes deformation

(+) Allow to add/subtract local expressions from pre-set poses
even extrapolate

(+) Allow expression transfert
Keep same D for difference faces b^0
Requires same connectivity

Requires non redundant/interfering poses

D is not an othogonal matrix

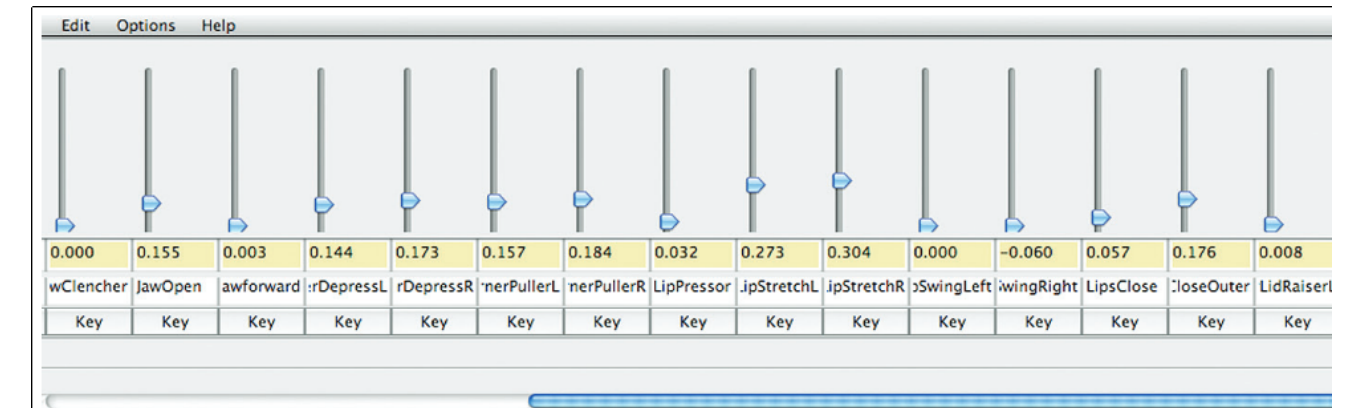
Different weights can give the same result

PCA can orthogonalize the matrix

but weights are less intuitive (not positive)

To go beyond:

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



0:00 / 1:04

