Computer Animation Algorithms and Techniques

Interpolating Values

Animation

Animator specified
- **interpolation**
- **key frame**

Algorithmically controlled
- **Physics-based**
- **Behavioral**

Data-driven
- **motion capture**

Motivation

<u>Common problem:</u> given a set of points
Smoothly (in time and space) move an object through the set of points

<u>Example additional temporal constraints:</u>
From zero velocity at first point, smoothly accelerate until time t1, hold a constant velocity until time t2, then smoothly decelerate to a stop at the last point at time t3

Motivation – solution steps

1. Construct a space curve that interpolates the given points with piecewise first order continuity

$p=P(u)$

2. Construct an arc-length-parametric-value function for the curve

$u=U(s)$

3. Construct time-arc-length function according to given constraints

$s=S(t)$

$$p=P(U(S(t)))$$

Interpolating function

Interpolation v. approximation

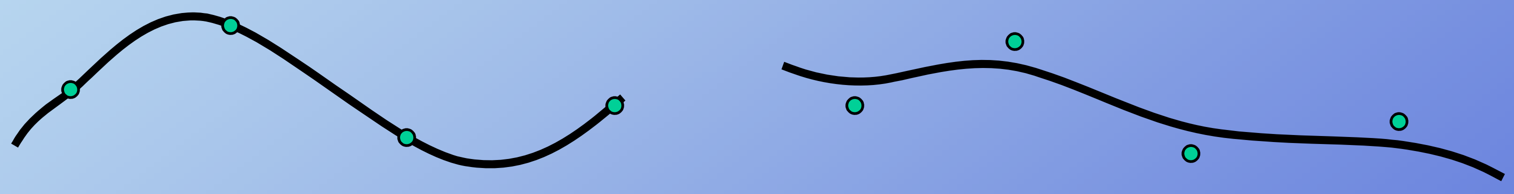
Complexity: cubic

Continuity: first degree (tangential)

Local v. global control: local

Information requirements: tangents needed?

Interpolation v. Approximation

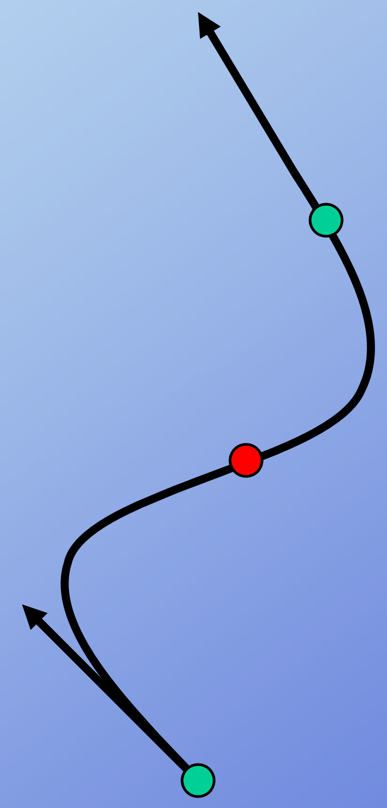


Complexity

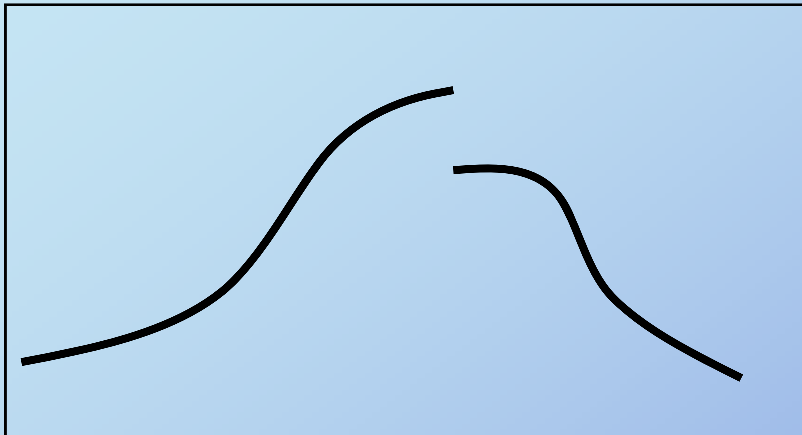
Low complexity
reduced computational cost

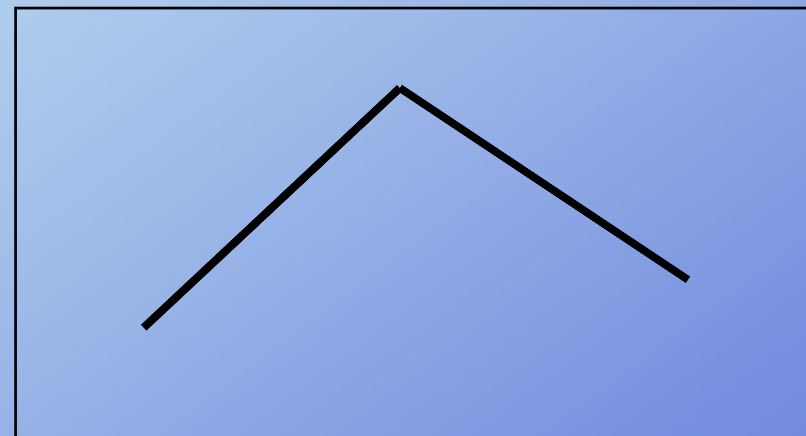
Point of Inflection
Can match arbitrary tangents at end points

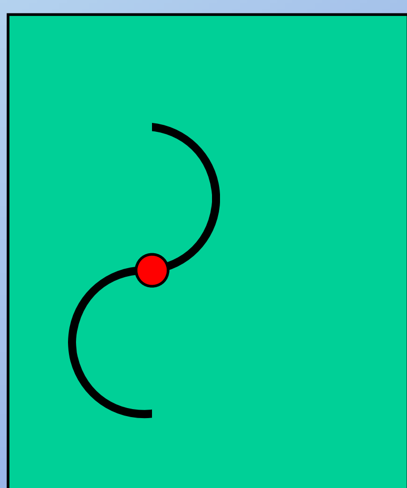
CUBIC polynomial

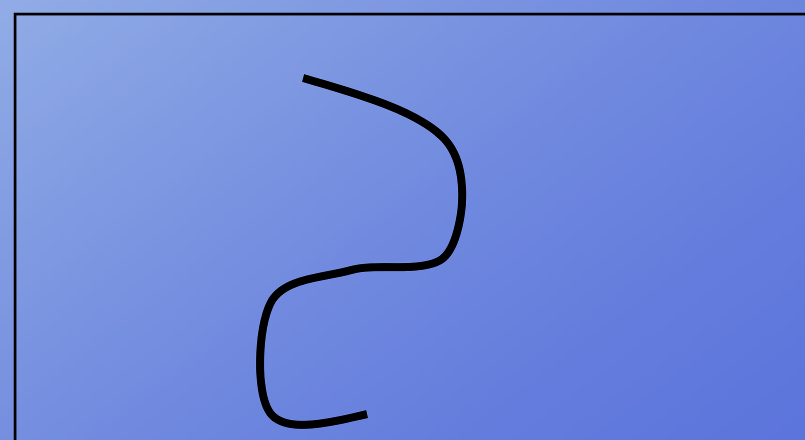


Continuity

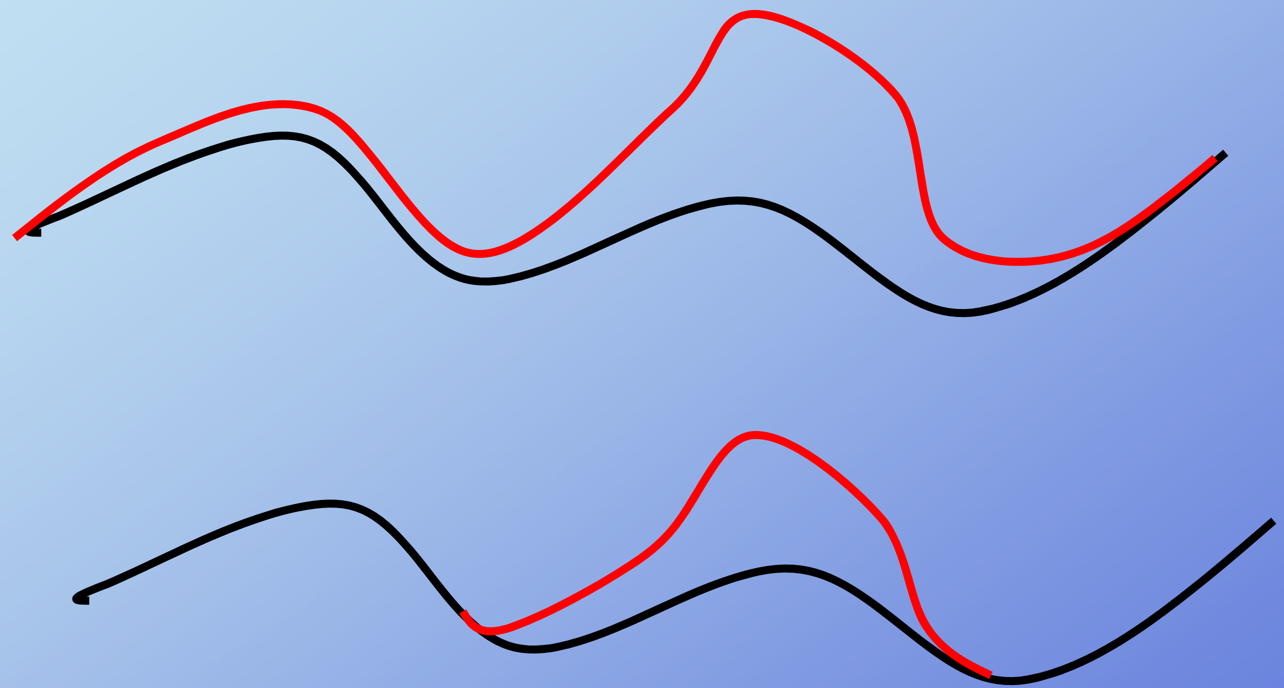








Local v. Global Control



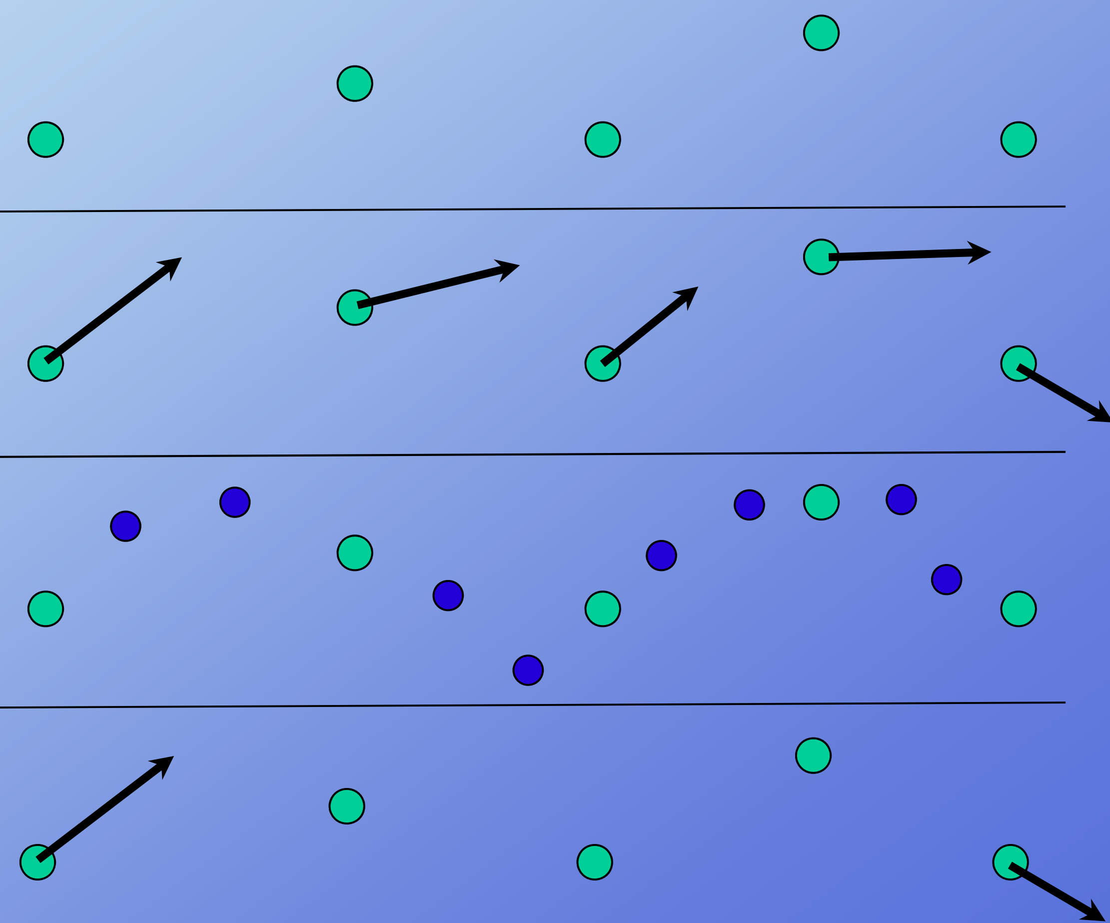
Information requirements

just the points

tangents

interior control points

just beginning and
ending tangents



Curve Formulations

Lagrange Polynomial

<u>Piecewise cubic polynomials</u>

Hermite

Catmull-Rom

Blended Parabolas

Bezier

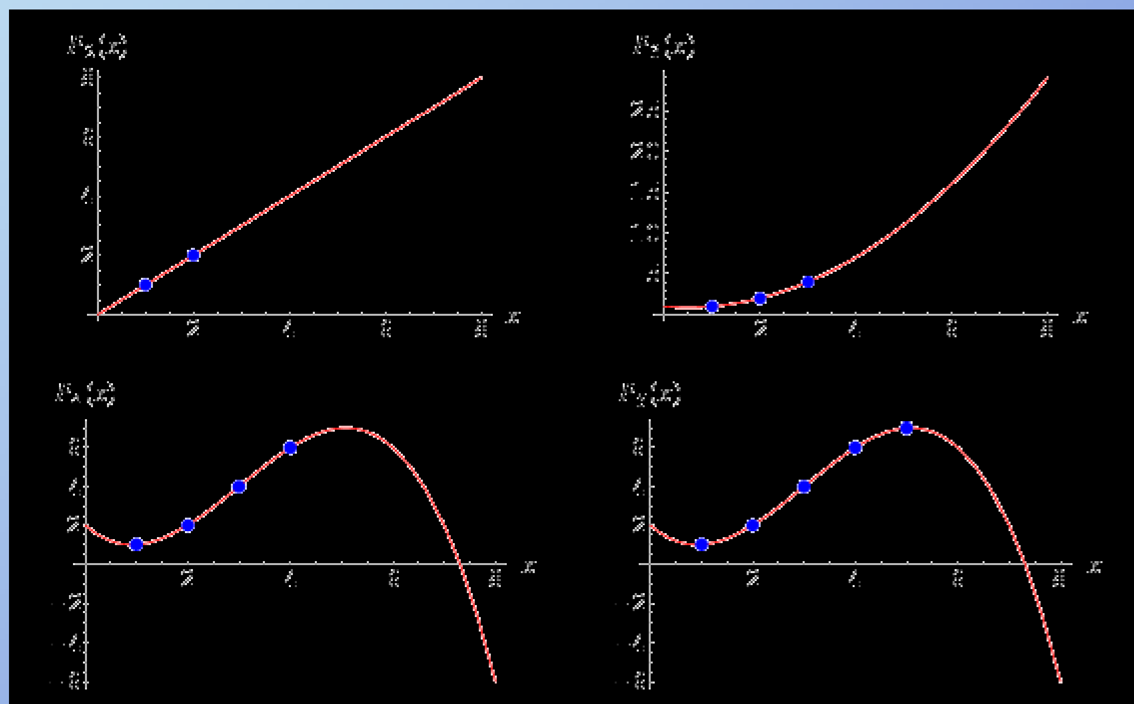
B-spline

Tension-Continuity-Bias

4-Point Form

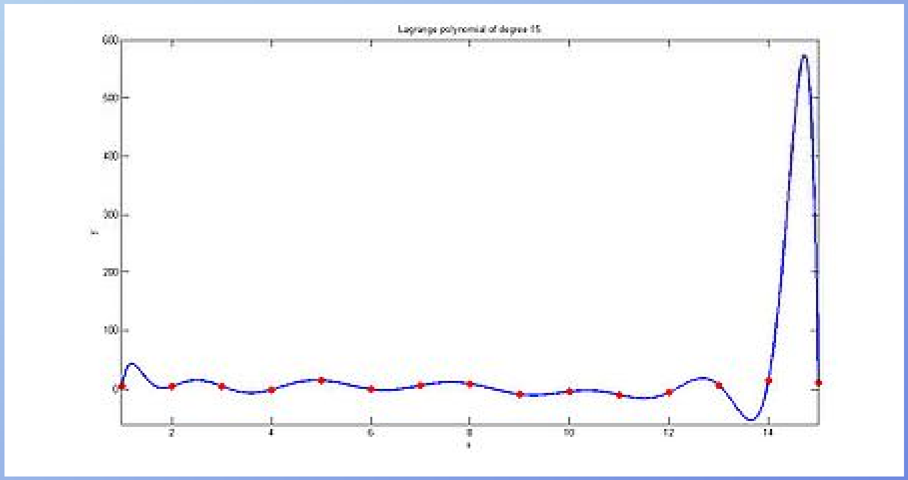
Lagrange Polynomial

$$P_j(x) = y_j \prod_{\substack{k=1 \\ k \neq j}}^{x} \frac{x - x_k}{x_j - x_k}$$



Lagrange Polynomial

$$P_j(x) = y_j \prod_{\substack{k=1 \\ k \neq j}}^{x} \frac{x - x_k}{x_j - x_k}$$



Polynomial Curve Formulations

Need to match real-world data v. design from scratch
Information requirements: just points? tangents?
Qualities of final curve?
Intuitive enough?
Other shape controls?

Blending Functions

$$P = U^T M B = F B = U^T A$$

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

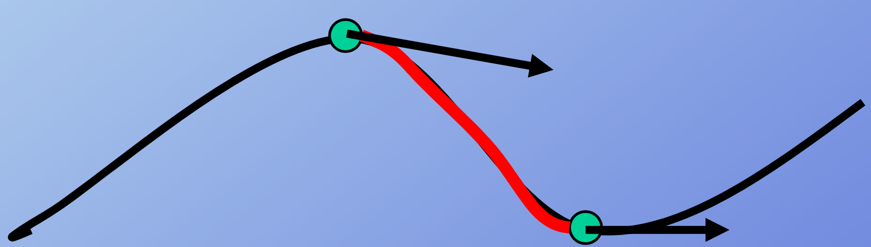
Coefficient matrix

Geometric information

Algebraic coefficient matrix

Hermite

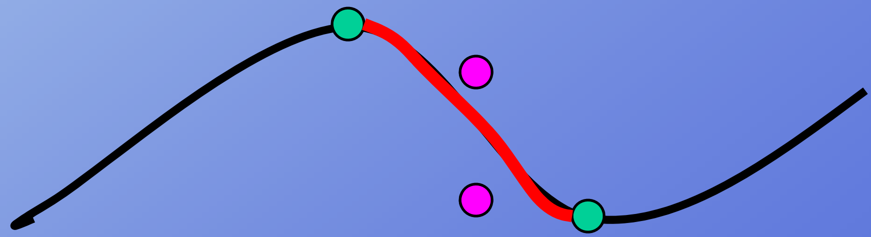
$$P = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 2.0 & -2.0 & 1.0 & 1.0 \\ -3.0 & 3.0 & -2.0 & -1.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \begin{bmatrix} p_i \\ p_{i+1} \\ p_i' \\ p_{i+1}' \end{bmatrix}$$



Cubic Bezier

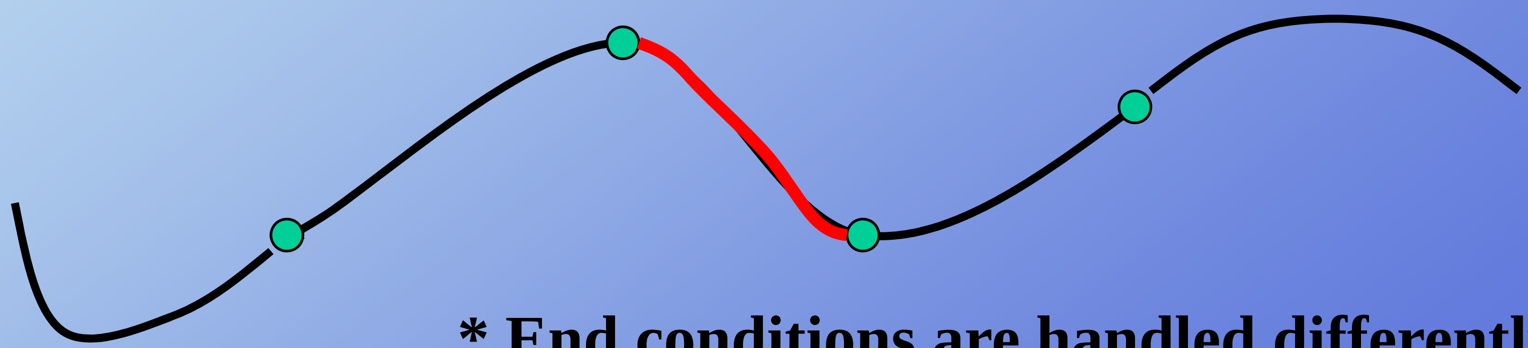
$$P = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 1.0 & 3.0 & -3.0 & 1.0 \\ 3.0 & -6.0 & 3.0 & 0.0 \\ -3.0 & 3.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \begin{bmatrix} p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \end{bmatrix}$$

Interior control points play the same role as the tangents of the Hermite formulation



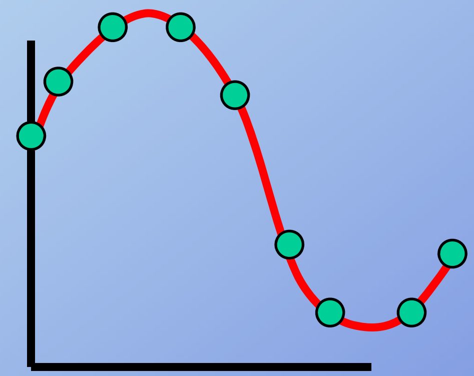
Blended Parabolas/Catmull-Rom*

$$P = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1 & -2.5 & 2 & -0.5 \\ -0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \end{bmatrix}$$



*** End conditions are handled differently**

Controlling Motion along p=P(u)



Step 2. Reparameterization by arc length

u = U(s)   where s is distance along the curve

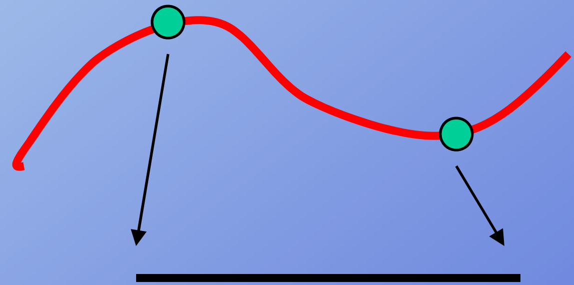
Step 3. Speed control

for example, ease-in / ease-out

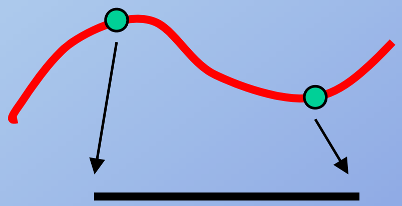
s = ease(t)   where t is time

Reparameterizing by Arc Length

Analytic
Forward differencing
    Supersampling
    Adaptive approach
Numerically
    Adaptive Gaussian



Reparameterizing by Arc Length - analytic



$$P(u) = au^3 + bu^2 + cu + d$$

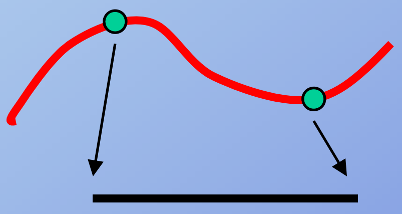
$$s = \int_{u_1}^{u_2} \left| dP/du \right| \quad du$$

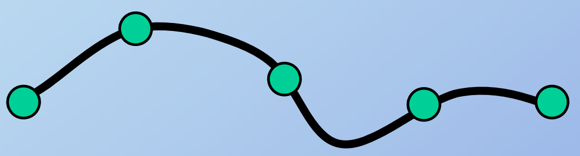
$$dP/du = \left(dx(u)/du \quad dy(u)/du \quad dz(u)/du \right)$$

$$\left| dP/du \right| = \sqrt{\left(dx(u)/du\right)^2 + \left(dy(u)/du\right)^2 + \left(dx(u)/du\right)^2}$$

Can’t always be solved analytically for our curves

Reparameterizing by Arc Length - supersample



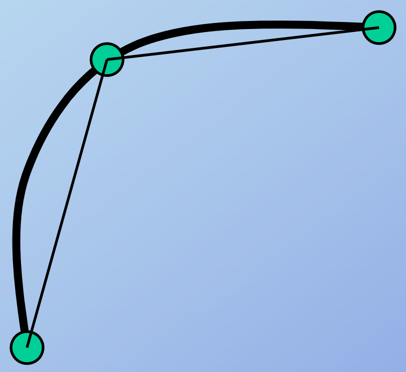


1. Calculate a bunch of points at small increments in u
2. Compute summed linear distances as approximation to arc length
3. Build table of (parametric value, arc length) pairs

Notes

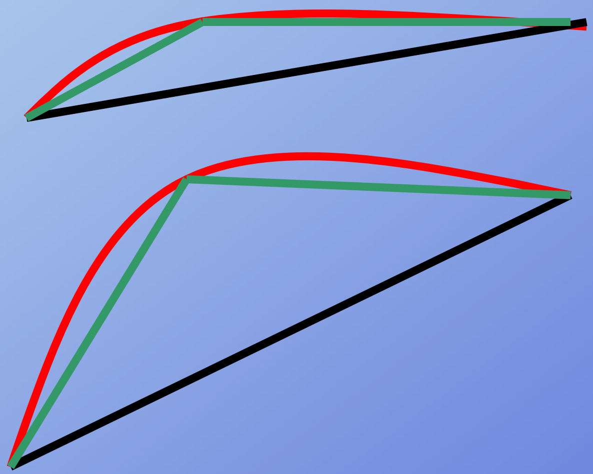
1. Often useful to normalize total distance to 1.0
2. Often useful to normalize parametric value for multi-segment curve to 1.0

Build table of approx. lengths

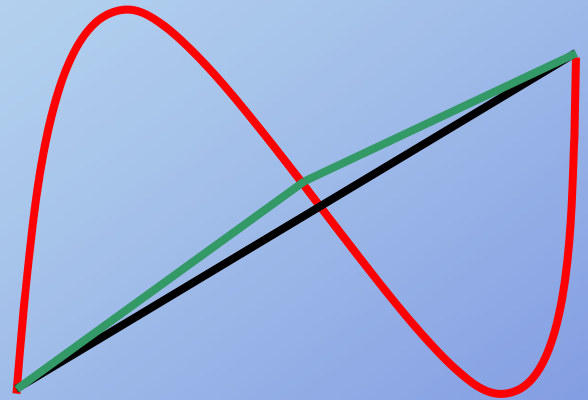


index	u	Arc Length
0	0.00	0.000
1	0.05	0.080
2	0.10	0.150
3	0.15	0.230
...	...	...
20	1.00	1.000

Adaptive Approach
How fine to sample?

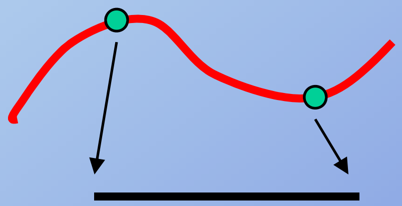


Compare successive approximations and see if they agree within some tolerance



Test can fail – subdivide to predefined level, then start testing

Reparameterizing by Arc Length - quadrature



$$\int_{-1}^{+1} f(u)du = \sum_i w_i f(u_i)$$

$$P(u) = au^3 + bu^2 + cu + d$$

$$\int_{-1}^{+1} \sqrt{Au^4 + Bu^3 + Cu^2 + Du + E}$$

Lookup tables of weights and parametric values

Can also take adaptive approach here

Reparameterizing by Arc Length

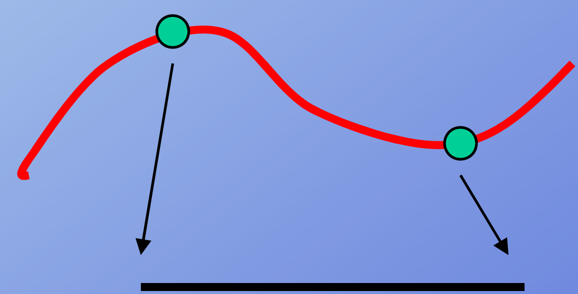
Analytic

Forward differencing

- Supersampling
- Adaptive approach

Numerically

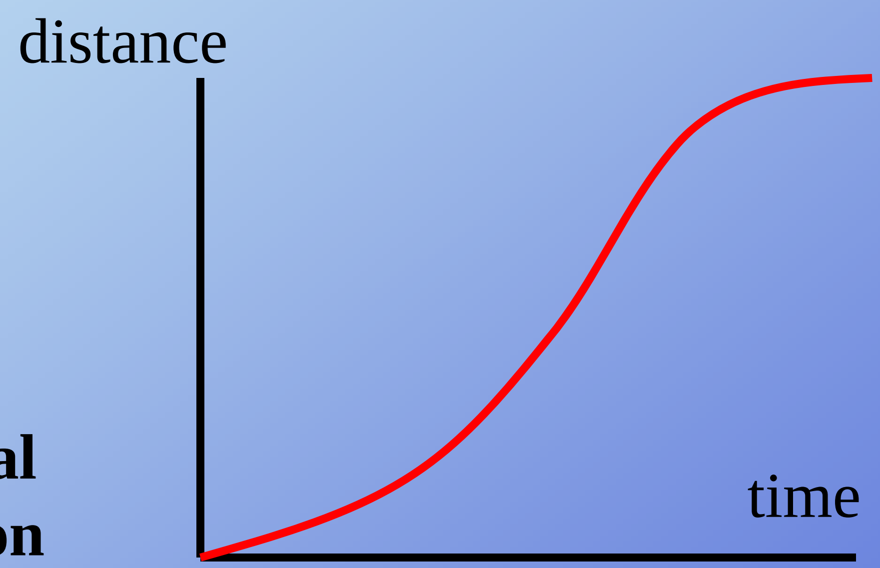
- Adaptive Gaussian



Sufficient for many problems

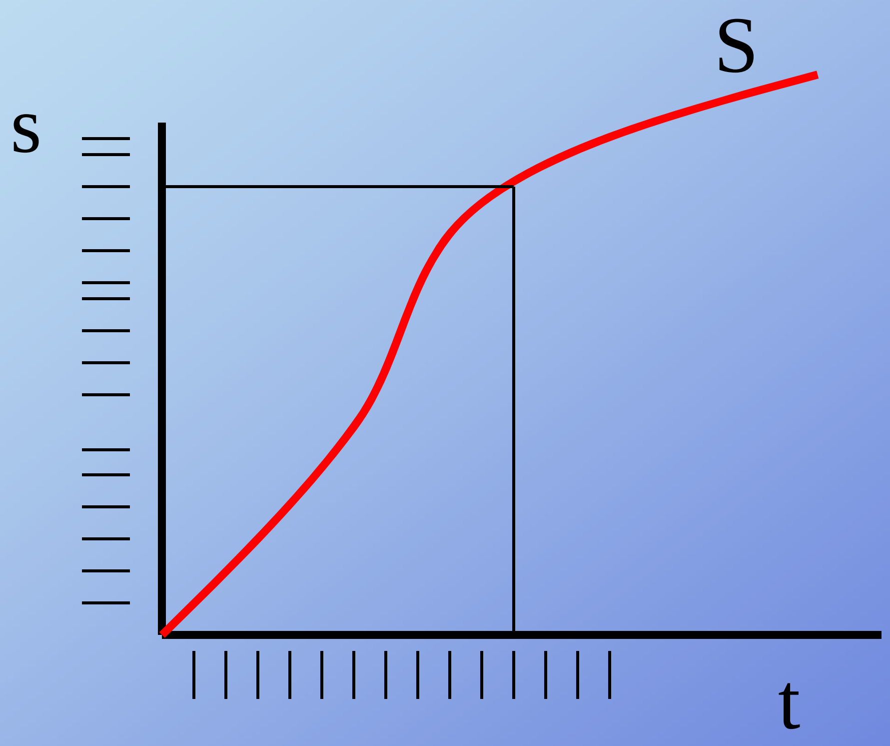
Speed Control

Time-distance function
- Ease-in
  - Cubic polynomial
  - Sinusoidal segment
  - Segmented sinusoidal
  - Constant acceleration
- General distance-time functions



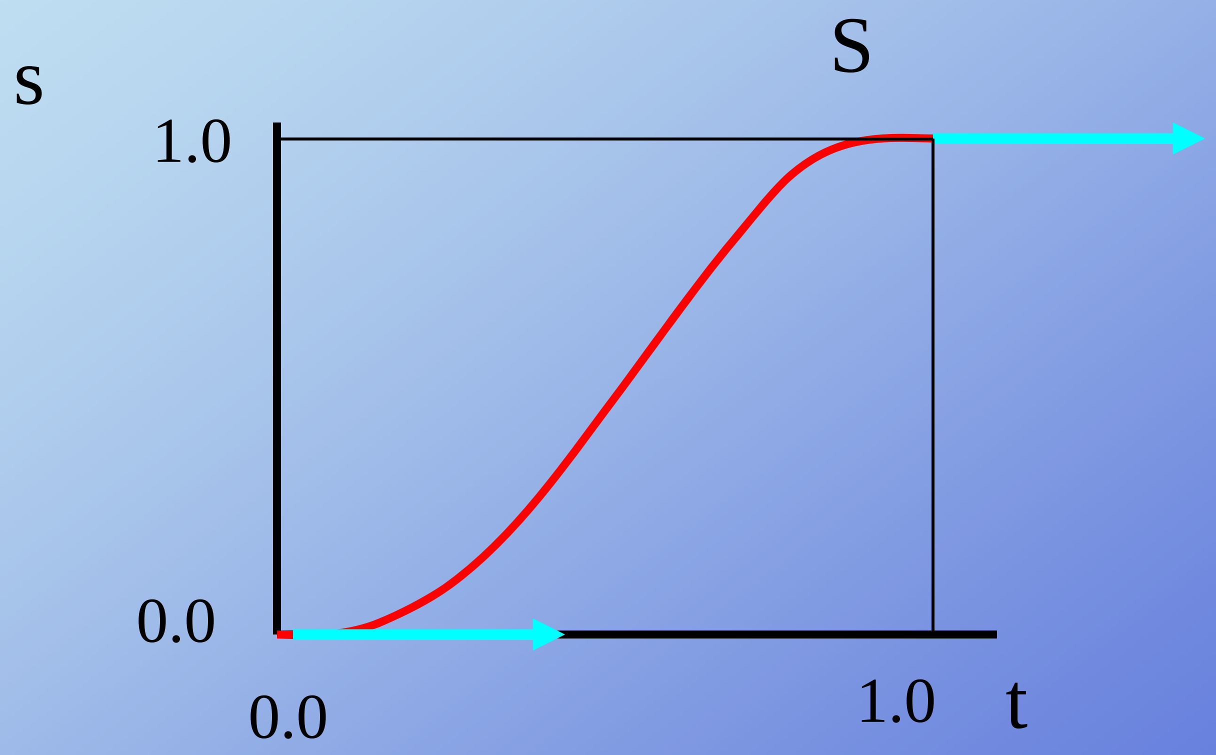
Time Distance Function

$s = S(t)$



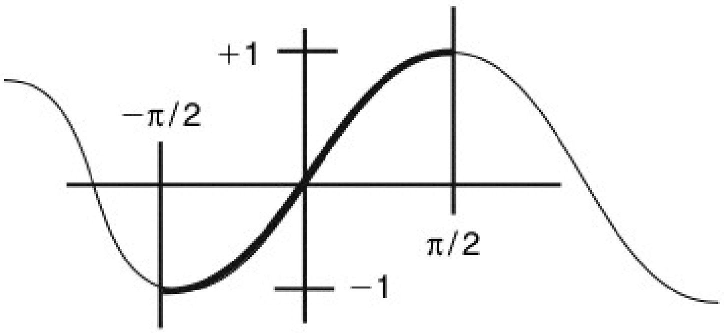
Ease-in/Ease-out Function

$s = S(t)$

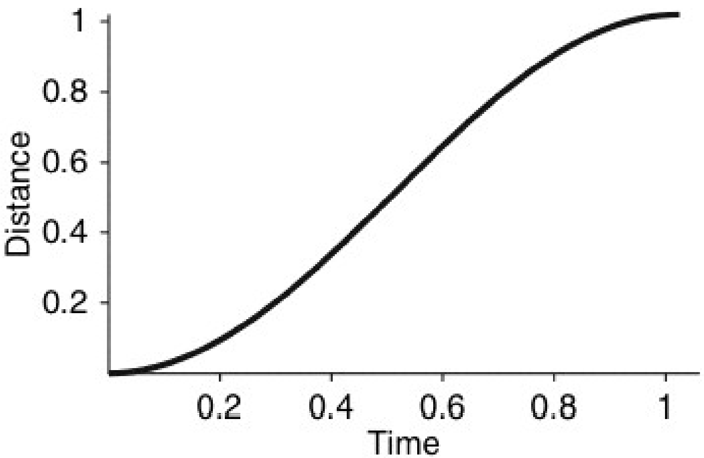


Normalize distance and time to 1.0 to facilitate reuse

Ease-in: Sinusoidal



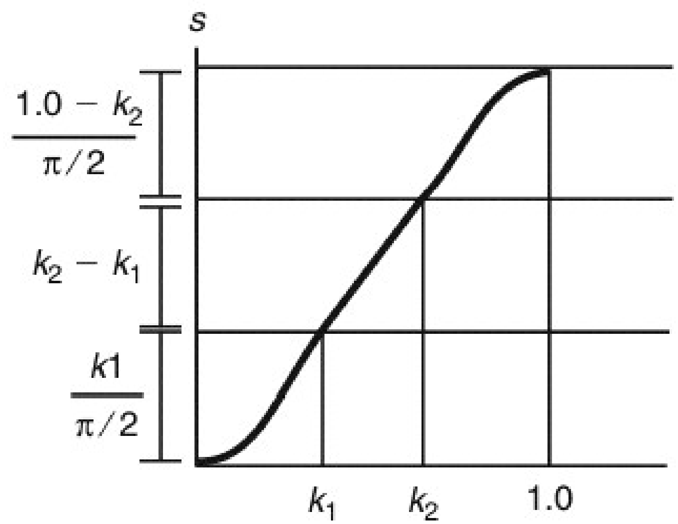
Sine curve segment to use as ease-in/ease-out control



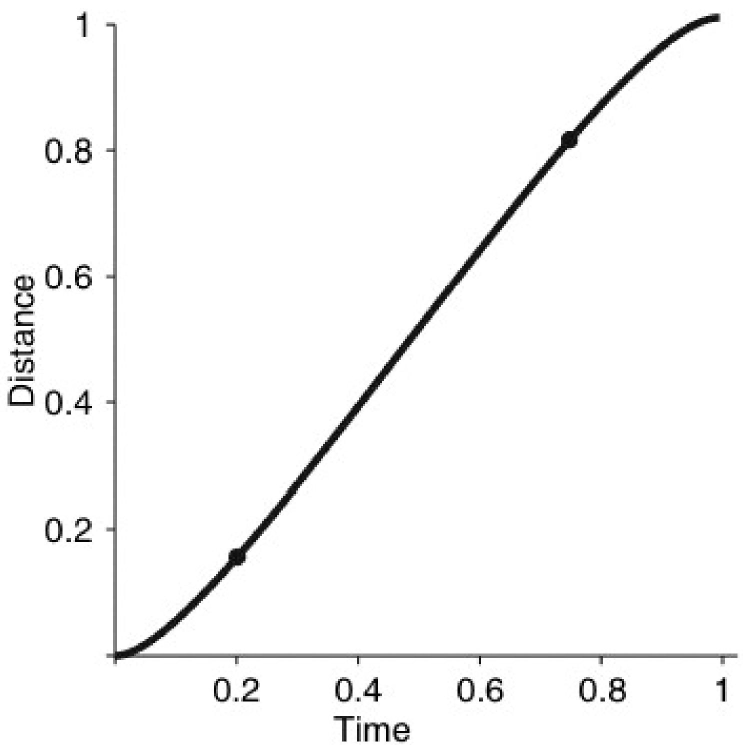
Sine curve segment mapped to useful values

$$s = ease(t) = \left(\sin(t\pi - \pi / 2\right) + 1) / 2$$

Ease-in: Piecewise Sinusoidal



Ease-in/ease-out curve as it is initially pieced together



Curve segments scaled into useful values with points marking segment junctions

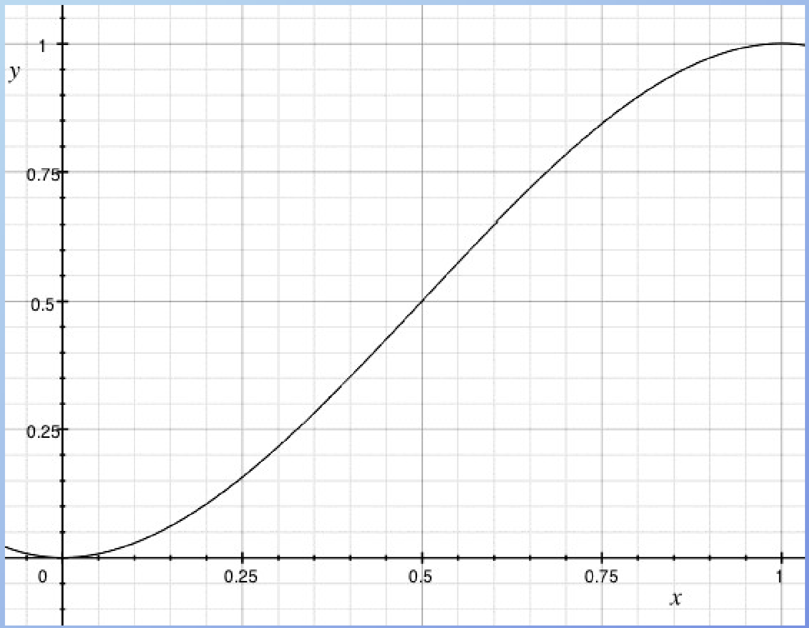
Ease-in: Piecewise Sinusoidal

$$ease(t) = \begin{cases} (k_1 \frac{2}{\pi}(\sin(\frac{t\pi}{2k_1} - \frac{\pi}{2}))) / f & t <= k_1 \\ (\frac{k_1}{\pi/2} + t - k1) / f & k_1 < t <= k_2 \\ (\frac{k_1}{\pi/2} + k_2 - k_1 + (1-k_2)\frac{2}{\pi}\sin(\frac{\pi(t-k_2)}{2(1-k_2)})) / f & k_2 < t \end{cases}$$

where $\quad f = k_1 \frac{2}{\pi} + k_2 - k_1 + (1-k_2)\frac{2}{\pi}$

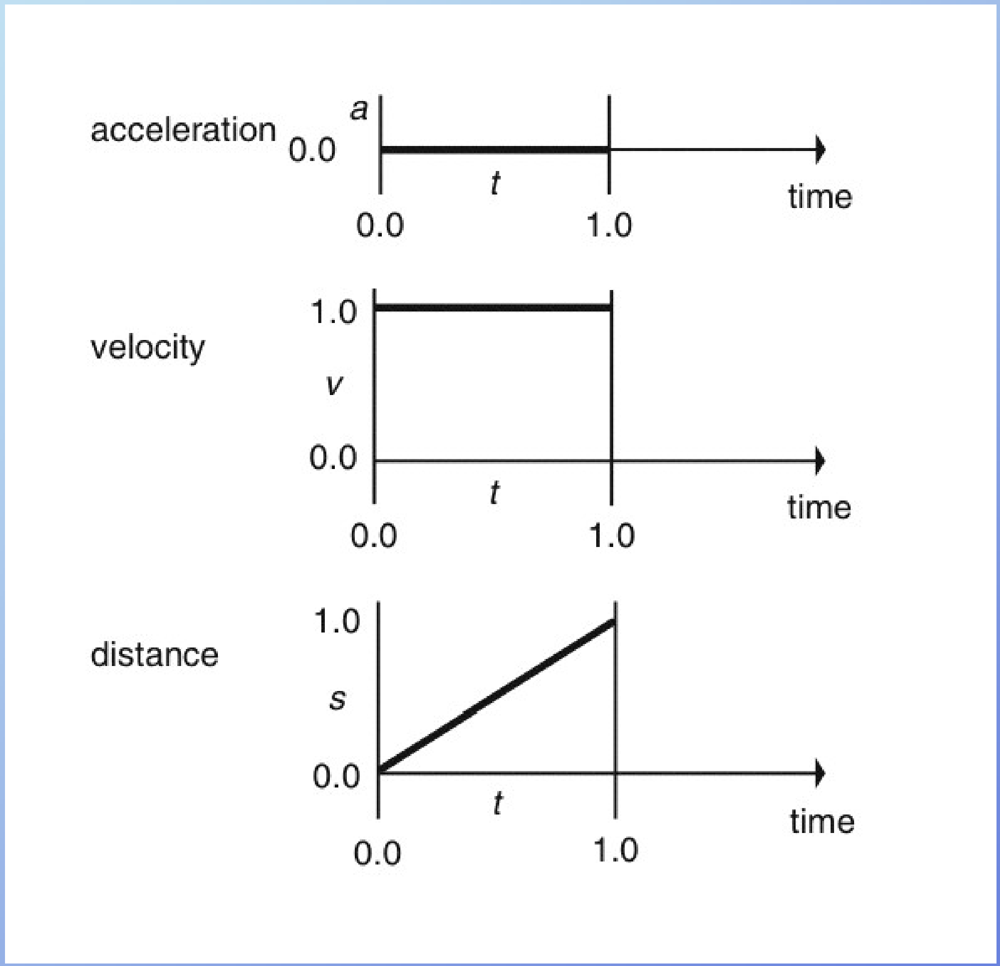
Provides linear (constant velocity) middle segment

Ease-in: Single Cubic

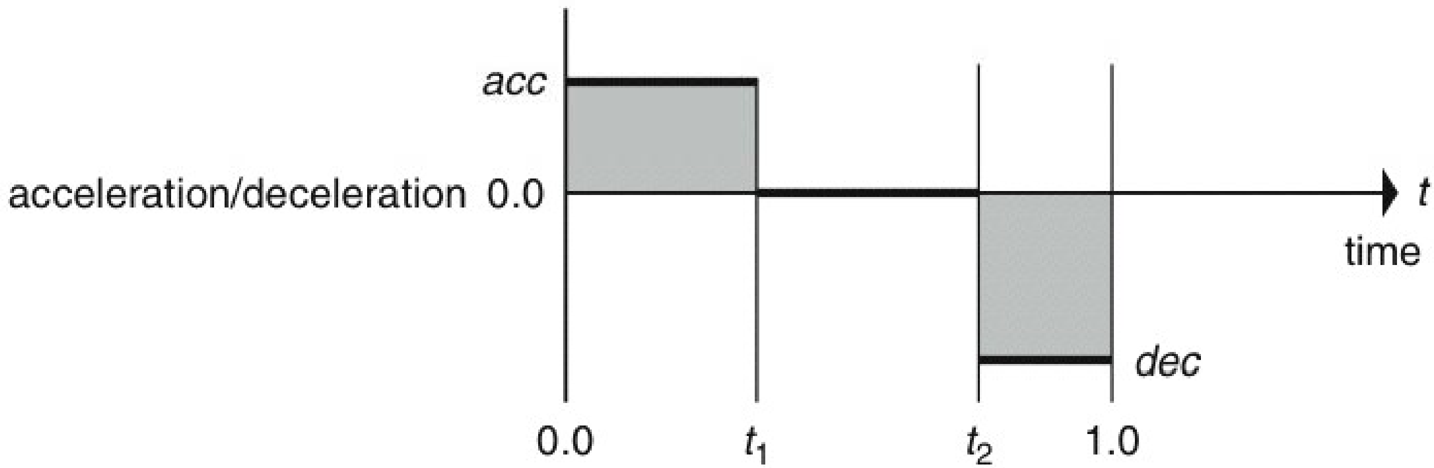


$$s = ease(t) = -2t^3 + 3t^2$$

Ease-in: Constant Acceleration



Ease-in: Constant Acceleration

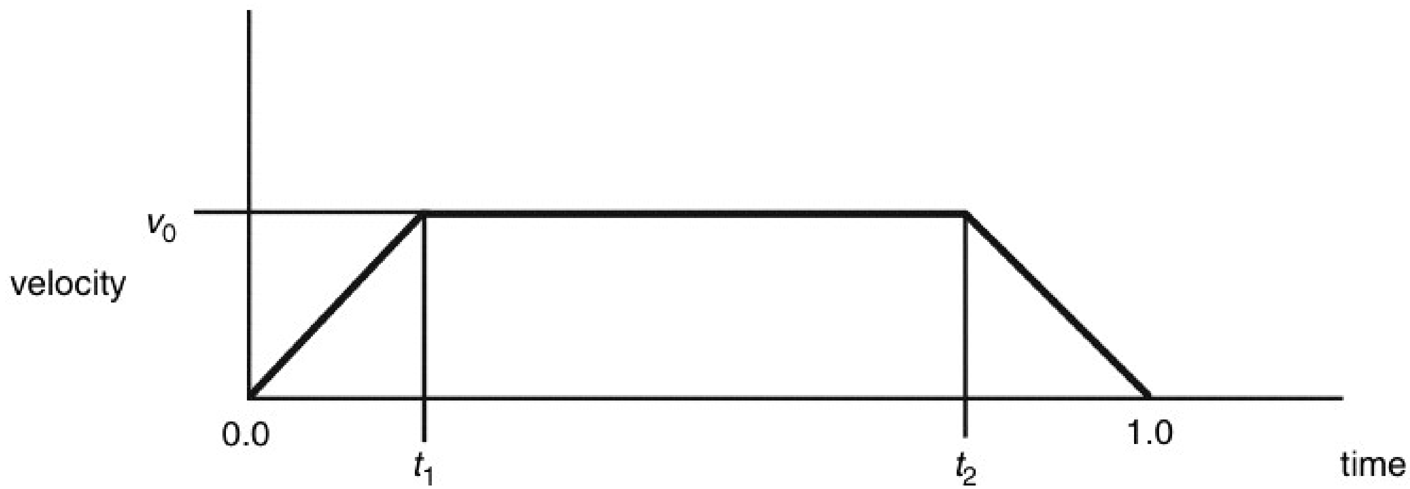


$a = acc \qquad 0 < t < t_1$

$a = 0.0 \qquad t_1 < t < t_2$

$a = dec \qquad t_2 < t < 1.0$

Ease-in: Constant Acceleration

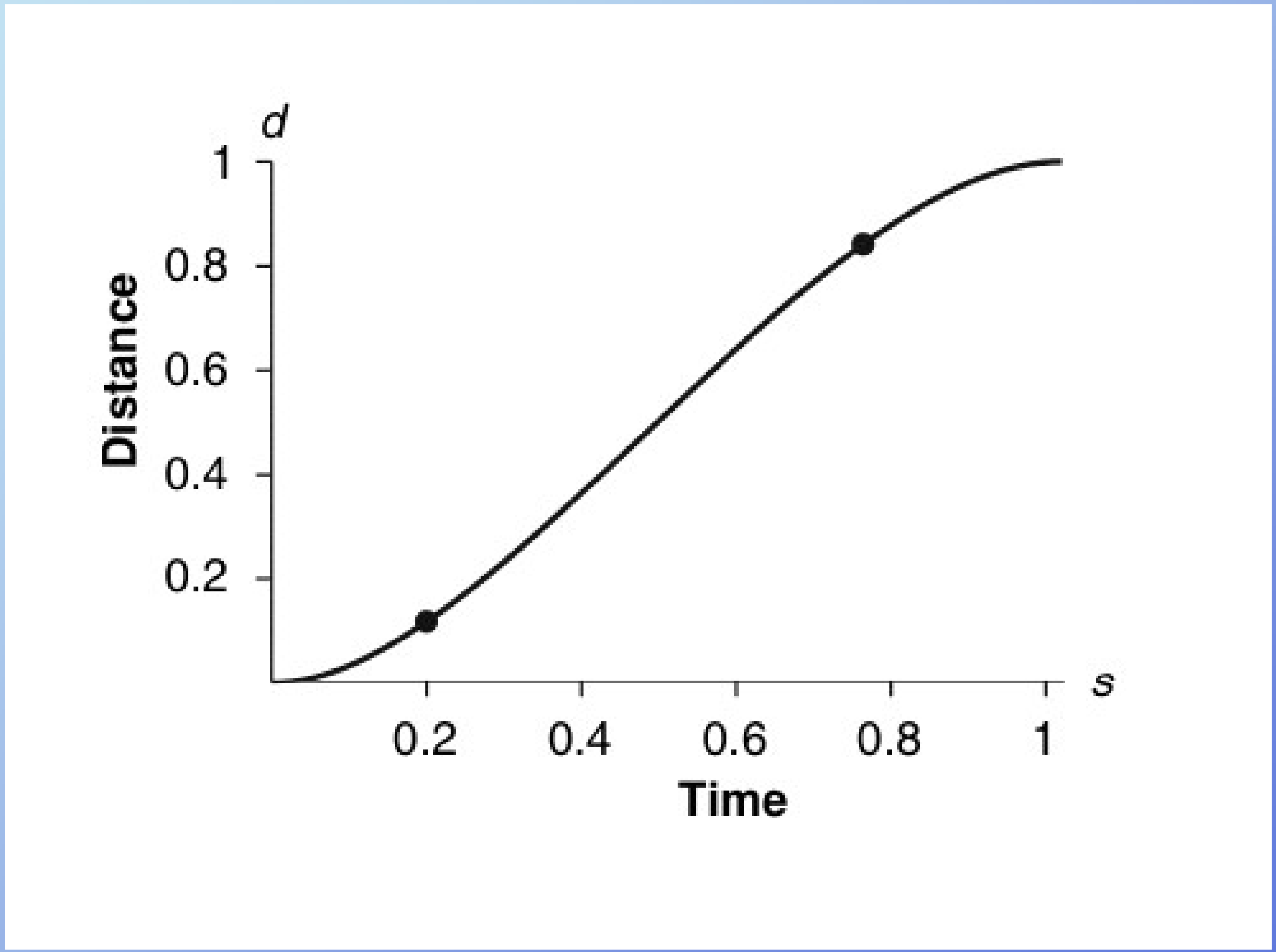


$$v = v_0 \cdot \frac{t}{t_1} \qquad 0.0 < t < t_1$$

$$v = v_0 \qquad t_1 < t < t_2$$

$$v = v_0 \cdot \left(1.0 - \frac{t - t_2}{1 - t_2}\right) \qquad t_2 < t < 1.0$$

Ease-in: Constant Acceleration



Constant Acceleration

$$d = v_0 \frac{t^2}{2t_1} \qquad 0.0 < t \le t_1$$

$$d = v_0 \frac{t_1}{2} + v_0 (t - t_1) \qquad t_1 < t \le t_2$$

$$d = v_0 \frac{t_1}{2} + v_0 (t_2 - t_1) + v_0 \left(1 - \frac{(t - t_2)}{2(1 - t_2)}\right)(t - t_2) \qquad t_2 < t \le 1.0$$

Motivation – solution steps

1. Construct a space curve that interpolates the given points with piecewise first order continuity

$p=P(u)$

2. Construct an arc-length-parametric-value function for the curve

$u=U(s)$

3. Construct time-arc-length function according to given constraints

$s=S(t)$

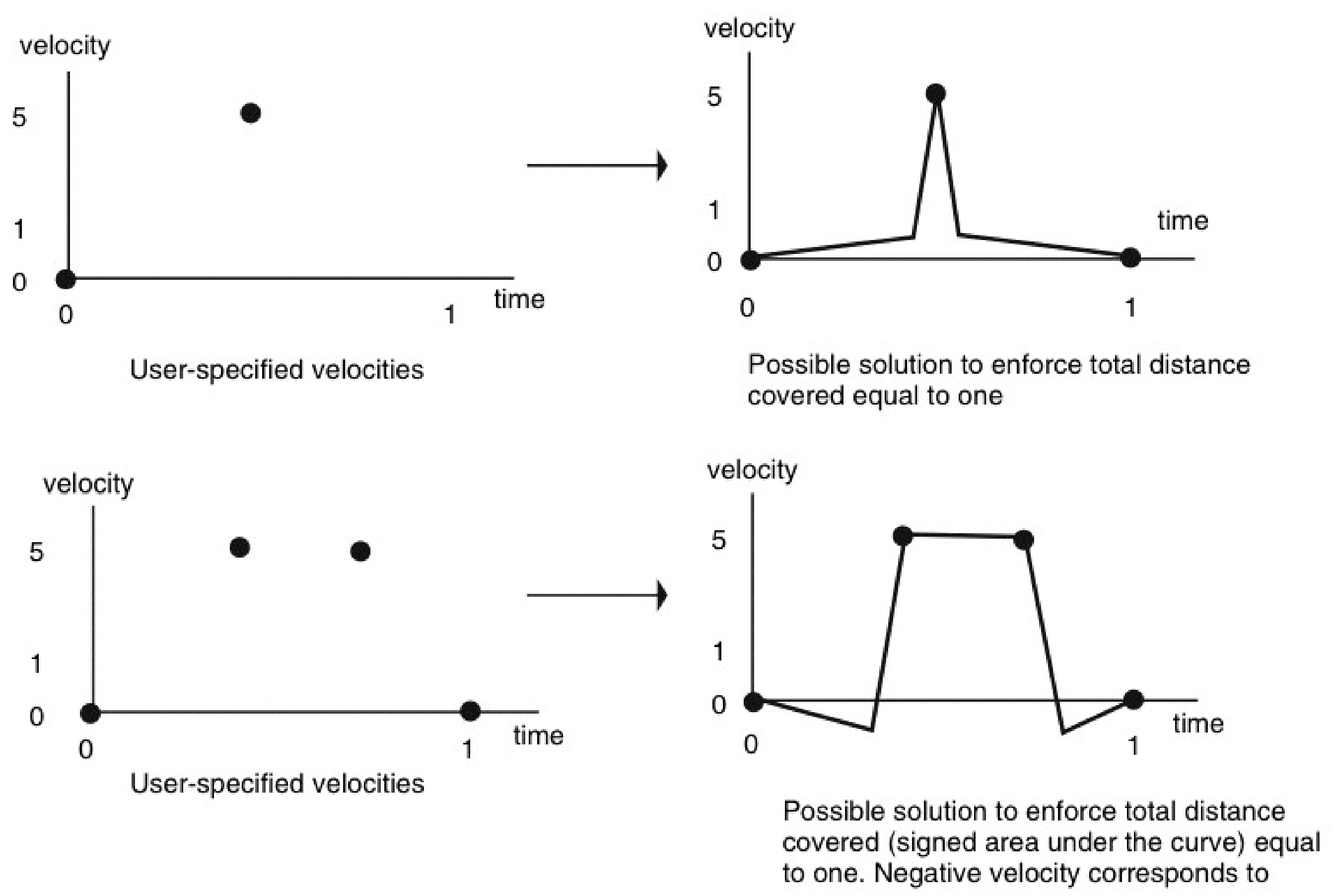
$$p=P(U(S(t)))$$

Arbitrary Speed Control

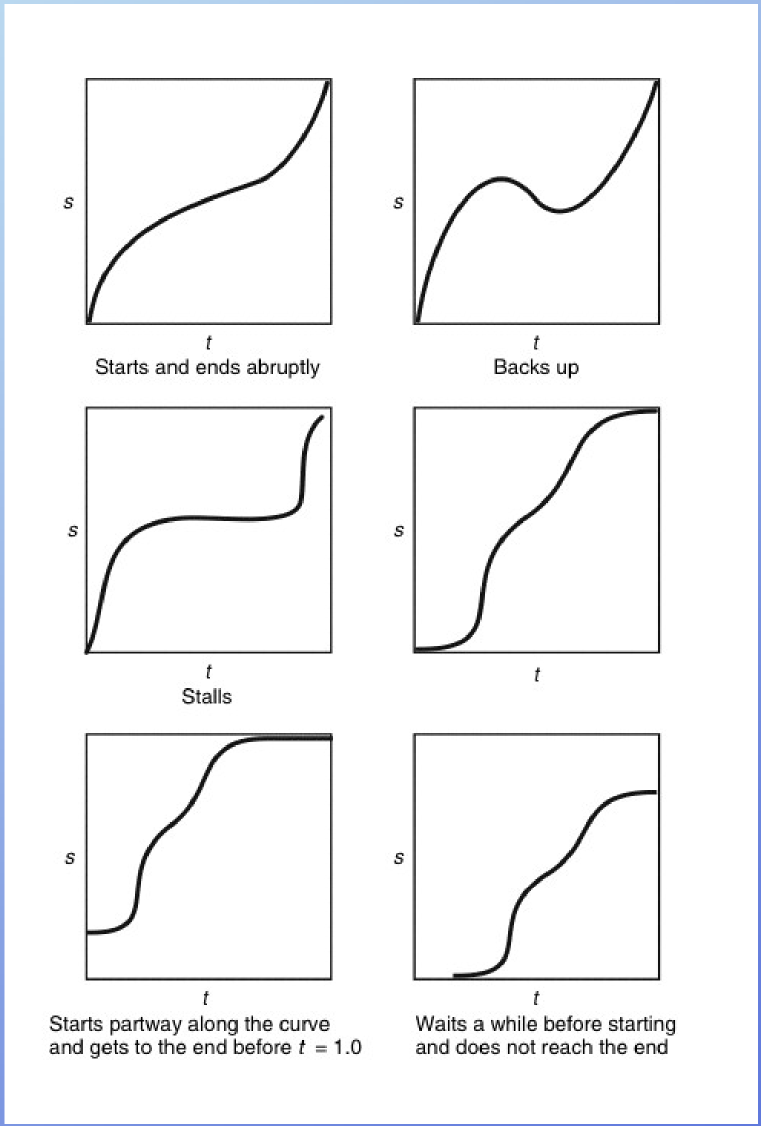
Animators can work in:

- **Distance-time space curves**
- **Velocity-time space curves**
- **Acceleration-time space curves**
- **Set time-distance constraints**
- **etc.**

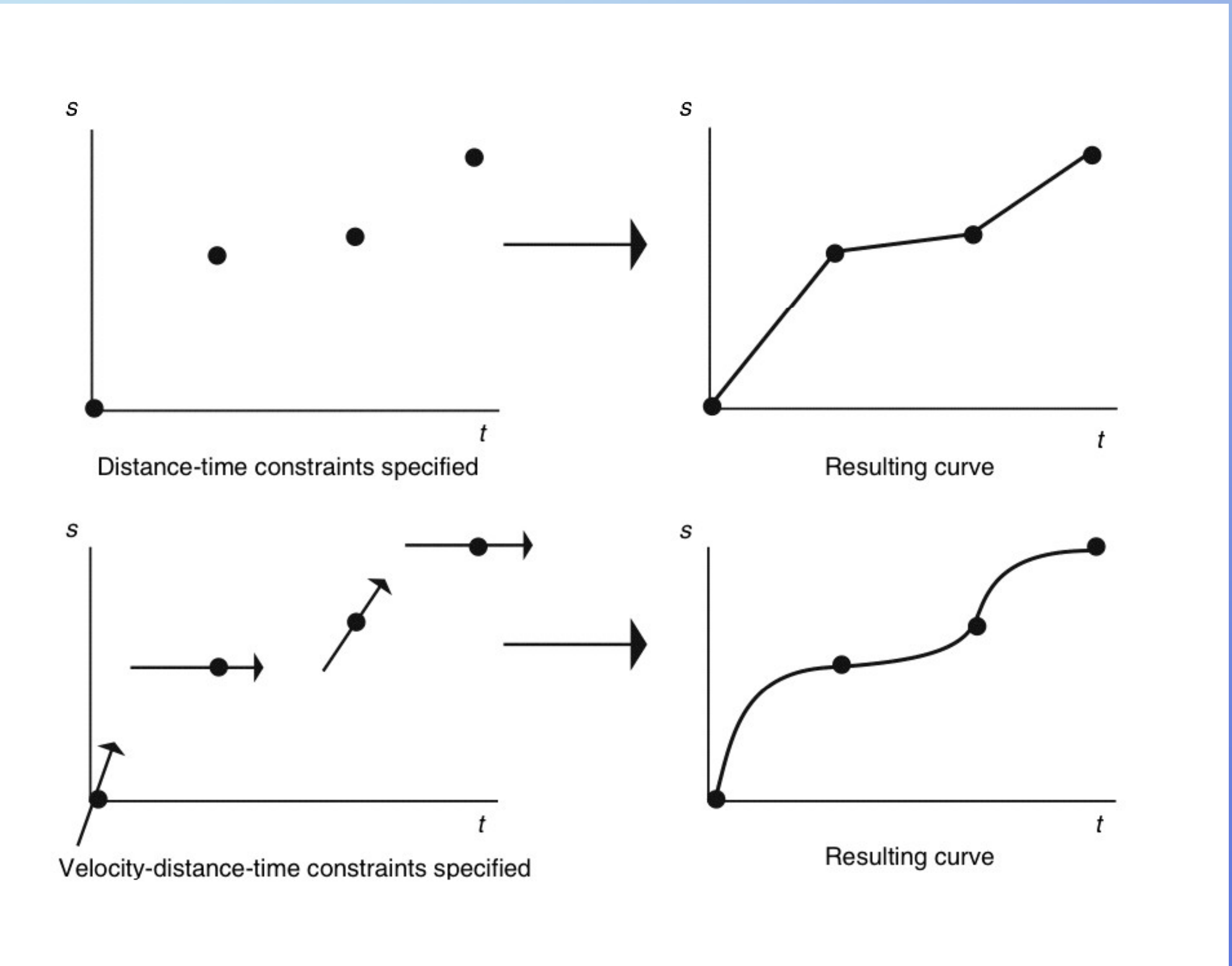
Curve fitting to distance-time pairs



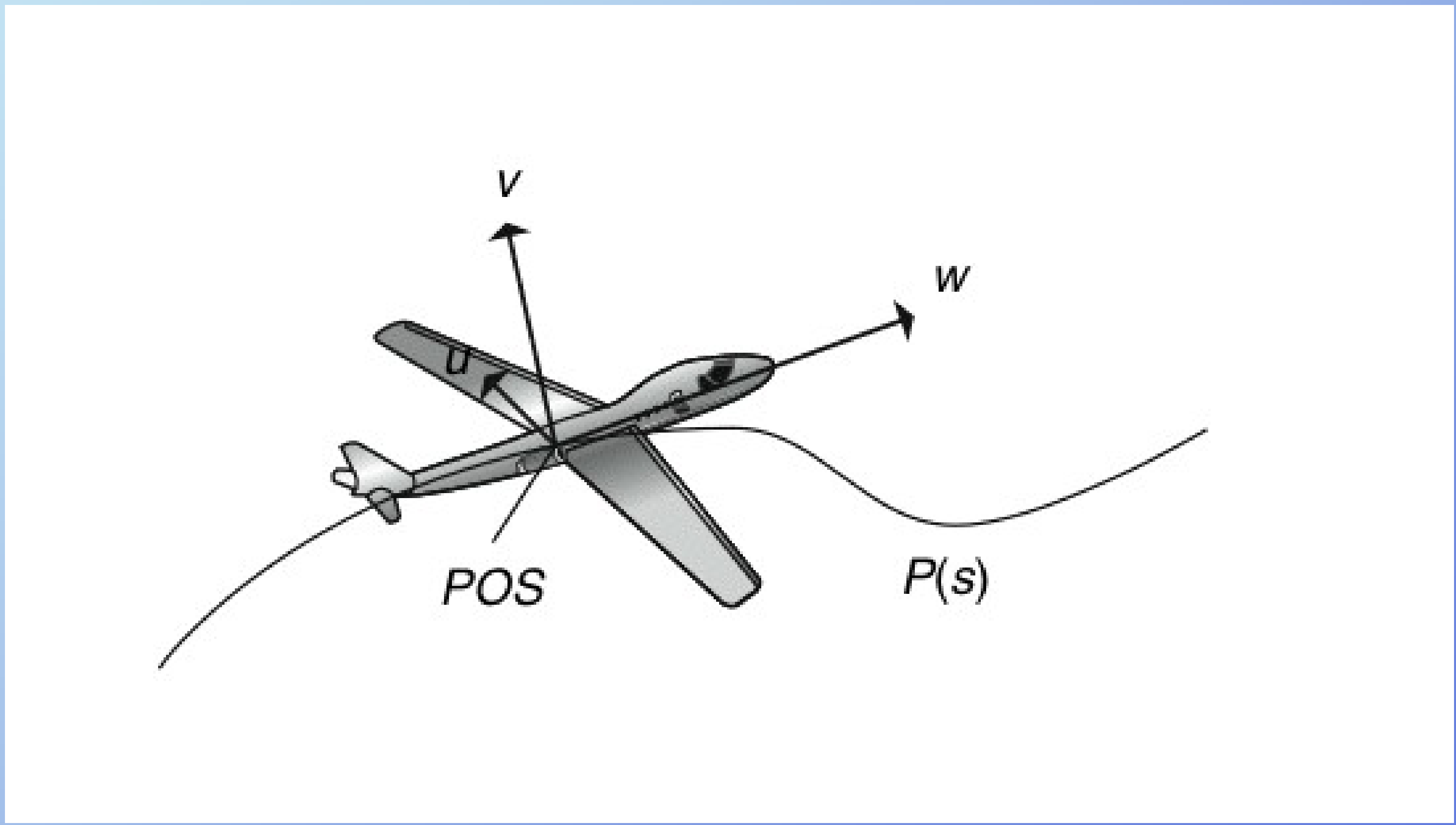
Working with time-distance curves



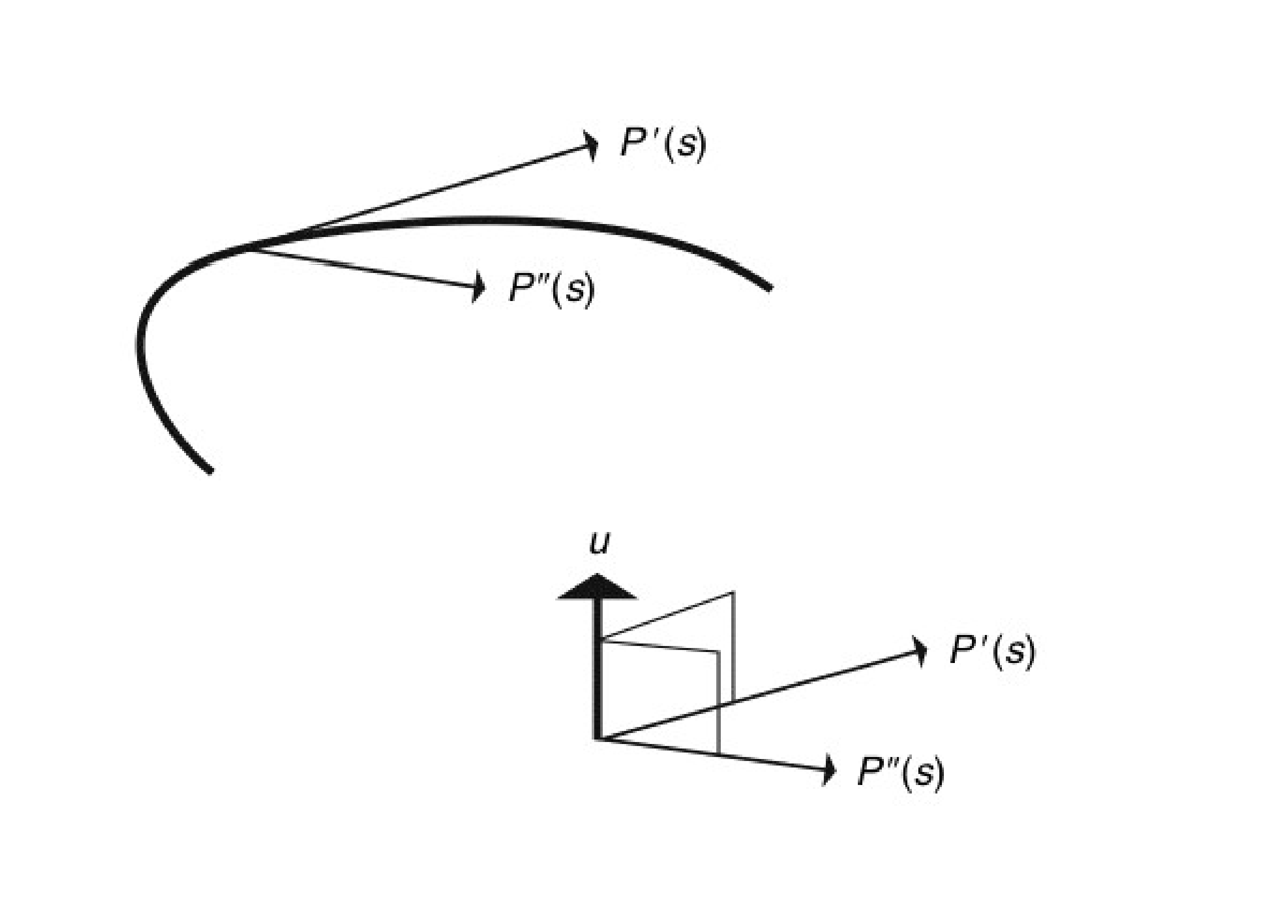
Interpolating distance-time pairs



Frenet Frame – control orientation



Frenet Frame
tangent & curvature vector

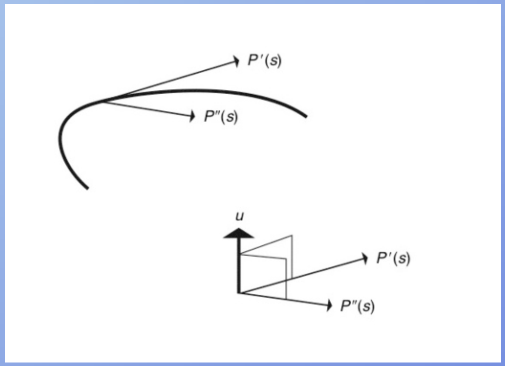


Frenet Frame
tangent & curvature vector

$$P(u) = UMB$$

$$P'(u) =$$

$$P''(u) =$$



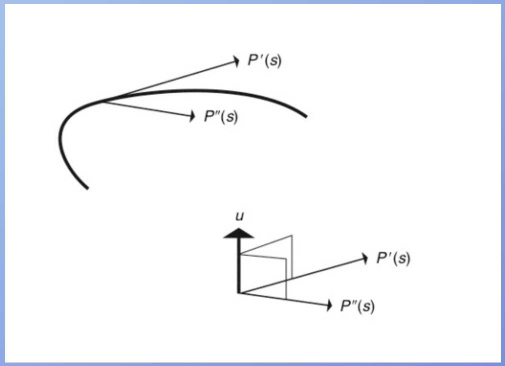
$$U = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}$$

Frenet Frame
tangent & curvature vector

$$P(u) = UMB$$

$$P'(u) = U'MB$$

$$P''(u) = U''MB$$



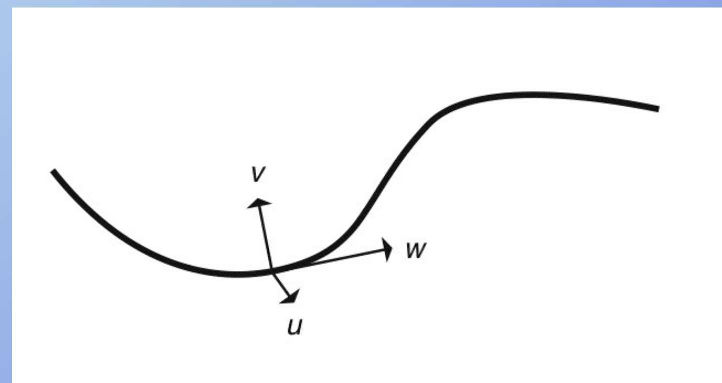
$$U = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}$$

$$U' = \begin{bmatrix} 3u^2 & 2u & 1 & 0 \end{bmatrix}$$

$$U'' = \begin{bmatrix} 6u & 2 & 0 & 0 \end{bmatrix}$$

Frenet Frame local coordinate system

•Directly control orientation of object/camera



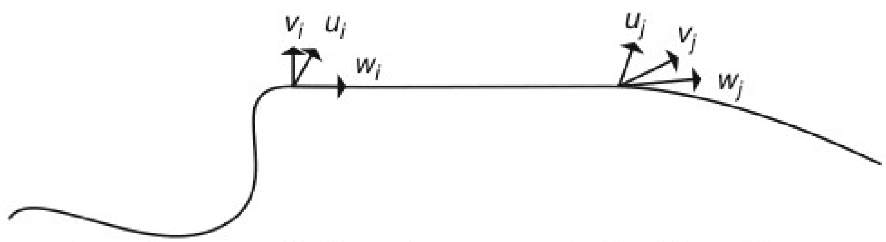
•Use for direction and bank into turn, especially for ground-planar curves (e.g. roads)

•v is perpendicular to w if curve is parameterized by arclength; otherwise probably not perpendicular

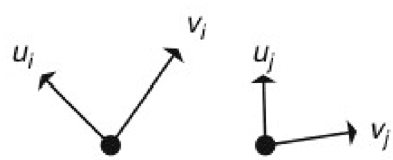
•For general curve must

v = wxu

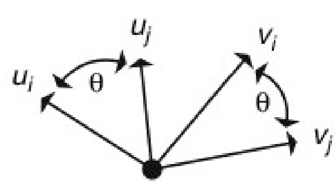
Frenet Frame - undefined



Frenet frames on the boundary of an undefined Frenet frame segment becuse of zero curvature.

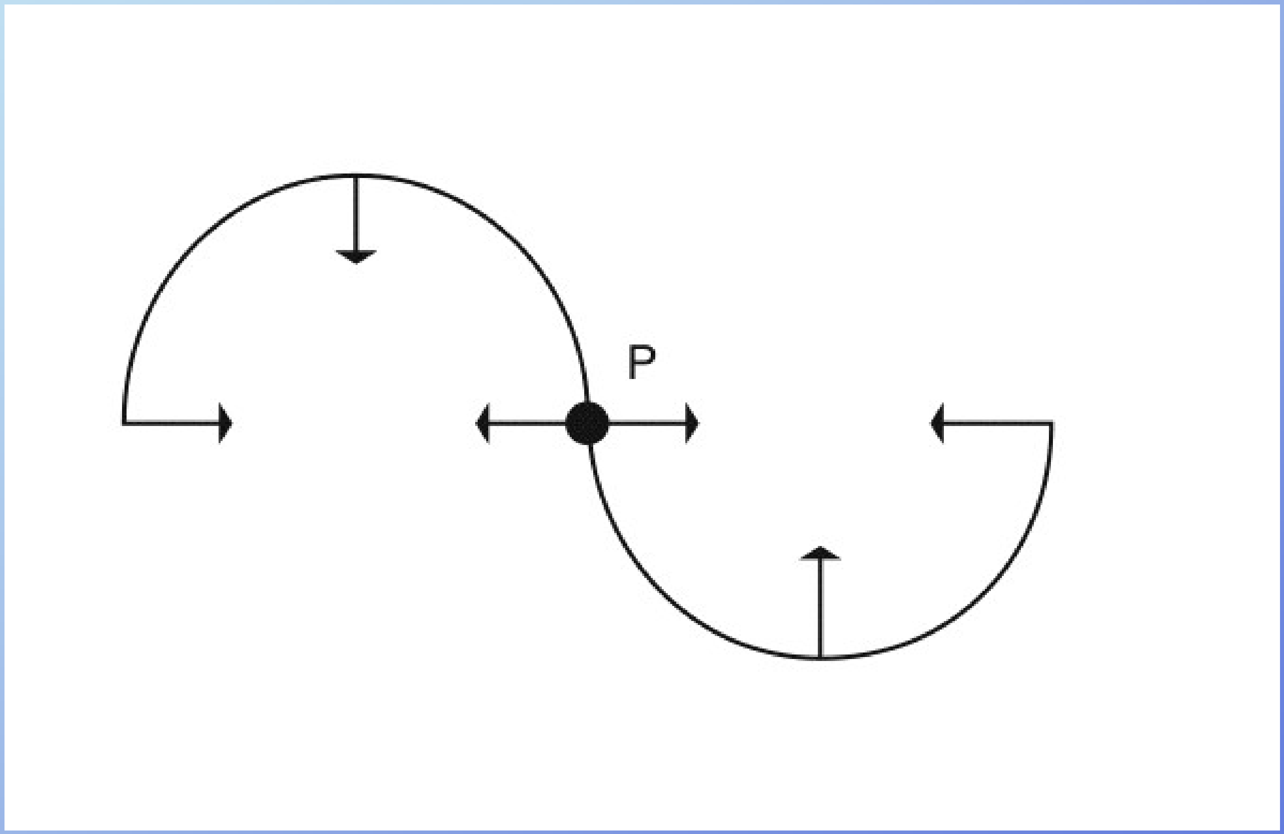


The two frames sighted down the (common) *w* vector.



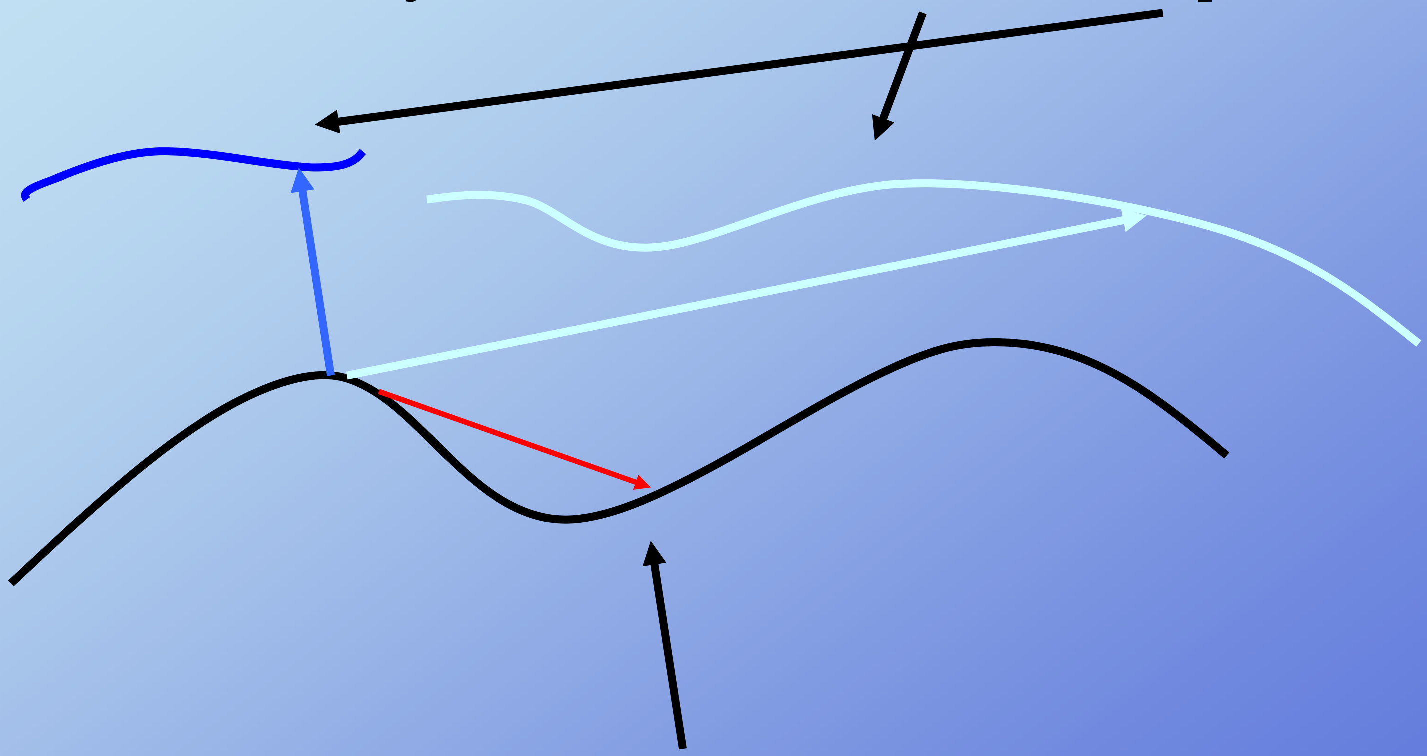
The two frames superimposed to identify angular difference.

Frenet Frame - discontinuity



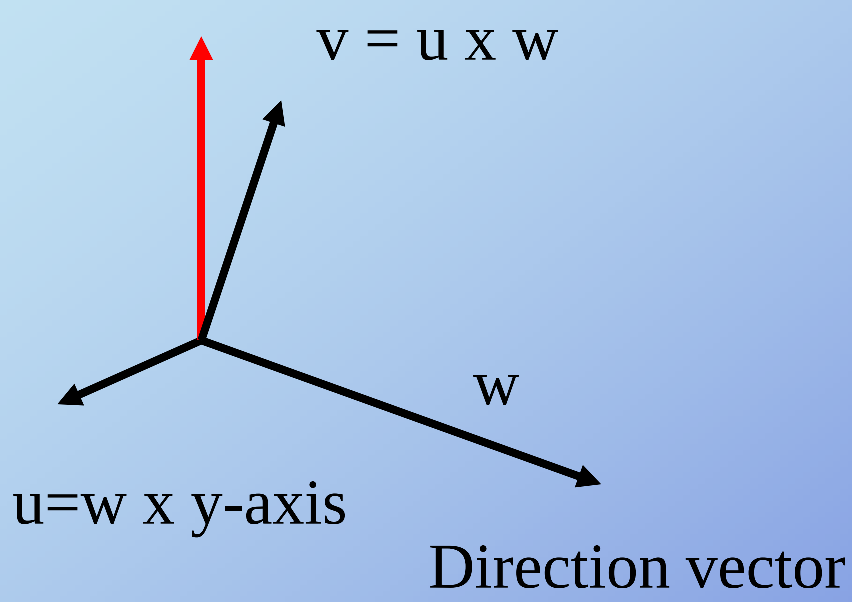
Other ways to control orientation

Use auxiliary curve to define direction or up vector



Use point P(s+ds) for direction

Direction & Up vector



To keep ‘head up’, use y-axis to compute over and up vectors perpendicular to direction vector

If up vector supplied, use that instead of y-axis

Orientation interpolation

Preliminary note:

1. **Remember that** $Rot_q(v) \equiv Rot_{kq}(v)$
2. **Affects of scale are divided out by the inverse appearing in quaternion rotation**
3. **When interpolating quaternions, use UNIT quaternions – otherwise magnitudes can interfere with spacing of results of interpolation**

Orientation interpolation

Quaternions can be interpolated to produce in-between orientations:

$$q = (1-k)q_1 + kq_2$$

2 problems analogous to issues when interpolating positions:

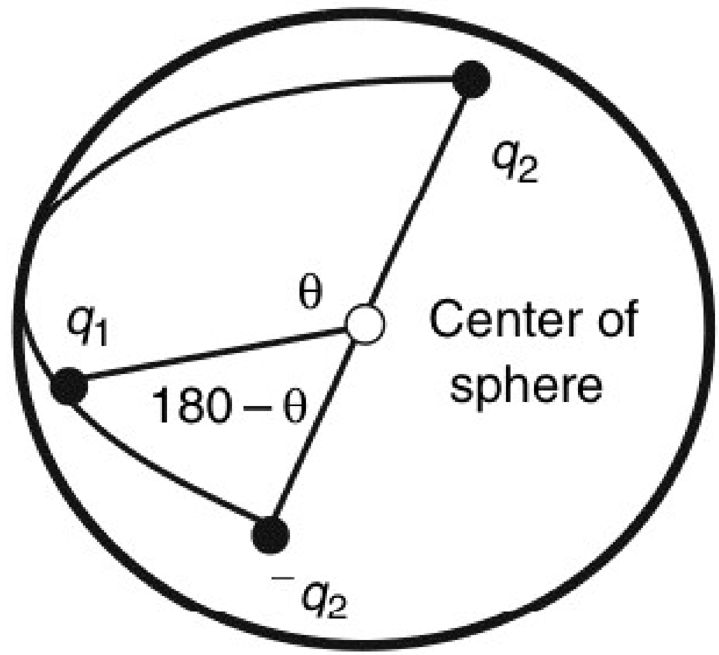
1. **How to take equi-distant steps along orientation path?**
2. **How to pass through orientations smoothly (1st order continuous)**

3. **And another particular to quaternions: with dual unit quaternion representations, which to use?**

Dual representation

$$Rot_q(v) = Rot_{kq}(v)$$

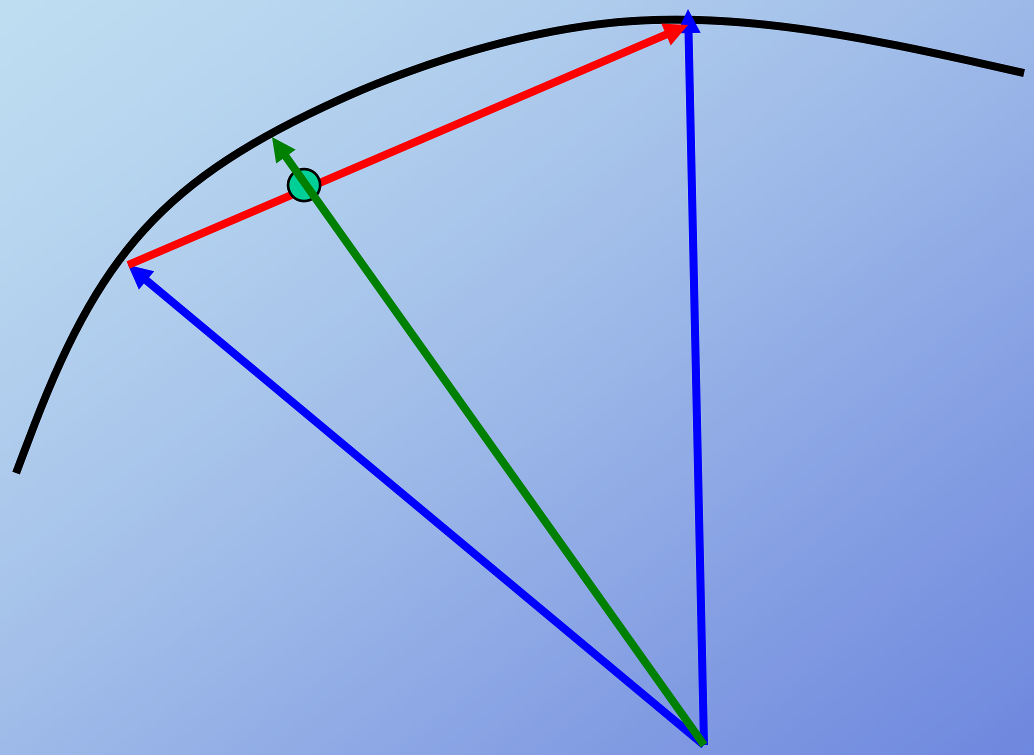
Dual unit quaternion representations



For Interpolation between q1 and q2, compute cosine between q1 and q2 and between q1 and –q2; choose smallest angle

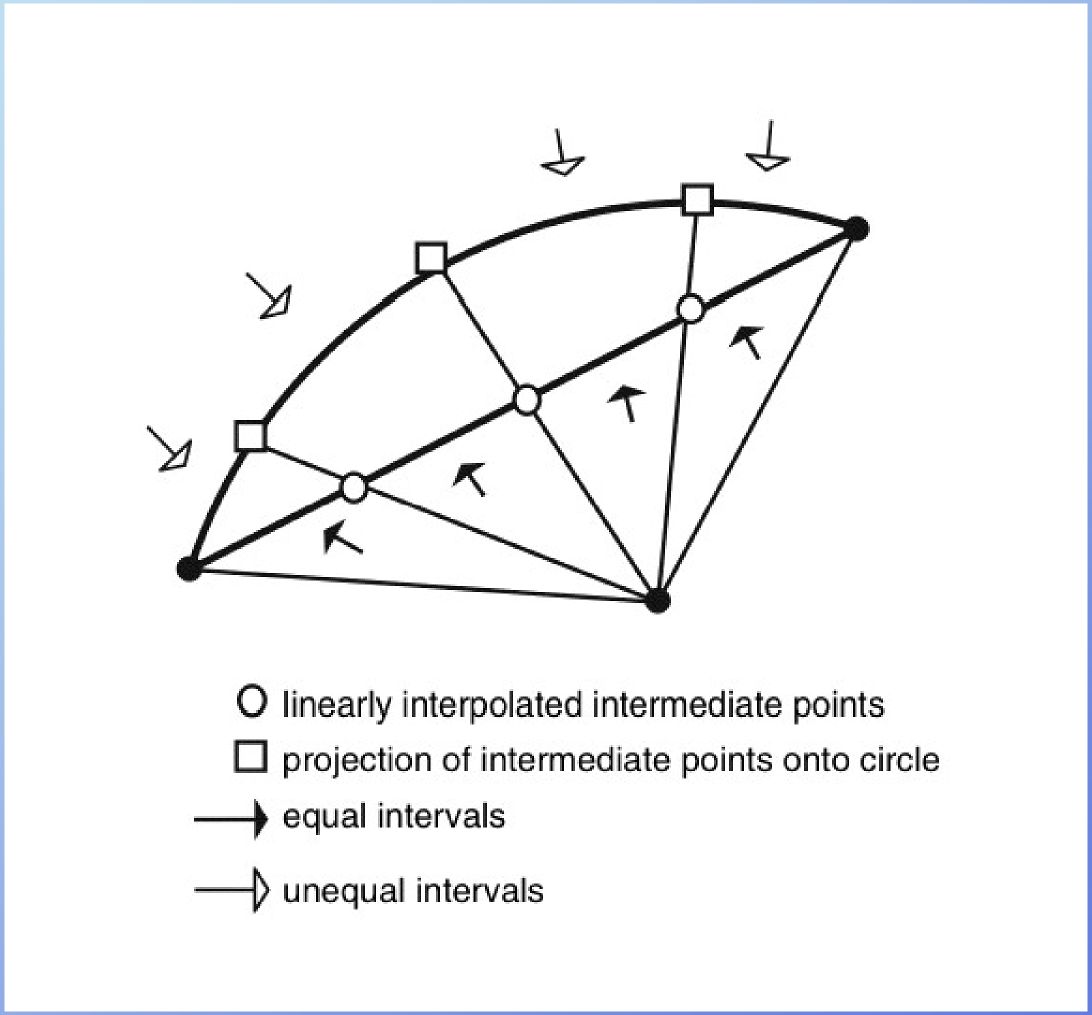
Interpolating quaternions

Unit quaternions form set of points on 4D sphere



Linearly interpolating unit quaternions: not equally spaced

Interpolating quaternions in great arc => equal spacing



Interpolating quaternions with equal spacing

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

where $q_1 \cdot q_2 = \cos\theta$

'slerp', sphereical linear interpolation is a function of

- **the beginning quaternion orientation, q1**
- **the ending quaternion orientation, q2**
- **the interpolant, u**

Smooth Orientation interpolation

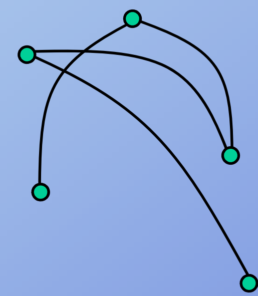
Use quaternions

Interpolate along great arc (in 4-space) using cubic Bezier on sphere

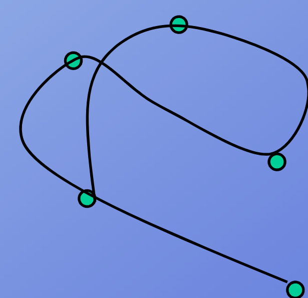
1. **Select representation to use from duals**
2. **Construct interior control points for cubic Bezier**
3. **use DeCastelajue construction of cubic Bezier**

Smooth quaternion interpolation

Similar to first order continuity desires with positional interpolation

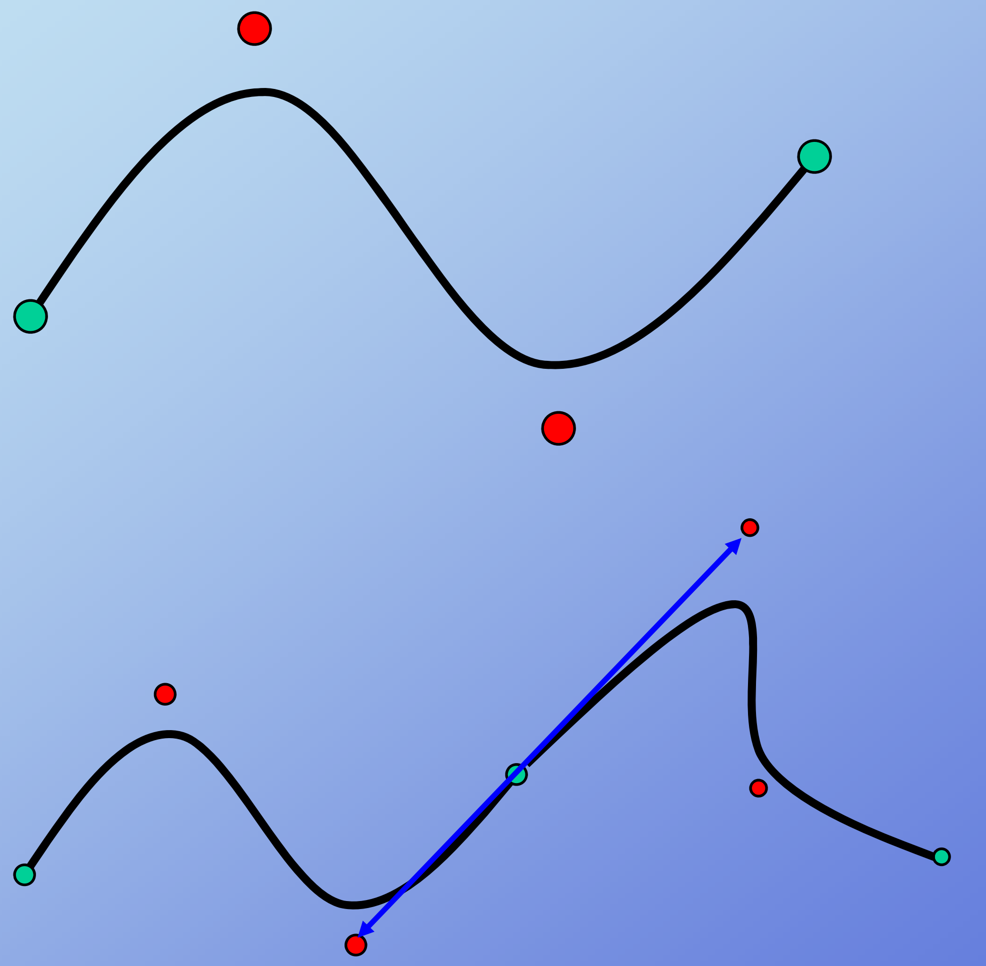


How to smoothly interpolate through orientations q_1, q_2, q_3,…q_n



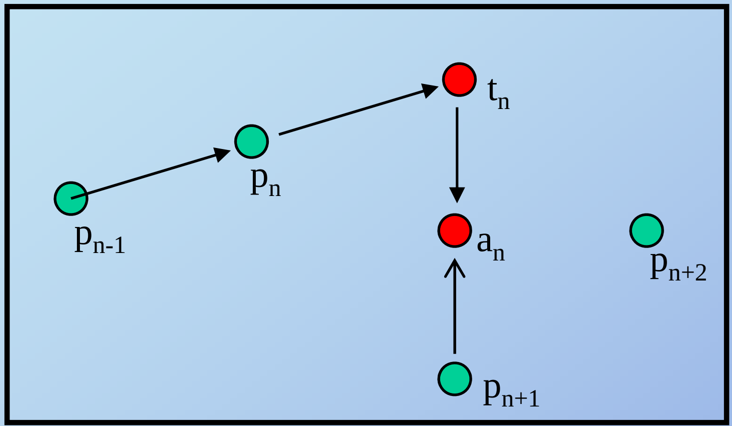
Bezier interpolation – geometric construction

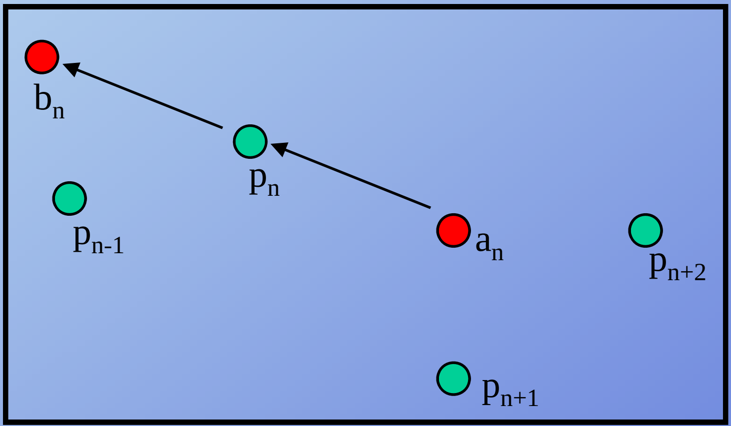
Bezier interpolation

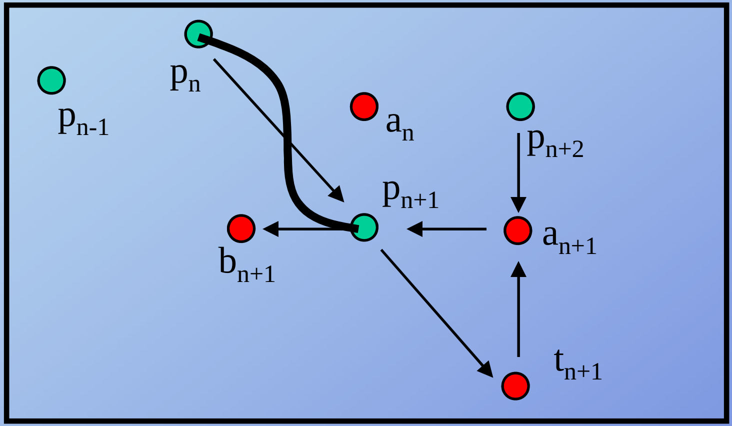


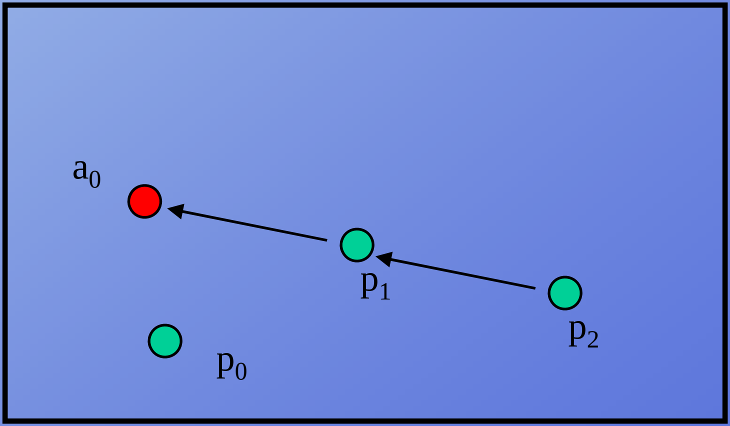
Bezier interpolation

Construct interior control points





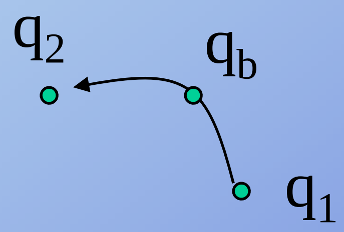




Quaternion operators

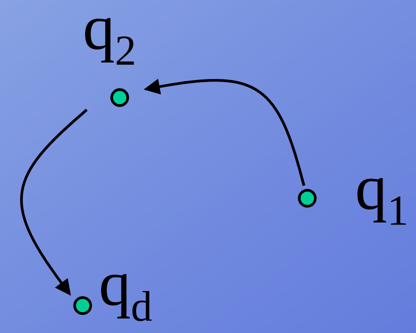
bisect(q_1, q_2)

Similar to forming a vector between 2 points, form the rotation between 2 orientations



double(q_1, q_2)

Given 2 orientations, form result of applying rotation between the two to 2nd orientation

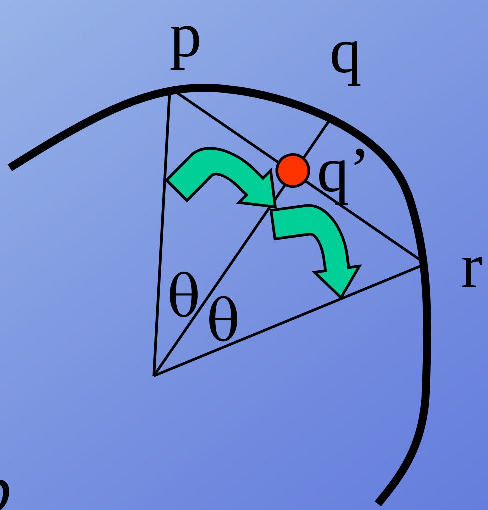


Quaternion operators:

$double(p,q)=r$

Given p and q, form r

'Double' where q' is the mid-orientation between p and the yet-to-be-determined r



If p and q are unit quaternions,
Then q' = cos(θ)q and cos(θ)= $p \cdot q$

$$q' = \cos(\theta) q = (p \cdot q) q$$

$$double(p,q) = r = q' + (q' - p) = 2(p \cdot q) q - p$$

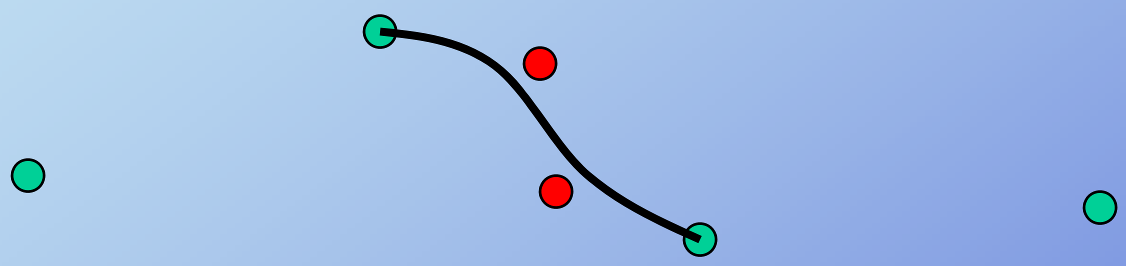
$\text{bisect}(p,r)=q$

Given p and r, form q

Bisect 2 orientations:
if p and r are unit length

$$\text{bisect}(p,r) = \frac{p+r}{\|p+r\|} = q$$

Bezier interpolation

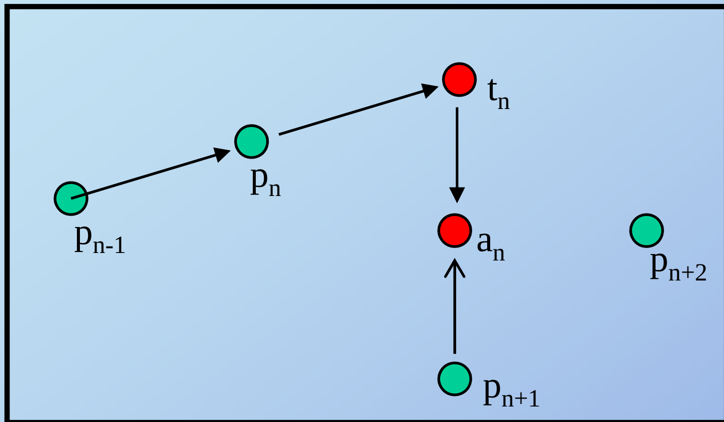


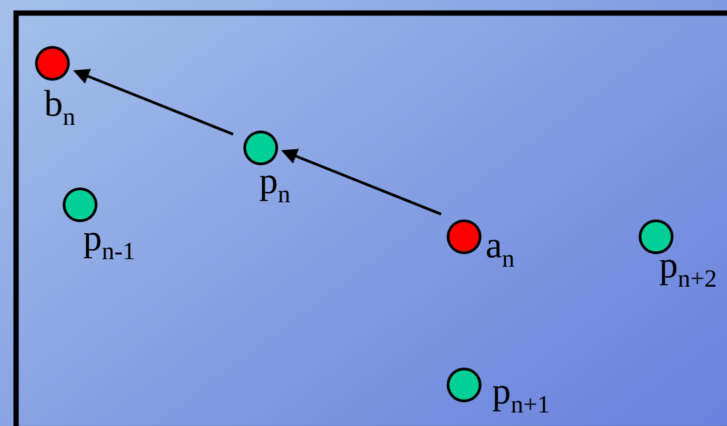
Need quaternion-friendly operators to form interior control points

Bezier interpolation

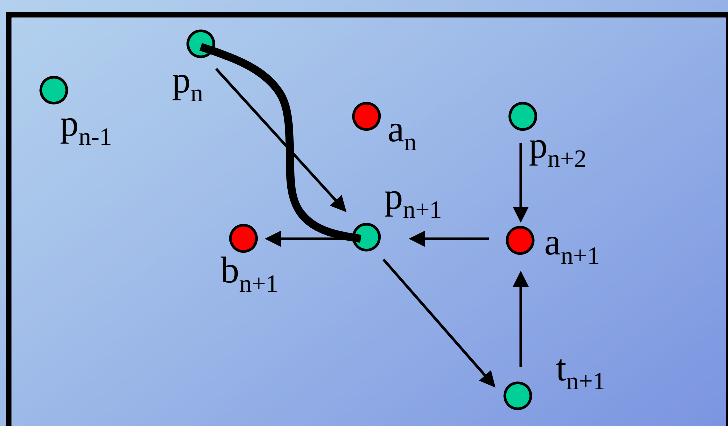
Construct interior control points

$$a_n = \text{bisect}(\text{double}(p_{n-1}, p_n), p_{n+1})$$





$$b_n = double(a_n, q_n)$$



Bezier segment:
q_n, a_n, b_{n+1}, q_{n+1}

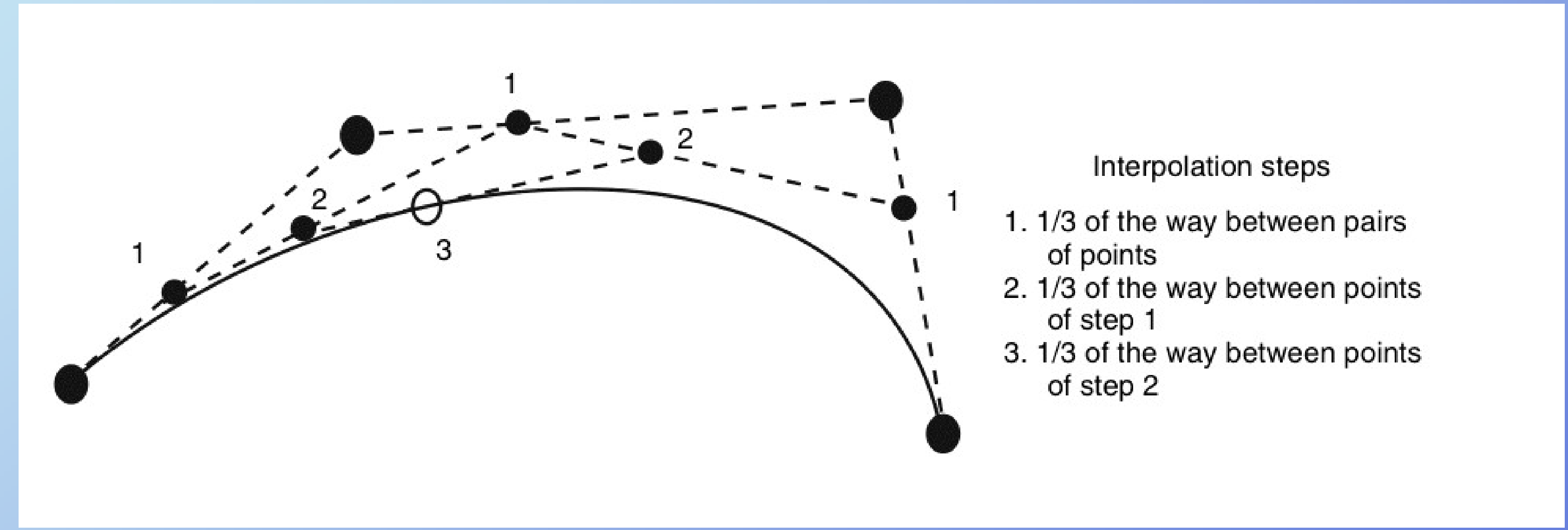
Bezier construction using quaternion operators

Need quaternion-friendly operations to interpolate cubic Bezier curve using 'quaternion' points

de Casteljau geometric construction algorithm

Bezier construction using quaternion operators

For p(1/3)



$t_1 = \mathrm{slerp}(q_n, a_n, 1/3)$
$t_2 = \mathrm{slerp}(a_n, b_{n+1}, 1/3)$
$t_3 = \mathrm{slerp}(b_{n+1}, q_{n+1}, 1/3)$

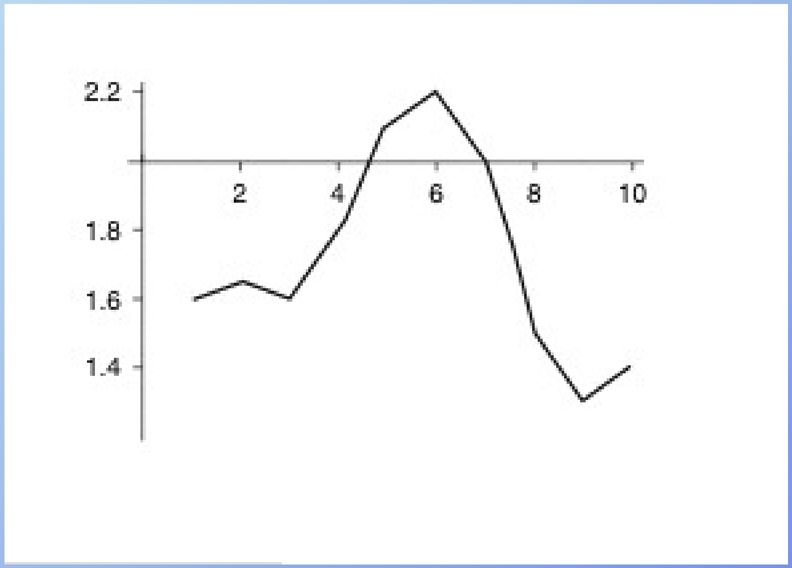
$t_{12} = \mathrm{slerp}(t_1, t_2, 1/3)$
$t_{23} = \mathrm{slerp}(t_{12}, t_{23}, 1/3)$

$q = \mathrm{slerp}(t_{12}, t_{23}, 1/3)$

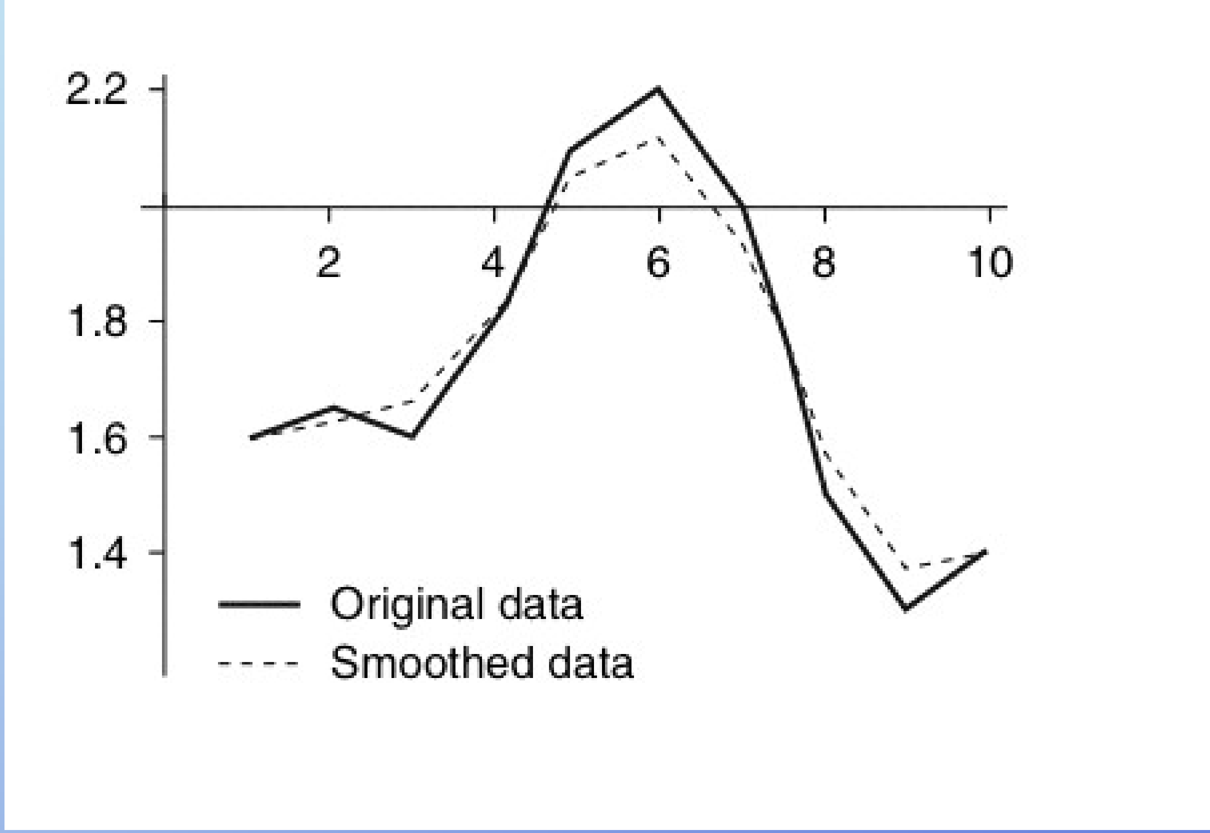
Working with paths

Smoothing a path
Determining a path along a surface
Finding downhill direction

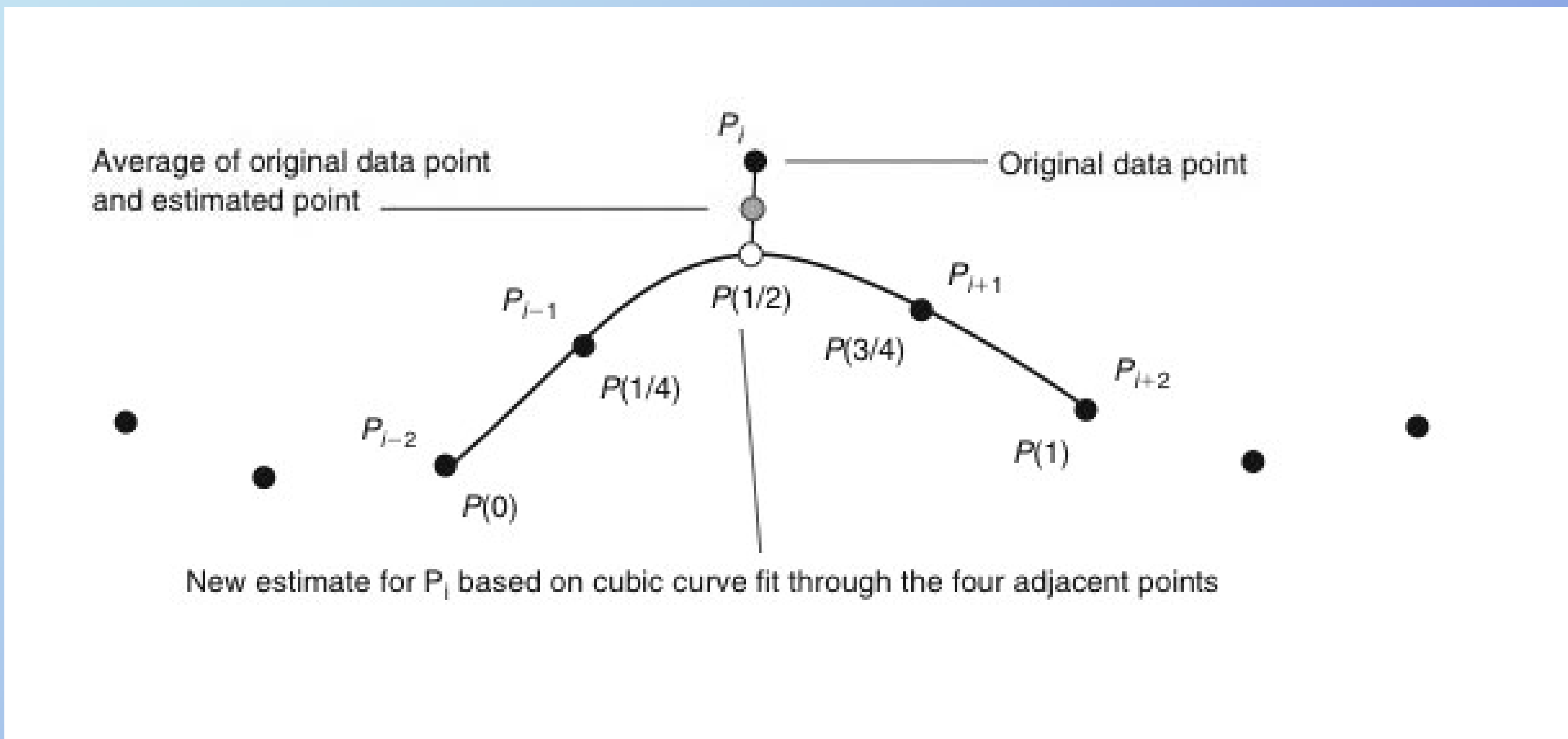
Smoothing data



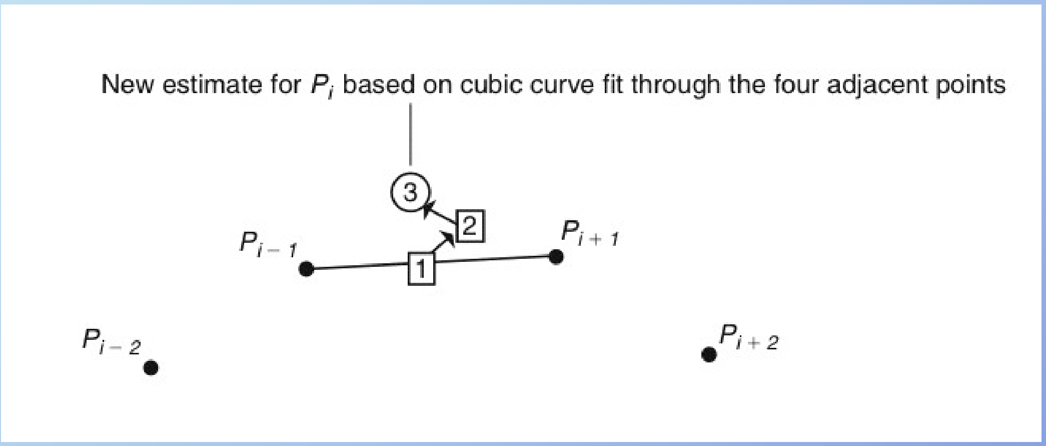
Smoothing data

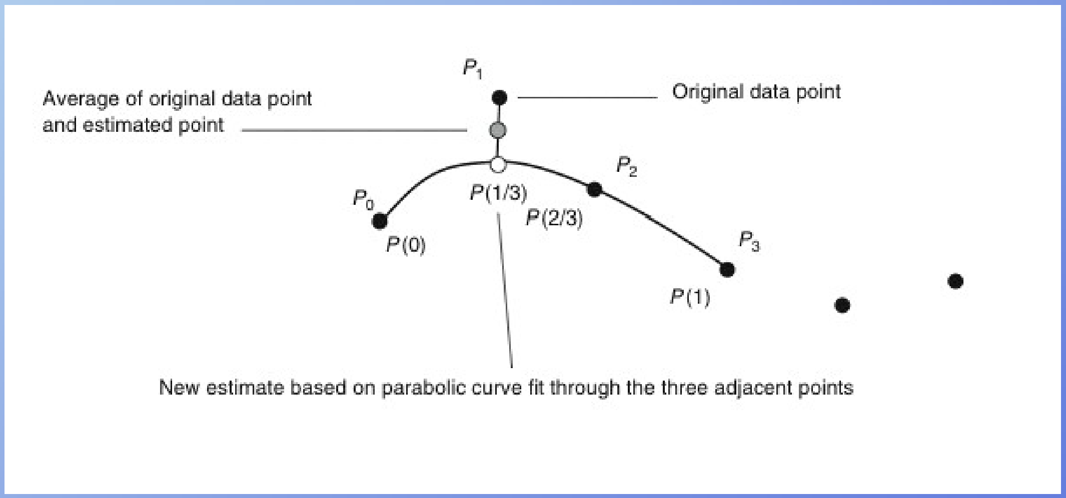


Smoothing data

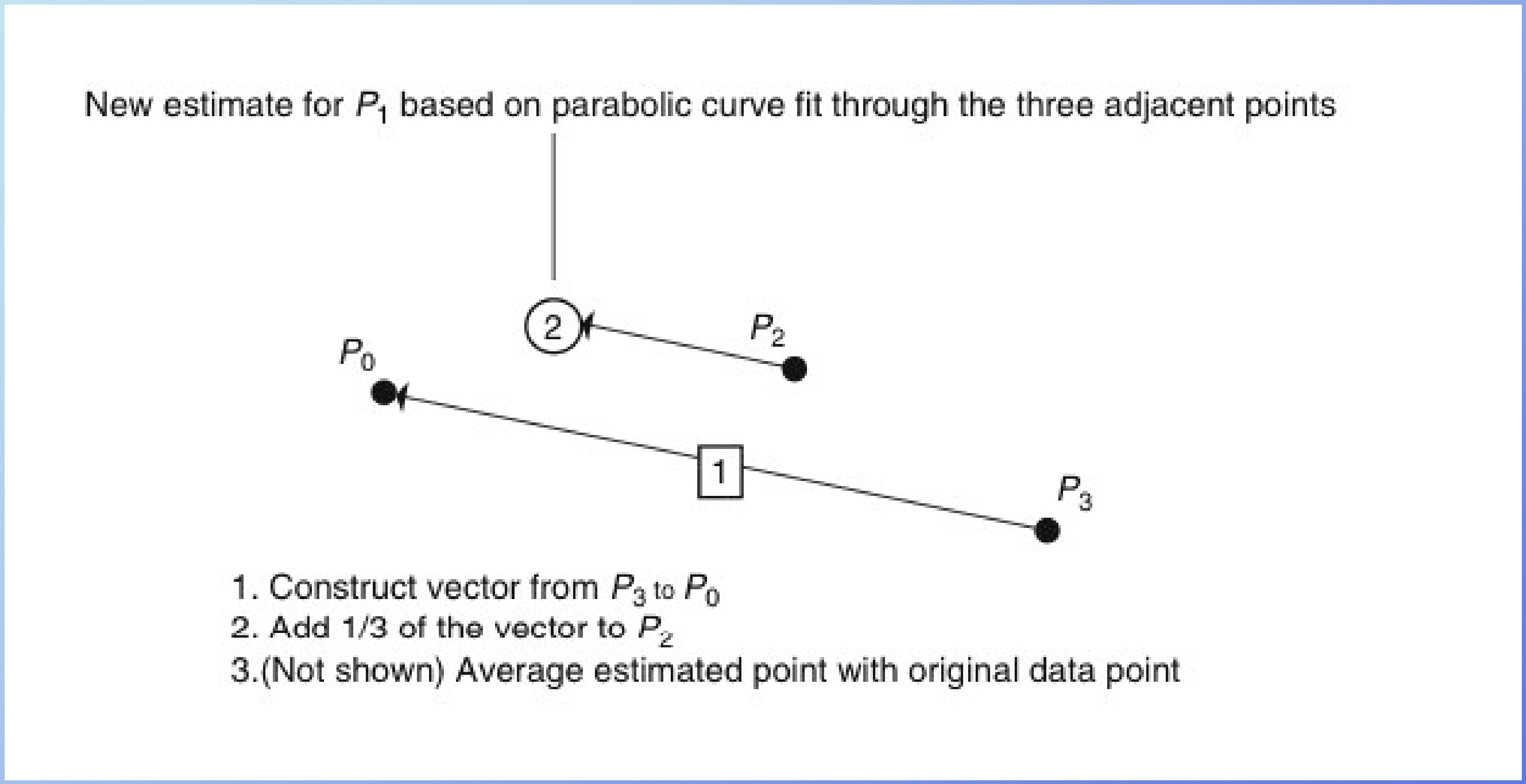


Smoothing data

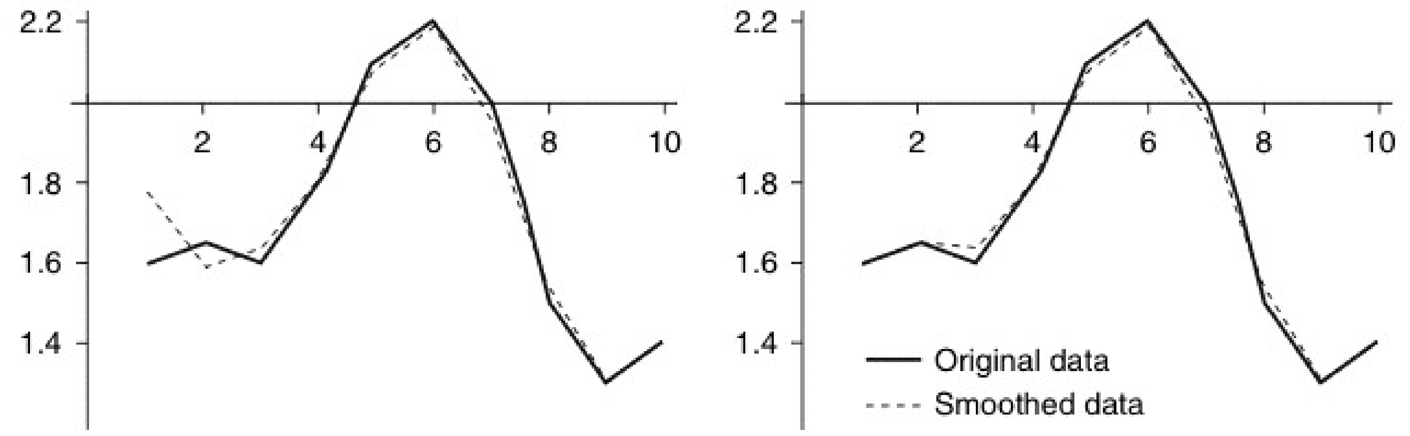




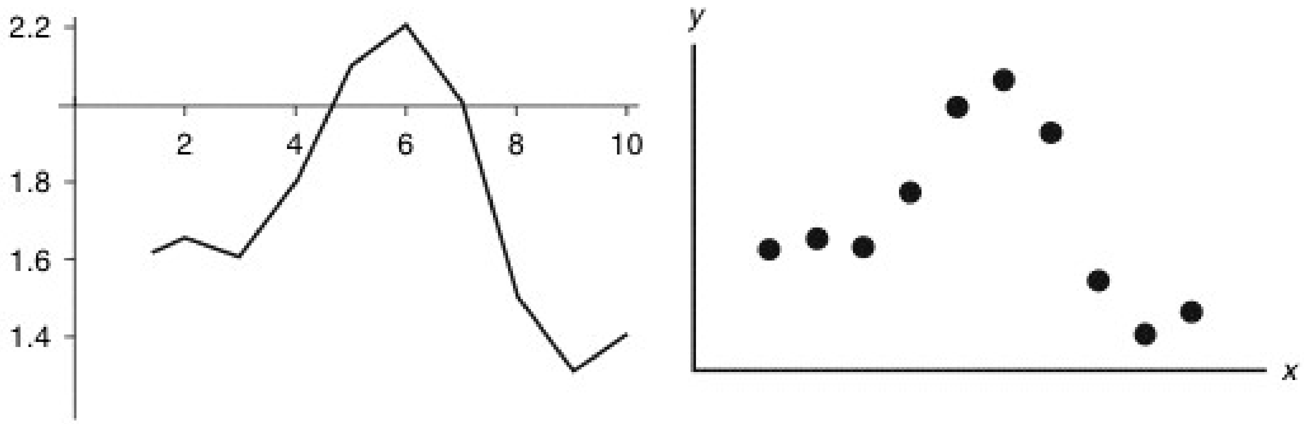
Smoothing data



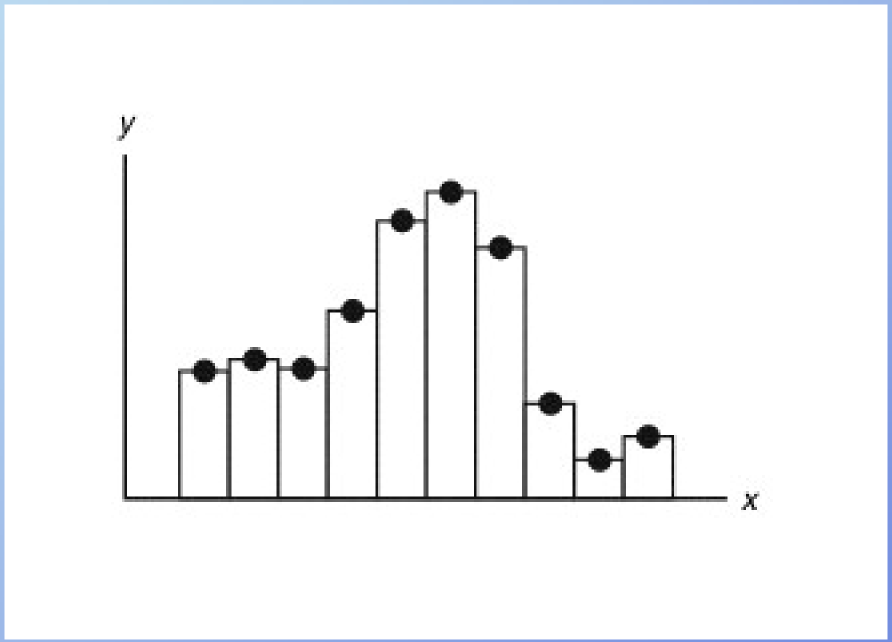
Smoothing data



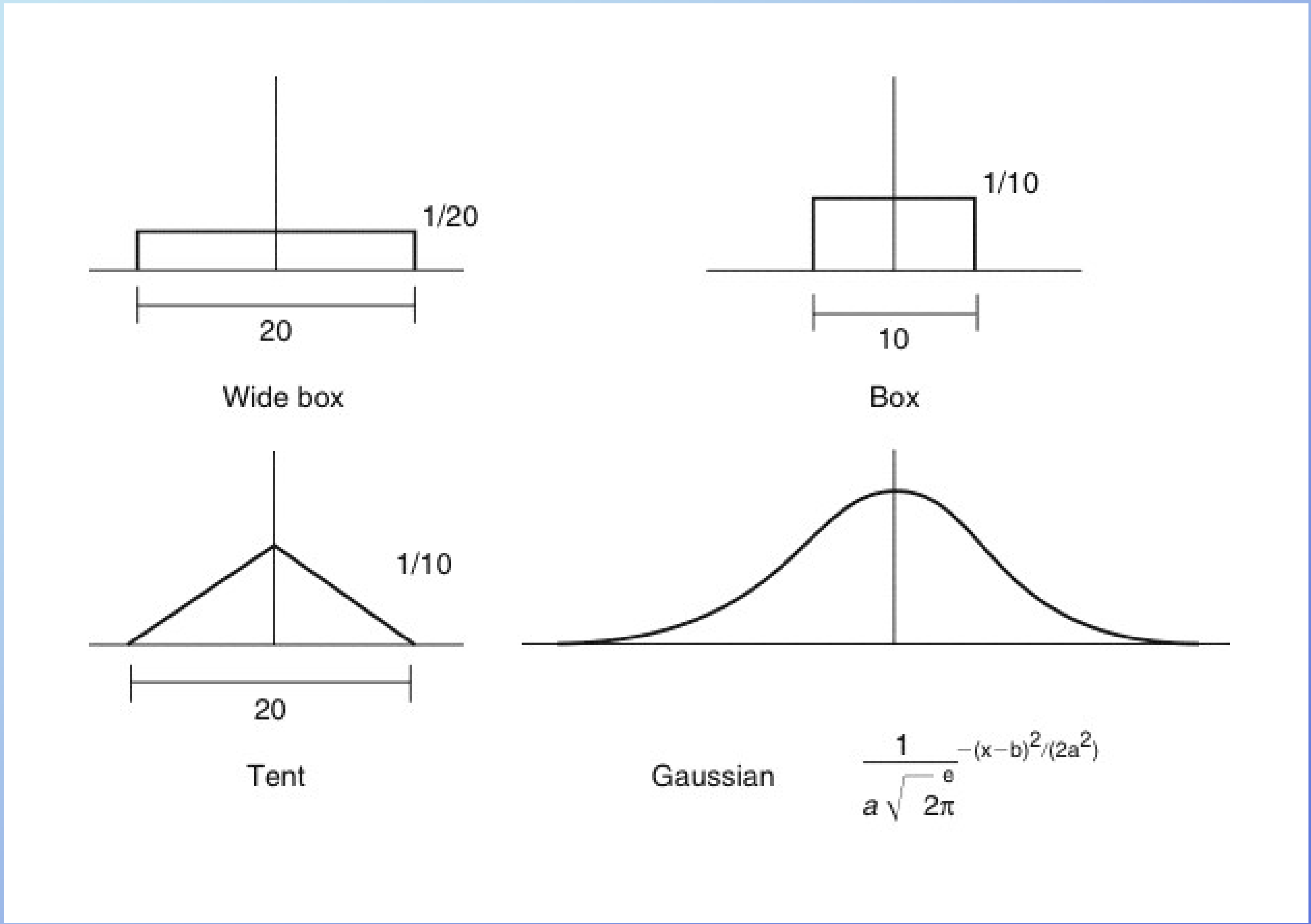
Smoothing data



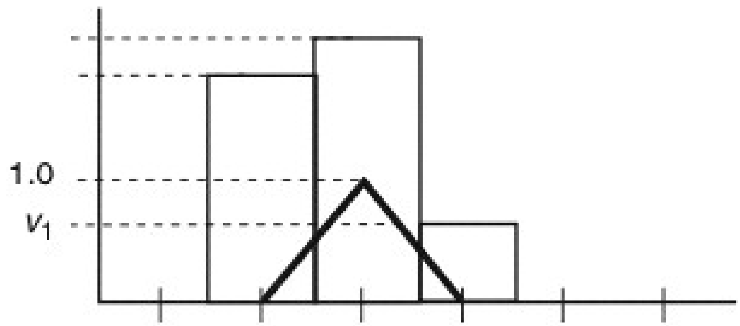
Smoothing data



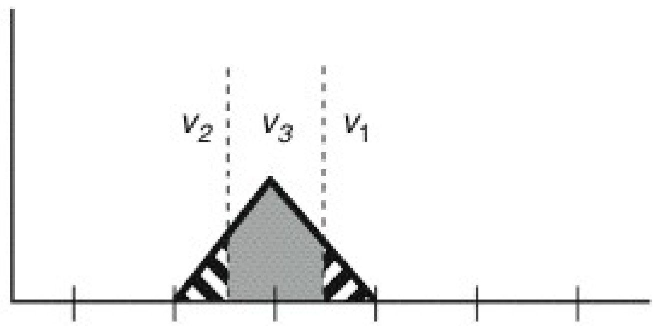
Smoothing data



Smoothing data



Smoothing kernel superimposed over step function

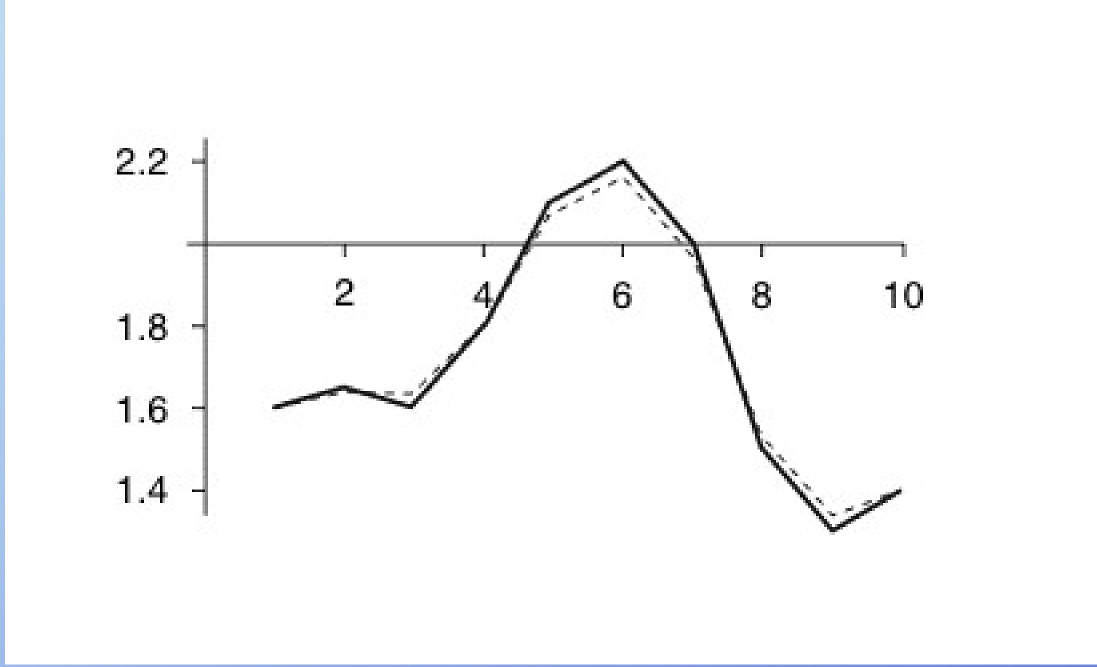


Areas of tent kernel under the different step function values

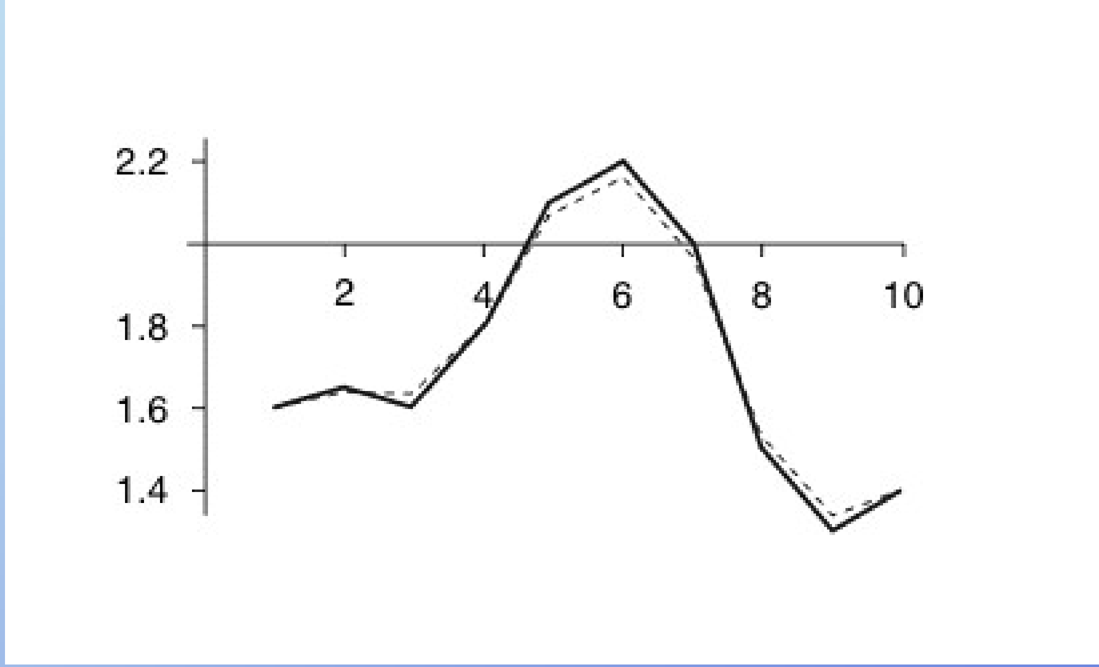
$$V = \frac{1}{8}v_1 + \frac{3}{4}v_2 + \frac{1}{8}v_3$$

Computation of value smoothed by applying area weights to step function values

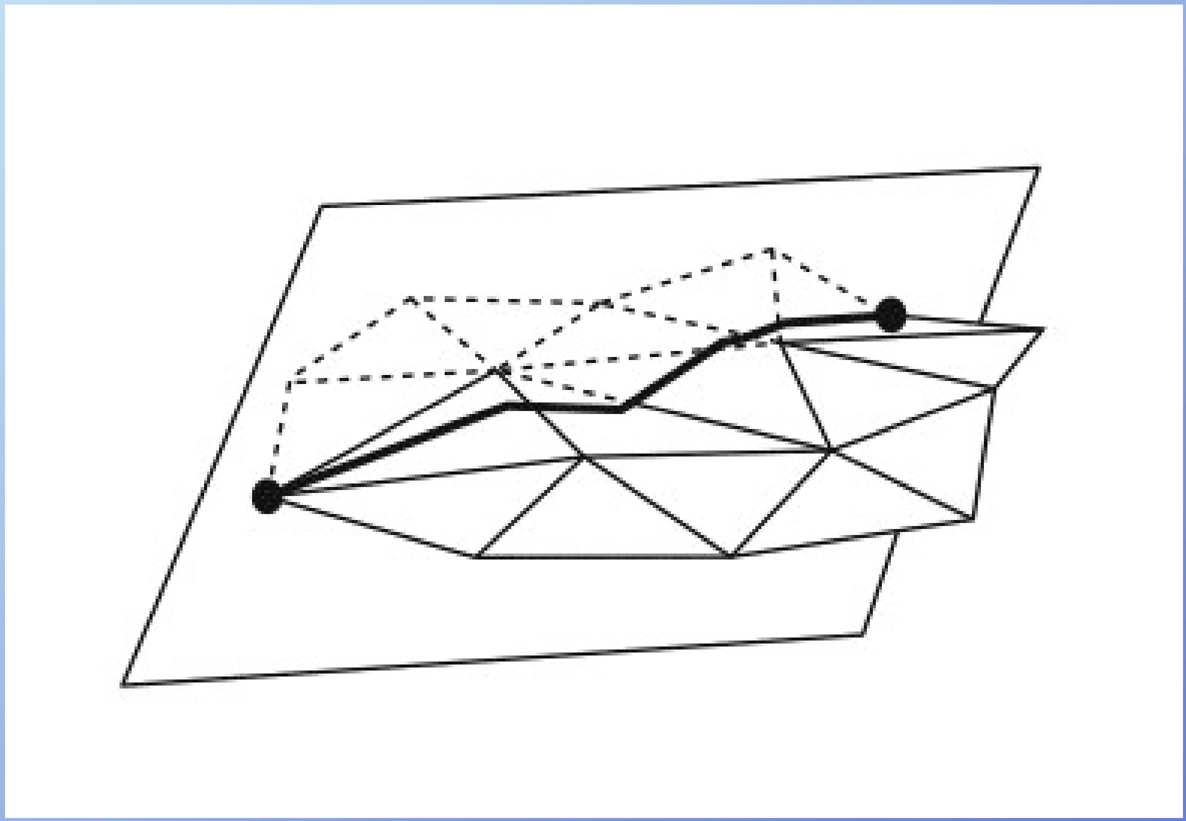
Smoothing data



Smoothing data



Path finding



Path finding - downhill