Keyframing

CS 448D: Character Animation
Prof. Vladlen Koltun
Stanford University

Keyframing in traditional animation

- Master animator draws key frames
- Apprentice fills in the in-between frames

Keyframing in computer animation

- Animator specifies object state for time t_i, for all i
- State for intermediate frames is computed by interpolation
- State can include:
  - Position
  - Orientation
  - Material properties
  - Many other things

Key values

- Not all parameters are specified for all key frames
- A key frame is only “key” for a subset of parameters

How do we interpolate?

- Depends on type of parameter
- This lecture: Position
- Orientation has issues, will be covered later

Polynomial interpolation

Theorem: Any (n+1) distinct points can be interpolated by a polynomial of degree n.

Given $(x_0, y_0), (x_2, y_2), \ldots, (x_n, y_n)$

there is a polynomial

$$p(x) = a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} + \ldots + a_n$$

such that

$$p(x_i) = y_i$$

Polynomial interpolation

$$
\begin{aligned}
y_0 &= a_0 {x_0}^n + a_1 {x_0}^{n-1} + a_2 {x_0}^{n-2} + \ldots + a_n \\
y_1 &= a_0 {x_1}^n + a_1 {x_1}^{n-1} + a_2 {x_1}^{n-2} + \ldots + a_n \\
\vdots & \quad \vdots \\
y_n &= a_0 {x_n}^n + a_1 {x_n}^{n-1} + a_2 {x_n}^{n-2} + \ldots + a_n
\end{aligned}
$$

Polynomial interpolation

$$\begin{pmatrix} {x_0}^n & {x_0}^{n-1} & \dots & 1 \\ {x_1}^n & {x_1}^{n-1} & \dots & 1 \\ \vdots & \vdots & \dots & \vdots \\ {x_n}^n & {x_n}^{n-1} & \dots & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Linear system. Solve (Gaussian elimination, LU decomposition).
Gives the desired polynomial.

$$p(x) = a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} + \ldots + a_n$$

Polynomial interpolation

- What happens in three dimensions?
- Express $$(x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)$$

  as $$(x(t_1), y(t_1), z(t_1)), \ldots, (x(t_n), y(t_n), z(t_n))$$

- Compute the polynomials x(t), y(t), and z(t)
- In dealing with position interpolation, we will sometimes discuss only the univariate case, knowing that all methods generalize to interpolating position in higher dimensions.

Lagrange interpolation

- Need to interpolate

$$(x_0, y_0), (x_2, y_2), \ldots, (x_n, y_n)$$

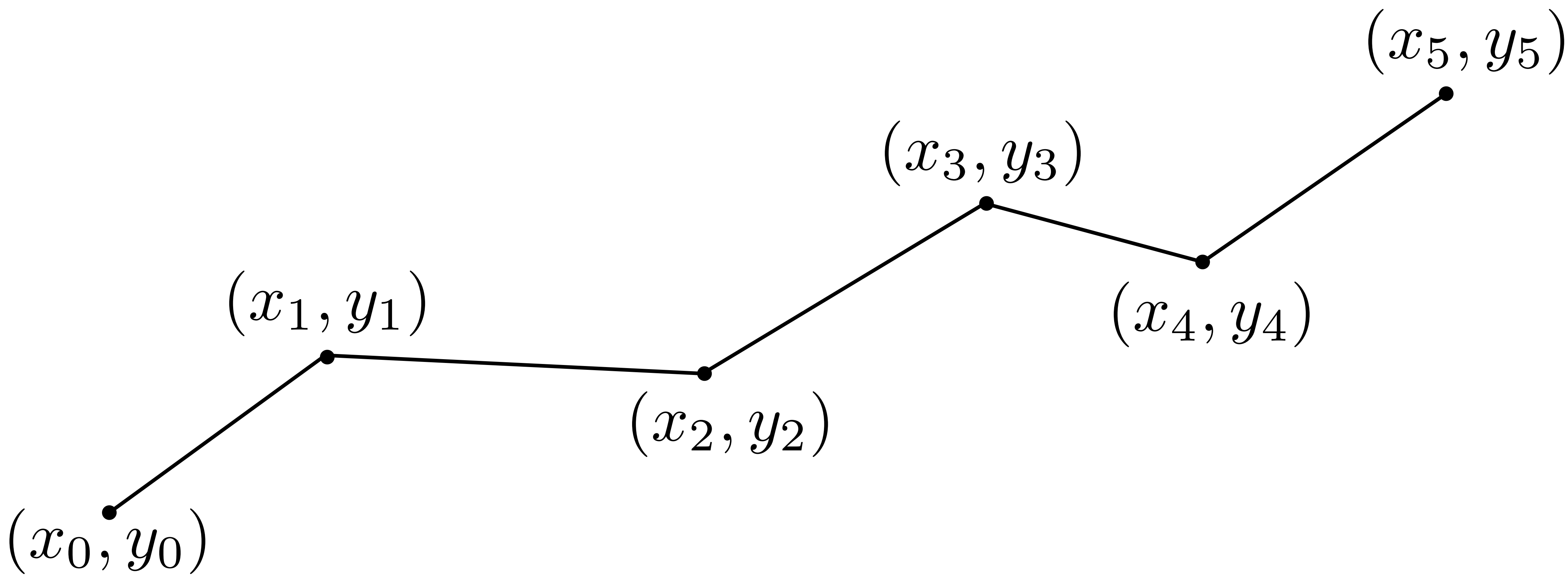
- Express p(x) as a linear combination of (n+1) basis polynomials $\mathcal{L}_i$, such that $\mathcal{L}_i(x_i) = 1$ and $\mathcal{L}_i(x_j) = 0$ for all $j \neq i$
- If we can find such $\mathcal{L}_i$, we can set $p(x) = \sum_{i=0}^{n} y_i \mathcal{L}_i(x)$
- Set

$$\mathcal{L}_i(x) = \prod_{0 \leq j \leq n,\ j \neq i} \frac{x - x_j}{x_i - x_j}$$

Global versus local interpolation

- These were global interpolation methods
- Computationally expensive. Potentially unstable numerically. A local change of an input point triggers a complete re-computation.
- Unweildy for animators, who want to be able to make local manipulations.
- Local interpolation methods connect input points with polynomial arcs

Linear interpolation

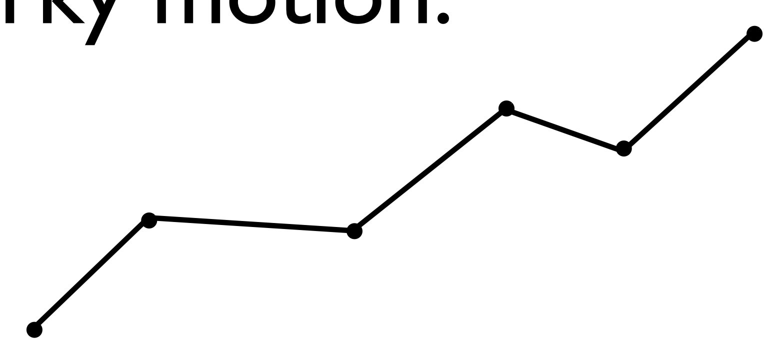


Interpolate between (x_i, y_i) and (x_{i+1}, y_{i+1}) with

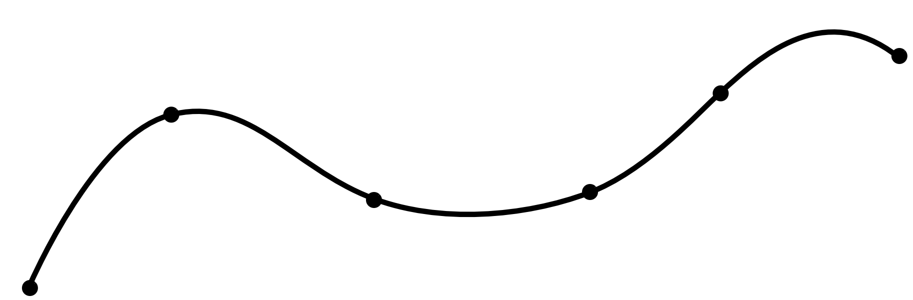
$$p_i(x) = y_i + \frac{x - x_i}{x_{i+1} - x_i}(y_{i+1} - y_i)$$

Orders of continuity

- C^n continuity: The n-th derivative is continuous.
- Linear interpolation provides C^0 continuity. Continuous but potentially jerky motion.

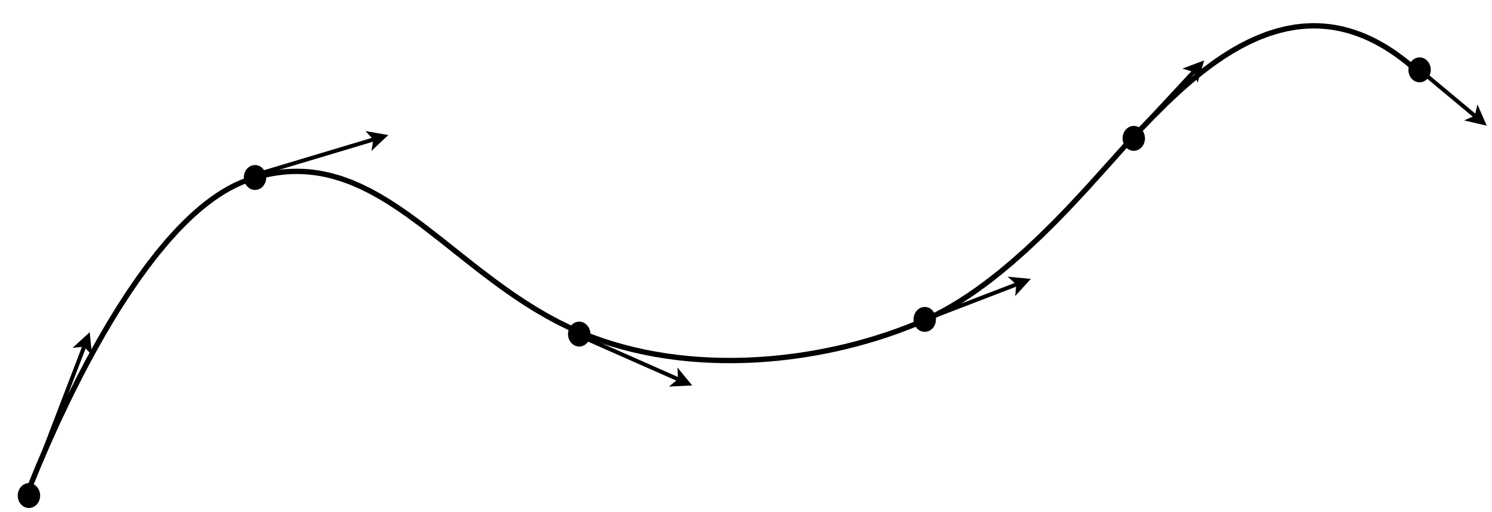


- Want to achieve at least C^1, and sometimes C^2 continuity.

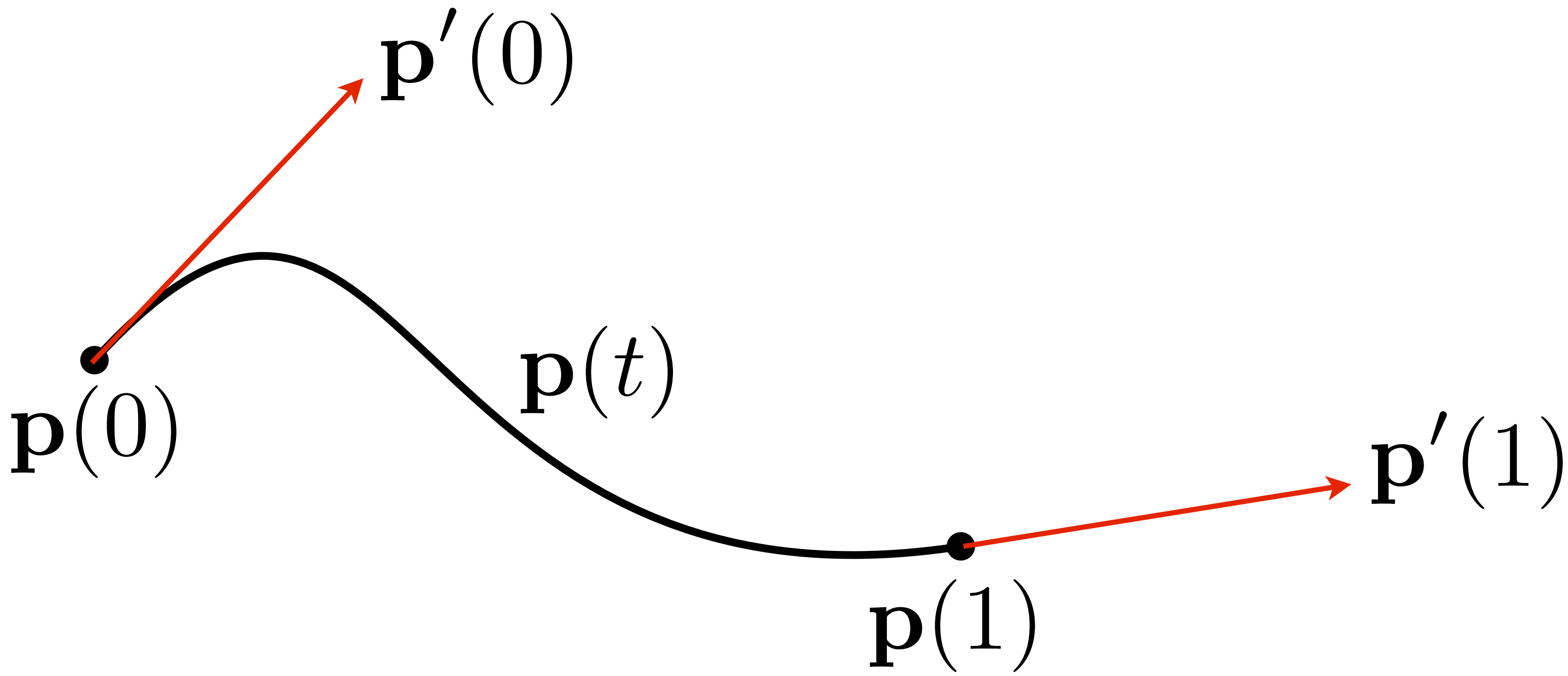


Hermite interpolation

- How do we achieve C^1 continuity and local control?
- We enforce shared tangents at control points and connect consecutive input points with polynomial arcs subject to the positional and tangential constraints at the endpoints.



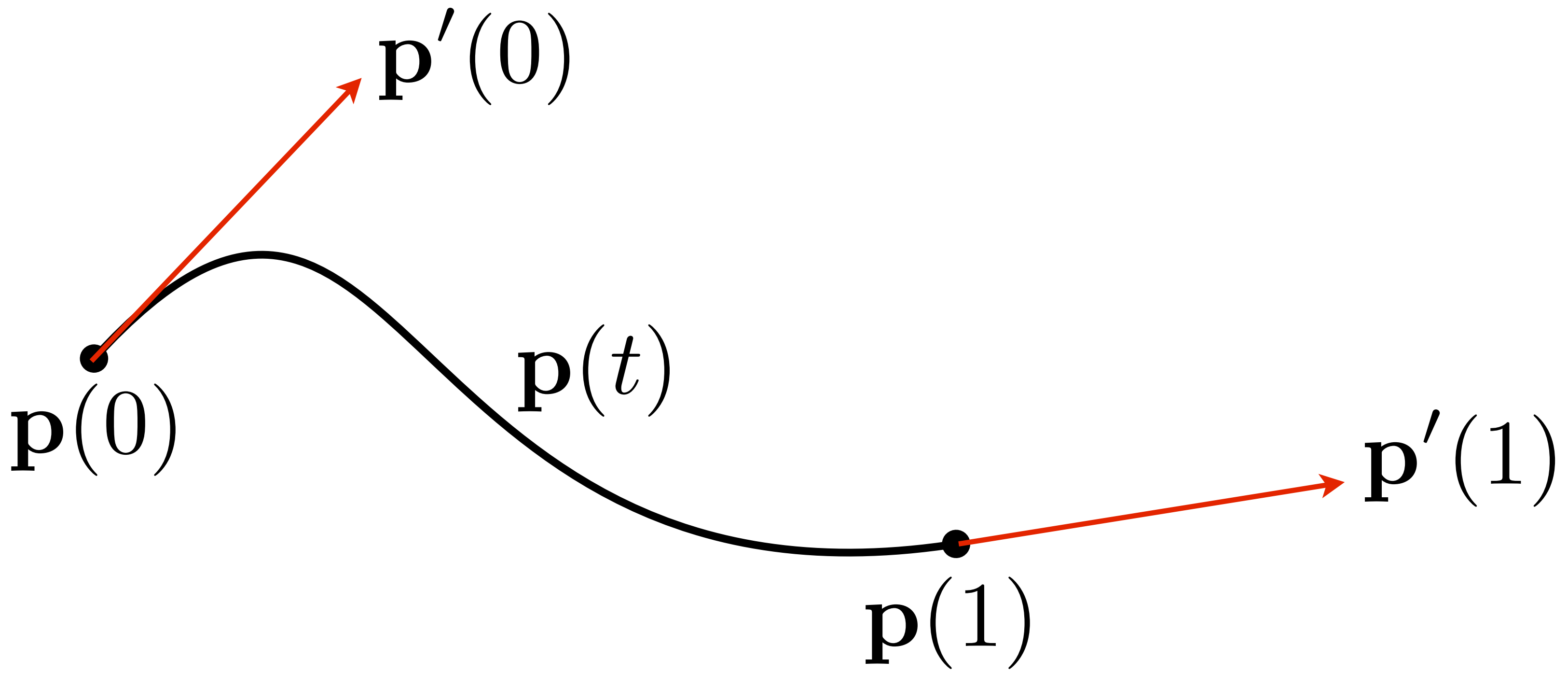
Hermite interpolation



Four linear equations that constrain the coefficients of p. How many coefficients do we need? Four. What is the degree of p? It's a cubic.

$$\begin{aligned}\mathbf{p}(t) &= a_0t^3 + a_1t^2 + a_2t + a_3 \\ \mathbf{p}'(t) &= 3a_0t^2 + 2a_1t + a_2\end{aligned}$$

Hermite interpolation



$$
\begin{aligned}
a_3 &= \mathbf{p}(0) \\
a_2 &= \mathbf{p}'(0) \\
a_0 + a_1 + a_2 + a_3 &= \mathbf{p}(1) \\
3a_0 + 2a_1 + a_2 &= \mathbf{p}'(1)
\end{aligned}
$$

Solve to obtain the coefficients.

Hermite interpolation

$$\mathbf{p}(t) \quad = \quad \begin{pmatrix} t^3 & t^2 & t & 1 \end{pmatrix} \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{p}(0) \\ \mathbf{p}(1) \\ \mathbf{p}'(0) \\ \mathbf{p}'(1) \end{pmatrix}$$

$$\mathbf{p}(t) = \begin{pmatrix} 3t^2 & 2t & 1 & 0 \end{pmatrix} \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{p}(0) \\ \mathbf{p}(1) \\ \mathbf{p}'(0) \\ \mathbf{p}'(1) \end{pmatrix}$$

Hermite interpolation

$$\mathbf{p}(t) = a_0 t^3 + a_1 t^2 + a_2 t + a_3$$

$$\mathbf{p}(t) = \mathcal{T}^{\mathrm{T}} M B$$

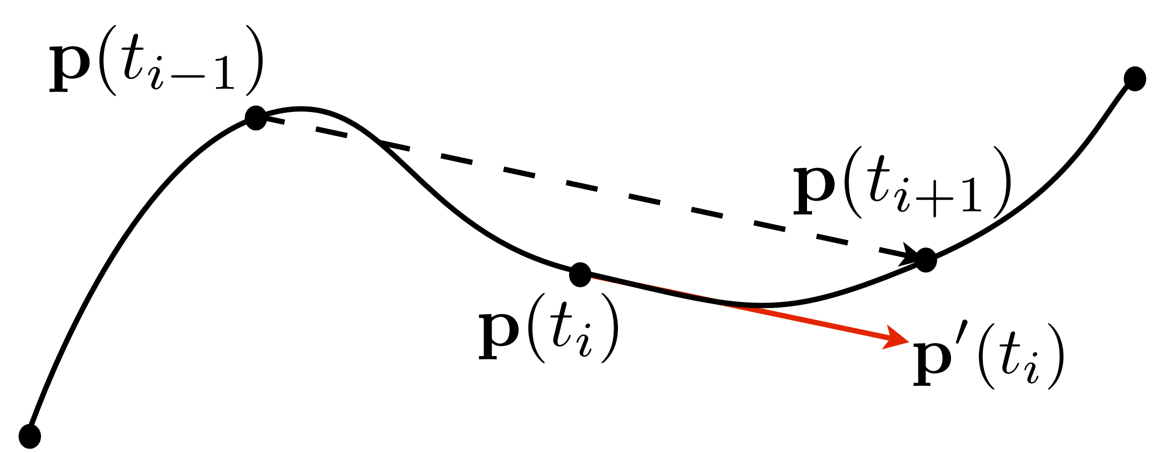
$$\mathcal{T} = \begin{pmatrix} t^3 & t^2 & t & 1 \end{pmatrix}$$

$$M = \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} \mathbf{p}(0) \\ \mathbf{p}(1) \\ \mathbf{p}'(0) \\ \mathbf{p}'(1) \end{pmatrix}$$

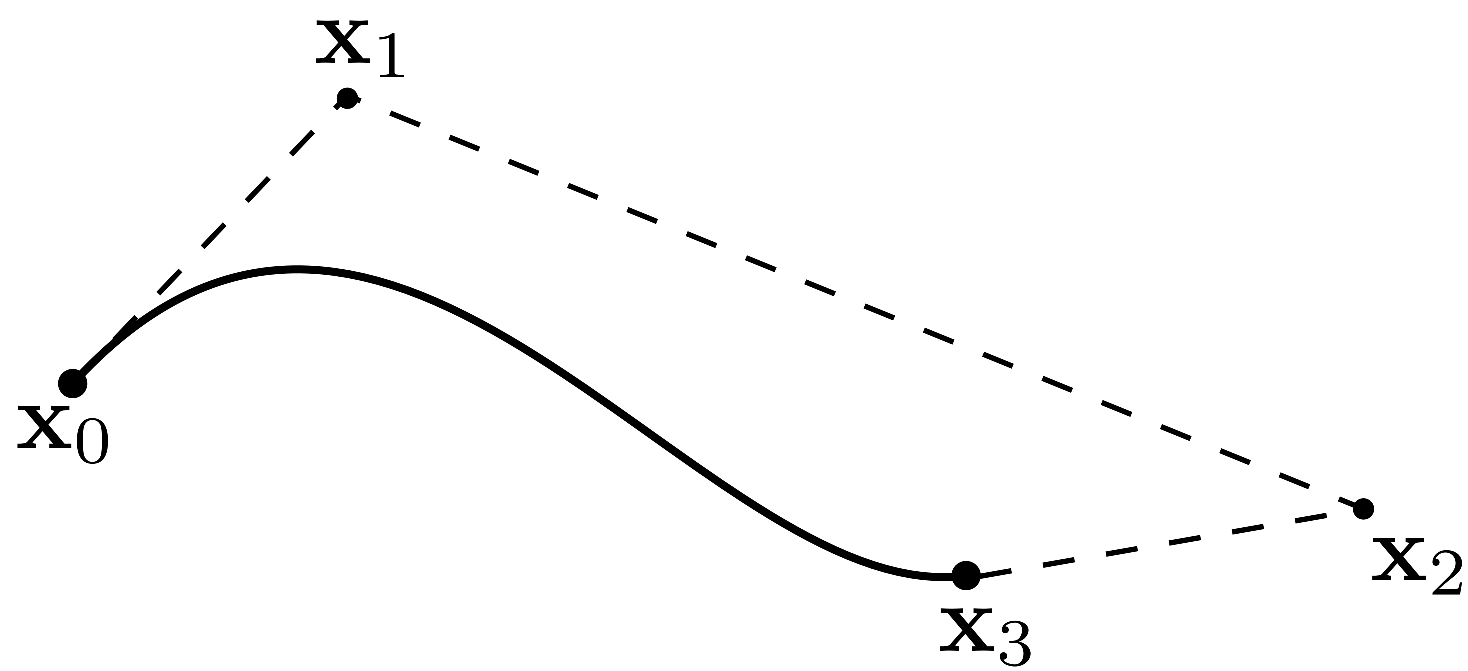
Catmull-Rom spline

- How do we get the tangents? Can be specified by the animator along with the control points, but this can be tedious and time-consuming.
- The Catmull-Rom idea:

$$\mathbf{p}'(t_i) = \frac{1}{2}\Big(\mathbf{p}(t_{i+1}) - \mathbf{p}(t_{i-1})\Big)$$



Bezier interpolation

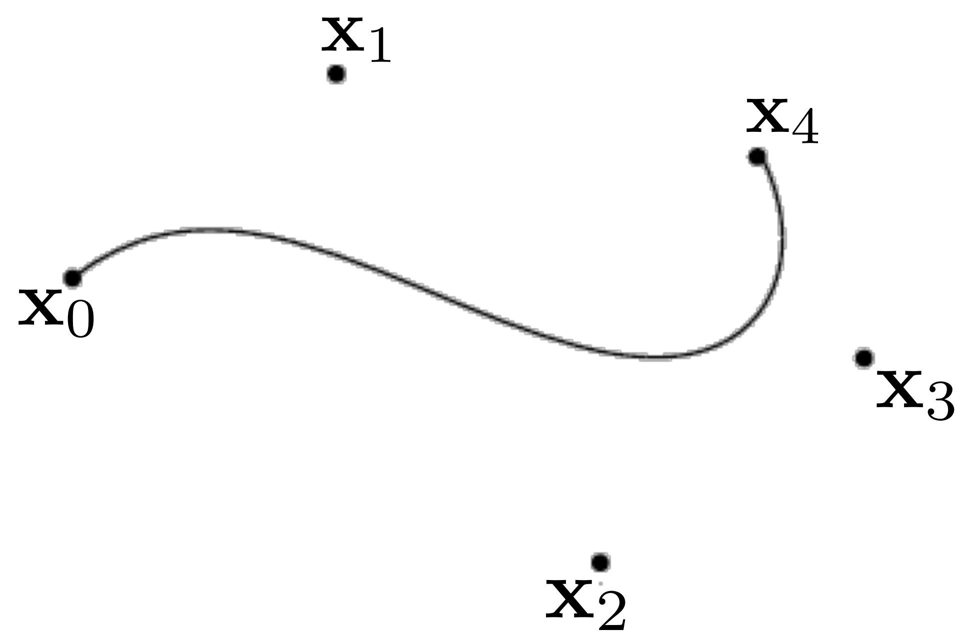


With two control points it's equivalent to Hermite interpolation.

$$p(t) = \mathcal{T}^{\mathrm{T}} M B \qquad \mathcal{T} = \left(t^3 \ t^2 \ t \ 1\right)$$

$$M = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix}$$

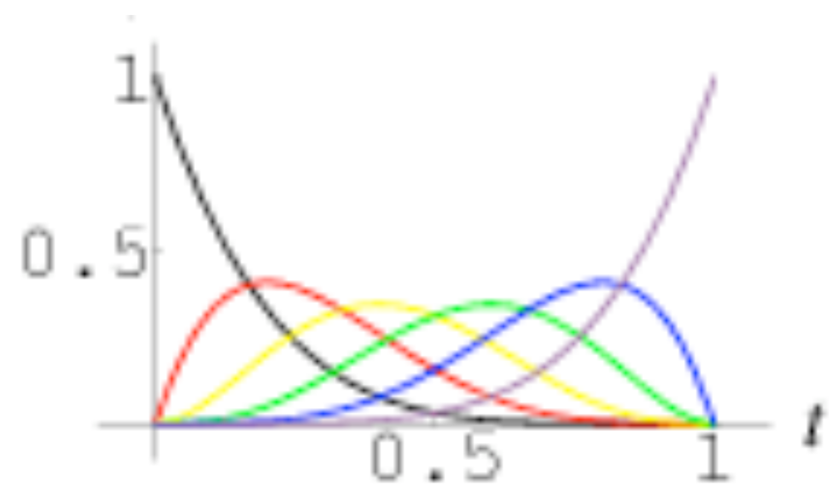
Diversion: Bezier curves



A Bezier curve can have any number of control points.

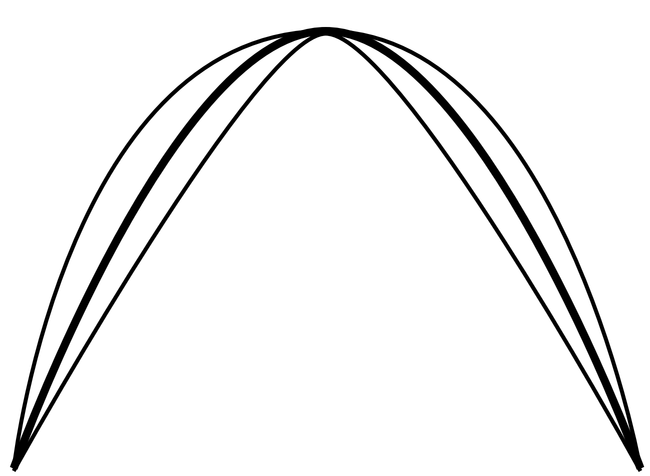
$$p(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{x}_i$$

Bernstein polynomials:

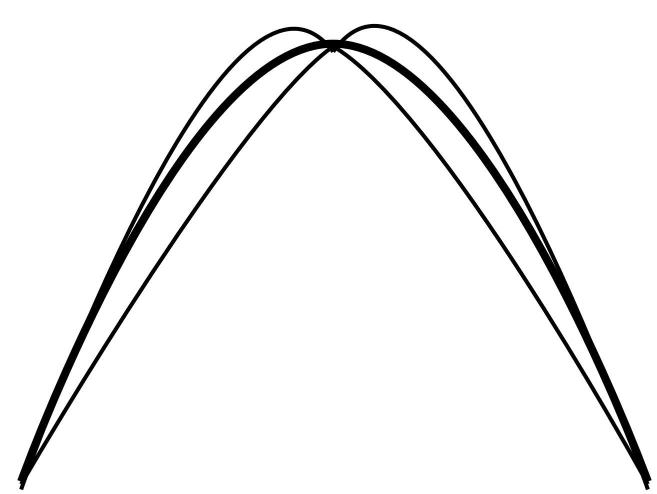


Kochanek-Bartels spline

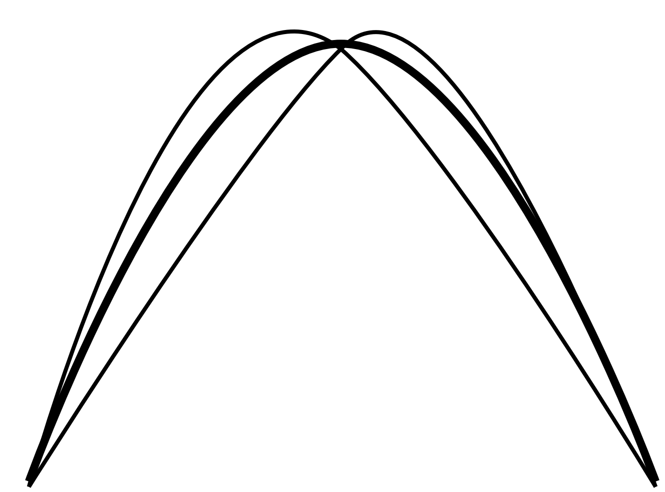
- Hermite lets us specify the tangents directly.
- Catmull-Rom completely automates the shape of the spline at the input points.
- Can we have some degree of control over the spline, but in a more intuitive way than direct tangent specification?
- Yes. Kochanek-Bartels gives us three intuitive degrees of freedom for the tangents: tension, continuity, and bias.



tension



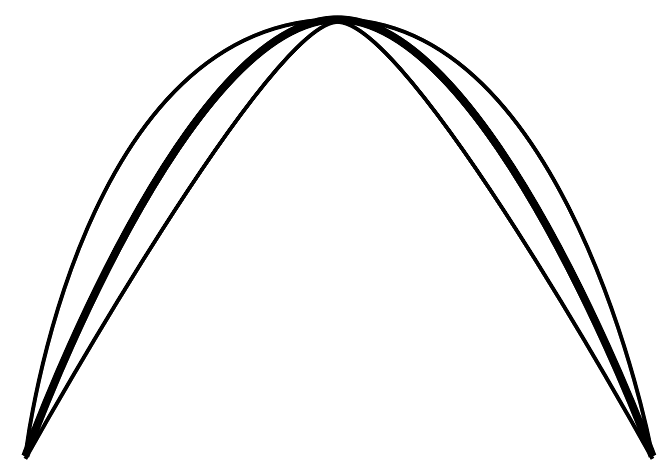
continuity



bias

Kochanek-Bartels spline

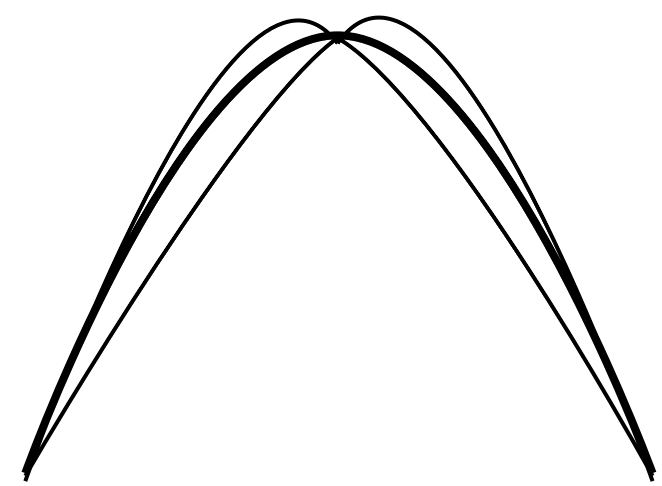
Tension



$$
\begin{aligned}
\mathbf{p}'_{\mathrm{left}}(t_i) &= \frac{1-T}{2}\Big(\mathbf{p}(t_i)-\mathbf{p}(t_{i-1})\Big)+\frac{1-T}{2}\Big(\mathbf{p}(t_{i+1})-\mathbf{p}(t_i)\Big)\\
\mathbf{p}'_{\mathrm{right}}(t_i) &= \frac{1-T}{2}\Big(\mathbf{p}(t_i)-\mathbf{p}(t_{i-1})\Big)+\frac{1-T}{2}\Big(\mathbf{p}(t_{i+1})-\mathbf{p}(t_i)\Big)
\end{aligned}
$$

Kochanek-Bartels spline

Continuity

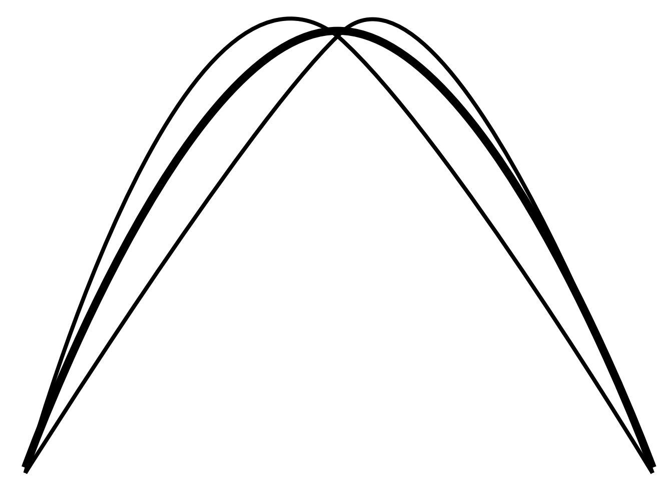


$$\mathbf{p}'_{\text{left}}(t_i) = \frac{1-C}{2}\Big(\mathbf{p}(t_i) - \mathbf{p}(t_{i-1})\Big) + \frac{1+C}{2}\Big(\mathbf{p}(t_{i+1}) - \mathbf{p}(t_i)\Big)$$

$$\mathbf{p}'_{\text{right}}(t_i) = \frac{1+C}{2}\Big(\mathbf{p}(t_i) - \mathbf{p}(t_{i-1})\Big) + \frac{1-C}{2}\Big(\mathbf{p}(t_{i+1}) - \mathbf{p}(t_i)\Big)$$

Kochanek-Bartels spline

Bias



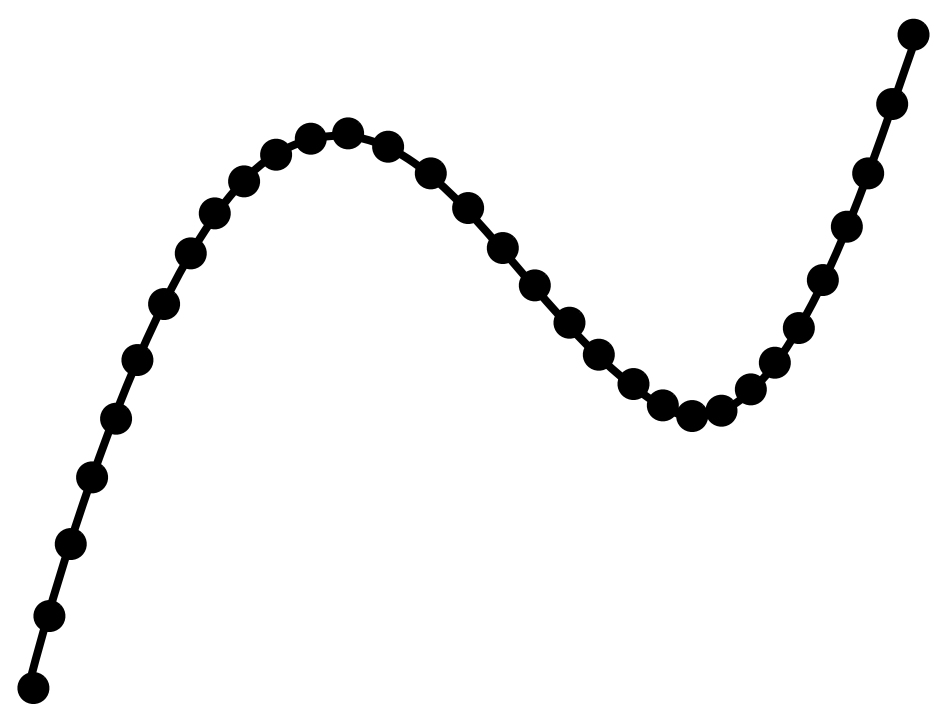
$$\begin{aligned}
\mathbf{p}'_{\text{left}}(t_i) &= \frac{1+B}{2}\Big(\mathbf{p}(t_i)-\mathbf{p}(t_{i-1})\Big)+\frac{1-B}{2}\Big(\mathbf{p}(t_{i+1})-\mathbf{p}(t_i)\Big)\\
\mathbf{p}'_{\text{right}}(t_i) &= \frac{1+B}{2}\Big(\mathbf{p}(t_i)-\mathbf{p}(t_{i-1})\Big)+\frac{1-B}{2}\Big(\mathbf{p}(t_{i+1})-\mathbf{p}(t_i)\Big)
\end{aligned}$$

Kochanek-Bartels spline

$$\begin{aligned}
\mathbf{p}'_{\text{left}}(t_i) &= \tfrac{(1-T)(1-C)(1+B)}{2}\big(\mathbf{p}(t_i) - \mathbf{p}(t_{i-1})\big) + \tfrac{(1-T)(1+C)(1-B)}{2}\big(\mathbf{p}(t_{i+1}) - \mathbf{p}(t_i)\big) \\
\mathbf{p}'_{\text{right}}(t_i) &= \tfrac{(1-T)(1+C)(1+B)}{2}\big(\mathbf{p}(t_i) - \mathbf{p}(t_{i-1})\big) + \tfrac{(1-T)(1-C)(1-B)}{2}\big(\mathbf{p}(t_{i+1}) - \mathbf{p}(t_i)\big)
\end{aligned}$$

Velocity Control

- We now have a parametric curve $\mathbf{p}(t)$ that smoothly interpolates keys. But we still have a problem: uncontrolled *velocity* of movement.

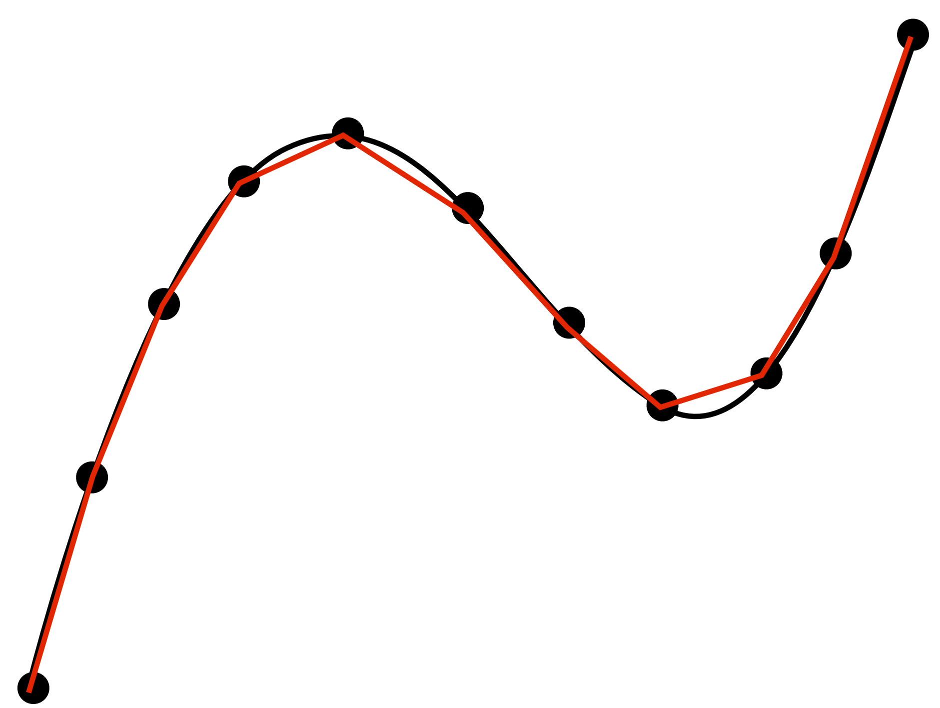


Reparameterization

- We want to be able to control the distance covered along the curve per unit of time. Need a function T that maps from normalized distance covered, s, to appropriate parameter value, t. Then $\mathbf{p}(T(s))$ will move at uniform velocity.
- We approximate T by approximating its inverse S that maps from parameter values, t, to distance covered, s.

Finite differencing

- Sample t uniformly and approximate **p**(t) by piecewise linear segments. Approximate S by normalized distance covered along the approximating curve.



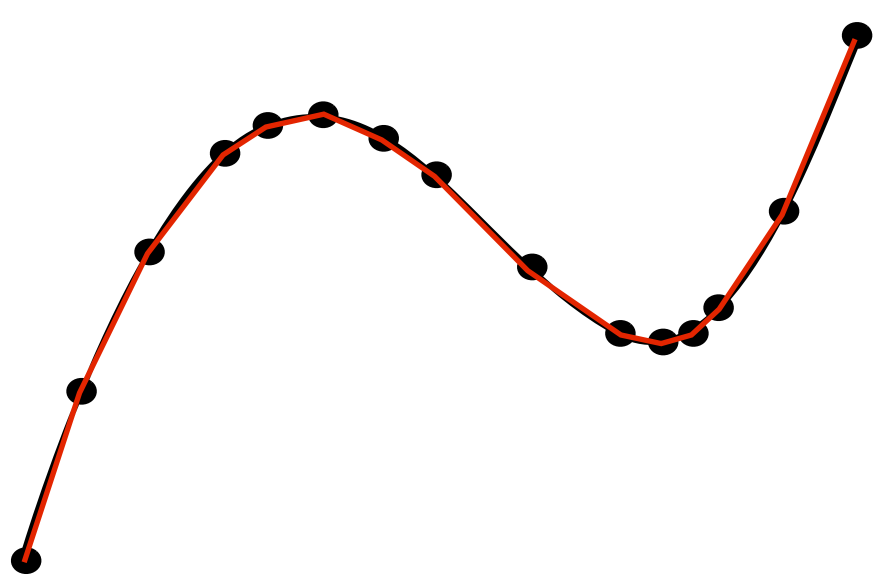
Adaptive finite differencing

Maintain a set of candidate curve segments. For each such segment (**p**(a),**p**(b)), if

$$\left| \left\| \frac{\mathbf{p}(a)+\mathbf{p}(b)}{2} - \mathbf{p}(a) \right\| + \left\| \mathbf{p}(b) - \frac{\mathbf{p}(a)+\mathbf{p}(b)}{2} \right\| - \|\mathbf{p}(b)-\mathbf{p}(a)\| \right| > \varepsilon$$

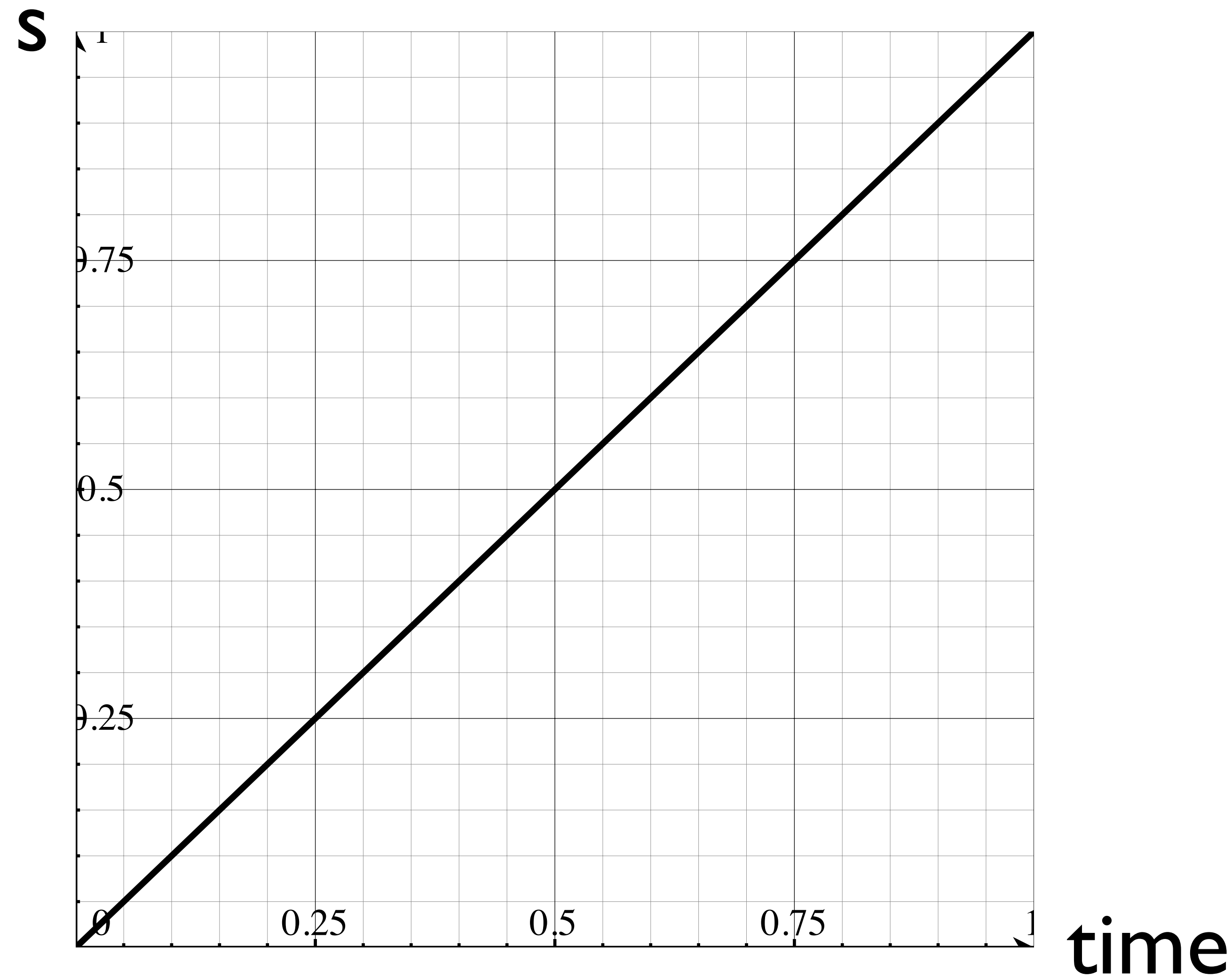
then replace $(\mathbf{p}(a), \mathbf{p}(b))$ with $\left(\mathbf{p}(a), \frac{\mathbf{p}(a)+\mathbf{p}(b)}{2}\right)$ and $\left(\frac{\mathbf{p}(a)+\mathbf{p}(b)}{2}, \mathbf{p}(b)\right)$

and iterate until no segments need to be broken up.



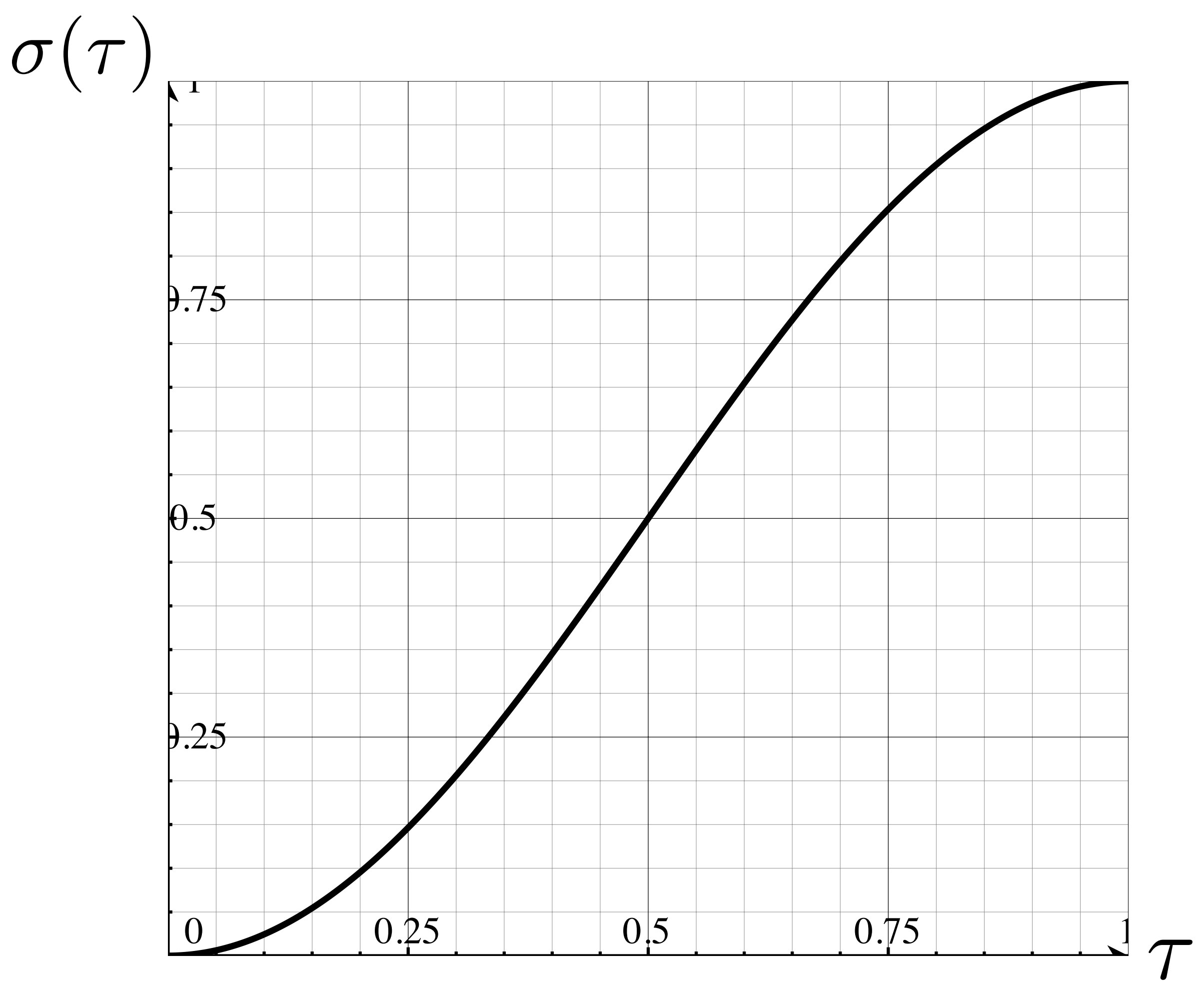
Velocity control

We can now produce uniform velocity motion along the curve by approximating $S(t)$, computing the resulting $T(s) = S^{-1}(t)$, and moving along $\mathbf{p}(T(s))$ as s increases uniformly from 0 to 1.



Velocity control

We can also drive the motion along the curve in more general ways, with the distance covered being a non-uniform function $\sigma(\tau)$ of time. Then the motion can be expressed as $\mathbf{p}(T(\sigma(\tau)))$. One example is the ease-in/ease-out behavior:



$$\sigma(\tau) = \frac{\sin\left(\tau\pi - \frac{\pi}{2}\right) + 1}{2}$$