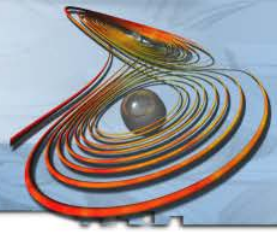


Simulation and Animation

3. Interpolation

Reinhold Preiner

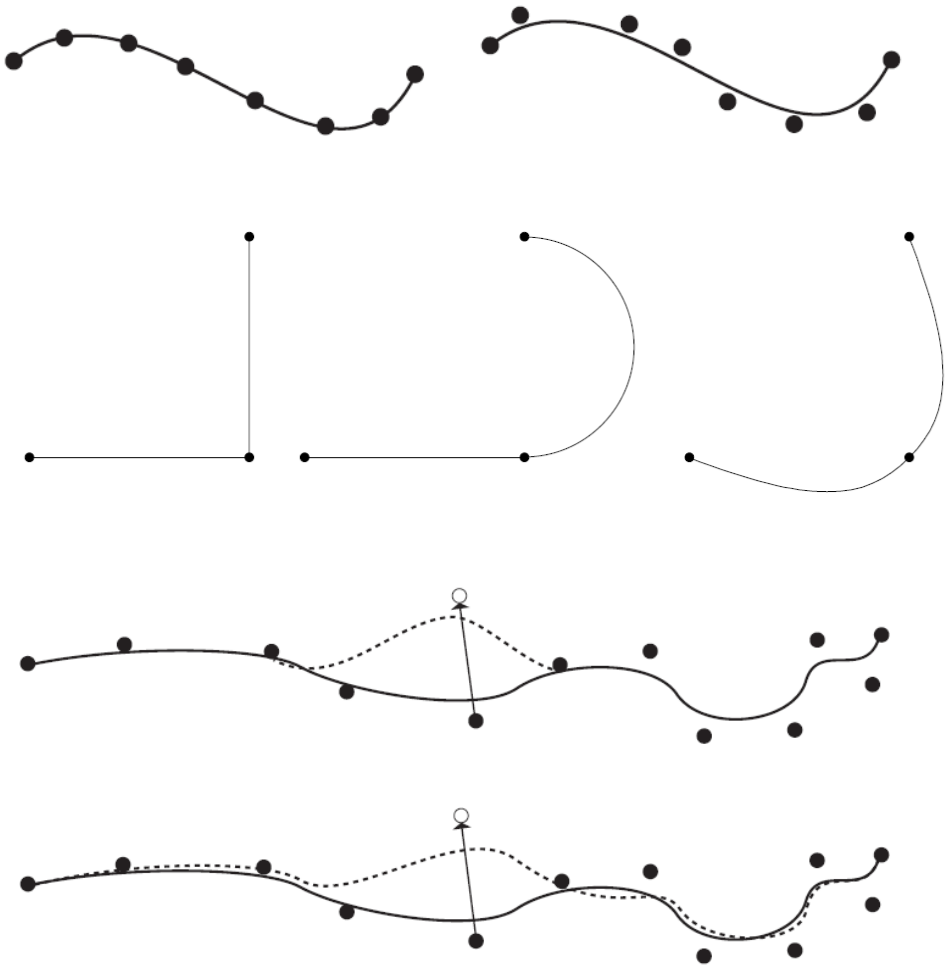
SS 2025

Key frame Animation

- Each key frame has a list of values, **articulation variables (avars)**, associated with a given parameter, i.e.
  - a coordinate of the position of an object
  - a joint angle of an appendage of a robot
  - the transparency attribute of an object
  - the speed of an object
  - ...
- Interpolation: Values for the parameter of interest must be generated for all of the frames between the key frames.

Types of Interpolation

- interpolation versus approximation
- continuity
- local control versus global control
- complexity



Interpolation

Many low degree **Polynomials** are used to interpolate a series of points. Polynomials can be pieced together to form spline curves.

- Linear Interpolation
- Lagrange Interpolating Polynomial
- Hermite Interpolating Polynomial
- Catmull-Rom Interpolating Polynomial
- Bezier spline

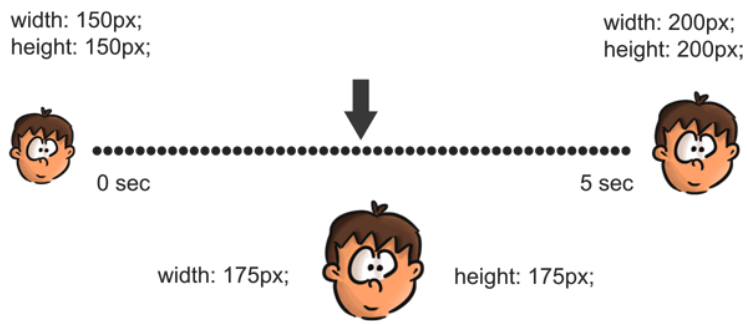
Cubic piecewise polynomials are the most common.

Linear Interpolation

Linear interpolation can be used to interpolate between two data points:

$$P(t) = P_0 + \frac{t - t_0}{t_1 - t_0}(P_1 - P_0)$$

- Easy to implement.
- Does not give smooth curves when interpolating more than two key frames.



Lagrange Interpolating Polynomial

The **Lagrange** interpolating polynomial is the polynomial $P(x)$ of degree $\leq n$ that passes through the $(n+1)$ points $P_0, \ldots, P_n$ given by (x_0, y_0), (x_1, y_1), $\ldots$, (x_n, y_n), and is given by

$$P(x) = \sum_{i=0}^{n} y_i L_i^n(x)$$

where

$$L_i^n(x) = \frac{(x - x_0)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_n)}{(x_i - x_0)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_n)}.$$

Lagrange Interpolating Polynomial

- The curve interpolates given points P_i.

- No local control.
- The degree is dependent on the number of data points interpolated.
- Higher degree will lead to oscillations between data points.
- Difficult to construct smooth spline curves.

Hermite Interpolation

The **Hermite** spline is a piecewise polymonial curve defined by end points P_i together with the corresponding derivative vectors $P_i', P_i'', \ldots$
so that

$$p(t) = P_i H_0^n(t) + P_i' H_1^n(t) + \cdots + P_{i+1}' H_{n-1}^n(t) + P_{i+1} H_n^n$$

where $H_i^n(t)$ are the Hermite interpolating polynomials of degree n.

Hermite Interpolation

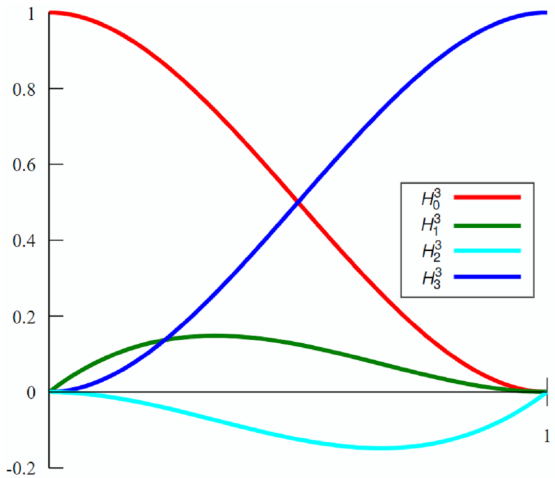
Example:

Cubic Hermite Interpolation

$$P(t) = P_i H_0^3(t) + P'_i H_1^3(t) + P'_{i+1} H_2^3(t) + P_{i+1} H_3^3(t)$$

The cubic Hermite basis functions are

$$H_0^3(t) = (1-t)^3 + 3t(1-t)^2$$
$$H_1^3(t) = t(1-t)^2$$
$$H_2^3(t) = -t^2(1-t)$$
$$H_3^3(t) = 3t^2(1-t) + t^3.$$



We thus have

$$P(t) = (2t^3 - 3t^2 + 1)P_i + (t^3 - 2t^2 + t)P'_i + (t^3 - t^2)P'_{i+1} + (-2t^3 + 3t^2)P_{i+1}$$

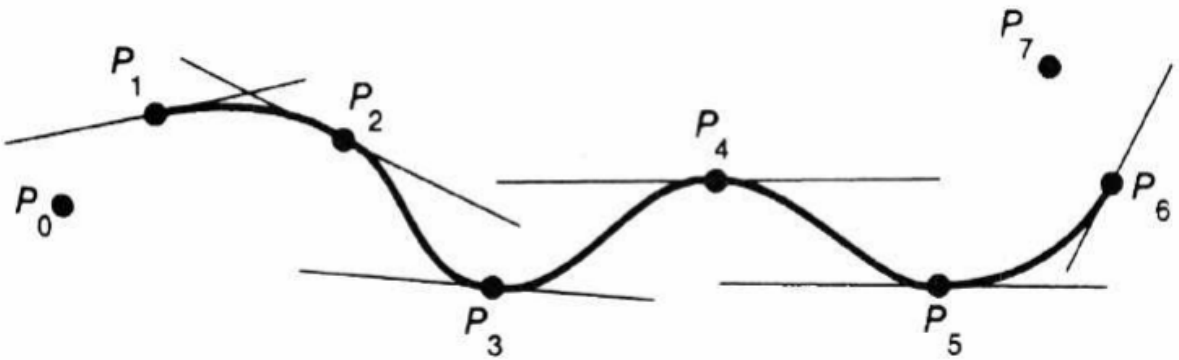
Hermite Interpolation

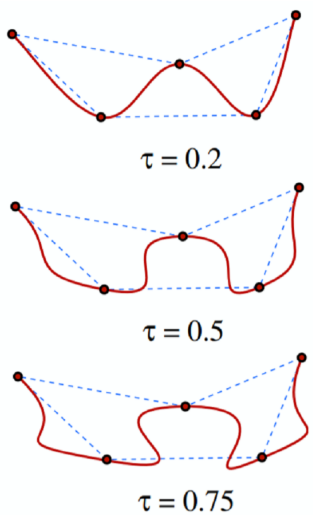
- Easy to construct piecewise polynomial curves with smooth joins:
  - Hermite formula is applied to each interval separately.
  - Resulting spline will be continuous and will have continuous first derivative
- Derivative information needs to be provided.
  - If only values are provided, the derivatives must be estimated from them.

Catmull-Rom Interpolating Polynomial

Catmull-Rom splines are a family of cubic interpolating splines (similar to Hermite)

- Require four points at a time: P_i and P_{i+1}, and uses P_{i-1} and P_{i+2} to derive derivative information at points P_i and P_{i+1} respectively.
- The tangent at each point P_i is calculated using $\tau(P_{i+1} - P_{i-1})$.
- The tension parameter, τ, affects how sharply the curve bends at the interpolated points. It is typically set to $1/2$.





Catmull-Rom Interpolating Polynomial

The geometry matrix is given by

$$P(t) = (1\ t\ t^2\ t^3) \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau-3 & 3-2\tau & -\tau \\ -\tau & 2-\tau & \tau-2 & \tau \end{bmatrix} \begin{bmatrix} P_{i-2} \\ P_{i-1} \\ P_i \\ P_{i+1} \end{bmatrix}$$

Setting $\tau = 1/2$ this simplifies to

$$P(t) = \frac{1}{2}(1\ t\ t^2\ t^3) \begin{bmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} P_{i-2} \\ P_{i-1} \\ P_i \\ P_{i+1} \end{bmatrix}$$

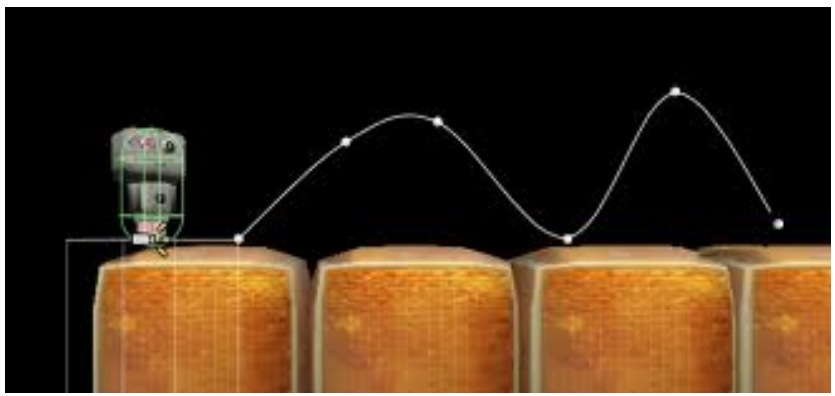
```
 float catmullrom(float t, float p0, float p1,
float p2, float p3)
{
  return 0.5f * (
       (2 * p1) +
       (-p0 + p2) * t +
       (2 * p0 - 5 * p1 + 4 * p2 - p3) * t * t +
       (-p0 + 3 * p1 - 3 * p2 + p3) * t * t * t
       );
}
```

- Here, $\tau = 1/2$
- Returns a point $P(t)$ between P_1 and P_2.

Catmull-Rom Interpolating Polynomial

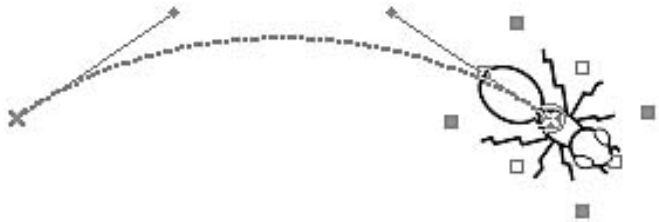
- Catmull-Rom splines are C^1 continuous.
- All points are interpolated.
- Offer local control.

- Spline curves do not lie within their convex hull of their control points.



Cubic Bézier Splines

$$P(t) = \sum_{i=0}^{3} P_i B_{i,3}(t),$$



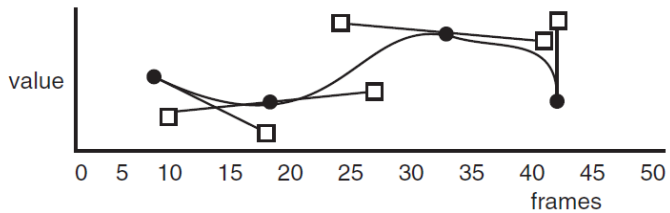
where $B_{i,3}$ are degree 3 Bernstein polynomials,
has properties:

- $P(0) = P_0$ and $P(1) = P_3$
- $P(t)$ is continuous, and the 1st and 2nd derivatives are continuous, C^2.
- Tangent vector at $t = 0$ is the line $\overline{P_0 P_1}$.
  The tangent to the curve at $t = 1$ is the line $\overline{P_2 P_3}$.
- Splines can be designed to be C^1 continuous.
- Within each spline section, the curve lies in the convex hull of it's control points.
- Have a nice geometric construction.

Interpolation

Systems based on interpolating key values (avars) at key frames are referred to as **track based**.

It is common for such systems to provide an interactive interface with which the animator can specify the key values and can control the interpolation curve by manipulating tangents or interior control points.



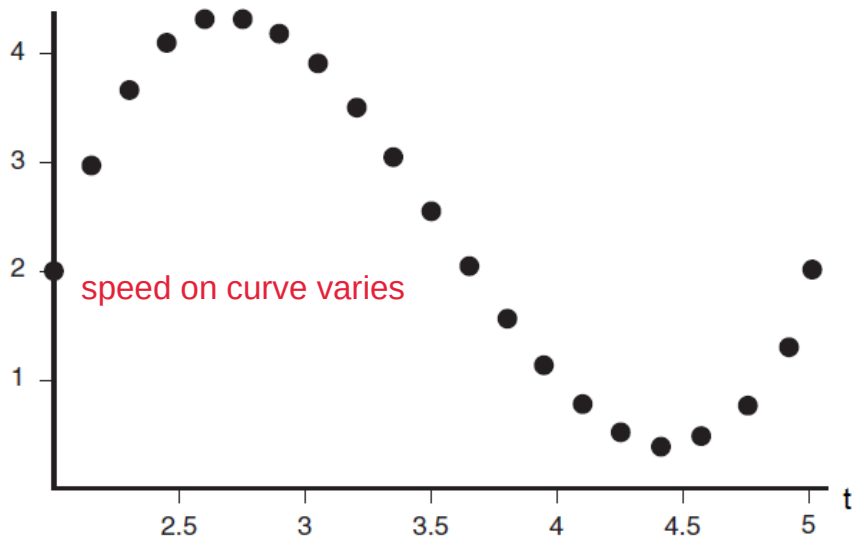
F-Curves in Blender:
https://docs.blender.org/manual/en/latest/editors/graph_editor/fcurves/introduction.html

Controlling the motion of a point along a curve

The animator needs to have control of

- the **shape** of a curve.
- the **speed** at which the curve is traced out.

  Speed is in unit of distance per unit time. For a point moving along a curve the distance traveled is the length of the curve.



changes at constant 'parametric' speed

Controlling the motion of a point along a curve

- The **space curve** defines the path taken.
  - The interpolating function relates parametric value to position on the space curve.

- The **distance-time function** relates time to a position on the space curve.
  1. The distance along a curve is referred to as **arc length**, *s*. The relationship between **arc length** and parametric value is established with the **arc length parameterization** of the space curve.
  2. The animator needs to specify the distance the object should move along the curve for each time step.

Arc Length Parameterization

Arc length parameterization of the curve allows

- movement along the curve at a constant speed
  by evaluating the curve at equal arc length intervals.
- acceleration and deceleration along the curve
  by controlling the distance traveled in a given time interval.

For a curve, parameterized by arc length, a unit change in the parameterizing variable results in a unit change in curve length. Many seemingly difficult problems in controlling the motion along a path become very simple.

Arc Length Parameterization

Any regular curve $P(t)$ can be parameterized by arc length.

To reparameterize a curve $P(t)$ by arc length, the procedure is

1. Find the arc length $s(t)$.
2. Calculate the inverse function $t(s)$, so that $s(t(s)) = s$.
3. Set the new parameter $u = t(s)$.

This gives a new curve function

$$P(u) = P(t(s)) = P^*(s)$$

that has arc length parameterization,
where $s \in [0, L]$, with $L =$length of the curve.

Compute arc length analytically

Given a regular parametric curve $P : [a, b] \rightarrow \mathbb{R}^3$, that is $P(t) = (x(t), y(t), z(t))$ with $t \in [a, b] \subset \mathbb{R}$. The **arc length** s of a curve P after time $t \in [a, b]$ is given by

$$s(t) = \int_a^t ||P'(r)|| dr$$

where $|P'(t)| = \sqrt{\left(\frac{dx(t)}{dt}\right)^2 + \left(\frac{dy(t)}{dt}\right)^2 + \left(\frac{dz(t)}{dt}\right)^2}$.

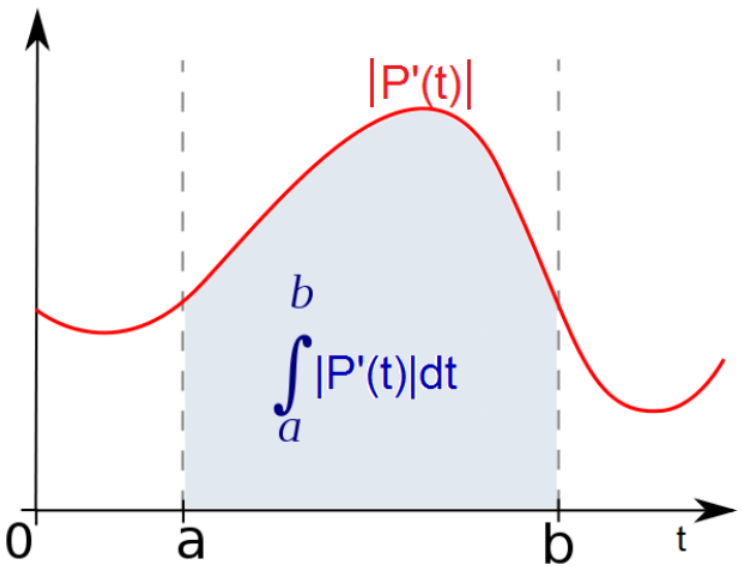
We can now set $u = t(s)$ as the inverse of $s(t)$, so that $P(u)$ has arc length parameterisation.

A closed form for arc length is often difficult or impossible to establish.

Compute arc length by numerical integration

The **arc length** s of a curve P after time $t \in [a, b]$ is given by

$$s(t) = \int_a^t ||P'(r)||dr$$



- Rectangular rule
- Trapezoidal rule
- Simpson's rule
- Gauss quadrature
- ...

Estimating Arc Length by Chord Length

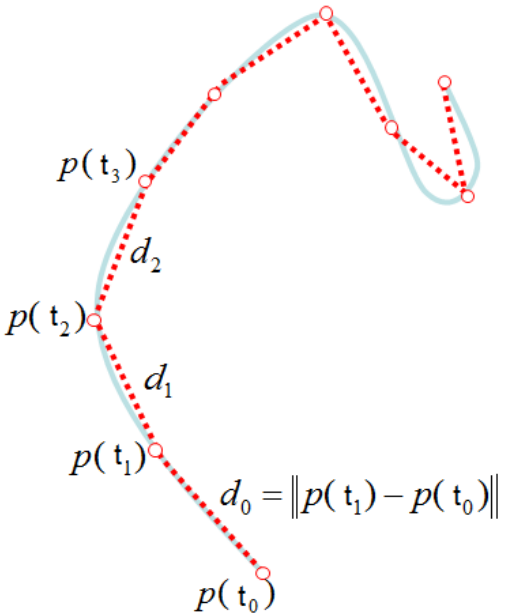
Arc length is commonly approximated using the concept of **chord length**.

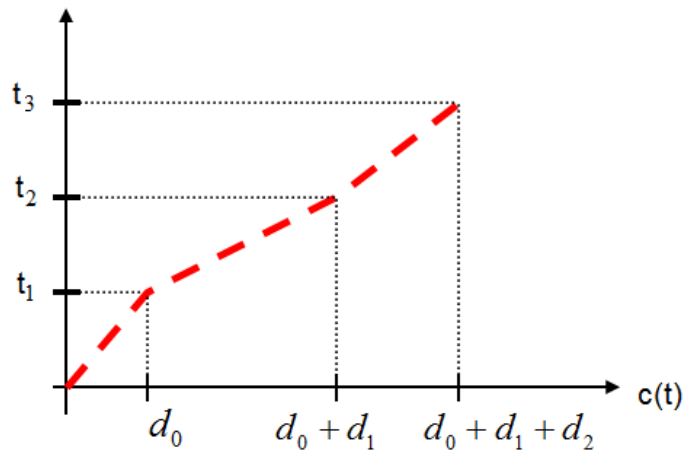
The chord length of a regular parametric curve $P(t)$ is given by

$$c(t) = \sum_i |\triangle P_{t_i}|$$

where $\Delta P_{t_i} = (P_{t_{i+1}} - P_{t_i})$ is the forward difference.

As $\Delta t \rightarrow 0$ the chord length converges to the arc length.





Estimating Arc Length by Chord Length

Easiest and simplest approach:

- Sample the curve at a multitude of parametric values.
- Compute the linear distance between adjacent points.
- Typically the space curve, parameterized wrt arc length s, is also normalized to s/L, so that the parametric variable varies between 0 and 1 as it traces out the space curve.
- Built up a table of chord length indexed by parametric values.
- Look up parametric value *t(s)* from arc length *s*

  If value is not in the table:
  - Take nearest value as an approximation. Or
  - Linearly interpolate between adjacent values.

Estimating Arc Length by Chord Length

A table can be built up of arc lengths (chord length!) indexed by parametric values. **Example**:

Index	Parametric Value (V)	Arc Length (G)
0	0.00	0.000
1	0.05	0.080
2	0.10	0.150
3	0.15	0.230
4	0.20	0.320
5	0.25	0.400
6	0.30	0.500
7	0.35	0.600
8	0.40	0.720
9	0.45	0.800
10	0.50	0.860
11	0.55	0.900
12	0.60	0.920
13	0.65	0.932
14	0.70	0.944
15	0.75	0.959
16	0.80	0.972
17	0.85	0.984
18	0.90	0.994
19	0.95	0.998
20	1.00	1.000

Estimating Arc Length by Chord Length

Both, estimate of arc length and interpolation of value introduce errors.

- **Errors may be reduced**
  - Supersampling.
  - Better methods of interpolation, i.e. higher-degree interpolation.
  - Adaptive chord length.

    i.e. Compare estimate of the segments length to the sum of the estimates for each of the segments halves. Define tolerance $\epsilon/2^n$

Arc Length Parameterisation

The velocity of a curve, $P(u)$, is equal to the tangent vector, $P'(u)$. The magnitude of its velocity is the speed, $|P'(u)|$.

Let $P(u)$ be a curve parameterised by arc length, so that parameter $u = t(s)$. The velocity of $P(u)$ is then

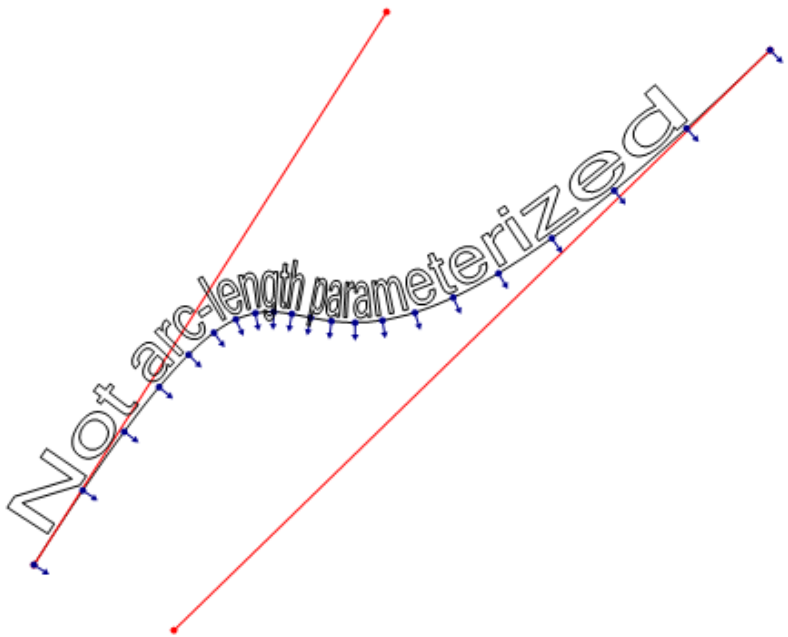
$$\begin{aligned} P'(u) &= \frac{d}{ds}P(t(s)) = P'(t)t'(s) \\ &= P'(t)/s'(t) \\ &= P'(t)/|P'(t)| \end{aligned} \qquad \rightarrow \quad |P'(u)| = 1$$

We have used the fact that $t'(s) = 1/s'(t)$ and the analytic expression for s.

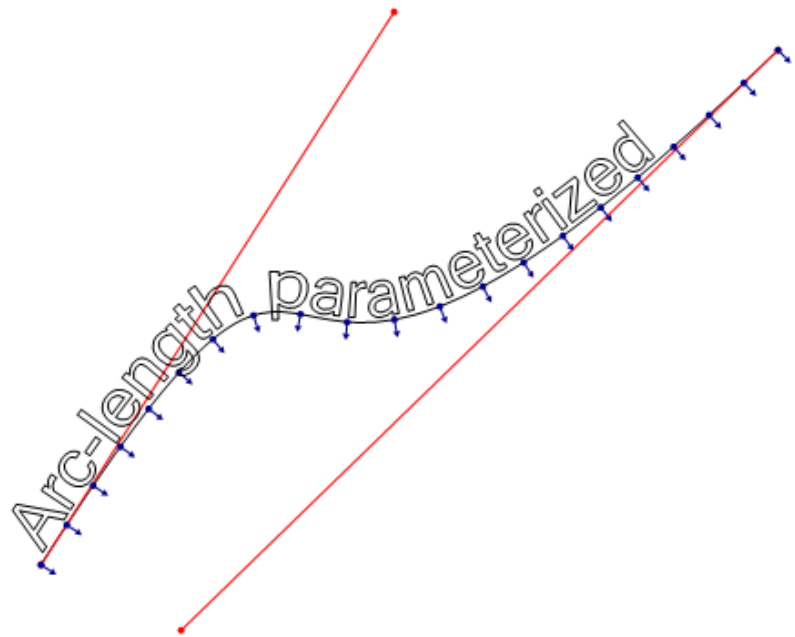
$$[f^{-1}]'(a) = \frac{1}{f'(f^{-1}(a))}$$

Also, for a curve P parameterised wrt arc length we have $P'(u) \cdot P''(u) = 0$, so that $P''(u)$ is orthogonal to $P'(u)$.

Arc Length Parameterisation



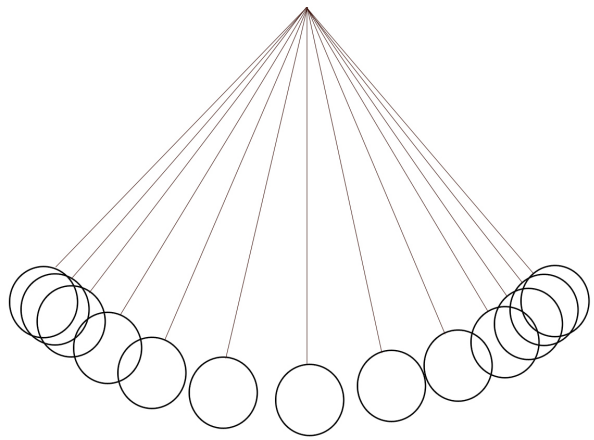
P(t)



P(t(s)) = P(u)

Speed Control

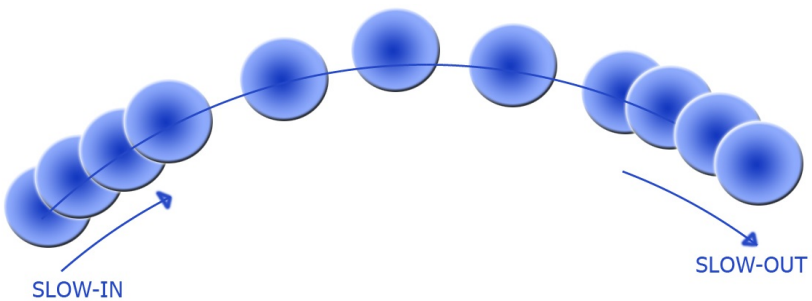
Stepping along the curve at equally spaced intervals of arc length will result in a constant speed traversal.

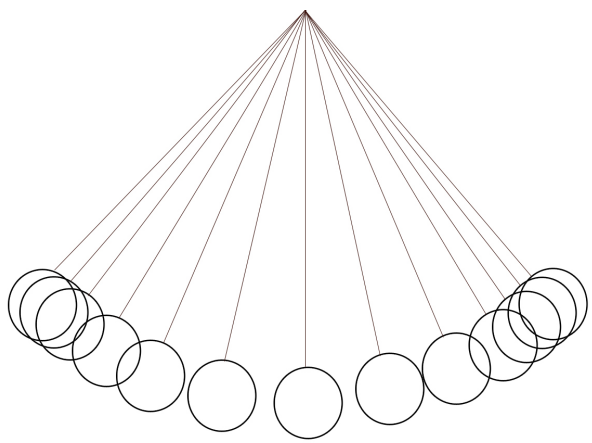


Speed Control

Speed along the curve can be controlled by varying the arc length values at something other than a linear function of t:

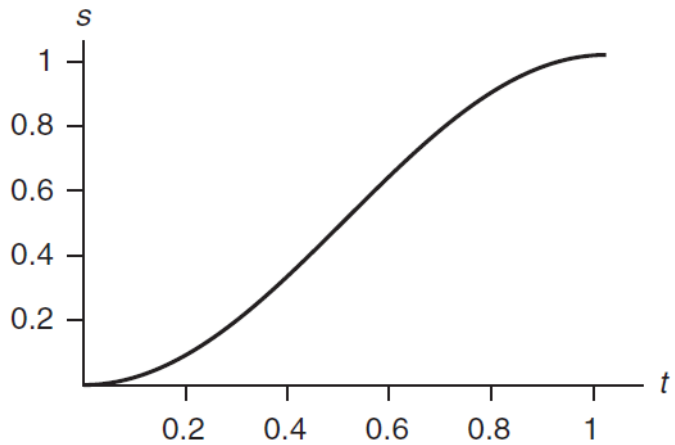
Interesting traversals can be generated by **speed control functions** (or **time funtions**) $S(t)$, that relate an equally spaced parametric value t to arc length s in such a way that a speed controlled traversal of the curve is generated.





Speed Control Function

Ease-in/ease-out traversal:



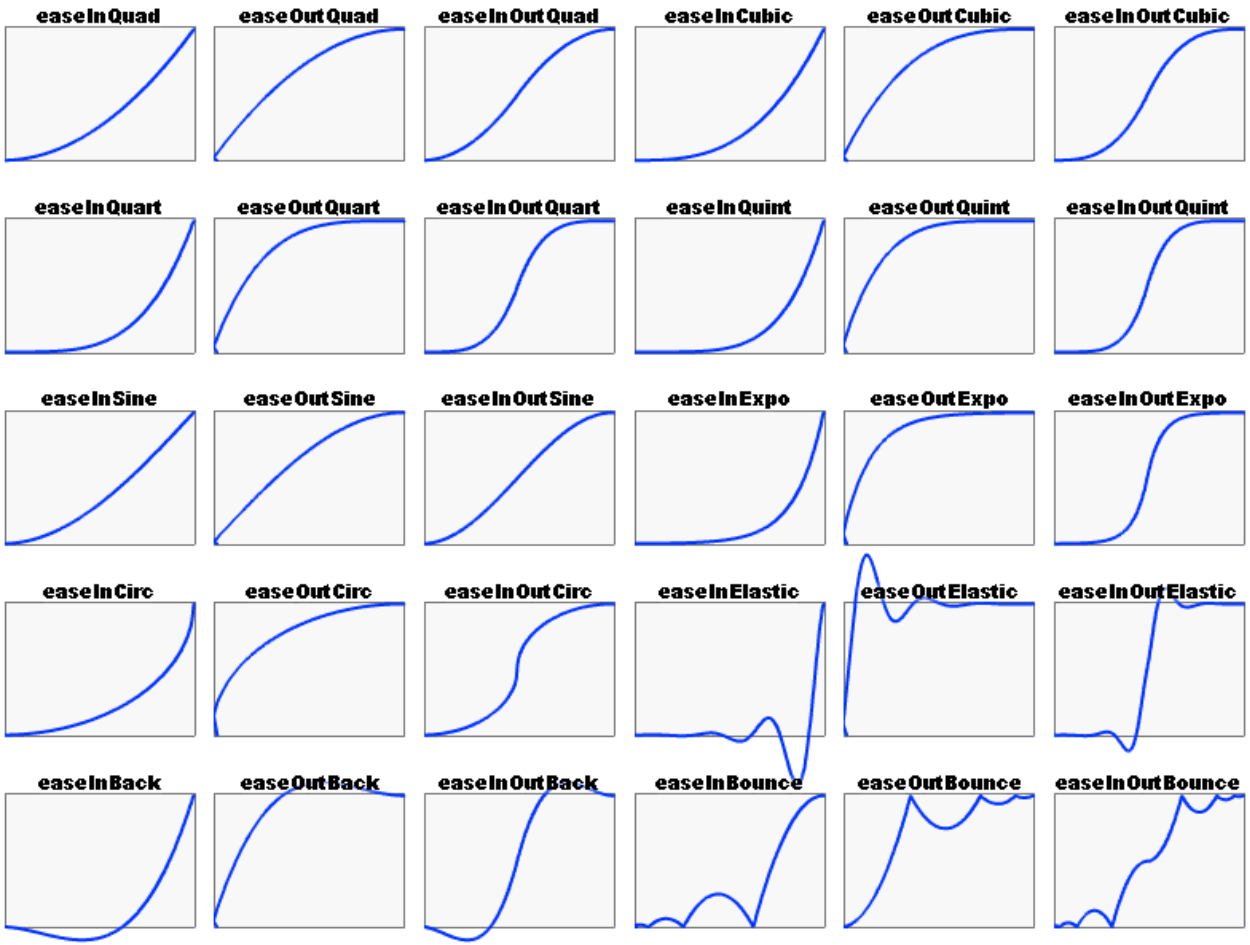
Object accelerates from a stopped position, reaches a maximum velocity, and then decelerates to a stopped position.

$$s = S(t) = \frac{\sin(t - \frac{\pi}{2}) + 1}{2}$$

maps the sine curve from $-\pi/2$ to $+\pi/2$ into the range $[0, 1]$.

Speed Control Function

Ease-in/ease-out traversal:



Speed Control Function

Many other speed control functions are possible, under two constraints:

1. The distance-time function should be continuous.
2. The entire space curve is to be traversed in the given total time, $0 = S(0)$ and $L = S(time_{total})$.

Typically, the speed control function is normalized s.t. the function ends at $1 = S(1)$. This way it can be easily mapped to our shape curve.

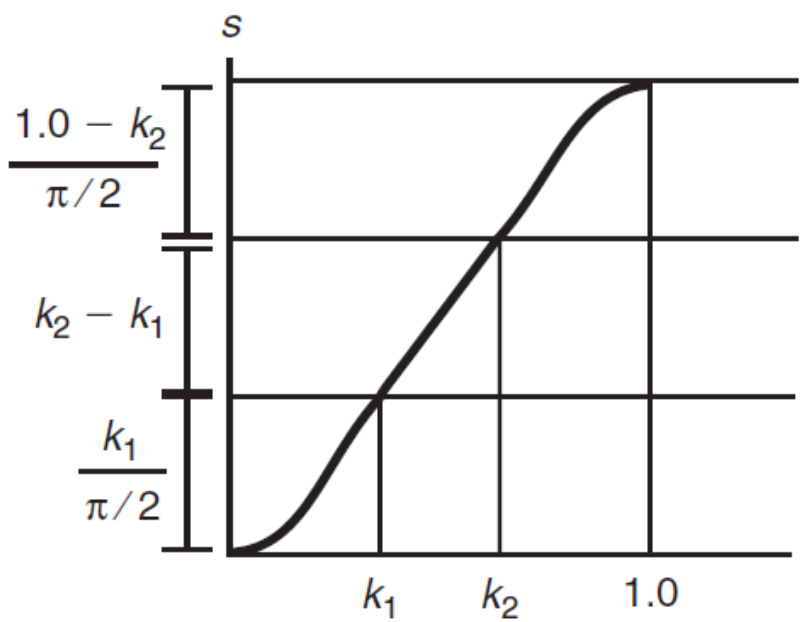
Speed Control Function

- The animated object translates along the path of the space curve at a speed indicated by $S'(t)$.
- At a given time t, $s = S(t)$ is the desired distance to have traveled along the curve.
- An arc length table can then be used to find the corresponding parametric value $u = U(s)$ that corresponds to that arc length.
- The space curve is then evaluated at u to produce a point on the space curve, $P = P(u) = P(U(S(t)))$.

Concatenating Speed Control Functions

Different distance - time functions may be pieced together for one animation. **Example**:

1. Ease in (accelerate) until time k_1: sine $-\pi/2$ to 0 from $0 - k_1$
2. constant velocity until k_2: straight line from $k_1 - k_2$
3. ease out (decelerate) until end of curve: sine 0 to $\pi/2$.



Concatenating Speed Control Functions

- First sine segment, sine from $-\pi/2$ to 0, is uniformly scaled by $k_1/(\pi/2)$, so that length is k_1.
- Length of line segment scaled to $k_2 - k_1$.
- Final line segment, sine from 0 to $\pi/2$, is uniformly scaled by $1 - k_2/(\pi/2)$, so that length is $1 - k_2$
- Finally, normalize the distance traveled: The resulting three segment curve must be scaled down by the total distance traveled: $k_1/(\pi/2) + k_2 - k_1 + (1 - k_2)/(\pi/2)$.

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

where $f = k_1(2/\pi + k_2 - k_1 + (1 - k_2)(2/\pi)$

Concatenating Speed Control Functions

Example:

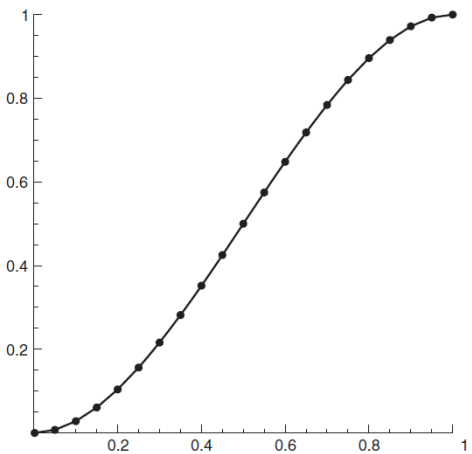
1. Ease in (accelerate) until time k_1: sine $-\pi/2$ to 0 from $0 - k_1$
2. constant velocity until k_2: straight line from $k_1 - k_2$
3. ease out (decelerate) until end of curve: sine 0 to $\pi/2$.

```
double ease(float t, float k1, float k2){
    double f, s;
    f = k1*2/PI + k2  k1 + (1.0  k2)*2/PI;
    if(t < k1)
       s = k1*(2/PI)*(sin((t/k1)*PI/2  PI/2)+1);
    else if (t < k2)
       s = (2*k1/PI + t  k1);
    else
       s = 2*k1/PI + k2  k1 + ((1-k2)*(2/PI)) *
             sin(((t  k2)/(1.0  k2))*PI/2);
    return (s/f);
}
```

Speed Control Functions

To avoid concatenating different distance - time functions it is often useful to use different types of polynomial functions, i.e. a single polynomial can be used to approximate the sinusoidal ease-in/ease-out function

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

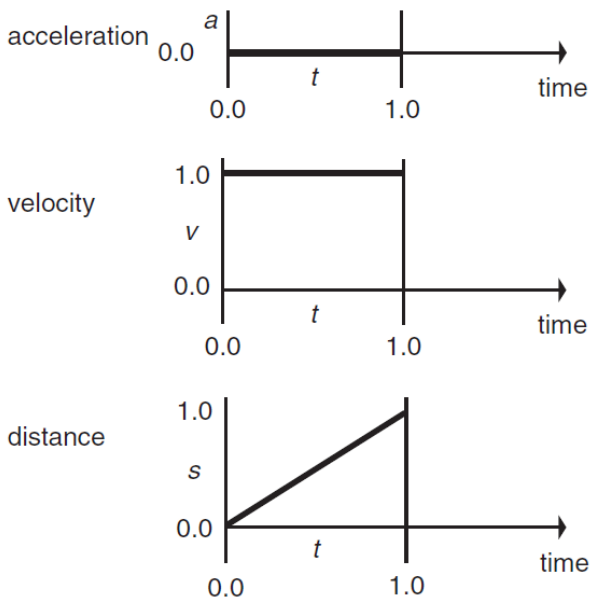


Note, there is no constant speed interval.

Speed Control Functions

Instead of providing a distance - time function, velocity or acceleration functions can be formulated. Those can then be integrated to get the required distance-time function.

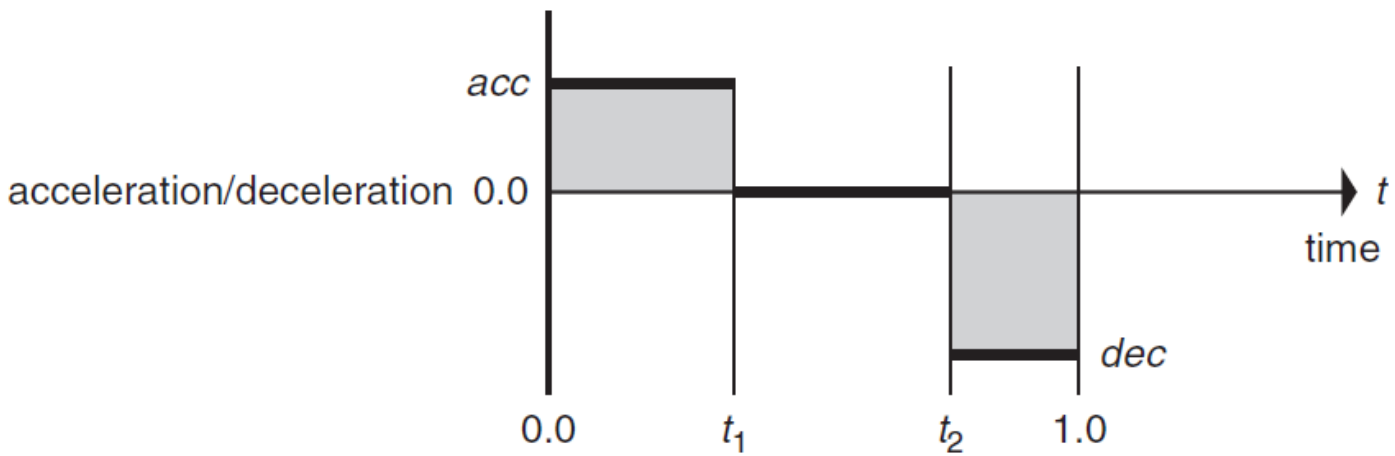
At constant speed along the curve, acceleration, velocity and distance are linear functions:



Speed Control Functions

Example: ease-in → constant speed → ease-out.

For this example the **acceleration-time curve** can be expressed in a piecewise linear function:

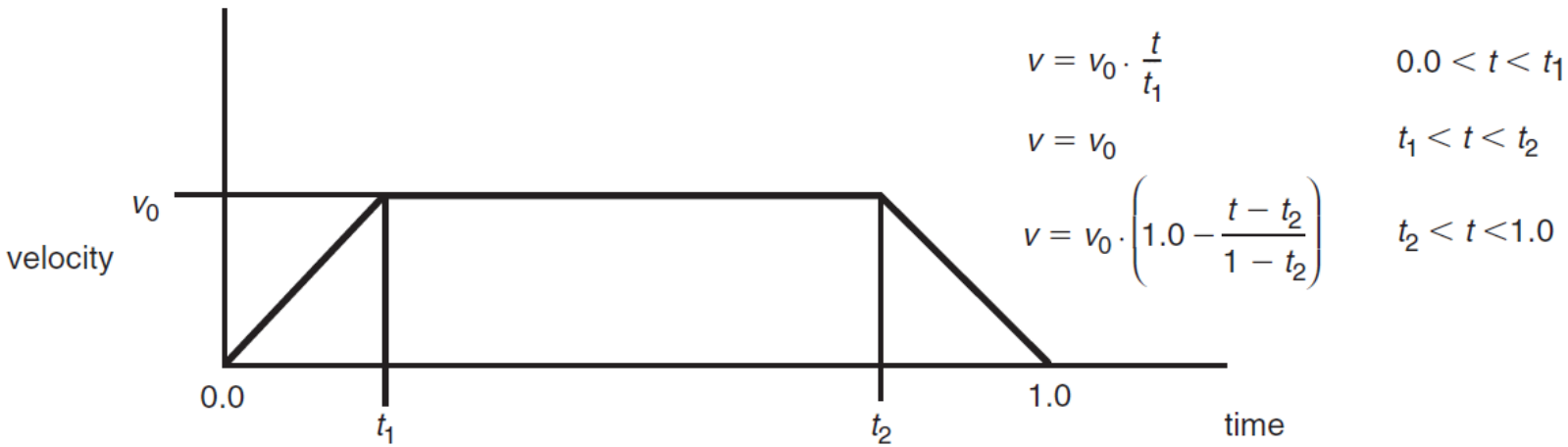


Note that this curve needs to integrate to 0 to come to a stop (or go back to starting velocity)!

Speed Control Functions

Example: ease-in → constant speed → ease-out.

The velocity time curve for constant acceleration:

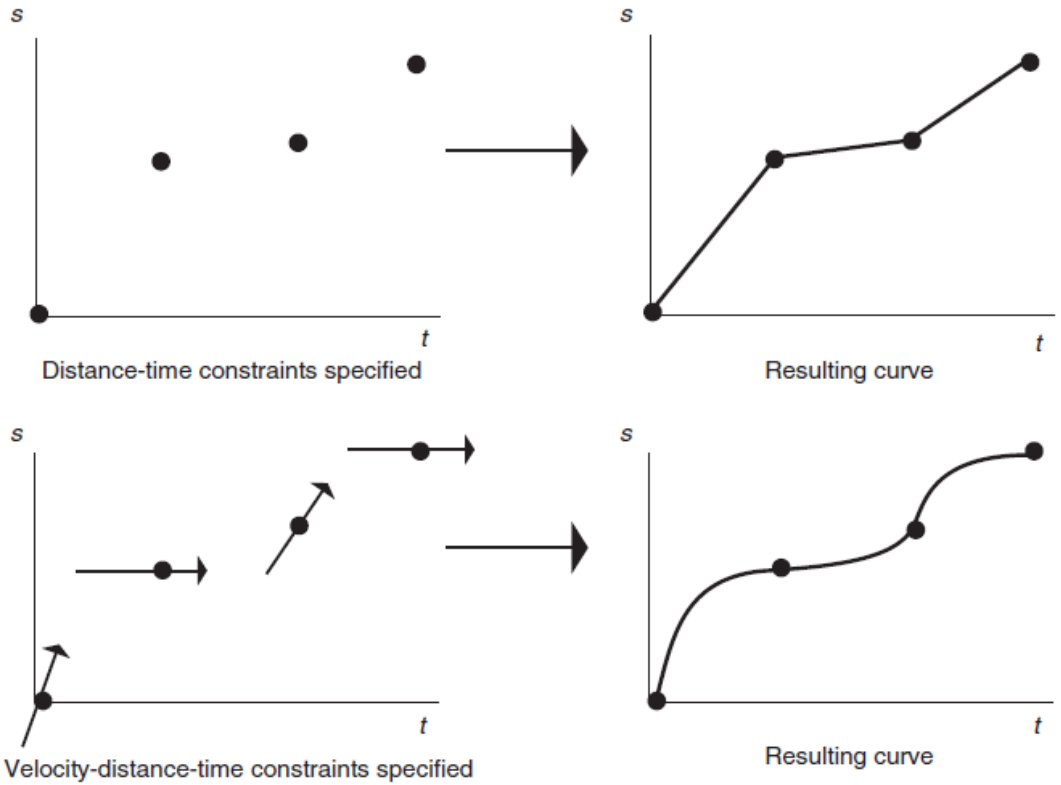


The area under the curve equals the distance traveled.
If time and distance have been normalised, we have:

$$1 = A_{acc} + A_{const} + A_{dec} = \frac{1}{2} v_0 t_1 + v_0(t_2 - t_1) + \frac{1}{2} v_0(1 - t_2)$$

Motion Control

Motion control frequently requires specifying positions and speeds along the space curve at specific times.

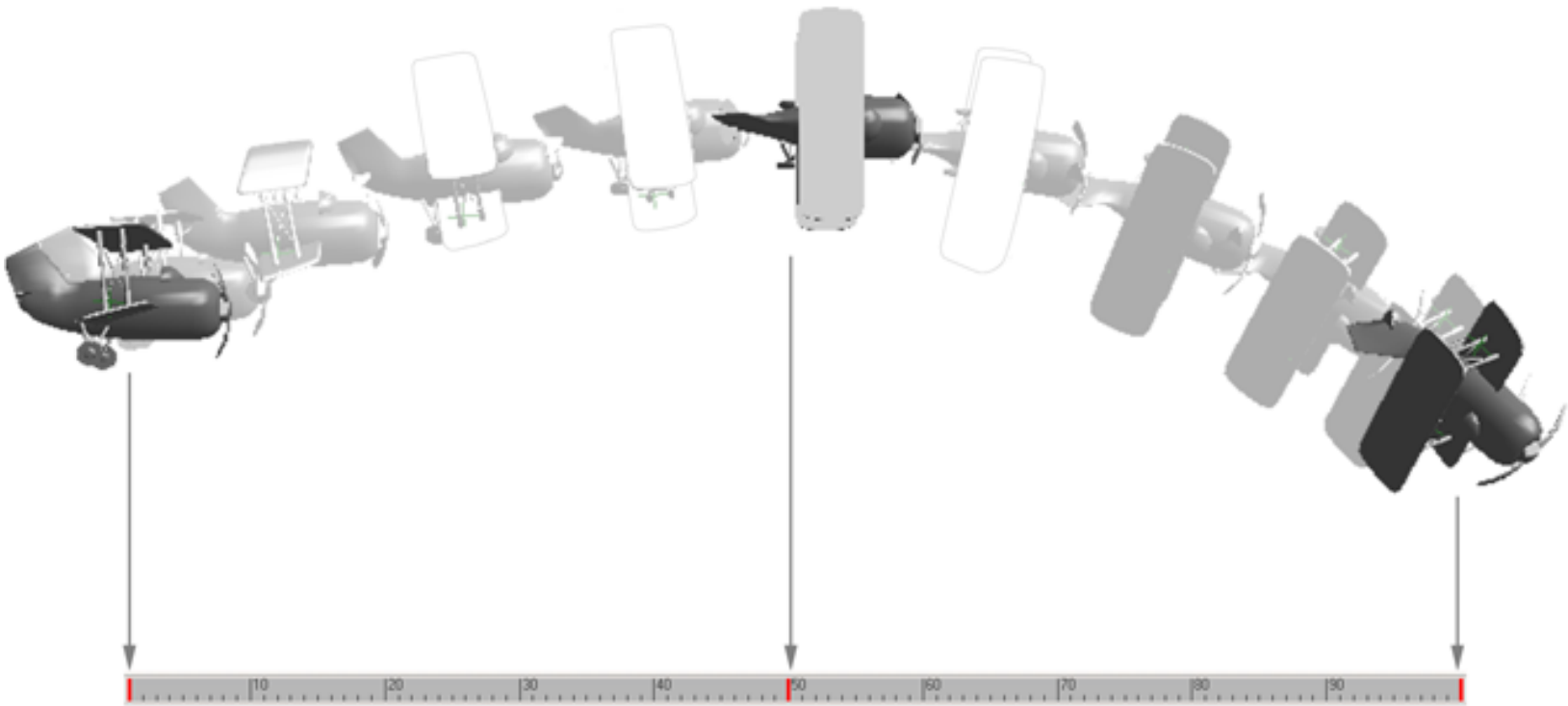


Constraints at a key point along a curve can be defined as an n-tuple

$< t_i, s_i, v_i, a_i, \cdots >$.

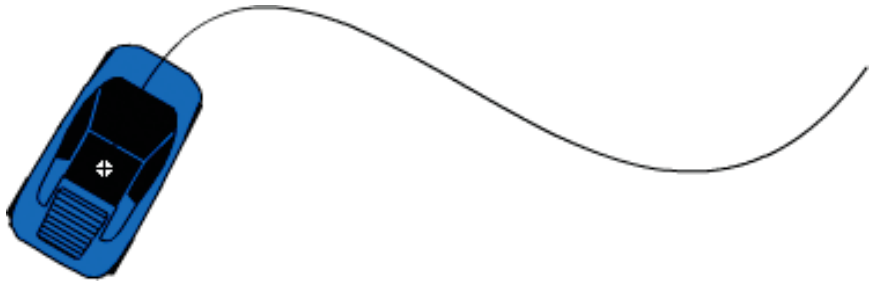
Working with Paths

Typical scenario in animation: An object is moving along a path.



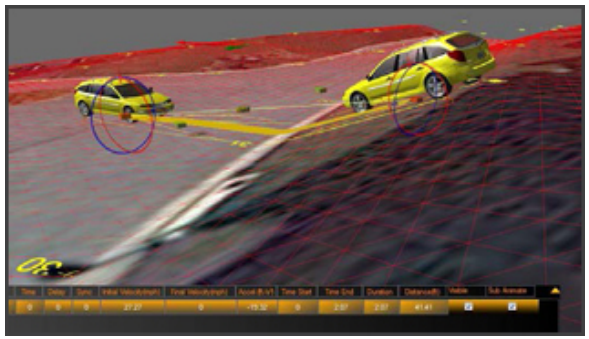
The path has been generated based on the frame number, arc length parameterization, and possibly ease-in/ease-out control.

Working with Paths



An object (or camera) following a path requires more than just translating along a space curve:

- The orientation of the object may change.
- A path may be constraint to lie on the surface of another object.
- Object needs to avoid other objects.

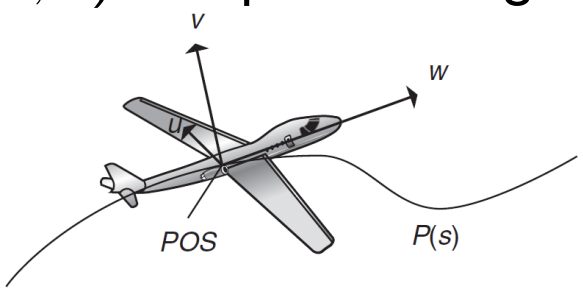


Orientation along a Path

Typically, a local coordinate system (u, v, w) is defined for an object to be animated.

- w = direction object is facing
- v = up vector
- u = perpendicular to v and w

The origin $(0, 0)$ is a point along the path $POS = P(s)$,



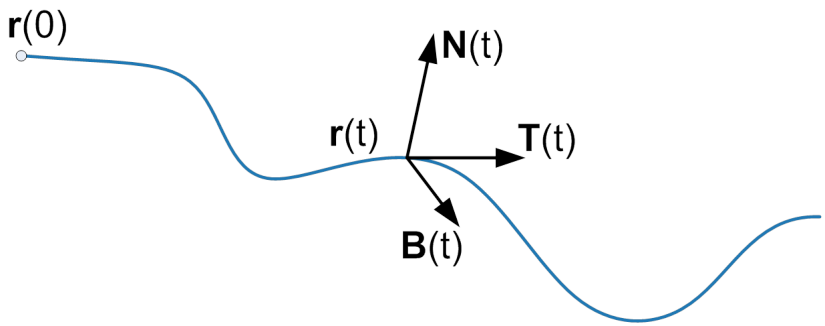
Frenet Frame

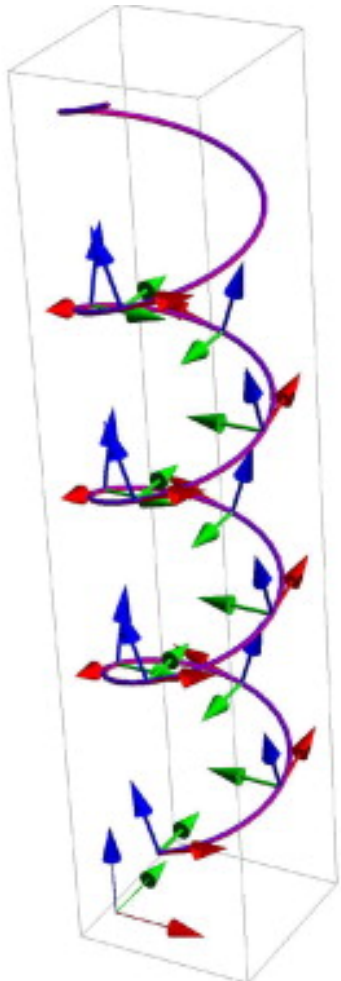
From differential geometry:

The Frenet frame is a local coordinate system, (T, N, B), moving along the curve:

- $T = P'(s)$ is the tangent vector
- $B = T \times N$ is the Binormal vector
- $N = P''(s)/|P''(s)|$ is the Principal normal vector

The Frenet frame changes orientation over the length of the curve.

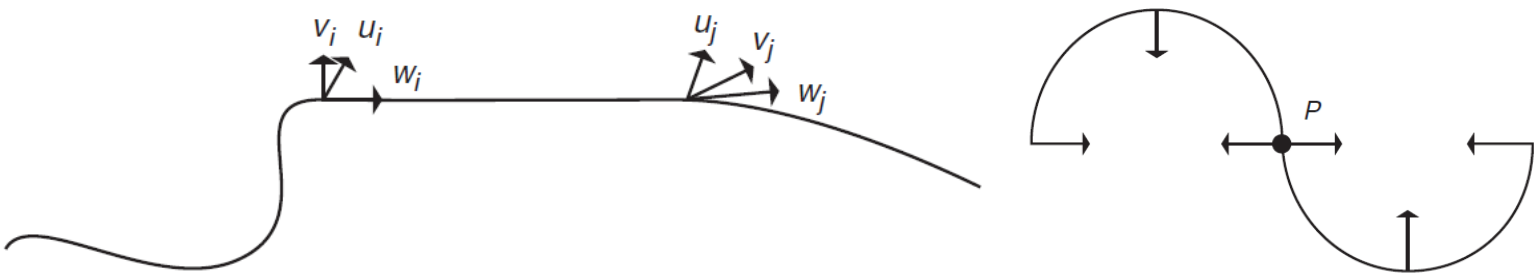




Frenet Frame

The Frenet frame may encounter problems:

- There may be segments of the curve that have no curvature $P''(s) = 0$.
  - The boundary Frenet frames differ by rotation only.
- There may be discontinuity in the curvature.



Useful to extract information about the curve, but not useful for **camera path following**.

Camera Path Following

Another local coordinate system:

- Set COI (centre of interest), a fixed point in the environment
- View vector $w = COI - POS$
- The up-vector, v, is orthogonal to w, in the plane defined by the global axis Y and w.
- The local coordinate system is then given by:
  - $w = COI - POS$
  - $u = w \times (0, 1, 0)$ (std., but a tilt may be defined)
  - $v = u \times w$

Works well, but problems when camera straight above the COI.

Camera Path Following

Define the COI as a function of the path:

- The COI may be defined as a function of the path using the **delta parametric value**
  - If the position of the camera on a curve is defined by $P(s)$, then the *COI* will be $P(s + \Delta s)$.
  - At the end of the curve, once $s + \Delta s$ is beyond the path parameterization, the view direction can be interpolated to the end tangent vector as s approaches 1.

Camera Path Following

Define COI using a separate path:

- Camera's position is specified by $P(s)$.
- COI is specified by some $C(s)$.
  - $w = C(s) - P(s)$
  - $u = w \times (0, 1, 0)$
  - $v = u \times w$

Requires more work, but provides greater control and more flexibility.

Camera Path Following

Fix COI at one location for an interval of time then move to another location
(using linear spatial interpolation or ease-in/ease-out interpolation)
and fix it there for a number of frames:

- $w = (x_i, y_i, z_i) - P(s)$
- $u = w \times (0, 1, 0)$
- $v = u \times w$

Summary Interpolation

The ability to understand and control the interpolation process is very important in computer animation programming.

Interpolation of values takes many forms, including

- position, velocity, and/or acceleration,
- image pixel color, and
- shape parameters.

The control of the interpolation process can be scripted and key framed.