Intermediate Graphics & Animation Programming

GPR-300
Daniel S. Buckstein

Curves, Splines & Interpolation Algorithms
Weeks 7 – 10

License

Curves & Interpolation

- Linear & bilinear interpolation
- Bézier splines and interpolation
- Catmull-Rom splines and interpolation
- Cubic Hermite splines and interpolation

Video

- “Creepy Watson” (1:42)
  - https://www.youtube.com/watch?v=13YlEPwOfmk
- Why is Watson creepy???
- Watson is forgetting a little something called…

Interpolation

- Given at least two *known values* (*keyframes*), ***interpolation*** is the estimation of some value in between!

- Watson sho have been interpolating his *position*

Types of Interpolation

- Different kinds for different situations:
- The most fundamental:
  <u>Linear Interpolation</u> (“LERP”)
  – Interpolation between exactly two values
  – Motion occurs over a linear path!
- Spherical Linear Interpolation (“SLERP”)
  – Interpolate uniformly over an arc (still two values)
  – Very useful for rotations

Types of Interpolation

- Normalized Linear Interpolation (“NLERP”)
  – An alternative to SLERP
  – Better performance but less precision

- Spline Interpolation
  – Interpolation over a segmented path
  – Many different kinds… a talk on its own!

Types of Interpolation

- We will be discussing all of the above, but before we go anywhere at all…
- ***NOTICE***: see how the word "*value*" is used?
- ***NEVER FORGET***: these are just *algorithms*
- They are used to process *DATA*
- How you use the algorithms is up to you
- Better for certain scenarios over others

Linear Interpolation

- Also known as "LERP" for short
  - ***L***inear Int***erp***olation
- Given exactly two known values (***keyframes***)…

- …we can move from p0 to p1 as time passes
- LERP represents change along a *line*!

Linear Interpolation

- Remember that this line does not have to describe something *physical*…
- Real numbers exist on a line, too!

Linear Interpolation

- So our two values are the known “endpoints”
- …but how do we *interpolate*?
- I.e. how do we determine any position in between?
- Control *value in between* by using *time*
- Watson’s problem: he did not take any *time* to reach his target *position*… let’s teach him how to linearly interpolate!

Linear Interpolation

- We can build an algorithm by talking through it (*please* remember that it is not only used for position)
- We start at some initial value or point, *p0*
- We want to reach an end value or point, *p1*
- We have some time in between, *t*

Linear Interpolation

- The function (i.e. the formula for LERP):

$$\mathbf{lerp}(\boldsymbol{p_0}, \boldsymbol{p_1}, \boldsymbol{t}) = \boldsymbol{p_0} + (\boldsymbol{p_1} - \boldsymbol{p_0}) \cdot \boldsymbol{t}$$

- Plain English: starting at *p0*, add a displacement* scaled by some value *t*
  - **remember a displacement is just a difference, as seen above*

Linear Interpolation

- The function:

Known endpoints

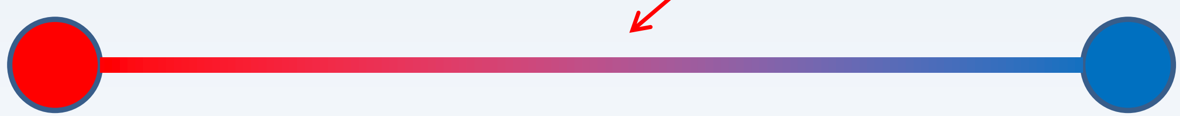
$$\mathrm{lerp}(\underbrace{p_0, p_1}_{\text{Start} \quad \text{Finish}}, t) = p_0 + (p_1 - p_0) \cdot t$$

p_0 (start) p_1 (finish)

Linear Interpolation

- The function:

$$\mathrm{lerp}(p_0, p_1, t) = p_0 + (\overbrace{p_1 - p_0}^{\text{Displacement between points/values}}) \cdot t$$



p_0 (start) p_1 (finish)

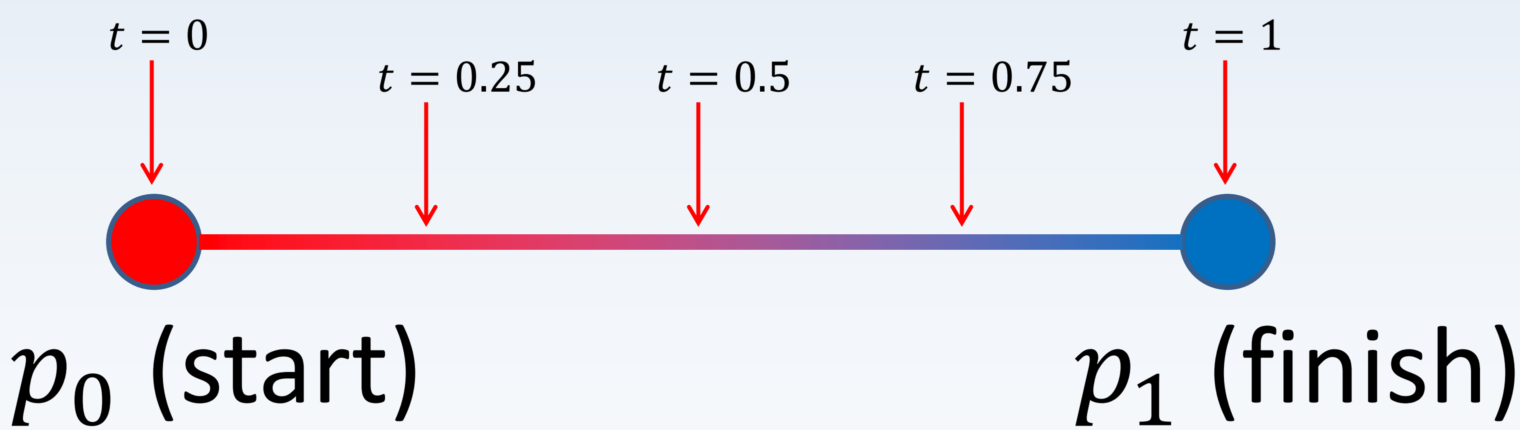
Linear Interpolation

- The function:

Control parameter represents *relative* value between p0 and p1

Control parameter
(e.g. time)

$$\operatorname{lerp}(p_0, p_1, \underline{t}) = p_0 + (p_1 - p_0) \cdot \underline{t}$$



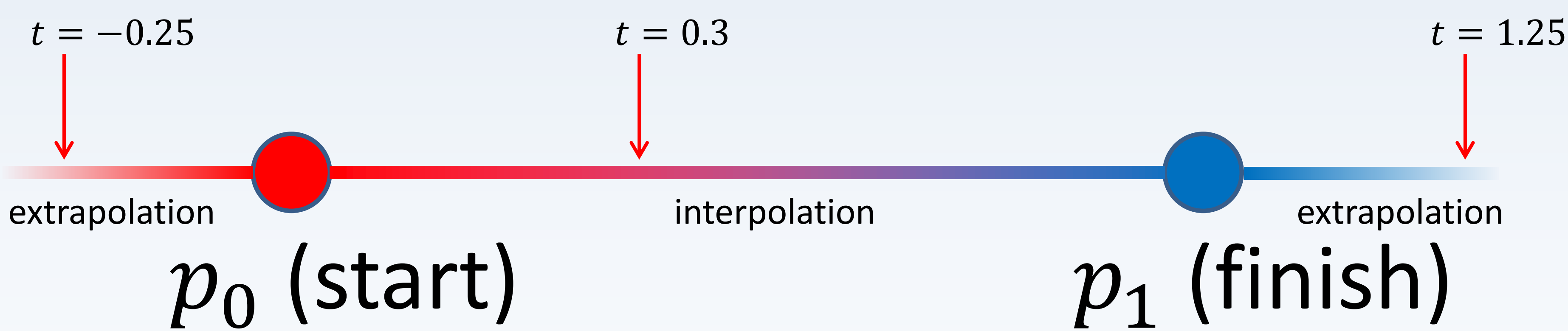
Linear Interpolation

- The function:

Control parameter represents *relative* value between p0 and p1

Control parameter
(e.g. time)

$$\mathrm{lerp}(p_0, p_1, \underline{t}) = p_0 + (p_1 - p_0) \cdot \underline{t}$$



Linear Interpolation

- The function:

$$\mathrm{lerp}(p_0, p_1, t) = p_0 + (p_1 - p_0)t$$

- Endpoints/known values can mean *anything* we want… (they're just data!)
- Control parameter is a number between 0 and 1 (weight, relative distance)

Linear Interpolation

- Another way to write LERP: “blend”

$$\mathrm{lerp}(p_0, p_1, t) = (1 - t)p_0 + (t)p_1$$

- Exactly the same as LERP with the terms expanded and refactored differently

Linear Interpolation

- Pro tip: you can factor the formulas for LERP into matrix form:

$$\mathbf{lerp}_{\boldsymbol{p}_0\boldsymbol{p}_1}(\boldsymbol{t})$$

$$= \begin{bmatrix} \boldsymbol{p}_0 & \boldsymbol{p}_1 \end{bmatrix} \begin{bmatrix} \mathbf{1} & \mathbf{-1} \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{1} \\ \boldsymbol{t} \end{bmatrix}$$

Influence matrix: the raw inputs stored as a column matrix

"***Kernel***"

Polynomial terms as a matrix: the control parameter

Linear Interpolation

- Proof: expanding the *left product* first yields one implementation of LERP…

$$\mathrm{lerp}_{p_0 p_1}(t) = \left(\begin{bmatrix} p_0 & p_1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \right) \begin{bmatrix} 1 \\ t \end{bmatrix}$$

$$= \begin{bmatrix} p_0 & -p_0 + p_1 \end{bmatrix} \begin{bmatrix} 1 \\ t \end{bmatrix}$$

$$= \boldsymbol{p_0 + (p_1 - p_0)t}$$

Linear Interpolation

- Proof: expanding the *right product* first yields another implementation of LERP…

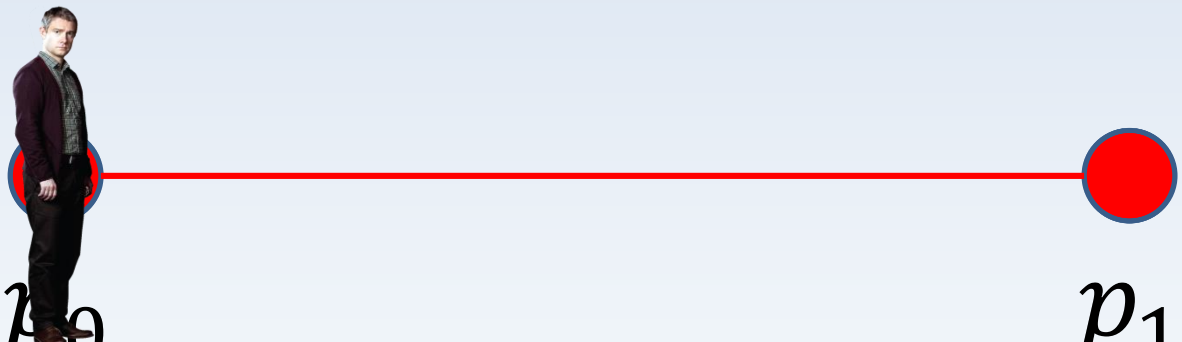
$$\mathrm{lerp}_{p_0 p_1}(t) = \begin{bmatrix} p_0 & p_1 \end{bmatrix} \left(\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ t \end{bmatrix} \right)$$

$$= \begin{bmatrix} p_0 & p_1 \end{bmatrix} \begin{bmatrix} 1 - t \\ 0 + t \end{bmatrix}$$

$$= \boldsymbol{(1 - t) p_0 + (t) p_1}$$

Applications of LERP

- *Motion*: move from one position in space to another
- Example: Not-So-Creepy Watson

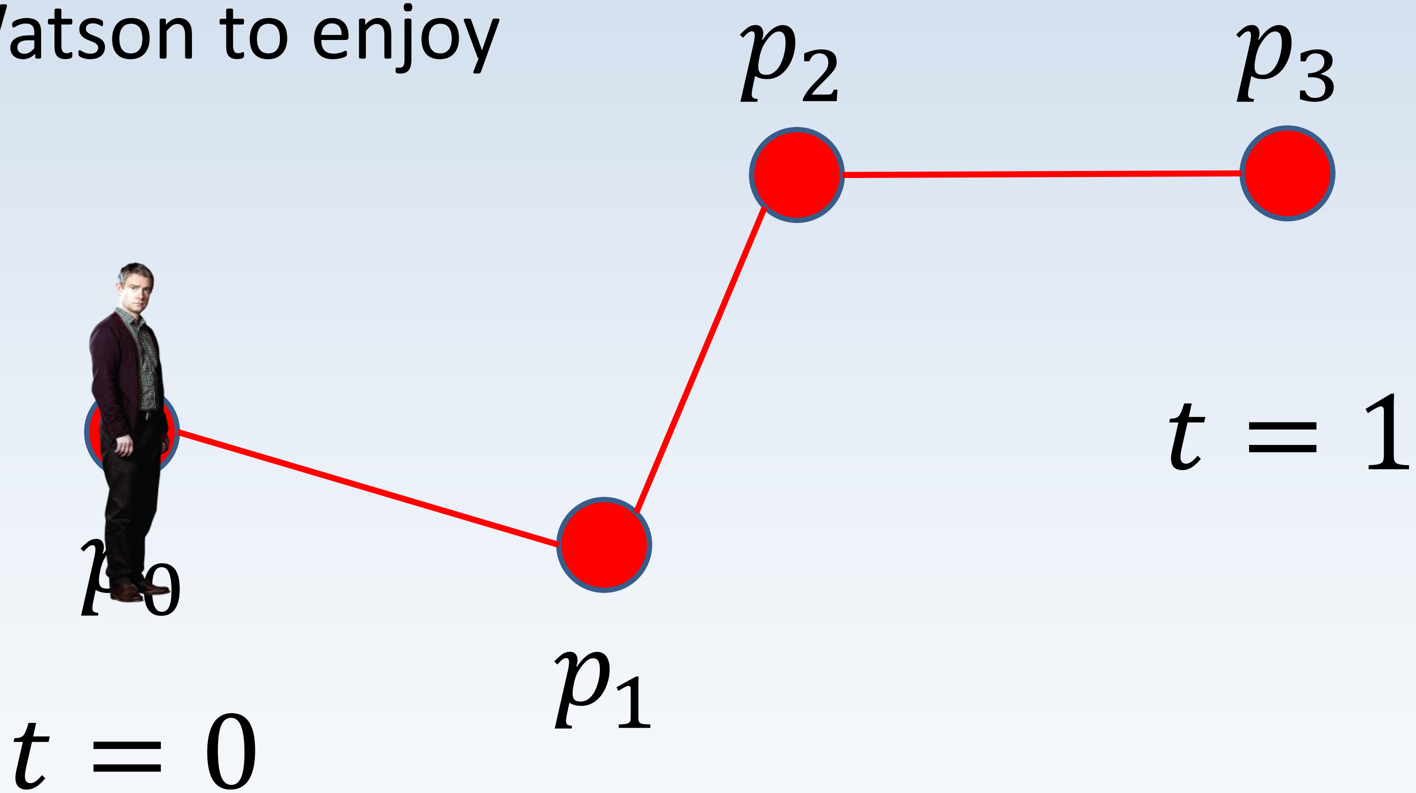


p_0 p_1

$t = 0$ $t = 0.5$ $t = 1$

Applications of LERP

- *<u>Paths & curves</u>*: multiple line segments for Watson to enjoy



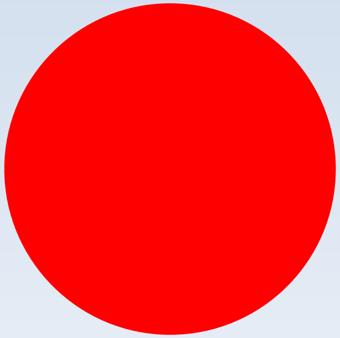
Applications of LERP

- *Graphics*: transition from red to blue

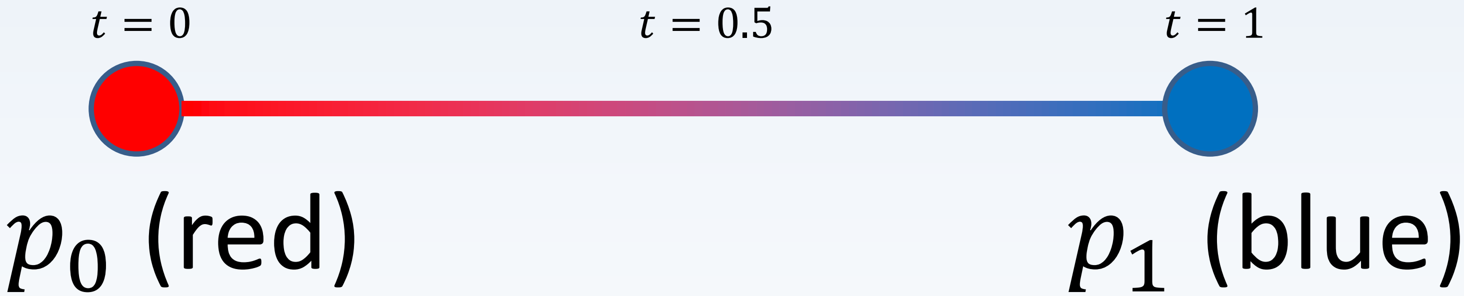
$t = 0$: red

$t = 0.5$: purple

$t = 1$: blue



(we've seen this example already today!)



Applications of LERP

- *<u>Film, animation, games</u>*: cross-fade *video* and *audio* (actual sound data)

$t = 0.5$: midway through transition

Applications of LERP

- *Animation*: smoothly blend between walking and running sequences

$t = 0$: walking

$t = 1$: running

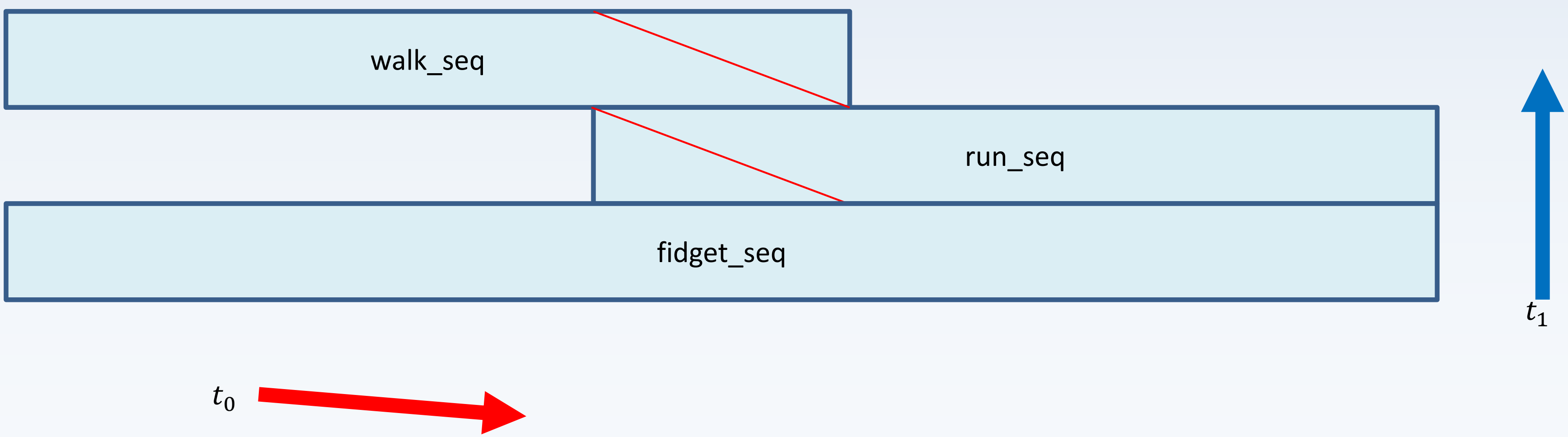
Applications of LERP

- ***Animation***: morph targets

http://deletionquality.net/blog

Applications of LERP

- ***Animation***: probably most applicable… blend between *frames* in multiple sequences *and* blend the sequences simultaneously!!!
- Algorithm called ***bilinear interpolation*** (next week)



Applications of LERP

- The important thing to remember:
- LERP is just a ***tool***! The values are ***just data***!
- The tool is always the same…
- …it's how you use it that changes its *context*
- ***Common problem:*** You are told that lerp is used for position, suddenly you believe that is the only thing lerp can be used for… false!

Recap: LERP

- The fundamental formula for everything!
- ***<u>Linear interpolation (LERP)</u>***

- Different implementations:

$$\mathrm{lerp}(p_0, p_1, t) = (1 - t)p_0 + (t)p_1$$

$$\mathrm{lerp}(p_0, p_1, t) = p_0 + t(p_1 - p_0)$$

Bilinear Interpolation

- We know about linear interpolation:

$$\mathrm{lerp}(p_0, p_1, t) = (1 - t)p_0 + (t)p_1$$

- Can be thought of as a *weighted average* of two values, where *t* is the weighting of *p1*
- The result of LERP is also a value that has the same type as its inputs

Bilinear Interpolation

- If the result of LERP is just a value…
- …could we not process it using LERP again?
- ***Bilinear Interpolation*** (BiLERP): linear interpolation *of* linearly interpolated values
- If LERP is a weighted average, then BiLERP is the weighted average of two other weighted averages
- Weighted-average-ception

Bilinear Interpolation

- BiLERP algorithm:
- Given *four* arbitrary values to interpolate: p_0, p_1, p_2, p_3 and *two* control parameters t_0, t_1
- t_0 is used to interpolate each pair of points
- t_1 is used to interpolate the results

Bilinear Interpolation

- BiLERP algorithm:
- It's just a *weighted average of weighted averages*, so BiLERP can be written as:

$$\mathbf{bilerp}(\boldsymbol{p_0}, \boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p_3}, \boldsymbol{t_0}, \boldsymbol{t_1})$$
$$= \mathbf{lerp}(\mathbf{lerp}(\boldsymbol{p_0}, \boldsymbol{p_1}, \boldsymbol{t_0}), \mathbf{lerp}(\boldsymbol{p_2}, \boldsymbol{p_3}, \boldsymbol{t_0}), \boldsymbol{t_1})$$

This expands to:

$$= (1 - t_1)\mathrm{lerp}(p_0, p_1, t_0) + (t_1)\mathrm{lerp}(p_2, p_3, t_0)$$

Bilinear Interpolation

- BiLERP algorithm:
- Using subscript notation:

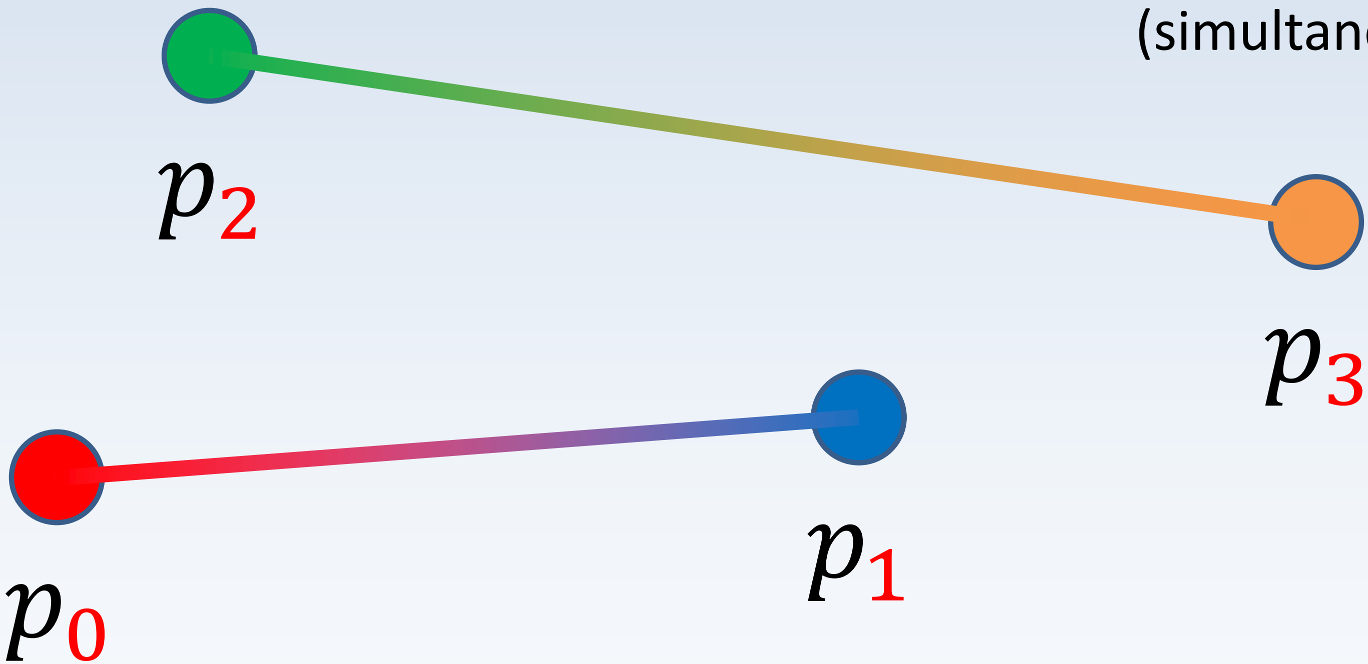
$$\mathrm{bilerp}_{p_0 p_1 p_2 p_3}(t_0, t_1)$$

- It is a function of the control parameters
- These are what we *control* to change the result of the algorithm

Bilinear Interpolation

- BiLERP visualization using 2D points:
- t_0 controls interpolation between value pairs

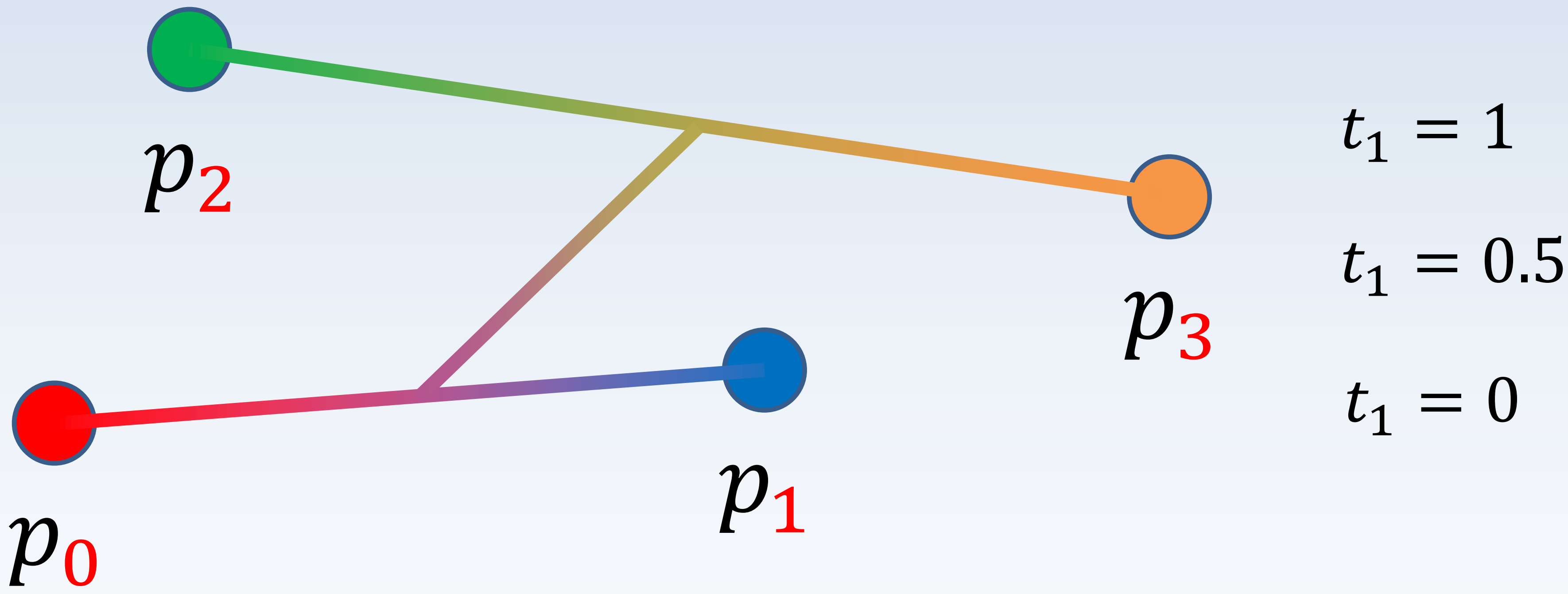
(simultaneous)



$t_0 = 0$ $t_0 = 0.5$ $t_0 = 1$

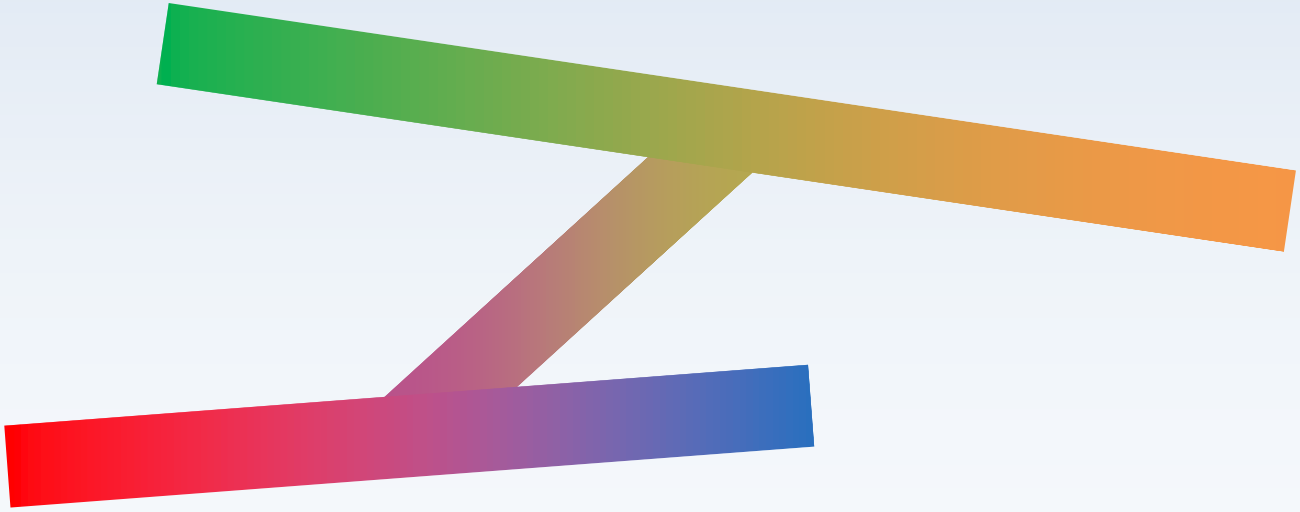
Bilinear Interpolation

- BiLERP visualization using 2D points:
- t_1 controls interpolation between results



Bilinear Interpolation

- Remember, it's just an algorithm!
- Our visualization not only interpolates the positions of the points, but also the colours!



Bilinear Interpolation

- Remember, it's just an algorithm, a weighted average… example with real numbers:

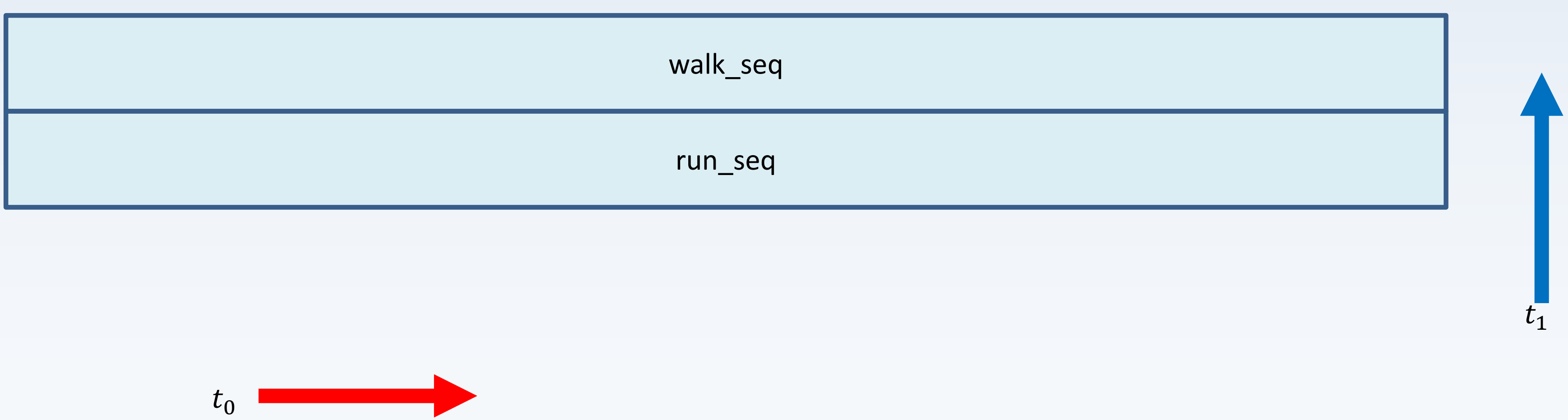
$p_0 = -1, \qquad p_1 = 4, \qquad p_2 = 3, \qquad p_3 = -6$

$t_0 = 0.5, \qquad t_1 = 0.25$

$$\begin{aligned}&\mathrm{bilerp}(p_0, p_1, p_2, p_3, t_0, t_1)\\ &= \mathrm{lerp}(\mathrm{lerp}(p_0, p_1, t_0), \mathrm{lerp}(p_2, p_3, t_0), t_1)\\ &= (1 - t_1)\mathrm{lerp}(p_0, p_1, t_0) + (t_1)\mathrm{lerp}(p_2, p_3, t_0)\\ &= \cdots \text{ what's the answer???}\end{aligned}$$

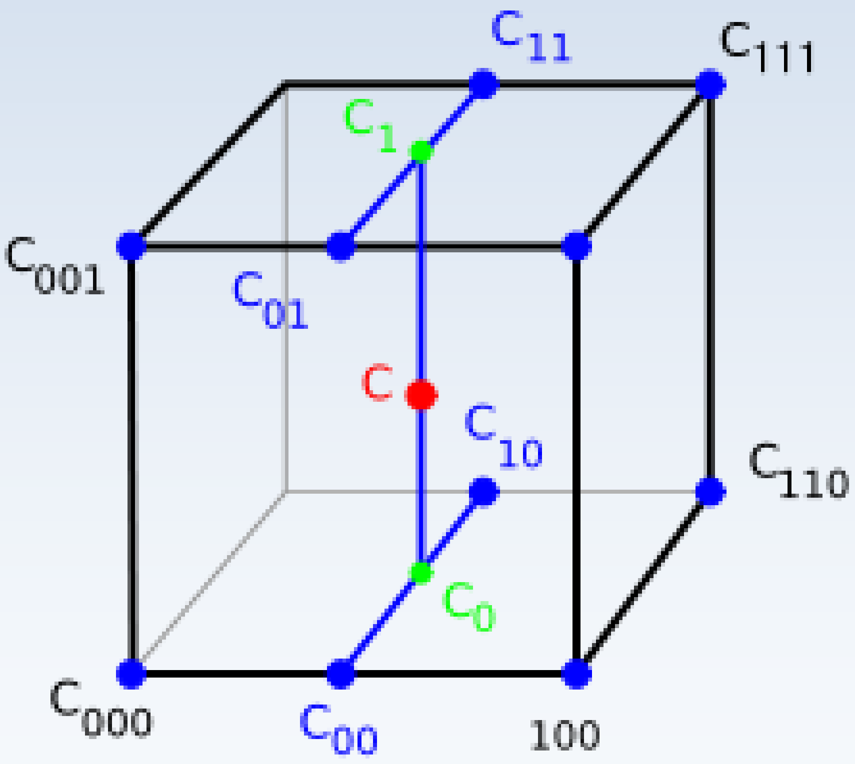
Bilinear Interpolation

- *__Animation__*: probably most applicable… blend between *frames* in multiple sequences *and* blend the sequences simultaneously!!!



Bilinear Interpolation

- What would "*trilinear interpolation*" be???



Paths

- Introduced paths in previous lecture
- So what is a *path*, really?

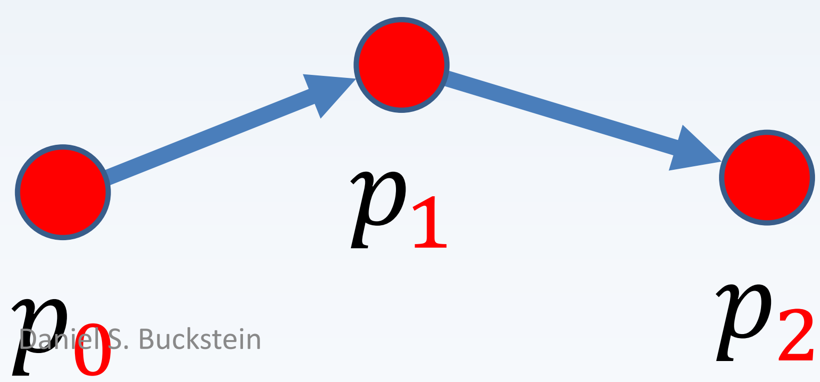
- "*Path*": sequence of segments connected by keyframes
- Use interpolation to find in-betweens
- Different kinds of interpolation

Paths

- Again: Please, *please*, ***please*** remember that these are just tools, more than one way to use these algorithms!!!
- Will explain algorithms using spatial examples
- I.e. “locomotion” as the base application
- Getting objects to physically follow a path!

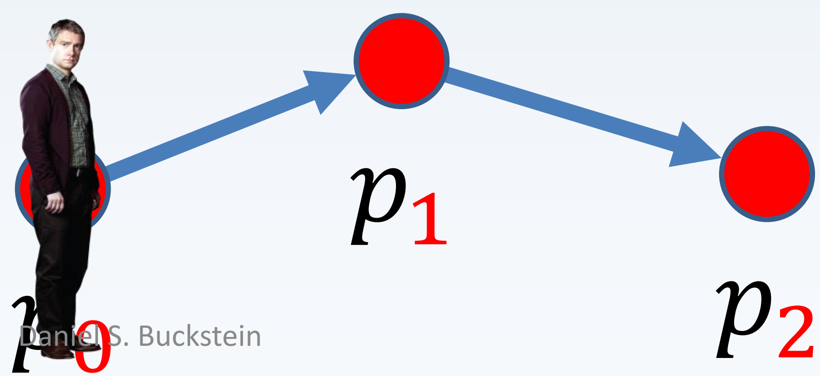
Paths

- So far we have learned about "linearly segmented paths"
- Points on path are connected by line segments
- In-betweens calculated using LERP



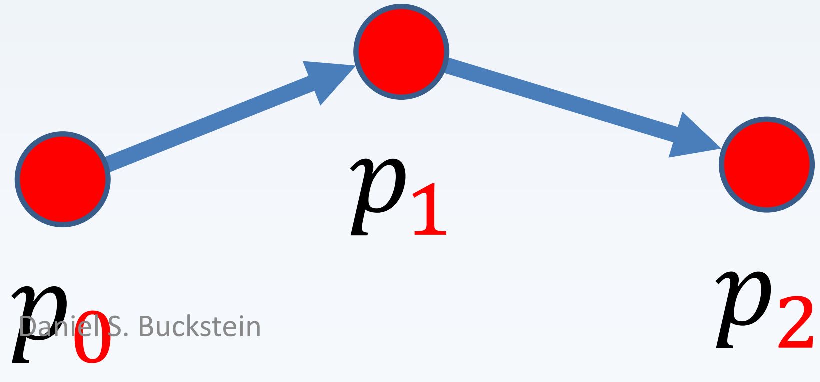
Paths

- Remember that "*keyframes*" just describe known states for some piece of data!
- Suppose that the *current state* of our object (i.e. current keyframe) is *p0* and it is *transitioning* towards *p1* (which transition???)
- Which keyframes *influence* our object's transition?



Paths

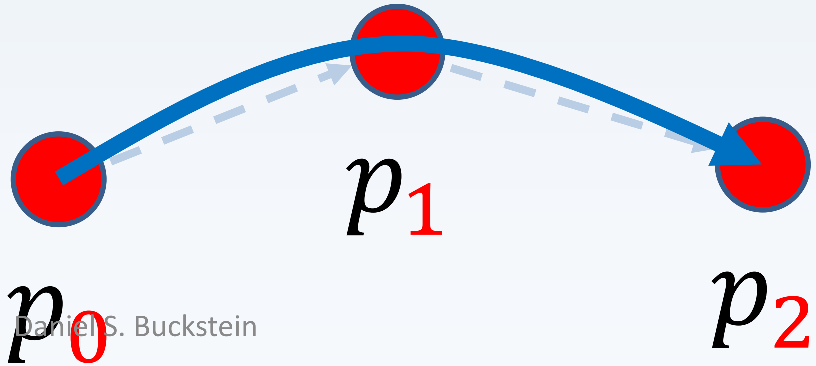
- LERP only has two influences: start point and end point
- The result: each transition is very rigid
- How do we achieve smoother transitions?
- How do we better accommodate motion along the *entire path* instead of just between two points???



Curves & Spline Interpolation

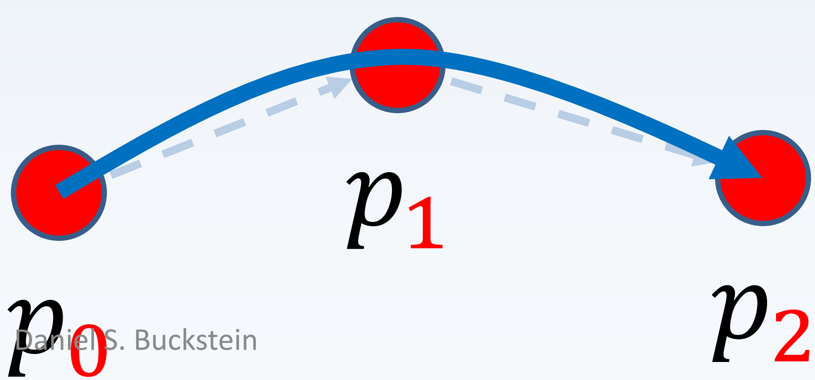
- ***<u>CURVES & SPLINES</u>***
- Different kinds of interpolation along curves
- Each takes into account different influences

- Catmull-Rom interpolation
- Bézier interpolation
- Cubic Hermite



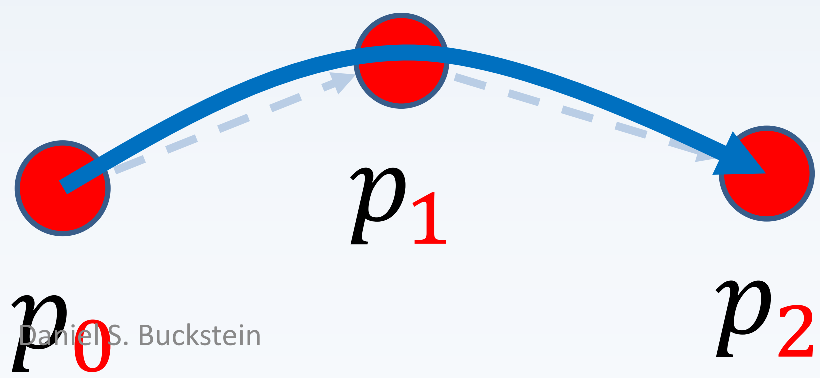
Curves & Spline Interpolation

- ***Spline***: a piecewise function defined by polynomials of varying degrees
- Polynomial: $f(x) = x^3 - 3x^2 + 3x - 1$
- Paths are piecewise
- Therefore splines are a good tool for path interpolation
- (piecewise tool for a piecewise application!!!)



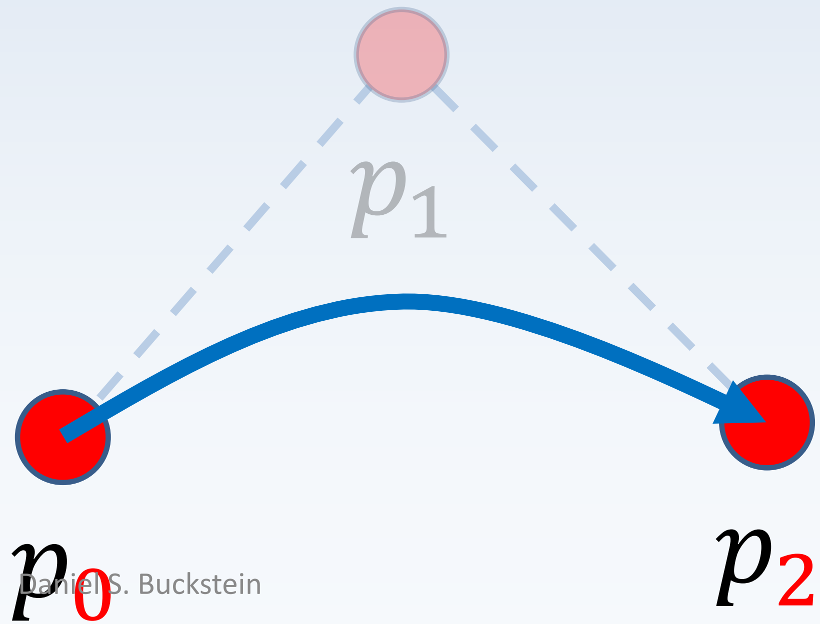
Curves & Spline Interpolation

- Splines use different polynomials and relate them to the key values on a path to determine the in-betweens
- This general category of algorithms is called "*spline interpolation*"
- The splines on which interpolation occurs are *curves*



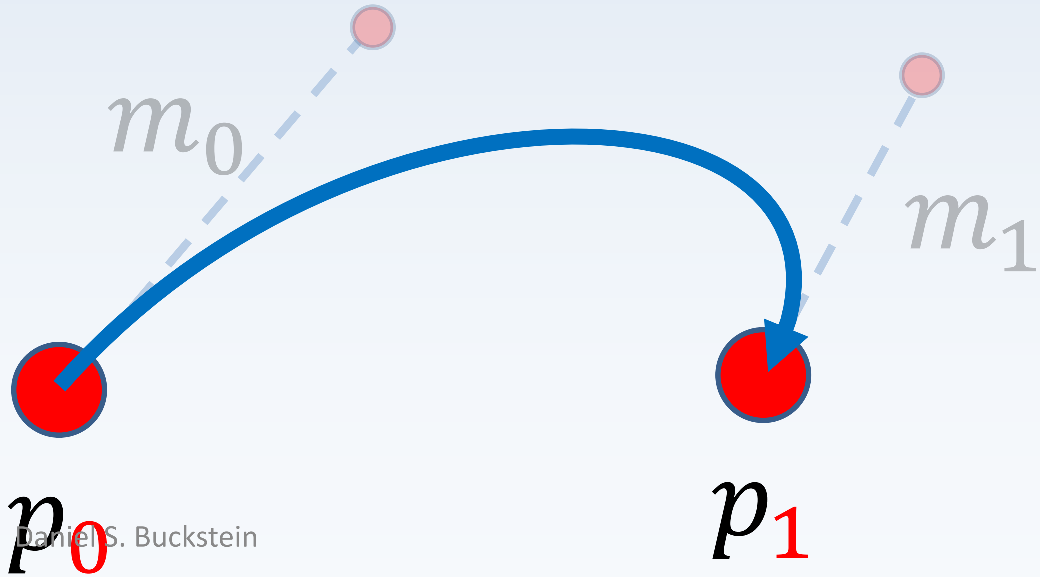
Curves & Spline Interpolation

- For different kinds of curves, the actual *values* that we use to compute curves may not be *on the curve itself*
- These are called *control values* or *handles*



Curves & Spline Interpolation

- For different kinds of curves, control values or handles define the *rate of change* or *tangent* of the curve
- This is then used to define the curve itself



Bézier Interpolation

- Named after Pierre Bézier (French)
- Very interesting math that is easy to code…
- …if we have a solid understanding of LERP!

- Concept: linearly interpolating linearly interpolated values…
- …but not the same as bilinear interpolation

Bézier Interpolation

- Let’s start with the basics:
- Like LERP, ***Bézier*** is a function of *one* control parameter ‘*t*’
- When *t=0*, we are at the beginning of the curve
- When *t=1*, we are at the end of the curve
- Same principle… so what’s different?

Bézier Interpolation

- How many influences do we have on a Bézier spline?
- LERP has *two* influences for any segment
- Catmull-Rom has *four* influences per segment
- Bézier???

- …as many as we want!

Bézier Interpolation

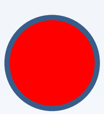
- Bézier curves only *pass through* the first and last control point; the rest are just *handles*
- Just as Catmull-Rom uses additional handles *beyond* the waypoints we are interpolating
- Bézier uses additional handles *between* the two endpoints
- Easiest to explain Bézier curves if we begin with a single point

Bézier Interpolation

- Bézier is a function of *t* generally defined as

$$\text{Bezier}(t) = B(t)$$

- A Bézier spline with only one control point is just that point (order zero):



p_0

$$B_{p_0}(t) = p_0$$

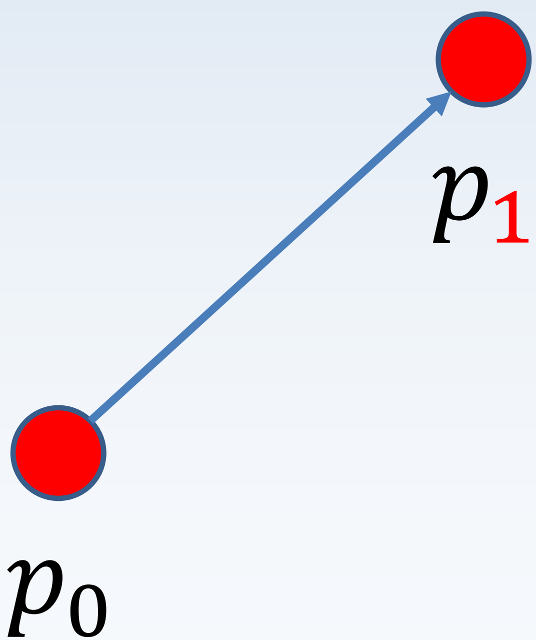
Bézier Interpolation

- A Bézier spline with *two* control points is just a line (first-order):

$$B_{p_0 p_1}(t) = \underline{(1-t)p_0 + tp_1}$$

wait… that seems pretty familiar…

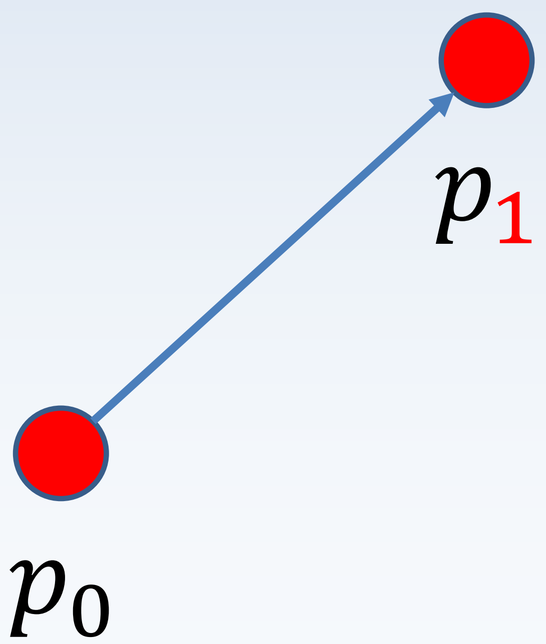
what is this???



Bézier Interpolation

- First-order Bézier interpolation is just LERP:

$$B_{p_0p_1}(t) = \mathrm{lerp}_{p_0p_1}(t) = \mathrm{lerp}(\underline{p_0, p_1,}\, t)$$

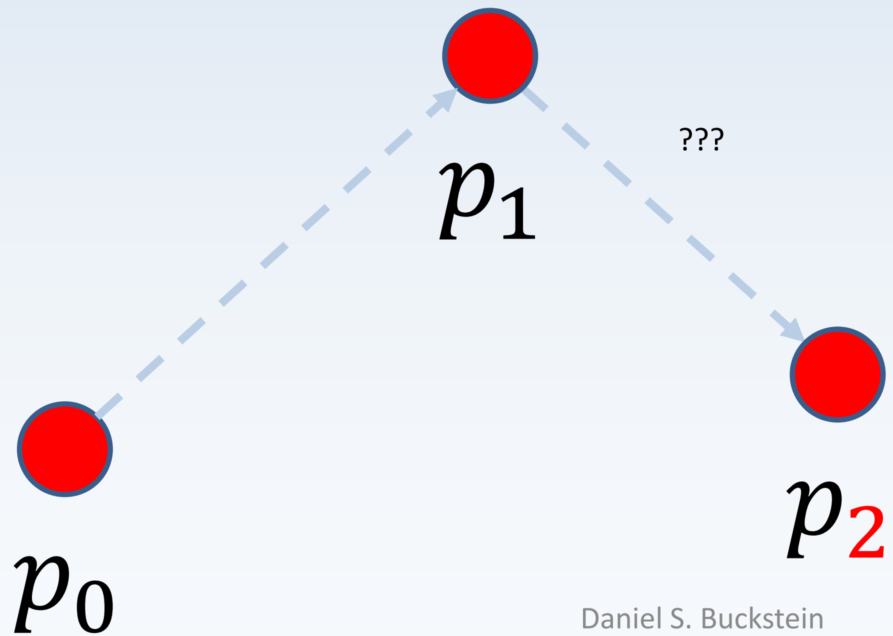


…something's familiar about this too…

…let's move on…

Bézier Interpolation

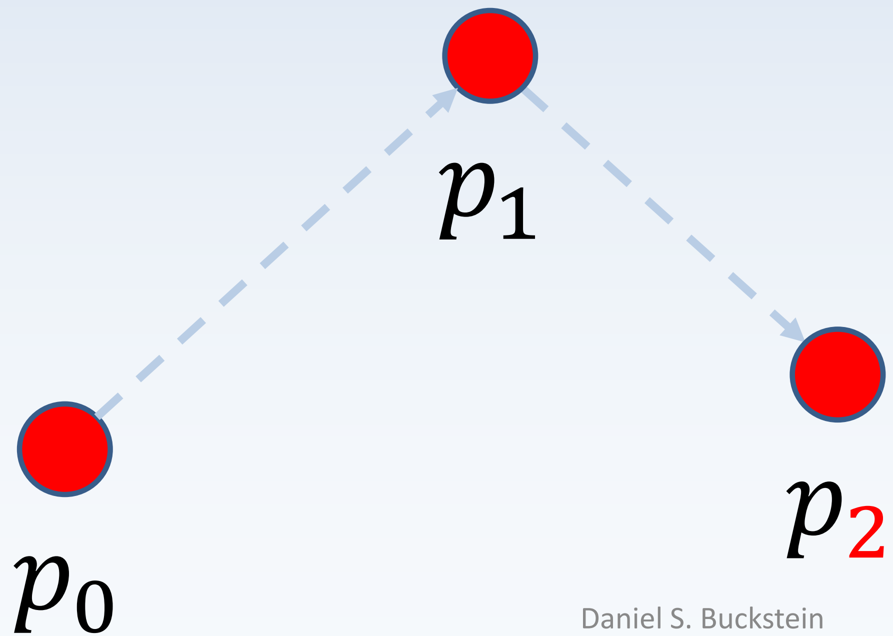
- Alas, the core problem: how do we compute *higher-order* Bézier interpolation?
- I.e. what happens if *p2* is the *endpoint* and *p1* is a *handle*?



Bézier Interpolation

- The solution:

1. LERP along each segment simultaneously
2. LERP the results using the *same 't' value!*

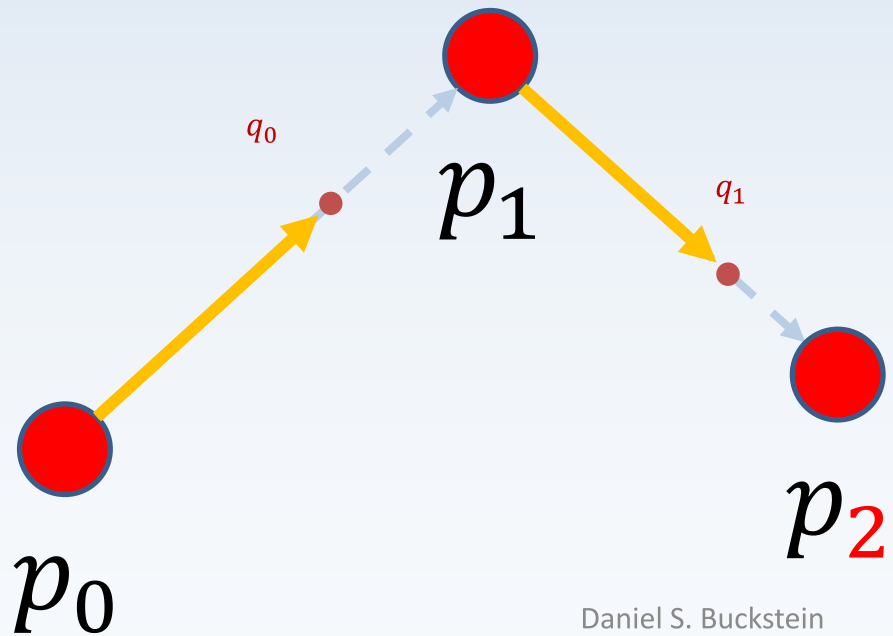


Bézier Interpolation

- The solution:

1. LERP along each segment simultaneously
2. LERP the results using the *same 't' value!*

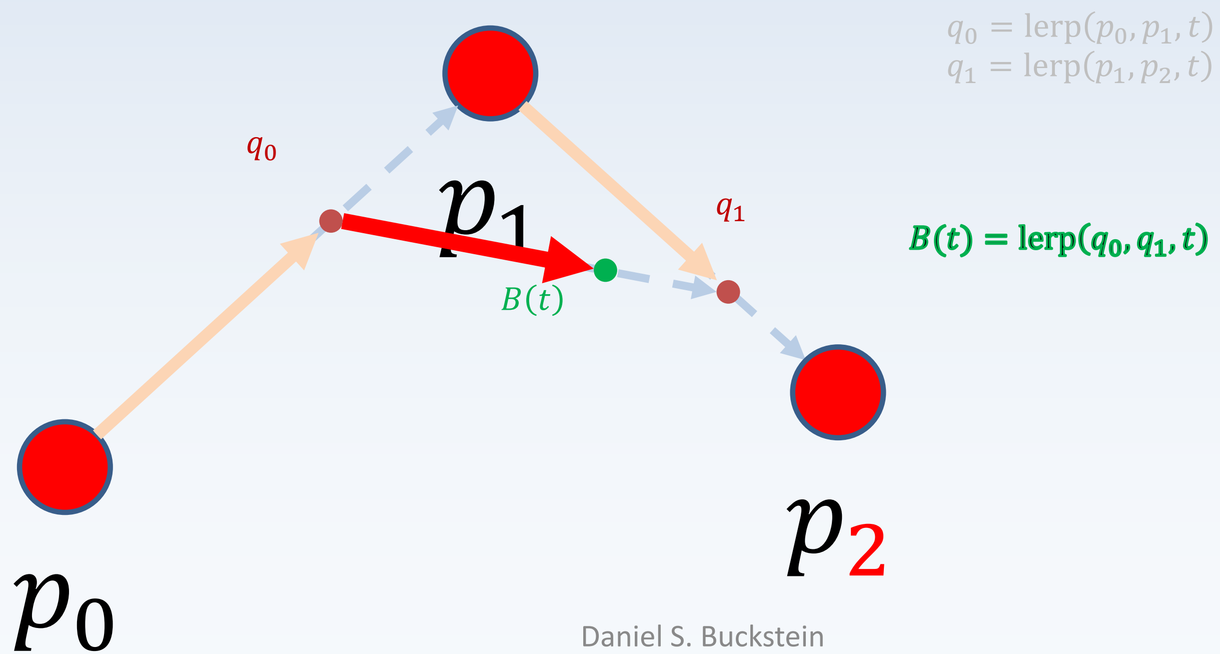
$$q_0 = \mathrm{lerp}(p_0, p_1, t)$$
$$q_1 = \mathrm{lerp}(p_1, p_2, t)$$



Bézier Interpolation

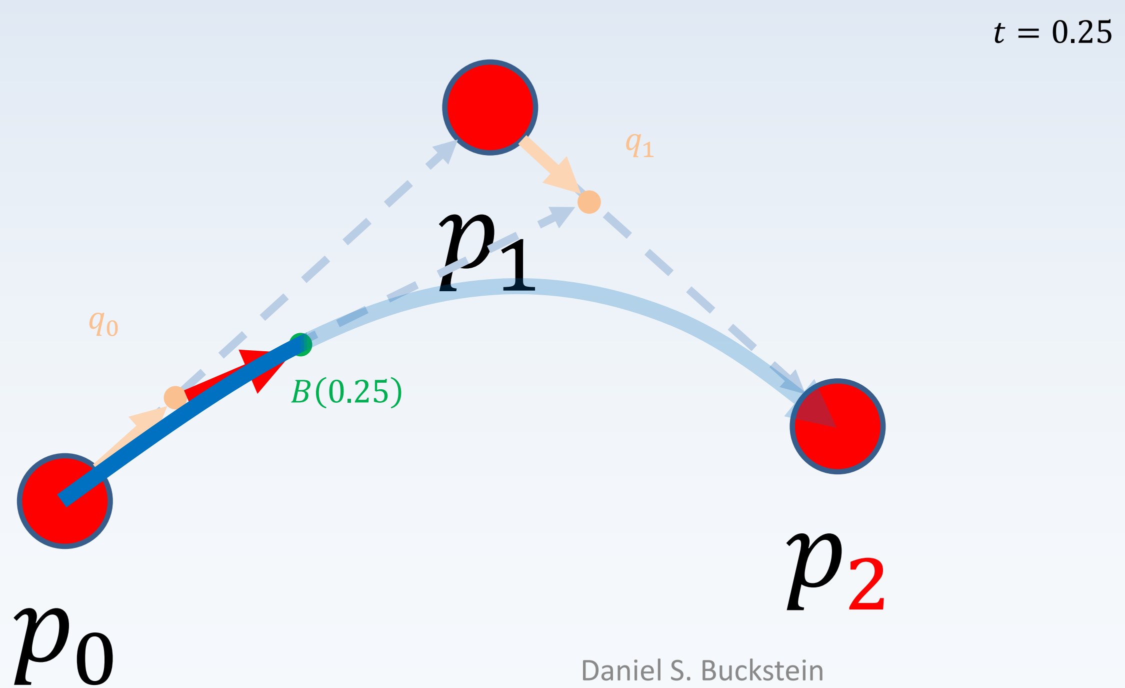
- The solution:

1. LERP along each segment simultaneously
2. LERP the results using the *same 't' value!*



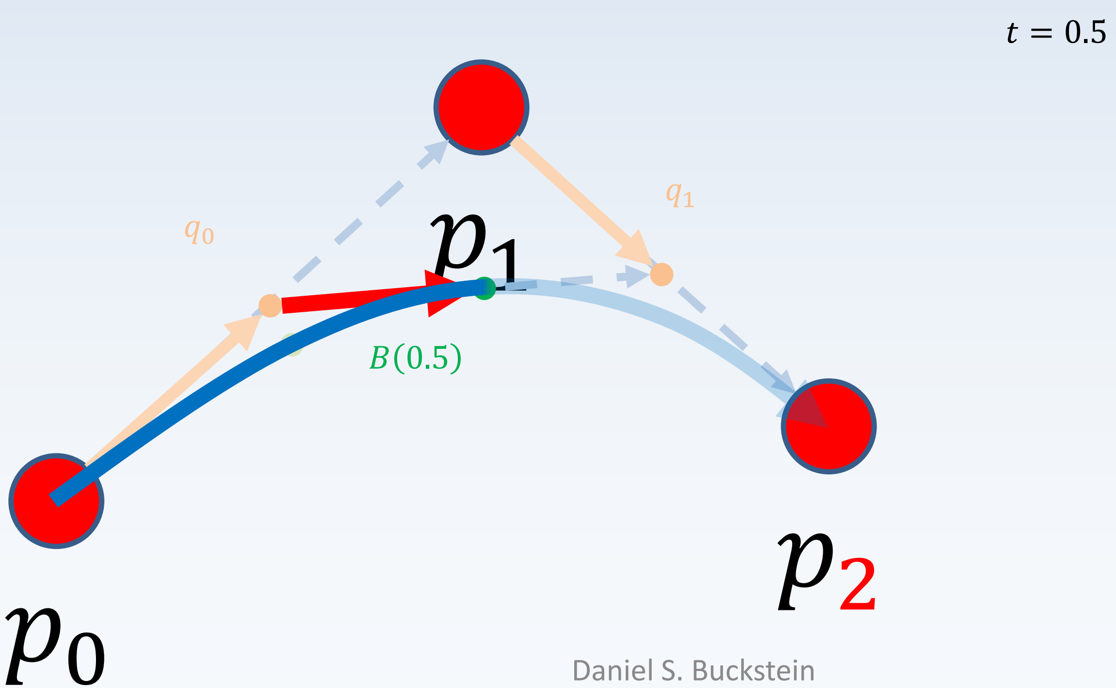
Bézier Interpolation

- Example curve formation (sampling at *t=0.25, t=0.5, t=0.75* and *t=1*)



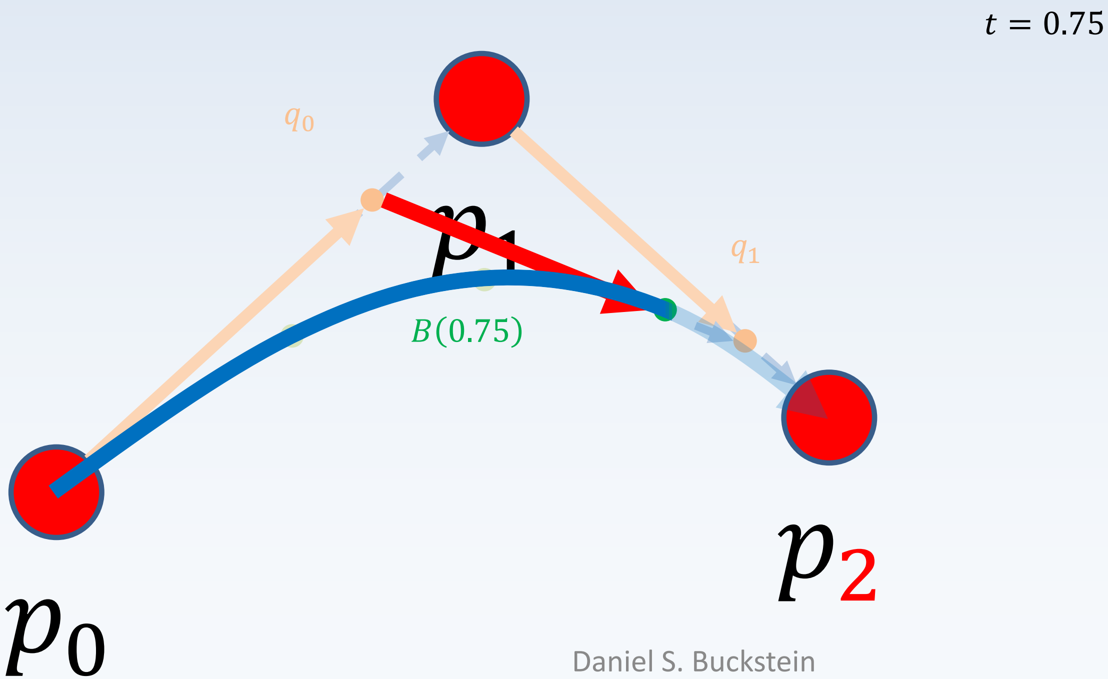
Bézier Interpolation

- Example curve formation (sampling at *t=0.25, t=0.5, t=0.75* and *t=1*)



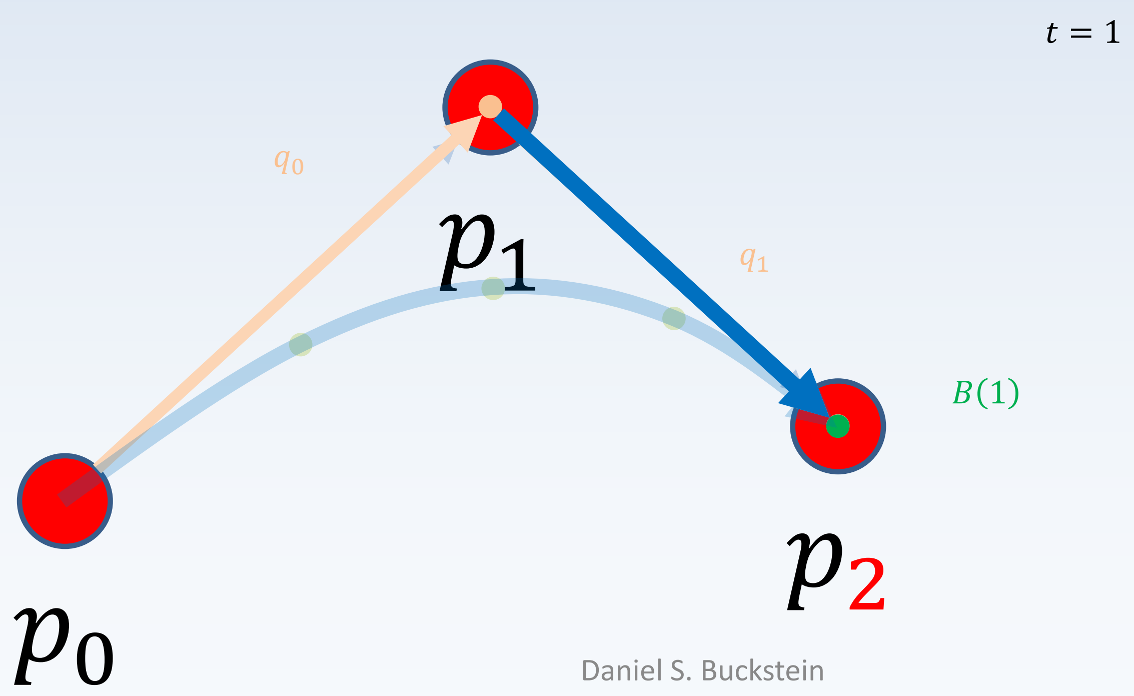
Bézier Interpolation

- Example curve formation (sampling at *t=0.25, t=0.5, t=0.75* and *t=1*)



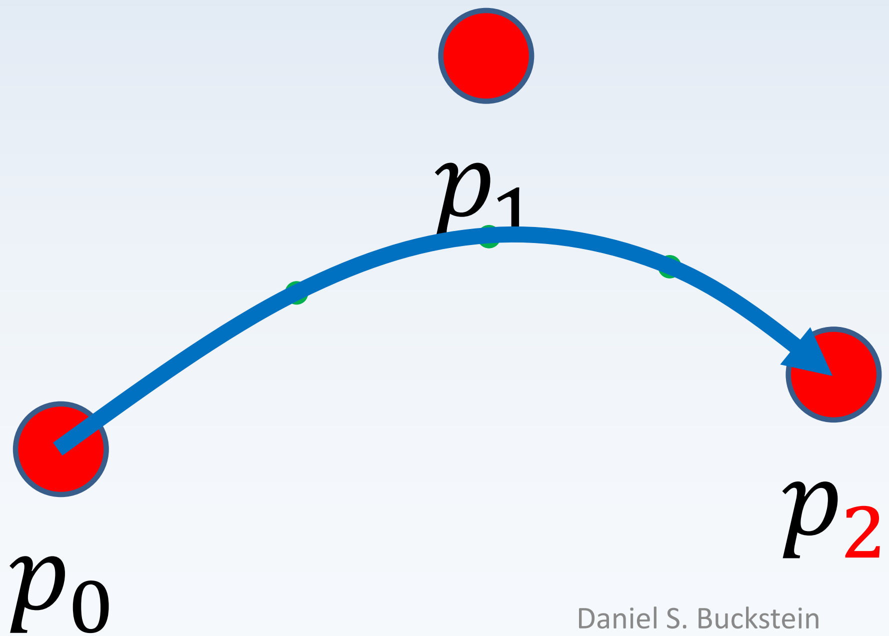
Bézier Interpolation

- Example curve formation (sampling at *t=0.25, t=0.5, t=0.75* and *t=1*)



Bézier Interpolation

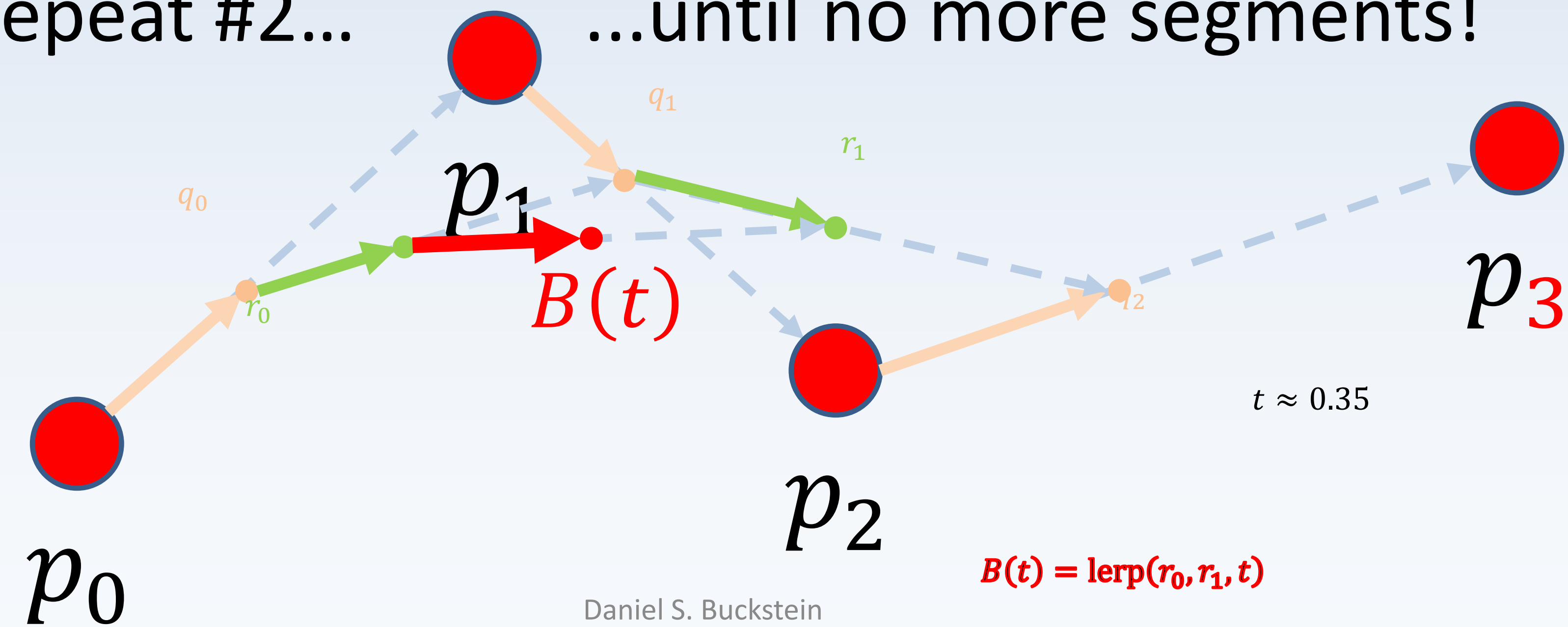
- Example curve formation (sampling at *t=0.25, t=0.5, t=0.75* and *t=1*)



Bézier Interpolation

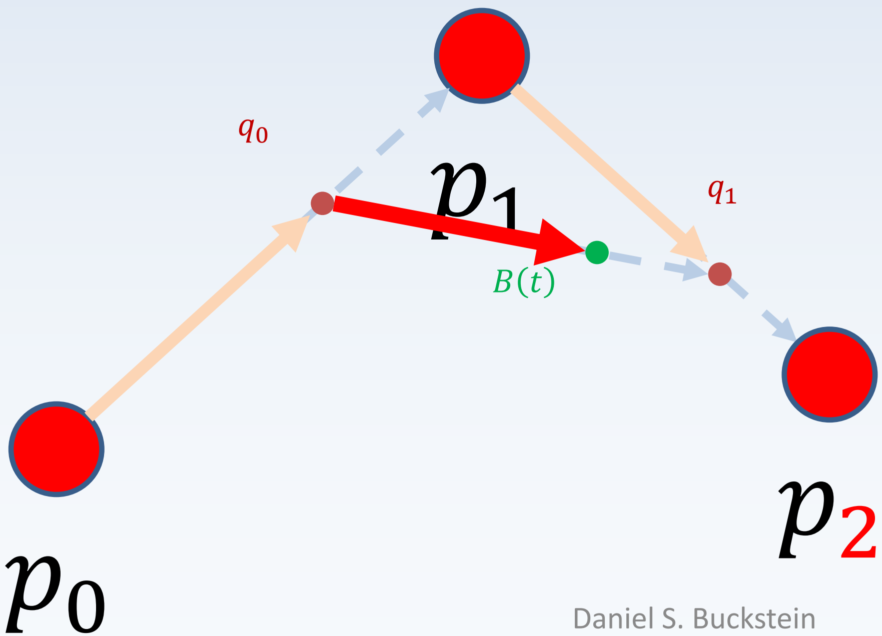
- Same principle for increased curve order:

1. LERP along each segment simultaneously
2. LERP the results using the *same 't' value!*
3. Repeat #2… …until no more segments!



Bézier Interpolation

- Hold on… back to quadratic…
- If the Bézier function is just LERP of *q0* and *q1*
- …and *q0* and *q1* are just LERP of *p* values…



$$q_0 = \mathrm{lerp}(p_0, p_1, t)$$
$$q_1 = \mathrm{lerp}(p_1, p_2, t)$$

$$B(t) = \mathrm{lerp}(q_0, q_1, t)$$

Bézier Interpolation

- …what happens if we *substitute* all of these variables wherever they appear in some function???

$$q_0 = \mathrm{lerp}(p_0, p_1, t)$$
$$q_1 = \mathrm{lerp}(p_1, p_2, t)$$

$$\begin{aligned} B_{p_0 p_1 p_2}(t) &= \mathrm{lerp}(q_0, q_1, t) \\ &= \mathrm{lerp}(\mathrm{lerp}(p_0, p_1, t), \mathrm{lerp}(p_1, p_2, t), t) \end{aligned}$$

Bézier Interpolation

- Wait a minute!!!
- Didn't we already determine that LERP is the same as first-order Bézier interpolation?

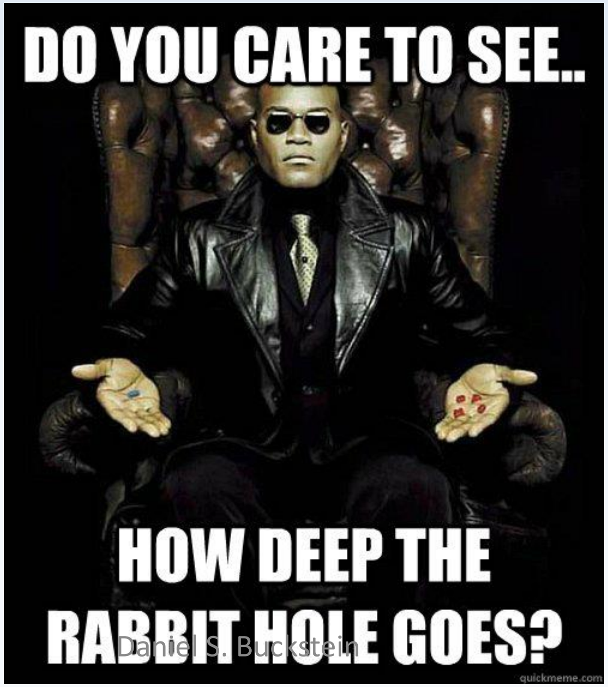
$$B_{p_0 p_1}(t) = \mathrm{lerp}(p_0, p_1, t)$$

$$B_{p_0 p_1 p_2}(t) = \mathrm{lerp}(\mathrm{lerp}(p_0, p_1, t), \mathrm{lerp}(p_1, p_2, t), t)$$

Bézier Interpolation

- Substitute again:

$$B_{p_0 p_1 p_2}(t) = \mathrm{lerp}\left(B_{p_0 p_1}(t), B_{p_1 p_2}(t), t\right)$$



Bézier Interpolation

- Remember that a *single point* is the definition of an *order-zero Bézier curve*:

$$B_{p_0}(t) = p_0$$

$$B_{p_0 p_1}(t) = \mathrm{lerp}(p_0, p_1, t)$$
$$= \mathrm{lerp}\left(B_{p_0}(t), B_{p_1}(t), t\right)$$

Bézier Interpolation

- Anyone see a pattern???
- Practically speaking, *what is Bézier interpolation actually doing???*

- ***Recursive LERP:***

$$B_{p_0 \ldots p_n}(t) = \mathrm{lerp}\left(B_{p_0 \ldots p_{n-1}}(t), B_{p_1 \ldots p_n}(t), t\right)$$

- Base case:

$$B_{p_0}(t) = p_0$$

Bézier Interpolation

- The number of control points involved with a Bézier curve defines the *curve order*

- The order is the *degree* of the polynomial we would end up with if we were to expand the algorithm

Bézier Interpolation

- Order zero (constant):

$$B_{p_0}(t) = (1-t)^0 t^0 p_0$$

- First-order (linear):

$$B_{p_0 p_1}(t) = (1-t)^1 t^0 p_0 + (1-t)^0 t^1 p_1$$

Bézier Interpolation

- Second-order (quadratic):

$$B_{p_0 p_1 p_2}(t)$$
$$= (1-t)^2 t^0 p_0 + 2(1-t)^1 t^1 p_1 + (1-t)^0 t^2 p_2$$

- Third-order (cubic):

$$B_{p_0 p_1 p_2 p_3}(t)$$
$$= (1-t)^3 p_0 + 3(1-t)^2 t p_1 + 3(1-t) t^2 p_2 + t^3 p_3$$

Bézier Interpolation

- Pure mathematical definition for interpolation on a Bézier curve of order n:

$$B_{p_0 \dots p_n}(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i p_i$$

where $\binom{n}{i}$ is the binomial coefficient

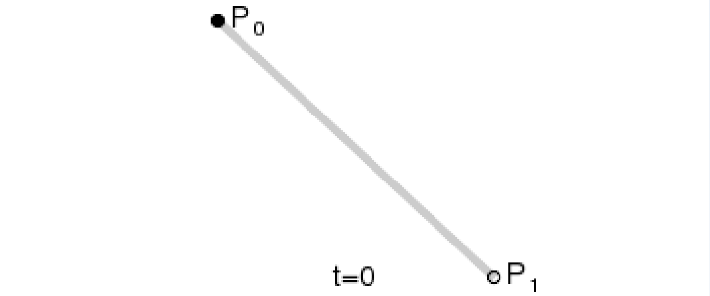
- See "*Pascal's triangle*" (great tool)

Bézier Interpolation

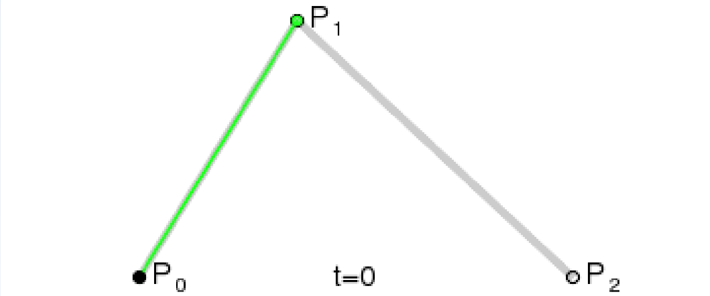
- A *Bézier curve* is created when we perform continuous Bézier interpolation
- The curve is just a visualization of all the values being interpolated
- This applies to all types of curves actually…
- Performing interpolation continuously will result in a mapping of all the points!

Bézier Interpolation

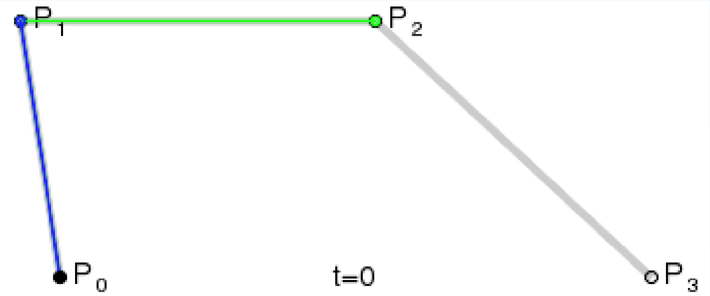
- Here are a bunch of Bézier interpolations in action to create curves!!! (gifs from Wikipedia)



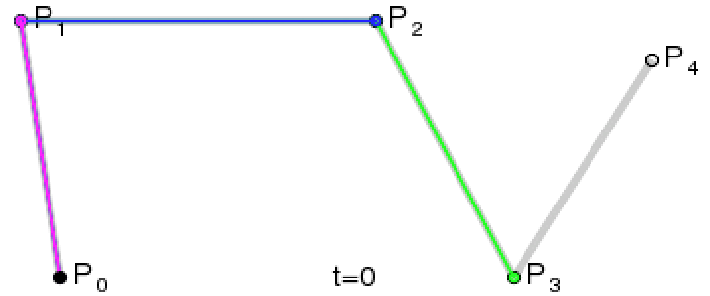
Linear



Quadratic



Cubic



Quartic

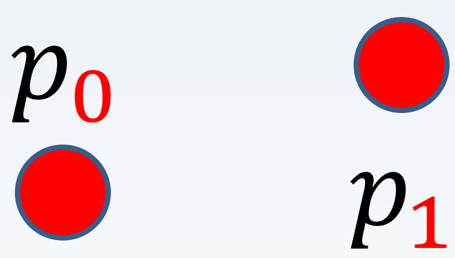
Bézier Interpolation

- ***Note***: Bézier interpolation only explicitly includes two endpoints
- Bézier curves can have any number of influences!!!

- When *t=0*, the result will always be p_0
- When *t=1*, the result will always be p_n, which is *the last control value in the set*

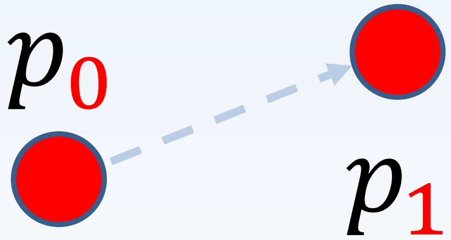
Catmull-Rom Interpolation

- ***Catmull-Rom interpolation***
- Developed by Edwin “Ed” Catmull (big name at *Pixar*) and Raphael Rom in 1974
- Originally a computer graphics technique for defining smooth surfaces between vertices
- *Solving the in-betweens given known points!!!*



Catmull-Rom Interpolation

- Advantage of Catmull-Rom curve:
- The values we are interpolating ***are part of the curve***; i.e. keyframes are still keyframes
- These are also called *waypoints* or *edit points*
- So we are still computing a value that lies between (or on) *p0* and *p1*

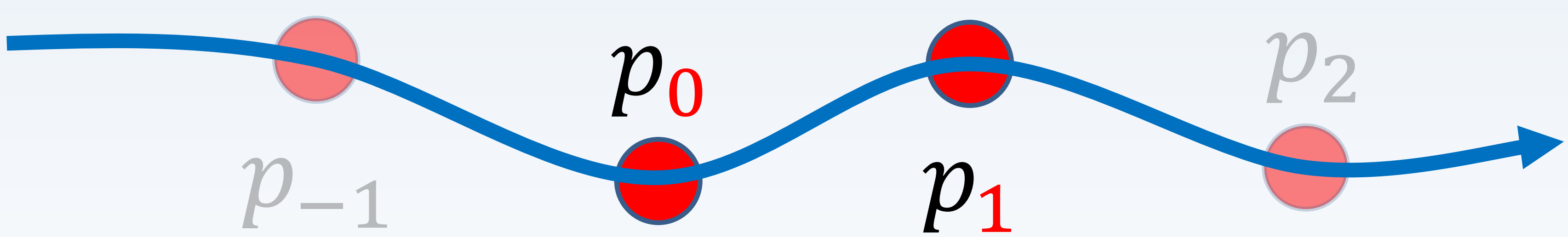


Catmull-Rom Interpolation

- Catmull-Rom uses *four* influences:
- The two that we are finding a value between (in this case *p0* and *p1*)
- The value after *p1*, we'll say *p2*, and…???
- The value before *p0*… we'll call it p(-1)

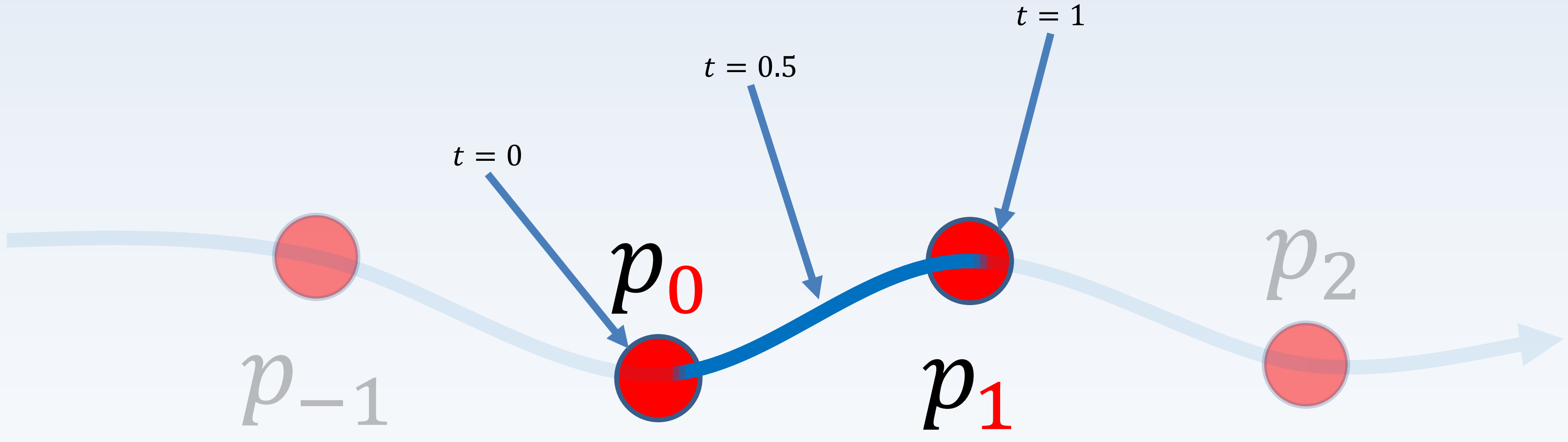
Catmull-Rom Interpolation

- Catmull-Rom uses *four* influences:
- All of these are *actual points on the path!!!*
- The result will be a smooth curve that passes through all the points!!!



Catmull-Rom Interpolation

- The Catmull-Rom function is also controlled by a normalized *t* value (control parameter)
- '*t*' has same behaviour as LERP: when *t=0*, we are at *p0*; when *t=1*, we are at *p1*



Catmull-Rom Interpolation

- The algorithm: polynomial of degree 3
- Clean representation: use matrices!

$$\mathrm{CatmullRom}_{p_{-1}p_0p_1p_2}(t)$$

$$= [p_{-1} \quad p_0 \quad p_1 \quad p_2]\underline{M_{CR}}\begin{bmatrix} t^0 \\ t^1 \\ t^2 \\ t^3 \end{bmatrix}$$

What are this?

Polynomial terms as a matrix

Influence matrix (insert keyframes here… just data!)

Catmull-Rom Interpolation

- The matrix M_{CR} is a preset kernel

$$M_{CR} = \mathbf{\frac{1}{2}}\begin{bmatrix} 0 & -1 & 2 & -1 \\ 2 & 0 & -5 & 3 \\ 0 & 1 & 4 & -3 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

- Ensures that the result is *p0* when *t=0* and the result is *p1* when *t=1*

Catmull-Rom Interpolation

- The matrix representation becomes:

$$\mathrm{CatmullRom}_{p_{-1}p_0p_1p_2}(t) =$$

$$\frac{\mathbf{1}}{\mathbf{2}}\begin{bmatrix} p_{-1} & p_0 & p_1 & p_2 \end{bmatrix}\begin{bmatrix} 0 & -1 & 2 & -1 \\ 2 & 0 & -5 & 3 \\ 0 & 1 & 4 & -3 \\ 0 & 0 & -1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix}$$

Catmull-Rom Interpolation

- Expand to solve:

$$\mathrm{CatmullRom}_{p_{-1}p_0p_1p_2}(t) =$$

$$\mathbf{\frac{1}{2}}\begin{bmatrix} p_{-1} & p_0 & p_1 & p_2 \end{bmatrix}\begin{bmatrix} -t + 2t^2 - t^3 \\ 2 - 5t^2 + 3t^3 \\ t + 4t^2 - 3t^3 \\ -t^2 + t^3 \end{bmatrix}$$

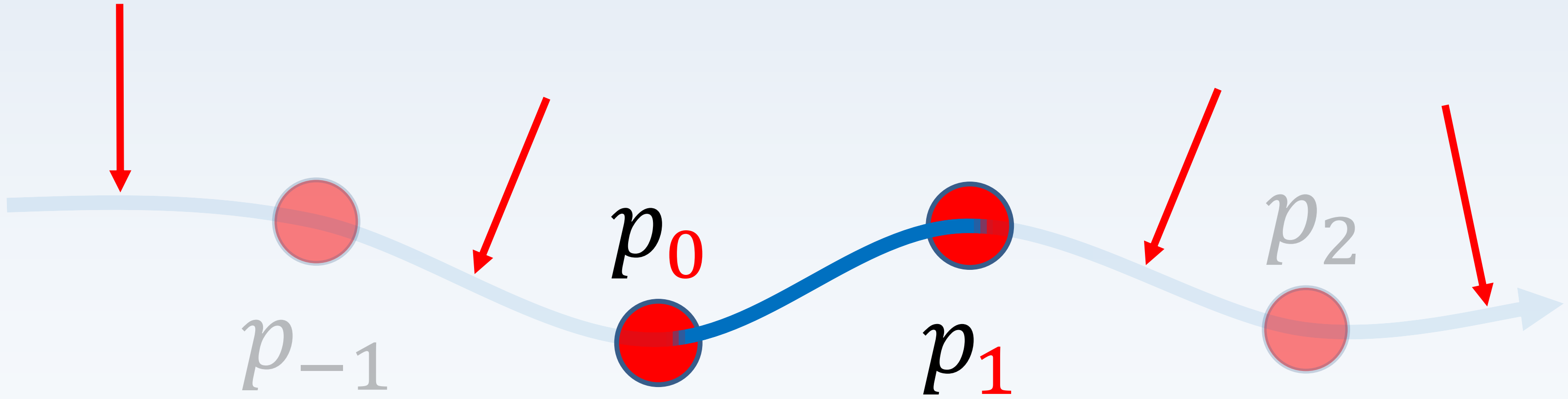
Catmull-Rom Interpolation

- The matrix representation is much more organized than the fully expanded algorithm (if we applied the matrix multiplication):

$$\mathrm{CatmullRom}_{p_{-1}p_0p_1p_2}(t)$$
$$= \frac{\mathbf{1}}{\mathbf{2}}[(-t + 2t^2 - t^3)p_{-1} + (2 - 5t^2 + 3t^3)p_0$$
$$+(t + 4t^2 - 3t^3)p_1 + (-t^2 + t^3)p_2]$$

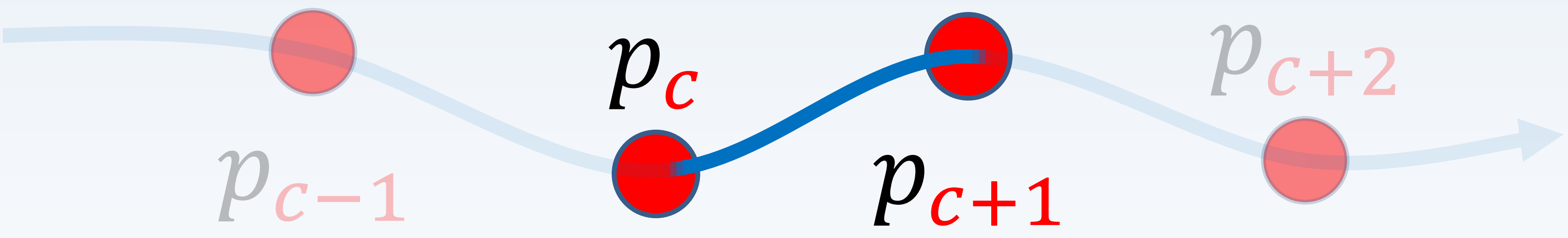
Catmull-Rom Interpolation

- Catmull-Rom gives us a value between the two main points of interest, *p0* and *p1*…
- …so how do we compute values on different path segments?



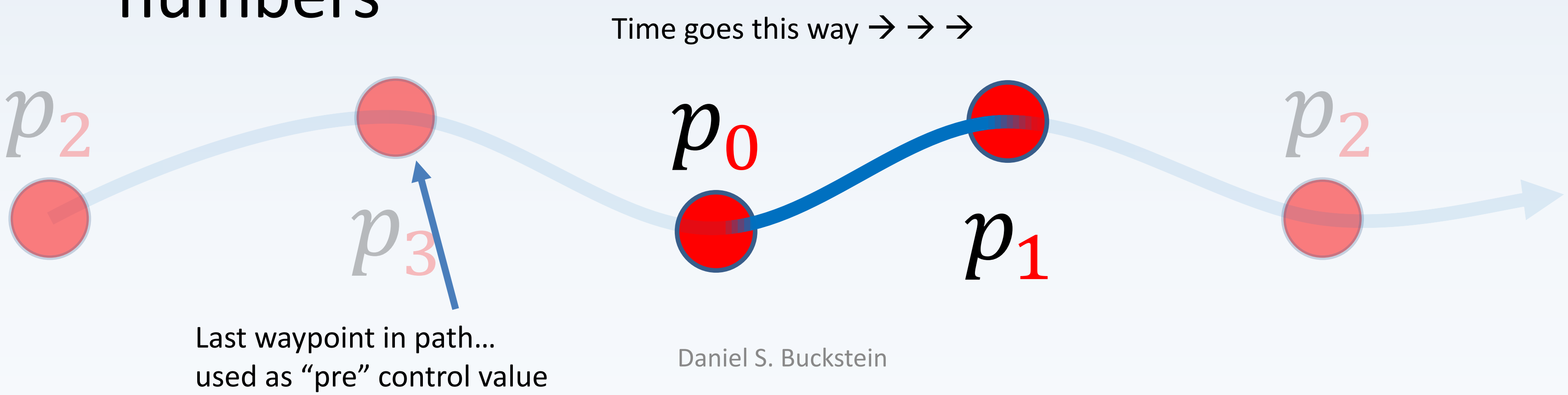
Catmull-Rom Interpolation

- ***"Current keyframe" controller***: exact same concept as linearly-segmented paths!!!
- Just store current index: '*c*'
- Controller determines next index, the following index, and the previous index based on sequence settings (looping, ping pong…)



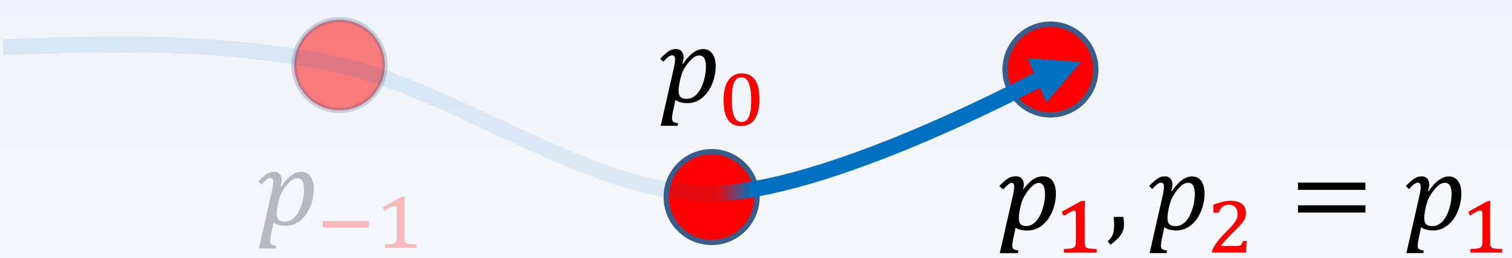
Catmull-Rom Interpolation

- How do we handle looping???
- When at the first value, use the last keyframe as our "pre" control value
- In this respect it helps to think of a path as a function of time, with waypoints as arbitrary numbers



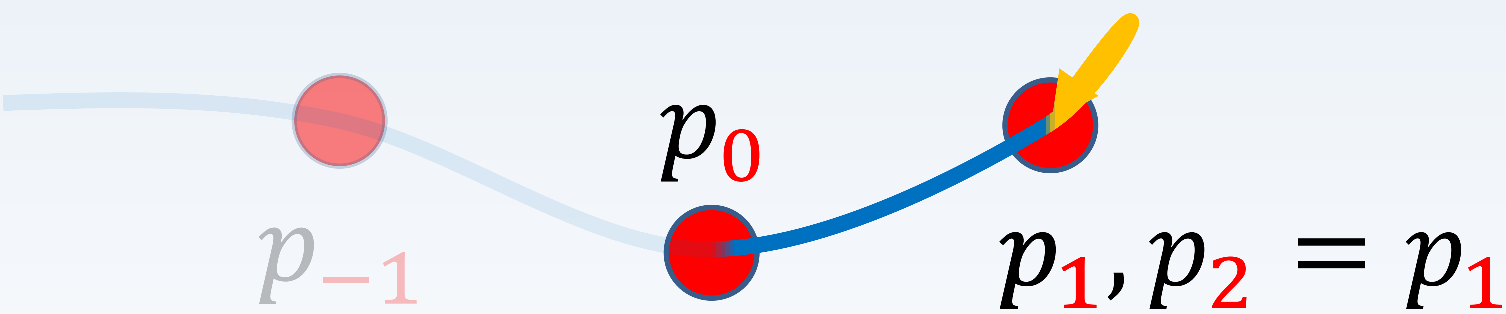
Catmull-Rom Interpolation

- What if we are approaching the end of the path and we are *not looping*???
- Why not use the last point for our *p2* argument?



Catmull-Rom Interpolation

- ***Do not overshoot with this method!!!***
- …because here's what the unseen part of the curve actually looks like!!!
- *Catmull-Rom curves will overshoot if you do not manage your keyframes properly!!!*



Cubic Hermite Spline

- Named after Charles Hermite (French)
- Commonly called "*csplines*"
- Cubic-polynomial-based algorithm
- Back to interpolating between a single segment between *p0* and *p1*
- Still a function of *t*

Cubic Hermite Spline

- Cubic Hermite splines have *four* influences…
- …but there is a key difference from what we've been dealing with so far:
- Only *two* of the influences are actual waypoints (*p0* and *p1*, the start and end values)
- What might the other two be???

Cubic Hermite Spline

- *Four* influences:
- p_0: the start value (current key)
- m_0: the *tangent* (rate of change) at *p0*
- p_1: the goal value
- m_1: the *tangent* at *p1*
- Instead of waypoints, csplines consider the *rates of change* at the end values!

Cubic Hermite Spline

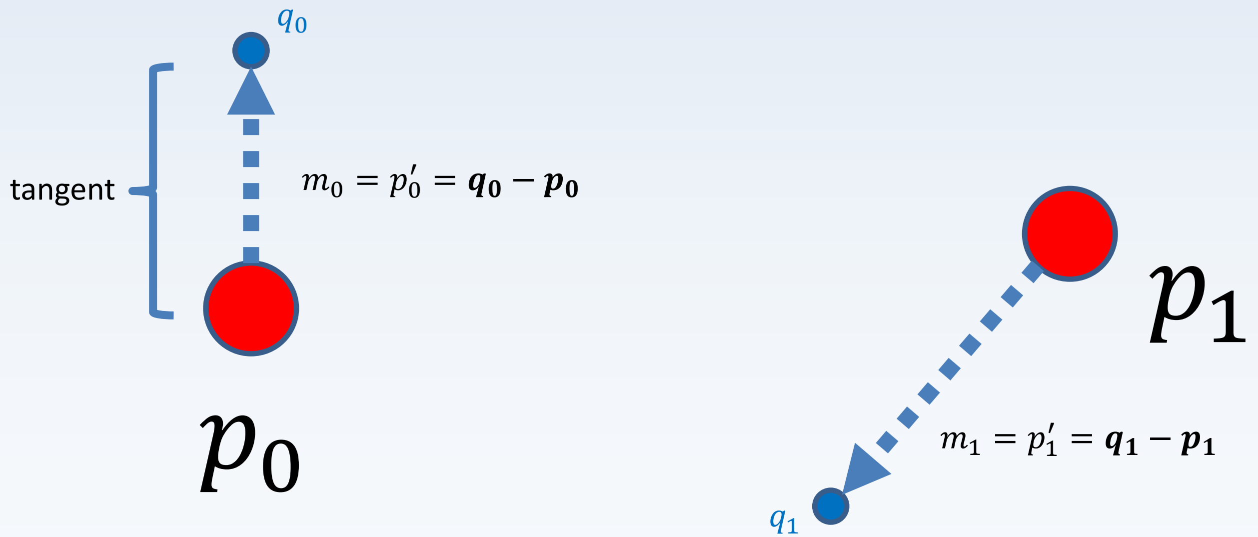
- Cubic Hermite spline (cspline) example
- I.e. what does it actually look like (before we get into the maths)
- Two *key values* that we want to interpolate between using a curve (*p0, p1*)

Cubic Hermite Spline

- Cubic Hermite spline (cspline) example
- The other two control values are not *points or values*…
- …they are the *rates of change* of the curve at *p0* and *p1*: *slope of the tangent* (*m*)

Cubic Hermite Spline

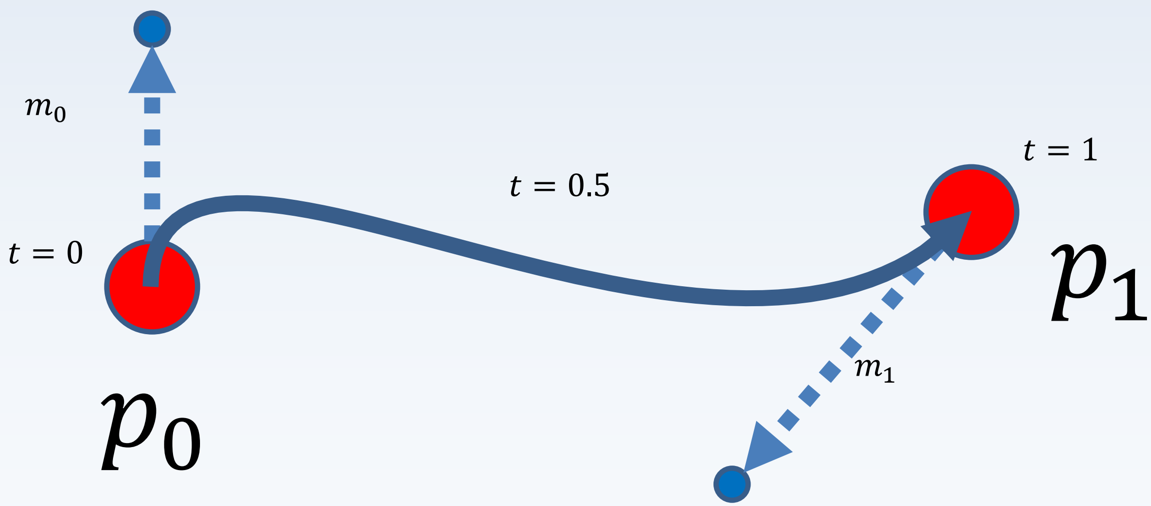
- Cubic Hermite spline (cspline) example
- How do we compute the tangent (rate of change) for any given value/point/key???
- Displacement to another *handle!*



Cubic Hermite Spline

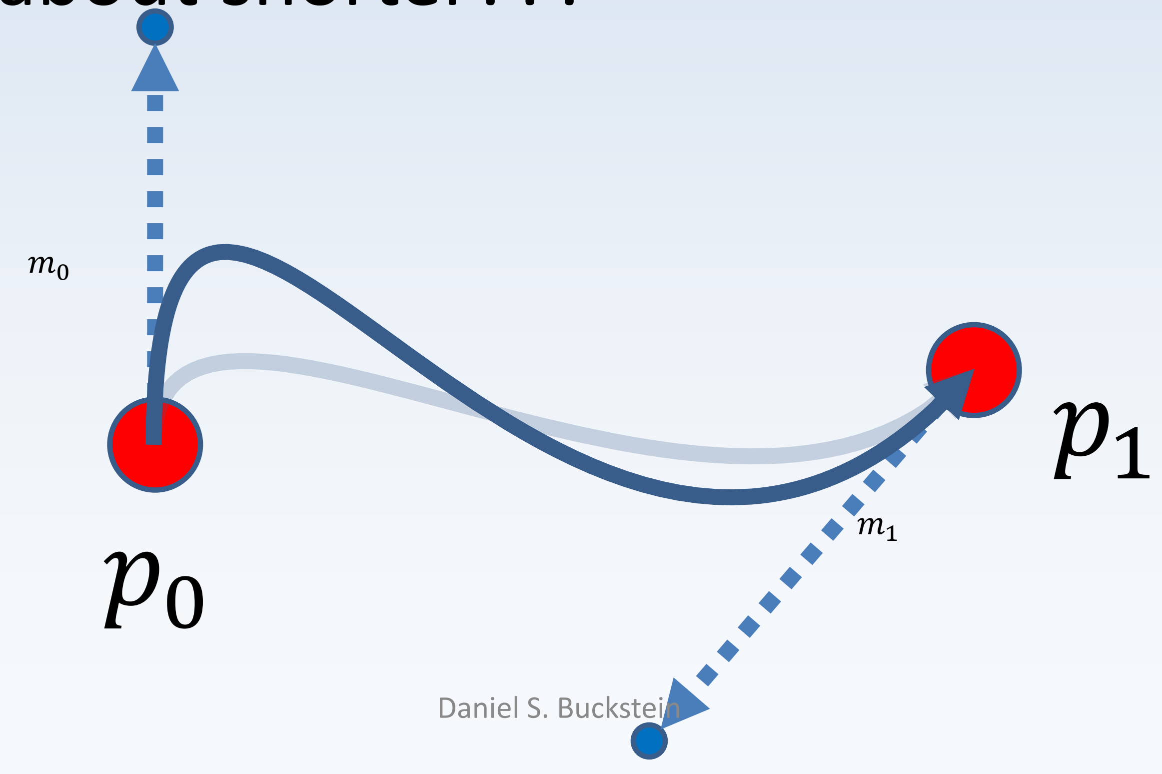
- Cubic Hermite spline (cspline) example
- Creating a spline:

$H(t)$



Cubic Hermite Spline

- Cubic Hermite spline (cspline) example
- What happens when tangents are longer???
- What about shorter???

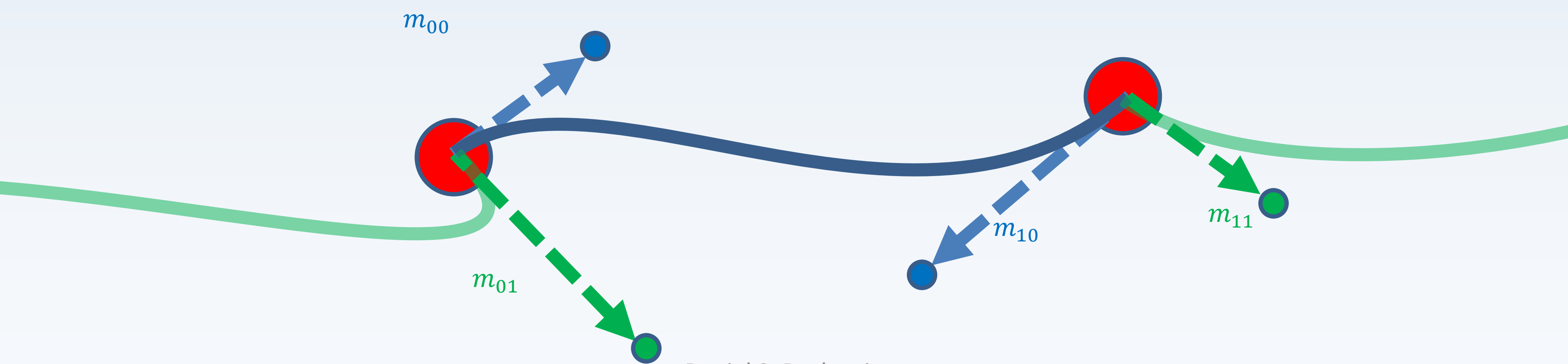


Cubic Hermite Spline

- Cubic Hermite spline (cspline) example
- You will often see a *bidirectional tangent*:
- Same effect, just used to manipulate other *segments* continuously! (i.e. paths)

Cubic Hermite Spline

- Cubic Hermite spline (cspline) example
- It is possible to connect multiple splines by breaking bidirectional tangents…
- …but you can figure this out!!!   ;)



Cubic Hermite Spline

- ***Cubic Hermite function:***
  $\mathrm{cspline}(t) = \mathrm{Hermite}_{p_0 m_0 p_1 m_1}(\mathrm{t}) = H(t)$

- Each control value (two keyframes and two rates of change at those keyframes) is paired with a *basis function*:

$$H(t) = h_{00}(t)p_0 + h_{10}(t)m_0 + h_{01}(t)p_1 + h_{11}(t)m_1$$

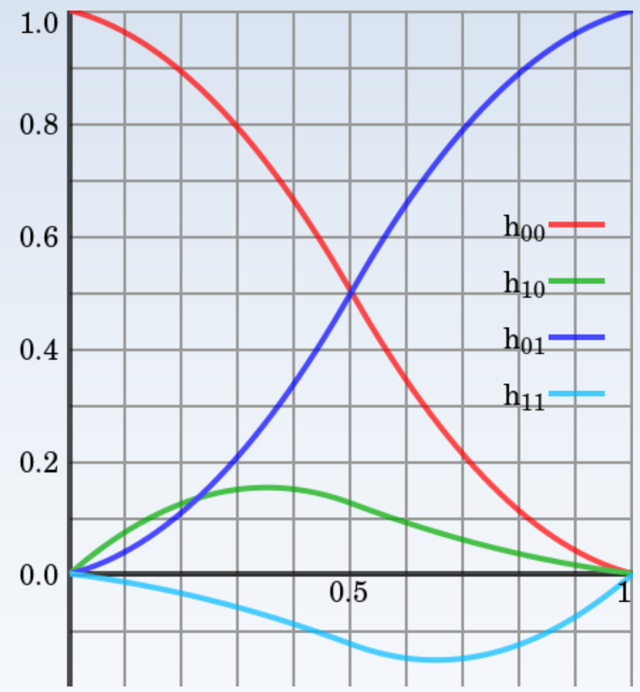
Cubic Hermite Spline

- ***Cubic Hermite function:***

$$H(t) = h_{00}(t)p_0 + h_{10}(t)m_0 + h_{01}(t)p_1 + h_{11}(t)m_1$$

- Basis functions:

$$h_{00}(t) = 1 - 3t^2 + 2t^3$$
$$h_{10}(t) = t - 2t^2 + t^3$$
$$h_{01}(t) = 3t^2 - 2t^3$$
$$h_{11}(t) = -t^2 + t^3$$



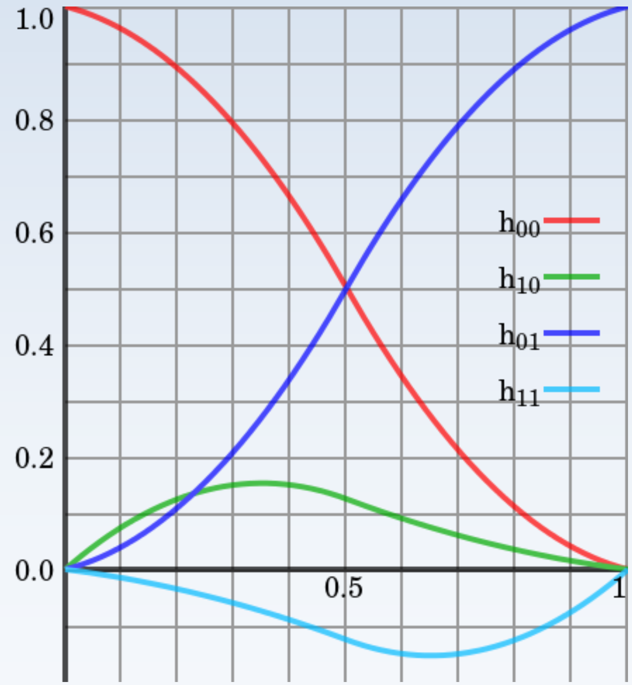
Cubic Hermite Spline

- ***Cubic Hermite function:***

$$H(t) = h_{00}(t)p_0 + h_{10}(t)m_0 + h_{01}(t)p_1 + h_{11}(t)m_1$$

- Basis functions (factored):

$$h_{00}(t) = (1 + 2t)(1 - t)^2$$
$$h_{10}(t) = t(1 - t)^2$$
$$h_{01}(t) = t^2(3 - 2t)$$
$$h_{11}(t) = t^2(t - 1)$$



Cubic Hermite Spline

- The above basis functions can be factored and expressed in matrix form:

$$\text{Hermite}_{p_0 m_0 p_1 m_1}(t)$$

$$= [p_0 \quad m_0 \quad p_1 \quad m_1] \color{red}{M_H} \begin{bmatrix} t^0 \\ t^1 \\ t^2 \\ t^3 \end{bmatrix}$$

Polynomial terms

Influence matrix

Cubic Hermite Spline

- The matrix M_H is a preset kernel

$$M_H = \begin{bmatrix} 1 & 0 & -3 & 2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 3 & -2 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

- Ensures that the result is *p0* when *t=0* and the result is *p1* when *t=1*

Cubic Hermite Spline

- The matrix representation becomes:

$$\text{Hermite}_{p_0 m_0 p_1 m_1}(t) =$$

$$\begin{bmatrix} p_0 & m_0 & p_1 & m_1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 & 2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 3 & -2 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix}$$

Cubic Hermite Spline

- Coincidentally, the Catmull-Rom formula is actually just a specific Hermite spline whose parameters have been simplified!
- Read this paper to find out how!

- http://graphics.cs.ucdavis.edu/~joy/ecs278/notes/Catmull-Rom-Spline.pdf

The end.

- Questions? Comments? Concerns?