Game Architecture
Overview

» Averaging and Blending
» Interpolation
» Parametric Equations
» Parametric Curves and Splines
  including:
  ✐ Bézier splines (linear, quadratic, cubic)
  ✐ Cubic Hermite splines
  ✐ Catmull-Rom splines
  ✐ Cardinal splines
  ✐ Kochanek–Bartels splines
  ✐ B-splines
Averaging and Blending
Averaging and Blending

First, we start off with the basics.
I mean, really basic.
Let’s go back to grade school.

How do you average two numbers together?

\[(A + B) / 2\]
Averaging and Blending

Let’s change that around a bit.

\[
\frac{A + B}{2}
\]

becomes

\[
\frac{1}{2}A + \frac{1}{2}B
\]

i.e. “half of A, half of B”, or

“a 50/50 blend of A and B”
We can, of course, also blend A and B unevenly (with different **weights**):

\[(.35 \times A) + (.65 \times B)\]

In this case, we are blending “35% of A with 65% of B”. We can use any blend weights we want, as long as they add up to 1.0 (100%).
Averaging and Blending

So if we generalized it, we would say:

$$(s \times A) + (t \times B)$$

...where $s$ is “how much of $A$” we want, and $t$ is “how much of $B$” we want

...and $s + t = 1.0$ (really, $s$ is just $1-t$)

so: $$((1-t) \times A) + (t \times B)$$

Which means we can control the balance of the entire blend by changing just one number: $t$
Averaging and Blending

There are **two ways** of thinking about this
(and a formula for each):

#1: “Blend some of $A$ with some of $B$”
\[
(s \times A) + (t \times B) \quad \text{← where } s = 1 - t
\]

#2: “Start with $A$, and then add some amount of the distance from $A$ to $B$”
\[
A + t \times (B - A)
\]
Averaging and Blending

In both cases, the result of our blend is just plain “A” if \( t = 0 \); i.e. if we don’t want any of B.

\[
(1.00 \times A) + (0.00 \times B) = A
\]

or:

\[
A + 0.00\times(B - A) = A
\]
Averaging and Blending

Likewise, the result of our blend is just plain “B” if $t=1$; i.e. if we don’t want any of $A$.

\[(0.00 \times A) + (1.00 \times B) = B\]

or:

\[A + 1.00 \times (B - A) = A + B - A = B\]
Averaging and Blending

However we choose to think about it, there’s a single “knob”, called $t$, that we are tweaking to get the blend of $A$ and $B$ that we want.
Blending Compound Data
Blending Compound Data

We can blend more than just simple numbers!

Blending 2D or 3D vectors, for example, is a cinch:

\[ \mathbf{P} = (s \mathbf{A}) + (t \mathbf{B}) \quad \leftarrow \quad \text{where } s = 1-t \]

Just blend each component \((x,y,z)\) separately, at the same time.

\[
\begin{align*}
\mathbf{P}_x &= (s \mathbf{A}_x) + (t \mathbf{B}_x) \\
\mathbf{P}_y &= (s \mathbf{A}_y) + (t \mathbf{B}_y) \\
\mathbf{P}_z &= (s \mathbf{A}_z) + (t \mathbf{B}_z)
\end{align*}
\]
Blending Compound Data
(such as Vectors)
Blending Compound Data

(such as Vectors)
Blending Compound Data
(such as Vectors)
Blending Compound Data
(such as Vectors)
Blending Compound Data

Need to be careful, though!

Not all compound data types will blend correctly with this approach.

Examples: Color RGBs, Euler angles (yaw/pitch/roll), Matrices, Quaternions...

...in fact, there are a bunch that won’t.
Blending Compound Data

Here’s an RGB color example:

If A is \textcolor{red}{\textbf{RGB}( 255, 0, 0 )} – bright red
...and B is \textcolor{green}{\textbf{RGB}( 0, 255, 0 )} – bright green

Blending the two (with $t = 0.5$) gives:
\textcolor{gold}{\textbf{RGB}( 127, 127, 0 )}
...which is a \textcolor{gold}{dull, swampy color}. Yuck.
Blending Compound Data

What we **wanted** was this:

...and what we got instead was this:
Blending Compound Data

For many compound classes, like RGB, you may need to write your own Blend() method that “does the right thing”, whatever that may be.

(For example, when interpolating RGBs you might consider converting to HSV, blending the hue, then converting back to RGB at the end.)

Will talk later about what happens when you try to blend Euler Angles (yaw/pitch/roll), Matrices, and Quaternions using this simple “naïve” approach of blending the components.
Interpolation
Interpolation

**Interpolation** (also called “Lerping”) is just changing blend weights to do **blending over time**.

i.e. Turning the knob (t) progressively, not just setting it to some position.

Often we crank slowly from t=0 to t=1.
Interpolation

In our Main Loop we usually have some Update() method that gets called, in which we have to decide what we're supposed to look like at this instant in time.

There are two main ways of approaching this when we're interpolating:

#1: Blend from A to B over the course of several frames (*parametric evaluation*);

#2: Blend one step forward from wherever-I’m-at now to wherever-I’m-going (*numerical integration*).
Interpolation

Games generally need to use both.

Most physics tends to use method #2 (numerical integration).

Many other systems, however, use method #1 (parametric evaluation).

(More on that in a moment)
Interpolation

We use “lerping” all the time, under different names.

For example:

an Audio crossfade
Interpolation

We use “lerping” all the time, under different names.

For example:

an Audio crossfade or this simple PowerPoint effect.
Interpolation

Basically:

whenever we do any sort of **blend over time**

we’re **lerping** (interpolating)
A simple **parametric equation** is one that has been rewritten so that it has one clear “input” parameter (variable) that everything else is based in terms of:

\[
\text{DiagonalLine2D}( t ) = (t, t)
\]

or

\[
\text{Helix3D}( t ) = (\cos t, \sin t, t)
\]

In other words, a simple parametric equation is basically **anything you can hook up to a single knob**. It’s a formula that you can feed in a single number (the “knob” value, “t”, usually from 0 to 1), and the formula gives back the appropriate value for that particular “t”.

Think of it as a **function that takes a float and returns... whatever** (a position, a color, an orientation, etc.):

\[
\text{someComplexData} \ \text{ParametricEquation}( \text{float} \ t );
\]
\[ P(t) = (t, t\cos(t), t\sin(t)) \]
Parametric Equations

Essentially:

\[ P(t) = \text{some formula with “t” in it} \]

...as \( t \) changes, \( P \) changes
  \( (P \text{ depends upon } t) \)

\( P(t) \) can return any kind of value; whatever we want to interpolate, for instance.
  - Position (2D, 3D, etc.)
  - Orientation
  - Scale
  - Alpha
  - etc.
Parametric Equations

Example: \( P(t) \) is a 2D position...
Pick some value of \( t \), plug it in, see where \( P \) is!
Parametric Equations

Example: \( P(t) \) is a 2D position...
Pick some value of \( t \), plug it in, see where \( P \) is!
Parametric Equations

Example: \( P(t) \) is a 2D position...
Pick some value of \( t \), plug it in, see where \( P \) is!
Parametric Equations

Example: $P(t)$ is a 2D position...
Pick some value of $t$, plug it in, see where $P$ is!

$P_x = 5 \cdot \cos(2\pi \cdot t)$
$P_y = 5 \cdot \sin(2\pi \cdot t)$  
(parametric)
Parametric Equations

Example: \( P(t) \) is a 2D position...
Pick some value of \( t \), plug it in, see where \( P \) is!
Parametric Equations

Example: $P(t)$ is a 2D position... Pick some value of $t$, plug it in, see where $P$ is!

\[ P_x = 5 \cdot \cos(2\pi \cdot t) \]
\[ P_y = 5 \cdot \sin(2\pi \cdot t) \]
Parametric Equations

Example: $P(t)$ is a 2D position...
Pick some value of $t$, plug it in, see where $P$ is!
Parametric Equations

Example: $P(t)$ is a 2D position...
Pick some value of $t$, plug it in, see where $P$ is!
Parametric Curves
Parametric Curves

Parametric curves are curves that are defined using parametric equations.
Parametric Curves

Here’s the basic idea:

We go from $t=0$ at $A$ (start) to $t=1$ at $B$ (end)
Parametric Curves

Set the knob to 0, and crank it towards 1
Parametric Curves

As we turn the knob, we keep plugging the latest $t$ into the curve equation to find out where $P$ is now.
Parametric Curves

Note: All parametric curves are directional; i.e. they have a start & end, a forward & backward
Parametric Curves

So that’s the basic idea.

Now how do we actually do it?
Bézier Curves

(pronounced “bay-zeeyay”)
Linear Bézier Curves

Bezier curves are the easiest kind to understand.

The simplest kind of Bezier curves are \textbf{Linear Bézier curves}.

They’re so simple, they’re not even curvy!
Linear Bézier Curves

\[ P = ((1-t) \times A) + (t \times B) \] // weighted average

or, as I prefer to write it:

\[ P = (s \times A) + (t \times B) \] ← where \( s = 1-t \)
Linear Bézier Curves

\[ P = ( (1-t) \ast A ) + (t \ast B) \] // weighted average

or, as I prefer to write it:

\[ P = (s \ast A) + (t \ast B) \] ← where \( s = 1-t \)
Linear Bézier Curves

\[ P = ((1-t) \cdot A) + (t \cdot B) \]  // weighted average

or, as I prefer to write it:

\[ P = (s \cdot A) + (t \cdot B) \]  \leftarrow \text{where } s = 1-t
Linear Bézier Curves

So, for $t = 0.75$ (75% of the way from $A$ to $B$):

$$P = ((1-t) \times A) + (t \times B)$$

or

$$P = (.25 \times A) + (.75 \times B)$$
Linear Bézier Curves

So, for $t = 0.75$ (75% of the way from $A$ to $B$):

$$P = ((1-t) \times A) + (t \times B)$$

or

$$P = (.25 \times A) + (.75 \times B)$$
Quadratic Bézier Curves
Quadratic Bézier Curves

A Quadratic Bezier curve is just:

a **blend of two Linear** Bezier curves.

The word “quadratic” means that if we sniff around the math long enough, we’ll see $t^2$. (In our Linear Beziers we saw $t$ and $1-t$, but never $t^2$).
Quadratic Bézier Curves

Three control points: A, B, and C
Quadratic Bézier Curves

Three control points: A, B, and C
Two different Linear Beziers: AB and BC
Quadratic Bézier Curves

Three control points: A, B, and C
Two different Linear Beziers: AB and BC
Instead of “P”, using “E” for AB and “F” for BC
Quadratic Bézier Curves

Interpolate $E$ along $AB$ as we turn the knob
Interpolate $F$ along $BC$ as we turn the knob
Move $E$ and $F$ simultaneously – only one “$t$”!
Quadratic Bézier Curves

Interpolate $E$ along $AB$ as we turn the knob
Interpolate $F$ along $BC$ as we turn the knob
Move $E$ and $F$ simultaneously – only one “t”!

t = .5
...for all segments!
Quadratic Bézier Curves

Interpolate E along AB as we turn the knob
Interpolate F along BC as we turn the knob
Move E and F simultaneously – only one “t”!
Quadratic Bézier Curves

Interpolate $E$ along $AB$ as we turn the knob
Interpolate $F$ along $BC$ as we turn the knob
Move $E$ and $F$ simultaneously – only one “$t$”!

$t = 1$ ...for all segments!
Quadratic Bézier Curves

Now let’s turn the knob again...
(from t=0 to t=1)
but **draw a line** between E and F as they move.
Quadratic Bézier Curves

Now let’s turn the knob again...
(from $t=0$ to $t=1$)
but **draw a line** between $E$ and $F$ as they move.
Quadratic Bézier Curves

Now let’s turn the knob again...
(from t=0 to t=1)
but **draw a line** between E and F as they move.
Quadratic Bézier Curves

Now let’s turn the knob again...
(from $t=0$ to $t=1$)
but **draw a line** between $E$ and $F$ as they move.
Quadratic Bézier Curves

Now let’s turn the knob again...
(from t=0 to t=1)
but **draw a line** between E and F as they move.
Quadratic Bézier Curves

This time, we’ll also **interpolate** $P$ from $E$ to $F$  
...using the same “$t$” as $E$ and $F$ themselves  
Watch **where $P$ goes**!
Quadratic Bézier Curves

This time, we’ll also **interpolate** $P$ from $E$ to $F$
...using the same “$t$” as $E$ and $F$ themselves
Watch **where $P$ goes**!
Quadratic Bézier Curves

This time, we’ll also **interpolate** P from E to F
...using the same “t” as E and F themselves
Watch **where P goes!**
Quadratic Bézier Curves

This time, we’ll also interpolate \( P \) from \( E \) to \( F \)...

...using the same “\( t \)” as \( E \) and \( F \) themselves

Watch where \( P \) goes!
Quadratic Bézier Curves

This time, we’ll also **interpolate P** from E to F
...using the same “t” as E and F themselves
Watch **where P goes!**
Quadratic Bézier Curves

Note that mathematicians use $P_0, P_1, P_2$ instead of $A, B, C$.

I will keep using $A, B, C$ here for simplicity and cleanliness.
Quadratic Bézier Curves

We know P starts at A, and ends at C. It is clearly influenced by B...

...but it never actually touches B
Quadratic Bézier Curves

B is a **guide point** of this curve; drag it around to change the curve’s contour.
Quadratic Bézier Curves

By the way, this is also that thing you were drawing in junior high when you were bored.

(when you weren’t drawing D&D maps, that is)
Quadratic Bézier Curves

By the way, this is also that thing you were drawing in junior high when you were bored.

(when you weren’t drawing D&D maps, that is)
Quadratic Bézier Curves

BONUS: This is also how they make True Type Fonts look nice and curvy.
Quadratic Bézier Curves

» Remember:

A Quadratic Bezier curve is just a **blend of two Linear** Bezier curves.

So the math is still pretty simple.

(Just a blend of two Linear Bezier equations.)
Quadratic Bézier Curves

\[ E(t) = (s \ast A) + (t \ast B) \]
\[ F(t) = (s \ast B) + (t \ast C) \]
\[ P(t) = (s \ast E) + (t \ast F) \]

- \( s = 1-t \)
- where \( s \)
- technically \( E(t) \) and \( F(t) \) here
Quadratic Bézier Curves

» Hold on! You said “quadratic” meant we’d see a $t^2$ in there somewhere.

$$E(t) = sA + tB$$
$$F(t) = sB + tC$$
$$P(t) = sE(t) + tF(t)$$

» $P(t)$ is an interpolation from $E(t)$ to $F(t)$
» When you plug the $E(t)$ and $F(t)$ equations into the $P(t)$ equation, you get...
Quadratic Bézier Curves

One equation to rule them all:

\[ E(t) = sA + tB \]
\[ F(t) = sB + tC \]
\[ P(t) = sE(t) + tF(t) \]

or

\[ P(t) = s(sA + tB) + t(sB + tC) \]

or

\[ P(t) = (s^2)A + (st)B + (st)B + (t^2)C \]

or

\[ P(t) = (s^2)A + 2(st)B + (t^2)C \]

(BTW, there’s our “quadratic” \( t^2 \))
Quadratic Bézier Curves

What if $t = 0$? (at the start of the curve)

so then... $s = 1$

$$P(t) = (s^2)A + 2(st)B + (t^2)C$$

becomes

$$P(t) = (1^2)A + 2(1*0)B + (0^2)C$$

becomes

$$P(t) = (1)A + 2(0)B + (0)C$$

becomes

$$P(t) = A$$
Quadratic Bézier Curves

What if \( t = 1 \) ? (at the end of the curve)

so then... \( s = 0 \)

\[
P(t) = (s^2)A + 2(st)B + (t^2)C
\]

becomes

\[
P(t) = (0^2)A + 2(0*1)B + (1^2)C
\]

becomes

\[
P(t) = (0)A + 2(0)B + (1)C
\]

becomes

\[
P(t) = C
\]
Non-uniformity

Be careful: most curves are not uniform; that is, they have variable “density” or “speed” throughout them.

(However, we can also use this to our advantage!)
Cubic Bézier Curves
Cubic Bézier Curves

A Cubic Bezier curve is just:

a **blend of two Quadratic** Bezier curves.

The word “cubic” means that if we sniff around the math long enough, we’ll see $t^3$. (In our Linear Beziers we saw $t$; in our Quadratics we saw $t^2$).
Cubic Bézier Curves

» Four control points: A, B, C, and D
» 3 different Linear Beziers: AB, BC, and CD
» 2 different Quadratic Beziers: ABC and BCD
Cubic Bézier Curves

As we turn the knob (one knob, one “t” for everyone):

Interpolate E along AB  // all three lerp simultaneously
Interpolate F along BC  // all three lerp simultaneously
Interpolate G along CD  // all three lerp simultaneously
As we turn the knob (one knob, one “t” for everyone):

- Interpolate E along AB  // all three lerp simultaneously
- Interpolate F along BC  // all three lerp simultaneously
- Interpolate G along CD  // all three lerp simultaneously
Cubic Bézier Curves

» As we turn the knob (one knob, one “t” for everyone):
  - Interpolate \( E \) along \( AB \) // all three lerp simultaneously
  - Interpolate \( F \) along \( BC \) // all three lerp simultaneously
  - Interpolate \( G \) along \( CD \) // all three lerp simultaneously
Cubic Bézier Curves

As we turn the knob (one knob, one “t” for everyone):

- Interpolate E along AB  // all three lerp simultaneously
- Interpolate F along BC  // all three lerp simultaneously
- Interpolate G along CD  // all three lerp simultaneously

\[ t = 0.75 \]
...for all segments!
As we turn the knob (one knob, one “t” for everyone):

- Interpolate E along AB  // all three lerp simultaneously
- Interpolate F along BC  // all three lerp simultaneously
- Interpolate G along CD  // all three lerp simultaneously
Cubic Bézier Curves

» Also:
  
  Interpolate \( Q \) along \( EF \)  // lerp simultaneously with \( E,F,G \)
  
  Interpolate \( R \) along \( FG \)  // lerp simultaneously with \( E,F,G \)
Cubic Bézier Curves

Also:

Interpolate Q along EF  // lerp simultaneously with E,F,G
Interpolate R along FG  // lerp simultaneously with E,F,G
Cubic Bézier Curves

» Also:

Interpolate Q along EF  // lerp simultaneously with E,F,G
Interpolate R along FG  // lerp simultaneously with E,F,G
Cubic Bézier Curves

» Also:

- Interpolate Q along EF // lerp simultaneously with E,F,G
- Interpolate R along FG // lerp simultaneously with E,F,G
Cubic Bézier Curves

Also:

Interpolate Q along EF // lerp simultaneously with E,F,G
Interpolate R along FG // lerp simultaneously with E,F,G
Cubic Bézier Curves

» And finally:
   Interpolate P along QR
   (simultaneously with E,F,G,Q,R)
» Again, watch where P goes!
Cubic Bézier Curves

» And finally:
  Interpolate $P$ along $QR$
  (simultaneously with $E,F,G,Q,R$)

» Again, watch **where $P$ goes**!
Cubic Bézier Curves

» And finally:
  Interpolate \( P \) along \( QR \)
  (simultaneously with \( E,F,G,Q,R \))

» Again, watch **where \( P \) goes**!
Cubic Bézier Curves

» And finally:
  Interpolate P along QR
  (simultaneously with E,F,G,Q,R)

» Again, watch where P goes!
Cubic Bézier Curves

» And finally:
   Interpolate $P$ along $QR$
   (simultaneously with $E,F,G,Q,R$)

Again, watch **where $P$ goes**!
Now P starts at A, and ends at D.
It never touches B or C...
so they are guide points.
Cubic Bézier Curves

» Remember:

A Cubic Bezier curve is just a blend of two Quadratic Bezier curves.

...which are just a blend of 3 Linear Bezier curves.

So the math is still not too bad.

(A blend of... blends of... Linear Bezier equations.)
Cubic Bézier Curves

\[ E(t) = sA + tB \]
\[ F(t) = sB + tC \]
\[ G(t) = sC + tD \]

\( t = .5 \)
...for all segments!

\( \text{where } s = 1-t \)
And then $Q$ and $R$ interpolate those results...

$Q(t) = sE + tF$

$R(t) = sF + tG$
Cubic Bézier Curves

And lastly $P$ interpolates from $Q$ to $R$

$$P(t) = sQ + tR$$
Cubic Bézier Curves

\[ E(t) = sA + tB \] \hspace{1cm} // Linear Bezier (blend of A and B)
\[ F(t) = sB + tC \] \hspace{1cm} // Linear Bezier (blend of B and C)
\[ G(t) = sC + tD \] \hspace{1cm} // Linear Bezier (blend of C and D)

\[ Q(t) = sE + tF \] \hspace{1cm} // Quadratic Bezier (blend of E and F)
\[ R(t) = sF + tG \] \hspace{1cm} // Quadratic Bezier (blend of F and G)

\[ P(t) = sQ + tR \] \hspace{1cm} // Cubic Bezier (blend of Q and R)

Okay! So let’s combine these all together...
Cubic Bézier Curves

Do some hand-waving mathemagic here...
...and we get **one equation to rule them all:**

\[
P(t) = (s^3)A + 3(s^2t)B + 3(st^2)C + (t^3)D
\]

(BTW, there’s our “cubic” \(t^3\))
Cubic Bézier Curves

However, I personally like this:

- \( E(t) = sA + tB \) // Linear Bezier (blend of A and B)
- \( F(t) = sB + tC \) // Linear Bezier (blend of B and C)
- \( G(t) = sC + tD \) // Linear Bezier (blend of C and D)
- \( Q(t) = sE + tF \) // Quadratic Bezier (blend of E and F)
- \( R(t) = sF + tG \) // Quadratic Bezier (blend of F and G)
- \( P(t) = sQ + tR \) // Cubic Bezier (blend of Q and R)

(besides, this one can be just as fast or faster!)

Better than this:

\[
P(t) = (s^3)A + 3(s^2t)B + 3(st^2)C + (t^3)D
\]
Quartic and Quintic Bézier Curves

By the way, you don’t have to stop with Cubic, either.

A Quartic \((t^4)\) Bezier curve is just a blend of two Cubic \((t^3)\) Bezier curves.

A Quintic \((t^5)\) Bezier curve is just a blend of two Quartic \((t^4)\) Bezier curves.

...and so on.

However, I find that cubic curves give you all the control you want in practice, and the higher order curves (quartic, quintic) usually aren’t worth their weight in math.

So let’s just stick with cubic, shall we?
Quartic and Quintic Bézier Curves

By the way, you don’t have to stop with Cubic, either.

A Quartic \( t^4 \) Bezier curve is just a blend of two Cubic \( t^3 \) Bezier curves. A Quintic \( t^5 \) Bezier curve is just a blend of two Quartic \( t^4 \) Bezier curves. ...and so on.

However, I find that cubic curves give you all the control you want in practice, and the higher order curves (quartic, quintic) usually aren’t worth their weight in math.

So let’s just stick with cubic, shall we?
Quartic and Quintic Bézier Curves

By the way, you don’t have to stop with Cubic, either.

A **Quartic** \((t^4)\) Bezier curve is just a blend of two Cubic \((t^3)\) Bezier curves.

A **Quintic** \((t^5)\) Bezier curve is just a blend of two Quartic \((t^4)\) Bezier curves.

...and so on.

**However**, I find that cubic curves give you all the control you want in practice, and the higher order curves (quartic, quintic) usually aren’t worth their weight in math.

So let’s just stick with cubic, shall we?
Cubic Bézier Curves

Let’s compare the three flattened Bezier equations (Linear, Quadratic, Cubic):

**Linear** \( (t) \) = \((s)A + (t)B\)

**Quadratic** \( (t) \) = \((s^2)A + 2(st)B + (t^2)C\)

**Cubic** \( (t) \) = \((s^3)A + 3(s^2t)B + 3(st^2)C + (t^3)D\)

There’s some nice symmetry here...
Cubic Bézier Curves

Write in all of the numeric coefficients...
Express each term as powers of $s$ and $t$

\[
P(t) = 1(s^1t^0)A + 1(s^0t^1)B
\]
\[
P(t) = 1(s^2t^0)A + 2(s^1t^1)B + 1(s^0t^2)C
\]
\[
P(t) = 1(s^3t^0)A + 3(s^2t^1)B + 3(s^1t^2)C + 1(s^0t^3)D
\]
Cubic Bézier Curves

Write in all of the \textit{numeric coefficients}...
Express each term as powers of $s$ and $t$

\begin{align*}
P(t) &= 1(s^1t^0)A + 1(s^0t^1)B \\
P(t) &= 1(s^2t^0)A + 2(s^1t^1)B + 1(s^0t^2)C \\
P(t) &= 1(s^3t^0)A + 3(s^2t^1)B + 3(s^1t^2)C + 1(s^0t^3)D
\end{align*}

Note: "$s$" exponents count down
Cubic Bézier Curves

Write in all of the numeric coefficients...
Express each term as powers of \( s \) and \( t \)

\[
P(t) = 1(s^1 t^0)A + 1(s^0 t^1)B \\
P(t) = 1(s^2 t^0)A + 2(s^1 t^1)B + 1(s^0 t^2)C \\
P(t) = 1(s^3 t^0)A + 3(s^2 t^1)B + 3(s^1 t^2)C + 1(s^0 t^3)D
\]

Note: “s” exponents count down
Note: “t” exponents count up
Cubic Bézier Curves

Write in all of the **numeric coefficients**...

Express each term as powers of \( s \) and \( t \)

\[
P(t) = 1(s^1t^0)A + 1(s^0t^1)B \\
P(t) = 1(s^2t^0)A + 2(s^1t^1)B + 1(s^0t^2)C \\
P(t) = 1(s^3t^0)A + 3(s^2t^1)B + 3(s^1t^2)C + 1(s^0t^3)D
\]

Note: numeric coefficients...
Cubic Bézier Curves

Write in all of the **numeric coefficients**...
Express each term as powers of $s$ and $t$

\[
\begin{aligned}
\mathbf{P}(t) &= 1(s^1t^0)\mathbf{A} + 1(s^0t^1)\mathbf{B} \\
\mathbf{P}(t) &= 1(s^2t^0)\mathbf{A} + 2(s^1t^1)\mathbf{B} + 1(s^0t^2)\mathbf{C} \\
\mathbf{P}(t) &= 1(s^3t^0)\mathbf{A} + 3(s^2t^1)\mathbf{B} + 3(s^1t^2)\mathbf{C} + 1(s^0t^3)\mathbf{D}
\end{aligned}
\]

Note: numeric coefficients...
    are from Pascal’s Triangle
Cubic Bézier Curves

Write in all of the **numeric coefficients**...
Express each term as powers of \( s \) and \( t \)

\[
P(t) = 1(s^1t^0)A + 1(s^0t^1)B \\
P(t) = 1(s^2t^0)A + 2(s^1t^1)B + 1(s^0t^2)C \\
P(t) = 1(s^3t^0)A + 3(s^2t^1)B + 3(s^1t^2)C + 1(s^0t^3)D
\]

You could continue this trend to easily deduce what the quartic (4\(^{th}\) order) and quintic (5\(^{th}\) order) equations would be...
Cubic Bézier Curves

What if $t = 0.5$? (halfway through the curve)

so then...  $s = 0.5$ also

$$P(t) = (s^3)A + 3(s^2t)B + 3(st^2)C + (t^3)D$$

becomes

$$P(t) = (.5^3)A + 3(.5^2*.5)B + 3(.5*.5^2)C + (.5^3)D$$

becomes

$$P(t) = (.125)A + 3(.125)B + 3(.125)C + (.125)D$$

becomes

$$P(t) = .125A + .375B + .375C + .125D$$
Cubic Bézier Curves

Cubic Bezier Curves can also be “S-shaped”, if their control points are “twisted” as pictured here.
Cubic Bézier Curves

Cubic Bezier Curves can also be “S-shaped”, if their control points are “twisted” as pictured here.
Cubic Bézier Curves

They can also loop back around in extreme cases.
Cubic Bézier Curves

They can also loop back around in extreme cases.
Cubic Bézier Curves

Seen in lots of places:

» Photoshop
» GIMP
» PostScript
» Flash
» AfterEffects
» 3DS Max
» Metafont

» Understable Disc Golf flight path
Quadratic vs. Quartic vs. Quintic

Just to clarify – since everyone always seems to get it wrong:

1. **Linear** Bezier curves have 2 points (0 guides), and are straight lines with order $t^1$
2. **Quadratic** Bezier curves have 3 points (1 guide), with order $t^2$
3. **Cubic** Bezier curves have 4 points (2 guides), with order $t^3$
4. **Quartic** Bezier curves have 5 points (3 guides), with order $t^4$
5. **Quintic** Bezier curves have 6 points (4 guides), with order $t^5$

Note: The fact that Quadratic means “squared” (and not “to the 4th”) is confusing for many folks – and rightfully so.

In geometry, **quadra**- usually means “four” (e.g. “quadrant”, “quadrilateral”). Similarly, **tri**- means “three (e.g. “triangle”).

However, in algebra – including polynomial equations (like these), **quadratic** means “square” or “squared” (as in $t^2$). Likewise, we use **cubic** to mean “cubed” (as in $t^3$). We use **quartic** to mean functions of degree four (as in $t^4$), **quintic** for five ($t^5$) and so on. I know, it sucks.
Splines
Okay, enough of Curves already.

So... what’s a Spline?
Splines

A **spline** is a chain of curves joined end-to-end.
Splines

A **spline** is a chain of curves joined end-to-end.
Splines

A *spline* is a chain of curves joined end-to-end.
Splines

A **spline** is a chain of curves joined end-to-end.
Splines

Curve end/start points (welds) are knots
Splines

Think of two different ts:

spline’s t: Zero at start of spline, keeps increasing until the end of the spline chain

local curve’s t: Resets to 0 at start of each curve (at each knot).
Splines

For a spline of 4 curve-pieces:

» Interpolate \texttt{spline\_t} from 0.0 to 4.0

» If \texttt{spline\_t} is \texttt{2.67}, then we are:
   In the third curve section (0,1,\texttt{2},3), and
   67\% through that section (\texttt{local\_t} = \texttt{.67})

» So... plug \texttt{local\_t} into \texttt{curve[2]}, i.e.
   \[ P(2.67) = \texttt{curve[2].EvaluateAt(.67)}; \]
Splines

Interpolating \textit{spline\_t} from 0.0 to 4.0...
Splines

Interpolating \texttt{spline}_t from 0.0 to 4.0...
Splines

Interpolating \texttt{spline\_t} from 0.0 to 4.0...
Splines

Interpolating *spline_t* from 0.0 to 4.0...
Splines

Interpolating `spline_t` from 0.0 to 4.0...
Splines

Interpolating \texttt{spline\_t} from 0.0 to 4.0...
Splines

Interpolating \texttt{spline\_t} from 0.0 to 4.0...
Splines

Interpolating \texttt{spline}_t from 0.0 to 4.0...
Splines

Interpolating `spline_t` from 0.0 to 4.0...
Quadratic Bezier Splines

This spline is a **quadratic Bezier spline**, since it is a spline made out of quadratic Bezier curves.
Continuity

Good continuity ($C^1$); connected and aligned

Poor continuity ($C^0$); connected but not aligned
Continuity

To ensure good continuity ($C^1$), make BC of first curve colinear (in line with) AB of second curve.
(derivative is continuous across entire spline)
Continuity

Excellent continuity ($C^2$) is when speed/density matches on either side of each knot.

(Second derivative is continuous across entire spline)
Cubic Bezier Splines

We can build a **cubic Bezier spline** instead by using cubic Bezier curves.
Cubic Bezier Splines

We can build a **cubic Bezier spline** instead by using cubic Bezier curves.
Cubic Bezier Splines

We can build a **cubic Bezier spline** instead by using cubic Bezier curves.
Cubic Hermite Splines

(pronounced “her-meet”)
Cubic Hermite Curves

A cubic Hermite curve is very similar to a cubic Bezier curve.
Cubic Hermite Curves

However, unlike a Bezier curve, we do not specify the B and C guide points. Instead, we give the velocity at point A (as U), and the velocity at D (as V) for each cubic Hermite curve segment.
Cubic Hermite Splines

To ensure connectedness ($C^0$), $D$ from curve #0 is typically assumed to be welded on top of $A$ from curve #1 (at a knot).
Cubic Hermite Splines

To ensure smoothness ($C^1$), velocity into $D$ ($V$) is typically assumed to match the velocity’s direction out of the next curve’s $A$ ($U$).
Cubic Hermite Splines

For best continuity ($C^2$), velocity into $D$ ($V$) matches direction and magnitude for the next curve’s $A$ ($U$).

i.e. We typically say there is a single velocity vector at each knot.
Cubic Hermite Splines

Hermite curves, and Hermite splines, are also parametric and work basically the same way as Bezier curves: plug in “t” and go!

The formula for cubic Hermite curve is:

\[ P(t) = s^2(1+2t)A + t^2(1+2s)D + s^2tU - st^2V \]

**Note, NOT:**

\[ P(t) = s^2(1+2t)A + t^2(1+2s)D + s^2tU + st^2V \]
Cubic Hermite Splines

Cubic Hermite and Bezier curves can be converted back and forth.

» To convert from cubic Hermite to Bezier:

\[
\begin{align*}
B &= A + \left(\frac{U}{3}\right) \\
C &= D - \left(\frac{V}{3}\right)
\end{align*}
\]

» To convert from cubic Bezier to Hermite:

\[
\begin{align*}
U &= 3(B - A) \\
V &= 3(D - C)
\end{align*}
\]
Cubic Hermite Spline

Cubic Hermite and Bezier curves can be converted back and forth.

» To convert from cubic Hermite to Bezier:

\[
B = A + \frac{U}{3} \\
C = D - \frac{V}{3}
\]

...and are therefore basically the exact same thing!

» To convert from cubic Bezier to Hermite:

\[
U = 3(B - A) \\
V = 3(D - C)
\]
Catmull-Rom Splines
Catmull-Rom Splines

A **Catmull-Rom spline** is just a cubic Hermite spline with special values chosen for the velocities at the start (\(U\)) and end (\(V\)) points of each section.

You can also think of Catmull-Rom not as a type of spline, but as a **technique for building cubic Hermite splines.**

Best application: curve-pathing through points
Catmull-Rom Splines

Start with a series of points (spline start, spline end, and interior knots)
Catmull-Rom Splines

1. Assume $\mathbf{u}$ and $\mathbf{v}$ velocities are zero at start and end of spline (points 0 and 6 here).
2. Compute a vector from point 0 to point 2.

\[(\text{Vec}_{0\text{ to }2} = P_2 - P_0)\]
Catmull-Rom Splines

That will be our tangent for point 1.
Catmull-Rom Splines

3. Set the velocity for point 1 to be $\frac{1}{2}$ of that.
Catmull-Rom Splines

Now we have set positions 0 and 1, and velocities at points 0 and 1. Hermite curve!
Catmull-Rom Splines

4. Compute a vector from point 1 to point 3.
   \( \text{Vec}_{1 \to 3} = P_3 - P_1 \)
Catmull-Rom Splines

That will be our tangent for point 2.
Catmull-Rom Splines

5. Set the velocity for point 2 to be $\frac{1}{2}$ of that.
Catmull-Rom Splines

Now we have set positions and velocities for points 0, 1, and 2. We have a Hermite spline!
Catmull-Rom Splines

Repeat the process to compute velocity at point 3.
Catmull-Rom Splines

Repeat the process to compute velocity at point 3.
Catmull-Rom Splines

And at point 4.
Catmull-Rom Splines

And at point 4.
Catmull-Rom Splines

Compute velocity for point 5.
Catmull-Rom Splines

Compute velocity for point 5.
Catmull-Rom Splines

We already set the velocity for point 6 to be zero, so we can close out the spline.
Catmull-Rom Splines

And voila! A Catmull-Rom (Hermite) spline.
Catmull-Rom Splines

Here’s the math for a Catmull-Rom Spline:

» Place knots where you want them (A, D, etc.)
» If we call the position at the Nth point $P_N$
» and the velocity at the Nth point $V_N$ then:
» $V_N = (P_{N+1} - P_{N-1}) / 2$

i.e. Velocity at point P is half of [the vector pointing from the previous point to the next point].
Cardinal Splines
Cardinal Splines

Same as a Catmull-Rom spline, but with an extra parameter: **Tension**.

Tension can be set from 0 to 1.

A tension of 0 is just a Catmull-Rom spline.

Increasing tension causes the velocities at all points in the spline to be scaled down.
Cardinal Splines

So here is a Cardinal spline with tension=0
(same as a Catmull-Rom spline)
Cardinal Splines

So here is a Cardinal spline with tension=.5
(velocities at points are ½ of the Catmull-Rom)
Cardinal Splines

And here is a Cardinal spline with tension=1
(velocities at all points are zero)
Cardinal Splines

Here’s the math for a Cardinal Spline:

» Place knots where you want them (A, D, etc.)
» If we call the position at the Nth point \( P_N \)
» and the velocity at the Nth point \( V_N \) then:
» \( V_N = (1 – \text{tension})(P_{N+1} - P_{N-1}) / 2 \)

i.e. Velocity at point P is **some fraction of** half of [the vector pointing from the previous point to the next point].

i.e. Same as Catmull-Rom, but \( V_N \) gets scaled down because of the \((1 – \text{tension})\) multiply.
Other Spline Types
Kochanek–Bartels (KB) Splines

Same as a Cardinal spline (includes Tension), but with two extra tweaks (usually set on the entire spline).

**Bias** (from -1 to +1):
- A zero bias leaves the velocity vector alone
- A positive bias rotates the velocity vector to be more aligned with the point BEFORE this point
- A negative bias rotates the velocity vector to be more aligned with the point AFTER this point

**Continuity** (from -1 to +1):
- A zero continuity leaves the velocity vector alone
- A positive continuity “poofs out” the corners
- A negative continuity “sucks in / squares off” corners
B-Splines

» Stands for “basis spline”.
» Just a generalization of Bezier splines.
» The basic idea:
   
   At any given time, P(t) is a weighted-average blend of 2, 3, 4, or more points in its “neighborhood”.

» Equations are usually given in terms of the blend weights for each of the nearby points based on where \( t \) is at.
Curved Surfaces

You can criss-cross splines and form 2d curved surfaces.
Parametric Manipulations

- Do NOT mess with the interpolation itself (e.g. color, position, AI disposition, etc.)
- Instead, just mess with the parameter!
- We’re going to play with ‘t’, in order to make P(t) more interesting (for any P)!
Parametric Opportunities

- Anytime you have a single float to change
- Anytime you can express something in terms of a single float
- Pretty much whenever you use “time”
Takeaway: The Big Idea

- You can make ANY parametric equation $P(t)$ much more cool and interesting...

...without modifying $P(t)$ itself...

...without knowing anything about $P(t)$!
Fast and Funky 1D Nonlinear Transformations

We’re gonna manipulate “t”!
The Two Most Important Number Ranges

The two most important number ranges of all time are:

\([0,1]\) and \([-1,1]\)
The Two Most Important Number Ranges

[0,1] is especially useful for fractions:

- % shadow
- % luminance
- % falloff
- % complete
- % damage
- % experience
- % cost
- % penalty

- % fog
- % AI aggression
- % chance to hit
- % chance to drop loot
- % time complete
- AI response curves
- Fuzzy logic
- Most anything parametric
The Two Most Important Number Ranges

[-1,1] is especially useful for deviations:

- noise
- perturbation
- terrain & map generation
- variation
- distribution
- sinusoidal
- AI response curves
- many others...
Fast and Funky 1D Nonlinear Transformations

Normalized

We’re gonna focus on the [0,1] range
Properties of **Normalized** Utility Functions

- **Linear**
- **x-squared**
Properties of **Normalized** Utility Functions

- **Linear**
- **x-squared**
Properties of **Normalized** Utility Functions

- **Linear**
- **x-squared**
Properties of Normalized Utility Functions

Linear

x-squared
Examples

- Position-over-time
- Scale-over-time
- Alpha-over-time
- (Color-over-time)
- (Strength-over-time)
- (Aggression-over-time)
- (anything-over-time...
Demo 1,2

\textbf{x-squared} is \textbf{far} more interesting than \textbf{linear}...

\[ P(t) = t^2 \quad \text{is better than} \quad P(t) = t \]

...but that’s just the tip of the iceberg...

\textit{Note: in \([0,1]\), x-squared lowers a value!}
Also Known As...

Fast and Funky 1D Nonlinear Transformations

Normalized

Also sometimes called “easing functions”, or “filter functions” (as in “run a filter on it”), or “lerping functions” (as in “linear interpolate”), or “tweening functions” (as in “in-betweening”).
Also Known As…

Many of these terms strongly imply an application.

E.g. “tweening” is often used in the context of animation, e.g. creating frames “in-between” animation key frames.

In truth, normalized utility functions are like a universal adaptor – they can be applied anywhere!

*Going to use “easing” a lot from here on out, ‘cuz it’s short. (And “tweening” sounds lame.)*
Range-Mapping

Can be applied during middle of range-mapping:

```cpp
out RangeMap( in, inStart, inEnd, outStart, outEnd )
{
    out = in - inStart;  // Puts in [0, inEnd - inStart]
    out /= (inEnd - inStart);  // Puts in [0,1]
    out = ApplySomeEasingFunction( out );  // in [0,1]
    out *= (outEnd - outStart);  // Puts in [0, outRange]
    return out + outStart;  // Puts in [outStart, outEnd]
}
```
Building our function library

Let’s start with the simplest category of easing function:

Function: $\text{SmoothStart}$

$\text{SmoothStart}_2(t) = t^2$
Building our function library

Let’s start with the simplest category of easing function:

Function: **SmoothStart**

\[
\text{SmoothStart}_2(t) = t^2 \\
\text{SmoothStart}_3(t) = t^3
\]
Building our function library

Let’s start with the simplest category of easing function:

**Function: SmoothStart**

\[
\text{SmoothStart}_2(t) = t^2 \\
\text{SmoothStart}_3(t) = t^3 \\
\text{SmoothStart}_4(t) = t^4 \\
\text{SmoothStart}_5(t) = t^5 \\
\text{SmoothStart}_6(t) = t^6
\]
Building our function library

Of course, in code we would actually write \( t^2 \), not \( \text{pow}() \)

**Function:** SmoothStart

\[
\begin{align*}
\text{SmoothStart}_2(t) &= t^2 \\
\text{SmoothStart}_3(t) &= t^3 \\
\text{SmoothStart}_4(t) &= t^4 \\
\text{SmoothStart}_5(t) &= \ldots \\
\text{SmoothStart}_6(t) &= \ldots 
\end{align*}
\]
Demo 4

Notes:

- Each function starts at the same time \((t=0)\)
- Each function ends at the same time \((t=1)\)
- The higher the power, the more exaggerated
- Also called “ease in”, though this is confusing...
Building our function library

If SmoothStart is \( x^2 \), what would SmoothStop be?

Function: **SmoothStop**

\[
\text{SmoothStop}^2(t) = ???
\]
Building our function library

If SmoothStart is $x^2$, what would SmoothStop be?

Function: \textbf{SmoothStop}

\textbf{SmoothStop2}(t) = \sqrt{t}?
Building our function library

If SmoothStart is $x^2$, what would SmoothStop be?

Function: **SmoothStop**

$\text{SmoothStop}^2(t) = \sqrt{t}$?

$\sqrt{t}$ doesn’t stop smooth!
And it’s not particularly **fast**.
Building our function library

If SmoothStart is $x^2$, what would SmoothStop be?

**Function:** SmoothStop

$\text{SmoothStop}_2(t) = ???$

Let’s evolve it ourselves!
Evolving Easing Functions

There are lots of different things we can do to a number in \([0,1]\).

So far, the most interesting one has been to \textbf{Square} it.

So what else can we do to it?

“Square” = \(x^2\)
Evolving Easing Functions

There are lots of different things we can do to a number in [0,1].

So far, the most interesting one has been to **Square** it.

So what else can we do to it?

How about flipping it?

“Flip” = 1-x
Evolving Easing Functions

Now what if we square \textbf{that}?

\[ \text{"Flip"} = 1-x \]
Evolving Easing Functions

Now what if we square that?
...and then flip it again?

\[ \text{Square}(\text{Flip}(x)) \]
Evolving Easing Functions

Now what if we square that? ...and then flip it again?

**BINGO!** It’s **SmoothStop2**!

We did: Flip, Square, Flip.

Flip( Square( Flip (x) ) )
Evolving Easing Functions

Now what if we square that?
...and then flip it again?

**BINGO!** It’s **SmoothStop2**!

We did: Flip, Square, Flip.
So, math-wise, that’s:
Evolving Easing Functions

Now what if we square that? ...and then flip it again?

**BINGO!** It’s SmoothStop2!

We did: Flip, Square, Flip.
So, math-wise, that’s:
**Flip** \(1-x\)
Evolving Easing Functions

Now what if we square **that**?  
...and then flip it again?

**BINGO!** It’s **SmoothStop2**!

We did: Flip, Square, Flip.  
So, math-wise, that’s:

- **Flip**  \( 1-x \)
- **Square**  \( (1-x)^2 \)
Evolving Easing Functions

Now what if we square that? ...and then flip it again?

**BINGO!** It’s **SmoothStop2**!

We did: Flip, Square, Flip. So, math-wise, that’s:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flip</td>
<td>1-x</td>
</tr>
<tr>
<td>Square</td>
<td>(1-x)^2</td>
</tr>
<tr>
<td>Flip</td>
<td>1 – (1-x)^2</td>
</tr>
</tbody>
</table>

1 – (1-x)^2