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 *that*?

“Flip” = 1 – \( x \)
Evolving Easing Functions

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

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.
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 \textbf{that}? …and then flip it again?

\textbf{BINGO!} It’s \textbf{SmoothStop2}!

We did: Flip, Square, Flip. So, math-wise, that’s:
\begin{itemize}
  \item \textbf{Flip} $1-x$
\end{itemize}
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** \((1-x)^2\)

\((1-x)^2\)
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:

- **Flip** 1-x
- **Square** (1-x)^2
- **Flip** 1 - (1-x)^2

\[1 - (1-x)^2\]
Building our function library

So:

Function: **SmoothStop**

\[
\text{SmoothStop2}(t) = 1 - (1-t)^2
\]
Building our function library

So:

**Function:** SmoothStop

\[
\text{SmoothStop}_2(t) = 1 - (1-t)^2
\]

\[
\text{SmoothStop}_3(t) = 1 - (1-t)^3
\]
Building our function library

So:

Function: $\text{SmoothStop}$

$\text{SmoothStop}_2(t) = 1 - (1-t)^2$
$\text{SmoothStop}_3(t) = 1 - (1-t)^3$
$\text{SmoothStop}_4(t) = 1 - (1-t)^4$
$\text{SmoothStop}_5(t) = 1 - (1-t)^5$
$\text{SmoothStop}_6(t) = 1 - (1-t)^6$

$1 - (1-x)^4$
Building our function library

A mathematician would likely be inclined to multiply these out into polynomial form:

\[ \text{SmoothStop}_3(x) = 3x - 3x^2 + x^3 \]

\[ \text{SmoothStop}_4(x) = 4x - 6x^2 + 4x^3 - x^4 \]

...
Building our function library

A mathematician would likely be inclined to multiply these out into polynomial form:

\[
\text{SmoothStop}_3(x) = 3x - 3x^2 + x^3
\]

\[
\text{SmoothStop}_4(x) = 4x - 6x^2 + 4x^3 - x^4
\]

... But that’d be dumb!

\[1-(1-x)^N\] is clearer,

and, in most cases, faster!

\[(2+ \text{ times faster for SS5})\]

And it preserves evolutionary history in its natural form.
Building our function library

So far we have functions:

\[ \text{SmoothStart}_N(x) = x^N \]
\[ \text{SmoothStop}_N(x) = 1 - (1-x)^N \]

We also have some techniques:

**Square:** \( x^2 \)
**Flip:** \( 1 - x \)

Let’s build up a library of these.

Let’s get more of these also.

**Exponentiate:** \( x^N \)
Evolving Easing Functions

\[
\text{Mix}(a, b, \text{weightB}, t) = a + \text{weightB}(b - a)
\]

or

\[
(1-\text{weightB})a + (\text{weightB})b
\]

For example:

\[
\text{Mix}(\text{SmoothStart2}, \text{SmoothStop2}, \text{blend}, t)
\]

BTW: \[
\text{Mix}(\text{SmoothStart2}, \text{SmoothStop2}, 0.5, t) \text{ is Linear!}
\]
Evolving Easing Functions

t = pow( t, 2.2 );
t = pow( t, 2.6 );
t = pow( t, 9.4 );

We know pow() is at least as expensive as sqrt()...
...since pow() can ALSO do sqrt() by itself:

    float root = pow( number, 0.5 );

In fact, on some machines it can be 10x as expensive!
Evolving Easing Functions

We know:

\[
\text{SmoothStart}_2 = x^2 \\
\text{SmoothStart}_3 = x^3 \\
\text{SmoothStart}_N = x^N
\]

So what about \textit{SmoothStart}_{2.2}?

\[
\text{SmoothStart}_{2.2} = x^{2.2}
\]

...except, wait, that requires \texttt{pow()}, right?
Evolving Easing Functions

We can instead do:

\[ \text{SmoothStart2.2} = \text{Mix}( \text{SmoothStart2}, \text{SmoothStart3}, 0.2 ); \]

and

\[ \text{SmoothStart2.6} = \text{Mix}( \text{SmoothStart2}, \text{SmoothStart3}, 0.6 ); \]

99% accurate in \([0,1]\)...

...and at least 10x faster!
Evolving Easing Functions

Or you could simply write:

\[
\text{FakePow}( x, 2.2 ) = (.8 \times x^2) + (.2 \times x^3)
\]

and

\[
\text{FakePow}( x, 2.6 ) = (.4 \times x^2) + (.6 \times x^3)
\]

99% accurate in [0,1]...

...and at least 10x faster!
Evolving Easing Functions

**Crossfade**: like Mix, but use \( t \) itself as the mix weight!

\[
\text{Crossfade}(a, b, t) = a + t(b - a)
\]

or

\[
(1-t)a + (t)b
\]

For example: **Demo 6**

\[
\text{Crossfade}(\text{SmoothStart2}, \text{SmoothStop2}, t) = \text{SmoothStep3}!
\]
Evolving Easing Functions

**Scale**($\text{Function}, t$) = $t \times \text{Function}(t)$

**ReverseScale**($\text{Function}, t$) = $(1-t) \times \text{Function}(t)$

$\text{Arch2}(t) = \text{Scale}(\text{Flip}(t)) = t \times (1-t)$

$\text{SmoothStartArch3}(t) = \text{Scale}(\text{Arch2}, t) = x^2(1-x)$

$\text{SmoothStopArch3}(t) = \text{ReverseScale}(\text{Arch2}, t) = x(1-x)^2$

$\text{SmoothStepArch4}(t) = \text{ReverseScale}(\text{Scale}(\text{Arch2}, t), t)$

$\text{BellCurve6}(t) = \text{SmoothStop3}(t) \times \text{SmoothStart3}(t)$

$???(t) = \text{SmoothStop3}(\text{SmoothStart3}(t)), \text{etc}...$
Evolving Easing Functions

// "Bounces" off the bottom of the [0,1] range since any negative values are now positive.
inline float BounceClampBottom( float t )
{
    return fabs( t );
}

// "Bounces" off the top of the [0,1] range since any values over 1 become inverted below 1.
inline float BounceClampTop( float t )
{
    return 1.f - fabs( 1.f - t );
}

inline float BounceClampBottomTop( float t )
{
    return BounceClampTop( BounceClampBottom( t ) );
}
Last but not least...

Use 1D Cubic (Quartic, Quintic...) Bezier curves from 0 to 1!

Can pre-optimize NormalizedBezier functions, assuming that first coefficient is 0, and last is 1:

```c
// Cubic (3rd) Bezier through A,B,C,D where A (start) and D (end) are assumed to be 0 and 1
inline float NormalizedBezier3( float B, float C, float t )
{
    float s = 1.f - t;
    float t2 = t*t;
    float s2 = s*s;
    float t3 = t2*t;
    return (3.f*B*s2*t) + (3.f*C*s*t2) + (t3);
}
```
Last but not least...

// 7th order Bezier through A,B,C,D,E,F,G,H where A (start) and H (end) are assumed to be 0 and 1
inline float NormalizedBezier7( float B, float C, float D, float E, float F, float G, float t )
{
    float s = 1.f - t;
    float t2 = t*t;
    float s2 = s*s;
    float t3 = t2*t;
    float s3 = s2*s;
    float t4 = t2*t2;
    float s4 = s2*s2;
    float t5 = t3*t2;
    float s5 = s3*s2;
    float t6 = t3*t3;
    float s6 = s3*s3;
    float t7 = t3*t2*t2;
    return (7.f*B*s6*t) + (21.f*C*s5*t2) + (35.f*D*s4*t3) + 
            (35.f*E*s3*t4) + (21.f*F*s2*t5) + (7.f*G*s*t6) + (t7);
}
Game Architecture

2/9/16: Quaternions
You will eventually regret any use of Euler angles.
Figure 2.1-24. IMU Gimbal Assembly
After Stafford's camera failed, he and Cernan had little to do except look at the scenery until time to dump the descent stage. Stafford had the vehicle in the right attitude 10 minutes early. Cernan asked, "You ready?" Then he suddenly exclaimed, "Son of a bitch!" Snoopy seemed to be throwing a fit, lurching wildly about. He later said it was like flying an Immelmann turn in an aircraft, a combination of pitch and yaw. Stafford yelled that they were in gimbal lock - that the engine had swiveled over to a stop and stuck - and they almost were. He called out for Cernan to thrust forward. Stafford then hit the switch to get rid of the descent stage and realized they were 30 degrees off from their previous attitude. The lunar module continued its crazy gyrations across the lunar sky, and a warning light indicated that the inertial measuring unit really was about to reach its limits and go into gimbal lock. Stafford then took over in manual control, made a big pitch maneuver, and started working the attitude control switches. Snoopy finally calmed down.
Spacecraft structure

FIGURE 1
Linear Interpolation

\[ L = LERP(A, B, \beta) = (1 - \beta)A + \beta B \]

\[ = [(1 - \beta)A_x + \beta B_x, (1 - \beta)A_y + \beta B_y, (1 - \beta)A_z + \beta B_z] \]
Interpolating Rotation

• Not as simple, but more important
• E.g. camera control
  • Store orientations for camera, interpolate
• E.g. character animation
  • Body location stored as point
• Joints stored as rotations
Orientation vs. Rotation

- Orientation is described relative to some reference alignment.
- A rotation changes object from one orientation to another.
- Can represent orientation as a rotation from the reference alignment.
Ideal Orientation Format

- Represent 3 degrees of freedom with minimum number of values
- Allow concatenations of rotations
- Math should be simple and efficient
  - concatenation
  - interpolation
  - rotation
Matrices as Orientation

- Matrices just fine, right?
- Well…
  - 9 values to interpolate
  - don’t interpolate well
Interpolating Matrices

• Say we interpolate halfway between each element

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\quad \frac{1}{2}
\begin{bmatrix}
0 & -1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\]

• Result isn’t a rotation matrix!

• Need Gram-Schmidt orthonormalization
Why Not Euler Angles?

• Three angles
  • Heading, pitch, roll
• However
  • Dependent on coordinate system
  • No easy concatenation of rotations
  • Still has interpolation problems
  • Can lead to gimbal lock
Quat ernion

• “Pre-cooked” axis-angle format
• 4 data members
• Well-formed
• Reasonably-ish simple math
  • concatenation
  • interpolation
  • rotation
What transformation \( x \), when applied twice, turns 1 to 9?

\[ x^2 = 9 \]

\[ 1 \cdot x \cdot x = 9 \]
What transformation $x$, when applied twice, turns 1 to -1?
\[-1n = \text{rotate}(\pi)\]

\[\sqrt{-1}\]?

\[\sqrt{-1}n = \text{rotate}\left(\frac{\pi}{2}\right)\]
$-1^n = \text{rotate} (\pi)$

$\sqrt{-1}$?

$\sqrt{-1}^n = \text{rotate} \left( \frac{\pi}{2} \right)$
• Numbers can be multi-dimensional! \( i \) is a new “imaginary” dimension to measure a number.
• \( i \) (or \(-i\)) is what numbers “become” when rotated
• Multiplying \( i \) is a rotation by 90 degrees counter-clockwise
• Multiplying by \(-i\) is a rotation of 90 degrees clockwise
• Two rotations in either direction is \(-1\): it brings us back into the familiar dimensions of positive and negative numbers
$i^2 = -1$

$i^3 = -i$

$i^4 = 1$
$r = 1$

$\sin \theta$

$\cos \theta$
The complex number $3 + 4i$ is located at a $45^\circ$ angle from the origin.
\[(3 + 4i) \cdot (1 + i)\]

\[3 + 3i + 4i + 4i^2\]

\[3 + 7i + 4(-1)\]

\[-1 + 7i\]
\[ i^2 = -1 \]

\[ i^2 = j^2 = k^2 = ijk = -1 \]

\[
\begin{align*}
ij &= k \\
ji &= -k \\
jk &= i \\
kj &= -i \\
ki &= j \\
iki &= -j
\end{align*}
\]
Unit-Length Restriction

\[ q \cdot q = (q_1)^2 + (q_2)^2 + (q_3)^2 + (q_0)^2 = q \cdot q + (q_0)^2 = 1 \]
What is a Quaternion?

- Vector space over the real numbers of dimension 4
- An extension to complex numbers
- 4 components, one is a ‘real’ scalar number, and the other 3 form a vector in ‘imaginary’ ijk space

\[ q = (q_1, q_2, q_3, q_0) = (q, q_0) = iq_1 + jq_2 + kq_3 + q_0 \]
Spheres

- Think of a person standing on the surface of a big sphere (like a planet)
- From the person’s point of view, they can move in along two orthogonal axes (front/back) and (left/right)
- There is no perception of any fixed poles or longitude/latitude, because no matter which direction they face, they always have two orthogonal ways to go
- From their point of view, they might as well be moving on an infinite 2D plane, however if they go too far in one direction, they will come back to where they started
Hyperspheres

• Now extend this concept to moving in the hypersphere of unit quaternions

• The person now has three orthogonal directions to go

• No matter how they are oriented in this space, they can always go some combination of forward/backward, left/right and up/down

• If they go too far in any one direction, they will come back to where they started (whoa)
Hyperspheres

• Now consider that a person’s location on this hypersphere represents an orientation

• Any incremental movement along one of the orthogonal axes in curved space corresponds to an incremental rotation along an axis in real space (distances along the hypersphere correspond to angles in 3D space)

• Moving in some arbitrary direction corresponds to rotating around some arbitrary axis

• If you move too far in one direction, you come back to where you started (corresponding to rotating 360 degrees around any one axis)
Hyperspheres

• A distance of x along the surface of the hypersphere corresponds to a rotation of angle 2x radians

• This means that moving along a 90 degree arc on the hypersphere corresponds to rotating an object by 180 degrees

• Traveling 180 degrees corresponds to a 360 degree rotation, thus getting you back to where you started

• This implies that q and -q correspond to the same orientation
Hyperspheres

- Consider what would happen if this was not the case, and if 180 degrees along the hypersphere corresponded to a 180 degree rotation.

- This would mean that there is exactly one orientation that is 180 opposite to a reference orientation.

- In reality, there is a continuum of possible orientations that are 180 away from a reference.

- They can be found on the equator relative to any point on the hypersphere.
Hyperspheres

• Also consider what happens if you rotate an object 180 around x, then 180 around y, and then 180 around z

• You end up back where you started

• This corresponds to traveling along a triangle on the hypersphere where each edge is a 90 degree arc, orthogonal to each other edge
“Mr. Osborne, may I be excused? My brain is full.”
Quaternions as Rotations

• A quaternion can represent a rotation by an angle around a unit axis $\mathbf{a}$

\[ q = (a_x \sin \frac{\theta}{2}, a_y \sin \frac{\theta}{2}, a_z \sin \frac{\theta}{2}, \cos \frac{\theta}{2}) \]
# Quaternion to Matrix

\[
\begin{array}{ccc}
1 - 2q_2^2 - 2q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\
2q_1q_2 + 2q_0q_3 & 1 - 2q_1^2 - 2q_3^2 & 2q_2q_3 - 2q_0q_1 \\
2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & 1 - 2q_1^2 - 2q_2^2 \\
\end{array}
\]
Matrix to Quaternion

• That’s harder
```python
def matrix2Quat(self):
    """returns quaternion representation of current matrix"""
    a = [0,0,0,0]
    quat = Quaternion()
    nxt = [1,2,0]
    trace = self[0][0] + self[1][1] + self[2][2]
    # check diagonal
    if trace > 0:
        s = math.sqrt(trace + 1)
        quat.w = s / 2
        s = .5 / s
        quat.x = (self[2][1] - self[1][2]) * s
        quat.y = (self[0][2] - self[2][0]) * s
        quat.z = (self[1][0] - self[0][1]) * s
        return quat
    else:
        i = 0
        if self[1][1] > self[0][0]:
            i = 1
        if self[2][2] > self[i][i]:
            i = 2
        j = nxt[i]
        k = nxt[j]
        s = math.sqrt(((self[i][i] - (self[j][j] + self[k][k])) + 1)
        q[i] = s * .5
        if s != 1:
            s = .5 / s
            quat.w = (self[j][k] - self[k][j]) * s
            q[j] = (self[i][j] + self[j][i]) * s
            q[k] = (self[i][k] + self[k][i]) * s
            quat.x = q[0]
            quat.y = q[1]
            quat.z = q[2]
        return quat
```
Quaternion Multiplication

\[ pq = (p_1, p_2, p_3, p_0)(q_1, q_2, q_3, q_0) = (p, p_0)(q, q_0) \]

\[ = (p_0q + q_0p + (p \times q), p_0q_0 - p \cdot q) \]

\[ = \begin{vmatrix}
    p_1q_0 + p_0q_1 + p_2q_3 - p_3q_2 \\
    p_2q_0 + p_0q_2 + p_3q_1 - p_1q_3 \\
    p_3q_0 + p_0q_3 + p_1q_2 - p_2q_1 \\
    p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3
\end{vmatrix} \]
Quaternion Multiplication

• Note that two unit quaternions multiplied together will result in another unit quaternion

• This corresponds to the same property of complex numbers

• Remember that multiplication by complex numbers can be thought of as a rotation in the complex plane

• Quaternions extend the planar rotations of complex numbers to 3D rotations in space
Rotating a Vector

\[ v' = qvq^* \]

where \( q^* \) is the \textit{conjugate} of \( q \)

\[ q^* = (-q_1, -q_2, -q_3, q_0) \]

we’re rotating the vector out into the fourth dimension (woooo….), then rotating back into the third dimension

Remember, \( w \) is 0 for a vector in 3-space
Alternatively

\[ v' = v + 2q \times (q \times v + q_0 v) \]
SQT Transformation

• A quaternion only represents a rotation
• When combined with a translation vector and a scale factor, we have a viable alternative to the 4x4 matrix
• \( SQT = [s \, q \, t] \)
• 8 values (for uniform scaling) instead of 16 for the matrix
• Easily interpolated – use LERP for \( t \) and \( s \), and SLERP for \( q \)
Spherical Linear Interpolation

\[ \text{SLERP}(p, q, \beta) = w_p p + w_q q \]

\[ w_p = \frac{\sin((1 - \beta)\theta)}{\sin(\theta)} \]

\[ w_q = \frac{\sin(\beta\theta)}{\sin(\theta)} \]

\[ \theta = \cos^{-1}(p \cdot q) \]
Summary

• Quaternions are 4D vectors that can represent 3D rigid body orientations
• We choose to force them to be unit length
• Key animation functions:
  • Quaternion-to-matrix / matrix-to-quaternion
  • Quaternion multiplication: faster than matrix multiplication
  • SLERP: interpolate between arbitrary orientations
Homework 3

• SLERP that cube!

• Extend Homework 2 to allow for SQT transformations

• Write a Quaternion object, with appropriate methods to go back and forth between matrix and quaternion representations

• SLERP method that can take either two matrices or two quaternions, and a $\beta$ value.