keyframe animation

- an animator describes snapshots of a 3D computer graphics animation at a set of discrete times.
  - called key frames
- Each keyframe is defined by some set of modeling parameters.
  - in our case this is a bunch of RBTs
  - the translations are 3 real scalars (start with these)
  - the rotations are quaternions (get back to in a bit).
- To create a smooth animation the computer's job is to smoothly "fill in" the parameter values over a continuous range of times.

interpolating

- let one of these animated parameters be called c, and each of our discrete snapshots is called c_i where i is some range of ints
- our job is to go from the c_i to a continuous function of time, $c(t)$
  - We will typically want the function $c(t)$ to be sufficiently smooth, so that the animation does not appear too jerky.
- see Figure: we show a function $c(t)$, with $t \in [0..8]$ that interpolates the discrete values associated with the integers c_i with $i \in -1..9$
  - the need for the extra non-interpolated values at -1 and 9 will be made clear later
- until now, we have used piecewise linear interpolation, which is not smooth.

splines

- Our spline is made up of individual pieces, where each piece is some low order polynomial function
- These polynomial pieces will be selected so that they "stitch up" smoothly
- easy to represent, evaluate and control.
  - spline behavior much easier to predict than, say, a a single high-order polynomial function.
- also useful for curves in the plane (fonts) and space, and the basis for theory of smooth surfaces in space.

Cubic Bezier Functions

- We start by just looking at how to represent a cubic polynomial function $c(t)$ with $t \in [0..1]$
  - see Figure
- we will talk about the Bezier representation
  - the parameters have a direct geometric interpretation
  - evaluation reduces to repeated linear interpolation.

bezier control polygon

- we will specify a cubic function using four "control values" c_0, d_0, e_0 and c_1
  - see fig: visualized as points in the 2D (t, c) plane
  - with coordinates $[0, c_0]^t$, $[1/3, d_0]^t$, $[2/3, e_0]^t$ and $[1, c_1]^t$.
  - We have also drawn in light blue a poly-line connecting these points; this is called the *control polygon*.

bezier evaluation

- To evaluate the function $c(t)$ at any value of t, we perform the following sequence of linear interpolations

$$\begin{aligned} f &= (1-t)c_0 + td_0 && (1) \\ g &= (1-t)d_0 + te_0 && (2) \\ h &= (1-t)e_0 + tc_1 && (3) \\ m &= (1-t)f + tg && (4) \\ n &= (1-t)g + th && (5) \\ c(t) &= (1-t)m + tn && (6) \end{aligned}$$

- see figure for the steps of this computation for $t = .3$

Properties

- By unwrapping the evaluation steps above, we can verify that $c(t)$ has the form

$$c(t) = c_0(1-t)^3 + 3d_0t(1-t)^2 + 3e_0t^2(1-t) + c_1t^3$$

- it is a cubic function in the variable t.
- the c_i are interpolated: $c(0) = c_0$ and $c(1) = c_1$.
- Taking derivatives, we see that $c'(0) = 3(d_0 - c_0)$ and $c'(1) = 3(c_1 - e_0)$.
  - In Figure, we see indeed that the slope of $c(t)$ matches the slope of the control polygon at 0 and 1.
- if we set $c_0 = d_0 = e_0 = c_1 = 1$, then $c(t) = 1$ for all t.
  - This property is called *partition of unity*
  - means that adding adding a constant value to all control values results in simply adding this constant to $c(t)$.

Translated domain

- If we want a cubic function to interpolate values c_i and c_{i+1} at $t = i$ and $t = i + 1$, respectively, and calling our two other control points d_i and e_i,
- we just have to "translate" the evaluation algorithm to get

$$\begin{aligned} f &= (1-t+i)c_i + (t-i)d_i && (7) \\ g &= (1-t+i)d_i + (t-i)e_i && (8) \\ h &= (1-t+i)e_i + (t-i)c_{i+1} && (9) \\ m &= (1-t+i)f + (t-i)g && (10) \\ n &= (1-t+i)g + (t-i)h && (11) \\ c(t) &= (1-t+i)m + (t-i)n && (12) \end{aligned}$$

- within the range $i..i+1$ this looks just like the untranslated algorithm operating on the fractional component.

Catmull-Rom Splines

- Let us return now to our original problem of interpolating a set of values c_i for $i \in -1..n+1$.
- the "catmull rom spline" defines a function $c(t)$ for values of $t \in [0..n]$.
- The function is defined by n cubic functions, each supported over a unit interval $t \in [i..i+1]$.
- The pieces are chosen to interpolate the c_i values, and to agree on their first derivatives.

CRS construction

- Each cube function is described in its Bezier representation, using four control values: c_i, d_i, e_i, and c_{i+1}.
- From our input data, we already have the c_i values.

- To set the d_i and e_i values we impose the constraint $c'(t)|_i = \frac{1}{2}(c_{i+1} - c_{i-1})$.
  - ie. we look forward and backwards one sample to determine its slope at $t = i$;
  - this is why we need the extra c-values
- this also can be stated as $c'(t)|_{i+1} = \frac{1}{2}(c_{i+2} - c_i)$.
- Since $c'(i) = 3(d_i - c_i)$and $c'(i+1) = 3(c_{i+1} - e_i)$, in the Bezier representation, this tells us that we need to set

$$\begin{aligned} d_i &= \frac{1}{6}(c_{i+1} - c_{i-1}) + c_i && (13) \\ e_i &= \frac{-1}{6}(c_{i+2} - c_i) + c_{i+1} && (14) \end{aligned}$$

bf translational splining

- the translational part of an RBT is represented by three scalars (t_x, t_y, t_z).
- so we can just apply the splining algorithm three times over the three coordinates.

Quaternion Splining

- If we want to interpolate a set of quaternions we need to hack by association
- Bezier evaluation steps of the form

$$r = (1-t)p + tq$$

  become

$$r = \mathtt{slerp}(\mathtt{p}, \mathtt{q}, \mathtt{t})$$

- to do the catmull rom construction, we substitute appropriate quaternion operations for the scalar operations
  - scalar addition becomes quaternion multiplication, scalar negation becomes quaternion inversion, and scalar multiplication becomes quaternion power.
- so the d_i and e_i quaternion values are defined as

$$\begin{aligned} d_i &= ((c_{i+1}c_{i-1}^{-1})^{\frac{1}{6}})c_i \\ e_i &= ((c_{i+2}c_i^{-1})^{\frac{-1}{6}})c_{i+1} \end{aligned}$$

- in order to interpolate "the short way", we do conditional negation on $c_{i+1}c_{i-1}^{-1}$ before applying the power operator.

curves

- we can cleanly apply the scalar theory twice (three times) to describe curves in the plane (or space).
- the spline curve is controlled by a set of *control points* $\tilde{c}_i$ in 2D or 3D.
- Applying the spline construction independently to the x y and z coordinates, one gets a point-valued spline function $\tilde{c}(t)$;
- think of this as a point flying through space over time, tracing out the spline curve, γ.
- this can be further developed into a theory for surfaces.