Notes

- Assignment 0 is being marked
- Textbook reference for arc-length parameterization:
  - Section 3.2

Review

- **Cubic Hermite Spline**: the standard tool for animation. C¹, interpolating, local control. Smoothness easy to break if needed: flexible!
- **Catmull-Rom**: a good default choice for the slopes, based on finite difference formulas
- **Cubic B-Spline**: C², approximating, local control. Not so useful for animating in time, very useful for defining geometry (see CS424)

Example Motion Curves

- The position of an object: X(t), Y(t), Z(t)
- Three separate splines
- The angle of a simple joint (e.g. elbow)
- The angles of a complex joint (e.g. hip)
  - Two or more splines
- The size of an object
  - Maybe separated along separate axes
- The colour of an object
- ...

Using motion curves

- Simplest usage:
  - Look at every parameter that changes during the animation
  - Use Hermite interpolation (initialized as Catmull-Rom) based on time
  - Allow user to adjust values, adjust slopes, break continuity, add knots, move knots...
Problem

- Retiming animations is not so simple
  - If you adjust a knot position, it changes the shape of the curve, not just the speed
  - Particularly for Hermite curves - slopes will be off
- Partial solution: separate the shape of the curve from its timing

Time as a Motion Curve

- Rename parameter of motion curves to “u”
  - This is now just a measure of how far along the curve you are, not a real quantity (yet)
- Then make a motion curve for time: u(t)
  - At a particular time, say $t=5/24$ of a second, evaluate spline $u(t)=u(5/24)$
  - Then evaluate the other motion curves at this value of u
    - i.e. motion curves look like $x(u(t))$
- Could have one global timing curve $u(t)$
- Or separately adjust timing for each variable, or group of variables

Parameterization

- Unsatisfactory still: $u$ doesn’t really have a good meaning
- For example, to make an object move with constant speed along an arc, $u(t)$ may be quite complicated!
- For the case of position in space, introduce a new map based on arc length
  - Can easily control the speed of an object
  - Timing curve will now be $s(t)$, where $s$ means how far along the curve (in space)

Arc Length

- Arc length is just the length of a curve
  - Think of wrapping a tape measure along the curve
- Definition:
  \[ s(u) = \int_0^u \left| \frac{dx}{du} \right| du \]
  - Where $x(u)$ is the 3D position of the curve at parameter value $u$
    - Really three curves: $X(u)$, $Y(u)$, $Z(u)$
  - Recall how to measure vector norm:
  \[
  \left| \frac{dx}{du} \right| = \sqrt{\left( \frac{dX}{du} \right)^2 + \left( \frac{dY}{du} \right)^2 + \left( \frac{dZ}{du} \right)^2}
  \]
**Inverse Map**

- The question we really want to answer, though, is what value of $u$ gives us a specific length $s$ along the curve?
  - i.e. invert the arc length function $s(u)$
  - Let’s call this $u(s)$
- Then timing curve is $s(t)$, which feeds into $u(s)$, which feeds into motion curve $x(u)$:
  - Position at time $t$ is $x(u(s(t)))$
- Question remains: how to calculate $u(s)$?

**Numerical Inversion**

- Analytic approach is hopeless
  - Even analytically solving the integral $s(u)$ is hard, solving for $u$ in terms of $s$ is crazy
- Numerical approach works fine
- Use approximate evaluation of $s(u)$ to get a table of values
  - Cut up curve into small line segments, add up their lengths
- Then interpolate a smooth curve through the values (Catmull-Rom)
  - Use table of $s$ values as knots, associated $u$ values as control point values