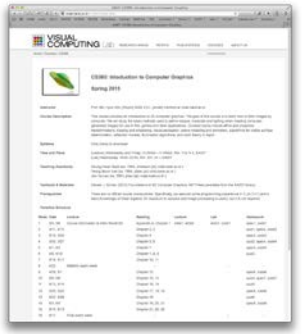
CS380: Introduction to Computer Graphics
Smooth Interpolation
Chapter 9

Min H. Kim
KAIST School of Computing

Assignment

- **HW4 Deadline: 2015.4.16 (Thur.) 23:55**
- Go to the course website and download the files
- Solve the tasks (described in the pdf) and submit your codes with some screenshots to the TA by email.
- **Yeong Beum Lee (yblee@vclab.kaist.ac.kr)**



Quaternion II

SUMMARY

RBT Interpolation Behavior

- Even though the quaternion rotation is left and right invariant, the quaternion rotation + object translation is left invariant.
- The translation of the origin plays special role.
- If we use different object frames for same geometry, we get different interpolations
  - Not right invariant

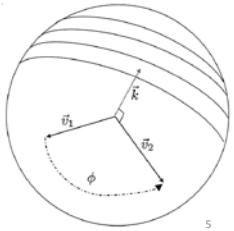
Mental model

- Define the unit direction vectors $\vec{v}_1, \vec{v}_2$: normalize $(\tilde{p}_1 - \tilde{o})$ and normalize $(\tilde{p}_2 - \tilde{o})$ respectively.
- Define the angle

$$\phi = \arccos(\vec{v}_1 \cdot \vec{v}_2)$$

- Define the axis

$$\vec{k} = normalize(\vec{v}_1 \times \vec{v}_2)$$



The balls

- Trackball: M is the rotation of ϕ degrees about the axis $\vec{k}$.
- Arcball: M is the rotation of 2ϕ degrees about the axis $\vec{k}$.
- Could be implemented with matrices or quaternions.
- Arcball is very easy with quaternions

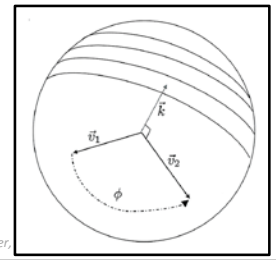
Getting eye coordinates

- Given the (x,y) window coordinates of click, the z coordinate on the sphere can be solved using

$$(x - c_x)^2 + (y - c_y)^2 + (z - 0)^2 = r^2$$

$$z = \sqrt{r^2 - (x - c_x)^2 - (y - c_y)^2}$$

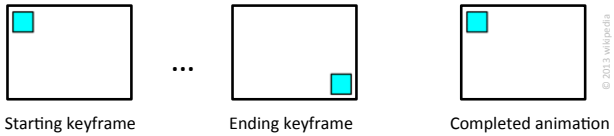
  - $[c_x, c_y, 0]^t$ are the window coordinates of the center of the sphere
  - *r* is the radius of the sphere measured in pixels
  - if outside of the sphere, then clamp to its boundary
  - All we need is normalized $\hat{\mathbf{v}}$, so just normalized such vectors



Chapter 9

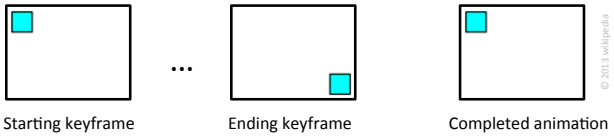
SMOOTH INTERPOLATION

Keyframe animation



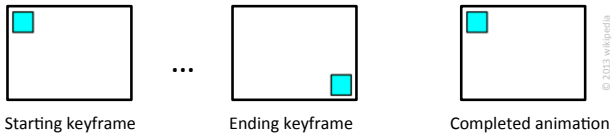
- An animator describes snapshots of a 3D computer graphics animation at a set of discrete times.
  - So-called *key frames*

Keyframe animation



- 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.
  - The rotations are quaternions.

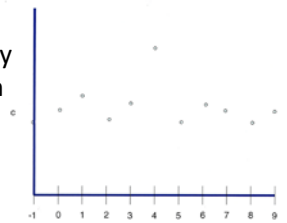
Keyframe animation



- To create a smooth animation, the computer's job is to smoothly 'fill in' the parameter values over a continuous range of times.

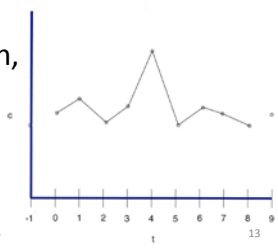
Interpolation

- 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 integers
- 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.



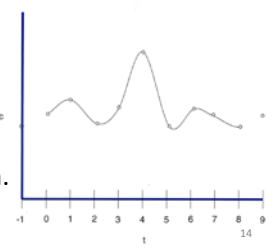
Interpolation

- We show a function $c(t)$, with $t \in [0..8]$ that interpolates the discrete values associated with the integers c_i with $t \in [-1..9]$
  - 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!



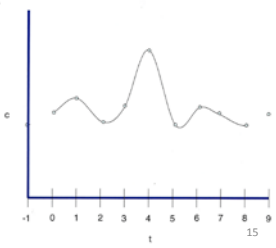
Splines

- 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 present, evaluate and control.
  - Spline behavior much easier to predict than, say, a single high-order polynomial function.



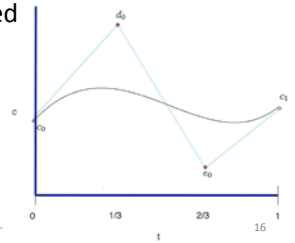
Splines

- Also useful for curves in the plane (e.g., fonts) and space, and the basis for theory of smooth surfaces in space.



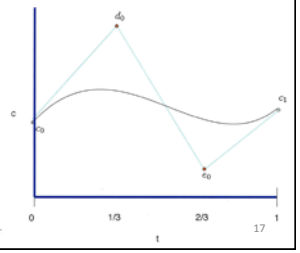
Cubic Bezier function

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



Bezier control polygon

- We will specify a cubic function using four '<u>control values</u>' $c_0, d_0, e_0,$ and c_1
  - Visualized as points in the 2D (t,c) plane
  - With coordinates of four points: $[0,c_0]^t, [1/3,d_0]^t, [2/3,e_0]^t,$ and $[1,c_1]^t$
  - We have also drawn in light blue a poly-line connecting these points; this is called the <u>*control polygon.*</u>



Bezier evaluation

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

$f = (1-t)c_0 + td_0$

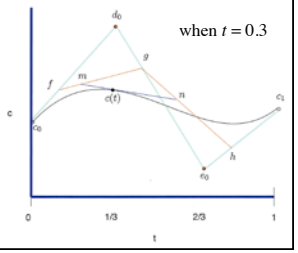
$g = (1-t)d_0 + te_0$

$h = (1-t)e_0 + tc_1$

$m = (1-t)f + tg$

$n = (1-t)g + th$

$c(t) = (1-t)m + tn$



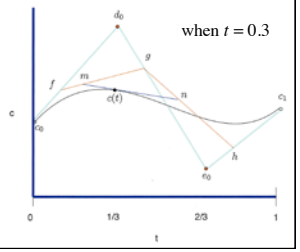
Bezier properties

- By unwrapping the evaluation steps before, we can verify $c(t)$ 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
- The c_i are interpolated:

$c(0) = c_0$ and $c(1) = c_1$



Bezier properties

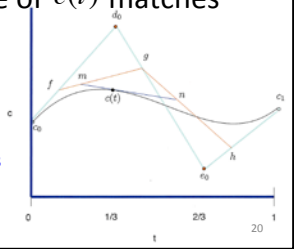
- Let's take the first-order derivatives:

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

$$c'(t) = 3c_1t^2 - 3e_0t^2 - 3c_0(t-1)^2 + 3d_0(t-1)^2 - 6e_0t(t-1) + 3d_0t(2t-2)$$

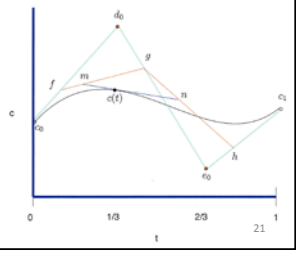
- We see that $c'(0) = 3(d_0 - c_0),\quad c'(1) = 3(c_1 - e_0)$
- We see indeed that the slope of $c(t)$ matches the slope of the control polygon at 0 and 1.

```
syms a b c d t
f(t) = a*(1-t)^3 + 3*b*t*(1-t)^2 + 3*c*t^2*(1-t) + d*t^3
df = diff(f)
df(0) = 3*b - 3*a
df(1) = 3*d - 3*c
```



Bezier properties

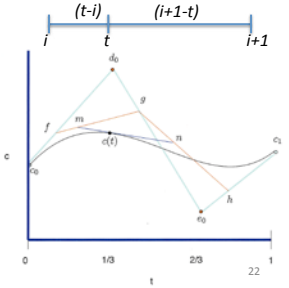
- If we set $c_0 = d_0 = e_0 = c_1 = 1$, then $c(t) = 1$ for all t
  - This property is called *partition of unity*
  - Means that adding a constant value to all control values results in simply adding this constant to $c(t)$



Translated domain

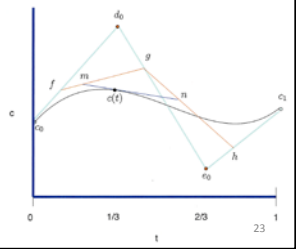
- If we want a cubic function to interpolate value c_i and c_{i+1} at $t = i$ and $t = i+1$, respectively, and calling our two other points d_i and e_i,
- We just have to 'translate' the evaluation algorithm to get

$$f = (1-t+i)c_i + (t-i)d_i$$
$$g = (1-t+i)d_i + (t-i)e_i$$
$$h = (1-t+i)e_i + (t-i)c_{i+1}$$
$$m = (1-t+i)f + (t-i)g$$
$$n = (1-t+i)g + (t-i)h$$
$$c(t) = (1-t+i)m + (t-i)n$$



Translated domain

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

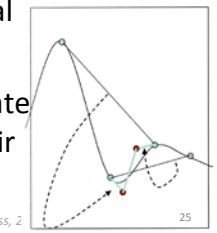


Cubic Bezier Splines

http://www.math.ucla.edu/~baker/java/hoefer/Spline.htm

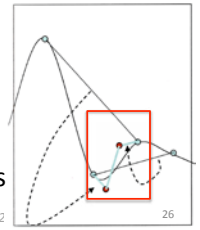
Catmull-Rom Splines (CRS)

- 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 splines' 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 cubic 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
  $c'(t)|_i = \frac{1}{2}(c_{i+1} - c_{i-1})$
  - We look forward and backwards one sample to determine its slope at $t = i$;
  - This is why we need the extra c-values

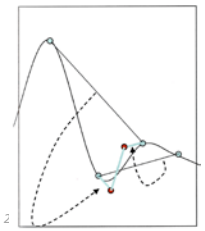


CRS construction

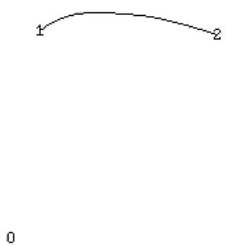
- This also can be stated as $c'(t)|_{i+1} = \frac{1}{2}(c_{i+2} - c_i)$
- Since $c'(i) = 3(d_i - c_i)$ and $c'(i+1) = 3(c_{i+1} - e_i)$, in the Bezier representation, this tells us that we need to set:

$$d_i = \frac{1}{6}(c_{i+1} - c_{i-1}) + c_i$$

$$e_i = \frac{-1}{6}(c_{i+2} - c_i) + c_{i+1}$$



Catmull-Rom Splines



http://www.cse.unsw.edu.au/~lambert/splines/CatmullRom.html

Quaternion Splining

- Time-varying translation (t_x, t_y and t_z)
- Splines are based on scalars, or vectors.
- Theory does not directly apply (ongoing research)
- If we want to interpolates a set of quaternions we need to hack by association
- [Hacking] Substitute appropriate quaternion operations for the scalar operations
  - Bezier scalar addition → quaternion multiplication
  - Bezier scalar negation → quaternion inversion
  - Bezier scalar multiplication → quaternion power

Quaternion Splining

- Bezier evaluation steps of the form:

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

- Become: $\mathrm{r = slerp(p, q, t)}$
- And the d_i and e_i equation values are defined as

$$d_i = ((c_{i+1}c_{i-1}^{-1})^{1/6})c_i \quad \Leftarrow d_i = \frac{1}{6}(c_{i+1} - c_{i-1}) + c_i$$

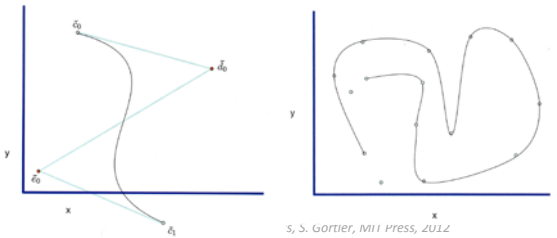
$$e_i = ((c_{i+2}c_i^{-1})^{-1/6})c_{i+1} \quad \Leftarrow e_i = \frac{-1}{6}(c_{i+2} - c_i) + c_{i+1}$$

- 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 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.

Bezier curve (2D) Catmull-Rom curve (2D)



Curves

- Applying the spline construction independently to the x, y and z coordinates, one gets a point-valued spline function $\tilde{c}_i$;
- 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.