418382 สภาพแวดล้อมการทำงานคอมพิวเตอร์กราฟิกส์
การบรรยายครั้งที่ 7

pramook@gmail.com

Animation Pipeline
Keyframing Introduction

COMPUTER ANIMATION

15-497/15-861

Producing an Animation

- Film runs at 24 frames per second (fps)
  - That's 1440 pictures to create per minute
  - 1800 fpm for video (30fps)
- Productions issues:
  - Need to stay organized for efficiency and cost reasons
  - Need to create the frames systematically
- Artistic issues:
  - How to create the desired look and mood while conveying story?
  - Artistic vision has to be converted into a sequence of still frames
  - Not enough to get the stills right--must look right at full speed
    - » Hard to "see" the motion given the stills
    - » Hard to "see" the motion at the wrong frame rate

    A lesson you will painfully learn in this class!

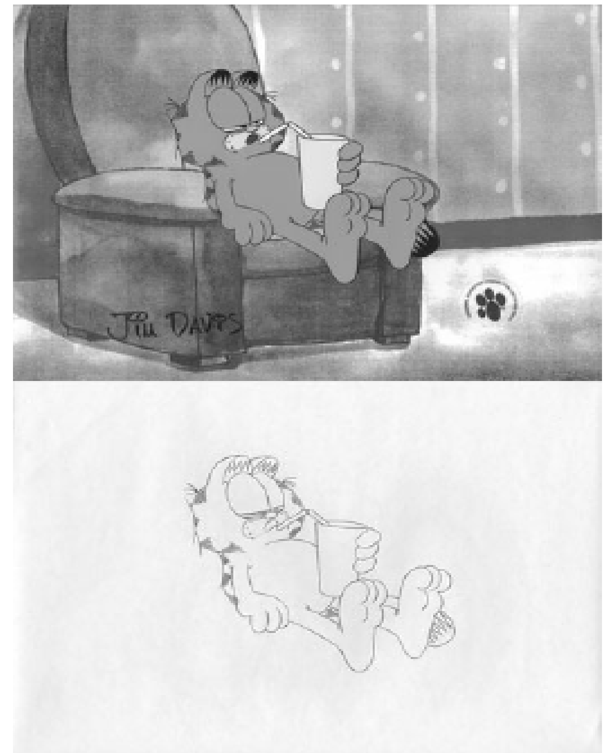
Traditional Animation: The Process

- Story board
  - Sequence of drawings with descriptions
  - Story-based description
- Key Frames
  - Draw a few important frames as line drawings
    - » For example, beginning of stride, end of stride
  - Motion-based description
- Inbetweens
  - Draw the rest of the frames
  - People who draw these don't get paid much
- Painting
  - Redraw onto acetate *Cels*, color them in
  - These people get paid even less

From http://www.animationartgallery.com/

Layered Motion

- It's often useful to have multiple layers of animation
  - How to make an object move in front of a background?
  - Use one layer for background, one for object
  - Can have multiple animators working simultaneously on different layers, avoid re-drawing and flickering
- Transparent acetate allows multiple *layers*
  - Draw each separately
  - Stack them together on a copy stand
  - Transfer onto film by taking a photograph of the stack



From http://www.animationartgallery.com/

Computer-Assisted Animation

- Computerized Cel painting
  - Digitize the line drawing, color it using seed fill
  - Eliminates cel painters (low rung on totem pole)
  - Widely used in production (little hand painting any more)
  - e.g. *Lion King*
- Cartoon Inbetweening
  - Automatically interpolate between two drawings to produce inbetweens (morphing)
  - Hard to get right
    - » inbetweens often don't look natural
    - » what are the parameters to interpolate? Not clear...
    - » not used very often

True Computer Animation

- Generate the images by rendering a 3-D model
- Vary the parameters to produce the animation
- Brute force
  - Manually set the parameters for each and every frame
  - For an n parameter model: $1440n$ values per minute
- Computer keyframing
  - Lead animators create the important frames with 3-D computer models
  - Unpaid computers draw the inbetweens
  - The dominant production method

Digital Production Pipeline

- Story
- Visual Development
- Character Design
- Storyboards
- Scene Layout
- Modeling
- Animation
- Shading and Texturing
- Lighting
- Rendering
- Post Production

Animatic

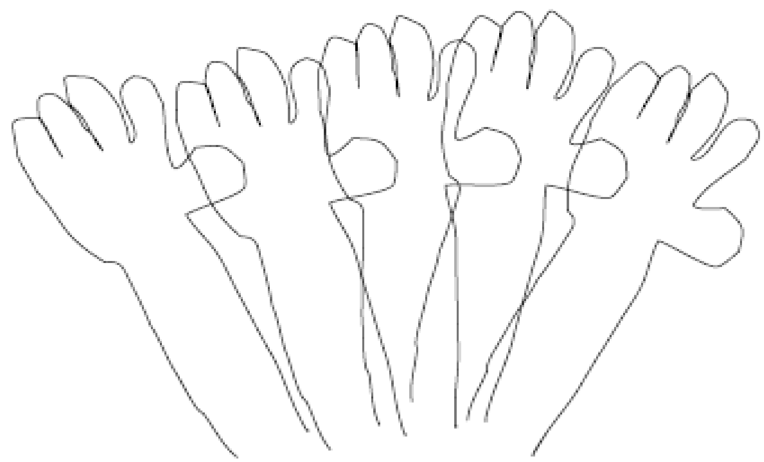
Keyframing

COMPUTER ANIMATION

15-497/15-861

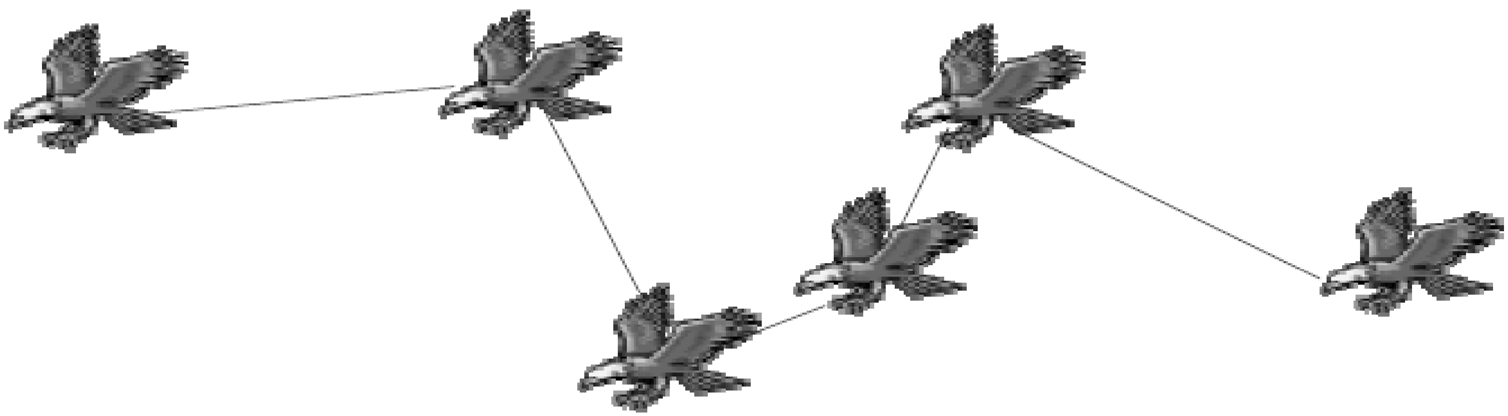
Keyframing in 2D

- Highly skilled animator draws the important, or key frames
- Less skilled (lower paid) animator draws the in-between frames



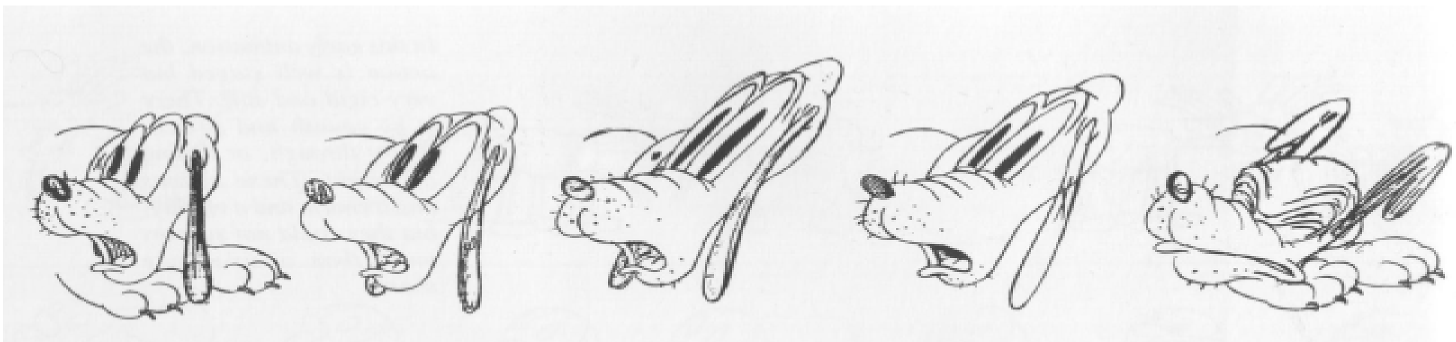
Keyframing in 3D

- Animator specifies the important key frames
- Computer generates the in-betweens automatically using interpolation
- Rigid body motion isn't nearly enough—even for this sprite



What is a key?

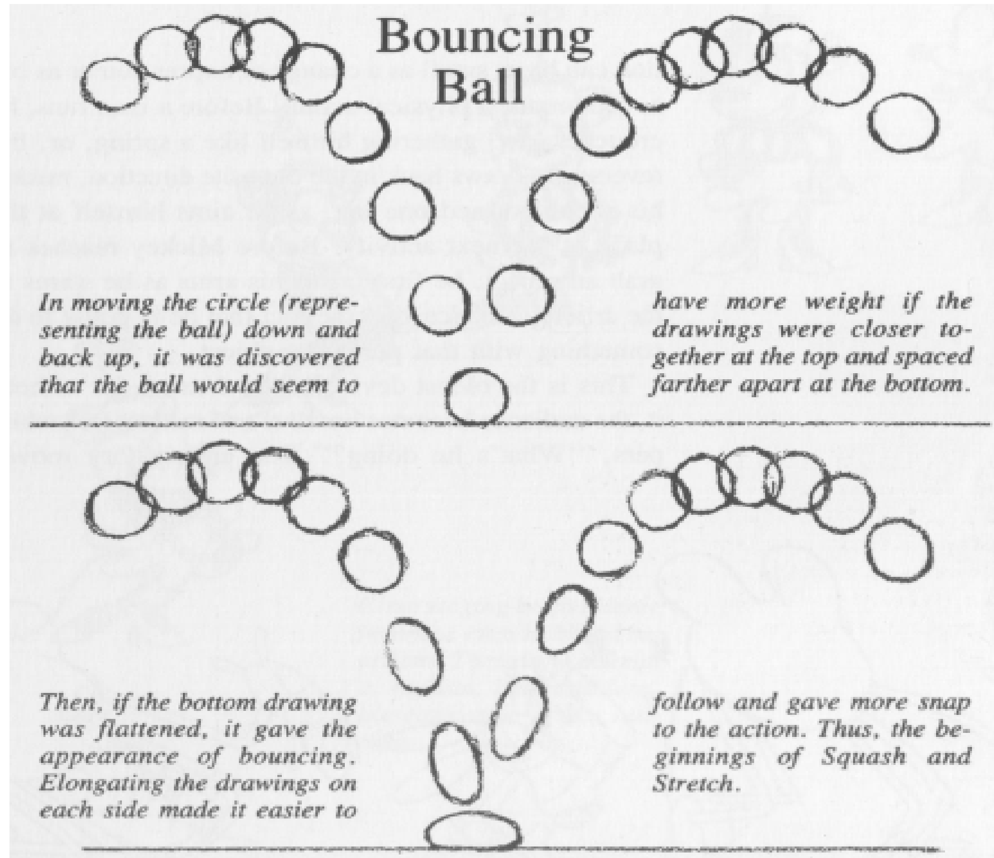
- Hard to interpolate hand-drawn images
  - Computers don't help much



- The situation is different in computer animation:
  - Each keyframe is a defined by a bunch of parameters (state)
  - Sequence of keyframes = points in high-dimensional state space
- Computer inbetweening interpolates these points

What is a key?

- For a bounding ball?
  - Position in 3D
  - Orientation?
  - Squishedness?



What is a key?

- For a monster?
  - –Position and orientation in 3D
  - –Joint angles of the hierarchy
  - –Deformations?
  - –Facial features
  - –Hair/fur???
  - –Clothing???
- Scene elements?
  - –Lights
  - –Camera

Monster trailers…

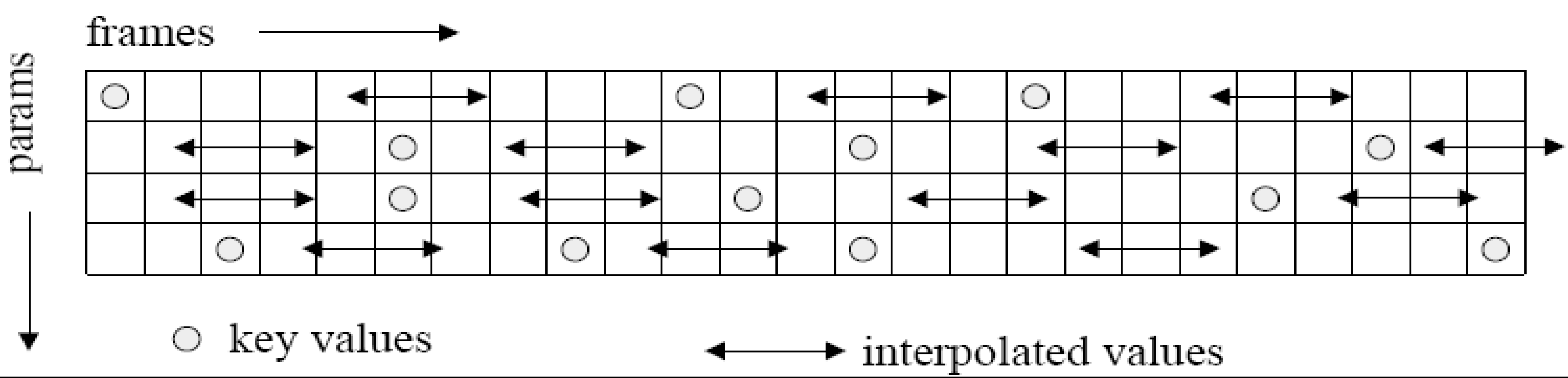
Keyframe Animation: Production Issues

- How to learn the craft?
  - apprentice to an animator
  - practice, practice, practice
- Pixar starts with animators, teaches them computers and starts with computer folks and teaches them some art

- Gives good control over motion
- Eliminates much of the labor of traditional animation
  - But still very labor-intensive
- Impractical for complex scenes with everything moving: grass in the wind, water, and crowd scenes, for example

Keyframing Basics

- Despite the name, there aren't really keyframes, *per se*.
- For each variable, specify its value at the "important" frames. Not all variables need agree about which frames are important.
- Hence, *key values* rather than key frames
- Create path for each parameter by interpolating key values

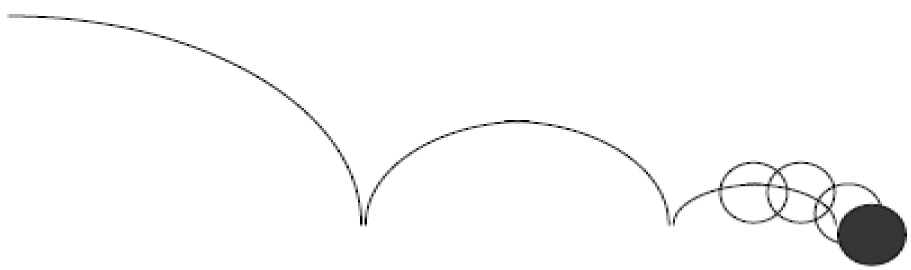


Keyframing Recipe

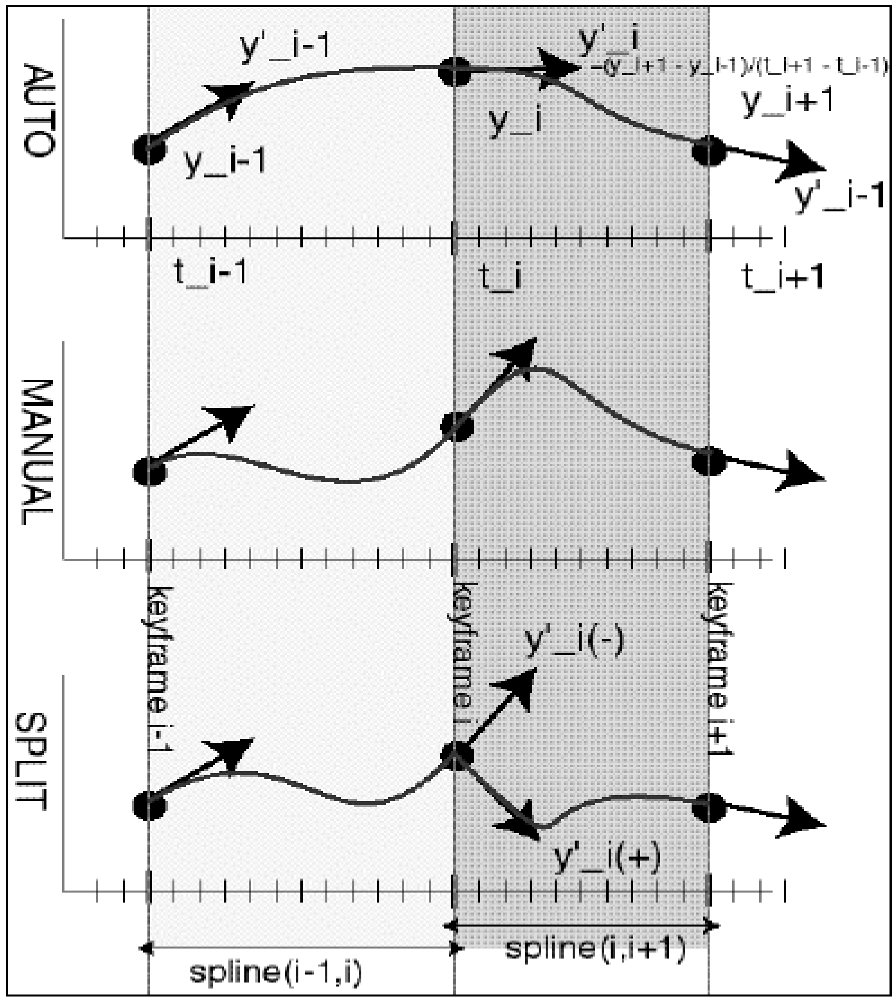
- Specify the key frames
  - rigid transforms, forward kinematics, inverse kinematics
- Specify the type of interpolation
  - linear,cubic, etc. parametric curves
- Specify the speed profile of the interpolation
  - constant velocity, ease-in,out, etc.
- Computer generates the in-between frames using this information

Splines for Interpolation

- Classic example - a ball bouncing under gravity
  - zero vertical velocity at start
  - high downward velocity just before impact
  - lower upward velocity after
  - motion produced by fitting a smooth spline looks unnatural
- What kind of continuity/control do we need?

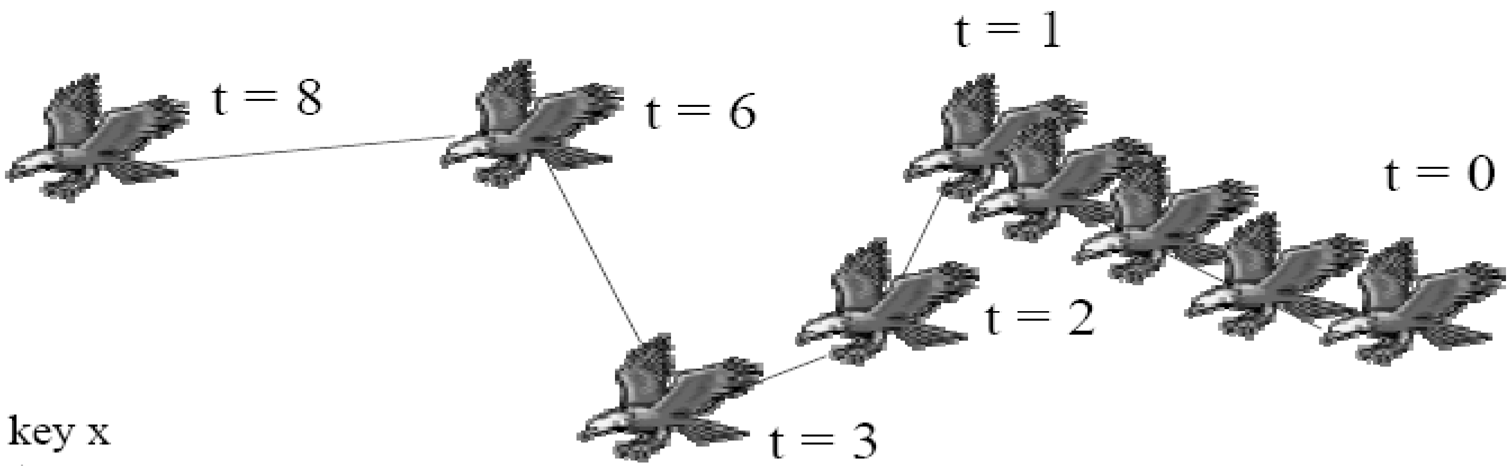


How Do You Interpolate Between Keys?



Linear Interpolation

Using linear "arcs" between keyframes



start key x

$$x = x_0 + \frac{t - t_0}{t_1 - t_0}(x_1 - x_0)$$

intermediate x

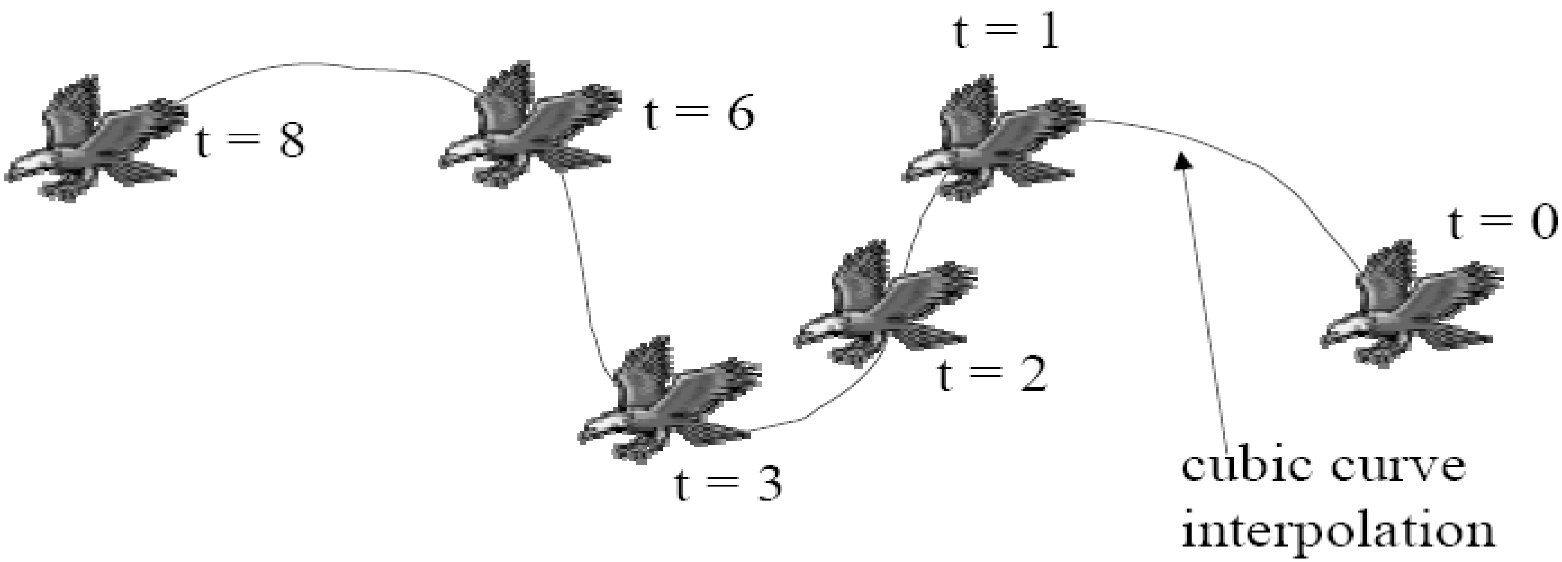
"a portion of"

total x distance between start and end key

where t_0 is the parameter value at the start of a segment, t_1 is the parameter value at the end of a segment and t is the parameter value for which you want to find the position, x

Cubic Curve Interpolation

- Like a thin strip that can be bent to interpolate the points of interest



CS 445 / 645
Introduction to Computer Graphics

Lecture 22

Hermite Splines

UNIVERSITY of VIRGINIA

Splines – Old School

Duck

Spline

Representations of Curves

Use a sequence of points…

- Piecewise linear - does not accurately model a smooth line
- Tedious to create list of points
- Expensive to manipulate curve because all points must be repositioned

Instead, model curve as piecewise-polynomial

- $x = x(t)$, $y = y(t)$, $z = z(t)$
  – where $x()$, $y()$, $z()$ are polynomials

UNIVERSITY of VIRGINIA

Specifying Curves (hyperlink)

Control Points

- A set of points that influence the curve’s shape

Knots

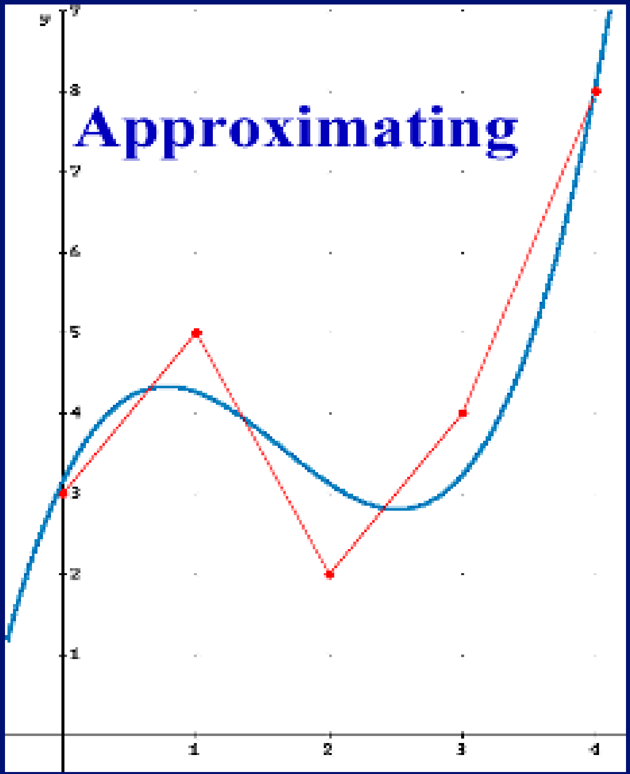
- Control points that lie on the curve

Interpolating Splines

- Curves that pass through the control points (knots)

Approximating Splines

- Control points merely influence shape

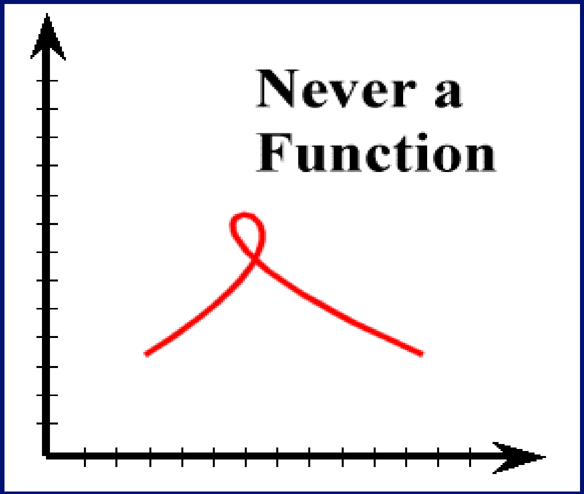


Parametric Curves

Very flexible representation

They are not required to be functions

- They can be multivalued with respect to any dimension



Cubic Polynomials

$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$

- Similarly for y(t) and z(t)

Let t: (0 <= t <= 1)

Let T = [t^3 t^2 t 1]

Coefficient Matrix C

$$\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} * \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix}$$

*Curve: Q(t) = T*C*

UNIVERSITY of VIRGINIA

Piecewise Curve Segments

One curve constructed by connecting many smaller segments end-to-end

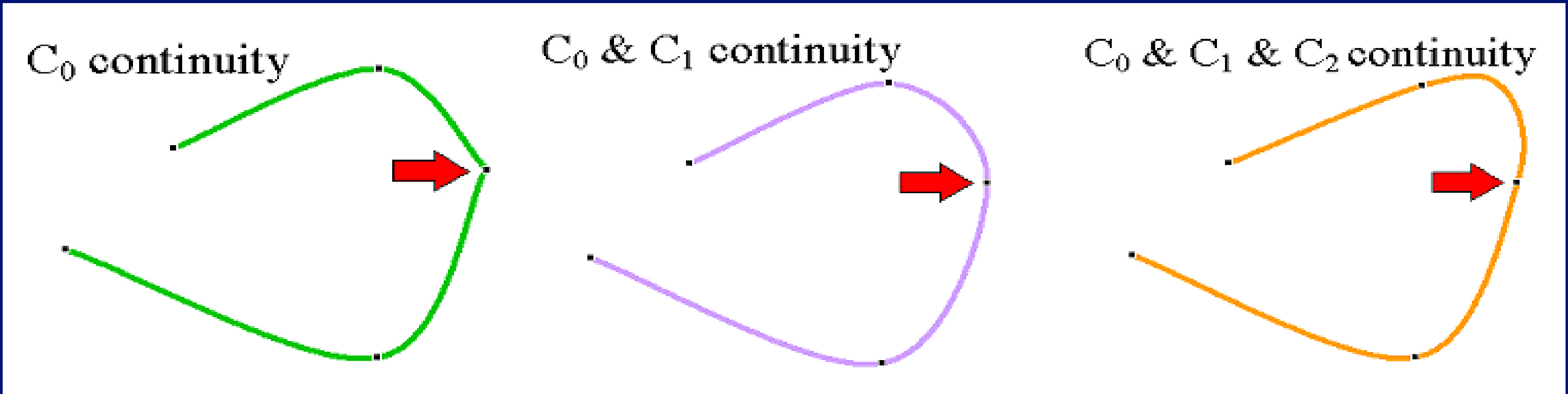
- Must have rules for how the segments are joined

Continuity describes the joint

- Parametric continuity
- Geometric continuity

Parametric Continuity

- C_1 is tangent continuity (velocity)
- C_2 is 2nd derivative continuity (acceleration)
- Matching direction and magnitude of d^n / dt^n
    - C^n continous



Geometric Continuity

If positions match

- G^0 geometric continuity

If direction (but not necessarily magnitude) of tangent matches

- G^1 geometric continuity
- The tangent value at the end of one curve is proportional to the tangent value of the beginning of the next curve

Parametric Cubic Curves

In order to assure C_2 continuity, curves must be of at least degree 3

Here is the parametric definition of a cubic (degree 3) spline in two dimensions

How do we extend it to three dimensions?

$$x = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y = a_y t^3 + b_y t^2 + c_y t + d_y$$

Parametric Cubic Splines

Can represent this as a matrix too

$$x = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y = a_y t^3 + b_y t^2 + c_y t + d_y$$

$$\begin{bmatrix} x & y \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix}$$

Coefficients

So how do we select the coefficients?

- $[a_x\ b_x\ c_x\ d_x]$ and $[a_y\ b_y\ c_y\ d_y]$ must satisfy the constraints defined by the knots and the continuity conditions

Parametric Curves

Difficult to conceptualize curve as

$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$

(artists don't think in terms of coefficients of cubics)

Instead, define curve as weighted combination of 4 well-defined cubic polynomials

(wait a second! Artists don't think this way either!)

Each curve type defines different cubic polynomials and weighting schemes

Parametric Curves

***Hermite** – two endpoints and two endpoint tangent vectors*

***Bezier** - two endpoints and two other points that define the endpoint tangent vectors*

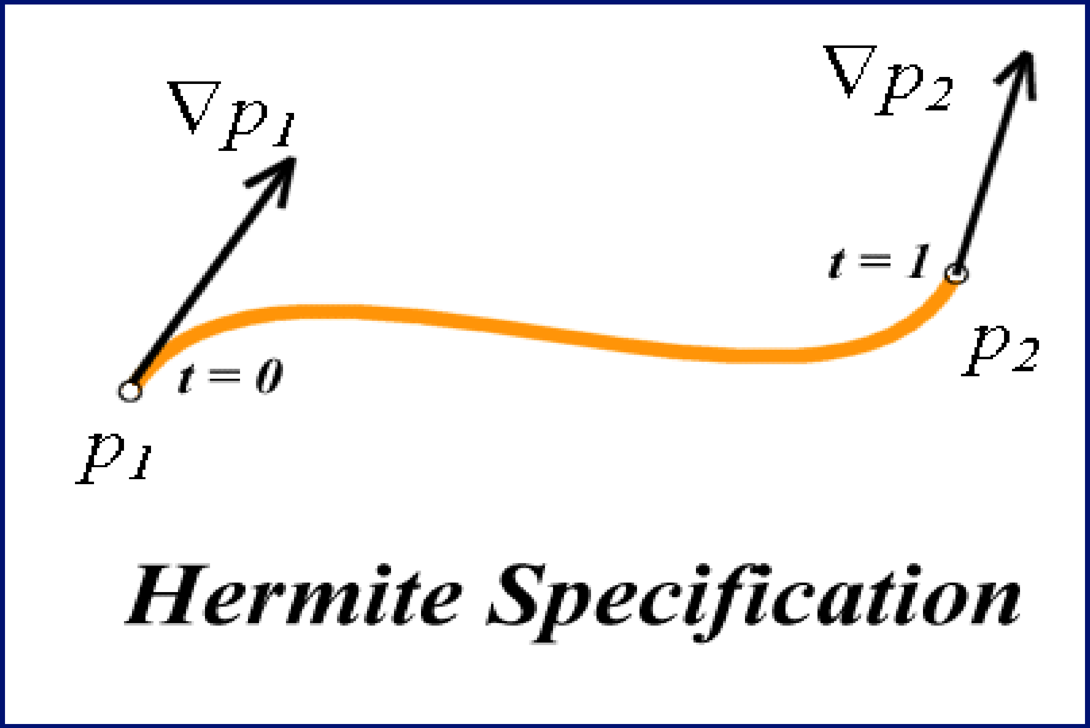
***Splines** – four control points*

- C1 and C2 continuity at the join points
- Come close to their control points, but not guaranteed to touch them

Examples of Splines

Hermite Cubic Splines

An example of knot and continuity constraints



Hermite Cubic Splines

One cubic curve for each dimension

A curve constrained to x/y-plane has two curves:

$$f_x(t) = at^3 + bt^2 + ct + d = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$f_y(t) = et^3 + ft^2 + gt + h = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix}$$

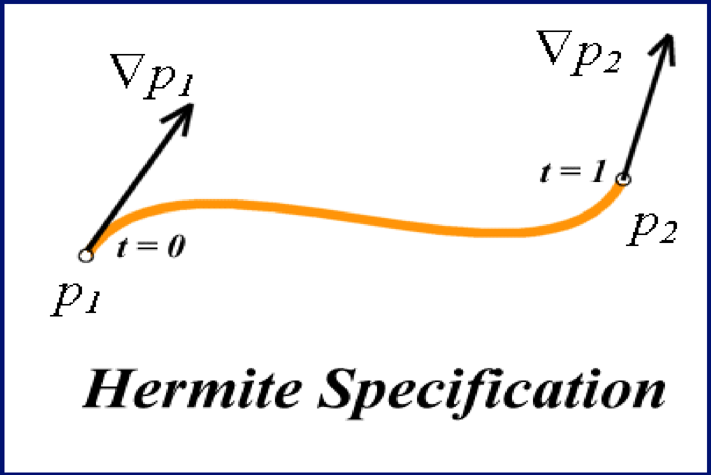
Hermite Cubic Splines

A 2-D Hermite Cubic Spline is defined by eight parameters: a, b, c, d, e, f, g, h

How do we convert the intuitive endpoint constraints into these (relatively) unintuitive eight parameters?

We know:

- (x, y) position at t = 0, p_1
- (x, y) position at t = 1, p_2
- (x, y) derivative at t = 0, dp/dt
- (x, y) derivative at t = 1, dp/dt



Hermite Cubic Spline

We know:

- (x, y) position at t = 0, p_1

$$f_x(0) = a0^3 + b0^2 + c0 + d$$

$$= \begin{bmatrix} 0^3 & 0^2 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$f_x(0) = d = p_{1_x}$$

$$f_y(0) = e0^3 + f0^2 + g0 + h$$

$$= \begin{bmatrix} 0^3 & 0^2 & 0 & 1 \end{bmatrix} \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix}$$

$$f_y(0) = h = p_{1_y}$$

Hermite Cubic Spline

We know:

- (x, y) position at t = 1, p_2

$$
\begin{aligned}
f_x(1) &= a1^3 + b1^2 + c1 + d \\
&= \begin{bmatrix} 1^3 & 1^2 & 1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}
\end{aligned}
$$

$$f_x(1) = a + b + c + d = p_{2_x}$$

$$
\begin{aligned}
f_y(1) &= e1^3 + f1^2 + g1 + h \\
&= \begin{bmatrix} 1^3 & 1^2 & 1 & 1 \end{bmatrix} \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix}
\end{aligned}
$$

$$f_y(1) = e + f + g + h = p_{2_y}$$

Hermite Cubic Splines

So far we have four equations, but we have eight unknowns

Use the derivatives

$$f_x(t) = at^3 + bt^2 + ct + d$$

$$f_x'(t) = 3at^2 + 2bt + c$$

$$f_x'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$f_y(t) = et^3 + ft^2 + gt + h$$

$$f_y'(t) = 3et^2 + 2ft + g$$

$$f_y'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix}$$

Hermite Cubic Spline

We know:

- (x, y) derivative at t = 0, dp/dt

$$f_x'(0) = 3a0^2 + 2b0 + c$$

$$= \begin{bmatrix} 3 \cdot 0^2 & 2 \cdot 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$f_x'(0) = c = {dp_{1_x}}/{dt}$$

$$f_y'(0) = 3e0^2 + 2f0 + g$$

$$= \begin{bmatrix} 3 \cdot 0^2 & 2 \cdot 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix}$$

$$f_y'(0) = g = {dp_{1_y}}/{dt}$$

Hermite Cubic Spline

We know:

- (x, y) derivative at t = 1, dp/dt

$$f_x'(1) = 3a1^2 + 2b1 + c$$

$$= \begin{bmatrix} 3 \cdot 1^2 & 2 \cdot 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$f_x'(1) = 3a + 2b + c = {dp_{1_x}}/{dt}$$

$$f_y'(1) = 3e1^2 + 2f1 + g$$

$$= \begin{bmatrix} 3 \cdot 1^2 & 2 \cdot 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix}$$

$$f_y'(1) = 3e + 2f + g = {dp_{1_y}}/{dt}$$

Hermite Specification

Matrix equation for Hermite Curve

$$\begin{array}{ccc}
 & \begin{array}{cccc} t^3 & t^2 & t^1 & t^0 \end{array} & \\
\begin{array}{c} p_1 \\ p_2 \\ \nabla p_1 \\ \nabla p_2 \end{array} &
\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}
\begin{bmatrix} a & e \\ b & f \\ c & g \\ d & h \end{bmatrix}
=
\begin{bmatrix} p_{1_x} & p_{1_y} \\ p_{2_x} & p_{2_y} \\ dp_{1_x}/dt & dp_{1_y}/dt \\ dp_{1_x}/dt & dp_{2_y}/dt \end{bmatrix}
& \begin{array}{c} t = 0 \\ t = 1 \\ t = 0 \\ t = 1 \end{array}
\end{array}$$

Solve Hermite Matrix

$$\begin{bmatrix}0&0&0&1\\1&1&1&1\\0&0&1&0\\3&2&1&0\end{bmatrix}^{-1}\begin{bmatrix}p_{1_x} & p_{1_y}\\ p_{2_x} & p_{2_y}\\ dp_{1_x}/dt & dp_{1_y}/dt \\ dp_{1_x}/dt & dp_{2_y}/dt\end{bmatrix} = \begin{bmatrix}a&e\\b&f\\c&g\\d&h\end{bmatrix}$$

Spline and Geometry Matrices

$$\underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{M_{Hermite}} \underbrace{\begin{bmatrix} p_{1_x} & p_{1_y} \\ p_{2_x} & p_{2_y} \\ dp_{1_x}/dt & dp_{1_y}/dt \\ dp_{1_x}/dt & dp_{2_y}/dt \end{bmatrix}}_{G_{Hermite}} = \begin{bmatrix} a & e \\ b & f \\ c & g \\ d & h \end{bmatrix}$$

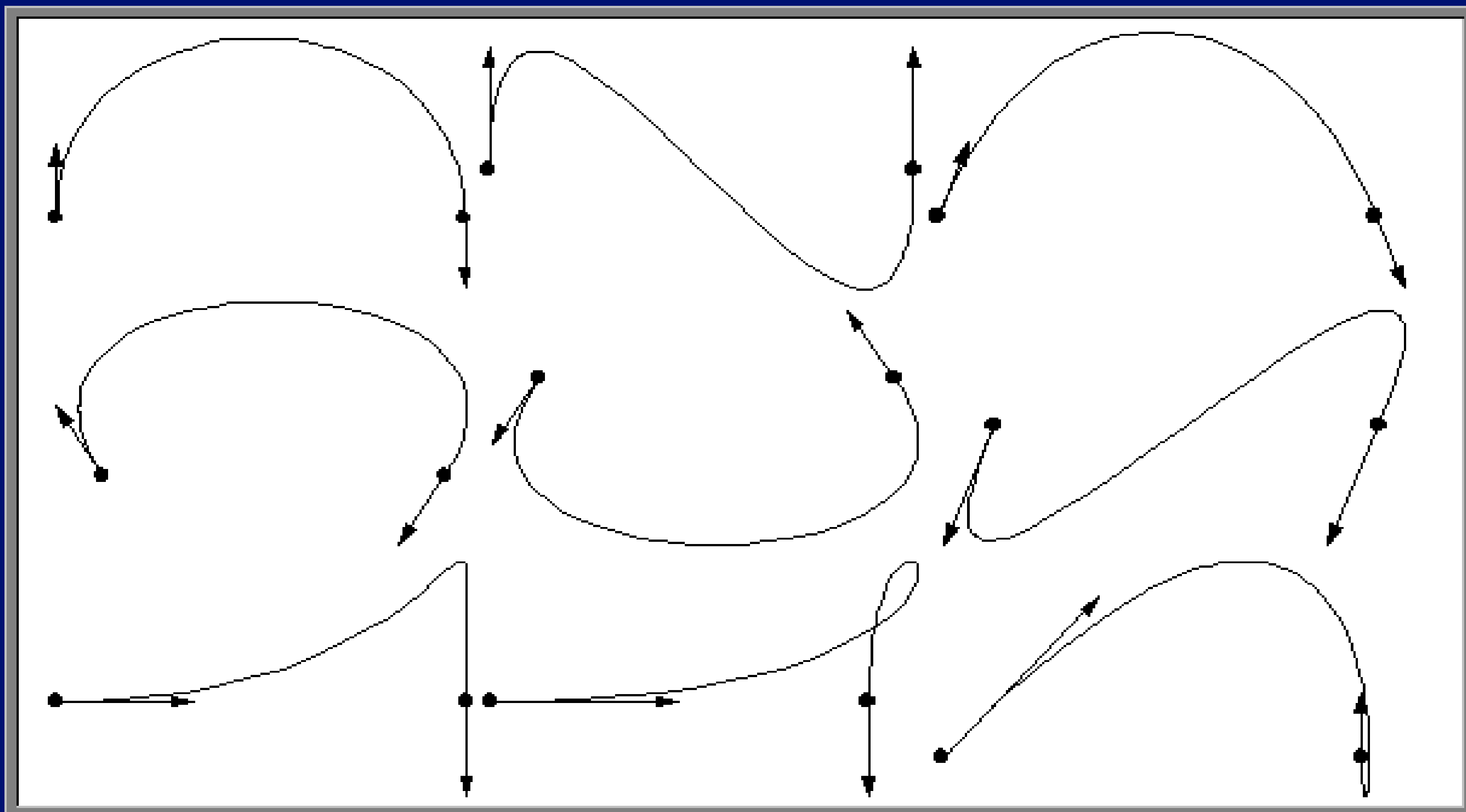
$M_{Hermite}$ $G_{Hermite}$

UNIVERSITY of VIRGINIA

Resulting Hermite Spline Equation

$$\begin{bmatrix} x & y \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{Hermite}} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \frac{dx_1}{dt} & \frac{dy_1}{dt} \\ \frac{dx_2}{dt} & \frac{dy_2}{dt} \end{bmatrix}}_{\mathbf{G}_{Hermite}}$$

Sample Hermite Curves



Blending Functions

By multiplying first two matrices in lower-left equation, you have four functions of 't' that blend the four control parameters

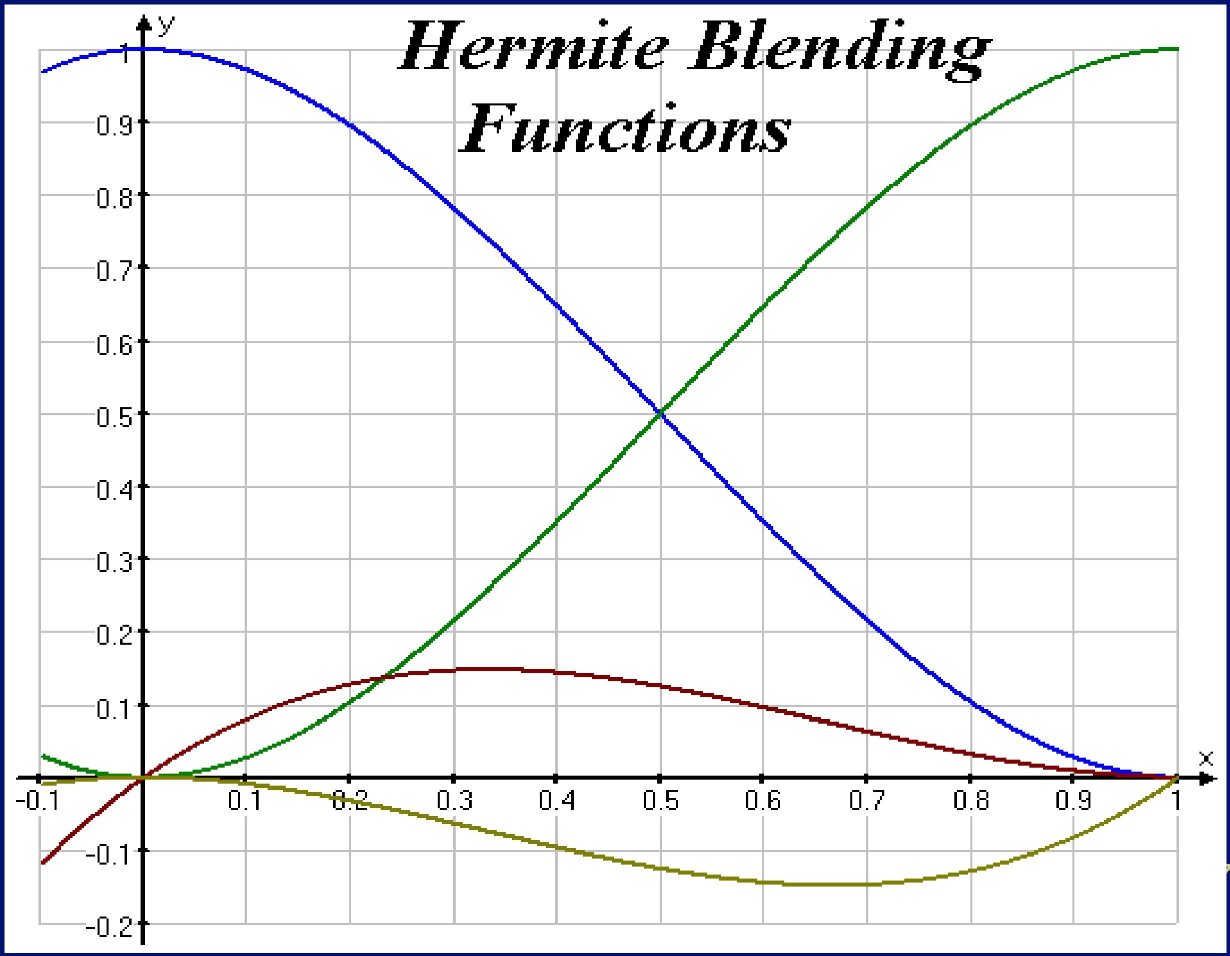
These are blending functions

$$[x \quad y] = [t^3 \quad t^2 \quad t \quad 1] \underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{Hermite}} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \frac{dx_1}{dt} & \frac{dy_1}{dt} \\ \frac{dx_2}{dt} & \frac{dy_2}{dt} \end{bmatrix}}_{\mathbf{G}_{Hermite}}$$

$$p(t) = \begin{bmatrix} 2t^3 - 3t^2 + 1 \\ -2t^3 + 3t^2 \\ t^3 - 2t^2 + t \\ t^3 - t^2 \end{bmatrix}^{\mathbf{T}} \begin{bmatrix} p_1 \\ p_2 \\ \nabla p_1 \\ \nabla p_2 \end{bmatrix}$$

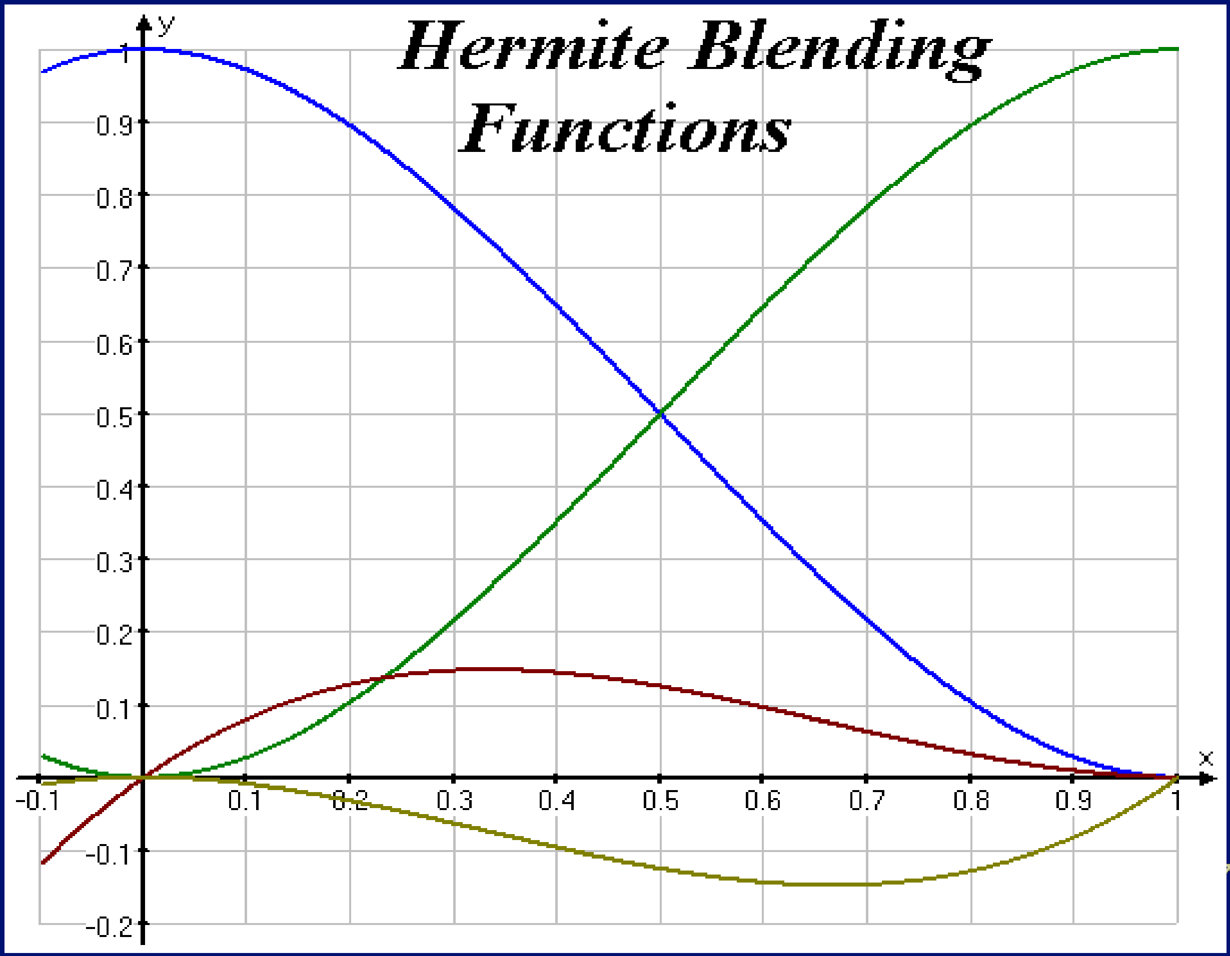
Hermite Blending Functions

If you plot the blending functions on the parameter 't'



Hermite Blending Functions

Remember, each blending function reflects influence of P_1, P_2, ΔP_1, ΔP_2 on spline's shape



CS 445 / 645
Introduction to Computer Graphics

Lecture 23

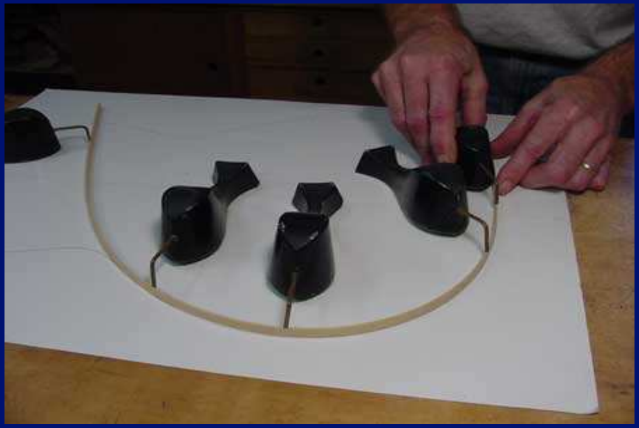
Bézier Curves

UNIVERSITY of VIRGINIA

Splines - History

- *Draftsman use ‘ducks’ and strips of wood (splines) to draw curves*
- *Wood splines have second-order continuity*
- *And pass through the control points*

A Duck (weight)

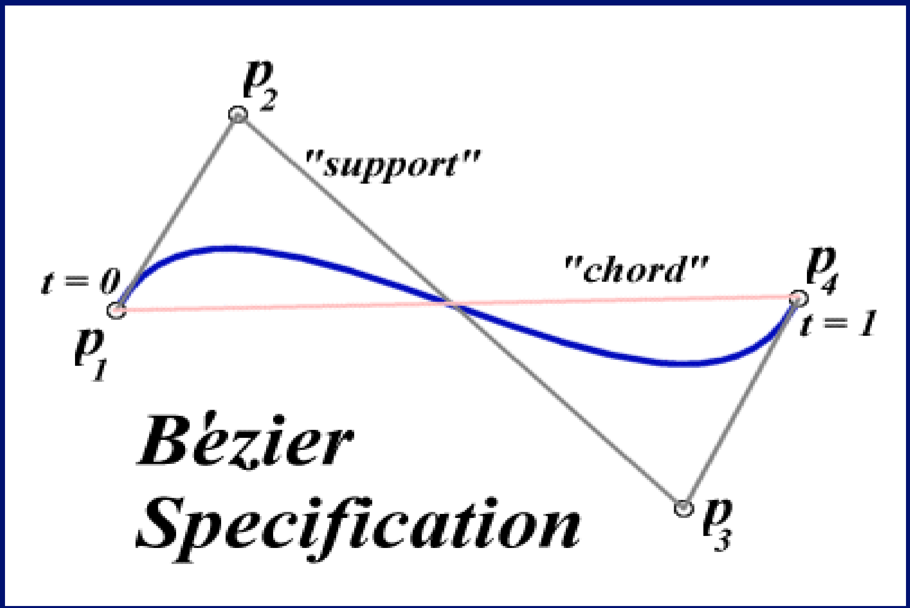


Ducks trace out curve

Bézier Curves

Similar to Hermite, but more intuitive definition of endpoint derivatives

Four control points, two of which are knots



Bézier Curves

The derivative values of the Bezier Curve at the knots are dependent on the adjacent points

$$\nabla p_1 = 3(p_2 - p_1)$$
$$\nabla p_4 = 3(p_4 - p_3)$$

The scalar 3 was selected just for this curve

Bézier vs. Hermite

We can write our Bezier in terms of Hermite

- Note this is just matrix form of previous equations

$$\underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \frac{dx_1}{dt} & \frac{dy_1}{dt} \\ \frac{dx_2}{dt} & \frac{dy_2}{dt} \end{bmatrix}}_{\mathbf{G}_{Hermite}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}}_{\mathbf{G}_{Bezier}}$$

Bézier vs. Hermite

Now substitute this in for previous Hermite

$$\begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} = \underbrace{\underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{Hermite}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix}}_{M_{Bezier}} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}}_{\mathbf{G}_{Bezier}}$$

Bézier Basis and Geometry Matrices

Matrix Form

$$\begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} = \underbrace{\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{Bezier}} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}}_{\mathbf{G}_{Bezier}}$$

But why is M_{Bezier} a good basis matrix?

Bézier Blending Functions

Look at the blending functions

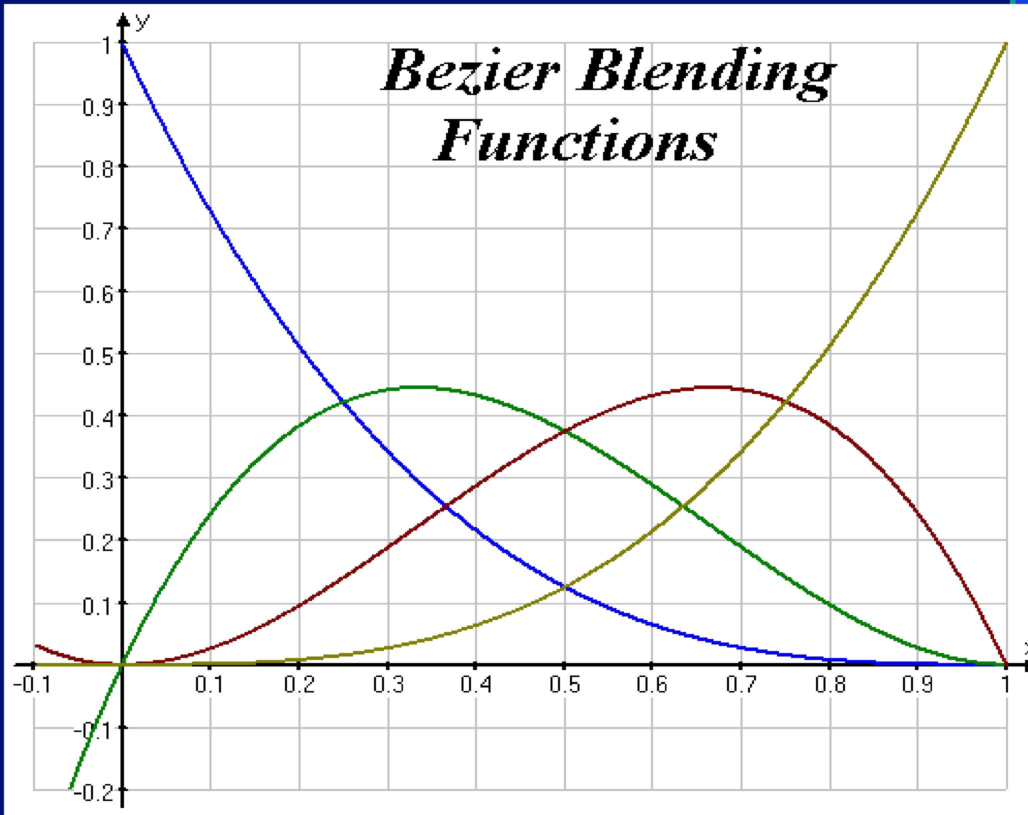
$$p(t) = \begin{bmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 3t^2(1-t) \\ t^3 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

This family of polynomials is called order-3 Bernstein Polynomials

- $C(3, k)\ t^k (1-t)^{3-k}$; $0 <= k <= 3$
- They are all positive in interval [0,1]
- Their sum is equal to 1

Bézier Blending Functions

- *Thus, every point on curve is linear combination of the control points*
- *The weights of the combination are all positive*
- *The sum of the weights is 1*
- *Therefore, the curve is a convex combination of the control points*



Convex combination of control points

Will always remain within bounding region (convex hull) defined by control points

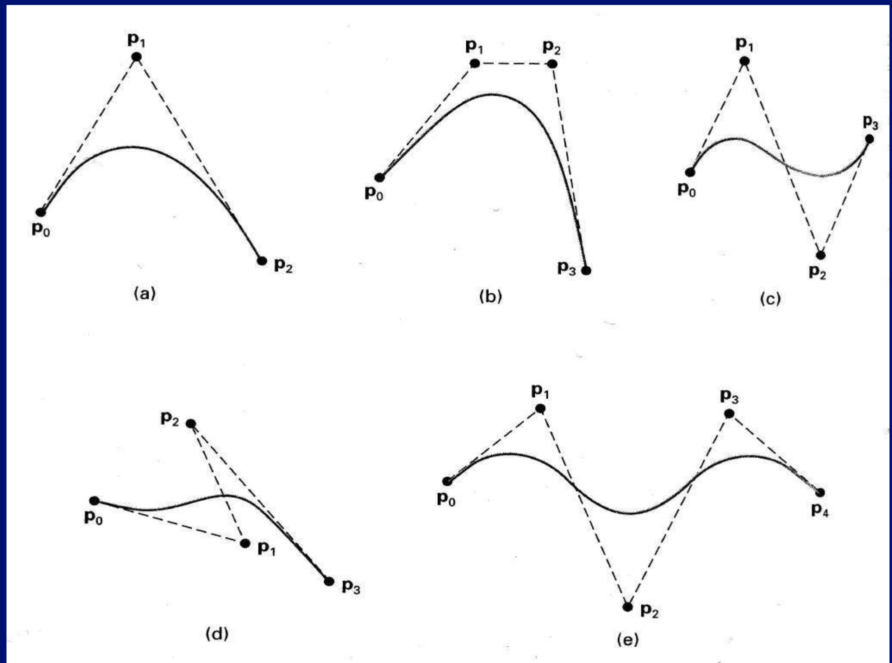
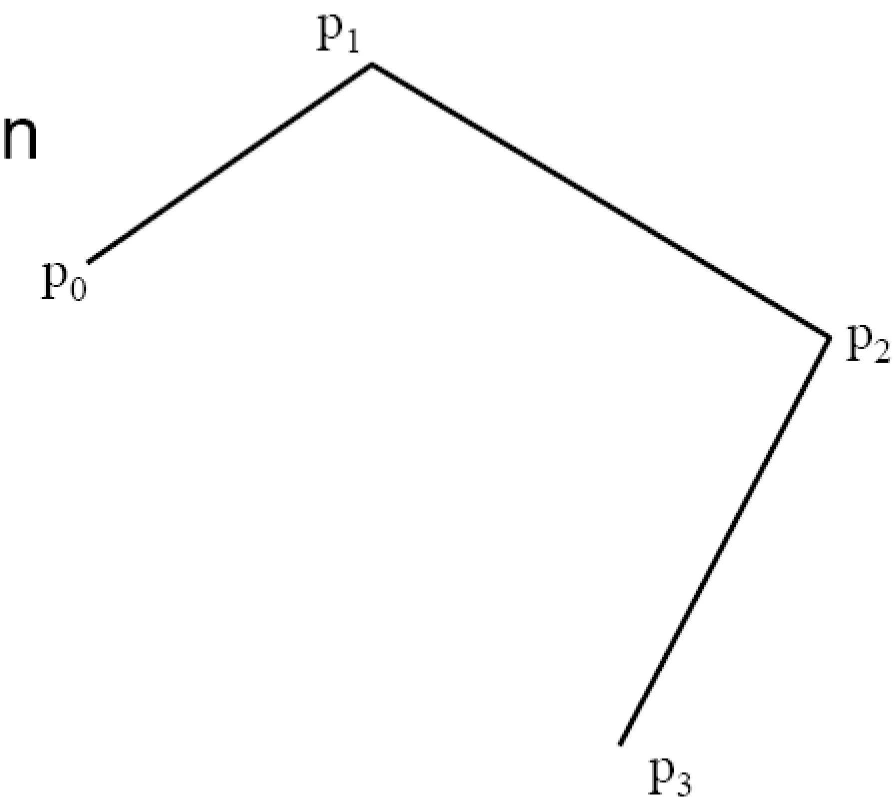


Figure 10-34
Examples of two-dimensional Bézier curves generated from three, four, and five control points. Dashed lines connect the control-point positions.

UNIVERSITY of VIRGINIA

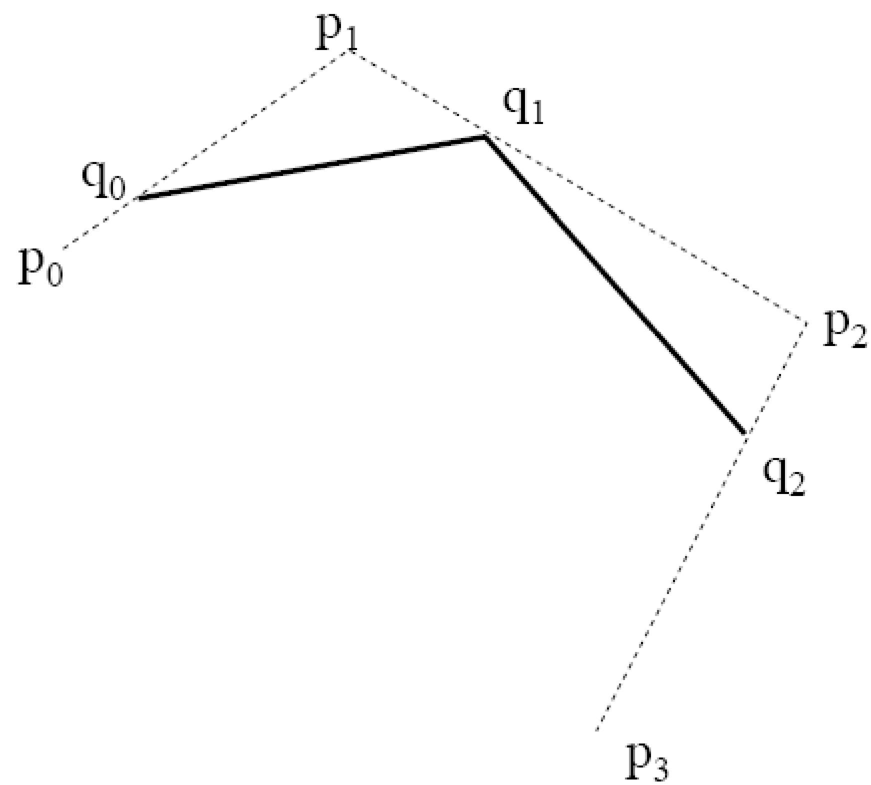
de Castlejau Algorithm

- Find the point **x** on the curve as a function of parameter t:



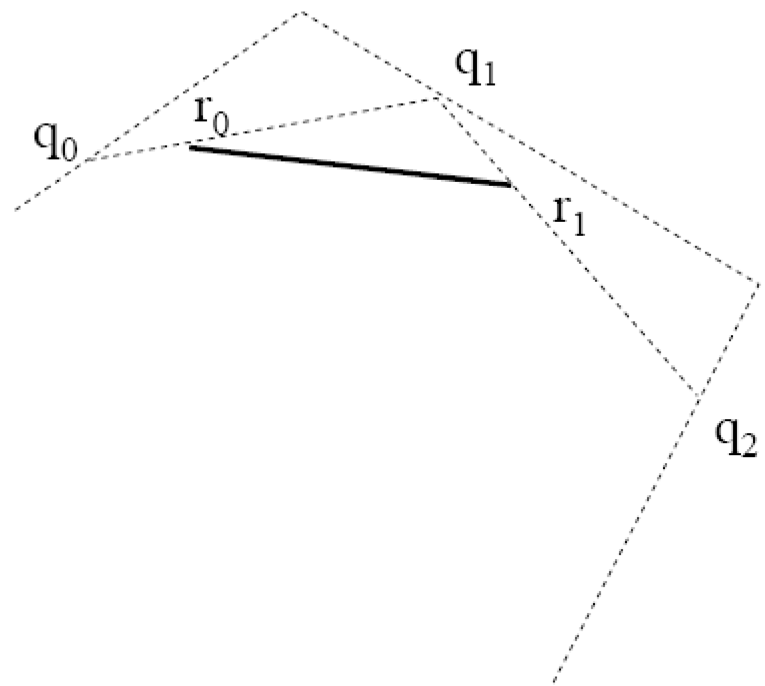
de Castlejau Algorithm

$$\mathbf{q}_0 = Lerp(t, \mathbf{p}_0, \mathbf{p}_1)$$
$$\mathbf{q}_1 = Lerp(t, \mathbf{p}_1, \mathbf{p}_2)$$
$$\mathbf{q}_2 = Lerp(t, \mathbf{p}_2, \mathbf{p}_3)$$

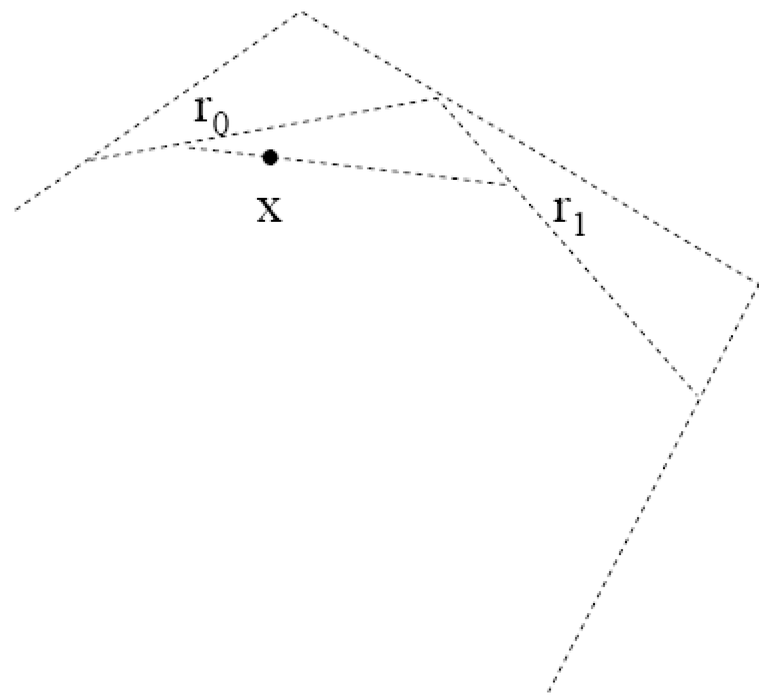


de Castlejau Algorithm

$$\mathbf{r}_0 = Lerp(t, \mathbf{q}_0, \mathbf{q}_1)$$
$$\mathbf{r}_1 = Lerp(t, \mathbf{q}_1, \mathbf{q}_2)$$



de Castlejau Algorithm



$$\mathbf{x} = Lerp(t, \mathbf{r}_0, \mathbf{r}_1)$$

Why more spline slides?

Bezier and Hermite splines have global influence

- One could create a Bezier curve that required 15 points to define the curve…
  - Moving any one control point would affect the entire curve
- Piecewise Bezier or Hermite don't suffer from this, but they don't enforce derivative continuity at join points

B-splines consist of curve segments whose polynomial coefficients depend on just a few control points

- Local control

Examples of Splines

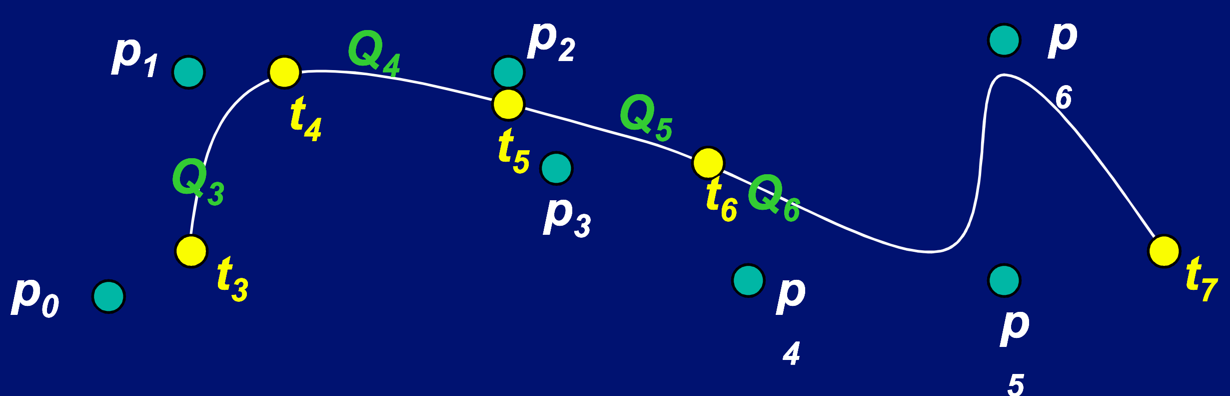
UNIVERSITY of VIRGINIA

B-Spline Curve (cubic periodic)

Start with a sequence of control points

Select four from middle of sequence** ($p_{i-2}, p_{i-1}, p_i, p_{i+1}$) **d

- Bezier and Hermite goes between p_{i-2} and p_{i+1}
- B-Spline doesn’t interpolate (touch) any of them but approximates going through p_{i-1} and p_i



UNIVERSITY of VIRGINIA

Uniform B-Splines

Approximating ***Splines***

Approximates n+1 control points

- $P_0, P_1, \ldots, P_n, n \geq 3$

Curve consists of n –2 cubic polynomial segments

- $Q_3, Q_4, \ldots Q_n$

t varies along B-spline as Q_i: $t_i <= t < t_{i+1}$

t_i (i = integer) are knot points that join segment Q_i to Q_{i+1}

Curve is uniform because knots are spaced at equal intervals of parameter, t

Uniform B-Splines

First curve segment, Q_3, is defined by first four control points

Last curve segment, Q_m, is defined by last four control points, P_{m-3}, P_{m-2}, P_{m-1}, P_m

Each control point affects four curve segments

B-spline Basis Matrix

Formulate 16 equations to solve the 16 unknowns

The 16 equations enforce the C_0, C_1, and C_2 continuity between adjoining segments, Q

$$M_{B-spline} = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

B-Spline

Points along B-Spline are computed just as with Bezier Curves

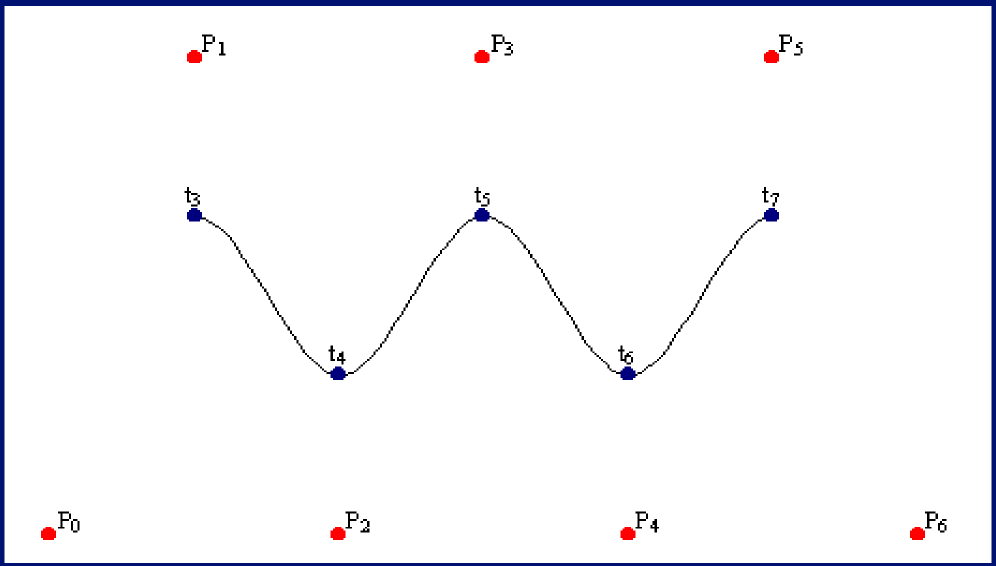
$$Q_i(t) = UM_{B-Spline}P$$

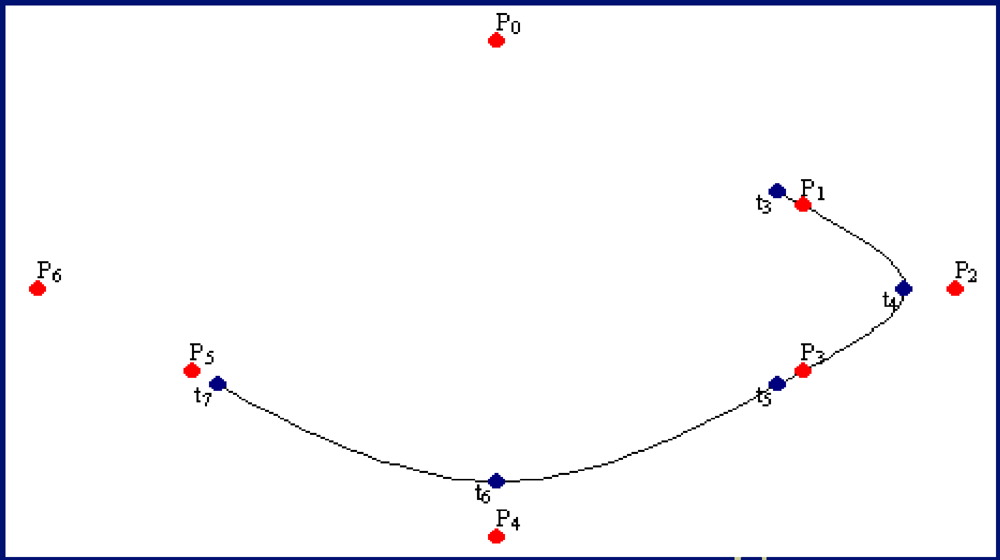
$$Q_i(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \end{bmatrix}$$

B-Spline

By far the most popular spline used

C_0, C_1, and C_2 continuous





Nonuniform, Rational B-Splines (NURBS)

The native geometry element in Maya

Models are composed of surfaces defined by NURBS, not polygons

NURBS are smooth

NURBS require effort to make non-smooth

Converting Between Splines

Consider two spline basis formulations for two spline types

$$P = T \times M_{spline_1} \times G_{spline_1}$$

$$P = T \times M_{spline_2} \times G_{spline_2}$$

$$T \times M_{spline_1} \times G_{spline_1} = T \times M_{spline_2} \times G_{spline_2}$$

Converting Between Splines

We can transform the control points from one spline basis to another

$$P = T \times M_{spline_1} \times G_{spline_1}$$

$$P = T \times M_{spline_2} \times G_{spline_2}$$

$$T \times M_{spline_1} \times G_{spline_1} = T \times M_{spline_2} \times G_{spline_2}$$

$$M_{spline_1} \times G_{spline_1} = M_{spline_2} \times G_{spline_2}$$

$$G_{spline_1} = M^{-1}_{spline_1} \times M_{spline_2} \times G_{spline_2}$$

Converting Between Splines

With this conversion, we can convert a B-Spline into a Bezier Spline

Bezier Splines are easy to render

Rendering Splines

Horner's Method

Incremental (Forward Difference) Method

Subdivision Methods

Horner's Method

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$x(t) = [(a_x t + b_x)t + c_x]t + d_x$$

Three multiplications

Three additions

Forward Difference

$$x_{k+1} = x_k + \Delta x_k$$

$$x_k = a_x t^3 + b_x t^2 + c_x t + d$$

$$x_{k+1} = a_x (t_k + \delta)^3 + b_x (t_k + \delta)^2 + c_x (t_k + \delta) + d_x$$

$$x_{k+1} - x_k = \Delta x_k = 3 a_x \delta t_k^{\,2} + (3 a_x \delta^2 + 2 b_x \delta) t_k + (a_x \delta^3 + b_x \delta^2 + c_x \delta)$$

But this still is expensive to compute

- Solve for change at k (Δ_k) and change at k+1 (Δ_{k+1})
- Boot strap with initial values for x_0, Δ_0, and Δ_1
- Compute x_3 by adding $x_0 + \Delta_0 + \Delta_1$

University of Virginia

Subdivision Methods

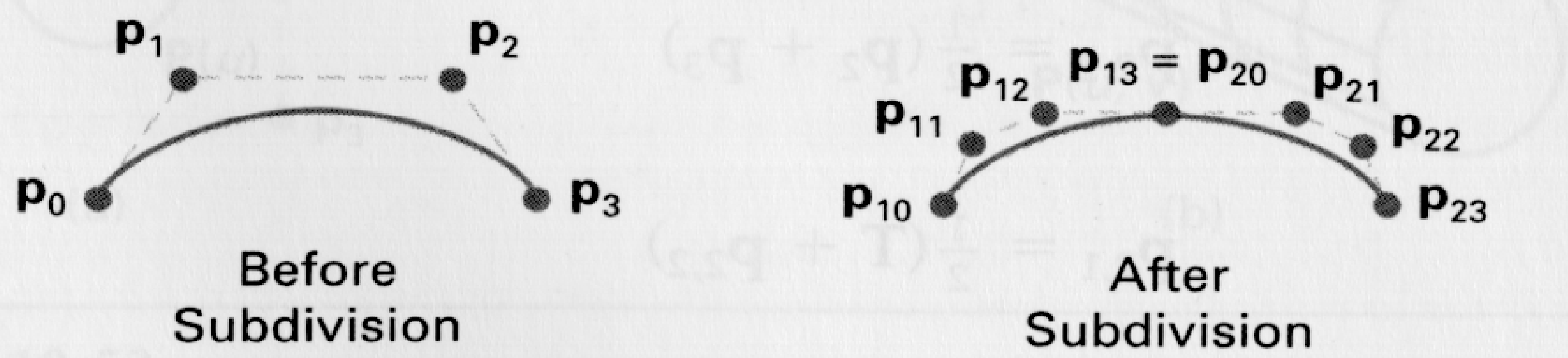


Figure 10-52
Subdividing a cubic Bézier curve section into two sections, each with four control points.

Bezier

Rendering Bezier Spline

```
public void spline(ControlPoint p0, ControlPoint p1,
                   ControlPoint p2, ControlPoint p3, int pix) {
  float len = ControlPoint.dist(p0,p1) + ControlPoint.dist(p1,p2)
              + ControlPoint.dist(p2,p3);
  float chord = ControlPoint.dist(p0,p3);
  if (Math.abs(len - chord) < 0.25f) return;
  fatPixel(pix, p0.x, p0.y);
  ControlPoint p11 = ControlPoint.midpoint(p0, p1);
  ControlPoint tmp = ControlPoint.midpoint(p1, p2);
  ControlPoint p12 = ControlPoint.midpoint(p11, tmp);
  ControlPoint p22 = ControlPoint.midpoint(p2, p3);
  ControlPoint p21 = ControlPoint.midpoint(p22, tmp);
  ControlPoint p20 = ControlPoint.midpoint(p12, p21);
  spline(p20, p12, p11, p0, pix);
  spline(p3, p22, p21, p20, pix);
}
```

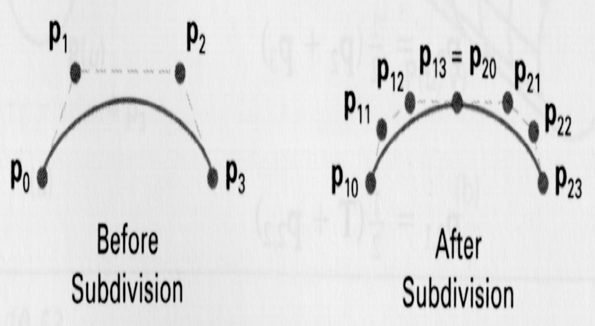


Figure 10-52
Subdividing a cubic Bézier curve section into two sections, each with four control points.

Orientation Representation and Interpolation

Parent:
Chapter 2.2, 3.3

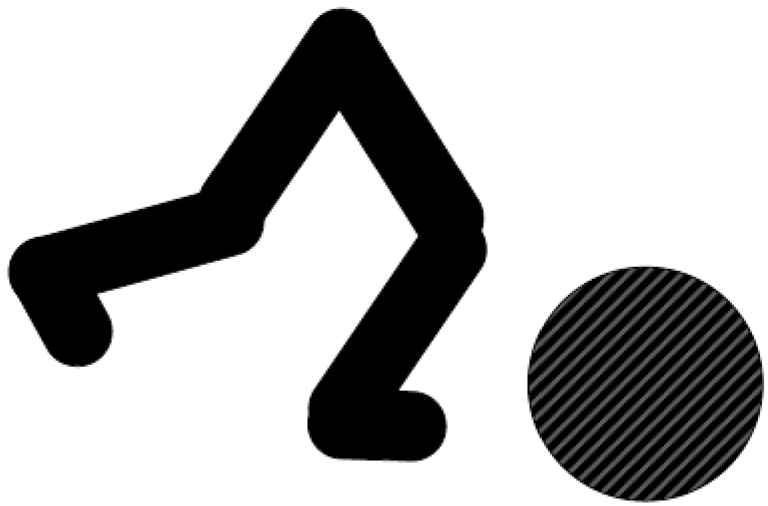
COMPUTER ANIMATION

15-497/15-861

02/22/02

Keyframing

- Last Class: how to interpolate positions/translations
- But we also need to orient things in 3D



Transformations (Review)

- Translation, scaling, and rotation:

$P' = T+P$	Translation
$P' = SP$	Scaling
$P' = RP$	Rotation

- treat all transformations the same so that they can be easily combined (streamline software and hardware)
- P is a point of the model
- Transformation is for animation, viewing, modeling
- P' is where it should be drawn

Homogenous Coordinates

- In graphics, we use homogenous coordinates for transformations
- 4x4 matrix can be used to represent translation, rotation, scaling, and other transformations
- We're dealing with 3-space, so the 4$^{\text{th}}$ coordinate is typically 1

$$\left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, w\right) = [x, y, z, w] \qquad (x, y, z) = [x, y, z, 1]$$

Translation

$$\begin{bmatrix} x+t_x \\ y+t_y \\ z+t_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

new point in space

transformation matrix

point in space

Scaling

$$\begin{bmatrix} xS_x \\ yS_y \\ zS_z \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation

In the upper left 3x3 submatrix

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

X axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Y axis

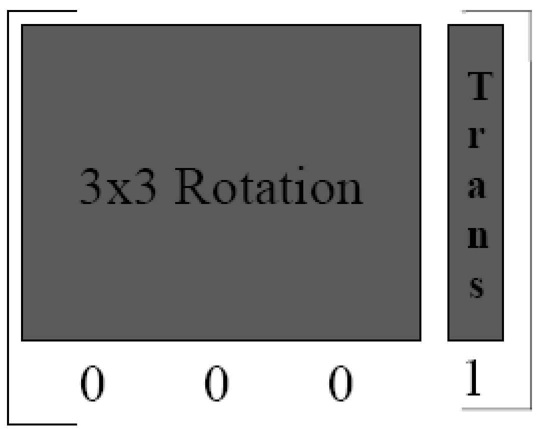
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Z axis

Composite Transformations

- We can now treat transformations as a series of matrix multiplications

$$P' = M_1 M_2 M_3 M_4 M_5 M_6 P$$
$$M = M_1 M_2 M_3 M_4 M_5 M_6$$
$$P' = MP$$



Back to Keyframing…

- In order to “move things” we need both translations and rotations
- Interpolating the translations was easy but what about rotations?

Interpolating Rotations

The upper left 3x3 submatrix of a transformation matrix is the rotation matrix

Maybe we can just interpolate the entries of that matrix to get the inbetween rotations?

Problem:

- Rows and columns are orthonormal (unit length and perpendicular to each other)
- Linear interpolation doesn't maintain this property, leading to nonsense for the inbetween rotations

Interpolating Rotation

Example:

- interpolate linearly from a positive 90 degree rotation about y to a negative 90 degree rotation about y

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Linearly interpolate each component and halfway between, you get this…

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

No longer a rotation matrix---not orthonormal
Makes no sense!

Orientation Representations

Direct interpolation of transformation matrices is not acceptable…

Where does that leave us?

How best do we represent orientations of an object and interpolate orientation to produce motion over time?

- Rotation Matrices
- Fixed Angle
- Euler Angle
- Axis Angle
- Quaternions

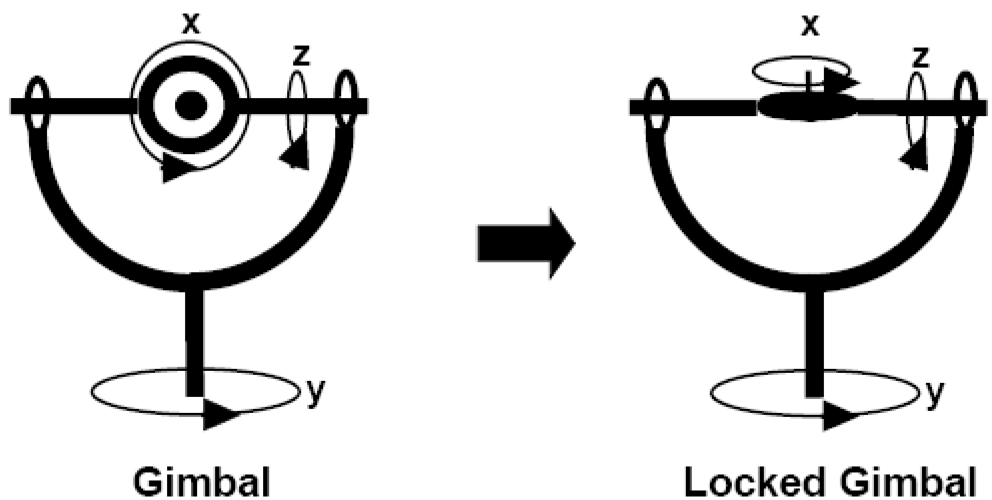
Fixed Angle Representation

- Angles used to rotate about fixed axes
- Orientations are specified by a set of 3 ordered parameters that represent 3 ordered rotations about fixed axes, i.e. first about x, then y, then z
- Many possible orderings, don't have to use all 3 axes, but can't do the same axis back to back

Fixed Angle

- A rotation of 10,45, 90 would be written as
  - Rz(90) Ry(45), Rx(10) since we want to first rotate about x, y, z. It would be applied then to the point P…. RzRyRx P
- Problem occurs when two of the axes of rotation line up. Gimbal Lock

Gimbal Lock

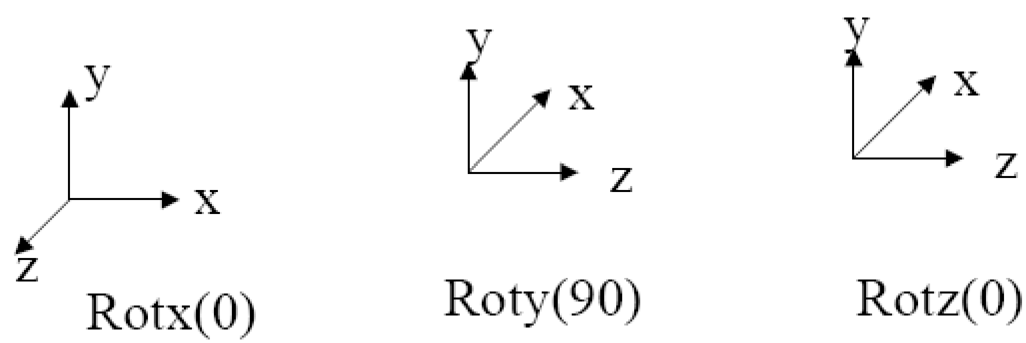


A *Gimbal* is a hardware implementation of Euler angles (used for mounting gyroscopes, expensive globes)

Gimbal lock is a basic problem with representing 3-D rotations using Euler angles or fixed angles

Gimbal Lock—Shown another way

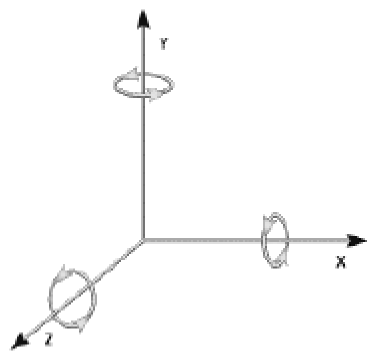
- A 90 degree rotation about the y axis aligns the first axis of rotation with the third.



- Incremental changes in x,z produce the same results
  - lost a degree of freedom

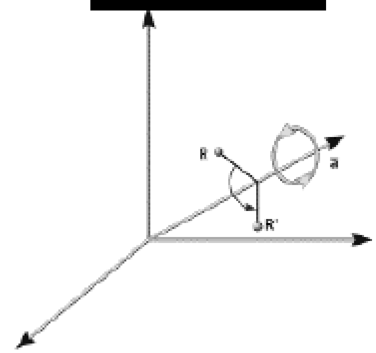
Interpolating Rotations

[redacted]



Euler angles

[redacted]



Axis-angle

Q: What kind of compound rotation do you get by successively turning about each of the 3 axes at a constant rate?

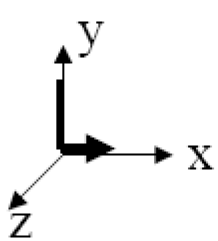
A: Not the one you want

Example

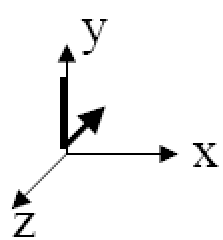
- Especially a problem if interpolating say…

(0,90,0) ⟶ (90, 45, 90)

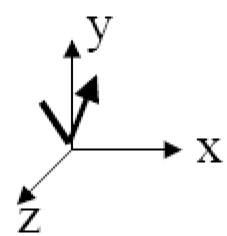
Just a 45 degree rotation from one orientation to the next, so we expect 90, 22.5, 90, but get 45, 67.5, 45



Initial Orientation
(object space)



(0,90,0)



(90, 45, 90)

Euler Angles

- Same as fixed axis, except now, the axes move with the object
- roll, pitch, yaw of an aircraft
- Euler Angle rotations about moving axes written in reverse order are the same as the fixed axis rotations.

$$\underbrace{R_x(\alpha)R_y(\beta)R_z(\gamma)}_{\text{Euler}}P = \underbrace{R_z(\gamma)R_y(\beta)R_x(\alpha)}_{\text{Fixed}}P$$

Same problem with Gimbal Lock

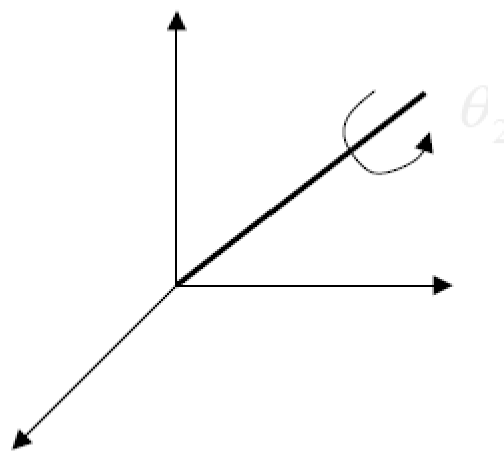
Axis Angle

Euler's Rotation Theorem:

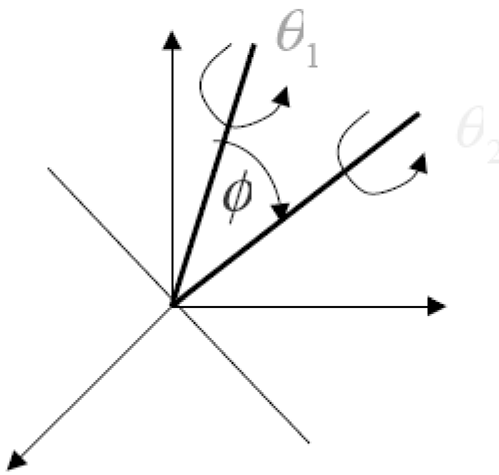
Any orientation can be represented by a 4-tuple

- angle, vector(x,y,z) where the angle is the amount to rotate by and the vector is the axis to rotate about

• Can interpolate the angle and axis separately



Axis Angle Interpolation



$B = A_1 \times A_2$

$$B = A_1 \times A_2$$

$$\phi = \cos^{-1}\left(\frac{A_1 \bullet A_2}{|A_1||A_2|}\right)$$

$$A_k = R_B(k\phi)\,A_1$$

$$\theta_1 = (1-k)\theta_1 + k\theta_2$$

Axis Angle

- Can interpolate the angle and axis separately
- No gimbal lock
- But, can't efficiently compose rotations…must convert to matrices first

Quaternions

- Good interpolation
- Can be multiplied (composed)
- No gimbal lock

Quaternions

- 4-tuple of real numbers
  - s,x,y,z or [s,v]
  - s is a scalar
  - v is a vector
- Same information as axis/angle but in a different form

$$q = Rot_{\theta(x,y,z)} = \left[\cos(\theta / 2), \sin(\theta / 2) \bullet (x, y, z)\right]$$

Quaternion Math

Addition:

$$[s_1, v_1] + [s_{2,} v_2] = [s_1 + s_2, v_1 + v_2]$$

Multiplication:

$$[s_1, v_1] \cdot [s_{2,} v_2] = [s_1 \cdot s_2 - v_1 \bullet v_{2,} s_1 \cdot v_2 + s_2 \cdot v_1 \times v_{2]}]$$

Multiplication is not commutative but is associative
(just like transformation matrices, as you would expect)

$$q_1 q_2 \neq q_2 q_1$$

$$(q_1 q_2) q_3 = q_1 (q_2 q_3)$$

Quaternion Math

A point in space is represented as $[0, v]$

$[1,(0,0,0)]$ multiplicative identity

$$q^{-1} = (1/\|q\|)^2 \cdot [s, -v]$$

$$where \quad \|q\| = \sqrt{s^2 + x^2 + y^2 + z^2}$$

$q \cdot q^{-1} = [1,(0,0,0)]$ the unit length quaternion (and multiplicative identity)

Quaternion Rotation

To rotate a vector, v using quaternions

–represent the vector as [0,v]

–represent the rotation as a quaternion, q

$$q = Rot_{\theta,(x,y,z)} = [\cos(\theta / 2), \sin(\theta / 2) \cdot (x, y, z)]$$

$$v' = Rot_q(v) = q \cdot v \cdot q^{-1}$$

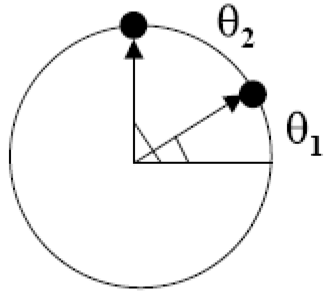
Can compose rotations as well

Looks good so far…we can easily specify and compose rotations!

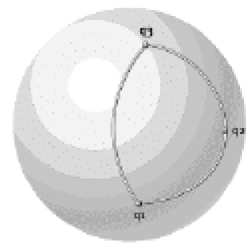
Quaternion Interpolation

- We can think of rotations as lying on an n-D unit sphere

[redacted]



1-angle (θ) rotation
(unit circle)



2-angle (θ-ϕ) rotation
(unit sphere)

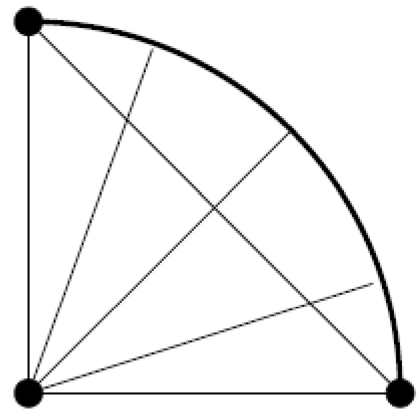
- Interpolating rotations means moving on n-D sphere
  - Can encode position on sphere by unit vector
  - How about 3-angle rotations?

Quaternion Interpolation

- Interpolating quaternions produces better results than Euler angles
- A quaternion is a point on the 4-D unit sphere
  - interpolating rotations requires a unit quaternion at each step - another point on the 4-D sphere
  - move with constant angular velocity along the great circle between the two points
  - Spherical Linear intERPolation (SLERPing)
- Any rotation is given by 2 quaternions, so pick the shortest SLERP
- To interpolate more than two points:
  - solve a non-linear variational constrained optimization (numerically)
- Further information: Ken Shoemake in the Siggraph '85 proceedings (*Computer Graphics,* V. 19, No. 3, P.245)

Quaternion Interpolation

- Direct linear interpolation does not work
  - Linearly interpolated intermediate points are not uniformly spaced when projected onto the circle



- Use a special interpolation technique
  - Spherical linear interpolation
  - viewed as interpolating over the surface of a sphere

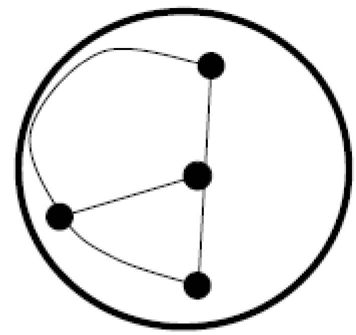
$$slerp(q1, q2, u)$$
$$= ((\sin((1-u)\cdot\theta))/(\sin\theta))\cdot q_1 + (\sin(u\cdot\theta))/(\sin\theta)\cdot q_2$$

- Normalize to regain unit quaternion

Two Representations of a Rotation

A quaternion and its negation [-s,-v] represent the same rotation:

$$
\begin{aligned}
-q &= Rot_{-\theta,-(x,y,z)} \\
&= [\cos(-\theta/2), \sin(-\theta/2)\cdot -(x,y,z)] \\
&= [\cos(\theta/2), -\sin(\theta/2)\cdot -(x,y,z)] \\
&= [\cos(\theta/2), \sin(\theta/2)\cdot (x,y,z)] \\
&= Rot_{\theta,(x,y,z)} \\
&= q
\end{aligned}
$$



Have to go the short way around!

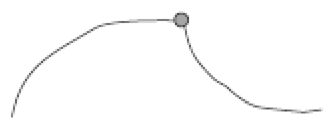
$$\cos(\theta) = q_1 \bullet q_2 = s_1 s_2 + v_1 \bullet v_2$$

$$\text{if } \cos(\theta) > 0 \Rightarrow q_1 \rightarrow q_2 \text{ shorter}$$

$$\text{else } q_1 \rightarrow -q_2 \text{ shorter}$$

Quaternion Interpolation

- As in linear interpolation in Euclidean space, we can have first order discontinuity



Solution is to formulate a cubic curve interpolation—see book for details

Quaternion Rotation

The rotation matrix corresponding to a quaternion,q, is

$$
\begin{aligned}
q &= Rot_{\theta,(x,y,z)} \\
&= [\cos(\theta/2), \sin(\theta/2)\cdot(x,y,z)] \\
&= [s,a,b,c]
\end{aligned}
$$

$$
\begin{bmatrix}
1-2b^2-2c & 2ab+2sc & 2ac-2sb \\
2ab-2sc & 1-2a^2-2c^2 & 2bc+2sc \\
2ac+2sb & 2bc-2sa & 1-2a^2-2b^2
\end{bmatrix}
$$

Rotations in Reality

- We can convert to/from any of these representations
  - but the mapping is not one-to-one
- Choose the best representation for the task
  - input: Euler angles
  - interpolation: quaternions
  - composing rotations: quaternions, orientation matrix
  - drawing: orientation matrix

Problems with Interpolation

- Splines don’t always do the right thing
- Classic problems
  - Important constraints may break between keyframes
    - » feet sink through the floor
    - » hands pass through walls
  - 3D rotations
    - » Euler angles don’t always interpolate in a natural way
- Solutions:
  - More keyframes!
  - Quaternions help fix rotation problems

Summary of Keyframing

- We know how to move points in 3D – translation and rotation
- So we can set keyframes – position, orientation
- We can describe interpolation methods – linear, cubic polynomial
- We can control interpolation speed with speed curves and arclength reparameterization

ควอเทอเนียน

- ควอเทอเนียนที่แทนการหมุนเป็นมุม θ รอบแกน (x,y,z) คือ

$$\left\langle \cos\frac{\theta}{2}; x\sin\frac{\theta}{2}, y\sin\frac{\theta}{2}, z\sin\frac{\theta}{2} \right\rangle$$

- ระวังว่า (x,y,z) ต้องเป็น**เวกเตอร์หนึ่งหน่วย**

ตัวอย่าง

- จงหาควอเทอเนียนที่แทนการหมุนเป็นมุม 60 องศารอบแกน (1,1,1)
  - เวกเตอร์หนึ่งหน่วยของแกนคือ
  - คำนวณค่า cos และ sin

$$(1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3})$$

$$\cos 30^\circ = \frac{\sqrt{3}}{2}, \sin 30^\circ = \frac{1}{2}$$

  - และจะได้ว่าควอเทอเนียนคือ

$$\left\langle \frac{\sqrt{3}}{2}; \frac{1}{2\sqrt{3}}, \frac{1}{2\sqrt{3}}, \frac{1}{2\sqrt{3}} \right\rangle$$

ตัวอย่าง

- ควอเทอเนียนต่อไปนี้แทนการหมุนกี่องศา รอบแกนอะไร?

$$\left\langle \frac{1}{2}; 0, \frac{\sqrt{6}}{4}, \frac{\sqrt{6}}{4} \right\rangle$$

  - เราได้ว่า $\cos\frac{\theta}{2} = \frac{1}{2} = \cos 60^\circ$
  - ฉะนั้น $\theta = 120^\circ$
  - แกนที่หมุนรอบคือ

$$\frac{1}{\sin 60^\circ}\left(0, \frac{\sqrt{6}}{4}, \frac{\sqrt{6}}{4}\right) = \frac{2}{\sqrt{3}}\left(0, \frac{\sqrt{6}}{4}, \frac{\sqrt{6}}{4}\right) = \left(0, \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right)$$

การคูณควอเทอเนียน

- **หลีกเลี่ยงการคูณควอเทอเนียนตรง ๆ**
- เพราะการคำนวณยุ่งยากและมีสิทธิ์ผิดมาก
- ใช้ความเข้าใจความหมายของควอเทอเนียนทำการคำนวณดีกว่า

ตัวอย่าง

- ให้

$$q_1 = \left\langle \frac{\sqrt{2}}{2}; \frac{3\sqrt{2}}{10}, 0, \frac{2\sqrt{2}}{5} \right\rangle$$

$$q_2 = \left\langle \frac{\sqrt{3}}{2}; \frac{3}{10}, 0, \frac{2}{5} \right\rangle$$

จงคำนวณ $q_1 q_2$

ตัวอย่าง

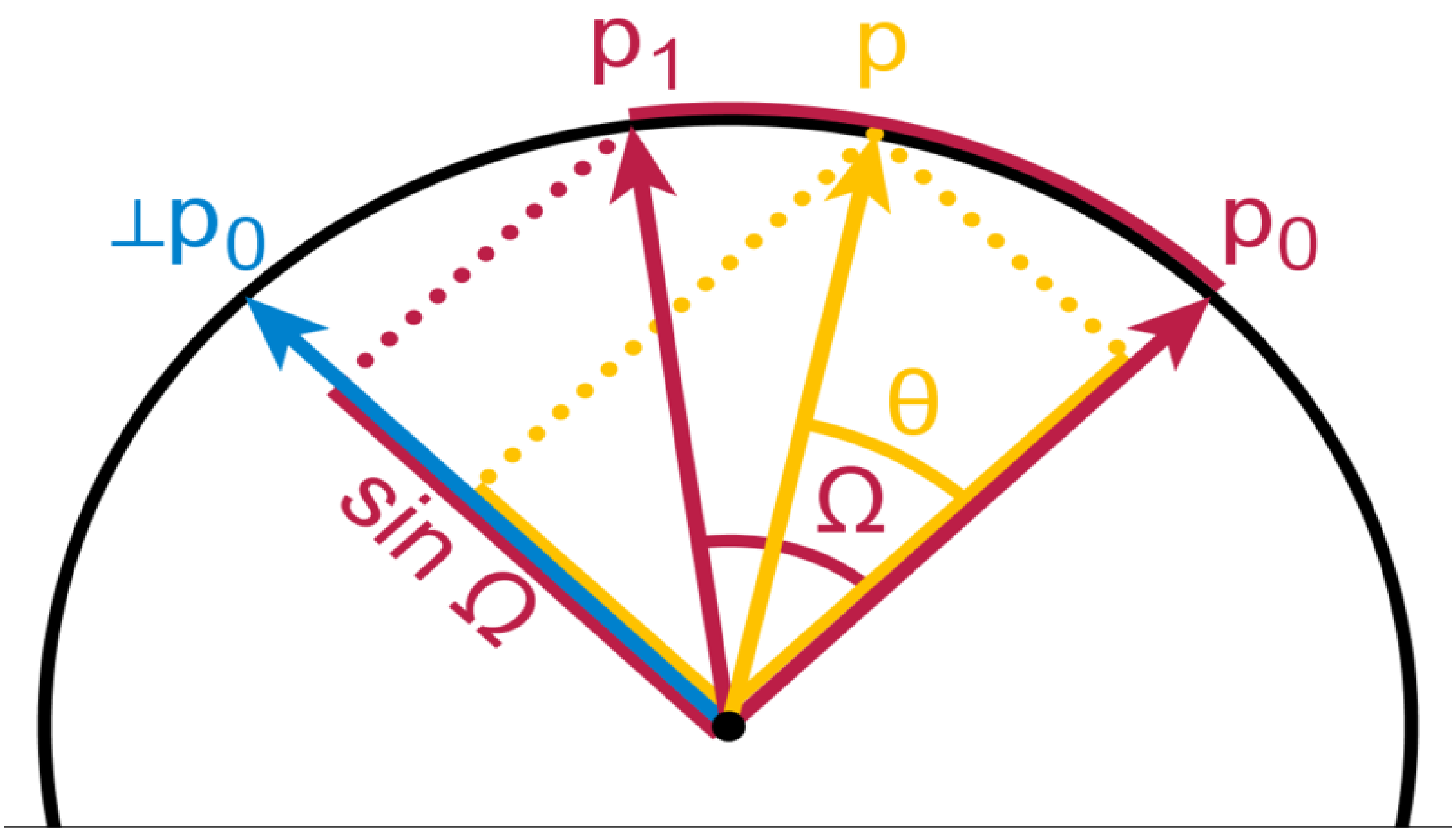
- q_1 คือการหมุนเป็นมุม 90 องศา รอบแกน (3/5, 0, 4/5)
- q_2 คือการหมุนเป็นมุม 60 องศา รอบแกน (3/5, 0, 4/5)
- ฉะนั้น q_1q_2 คือการหมุนเป็นมุม 60 องศาแล้วจึงหมุน 90 องศา
- รวมแล้วเป็นการหมุน 150 องศารอบแกน (3/5, 0, 4/5)
- ฉะนั้น

$$
\begin{aligned}
q_1q_2 &= \left\langle \cos 75^\circ; \frac{3}{5}\sin 75^\circ, 0, \frac{4}{5}\sin 75^\circ \right\rangle \\
&= \left\langle \frac{\sqrt{3}-1}{2\sqrt{2}}; \frac{3+3\sqrt{3}}{10\sqrt{2}}, 0, \frac{4+4\sqrt{3}}{10\sqrt{2}} \right\rangle
\end{aligned}
$$

Slerp

- อย่าคำนวณ slerp โดยตรงเช่นกัน
- สมมติว่าเราจะคำนวณ slerp(q_0,q_1,α) โดยให้ค่า α มีค่าเพิ่มขึ้นเรื่อยๆ จาก 0 ถึง 1 ถ้าเรา plot quaternion ค่าต่างๆ ที่เกิดขึ้น เราจะได้ว่ามันเรียงตัวกันเป็นเส้น geodesic ซึ่งคือเส้นบนทรงกลม 4 มิติที่สั้นที่สุดที่ผ่าน q_0 และ q_1
- ค่า α เป็นตัวบอกตำแหน่งบนเส้น geodesic นี้ กล่าวคือ
  - ถ้า $\alpha = 0$ จะอยู่ที่ q_0
  - ถ้า $\alpha = 1$ จะอยู่ที่ q_1
  - ถ้า $\alpha = 0.5$ จะอยู่ตรงกลางระหว่าง q_0 กับ q_1 พอดี
  - ฯลฯ

Slerp



ตัวอย่าง

- ให้

$$q_1 = \left\langle 1; 0, 0, 0 \right\rangle$$

$$q_2 = \left\langle 0; 0, 1, 0 \right\rangle$$

จงคำนวณ $\mathrm{slerp}(q_0, q_1, 1/3)$

ตัวอย่าง

- สังเกตว่า x component และ z component เป็น 0
- ดังนั้นที่ผลลัพธ์ x และ z ก็จะต้องมีค่าเป็น 0 ด้วย เนื่องจากเส้น geodesic จะไม่ผ่านบริเวณที่ x และ z ไม่เป็น 0 (ถ้าผ่านมันจะไม่สั้นสุด)
- ดังนั้นเราสามารถคิดว่าเส้น geodesic เป็นเส้นรอบวงของวงกลมใน 2 มิติ โดยที่แกนของระนาบสองมิตินั้นคือแกน w และแกน y

ตัวอย่าง

- มุมระหว่าง q_0 และ q_1 คือ 90 องศา
- slerp(q_0,q_1,1/3) คือตำแหน่งที่ทำมุมกับ q_0 เป็น 1/3 เท่าของมุม 90 องศา กล่าวคือทำมุม 30 องศากับ q_0
- ฉะนั้น slerp(q_0,q_1,1/3) จึงมีพิกัด (w,y) เท่ากับ $\left(\frac{\sqrt{3}}{2}, \frac{1}{2}\right)$
- กล่าวคือ

$$\mathrm{slerp}(q_0, q_1, 1/3) = \left\langle \frac{\sqrt{3}}{2}; 0, \frac{1}{2}, 0 \right\rangle$$

ตัวอย่าง