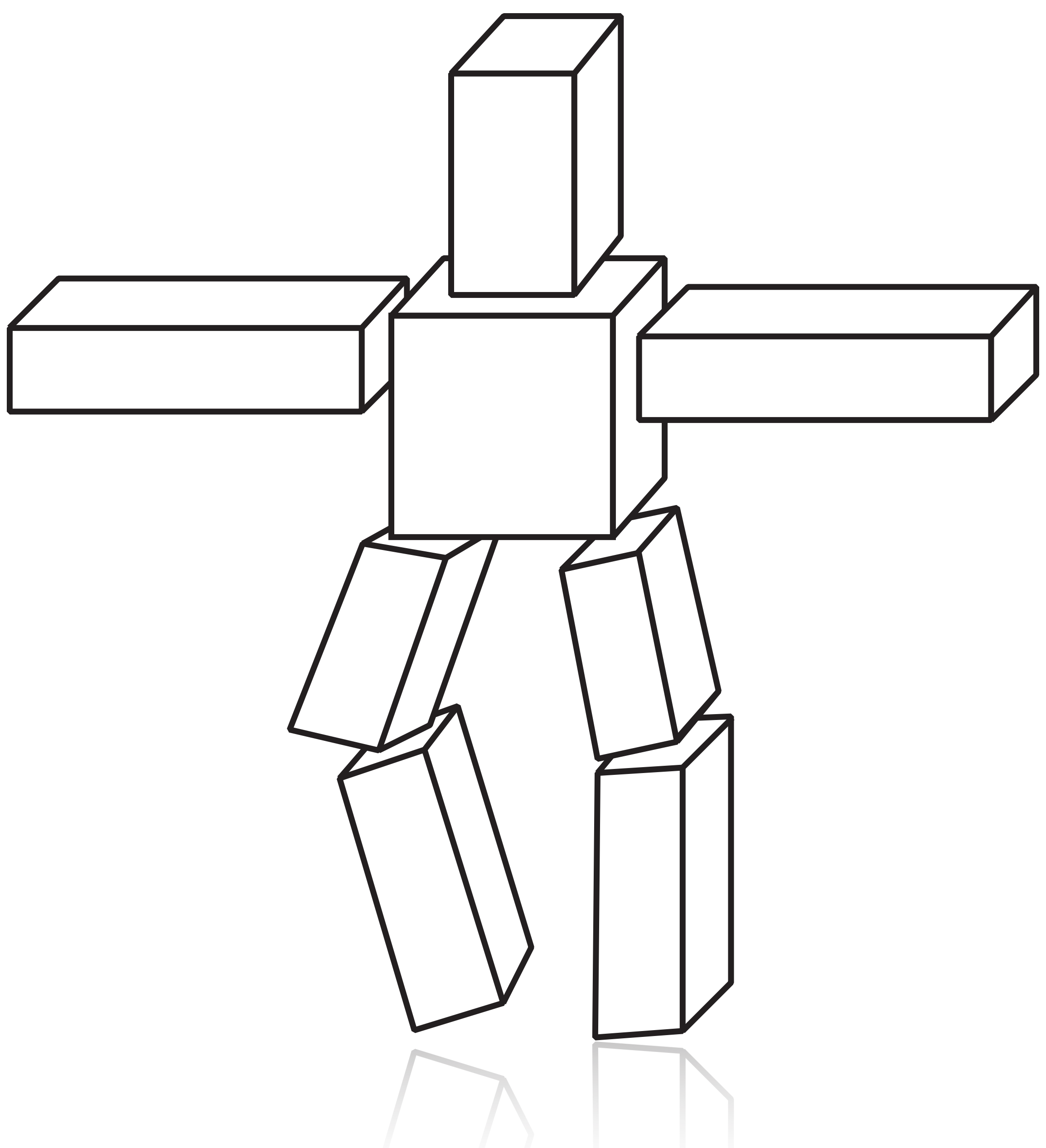
Lecture 12

Introduction to Animation

Interactive Computer Graphics
Stanford CS248, Spring 2018

Increasing the complexity of our world model

Transformations



Geometry

Materials, lighting, ...

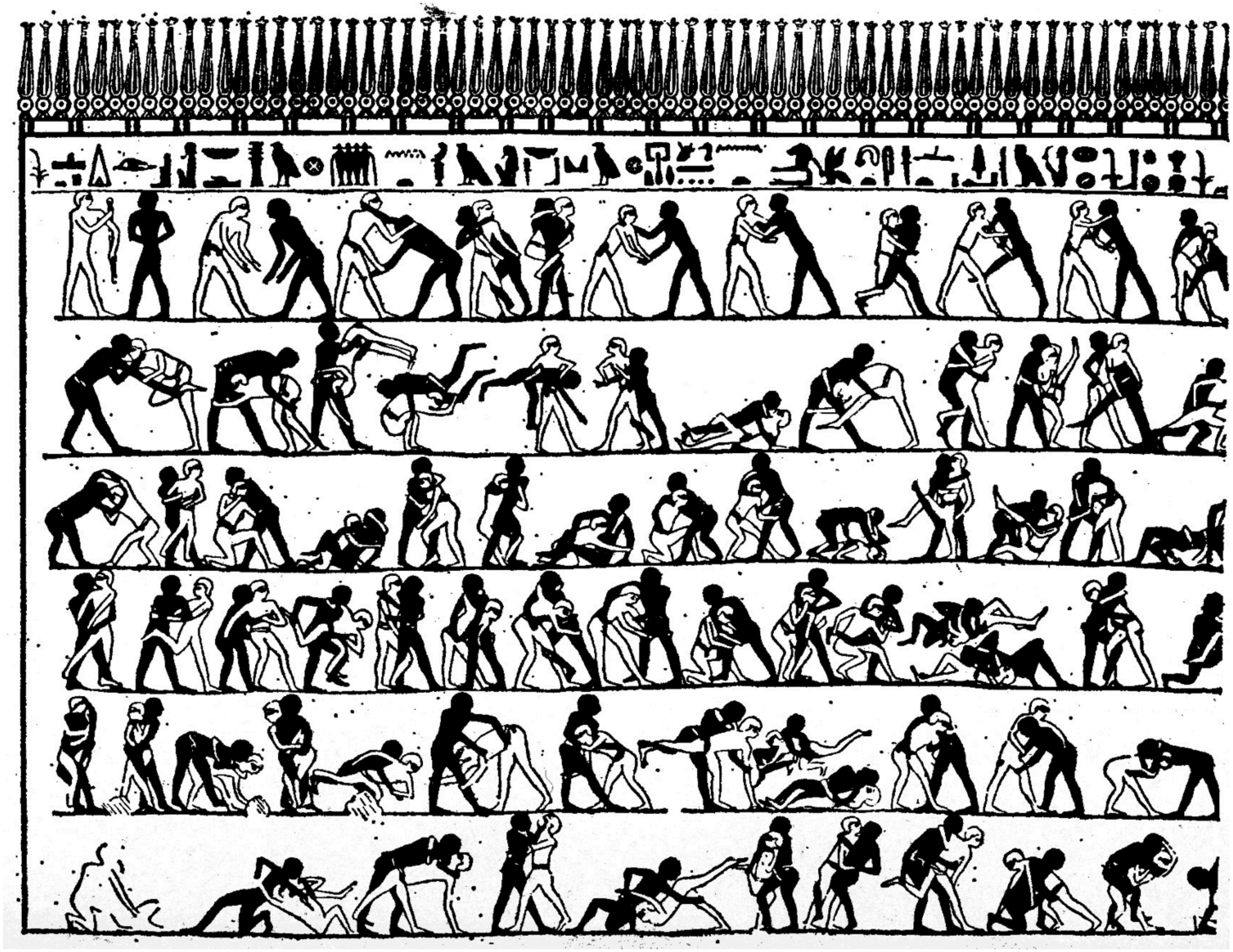
Increasing the complexity of our models

...but what about *motion?*

First animation

(Shahr-e Sukhteh, Iran 3200 BCE)

History of animation



(tomb of Khnumhotep, Egypt 2400 BCE)

History of animation

(Phenakistoscope, 1831)

First film

- Originally used as scientific tool rather than for entertainment
- Critical *technology* that accelerated development of animation

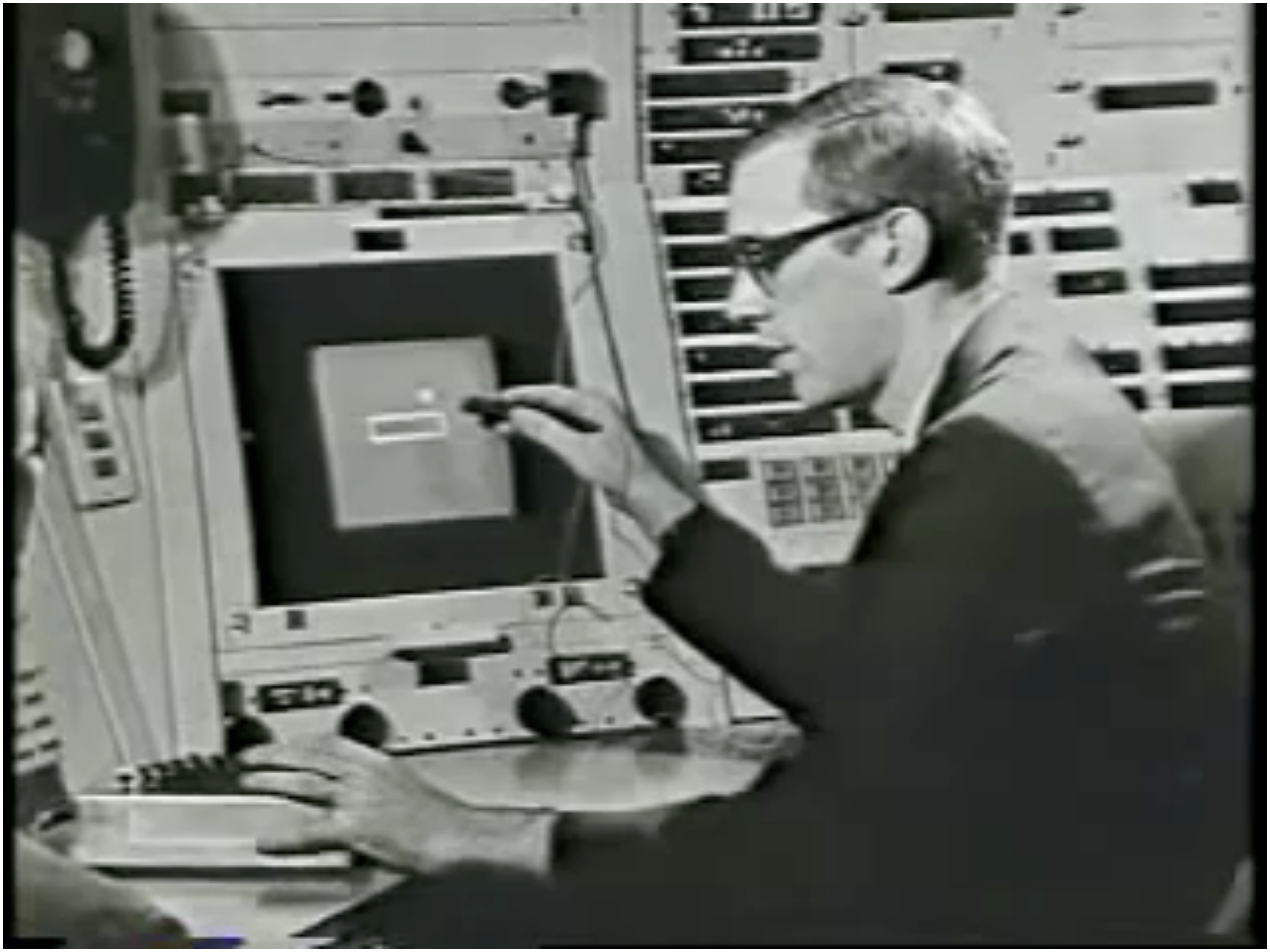
Eadweard Muybridge, *"Sallie Gardner"* (1878)

Interesting note: study commissioned by Leland Stanford (to determine if horse's feet ever off the ground)

First hand-drawn feature-length animation

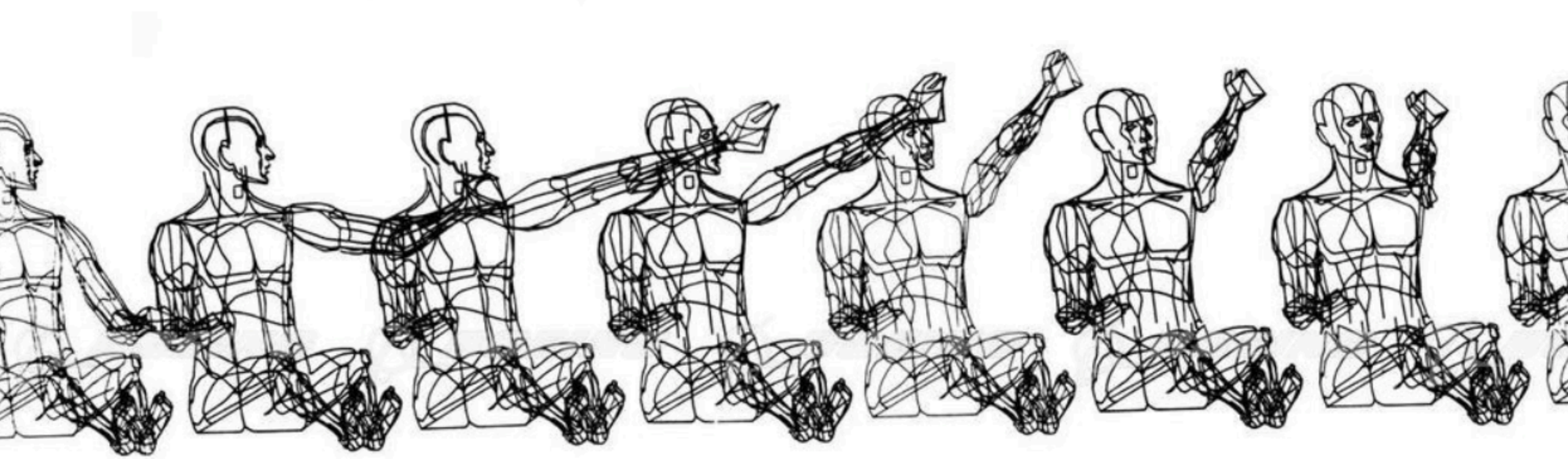
Disney, “Snow White and the Seven Dwarves” (1937)

First digital-computer-generated animation



Ivan Sutherland, "Sketchpad" (1963)

First 3D computer animation

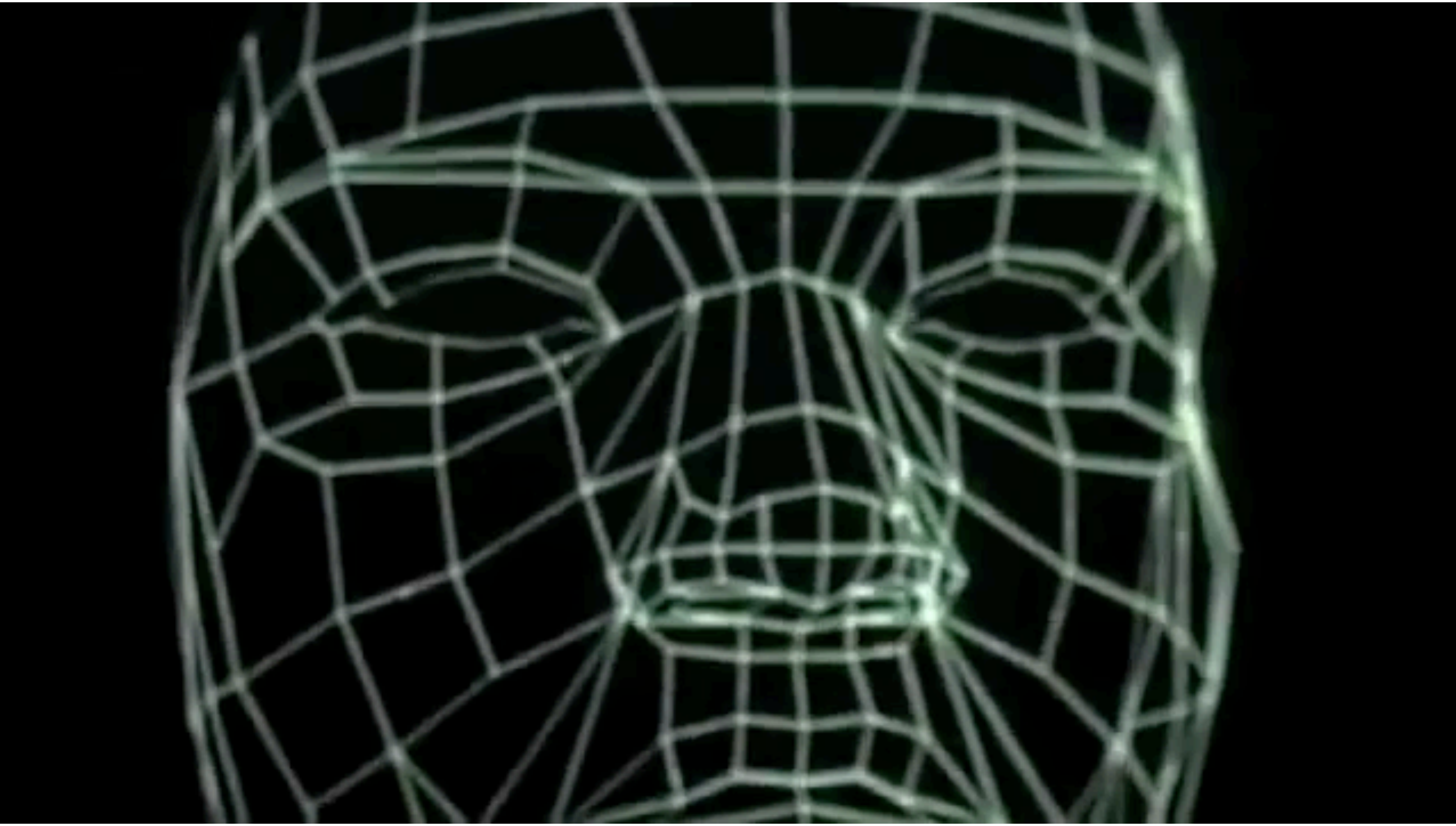


William Fetter, "Boeing Man" (1964)

Early computer animation

Nikolay Konstantinov, "Kitty" (1968)

Early computer animation



Ed Catmull & Fred Park, “Computer Animated Faces” (1972)

First *attempted* CG feature film

NYIT [Williams, Heckbert, Catmull, ...], "The Works" (1984)

First CG feature film

Pixar, "Toy Story" (1995)

Computer animation - present day

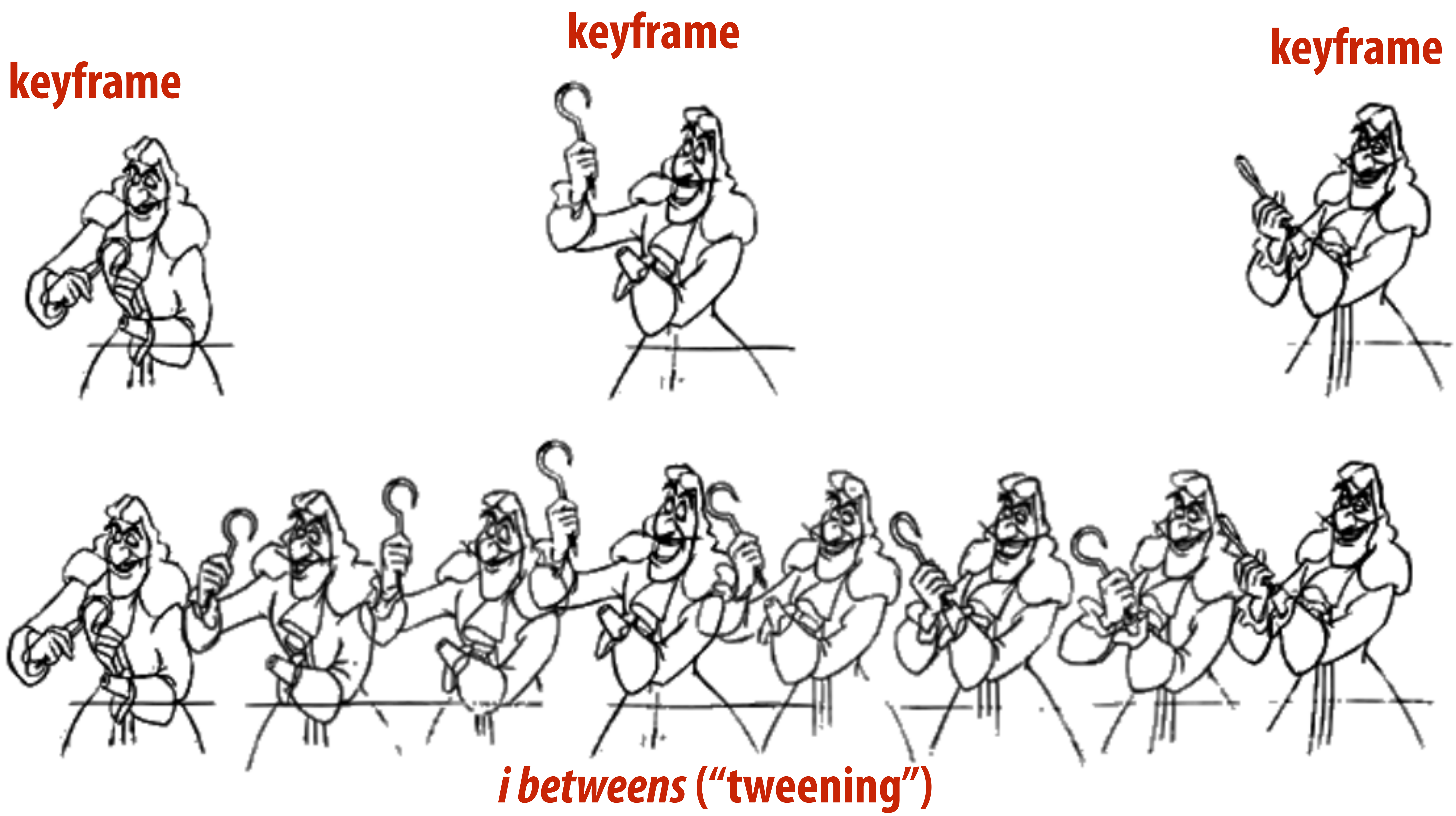
Notice combination of character animation, camera animation, and physical simulation in this clip.

Pixar's Coco (2017)

https://www.youtube.com/watch?v=GvicFasn_yM&t=4m5s

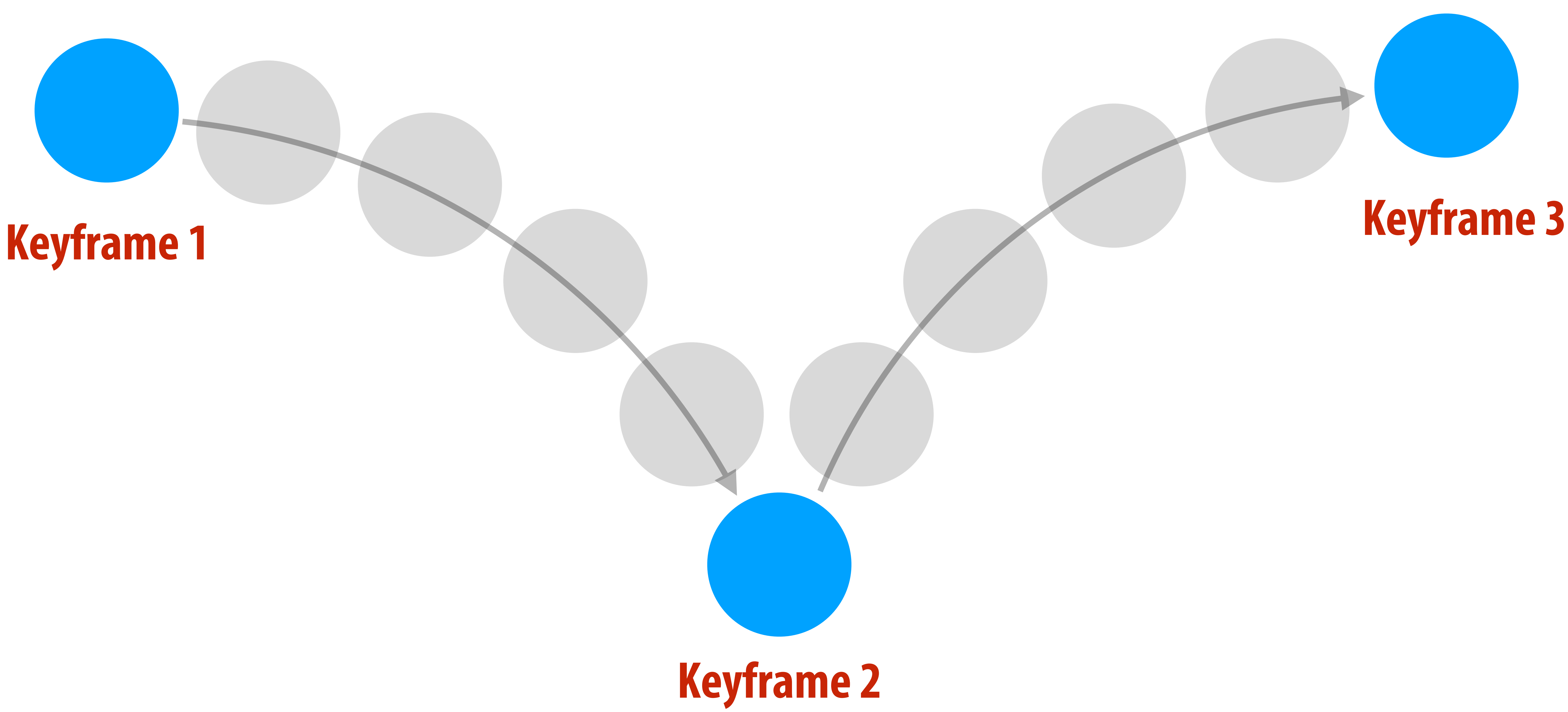
Generating motion (hand-drawn)

- Senior artist draws *keyframes*
- Assistant draws *inbetweens*
- Tedious / labor intensive (opportunity for technology!)



Keyframing

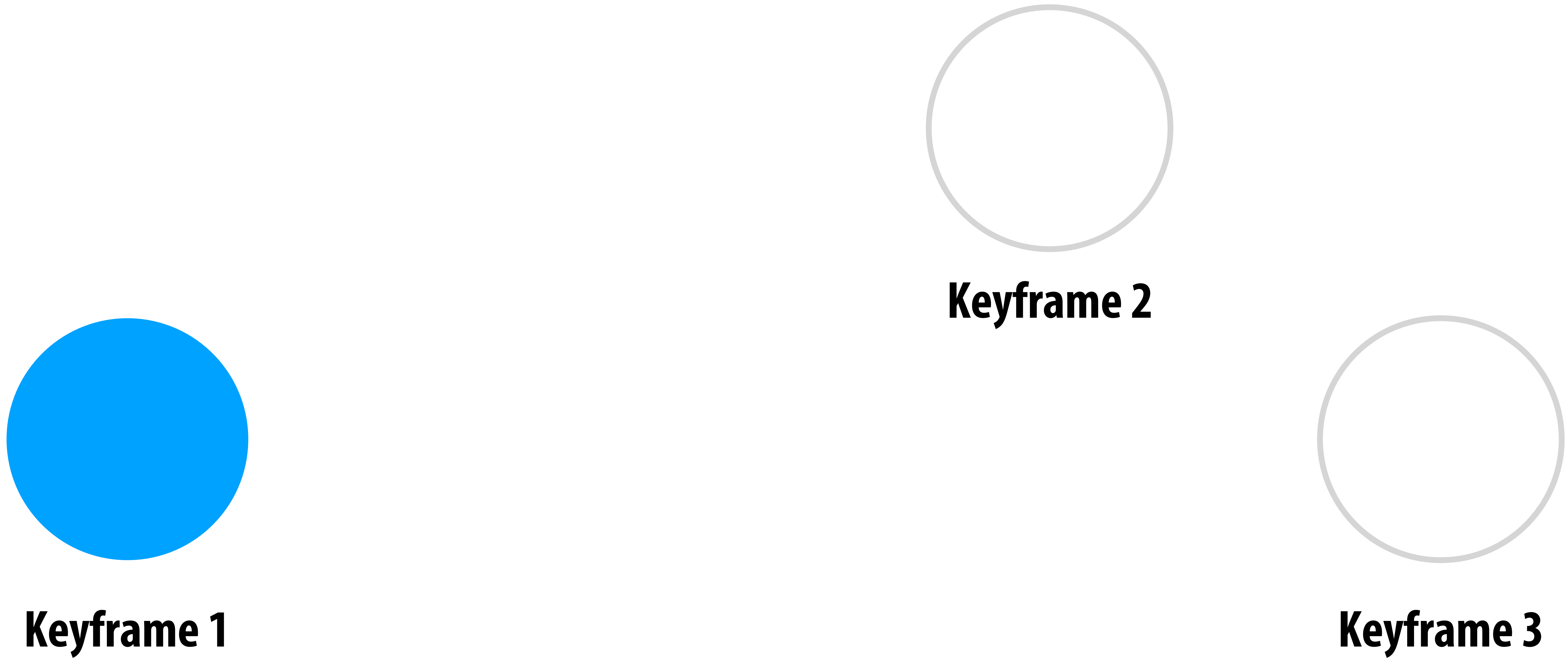
- Basic idea:
  - Animator specifies important events only
  - Computer fills in the rest via interpolation/approximation
- "Events" don't have to be position
- Could be color, light intensity, camera zoom, ...



Keyframing example

Keyframing example

Keyframing example

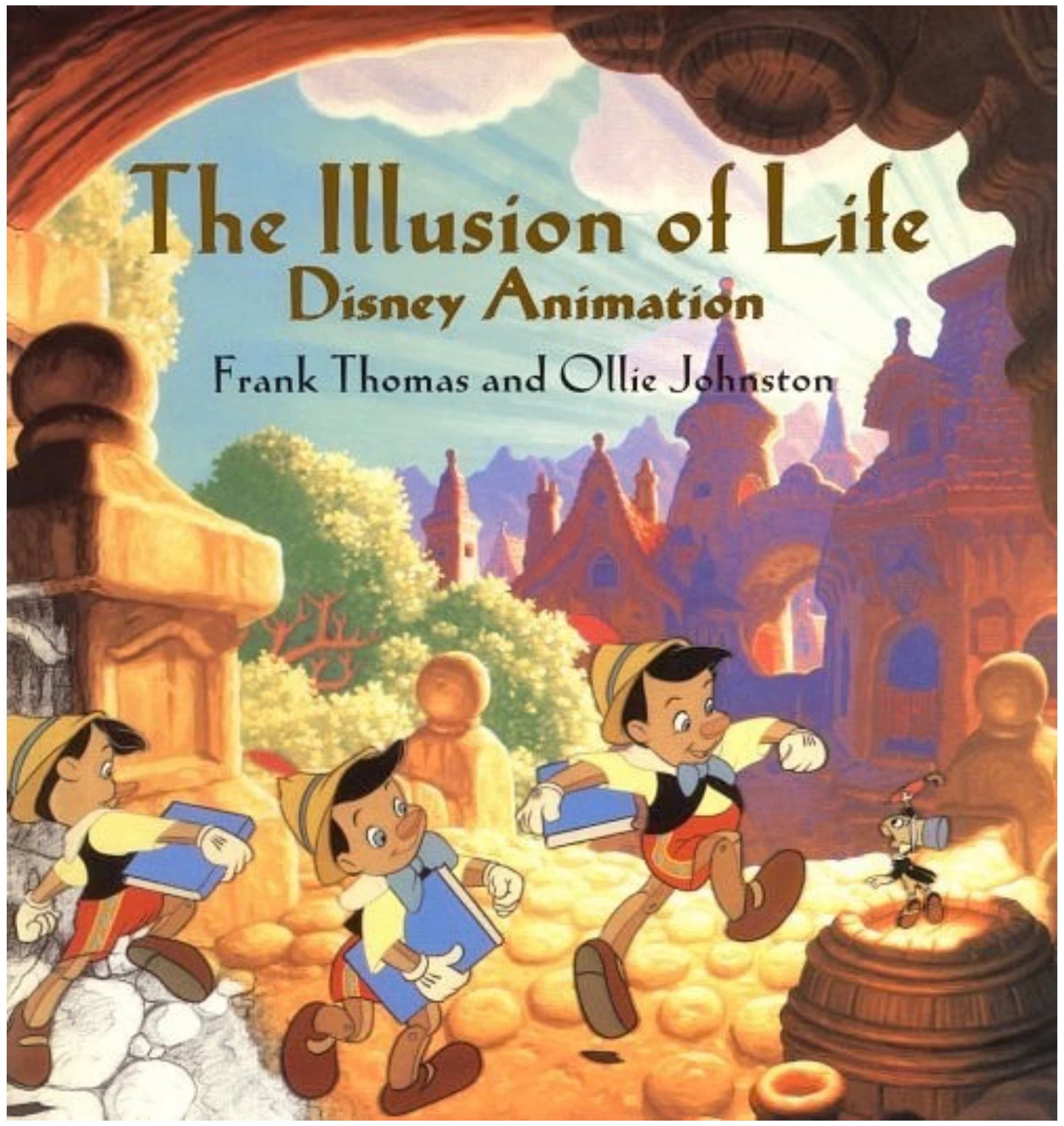


Principles of animation

Slide sequence credit Mark Pauly, Ren Ng

Animation principles

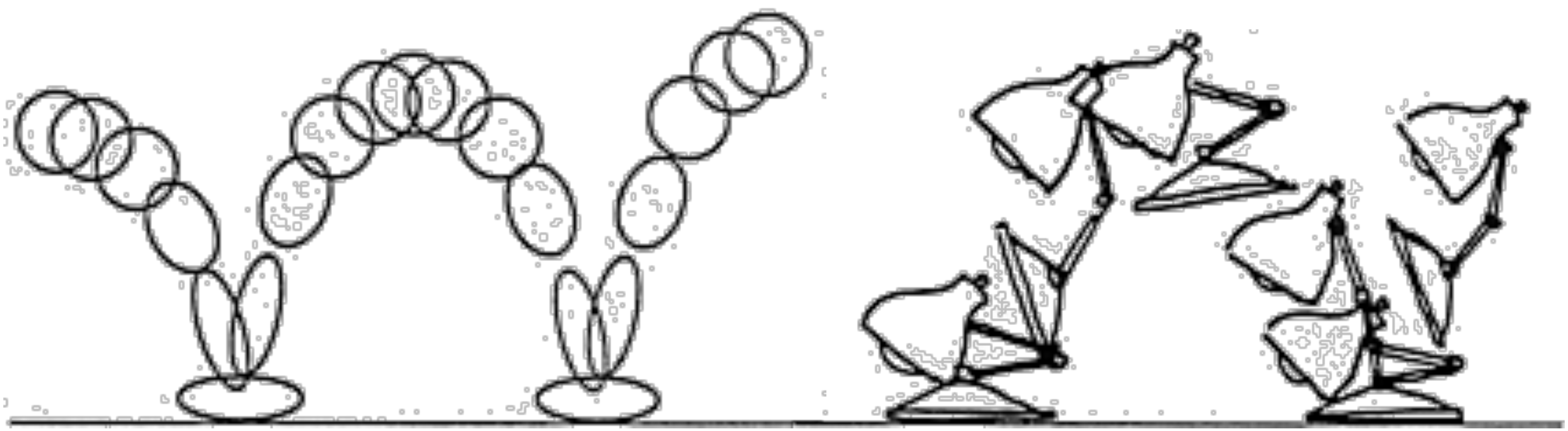
- From
  - "Principles of Traditional Animation Applied to 3D Computer Animation" - John Lasseter, ACM Computer Graphics, 21(4), 1987
- In turn from
  - "The Illusion of Life"
    Frank Thomas and Ollie Johnson
- Same for 2D and 3D



http://www.siggraph.org/education/materials/HyperGraph/animation/character_animation/principles/prin_trad_anim.htm

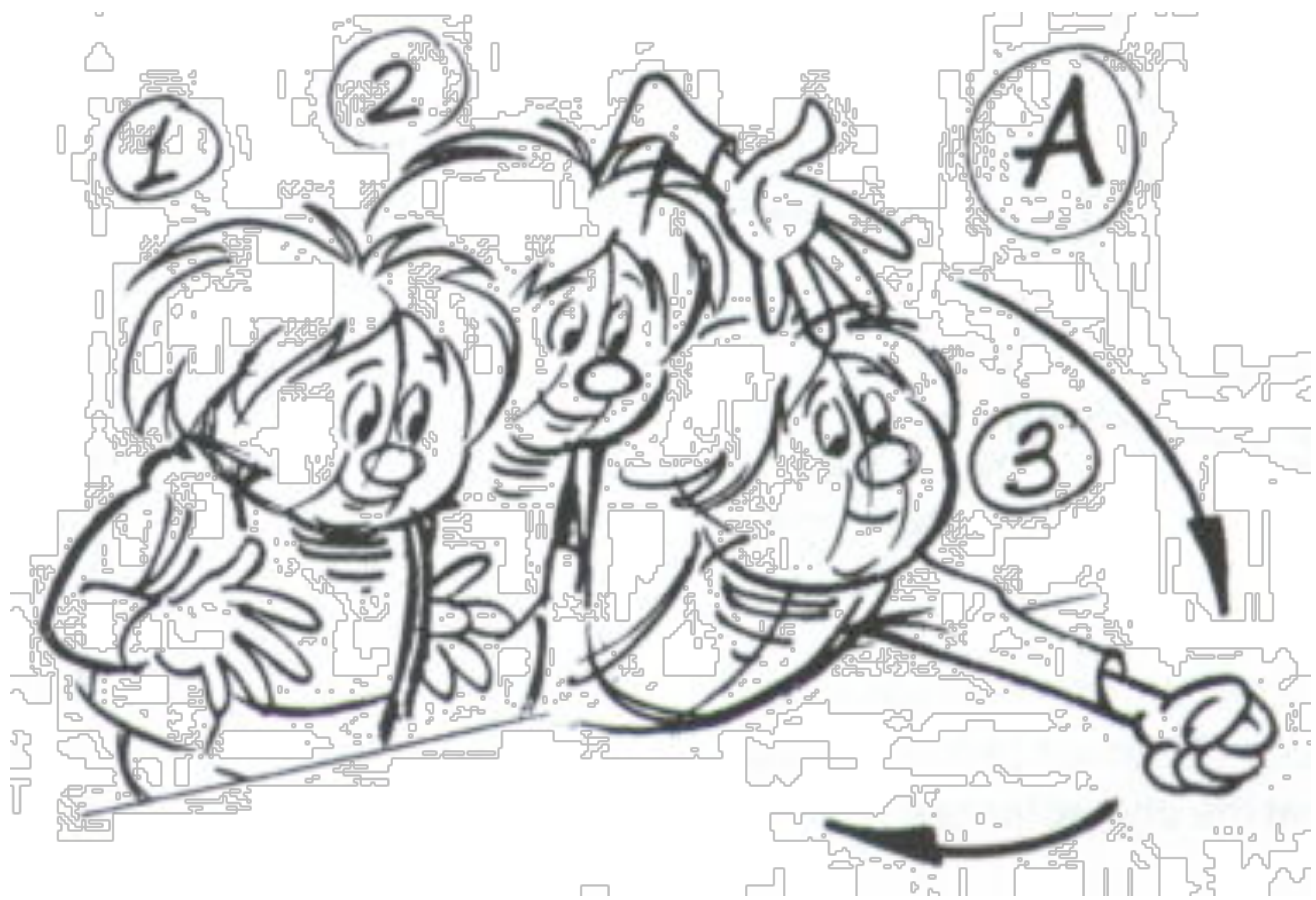
Squash and stretch

- Refers to defining the rigidity and mass of an object by distorting its shape during an action.
- Shape of object changes during movement, but not its volume.



Anticipation

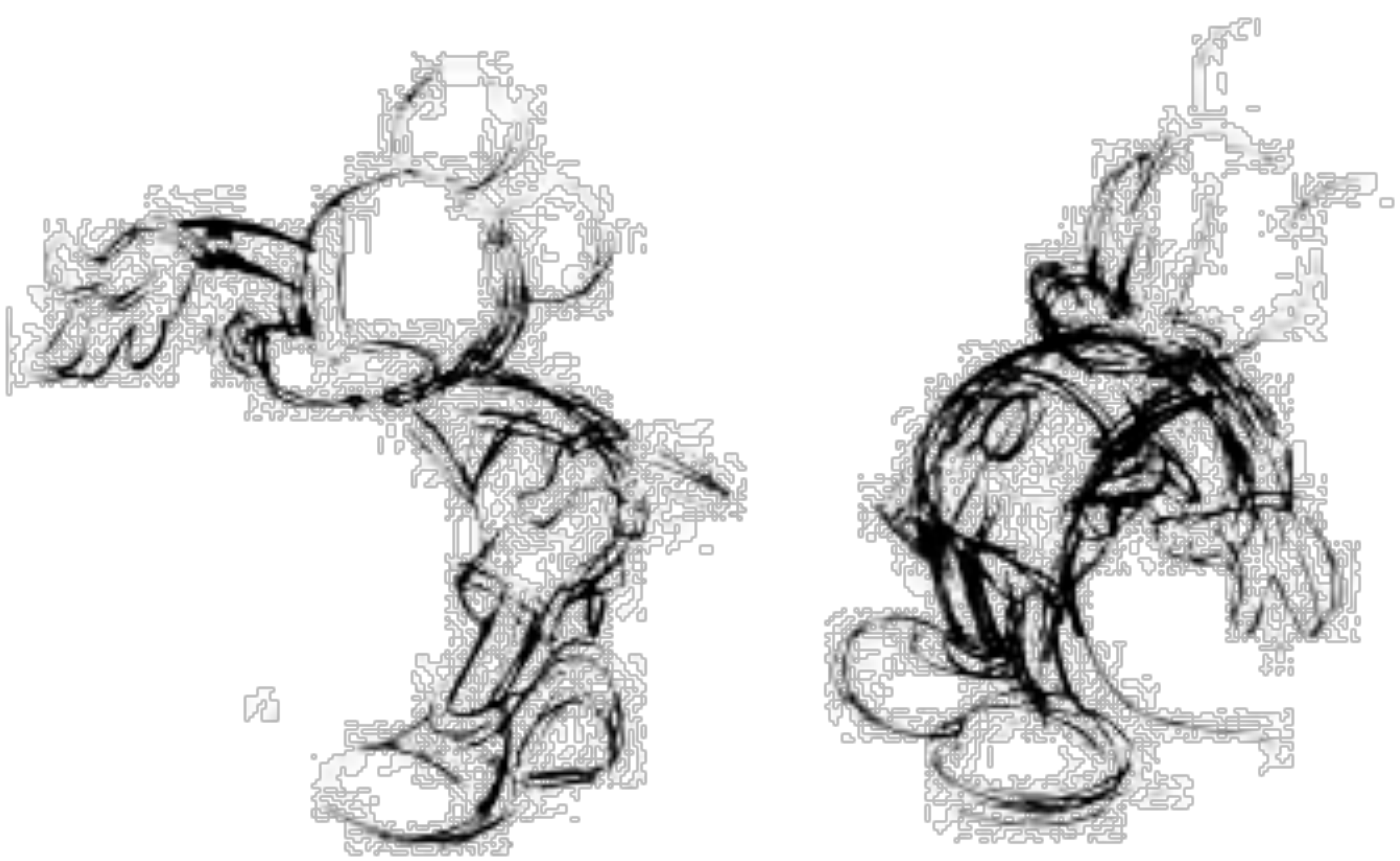
- Prepare for each movement
- For physical realism
- To direct audience's attention



Timing for Animation, Whitaker & Halas

Staging

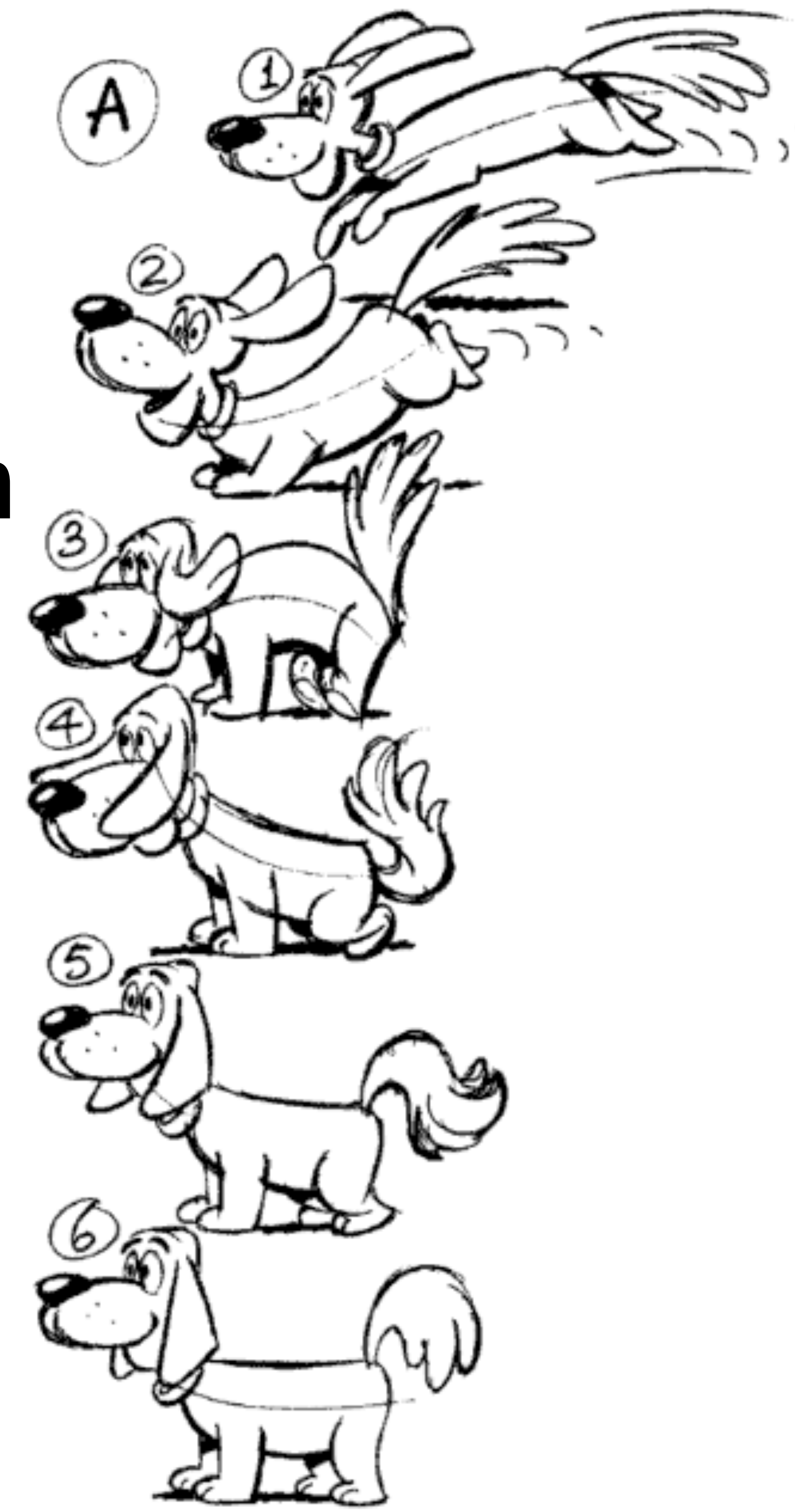
- Picture is 2D
- Make situation clear
- Audience looking in right place
- Action clear in silhouette



Disney Animation: The Illusion of Life

Follow through

- Overlapping motion
- Motion doesn't stop suddenly
- Pieces continue at different rates
- One motion starts while previous is finishin
  animation smooth

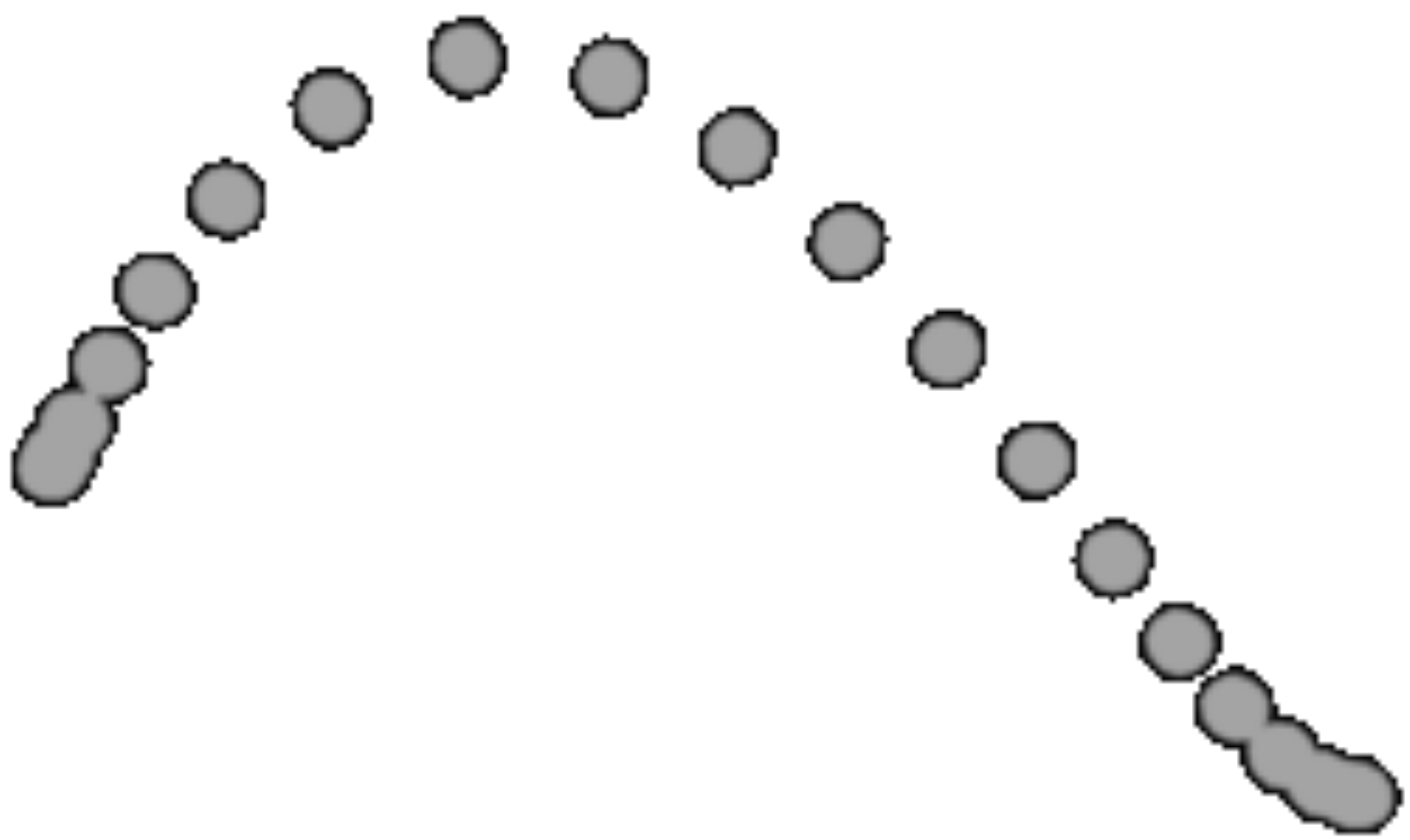


Timing for Animation, Whitaker & Halas

Ease-in and ease-out

Movement doesn't start and stop abruptly

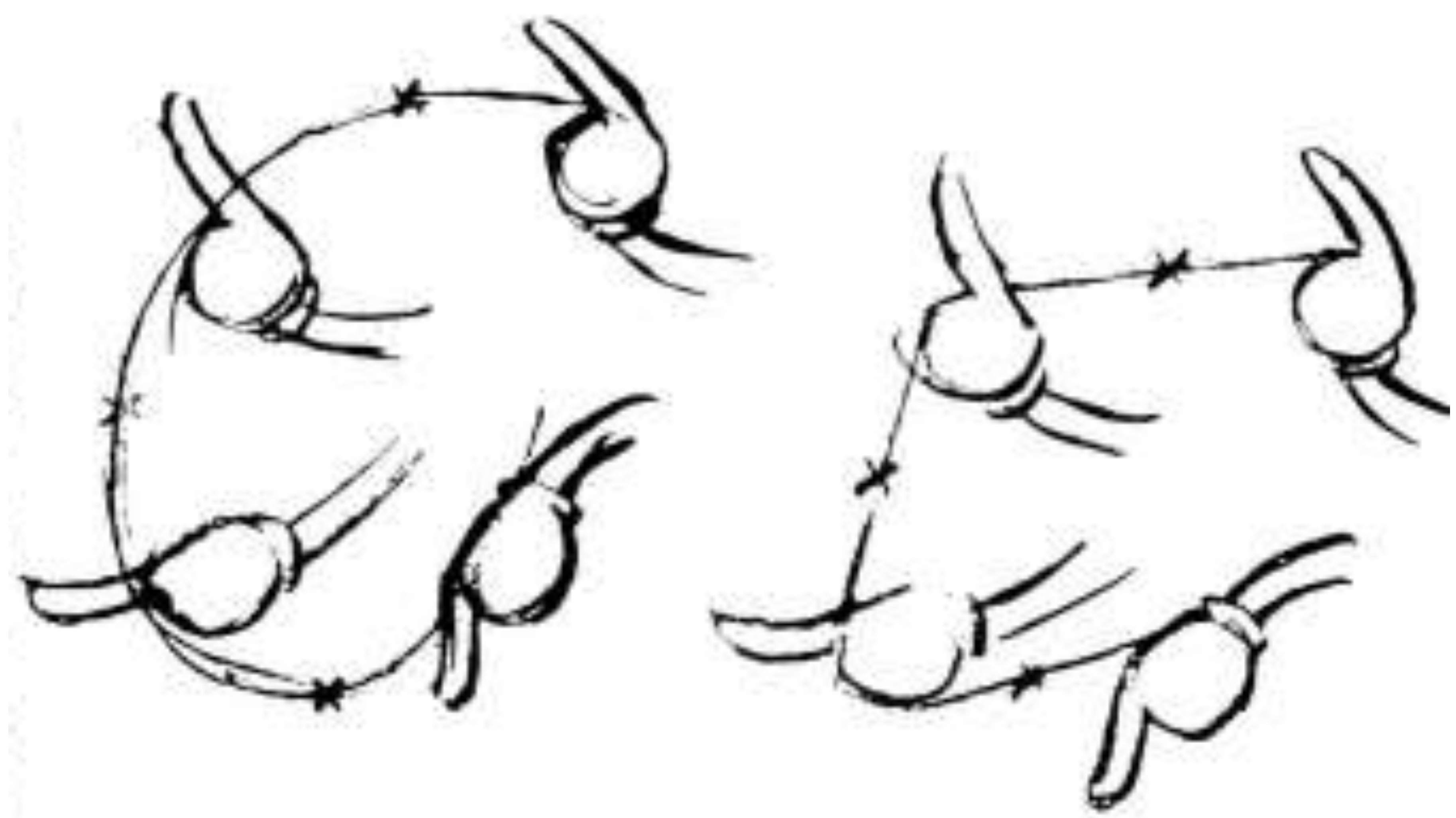
Also contributes to weight and emotion



Arcs

Move in curves, not in straight lines

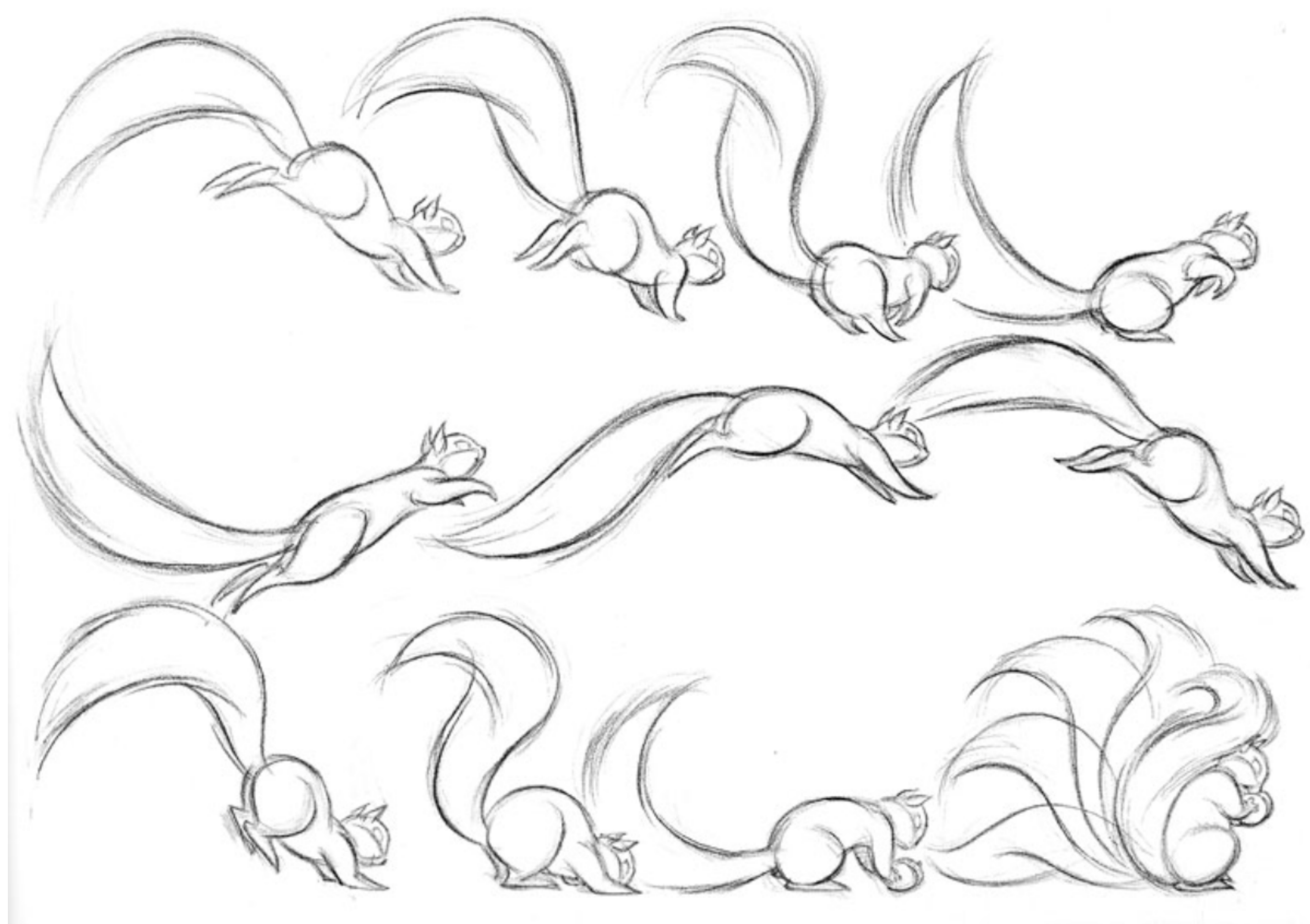
This is how living creatures move



Disney Animation: The Illusion of Life

Secondary action

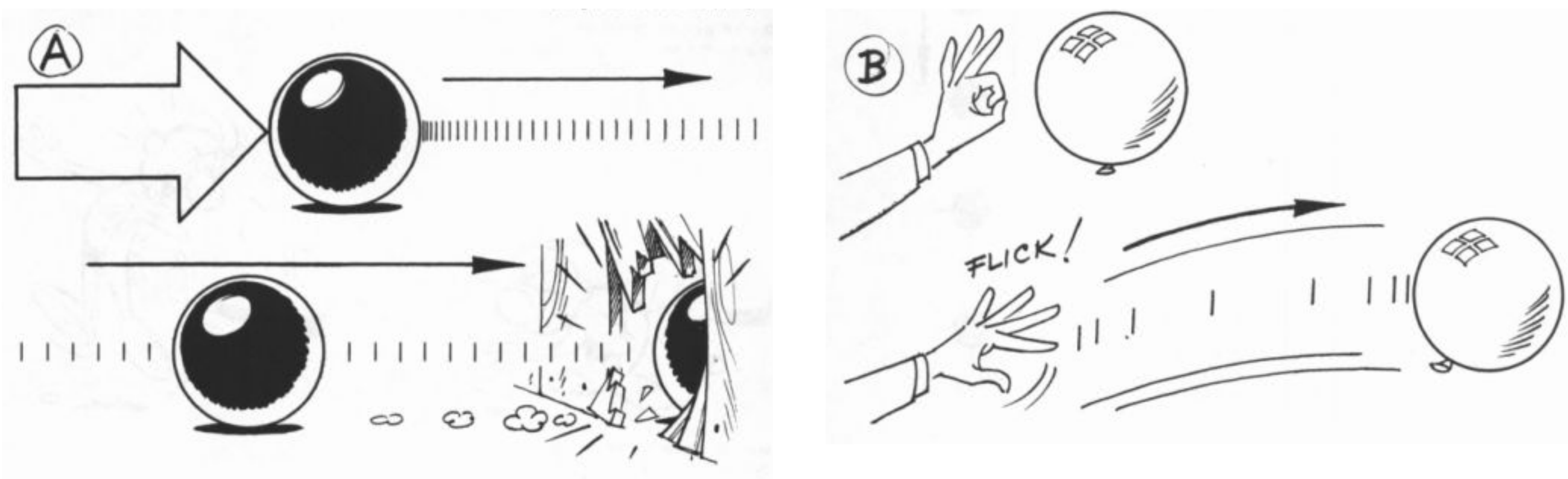
- Motion that results from some other action
- Needed for interest and realism
- Shouldn't distract from primary motion



Cartoon Animation, Preston Blair

Timing

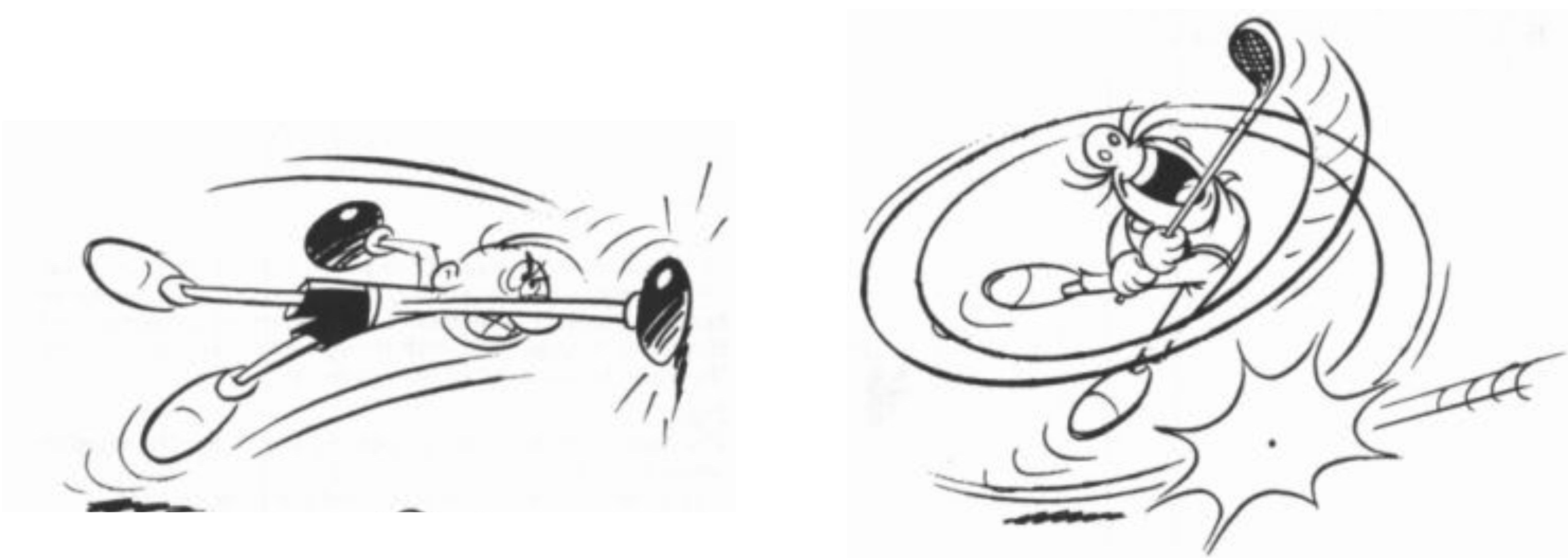
- Rate of acceleration conveys weight
- Speed and acceleration of character's movements convey emotion



Timing for Animation, Whitaker & Halas

Exaggeration

- Helps make actions clear
- Helps emphasize story points and emotion
- Must balance with non-exaggerated parts



Timing for Animation, Whitaker & Halas

Appeal

- Attractive to the eye, strong design
- Avoid symmetries

Disney Animation: The Illusion of Life

12 Animation principles

Cento Lodgiani, https://vimeo.com/93206523

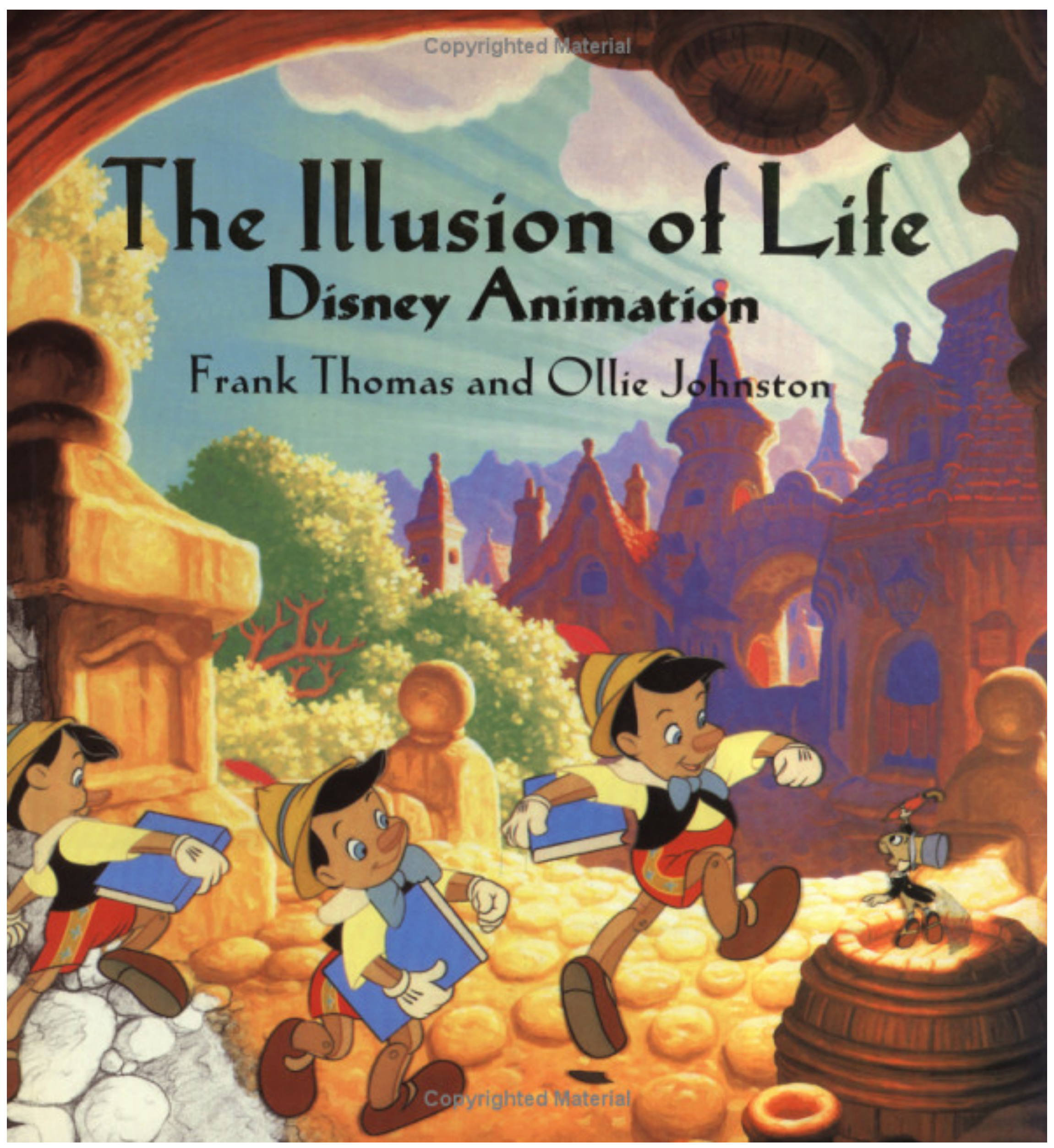
12 animation principles

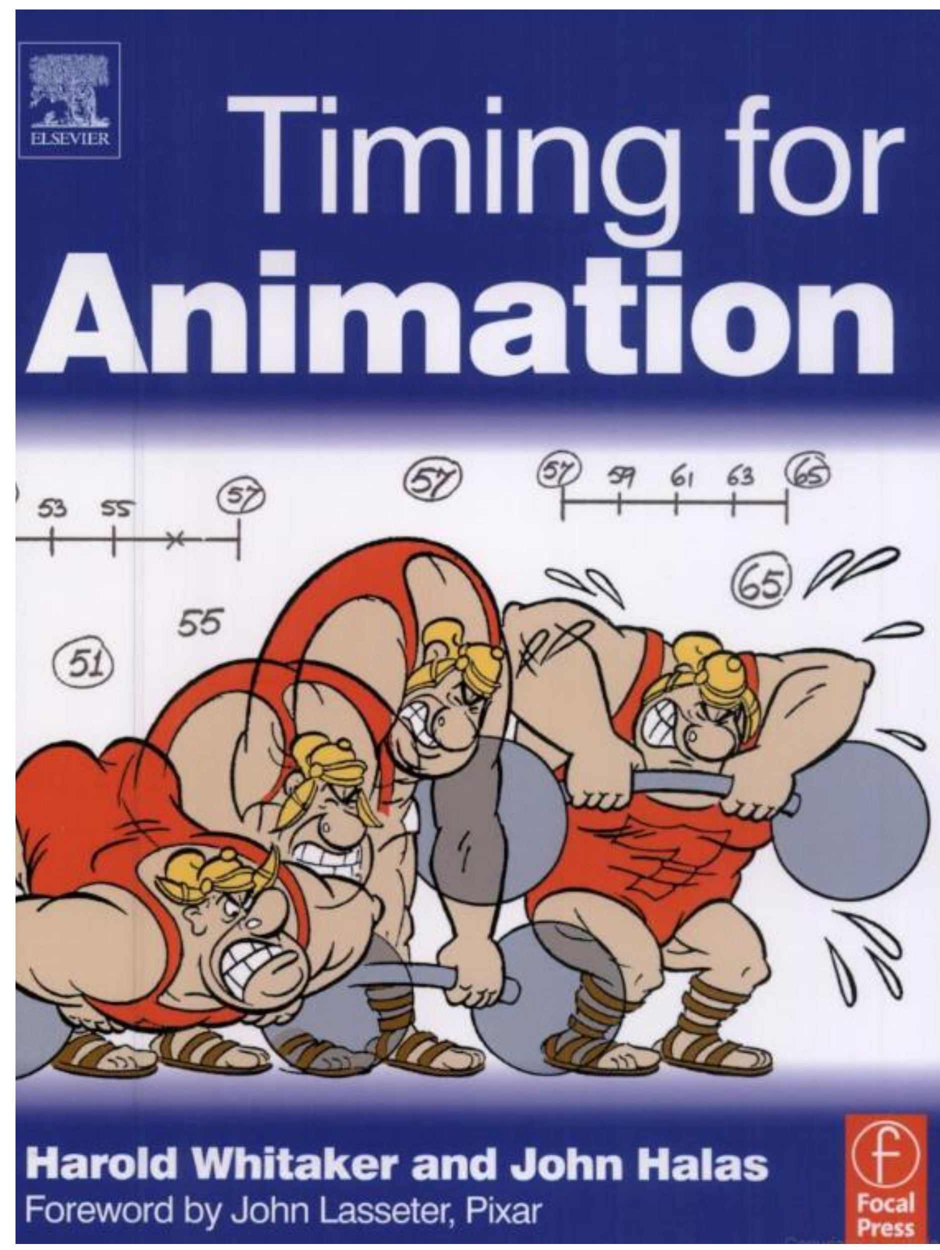
1. Squash and stretch
2. Anticipation
3. Staging
4. Straight ahead and pose-to-pose
5. Follow through
6. Ease-in and ease-out
7. Arcs
8. Secondary action
9. Timing
10. Exaggeration
11. Solid drawings
12. Appeal

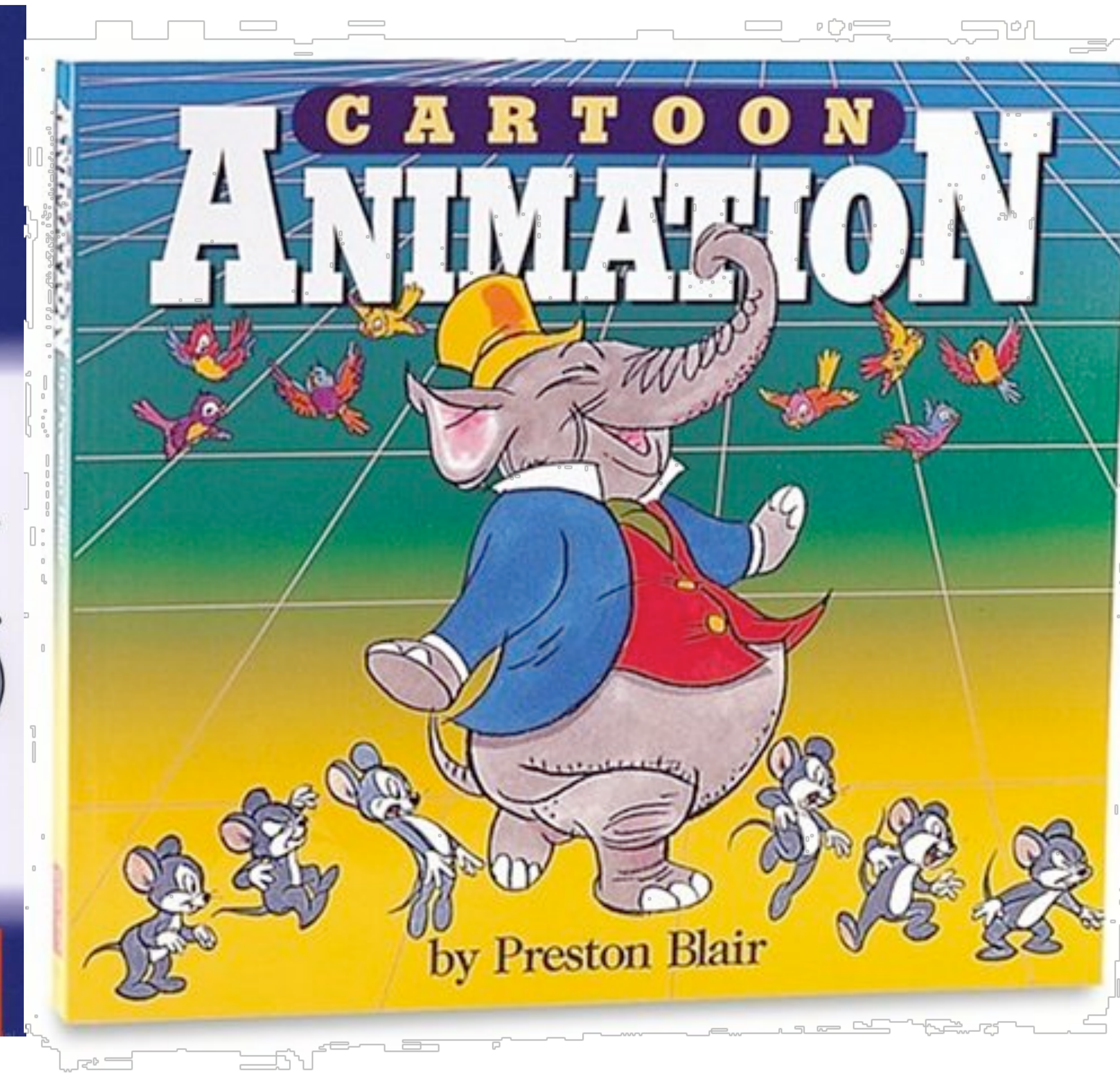
Personality

- Action of character is result of its thoughts
- Know purpose and mood before animating each action
- No two characters move the same way

Further reading



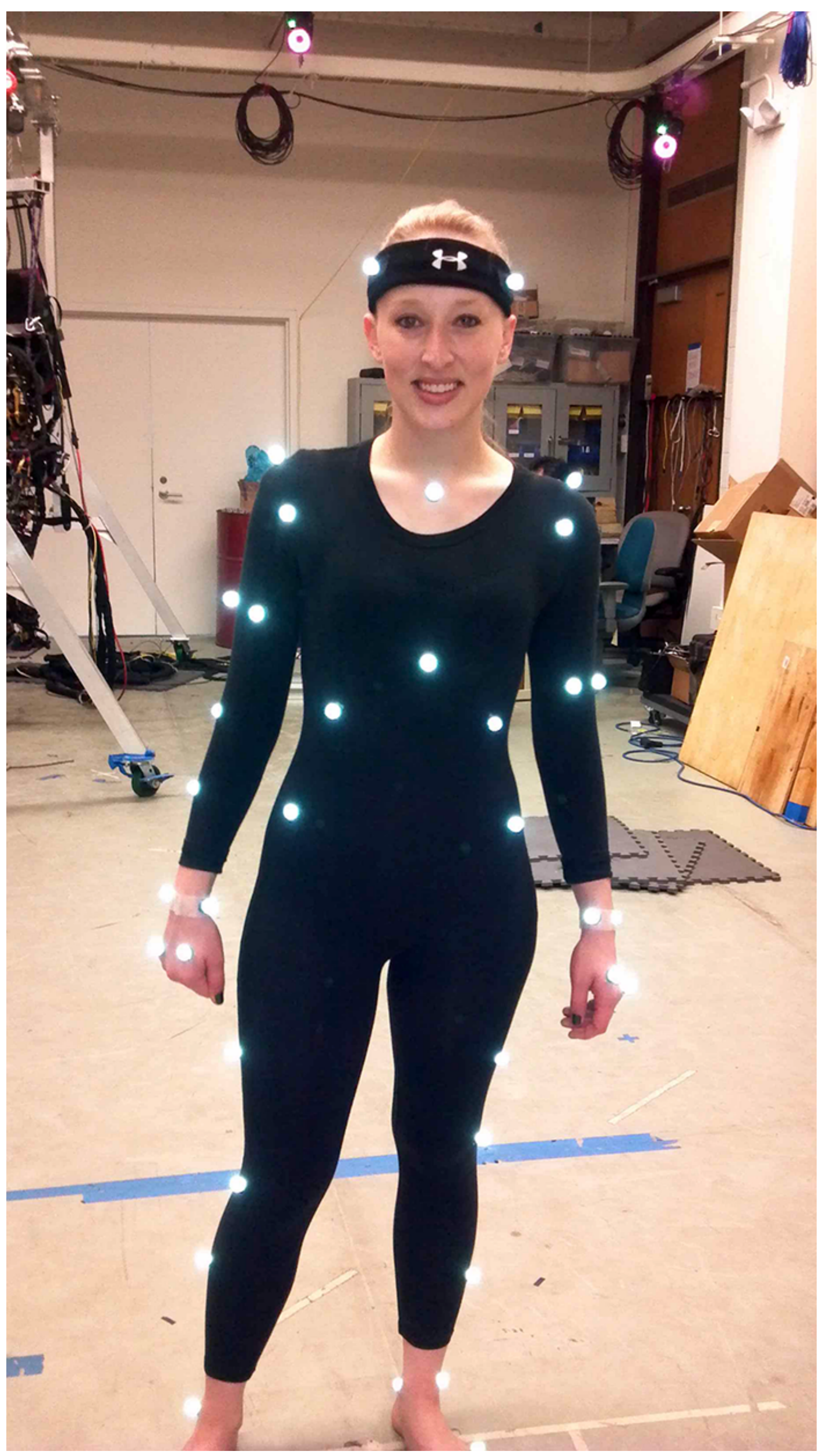


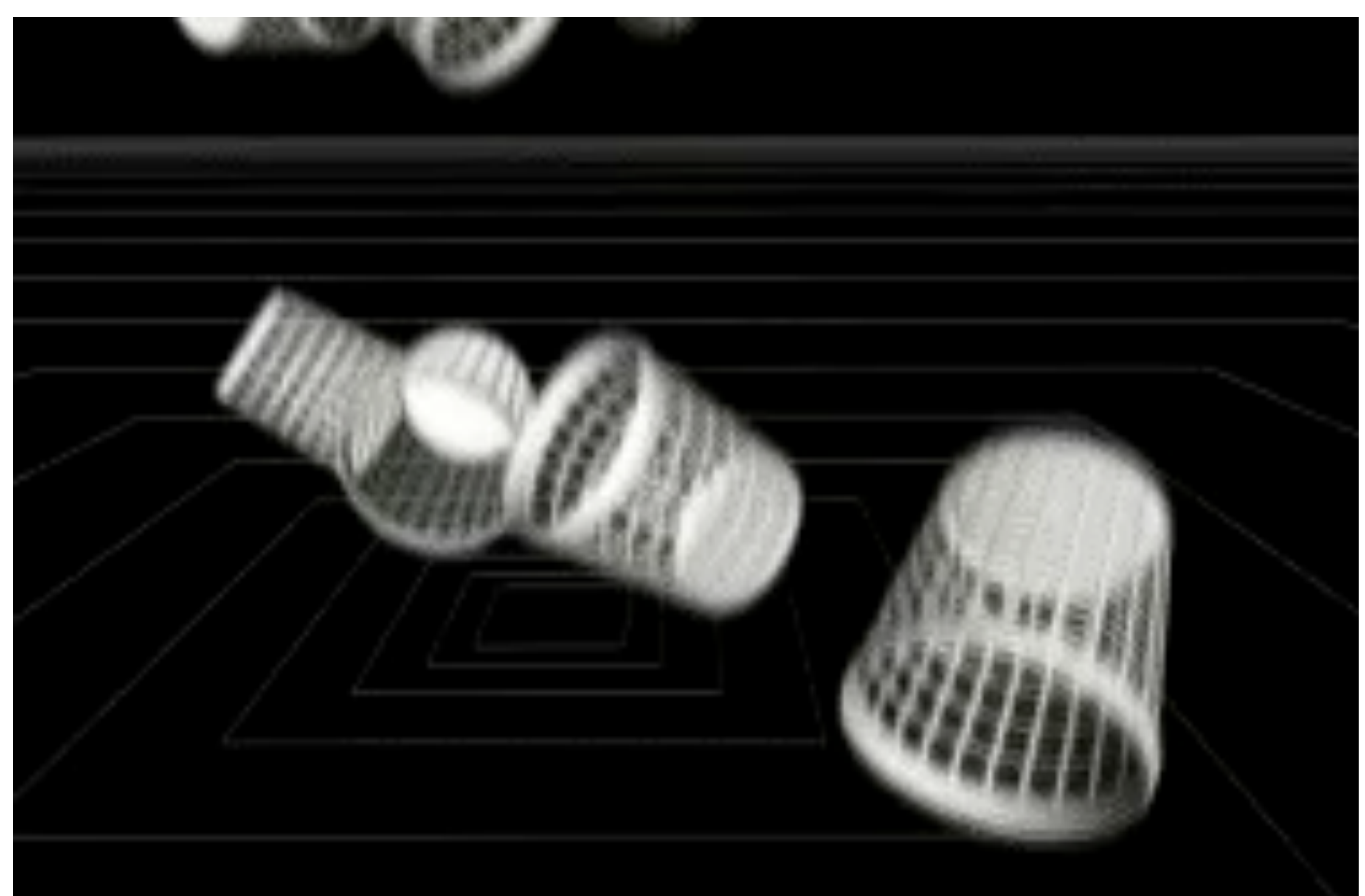


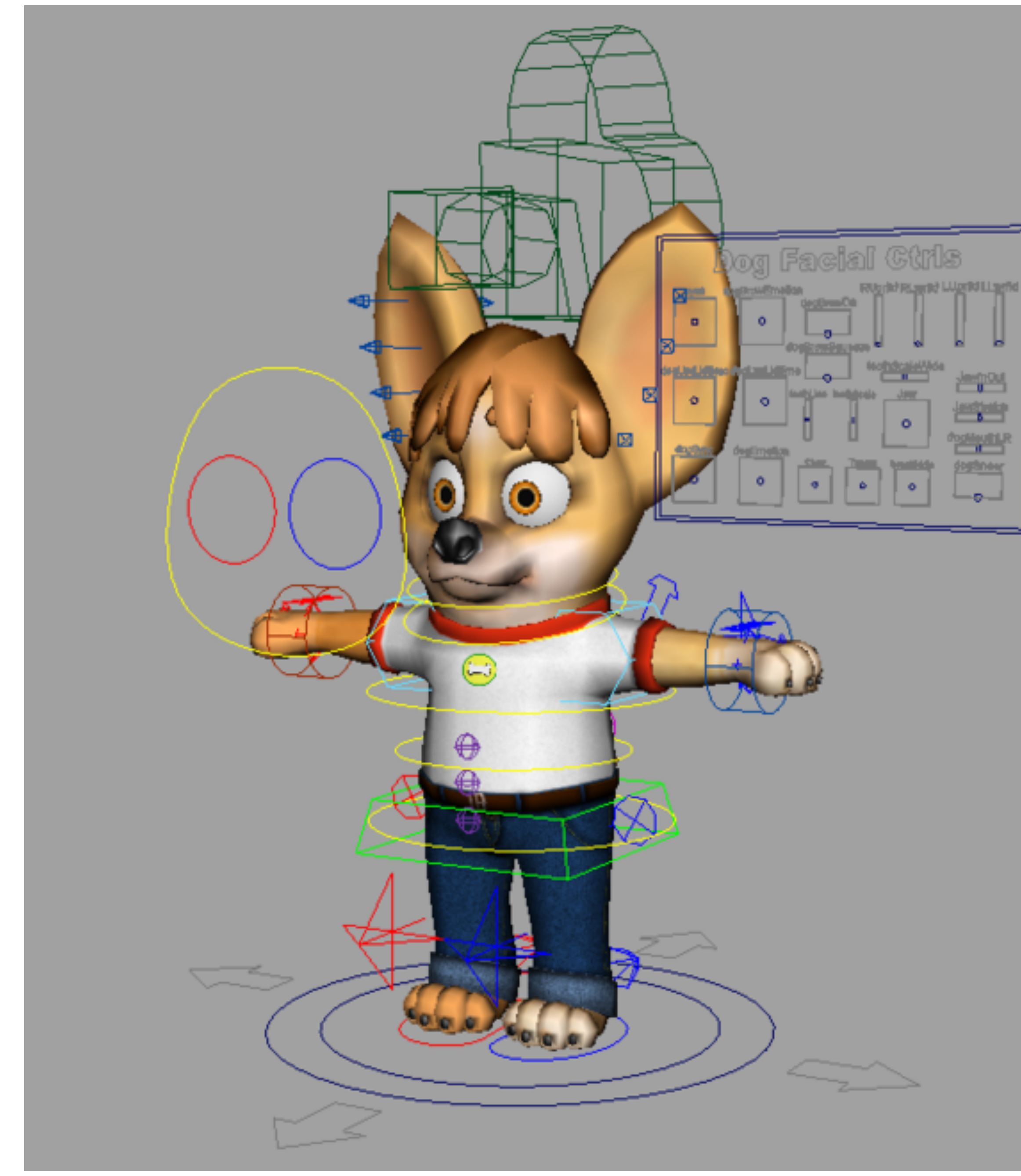
How do we describe motion on a computer?

Basic techniques in computer animation

- Artist-directed (e.g., keyframing)
- Data-driven (e.g., motion capture)
- Procedural (e.g., simulation)



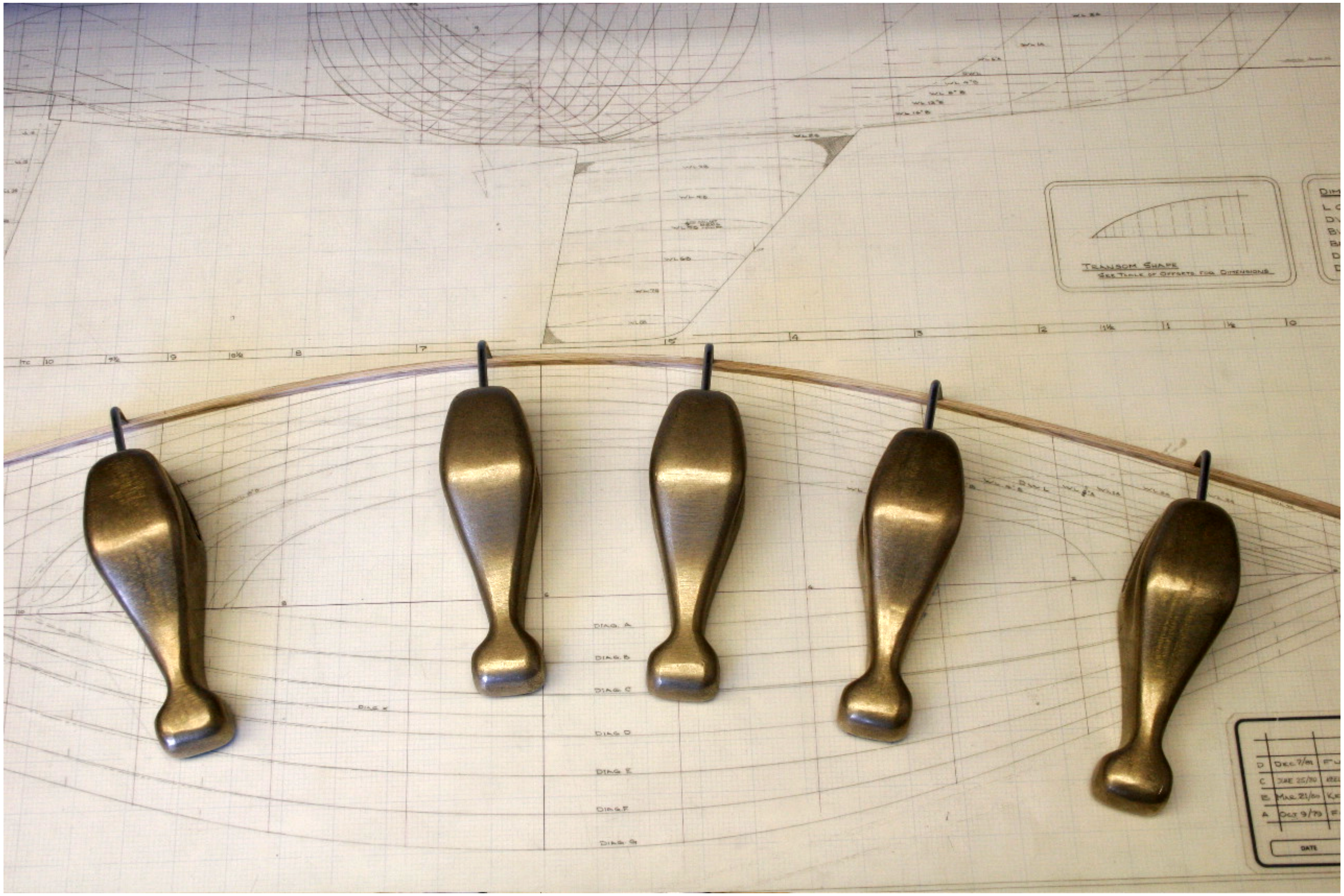




How do we interpolate data?

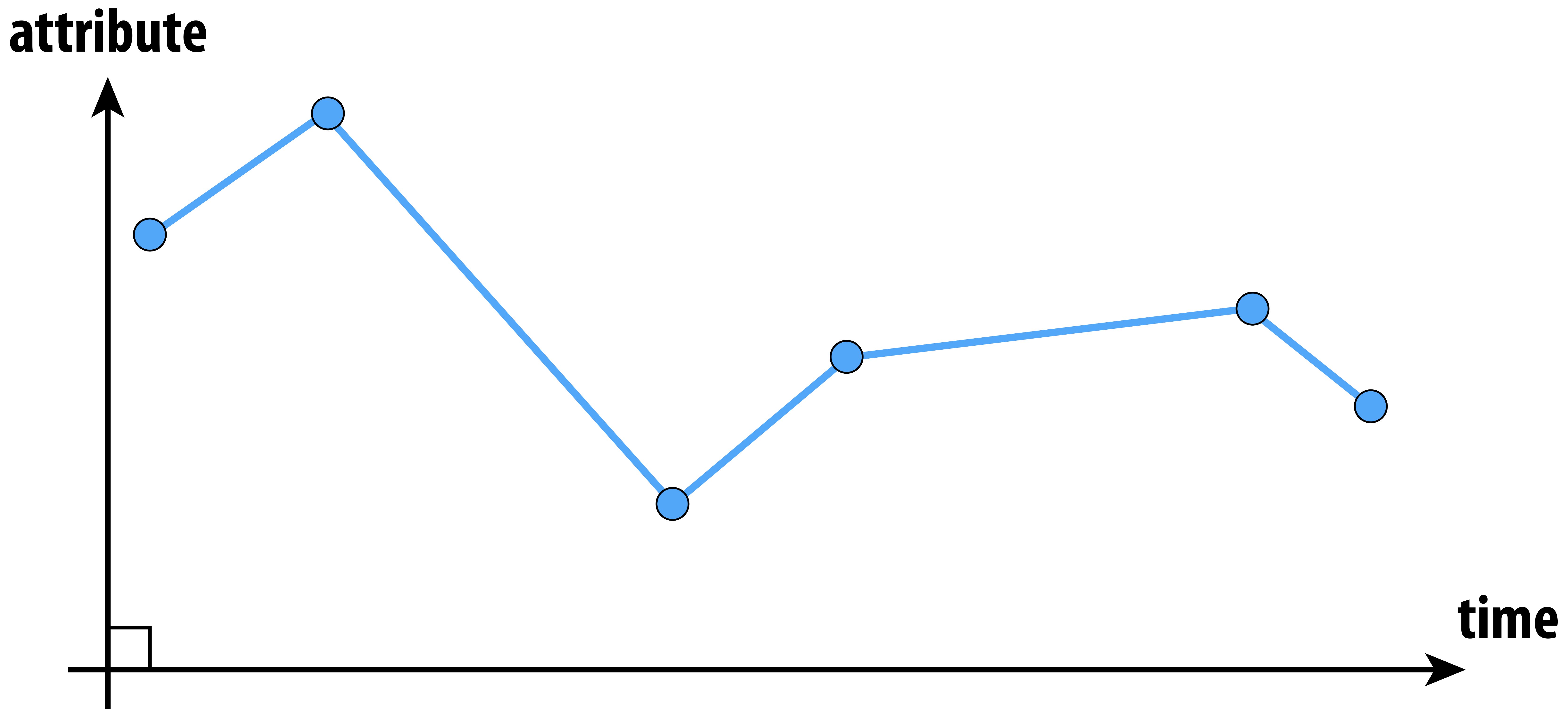
Spline interpolation

- **Mathematical theory of interpolation arose from study of thin strips of wood or metal ("splines") under various forces**



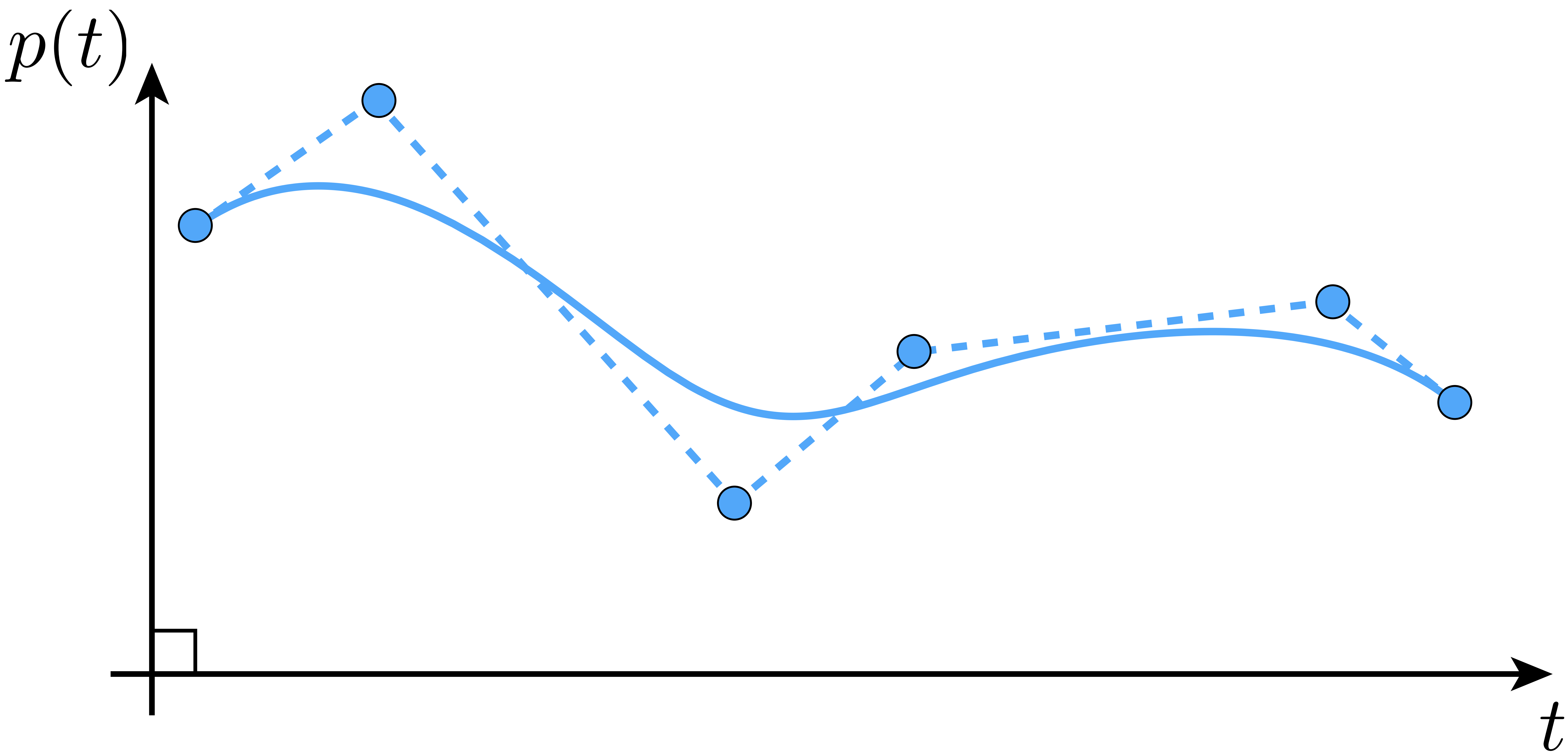
Interpolation

- Basic idea: "connect the dots"
- E.g., *piecewise linear interpolation*
- Simple, but yields rather rough motion (infinite acceleration)



Piecewise polynomial interpolation

- **Common interpolant: piecewise polynomial "spline"**



Basic motivation: get better continuity than piecewise linear!

Splines

- In general, a *spline* is any piecewise polynomial function
- In 1D, spline interpolates data over the real line:

$$(\underbrace{t_i}_{\text{"knots"}}, \underbrace{f_i}_{\text{values}}), \quad i = 0, \ldots, n$$

"knots"

values

$$t_i < t_{i+1}$$

- "Interpolates" means that the function *exactly* passes through those values:

$$f(t_i) = f_i \quad \forall i$$

- The only other condition is that the function is a *polynomial* when restricted to any interval between knots:

polynomial

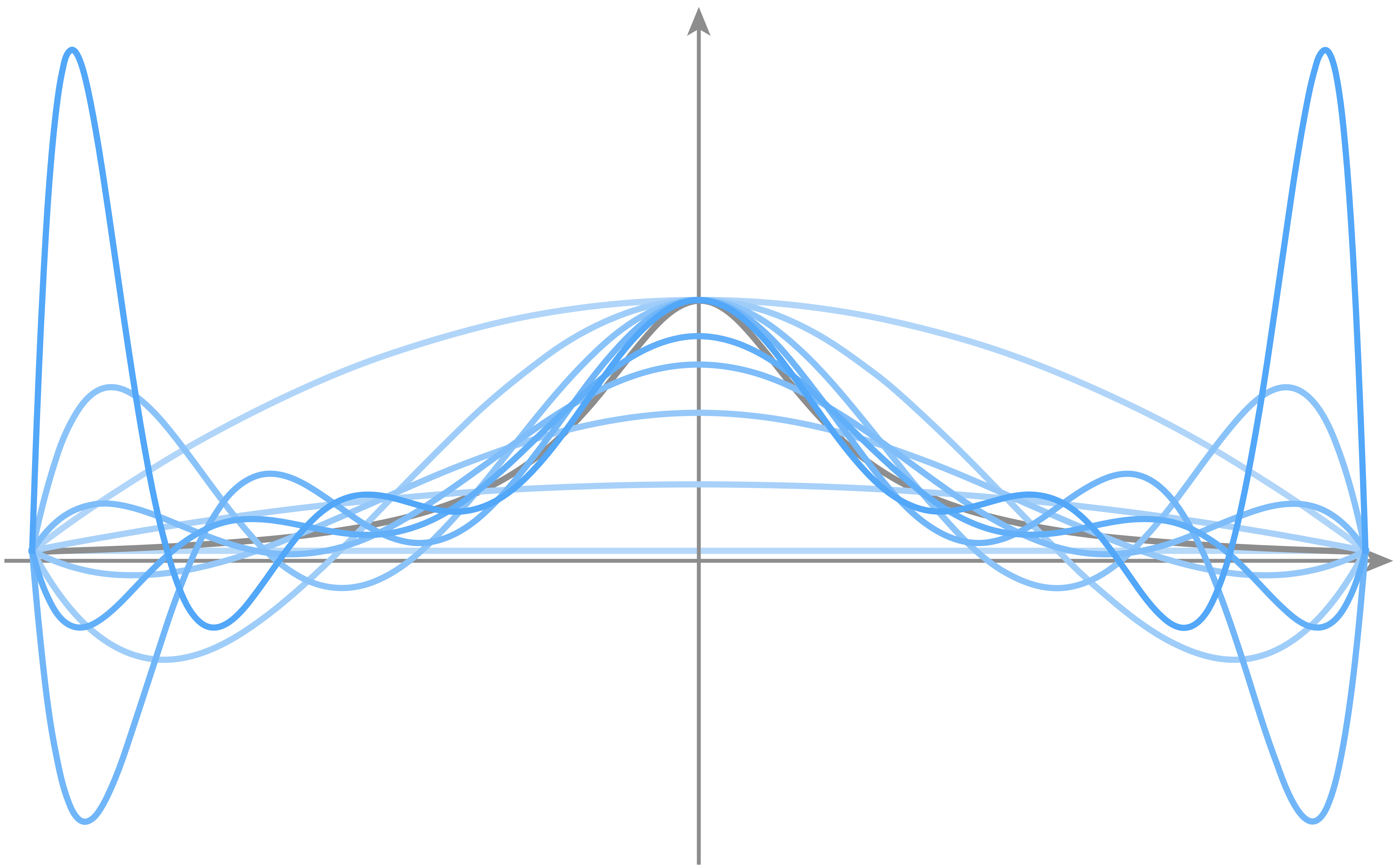
degree

$$\text{for } t_i \le t \le t_{i+1}, f(t) = \sum_{j=1}^{d} c_j t^j =: p_i(t)$$

coefficients

What's so special about *cubic* polynomials?

- Splines most commonly used for interpolation are *cubic* (d=3)
- Can provide "reasonable" continuity
- Tempting to use higher-degree polynomials to get higher-order continuity
- Can lead to oscillation, ultimately *worse* approximation:

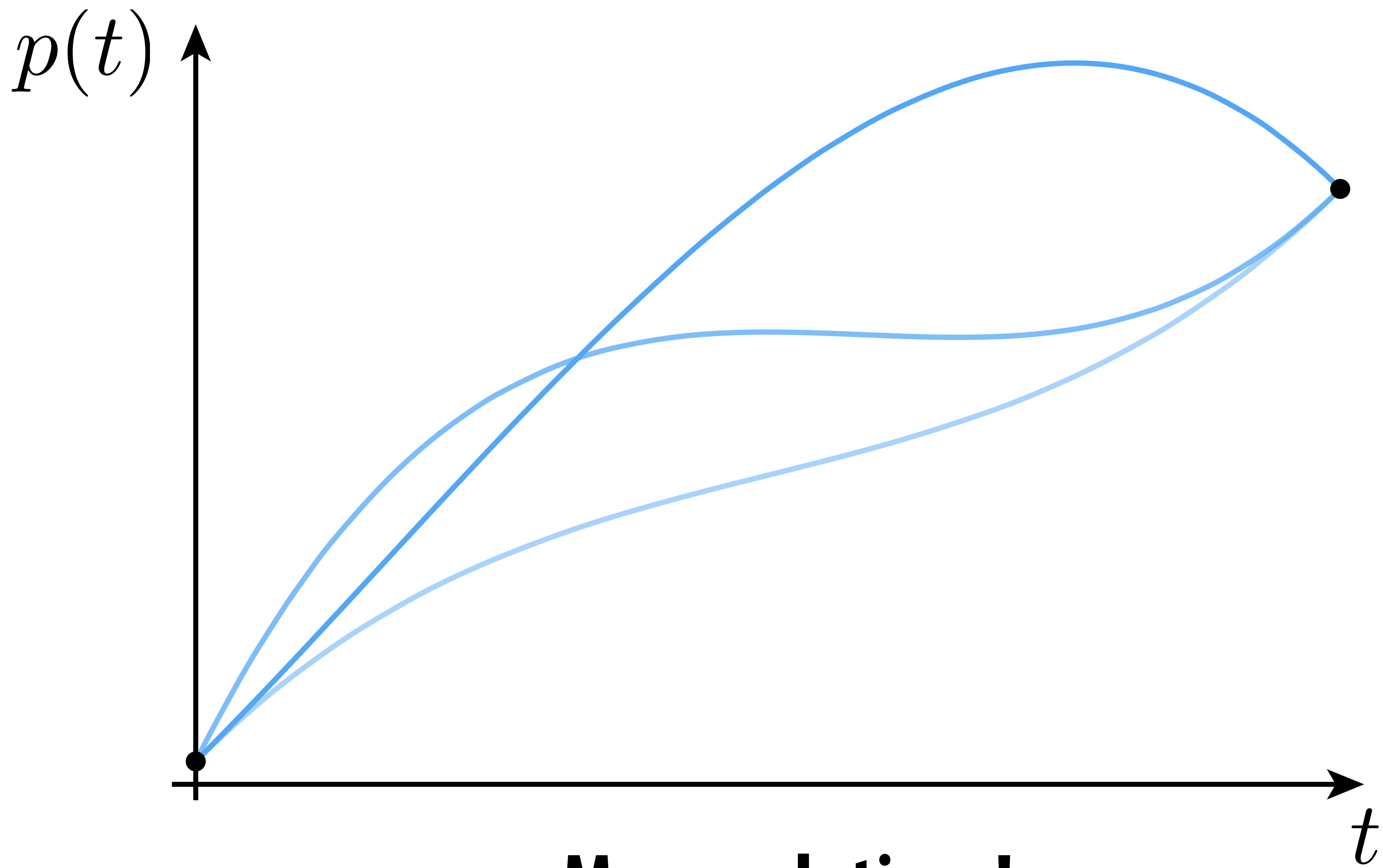


Fitting a cubic polynomial to endpoints

- **Consider a *single* cubic polynomial**

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

- **Suppose we want it to match given endpoints:**



Many solutions!

Cubic polynomial - degrees of freedom

- **Why are there so many different solutions?**
- **Cubic polynomial has four *degrees of freedom (DOFs)*, namely four coefficients (a,b,c,d) that we can manipulate/control**
- **Only need *two* degrees of freedom to specify endpoints:**

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

$$p(0) = p_0 \qquad \Rightarrow d = p_0$$

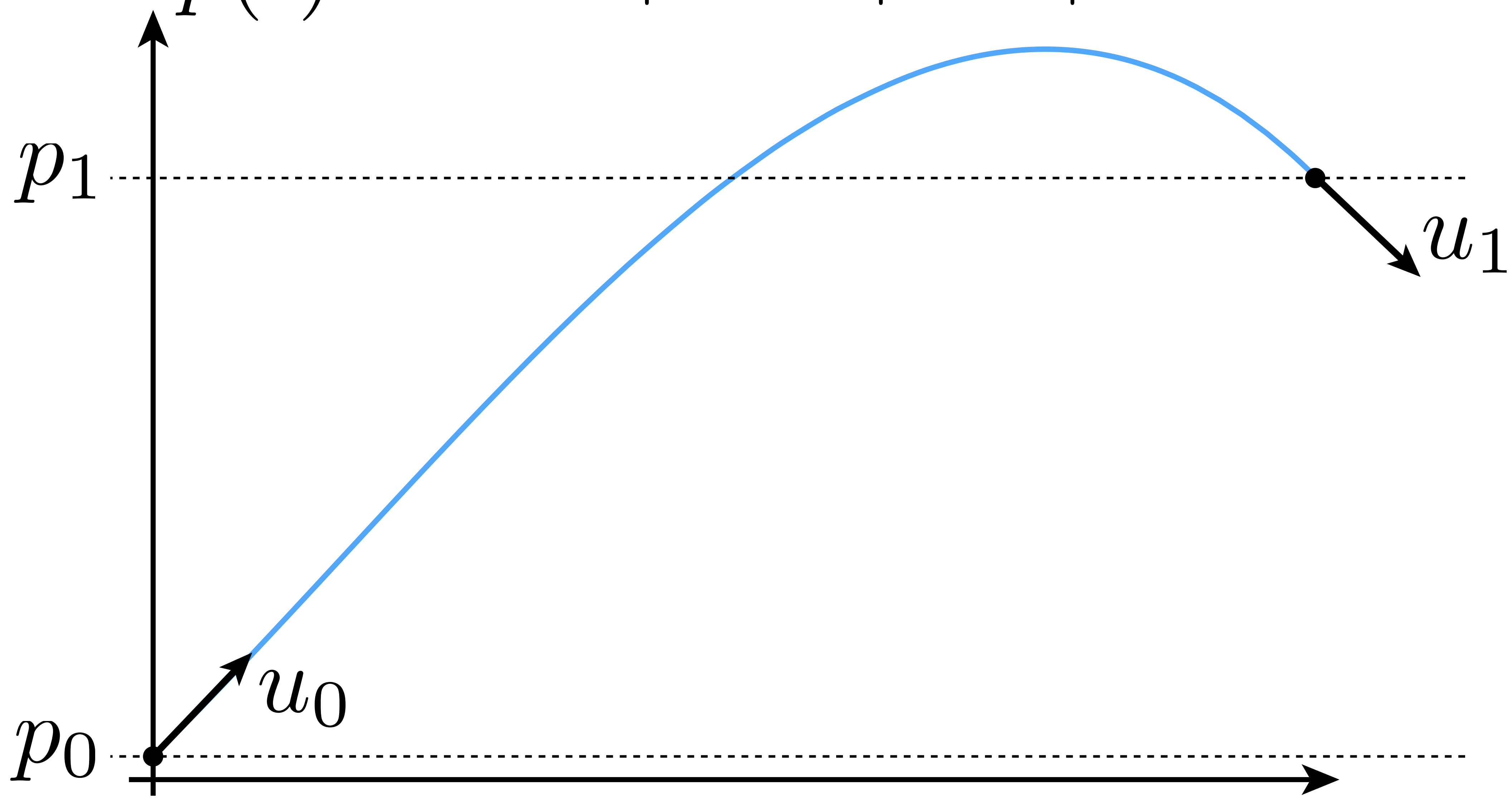
$$p(1) = p_1 \qquad \Rightarrow a + b + c + d = p_1$$

- **Overall, four unknowns but only *two* equations**
- **Not enough to uniquely determine the curve!**

Fitting cubic to endpoints and derivatives

- **What if we also match specified *derivatives* at endpoints?**

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



$$p(0) = p_0 \qquad \Rightarrow d = p_0$$

$$p(1) = p_1 \qquad \Rightarrow a + b + c + d = p_1$$

$$p'(0) = u_0 \qquad \Rightarrow c = u_0$$

$$p'(1) = u_1 \qquad \Rightarrow 3a + 2b + c = u_1$$

Splines as linear systems

- This time, we have four equations in four unknowns
- Could also express as a matrix equation:

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} p_0 \\ p_1 \\ u_0 \\ u_1 \end{bmatrix}$$

- This is a common way to define a spline
  - Each condition on spline leads to a linear equality
  - Hence, if we have m degrees of freedom, we need m (linearly independent!) conditions to determine spline

Solve for polynomial coefficients

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} p_0 \\ p_1 \\ u_0 \\ u_1 \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ u_0 \\ u_1 \end{bmatrix}$$

Matrix form

- **Interpolates endpoints, matches derivatives**

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

$$p(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ u_0 \\ u_1 \end{bmatrix}$$

Interpretation 1: matrix rows = coefficient formulas

$$p(t) = \boxed{a}t^3 + bt^2 + ct + d$$

$$= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ u_0 \\ u_1 \end{bmatrix}$$

Interpretation 2: matrix cols = ???

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

$$= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ u_0 \\ u_1 \end{bmatrix}$$

$$= \begin{bmatrix} 2t^2 - 3t^2 + 1 \\ -2t^3 + 3t^2 \\ t^3 - 2t^2 + t \\ t^3 - t^2 \end{bmatrix}^T \begin{bmatrix} p_0 \\ p_1 \\ u_0 \\ u_1 \end{bmatrix}$$

Hermite basis functions

$$p(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} H_0(t) & H_1(t) & H_2(t) & H_3(t) \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ u_0 \\ u_1 \end{bmatrix}$$

One common basis for cubic polynomials

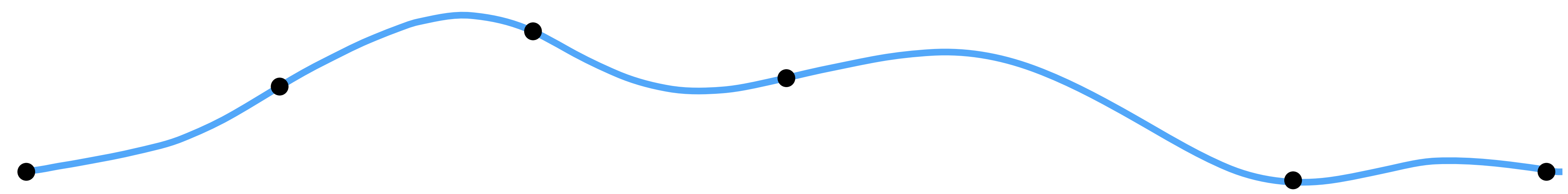
$$f_0(t) = t^3$$
$$f_1(t) = t^2$$
$$f_2(t) = t$$
$$f_3(t) = 1$$

Hermite Basis for cubic polynomials

$$H_0(t) = 2t^2 - 3t^2 + 1$$
$$H_1(t) = -2t^3 + 3t^2$$
$$H_2(t) = t^3 - 2t^2 + t$$
$$H_3(t) = t^3 - t^2$$

Either basis can represent a cubic polynomial through linear combination!

Natural splines



- Now consider *piecewise* spline made of *n* cubic polynomials p_i
- For each interval, want polynomial "piece" p_i to interpolate data (e.g., keyframes) at both endpoints:

$$p_i(t_i) = f_i,\ \ p_i(t_{i+1}) = f_{i+1},\ \ i = 0, \ldots, n-1$$

- Want tangents to agree at endpoints ("C^1 continuity"):

$$p_i'(t_{i+1}) = p_{i+1}'(t_{i+1}),\ \ i = 0, \ldots, n-2$$

- Also want curvature to agree at endpoints ("C^2 continuity"):

$$p_i''(t_{i+1}) = p_{i+1}''(t_{i+1}),\ \ i = 0, \ldots, n-2$$

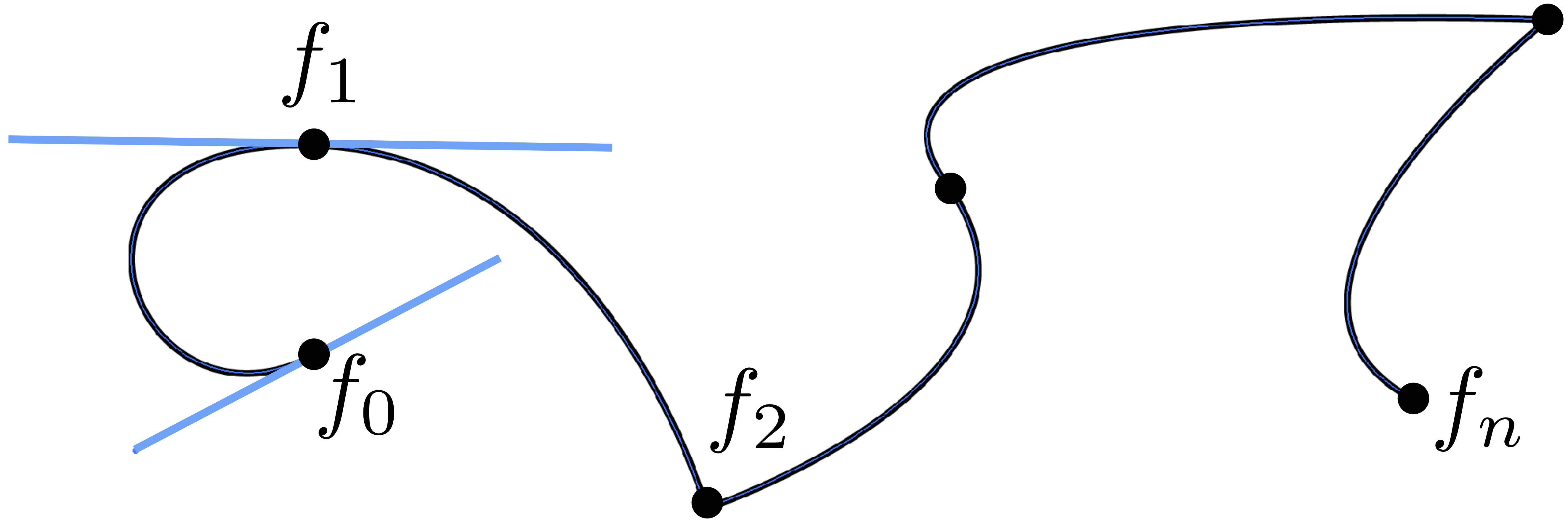
- How many equations do we have at this point?
  - 2n+(n-1)+(n-1) = 4n-2
- Pin down remaining DOFs by setting 2nd derivative (curvature) to zero at endpoints

Spline desiderata

- In general, what are some properties of a "good" spline?
  - INTERPOLATION: spline passes *exactly* through data points
  - CONTINUITY: at least *twice* differentiable everywhere (for animation = constant "acceleration")
  - LOCALITY: moving one control point doesn't affect whole curve
- How does our natural spline do?
  - INTERPOLATION: yes, by construction
  - CONTINUITY: C^2 everywhere, by construction
  - LOCALITY: no, coefficients depend on global linear system
- Many other types of splines we can consider
- Spoiler: there is "no free lunch" with cubic splines (can't simultaneously get all three properties)

Back to Hermite splines from earlier in lecture

- **Hermite: each cubic "piece" specified by endpoints and tangents:**



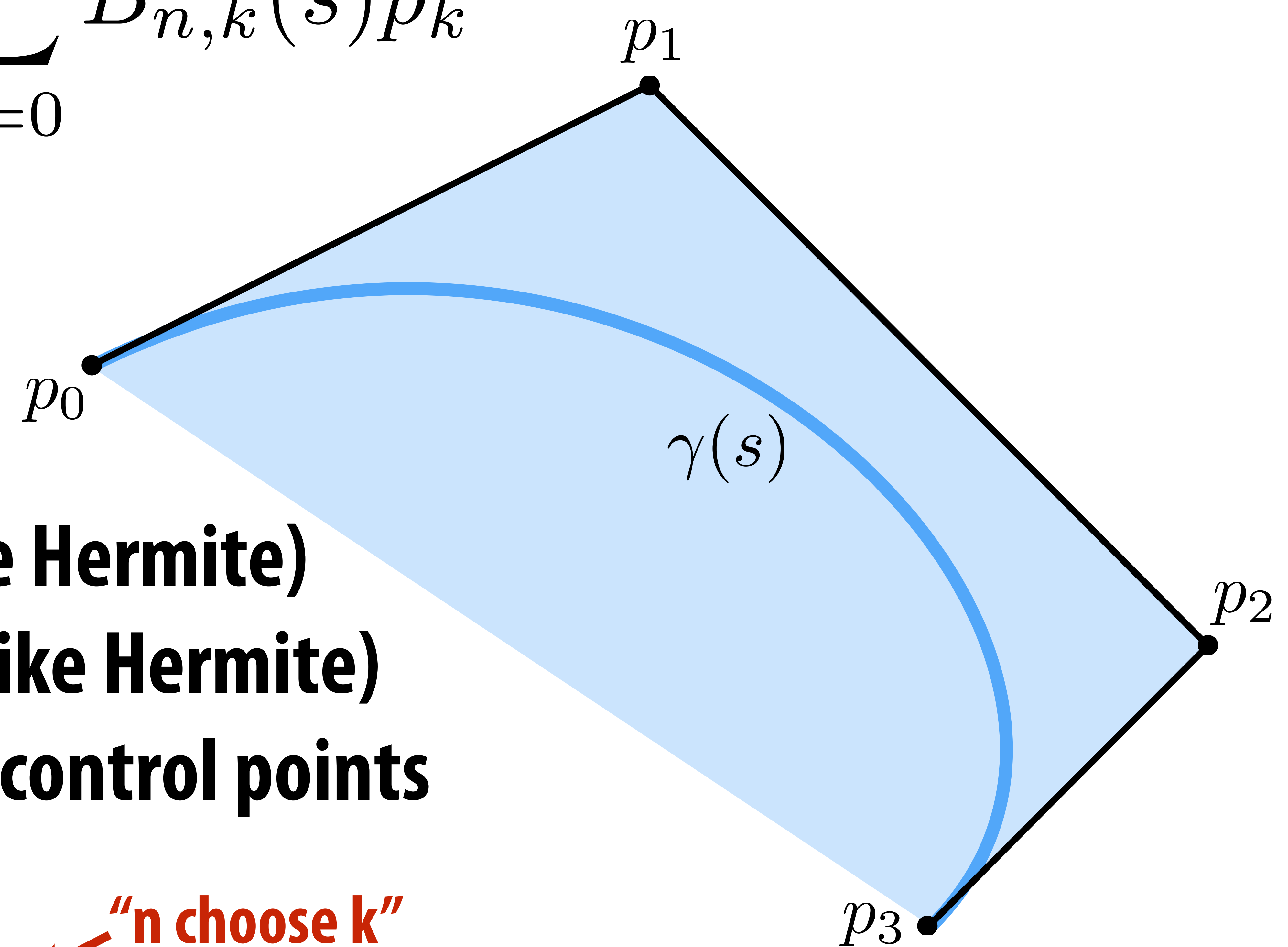
- **Commonly used for 2D vector art (Illustrator, Inkscape, SVG, ...)**
- **Can we get tangent (C1) continuity?**
- **Sure: set both tangents to same value on both sides of knot!**
  - **E.g., f_1 above, but not f_2**

Recall from geometry lecture: Bézier curves

- **A Bézier curve is a curve expressed in the Bernstein basis:**

control points

$$\gamma(s) := \sum_{k=0}^{n} B_{n,k}(s) p_k$$



- **For n=3, get "cubic Bézier":**
- **Properties:**
  1. **interpolates endpoints (like Hermite)**
  2. **tangent to end segments (like Hermite)**
  3. **contained in convex hull of control points**

degree

0≤x≤1

"n choose k"

$$B_k^n(x) := \binom{n}{k} x^k (1-x)^{n-k}$$

k=0,…,n

Properties of Hermite/Bézier spline

- **More precisely, want endpoints to interpolate data:**

$$p_i(t_i) = f_i,\ \ p_i(t_{i+1}) = f_{i+1},\ \ i = 0, \ldots, n-1$$

- **Also want tangents to interpolate some given data:**

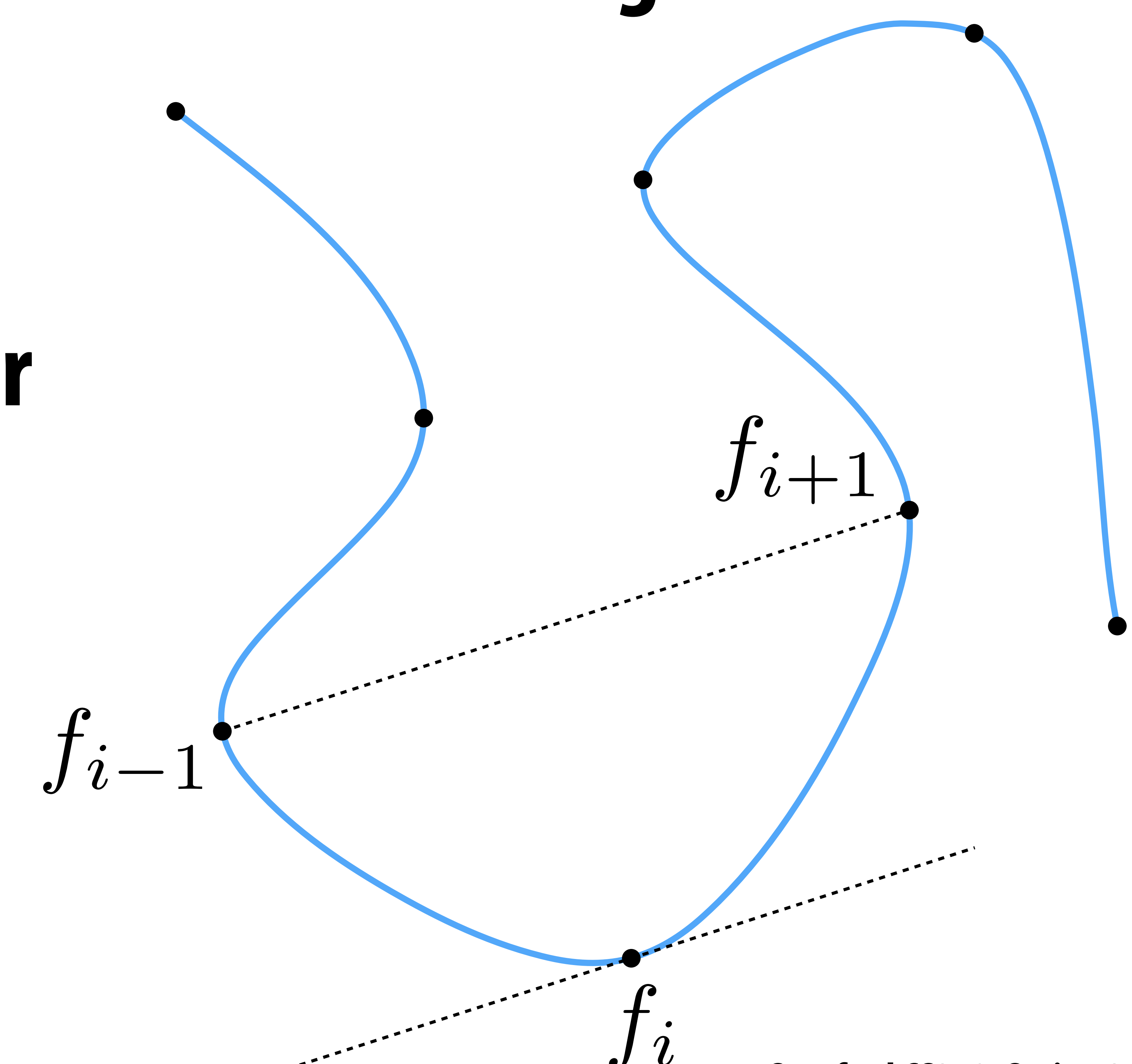
$$p_i'(t_i) = u_i,\ \ p'(i)_{t_{i+1}} = u_{i+1},\ \ i = 0, \ldots, n-1$$

- **How is this *different* from our natural spline's tangent condition?**
- **There, tangents didn't have to match any prescribed value— they merely had to be the same. Here, they are given.**
- **How many conditions overall?**
  - **2n + 2n = 4n**
- **What properties does this curve have?**
  - **INTERPOLATION and LOCALITY, but not C^2 CONTINUITY**

Catmull-Rom splines

- Sometimes makes sense to specify *tangents* (e.g., illustration)
- Often more convenient to just specify *values*
- Catmull-Rom: specialization of Hermite spline, determined by values alone
- Basic idea: use difference of neighbors to define tangent

$$u_i := \frac{f_{i+1} - f_{i-1}}{t_{i+1} - t_{i-1}}$$



- All the same properties as any other Hermite spline (locality, etc.)
- Commonly used to interpolate motion in computer animation.
- Many, many variants, but Catmull-Rom is usually good starting point

Spline desiderata, revisited

	INTERPOLATION	CONTINUITY	LOCALITY
natural	YES	YES	NO
Hermite	YES	NO	YES
???	NO	YES	YES

See B-Splines

But what exactly are we interpolating?

Simple example: camera path

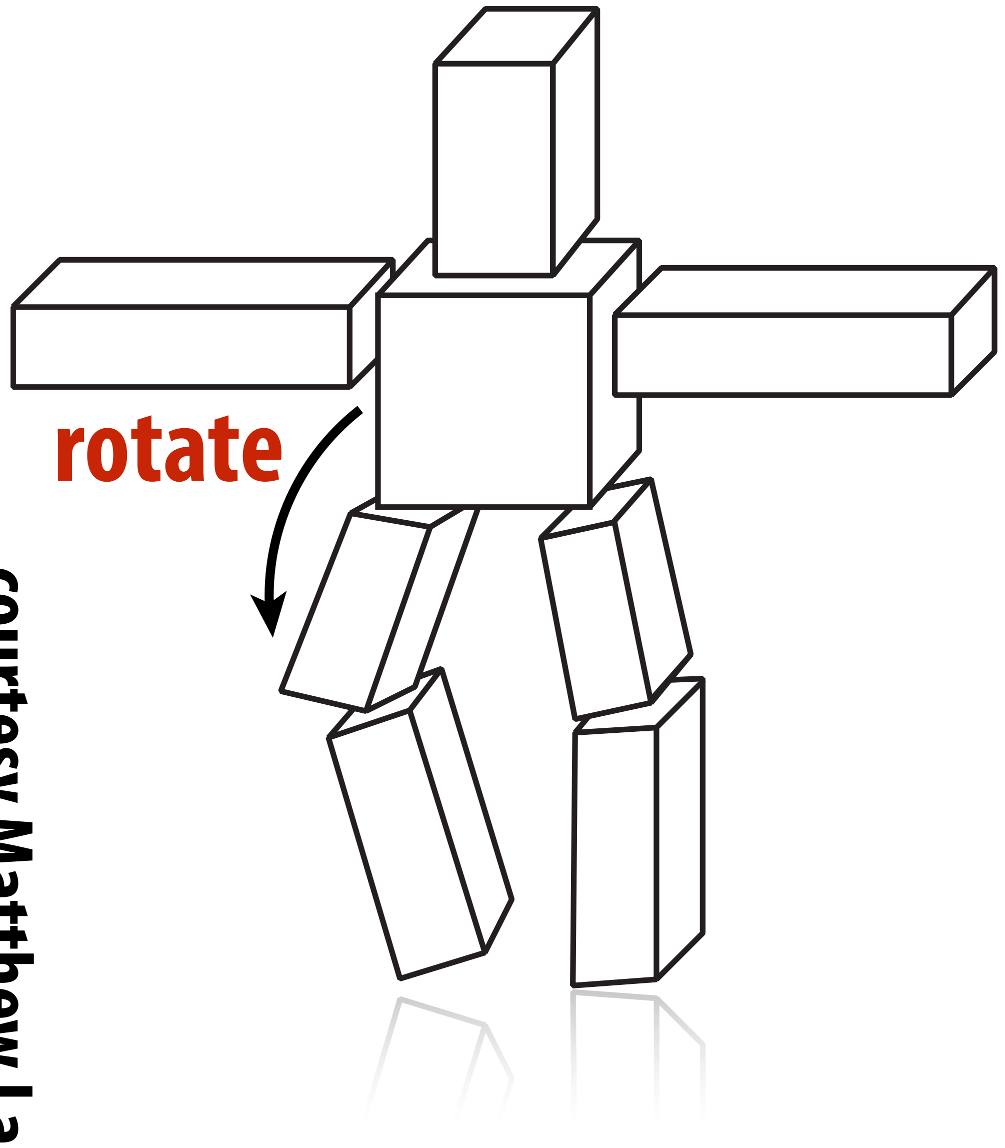
- Animate position, direction, "up" direction of camera
  - each path is a function f(t) = (x(t), y(t), z(t))
  - each component (x,y,z) is a spline

Zaha Hadid Architects—City of Dreams Hotel Tower

Character animation

- *Scene graph/kinematic chain*: scene as tree of transformations
- E.g. in our "cube man," configuration of a leg might be expressed as rotation relative to body
- Animate by interpolating transformations
- Often have sophisticated "rig":

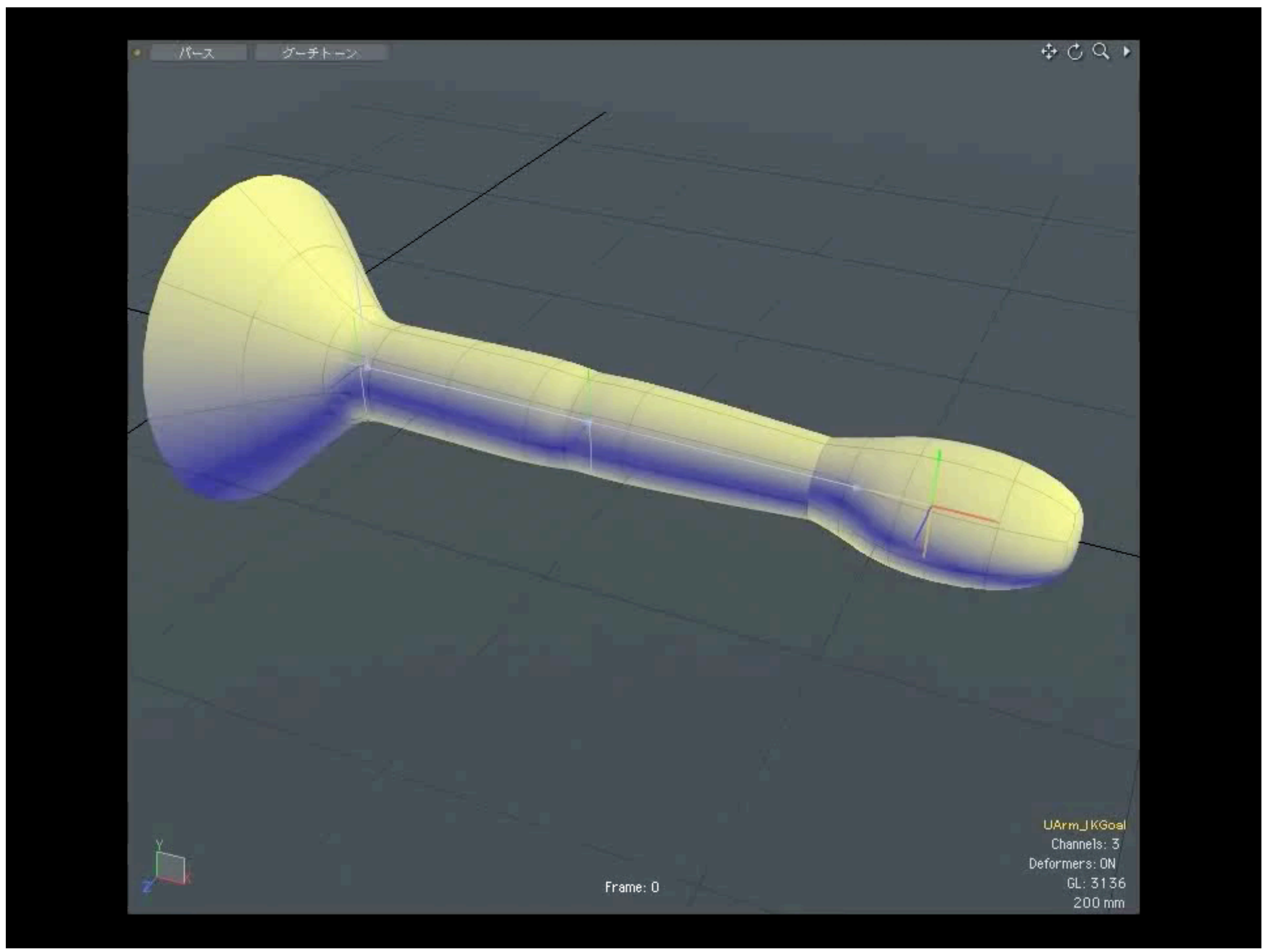
courtesy Matthew Lailler



Even w/ computer "tweening," its a lot of work to animate!

Inverse kinematics

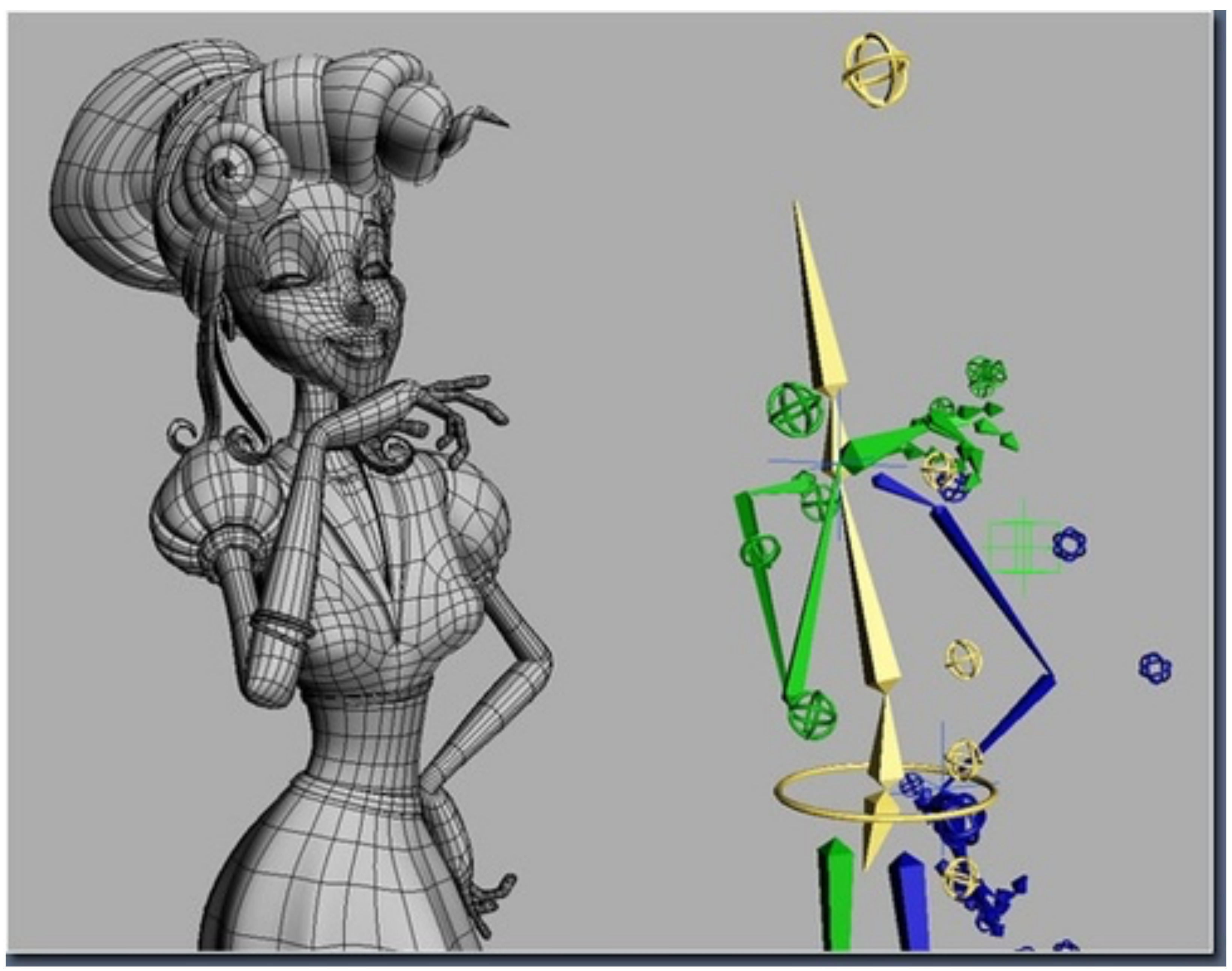
- Important technique in animation & robotics
- Rather than adjust individual transformations, set “goal” and use algorithm to come up with plausible motion:

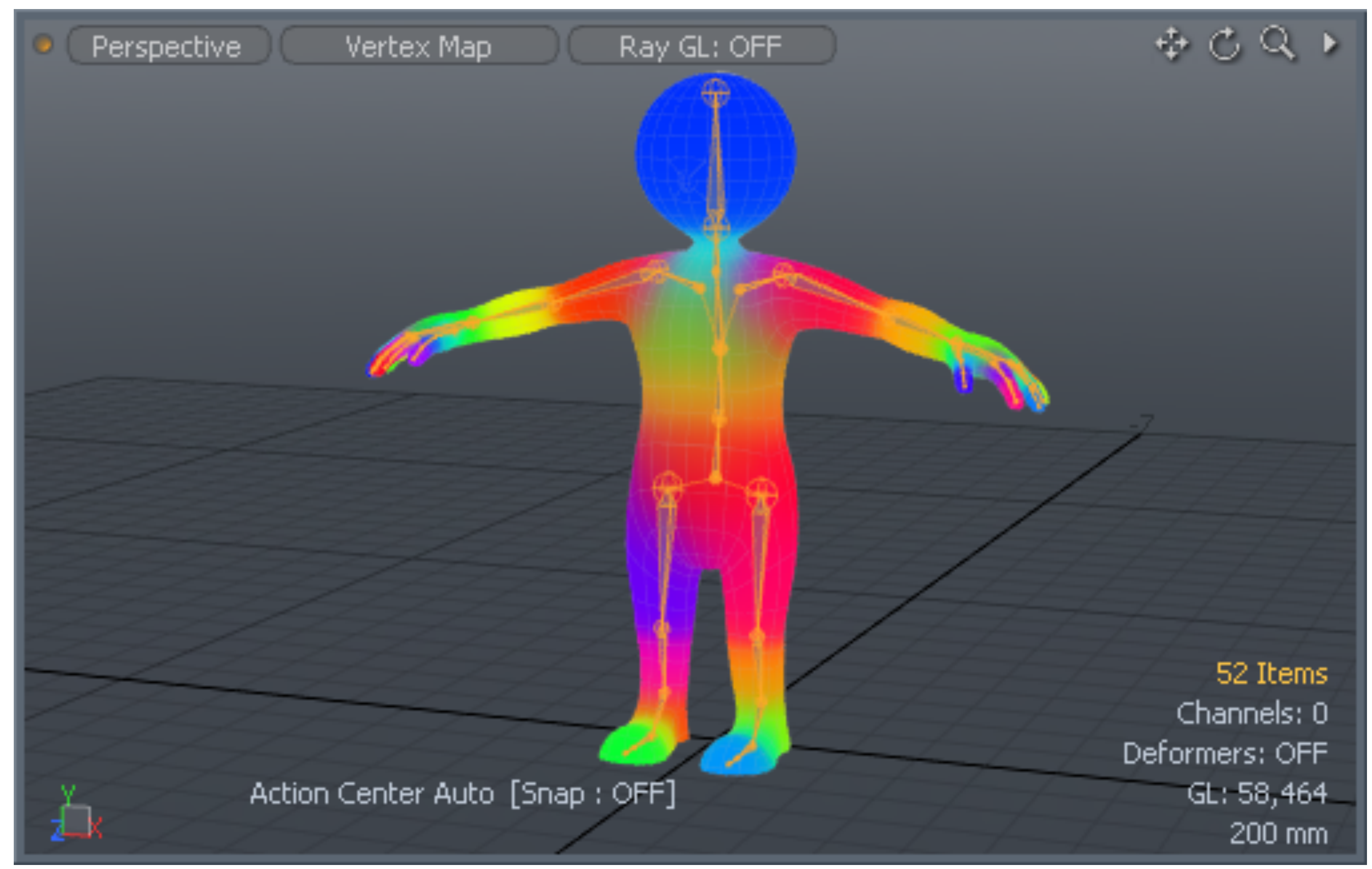


Many algorithms—to be discussed in a future lecture

Skeletal animation

- Previous characters looked a lot different from "cube man"!
- Often use "skeleton" to drive deformation of continuous surface
- Influence of each bone determined by, e.g., weighting function:





(Many, many other possibilities—still very active area of R&D)

Blend shapes

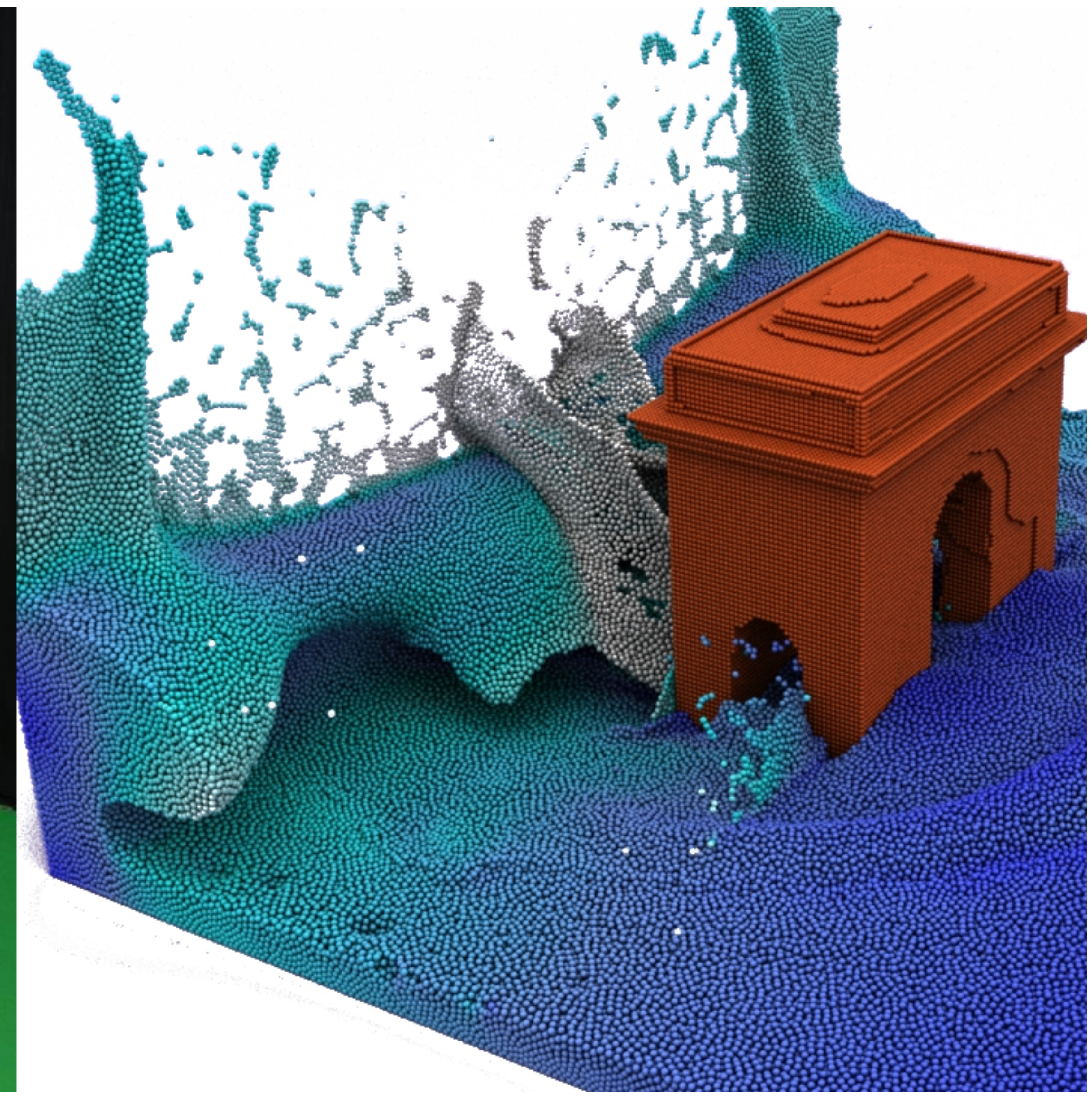
- Instead of skeleton, interpolate directly between surfaces
- E.g., model a collection of facial expressions:

courtesy Félix Ferrand

- Simplest scheme: take linear combination of vertex positions
- Spline used to control choice of weights over time

Coming up next...

- Even with "computer-aided tweening," animating a scene by hand takes a lot of work!
- Will see how data capture and physical simulation can help



Acknowledgements

- Thanks to Keenan Crane, Ren Ng, and Mark Pauly for presentation resources