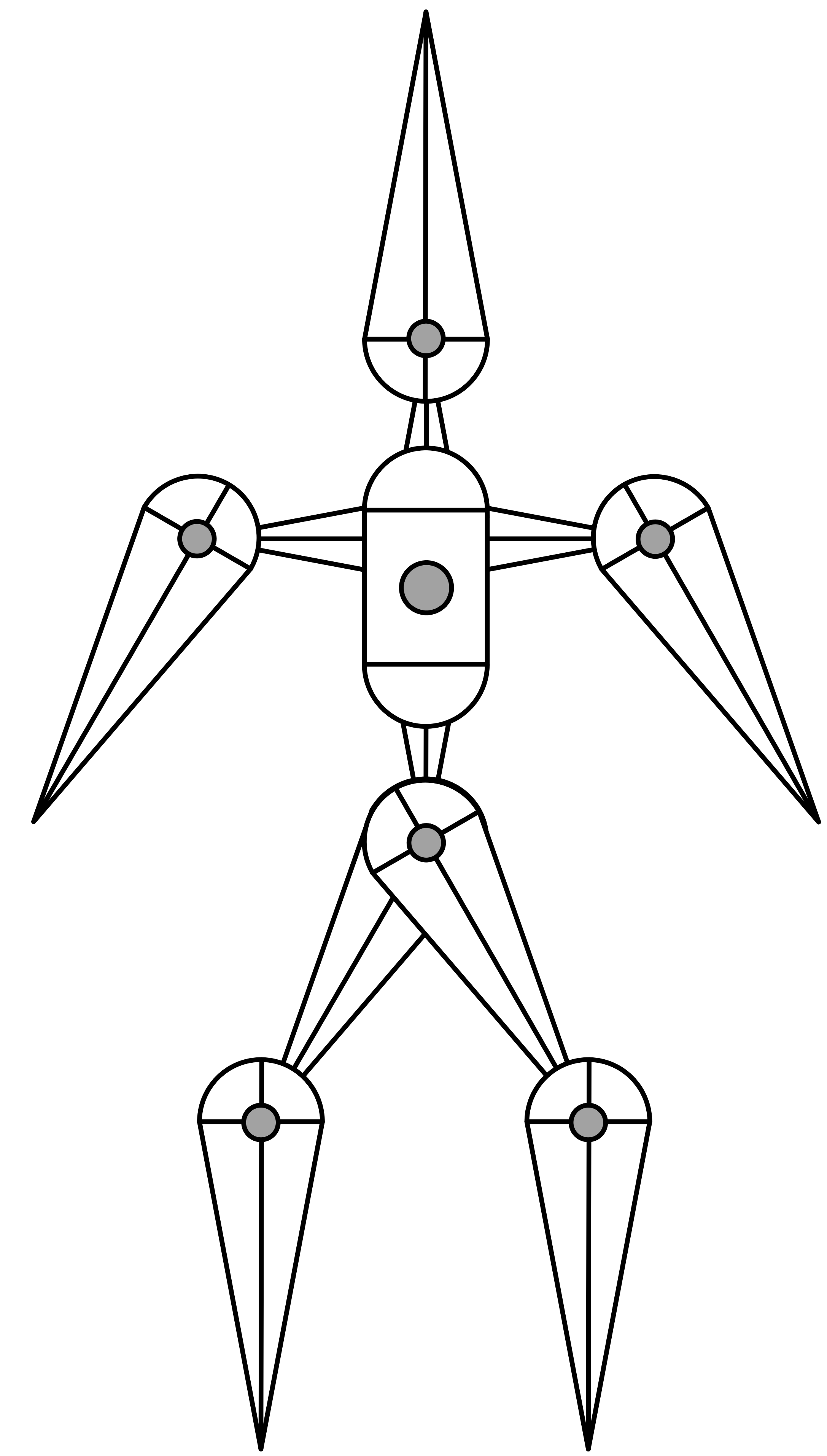
Lecture 13

Kinematics and Motion Capture

Interactive Computer Graphics
Stanford CS248, Winter 2019

Today

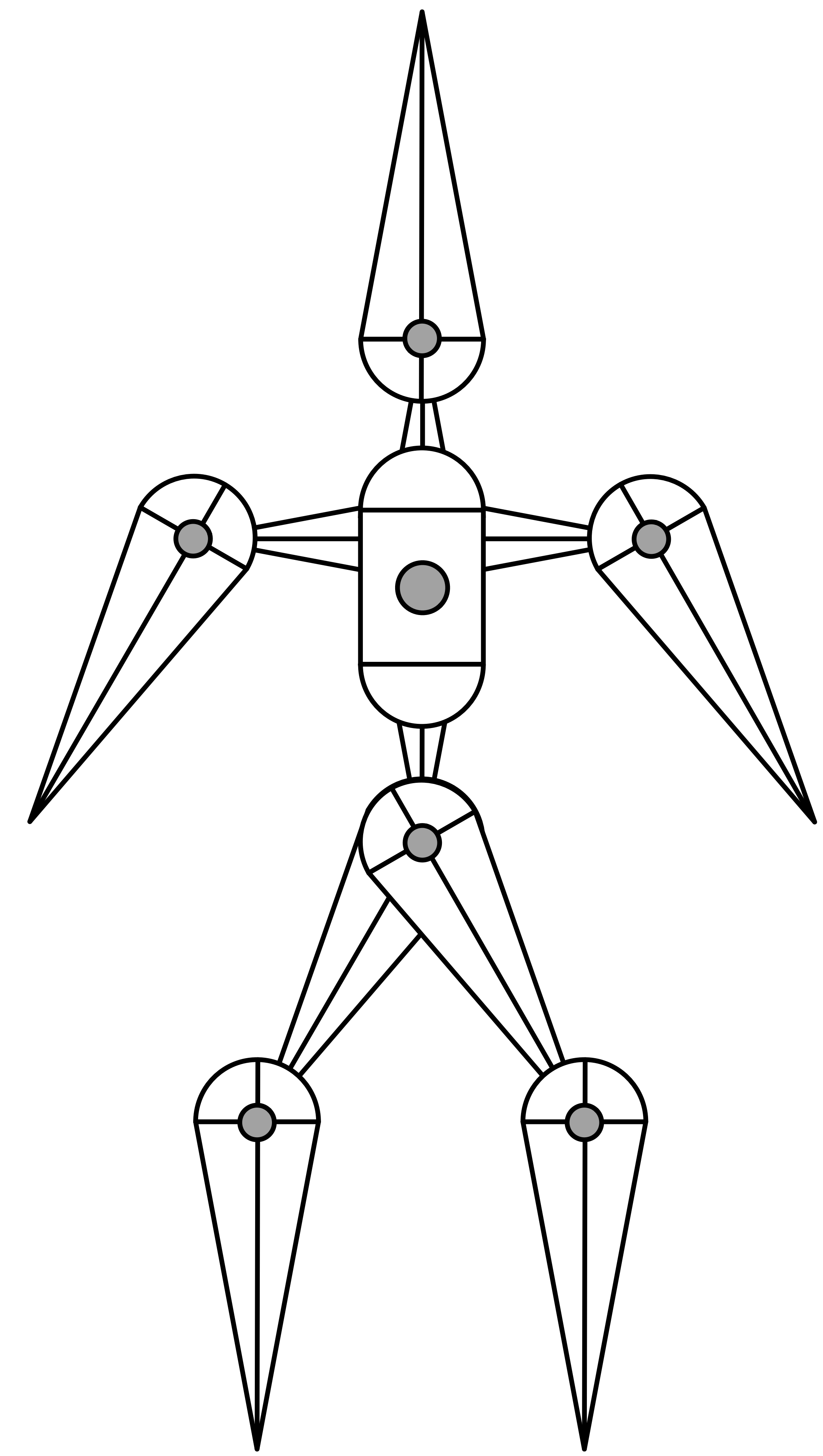
- **KINEMATICS: we are going to describe how objects move, without considering the underlying forces that generate that motion**



Forward kinematics

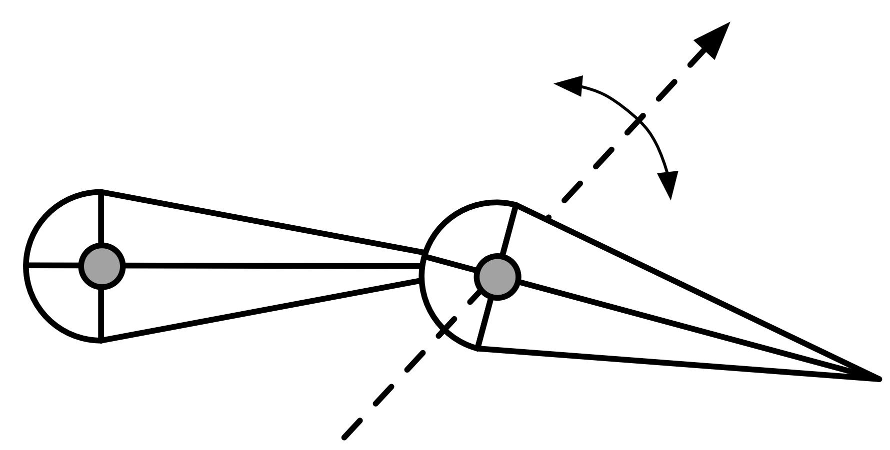
Articulated skeleton

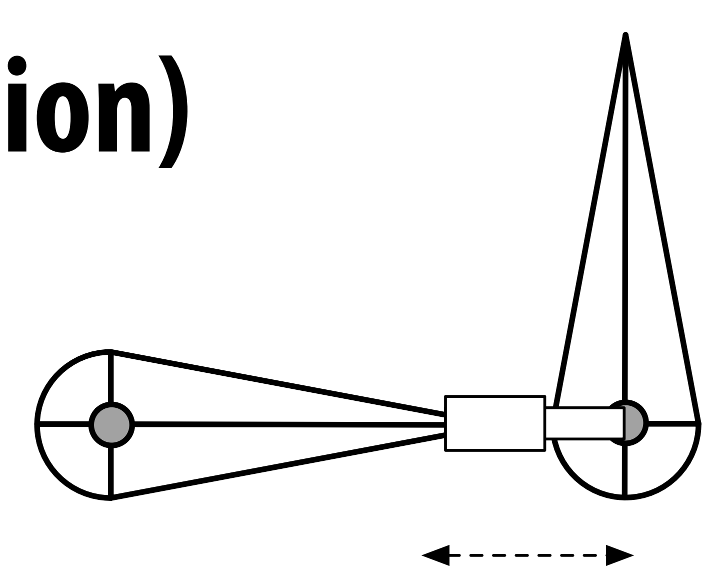
- Topology (what's connected to what)
- Geometric relations from joints
- Tree structure (in absence of loops)



Joint types

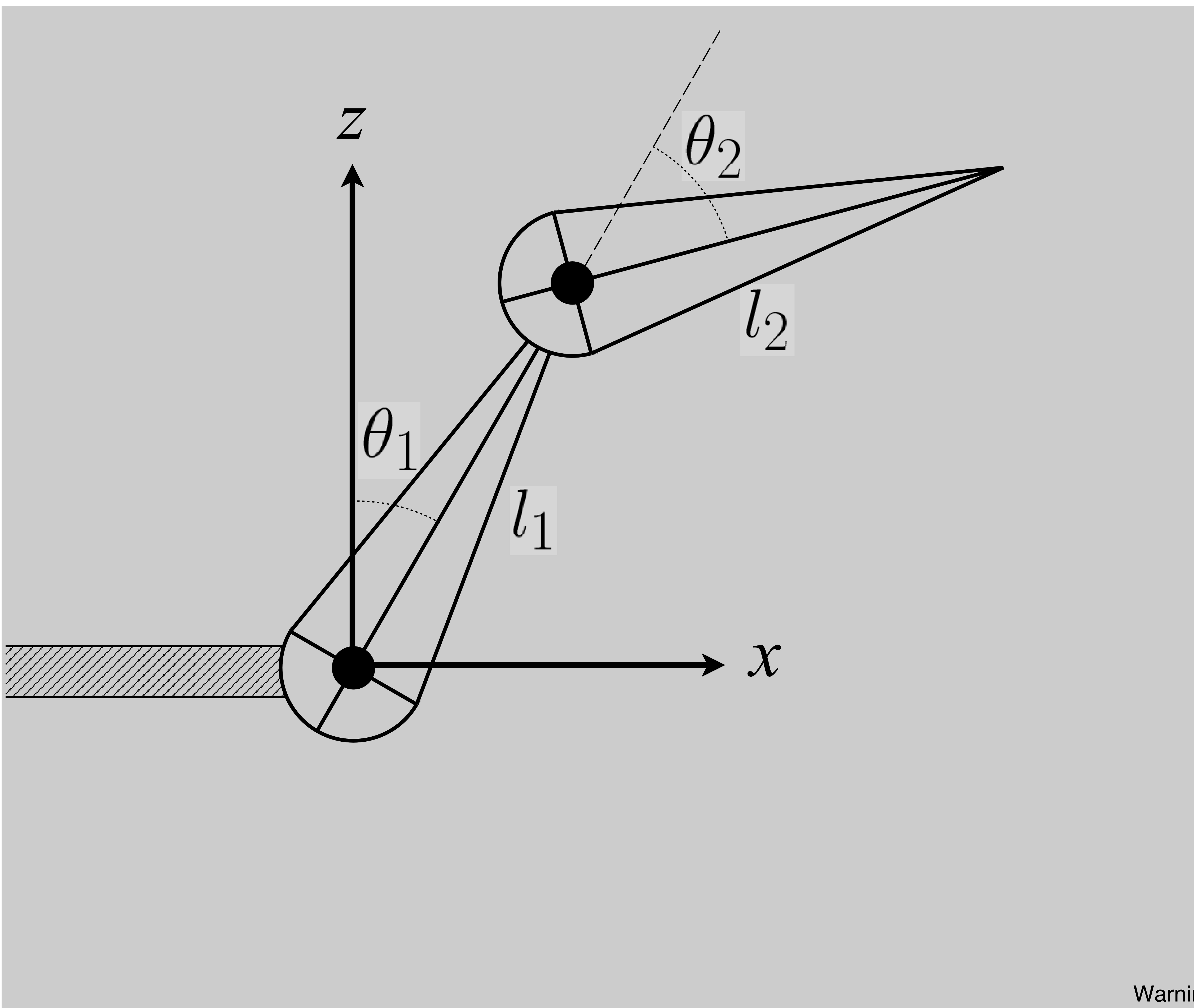
- Pin (1D rotation)
- Ball (2D rotation)
- Prismatic joint (translation)



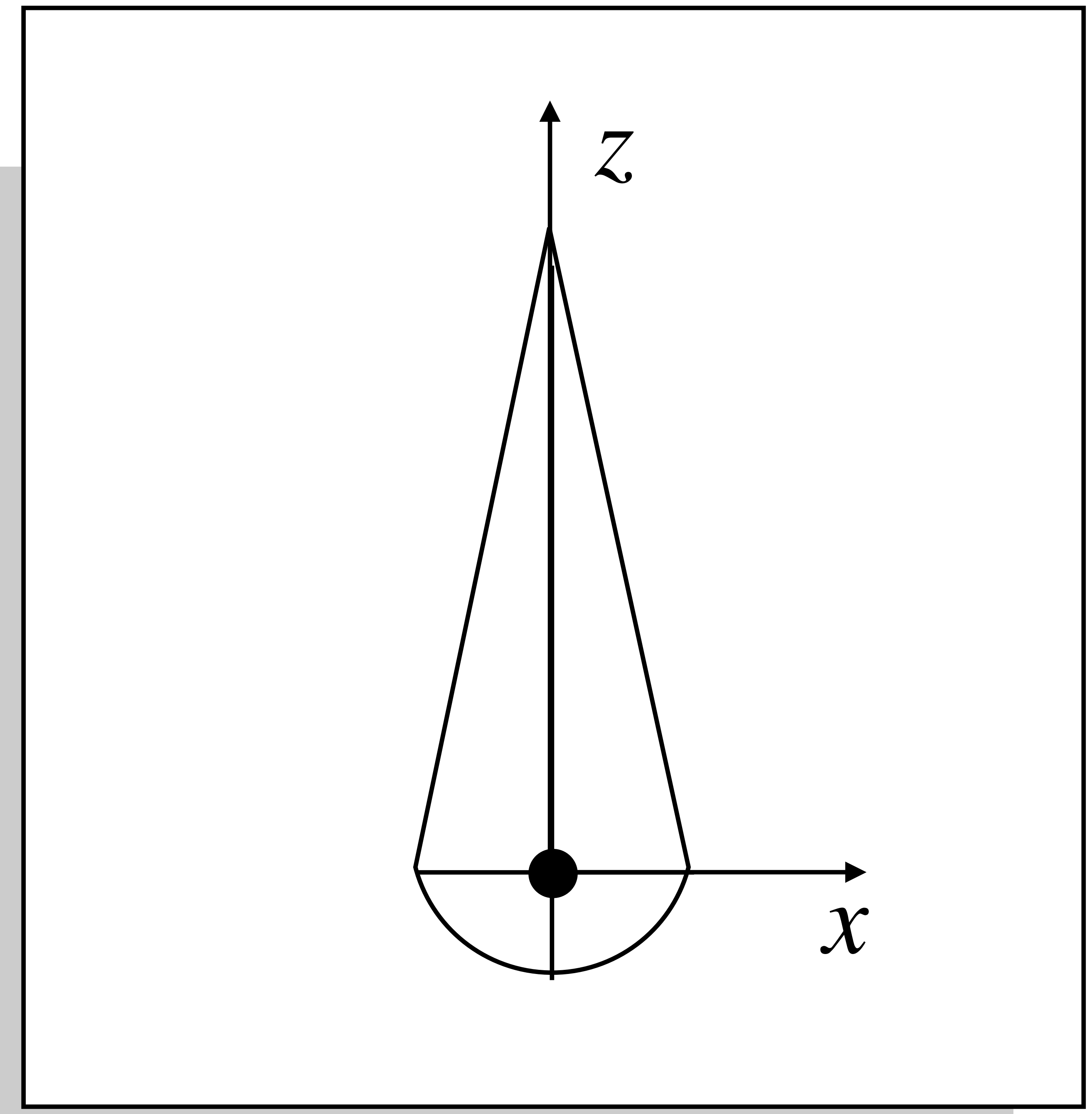


Forward kinematics

Example: simple two segment arm in 2D

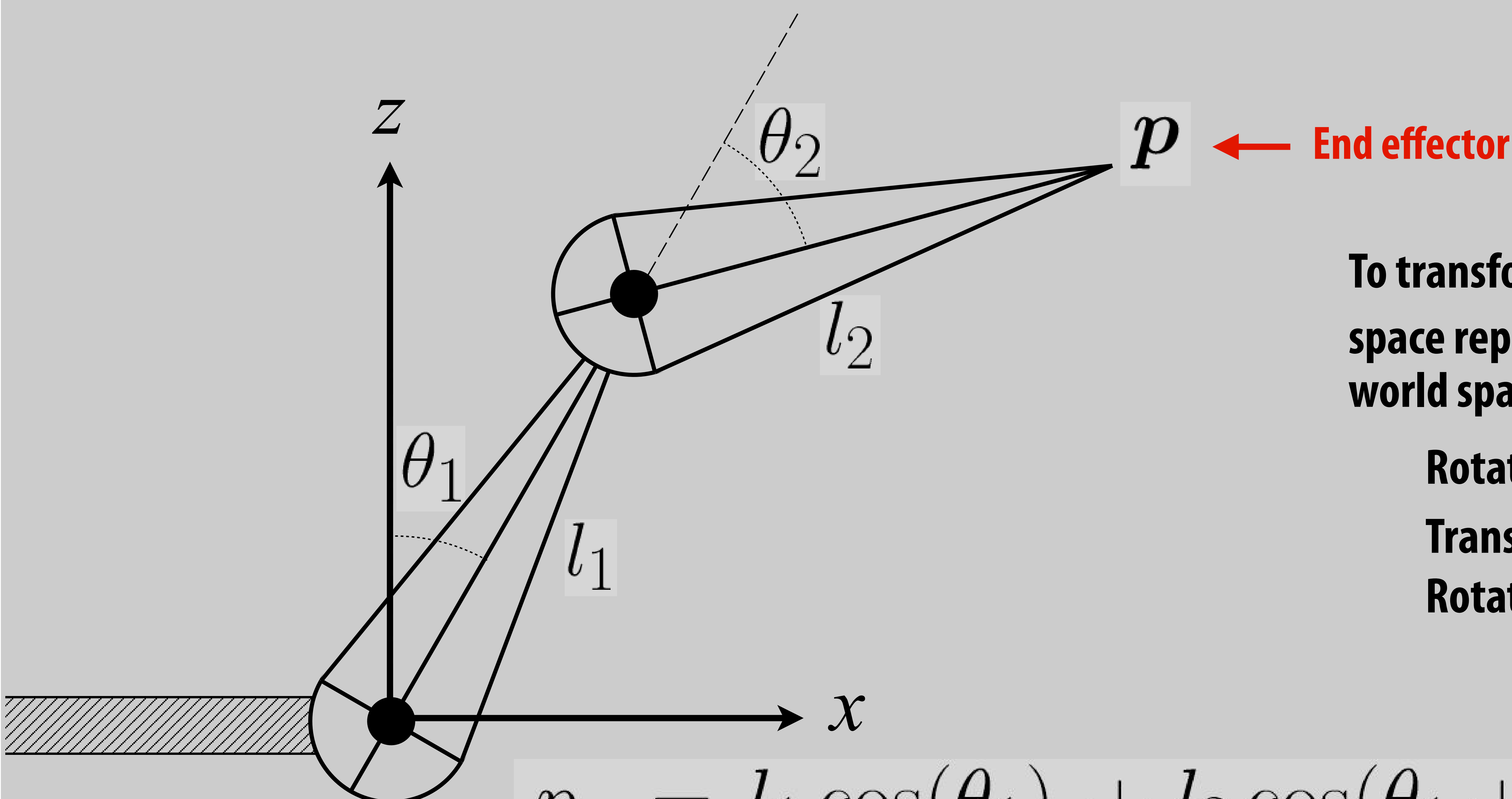


Object space position of part



Forward kinematics

Animator provides angles, and computer determines position *p* of end-effector



To transform point *p* with object space representation (0, l_2) into world space:

- **Rotate by θ_2**
- **Translate by (0, l_1)**
- **Rotate by θ_1**

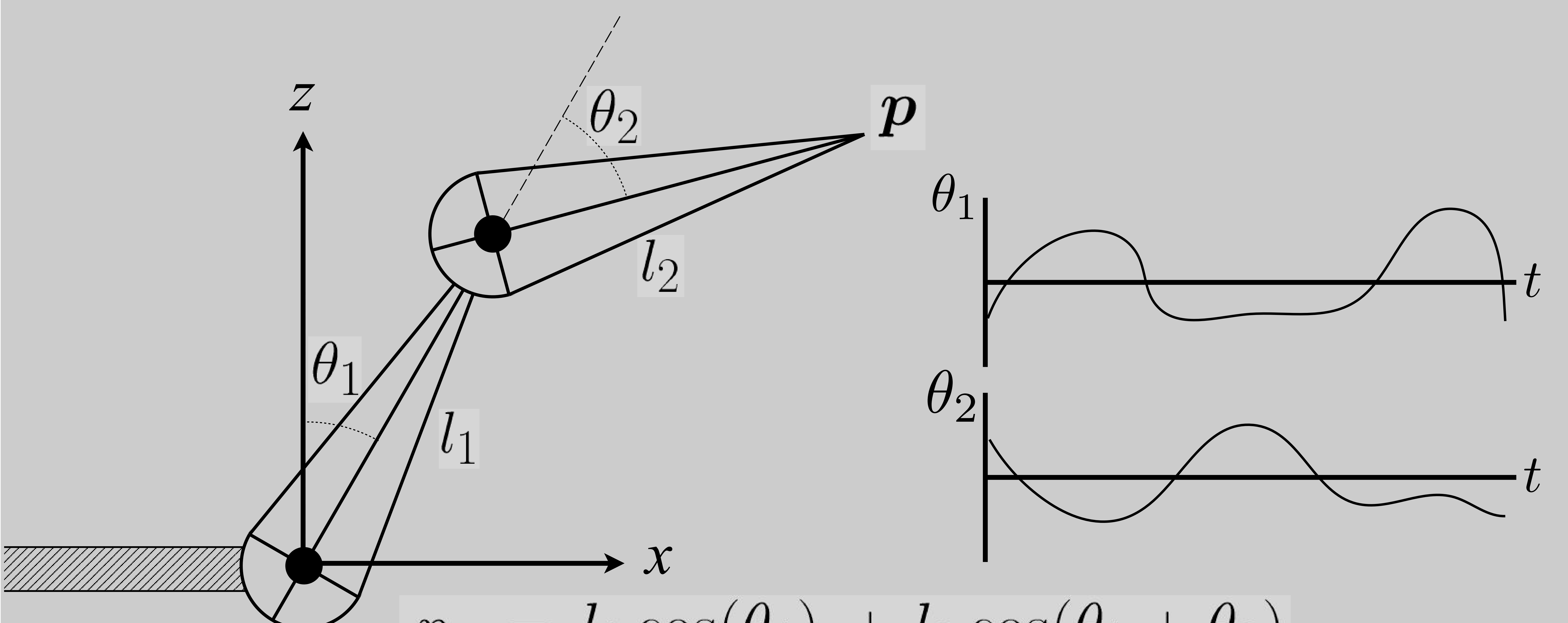
$$p_z = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2)$$

$$p_x = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2)$$

Warning: Z–up Coordinate System

Forward kinematics

Animation is described as angle parameter values as a function of time: $\theta_1(t), \theta_2(t)$



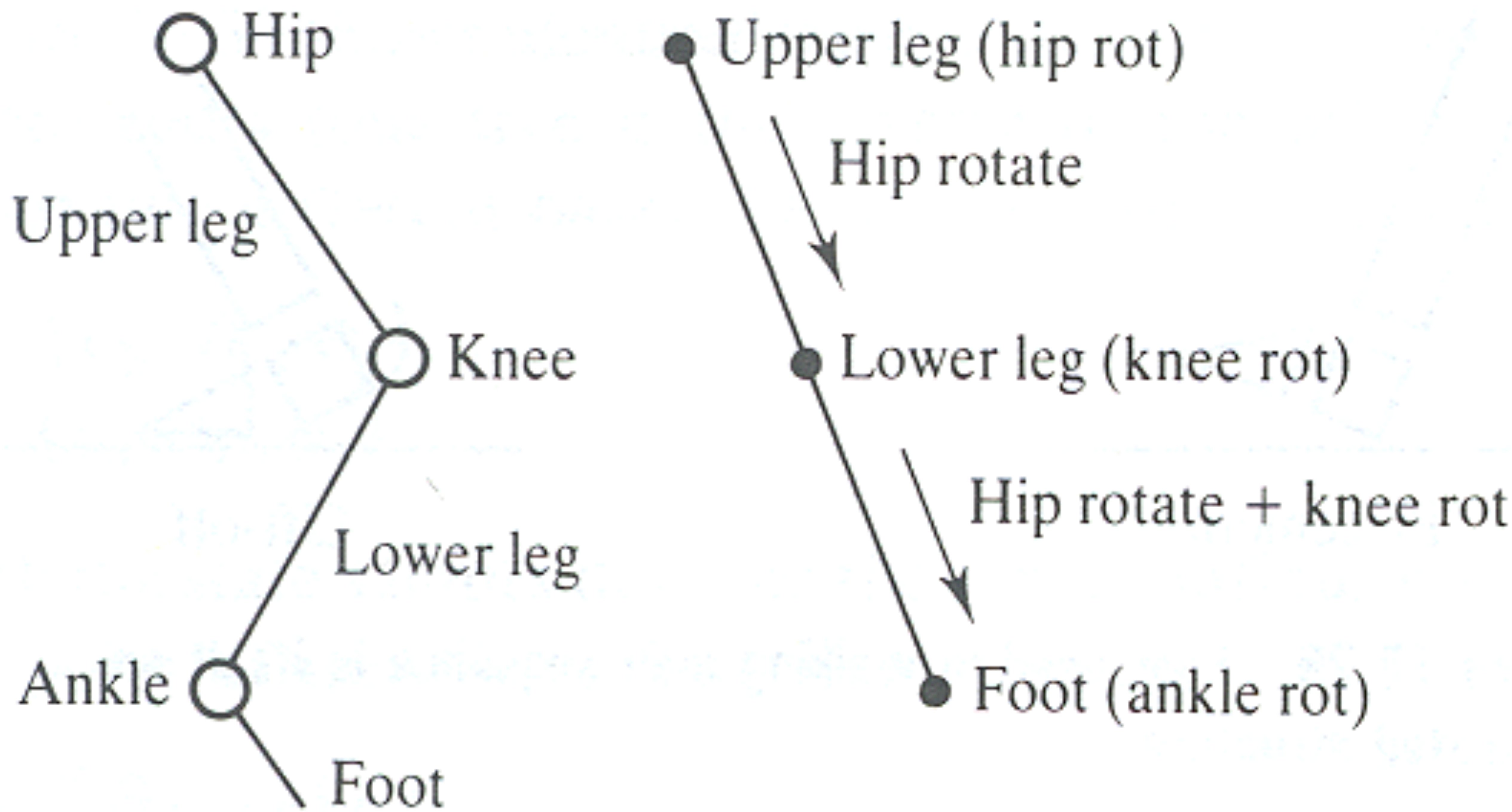
$$p_z = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2)$$

$$p_x = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2)$$

Warning: Z–up Coordinate System

Example: walk cycle

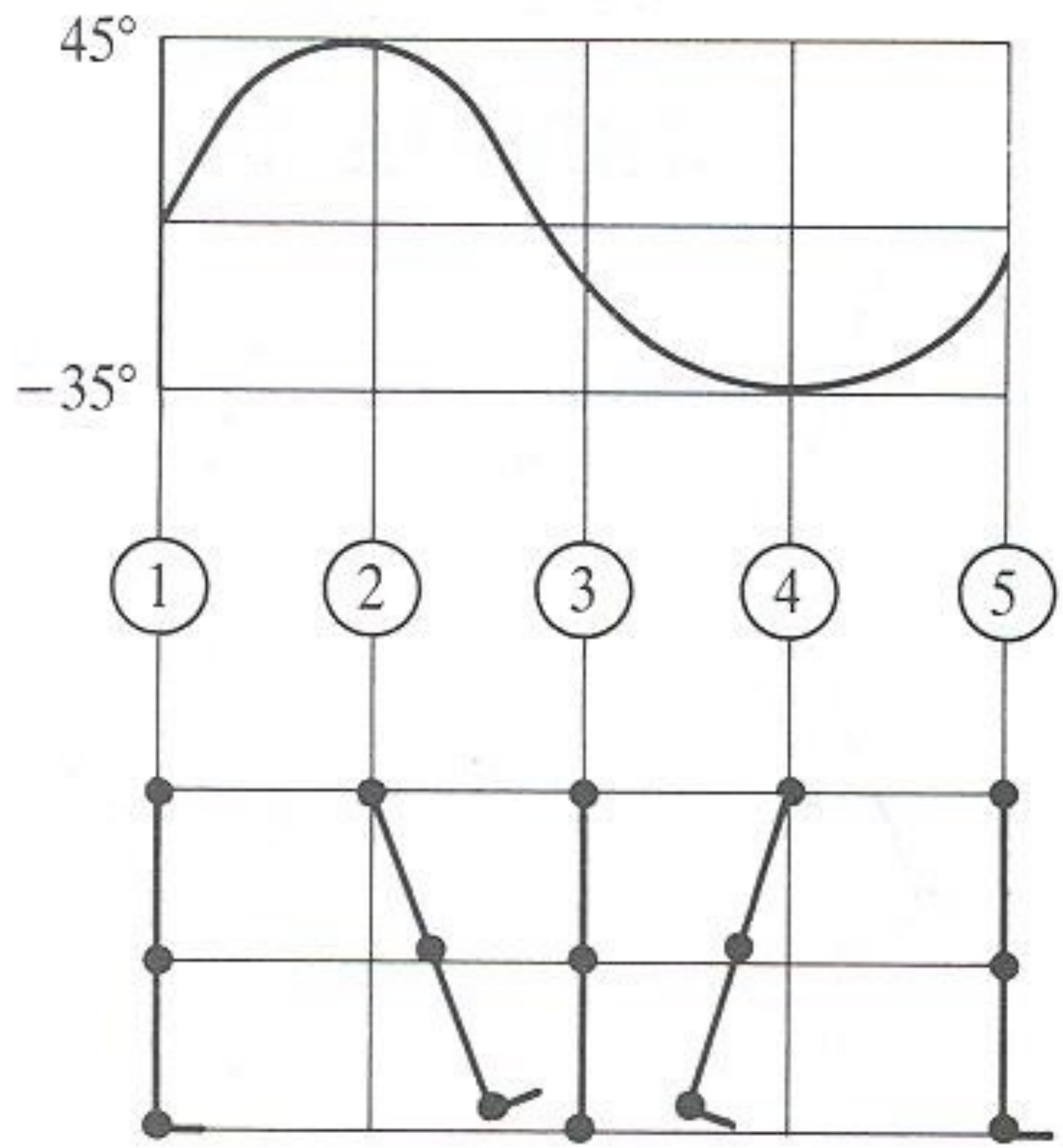
Articulated leg:



Watt & Watt

Example: walk cycle

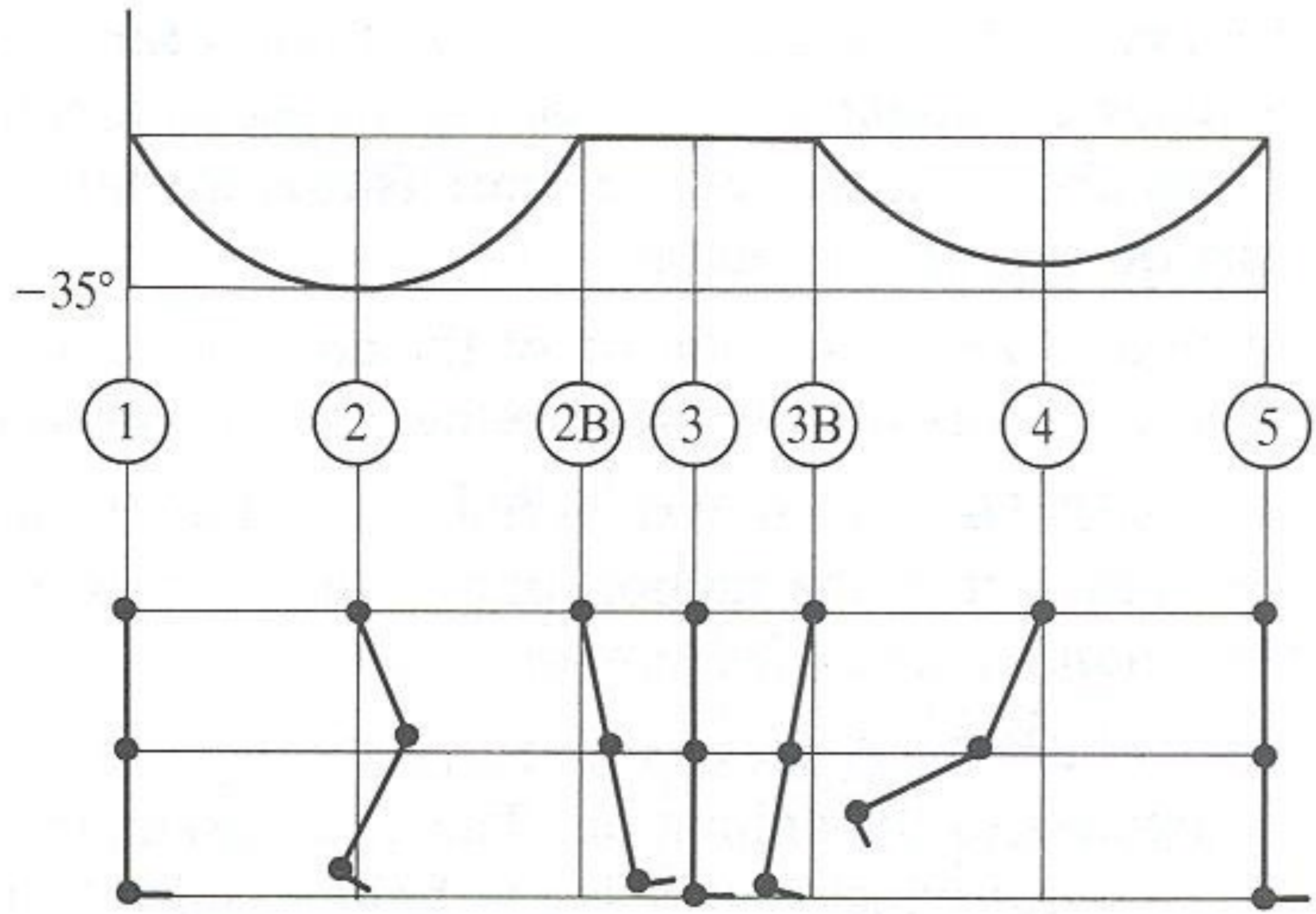
Hip joint angle



Watt & Watt

Example: walk cycle

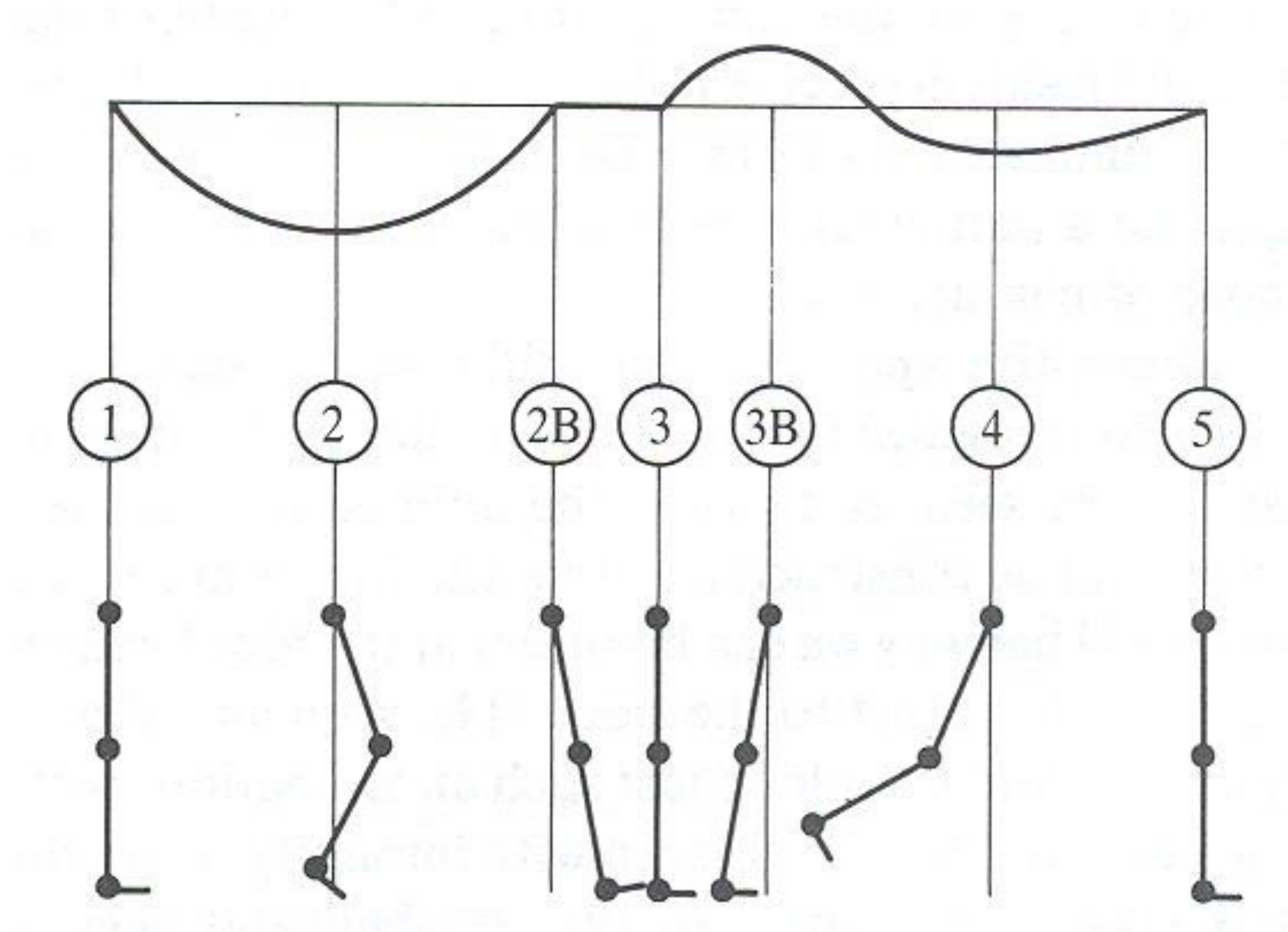
Knee joint angle



Watt & Watt

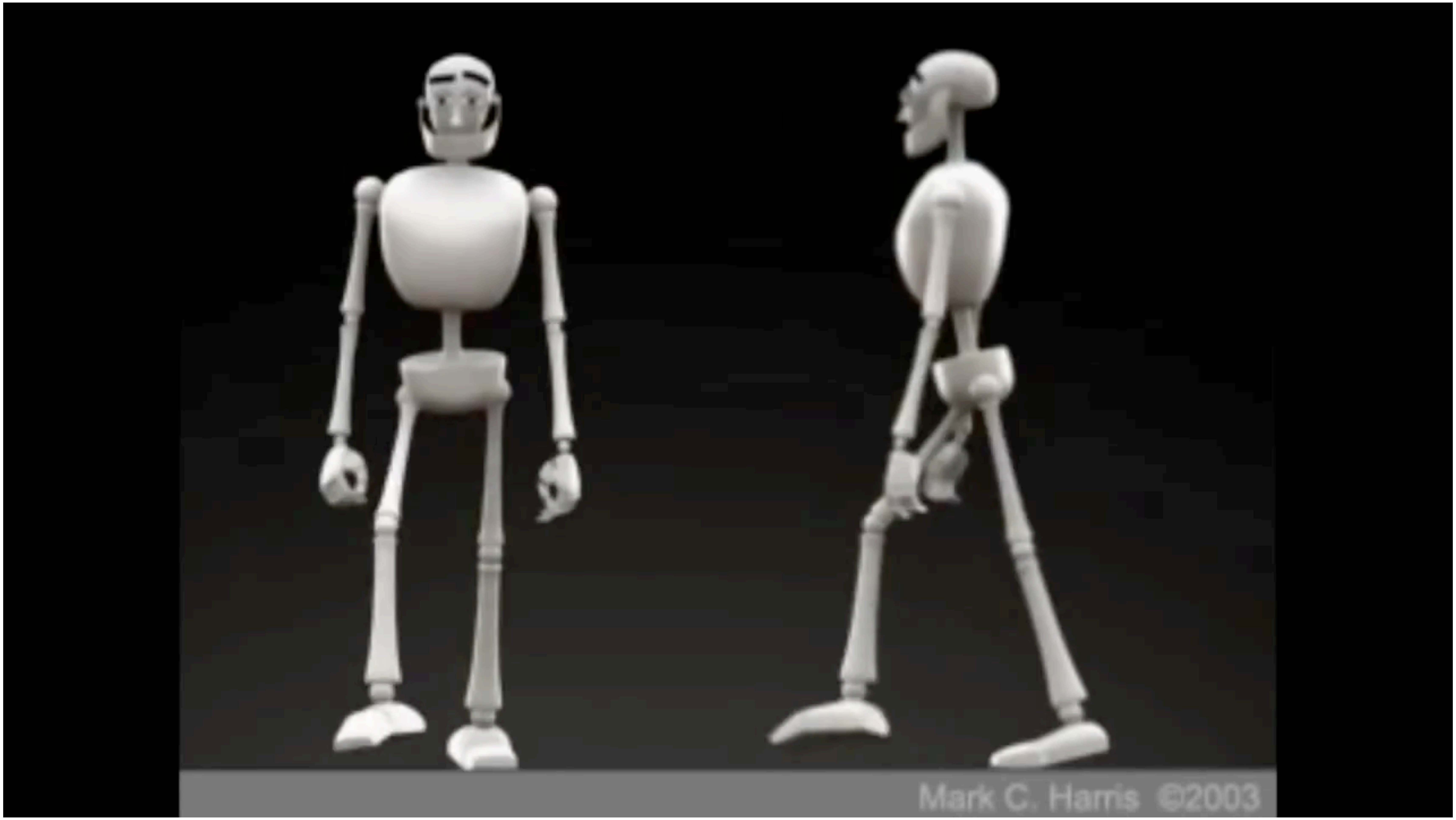
Example: walk cycle

Ankle joint angle

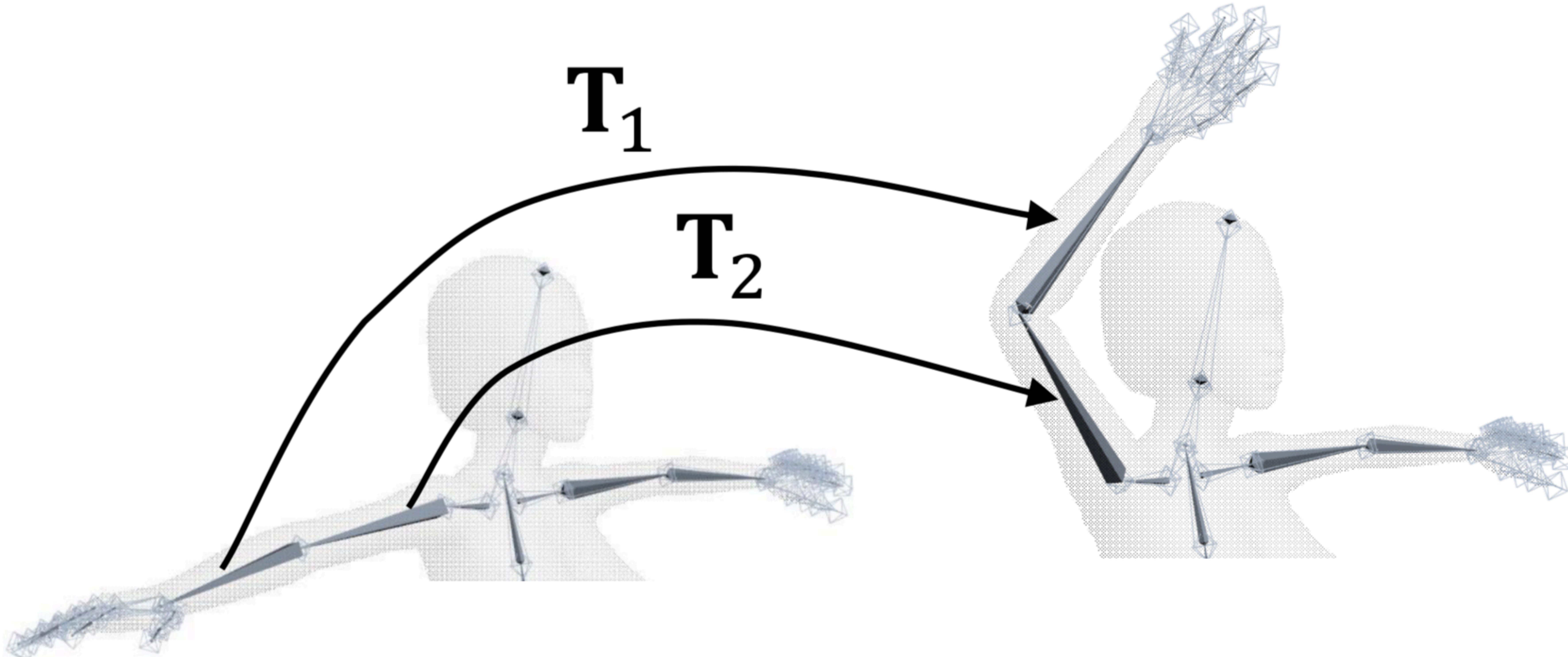


Watt & Watt

Example: walk cycle



Skinning: how to transform mesh vertices according to skeleton transforms



Skeleton joint transforms: T_1, T_2

Image credit: Ladislav Kavan

Vertex *i* on mesh

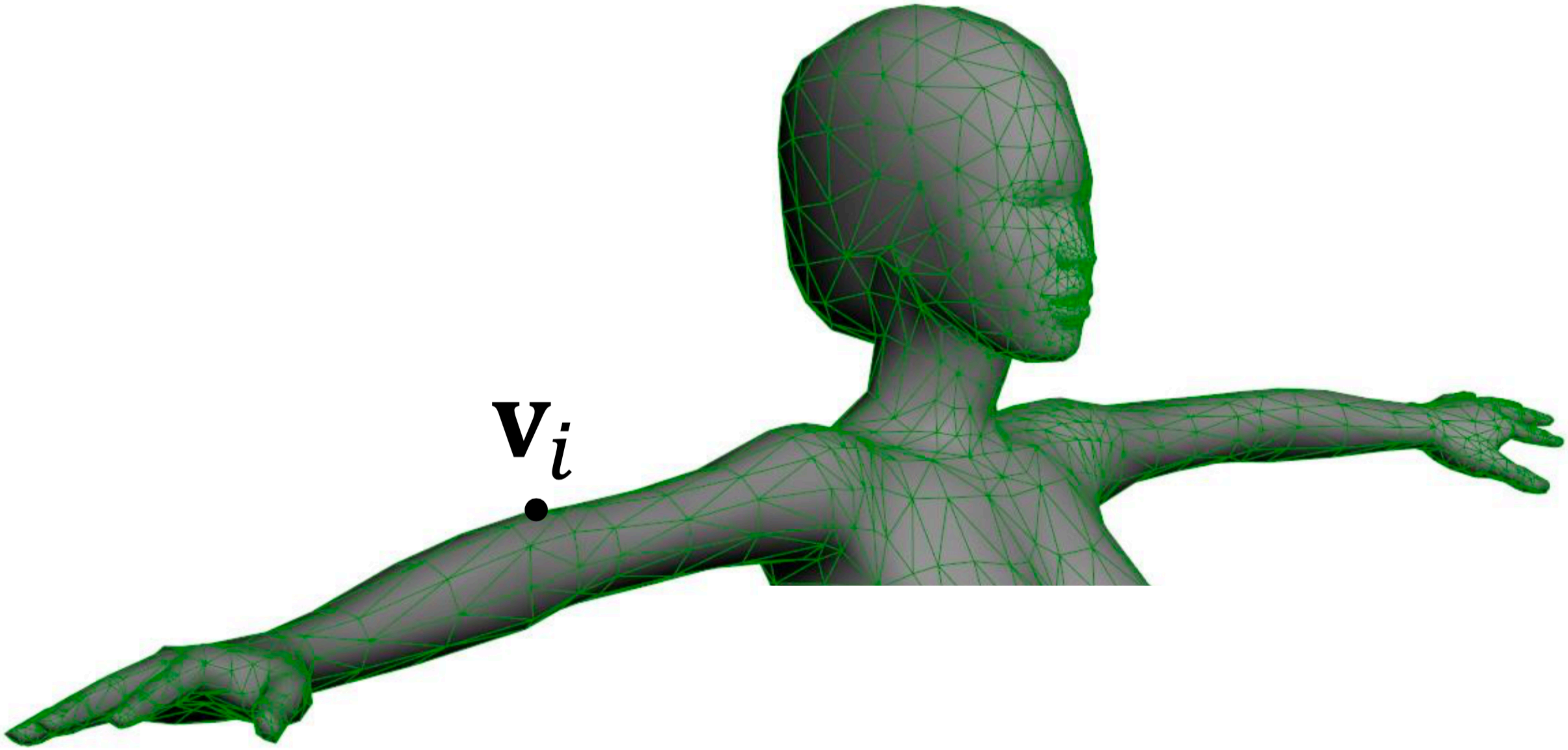
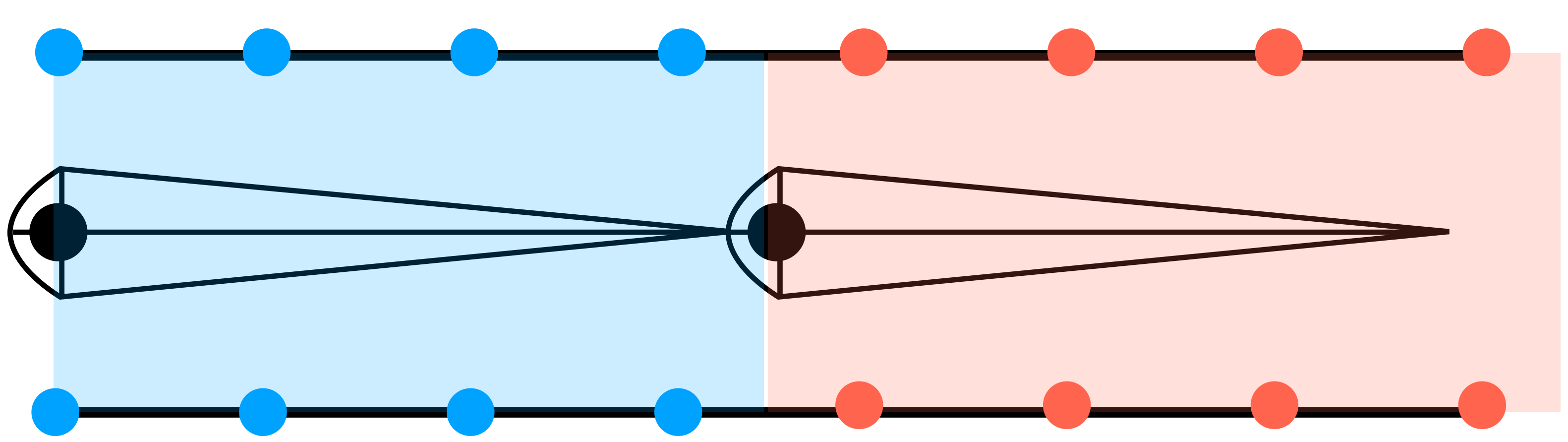


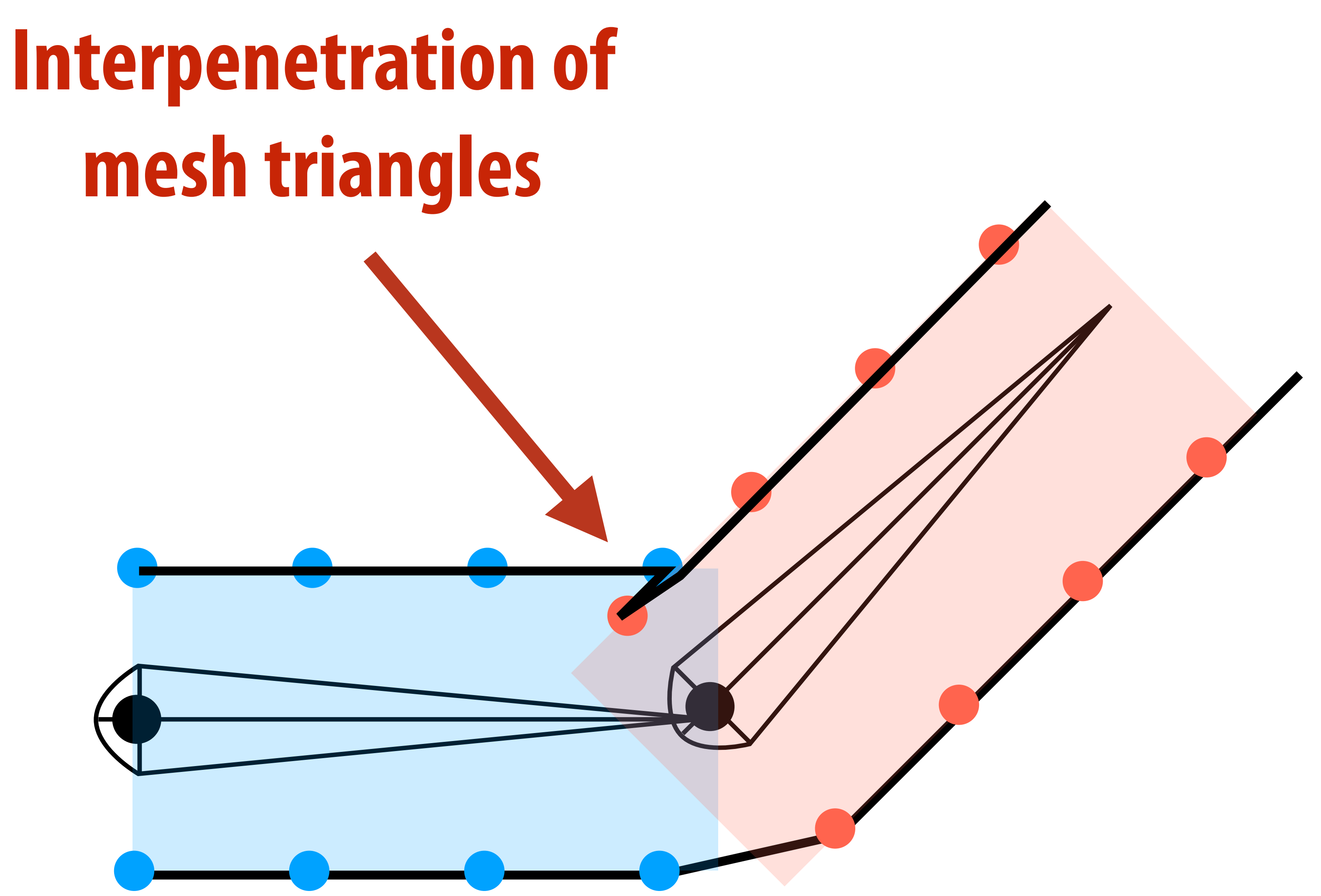
Image credit: Ladislav Kavan

Rigid body skinning

- One idea: transform mesh vertices according to transform for nearby skeleton joint



Original pose



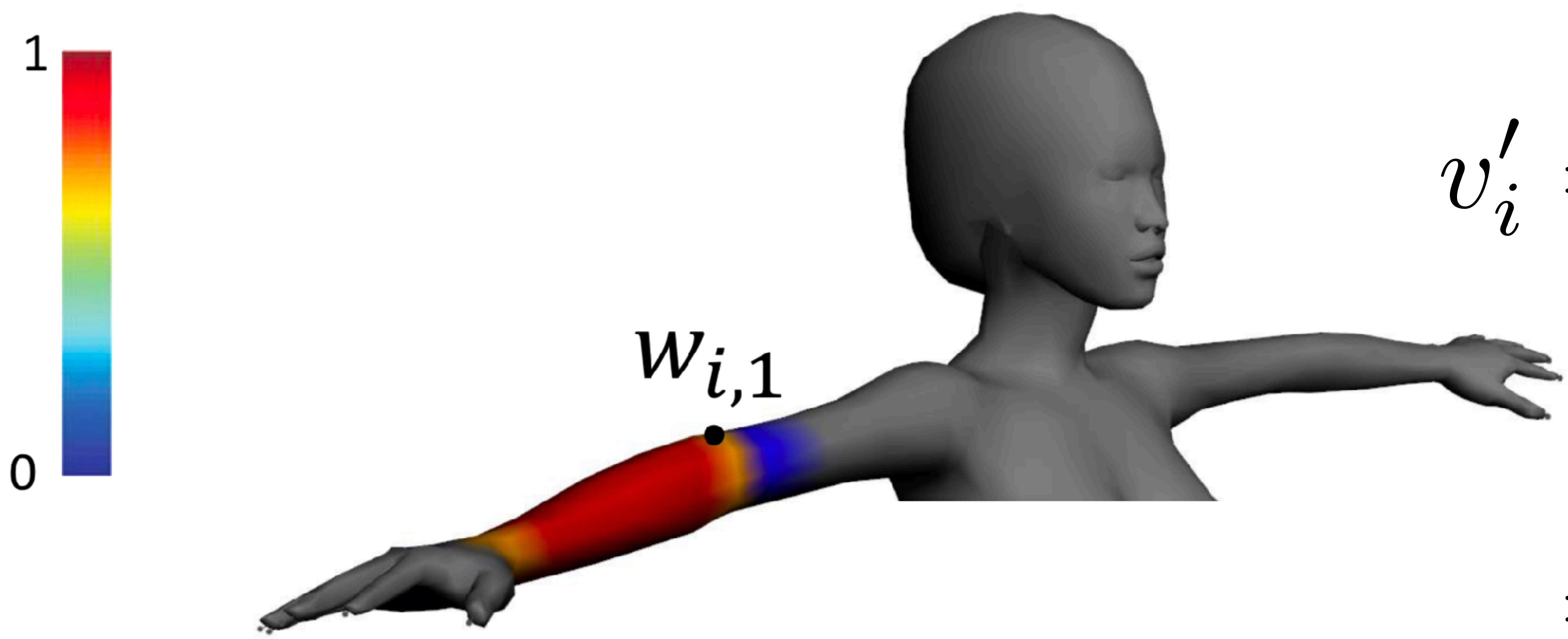
Vertices transforms according to corresponding joint transform (notice surface interpenetration)

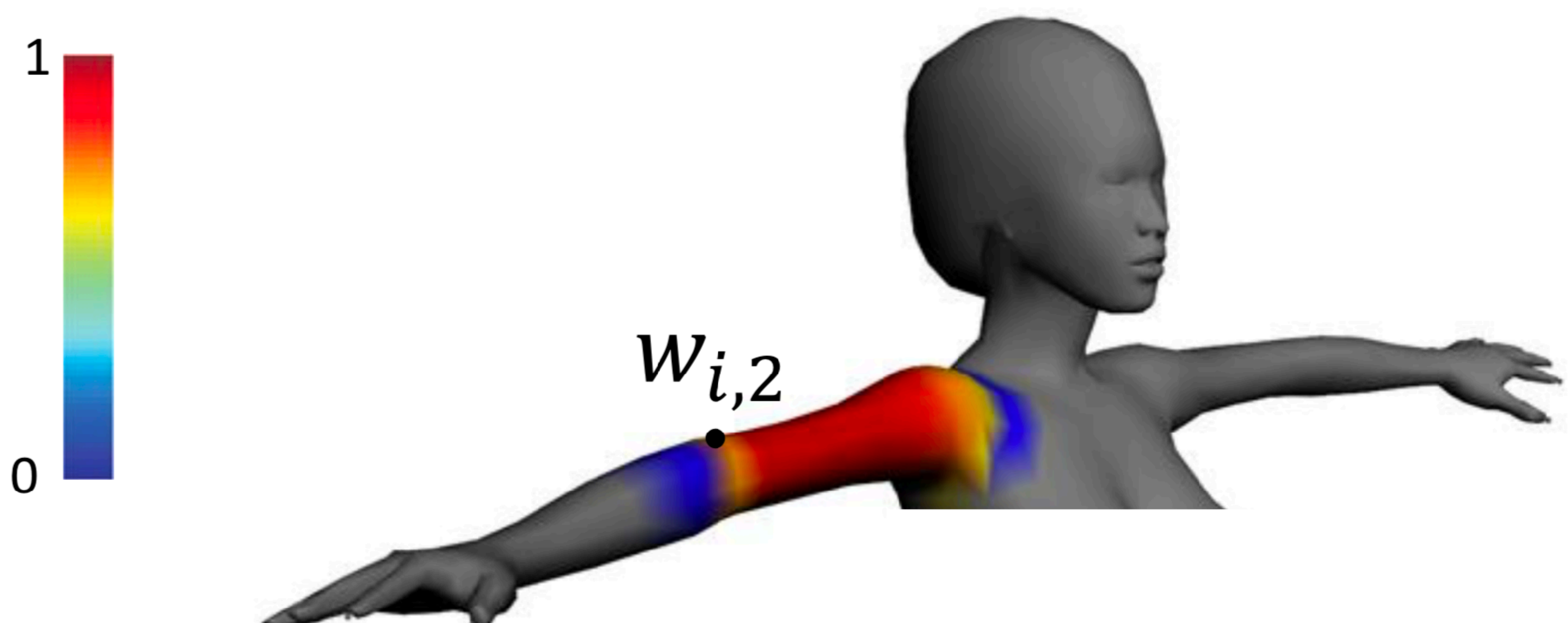
Blue verts = associated with first joint
Red verts = associated with second joint

Linear blend skinning *

Mesh vertices transformed by *linear combination* of nearby joint transforms

Very common technique for character animation in games





$$v_i' = \sum_j^N w_{ij} T_j v_i$$

$$= \left(\sum_j^N w_{ij} T_j \right) v_i$$

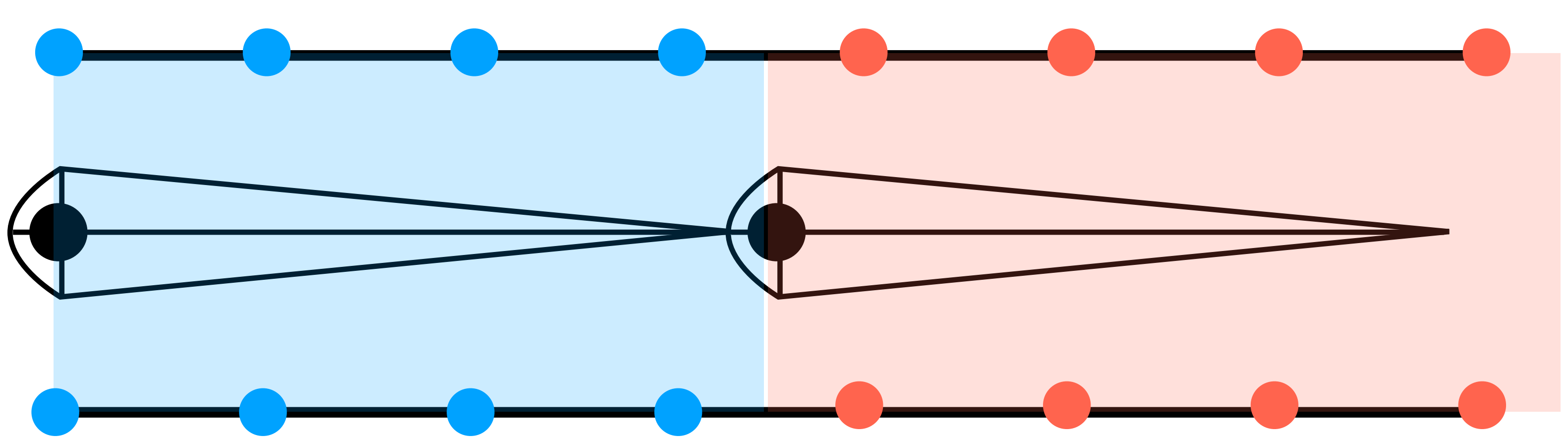
v_i = rest object space vertex position
T_j = transform for bone j
w_{ij} = weight of bone j on vertex i
N = number of bones

Image credit: Ladislav Kavan

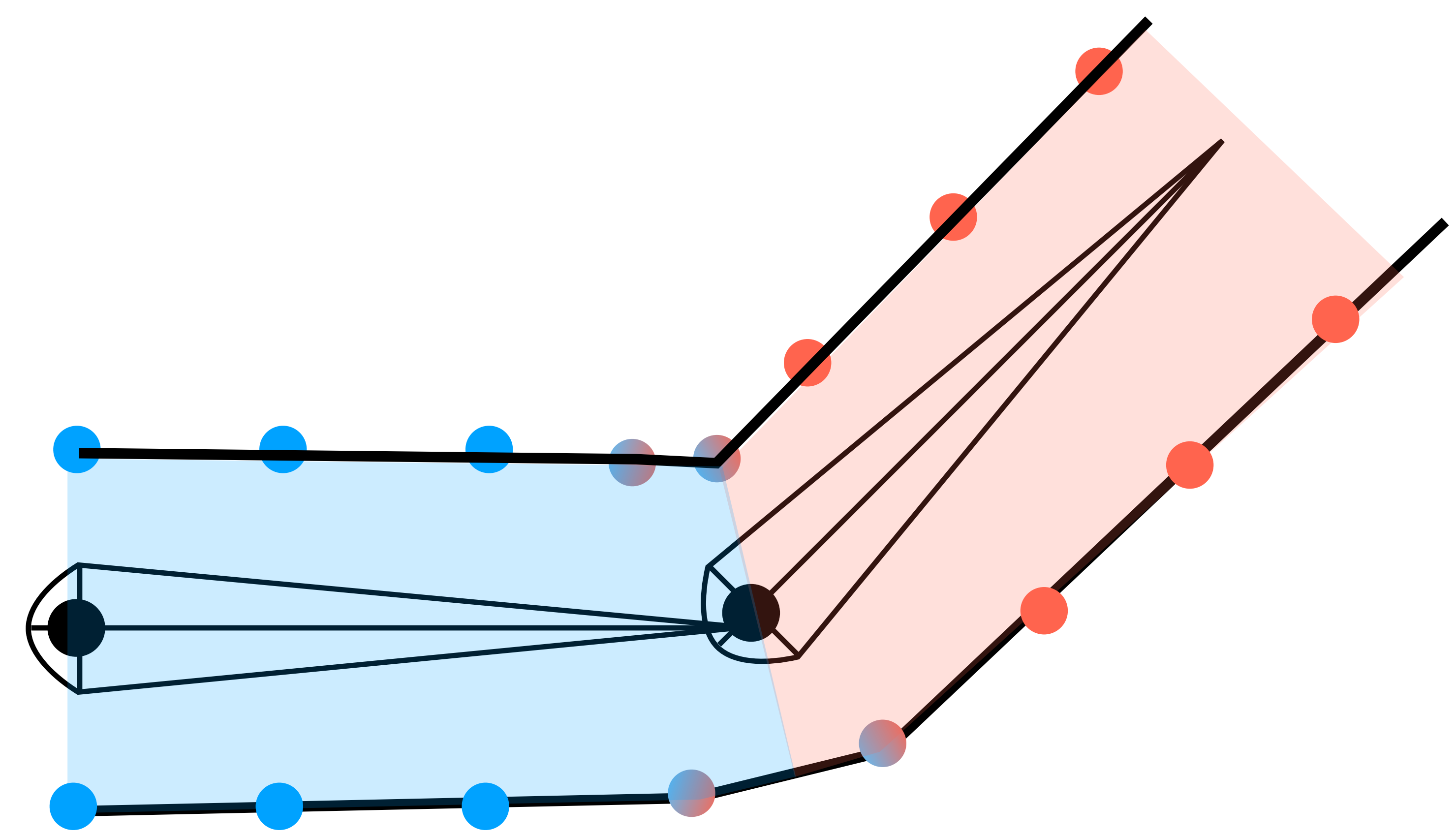
* Also called "matrix palette skinning" or "skeletal subspace deformation" (SSD)

Linear blend skinning

- Transform mesh vertices according to linear combination of transforms for nearby skeleton joint



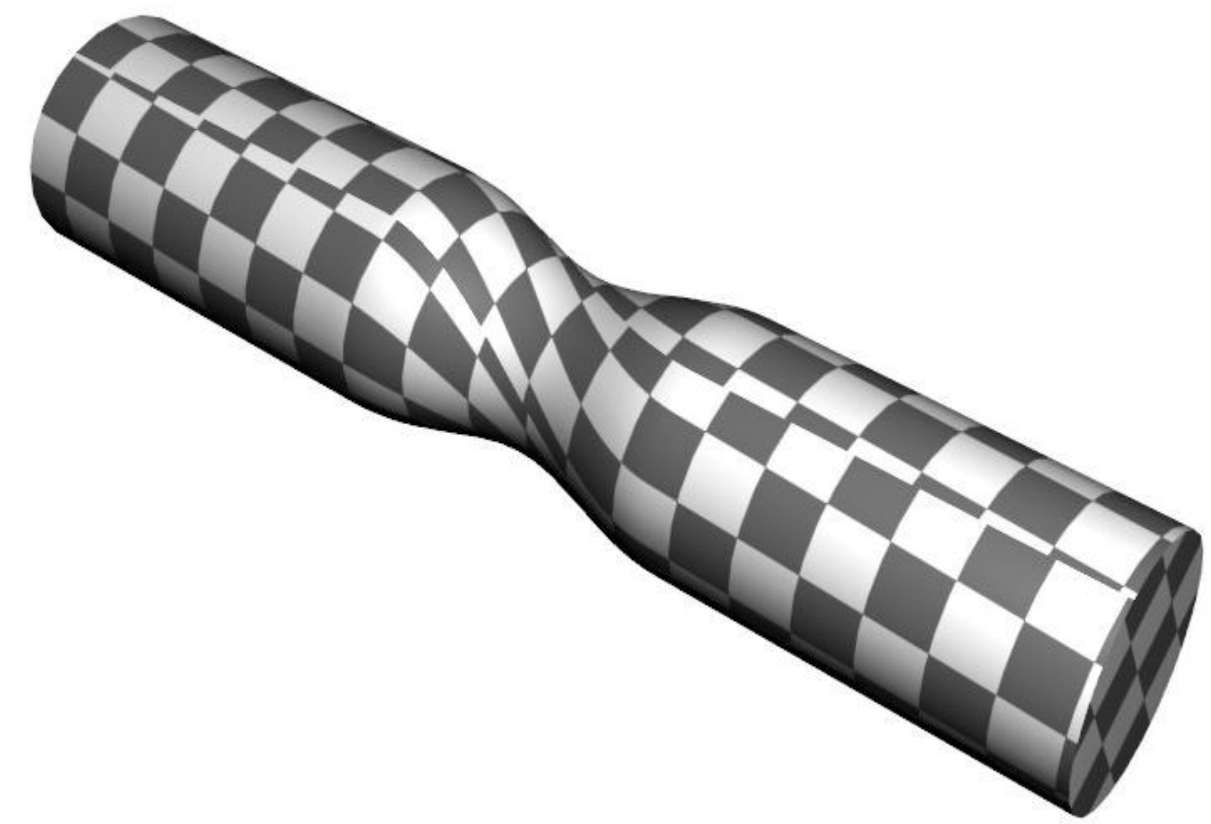
Original pose



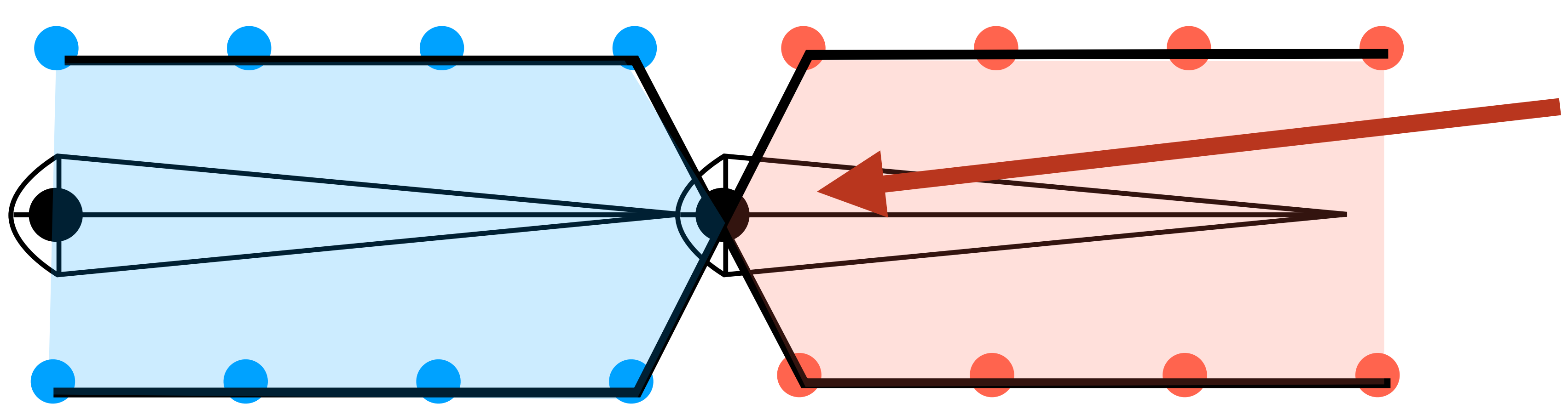
After transform

Shortcomings of linear blend skinning

- Loss of volume under large transformations



"candy wrapper effect"



Bone rotated 180 degrees radially

Many more advanced solutions in literature: dual-quaternion skinning, joint-based deformers, etc.

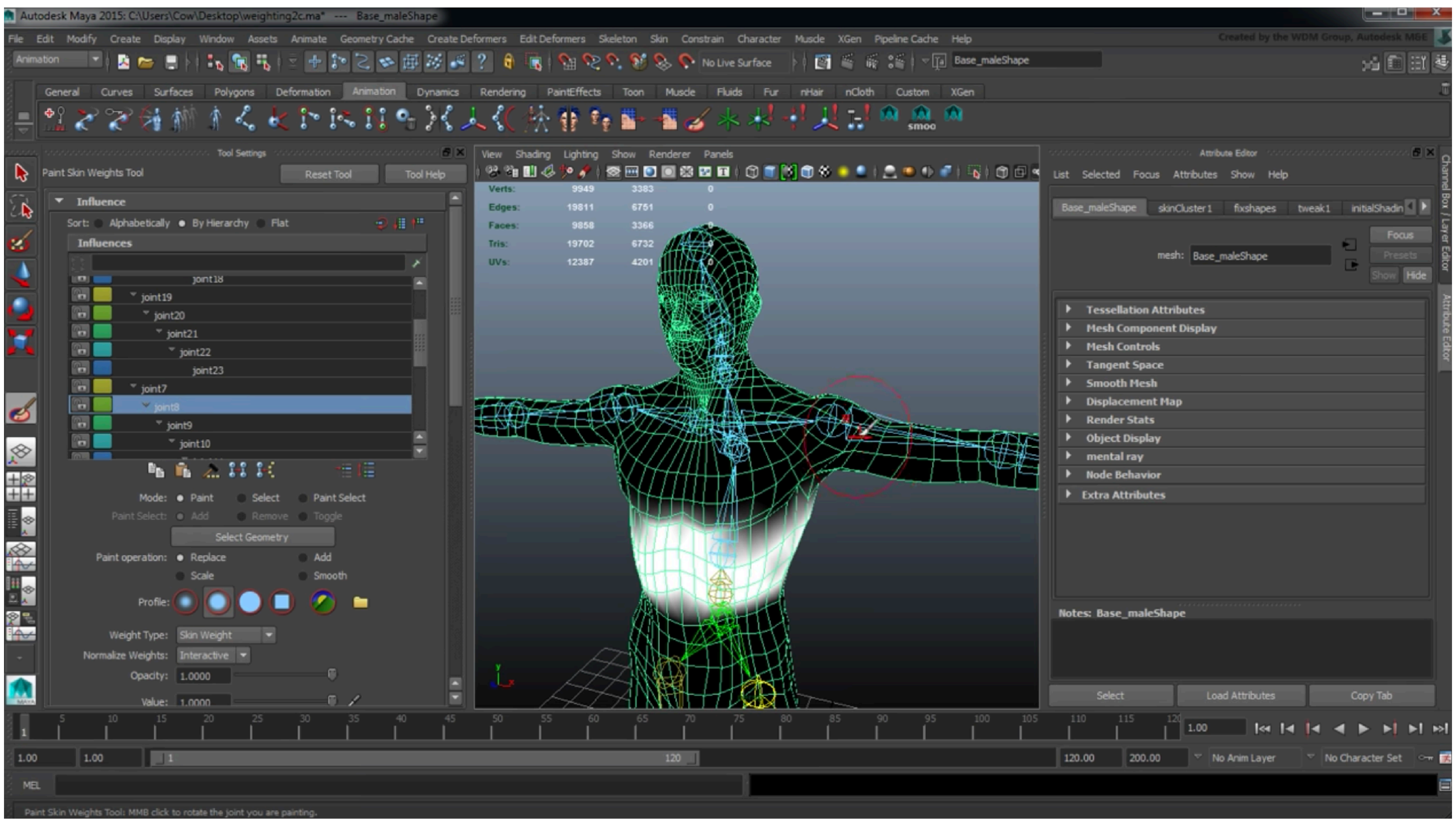
Image credit: Jacka et al.

Skinning example

Courtesy Matthew Lailler via Keenan Crane via Ren Ng

Rigging

- "Rigging" is the process of attaching a set of animation controls to a mesh
  - In the case of linear blend skinning: it is attaching a skeleton to the mesh (e.g., setting per vertex blend weights)



Example: artist painting vertex blend weights directly on mesh in Maya

Different ways to obtain joint angles

- Hand animate values (as discussed above)
- Measure angles from a performance via motion capture
- Solve for angles based on higher-level goal (optimization)

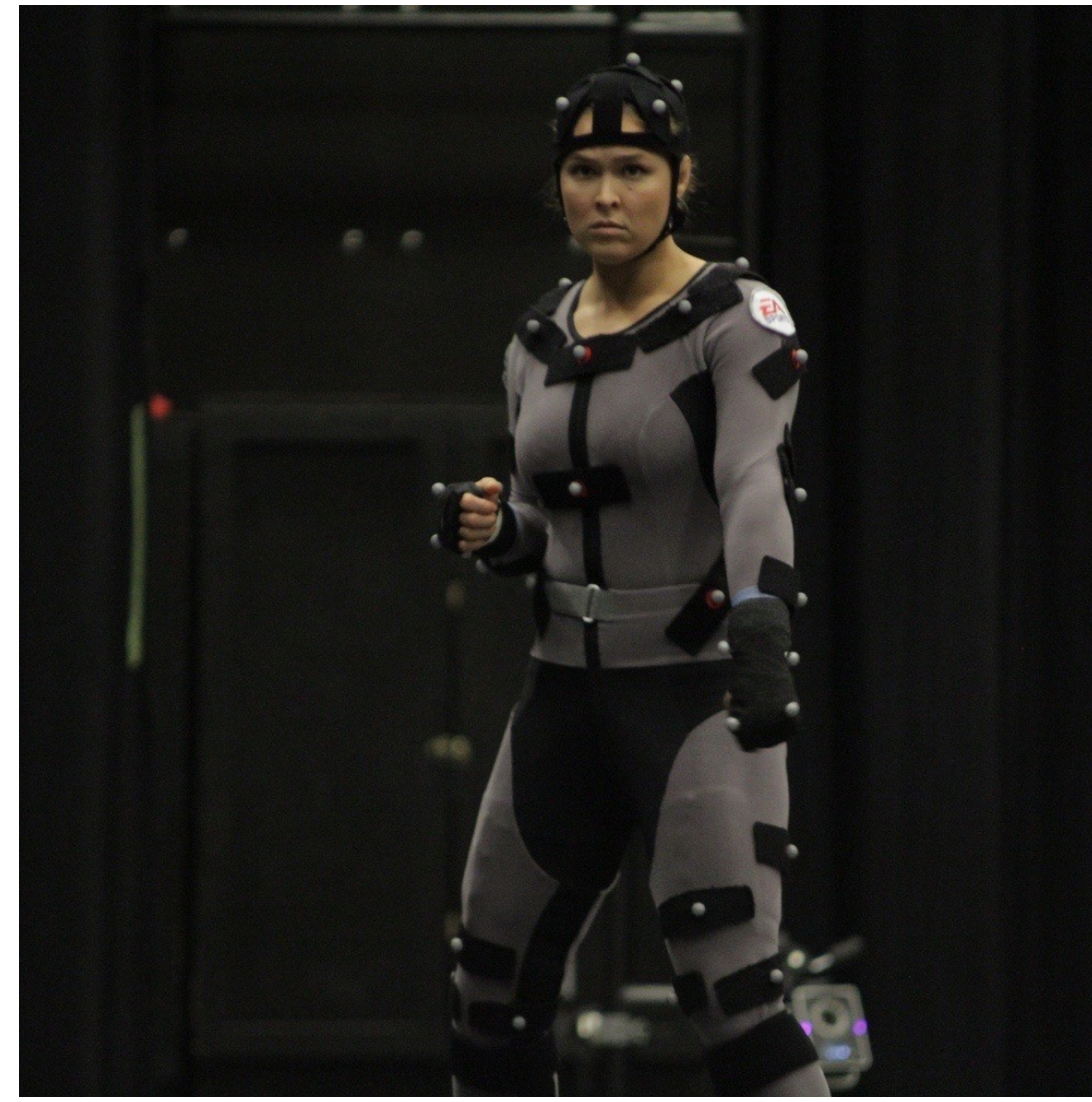
Motion Capture

Motion capture

- Data-driven approach to creating animation sequences
  - Record real-world performances (e.g. person executing an activity)
  - Extract pose as a function of time from the data collected

Motion capture room for ShaqFu

Optical motion capture

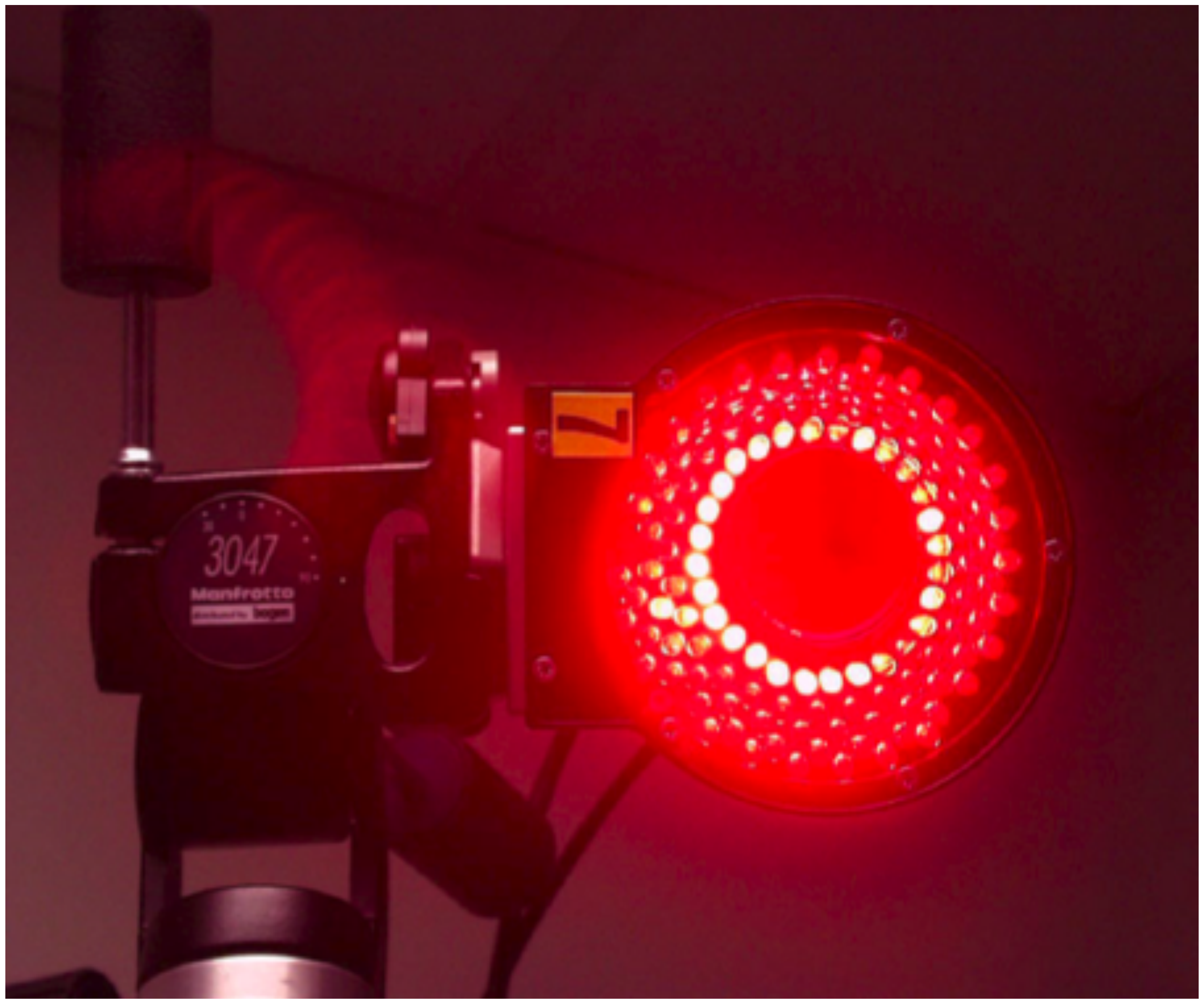


Source: http://fightland.vice.com/blog/ronda-rousey-20-the-queen-of-all-media

Ronda Rousey in Electronic Arts' motion capture studio

Optical motion capture

Retroreflective markers attached to subject



IR illumination and cameras

- Affix markers to joints of subject
- Compute 3D positions by triangulation from multiple cameras
- 8+ cameras, 240 Hz, occlusions are difficult

Slide credit: Steve Marschner

Motion capture pros and cons

- **Strengths**
  - Can capture large amounts of real data quickly
  - Realism can be high
- **Weaknesses**
  - Complex and costly set-ups (but progress in computer vision is changing this)
  - Captured animation may not meet artistic needs, requiring alterations

Challenges of facial animation

- "Uncanny valley"
  - In robotics and graphics
  - As artificial character appearance approaches human realism, our emotional response goes negative, until it achieves a sufficiently convincing level of realism in expression

Cartoon.
Brave, Pixar

Semi-realistic. Polar Express, Warner Bros

Challenges of facial motion capture

Final Fantasy Spirits Within

Facial motion capture

Discovery, "Avatar: Motion Capture Mirrors Emotions", https://youtu.be/1wK1Ixr-UmM

Aside: lower-cost forms of capture

Microsoft XBox 360 Kinect

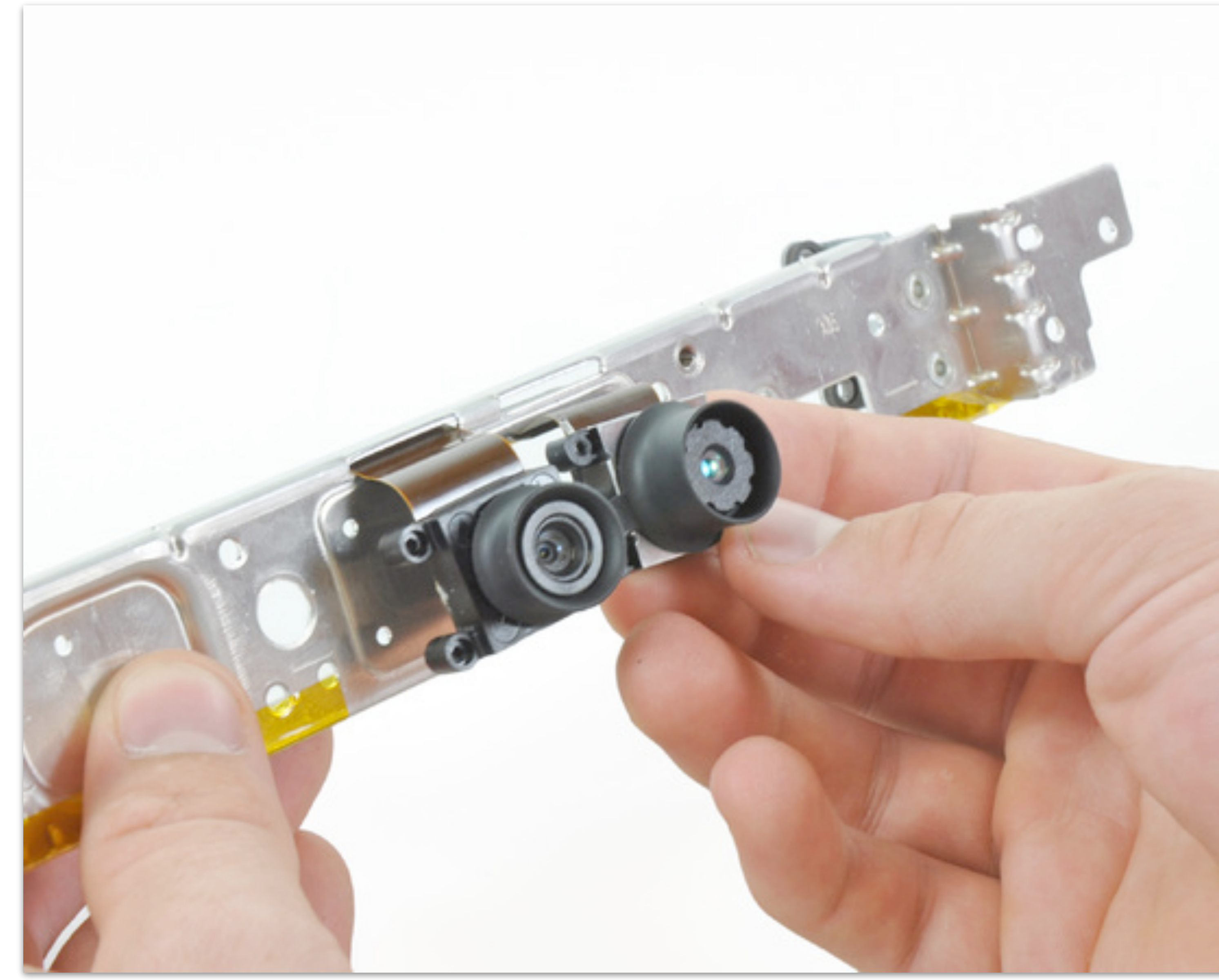


Image credit: iFixIt

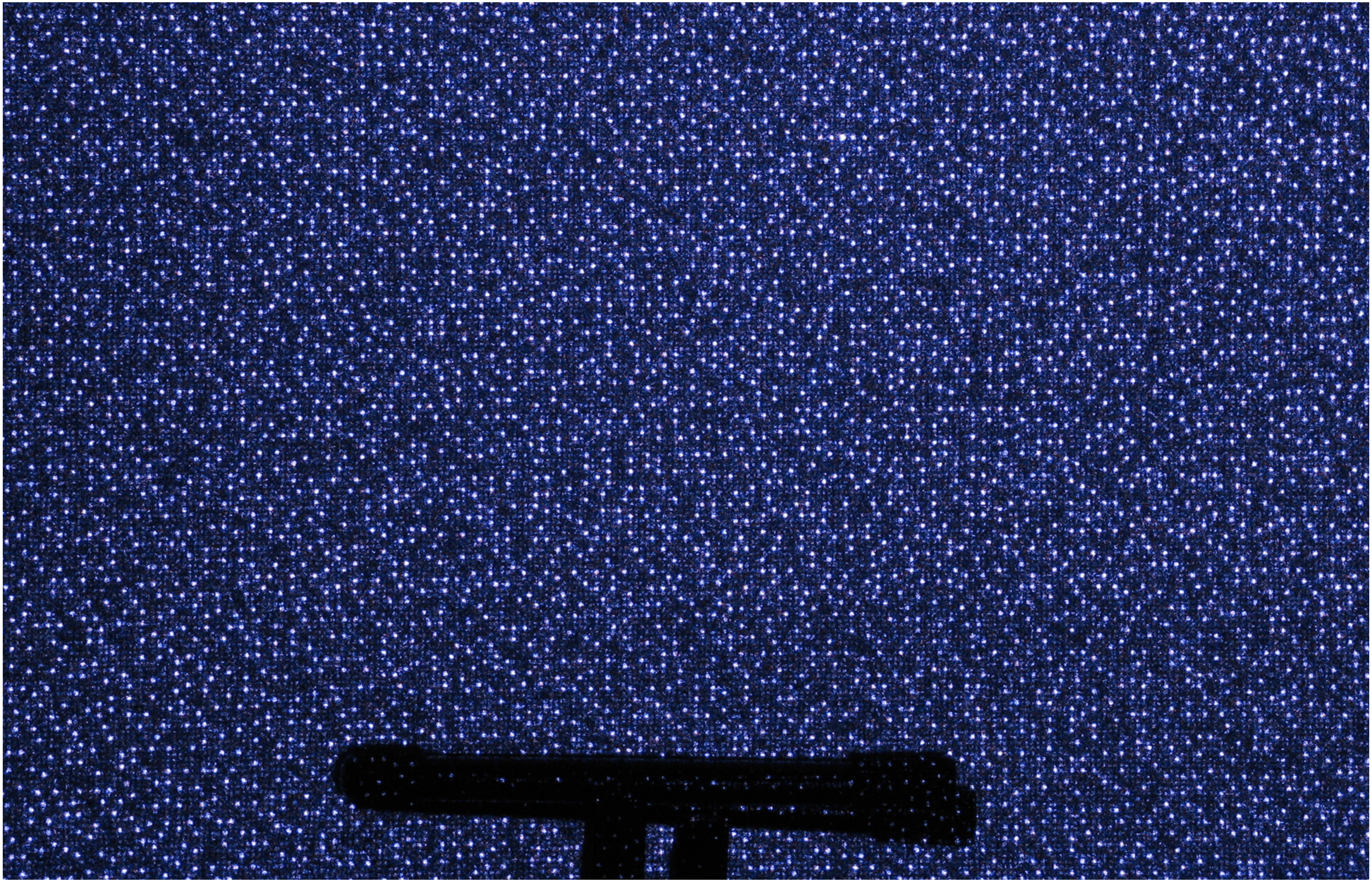
Illuminant
(Infrared Laser + diffuser)

RGB CMOS Sensor
640x480 (w/ Bayer mosaic)

Monochrome Infrared
CMOS Sensor
(Aptina MT9M001)
1280x1024 **

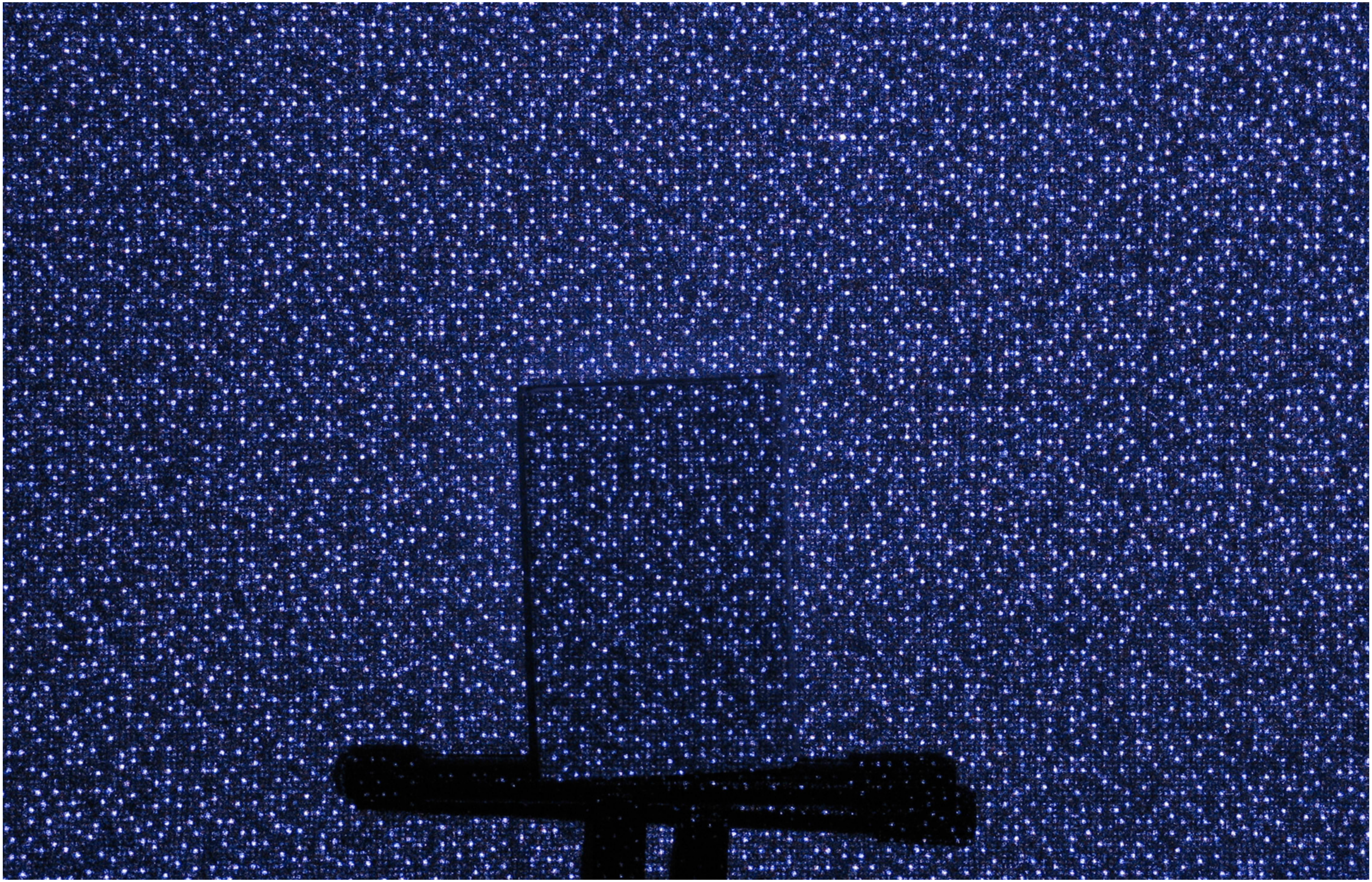
** Kinect returns 640x480 disparity image, suspect sensor is configured for 2x2 pixel binning down to 640x512, then crop

Infrared image of Kinect illuminant output



Credit: www.futurepicture.org

Infrared image of Kinect illuminant output



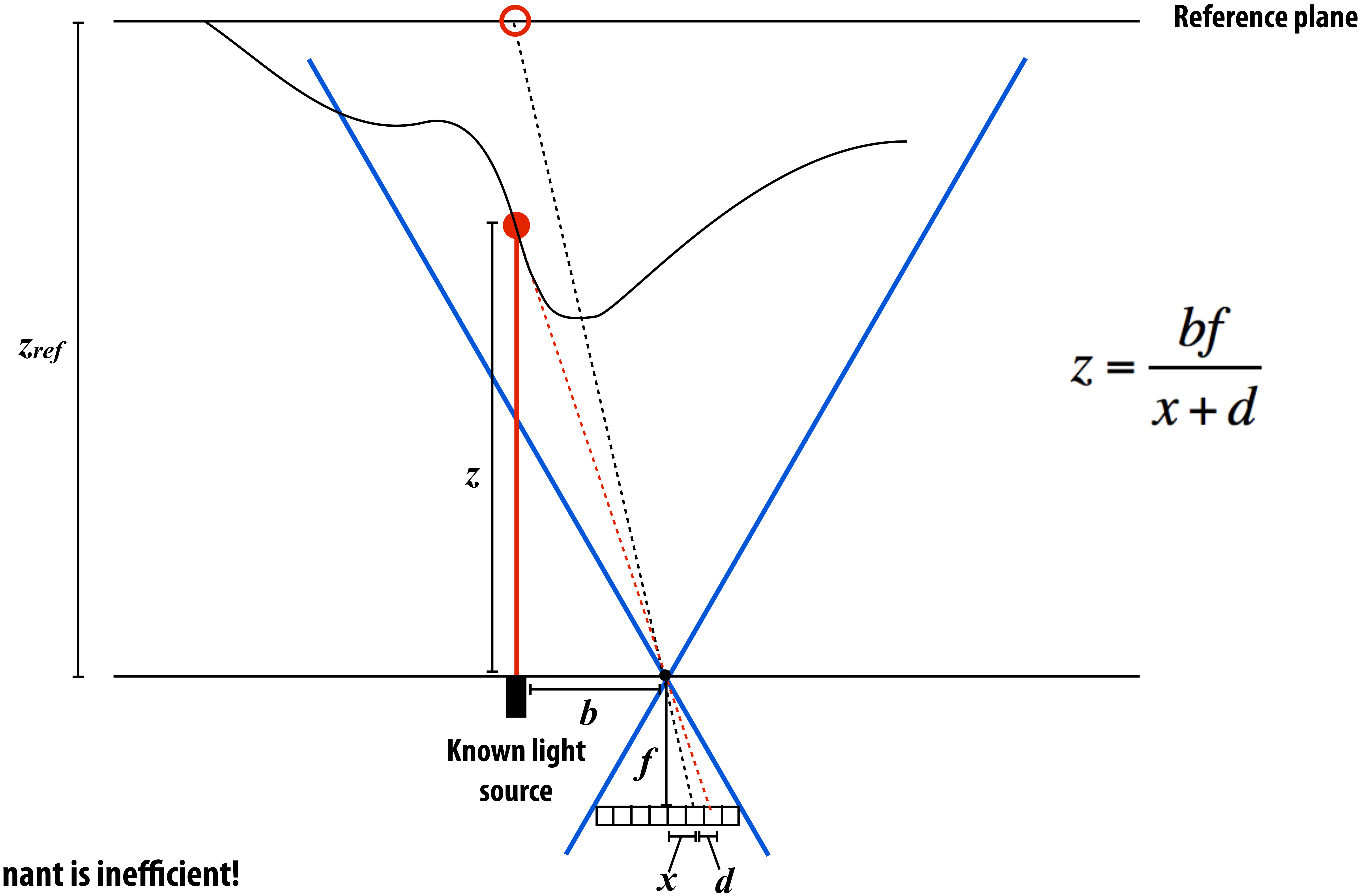
Credit: www.futurepicture.org

Depth from "disparity" using structured light

System: one light source emitting known beam + one camera measuring scene appearance

If the scene is at reference plane, image that will be recorded by camera is known

Movement of observed dot from from reference gives depth.



$$z = \frac{bf}{x+d}$$

Single spot illuminant is inefficient!

(Must "scan" scene to get depth, so high latency to retrieve a single depth image. Hence the dot pattern on the Kinect)

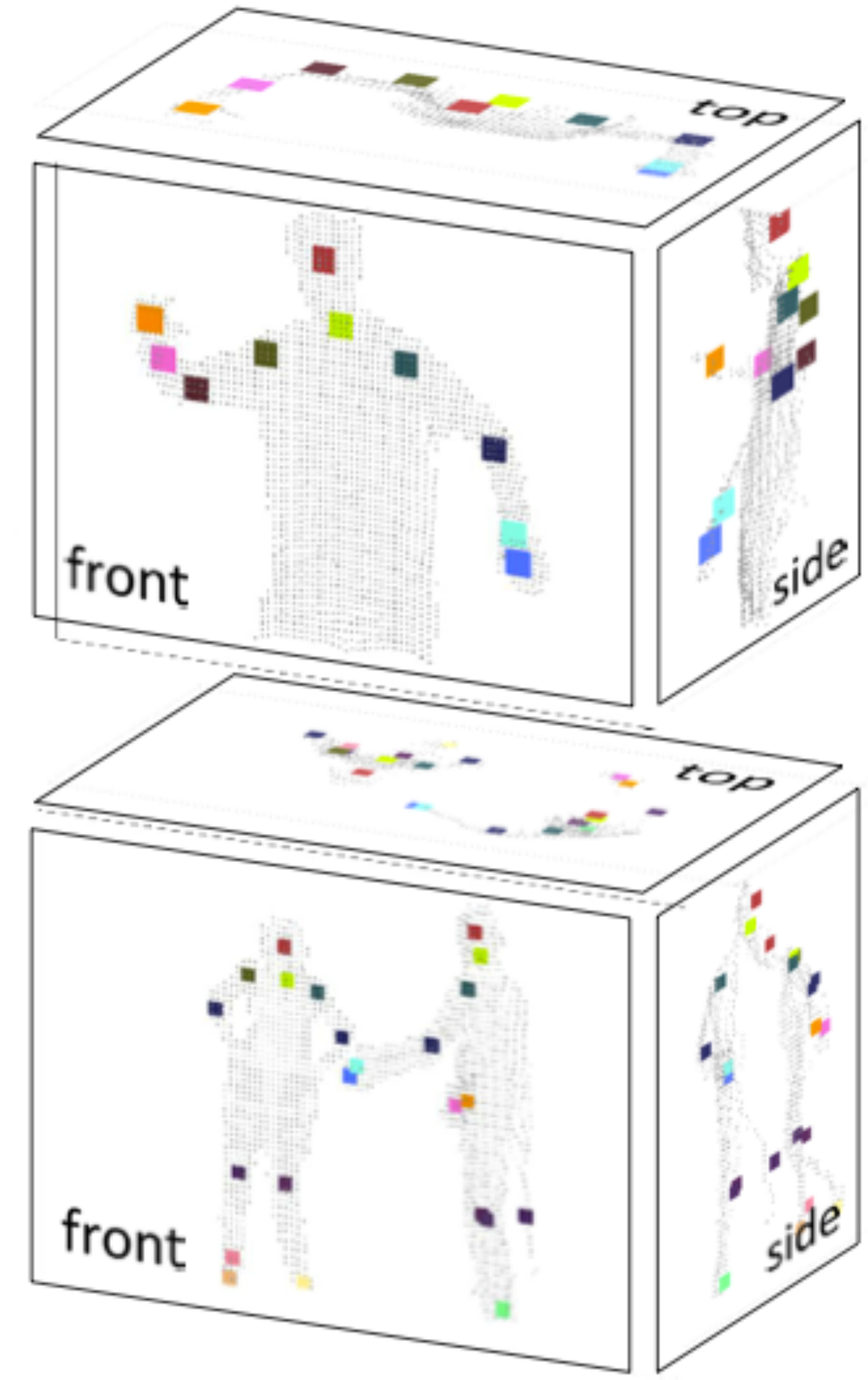
Extracting the player's 2D skeleton

[Shotton et al. 2011]

(enabling full-body game input)

Challenge: how to determine player's position and motion from (noisy) depth images... without consuming a large fraction of the XBox 360's compute capability?

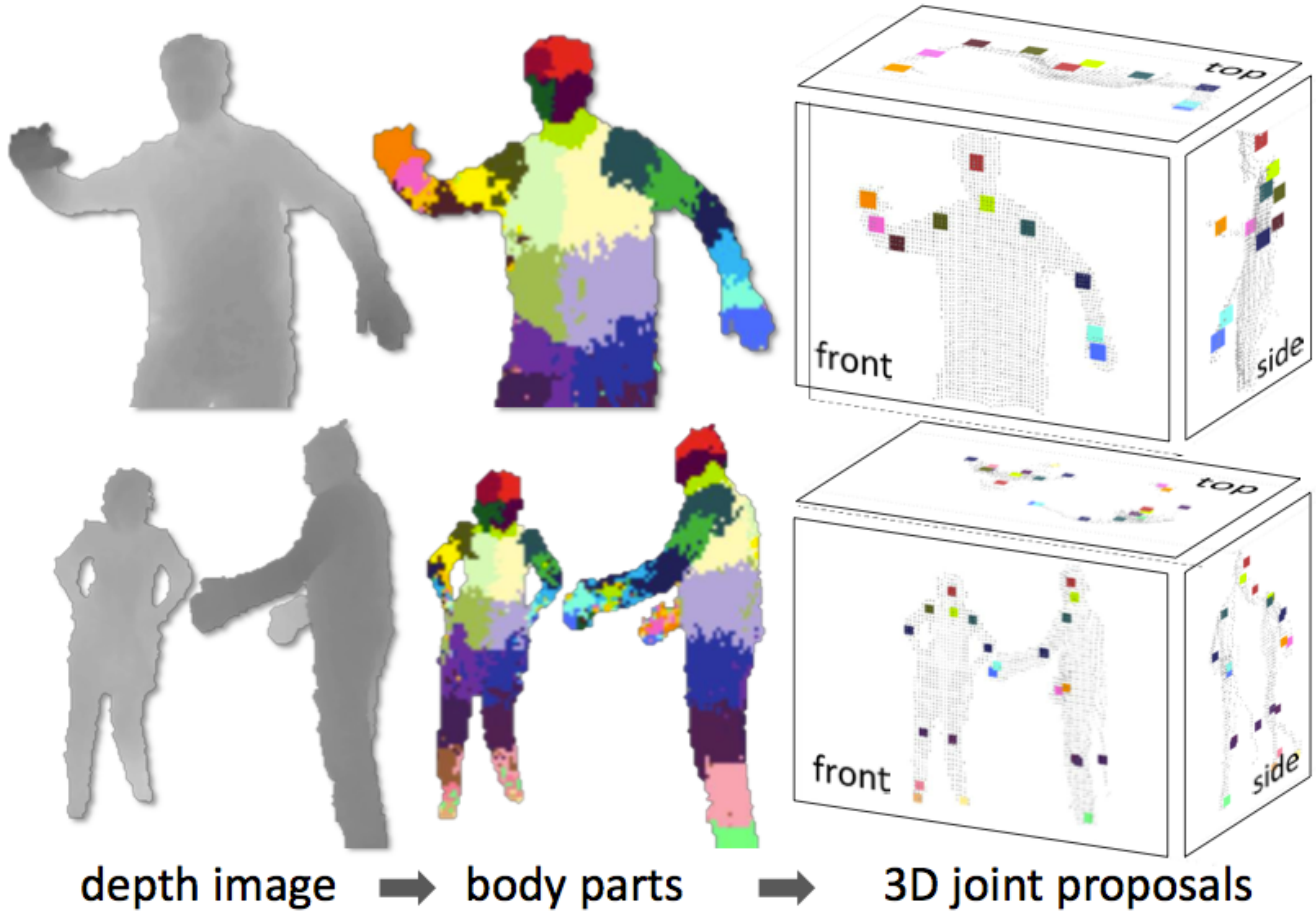
Depth Image



Character Joint Angles

Key idea: classify pixels into body regions

[Shotton et al. 2011]



Shotton et al. represents body with 31 regions

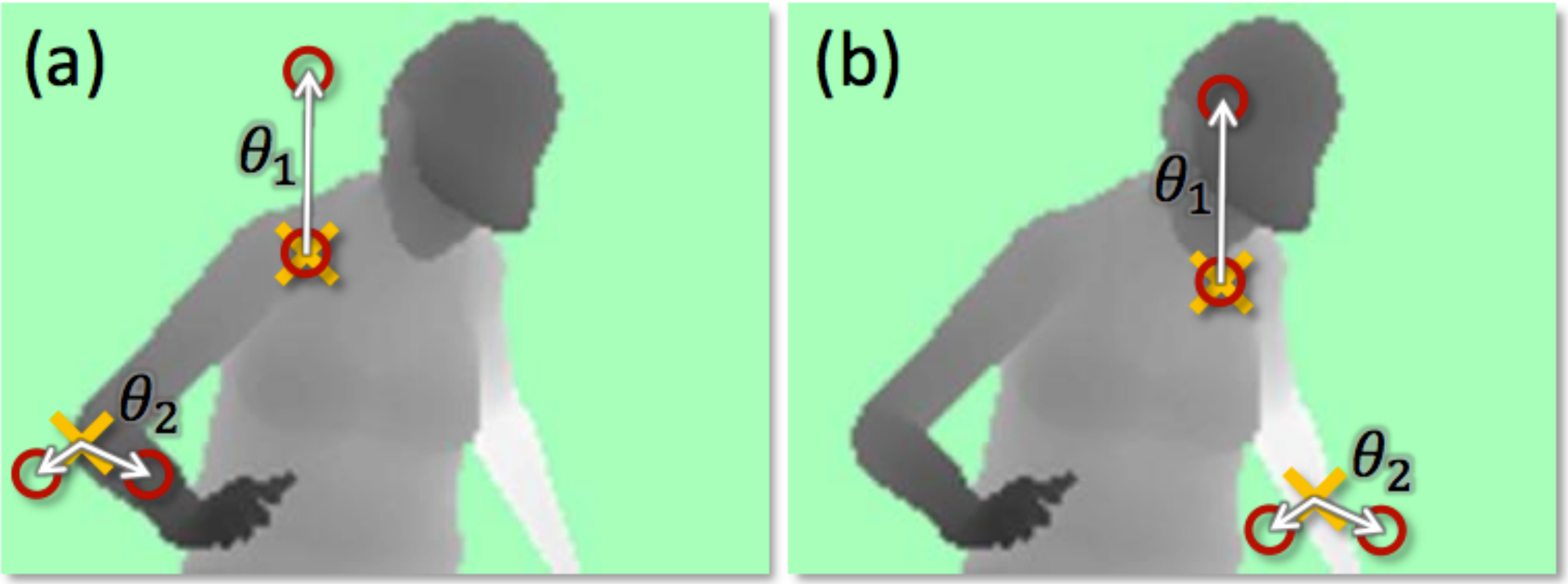
Pixel classification

[Shotton et al. 2011]

For each pixel: compute features from depth image

$$f_\theta(I,\mathrm{X}) = d_I\left(\mathrm{X} + \frac{u}{d_I(\mathrm{X})}\right) + d_I\left(\mathrm{X} + \frac{v}{d_I(\mathrm{X})}\right)$$

Where $\theta = (u, v)$ and $d_I(X)$ is the depth image value at pixel X.



Two example depth features

Features are cheap to compute + can be computed for all pixels in parallel
- Features do not depend on velocities: only information from current frame

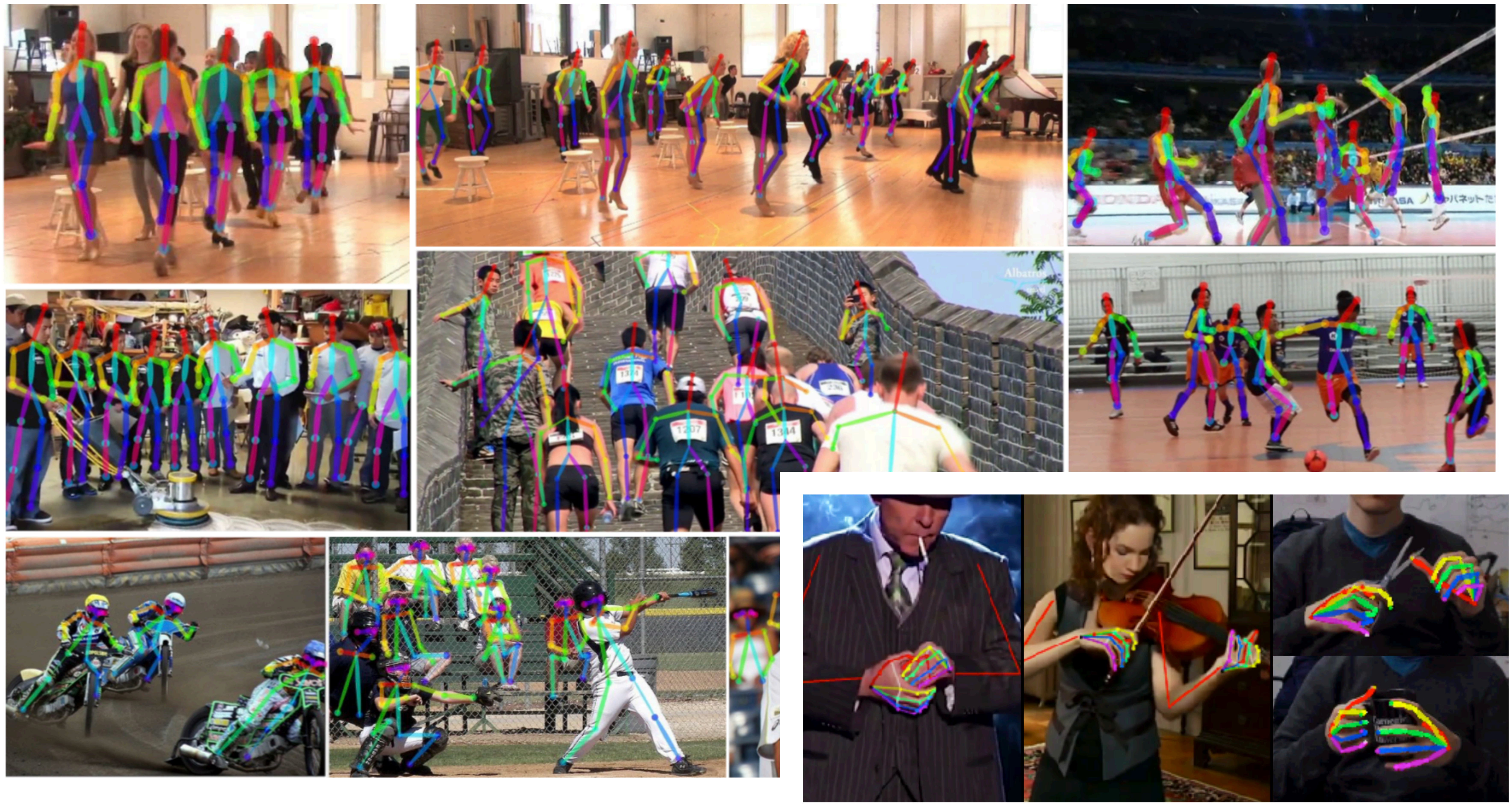
Classify pixels into body parts using randomized decision forest classifier
- Trained on 100K motion capture poses + database of rendered images as ground truth

Result of classification: $P(c|I, \mathbf{x})$ (probability pixel x in depth image I is body part c)

Per-pixel probabilities pooled to compute 3D spatial density function for each body part c
(joint angles inferred from this density)

Modern computer vision approaches

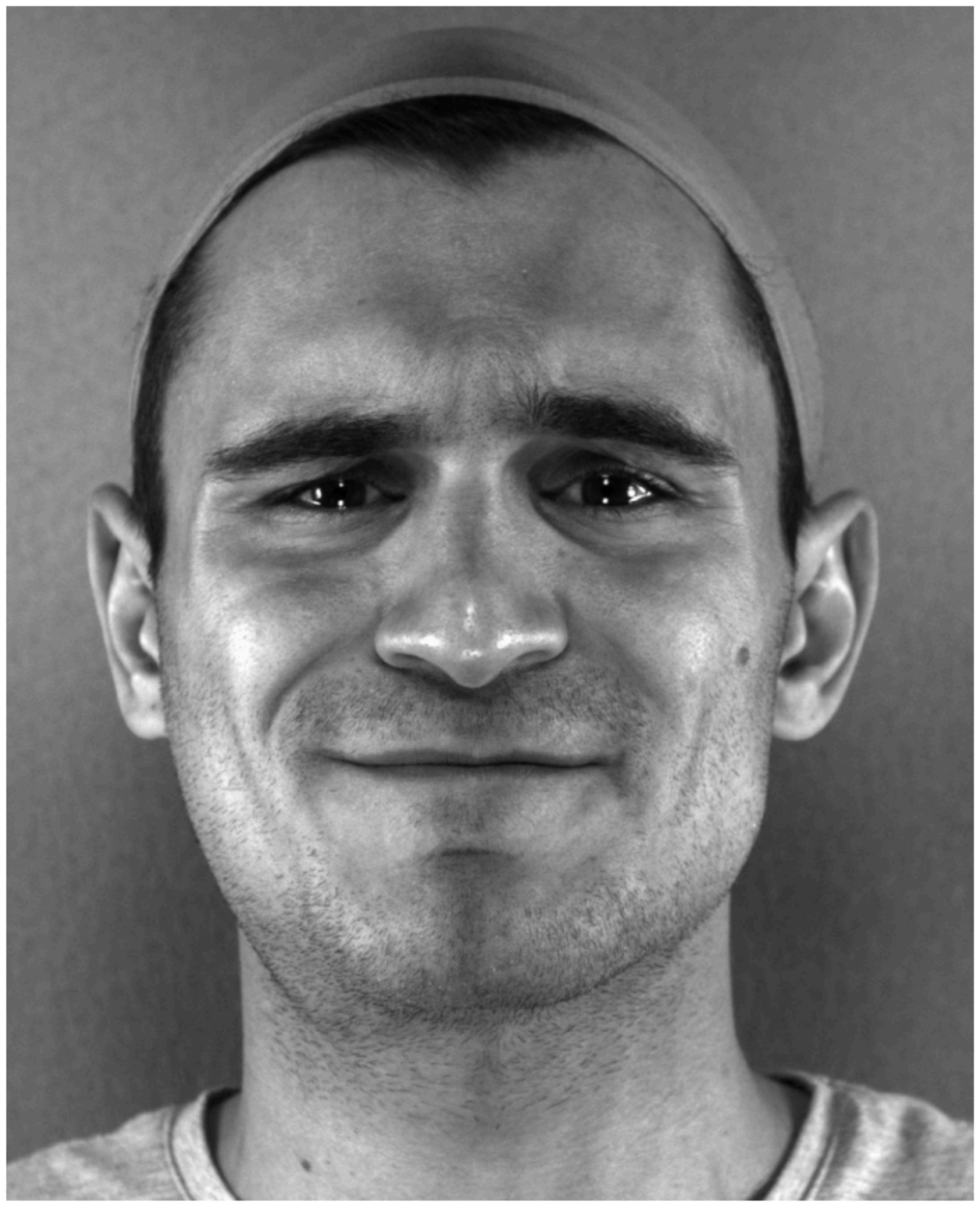
- "OpenPose": 2D (but not 3D) skeleton from single RGB image



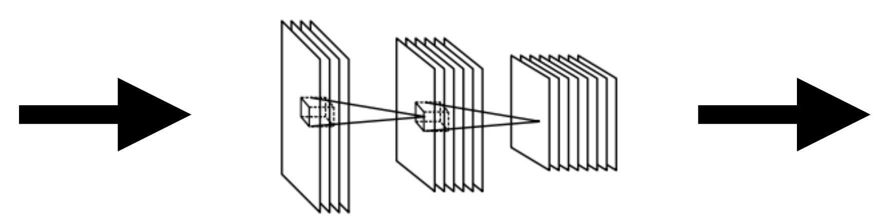
Hands/Fingers

Ongoing research to obtain high-quality 3D poses

Single camera facial performance capture

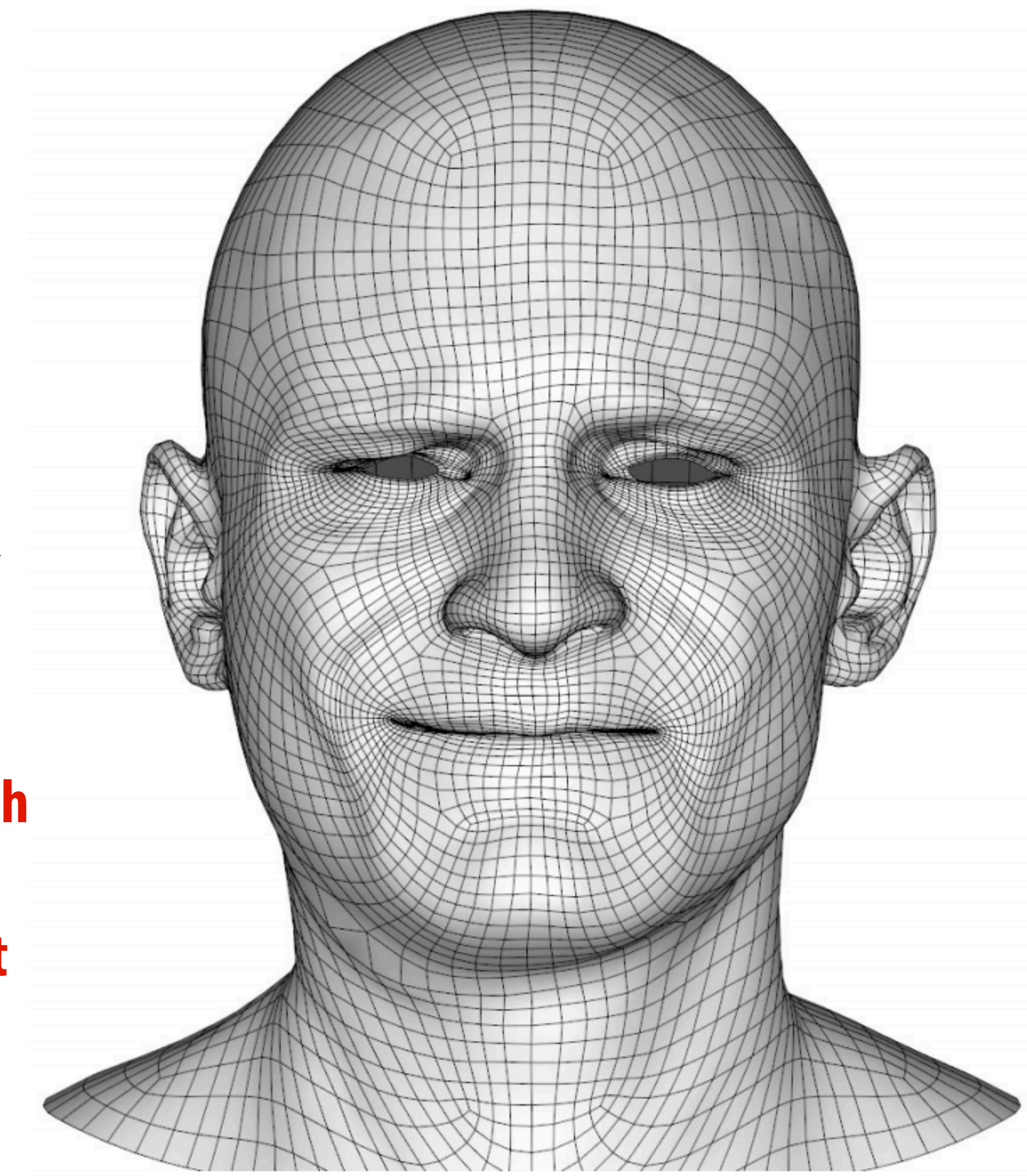


Input video frame



DNN

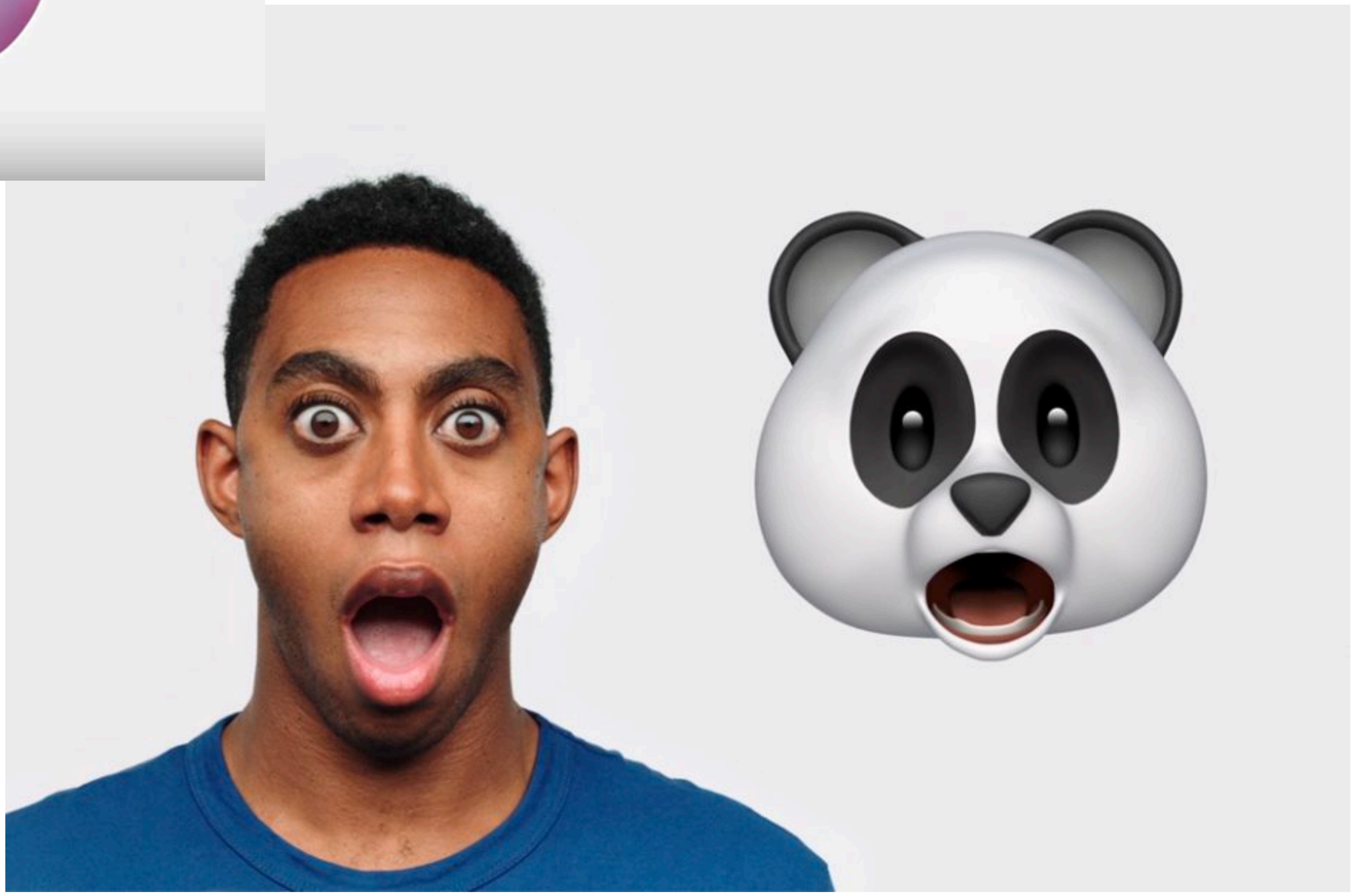
(trained on "ground truth" mesh data output by an expensive video processing pipeline that used 9 video cameras)



Output 3D mesh

[Image credit: "Production-Level Facial Performance Capture Using Deep Convolutional Neural Networks", Lehtinen et al 2017]

Single smartphone camera facial performance capture (Apple Animoji)

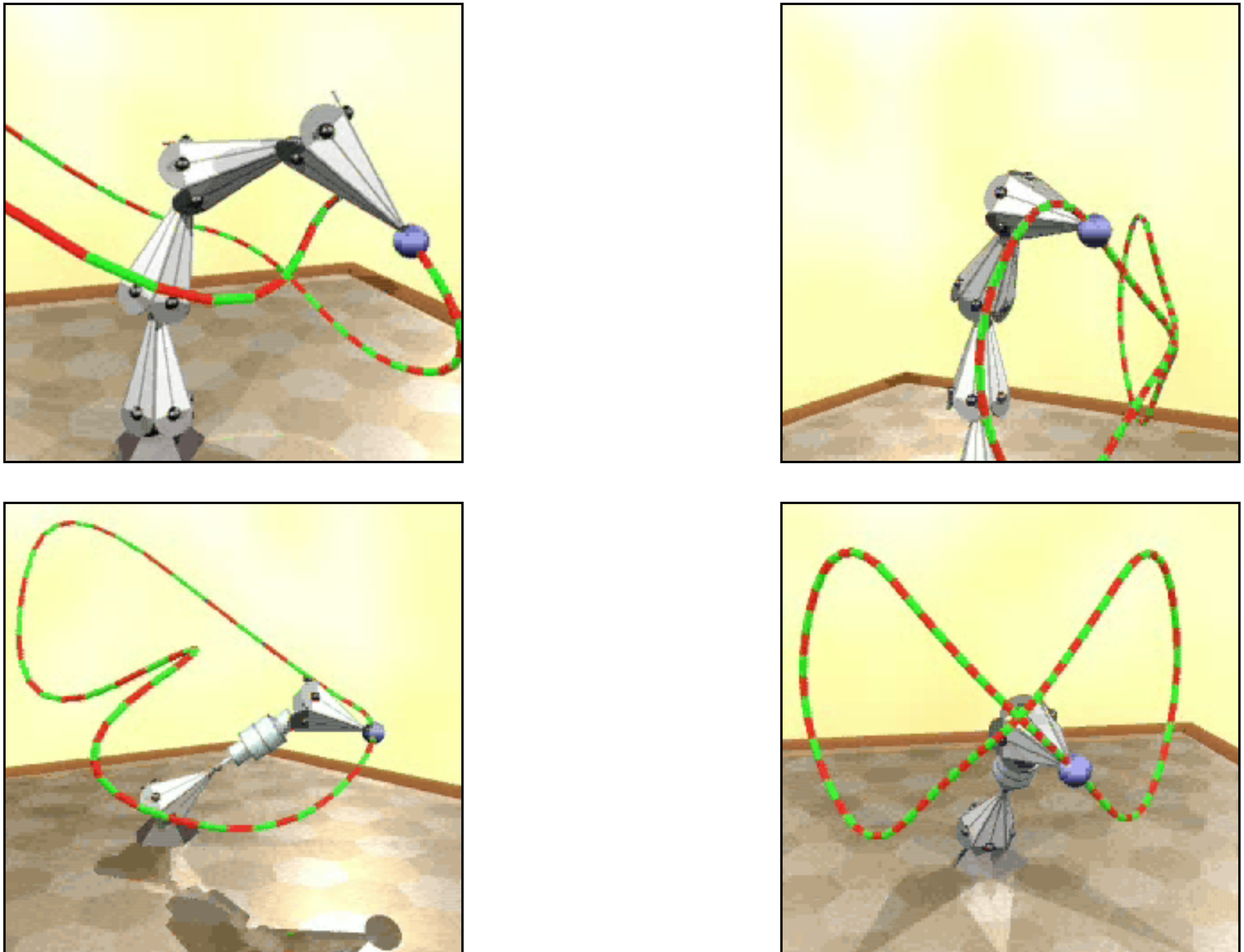


So far… we've discussed hand animating or directly measuring joint positions

Inverse Kinematics

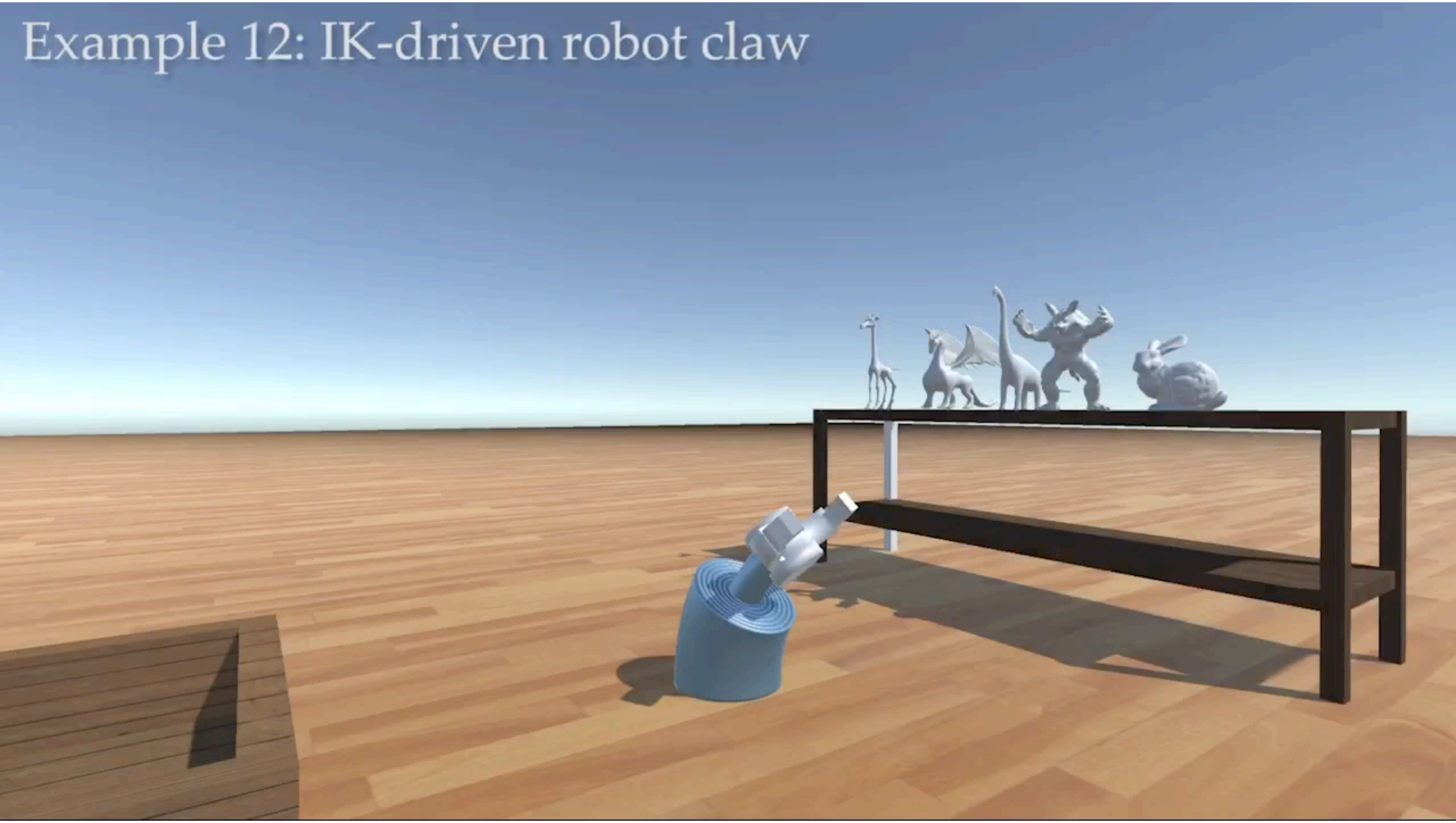
(computer solves for joint angles based on high-level goal)

Example: inverse kinematics



Egon Pasztor

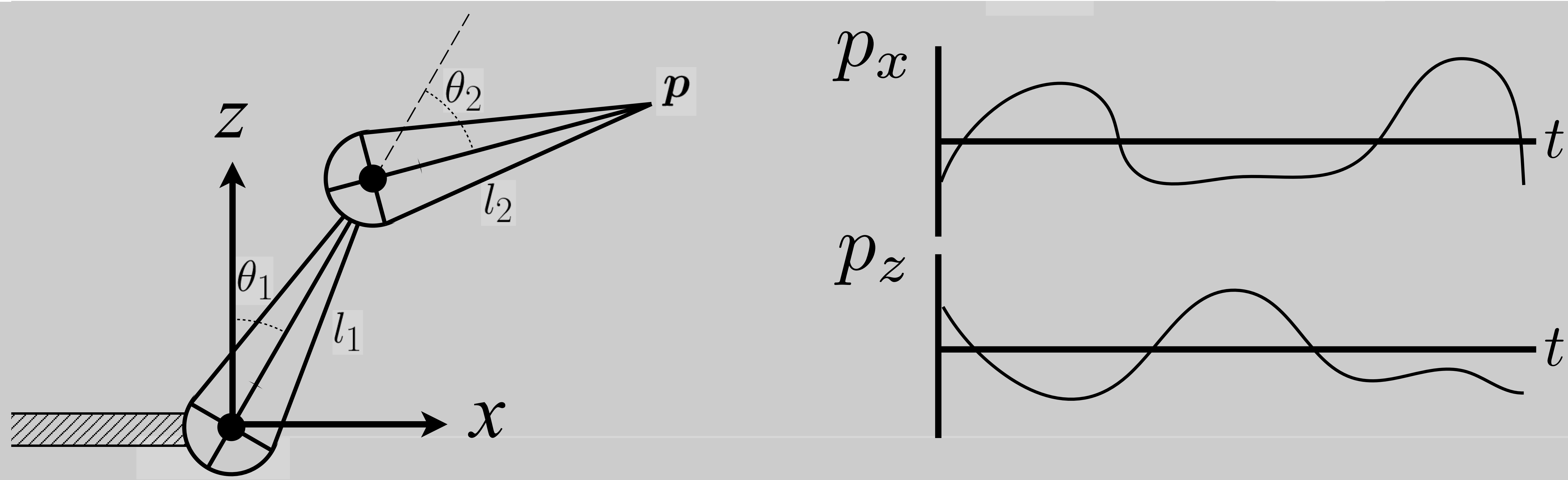
Example: inverse kinematics



Inverse kinematics

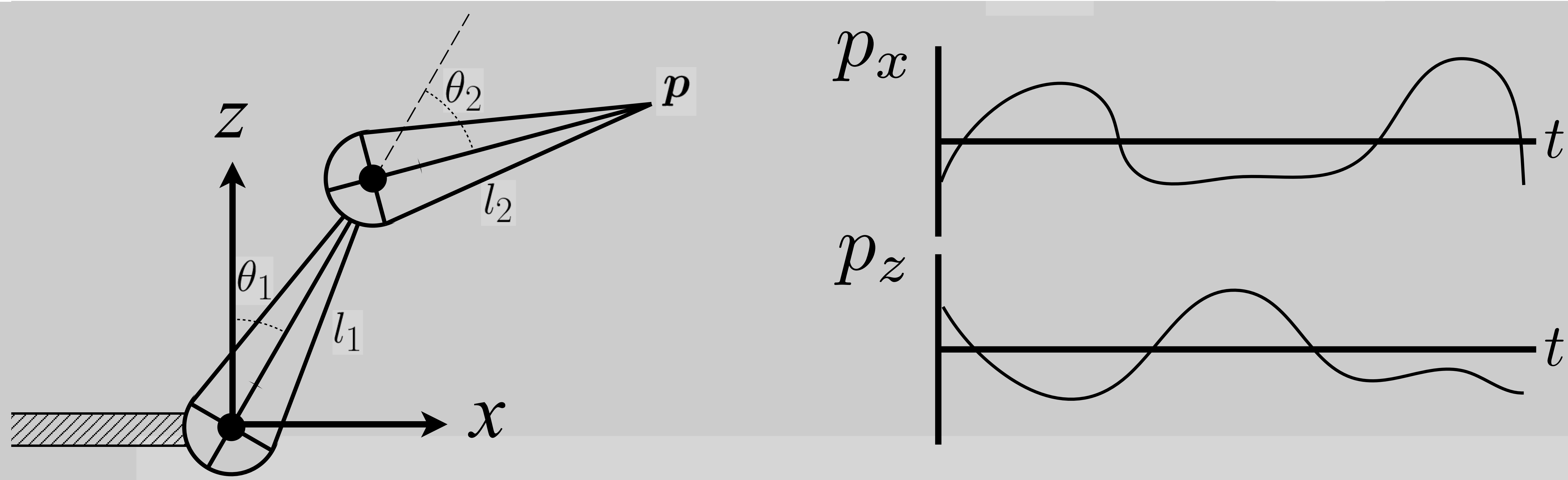
Input: animator provides position of end-effector

Output: computer must determine joint angles that satisfy constraints



Inverse kinematics

Direct inverse kinematics: for two-segment arm, can solve for parameters analytically (not true for general N-link problem)

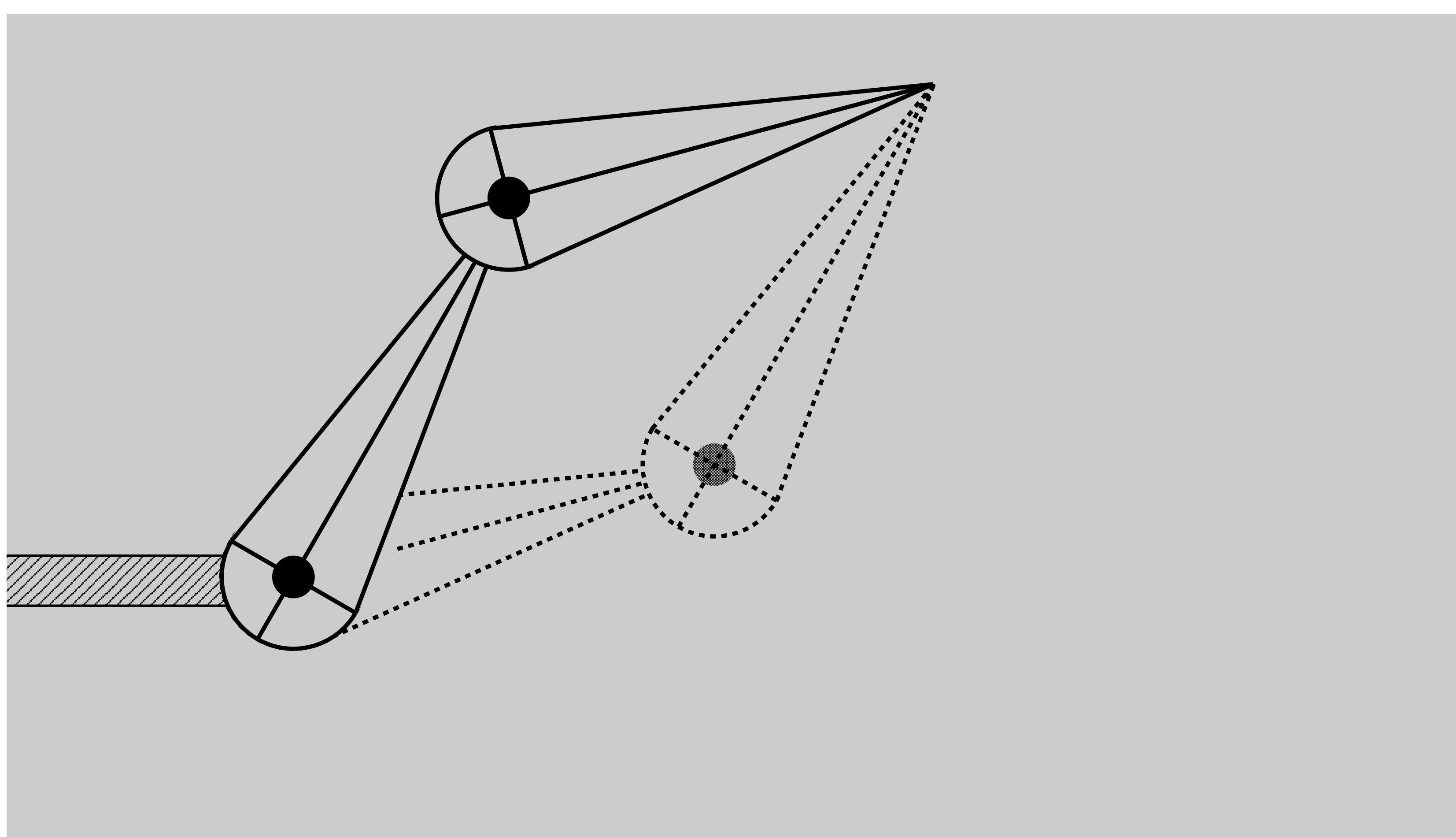


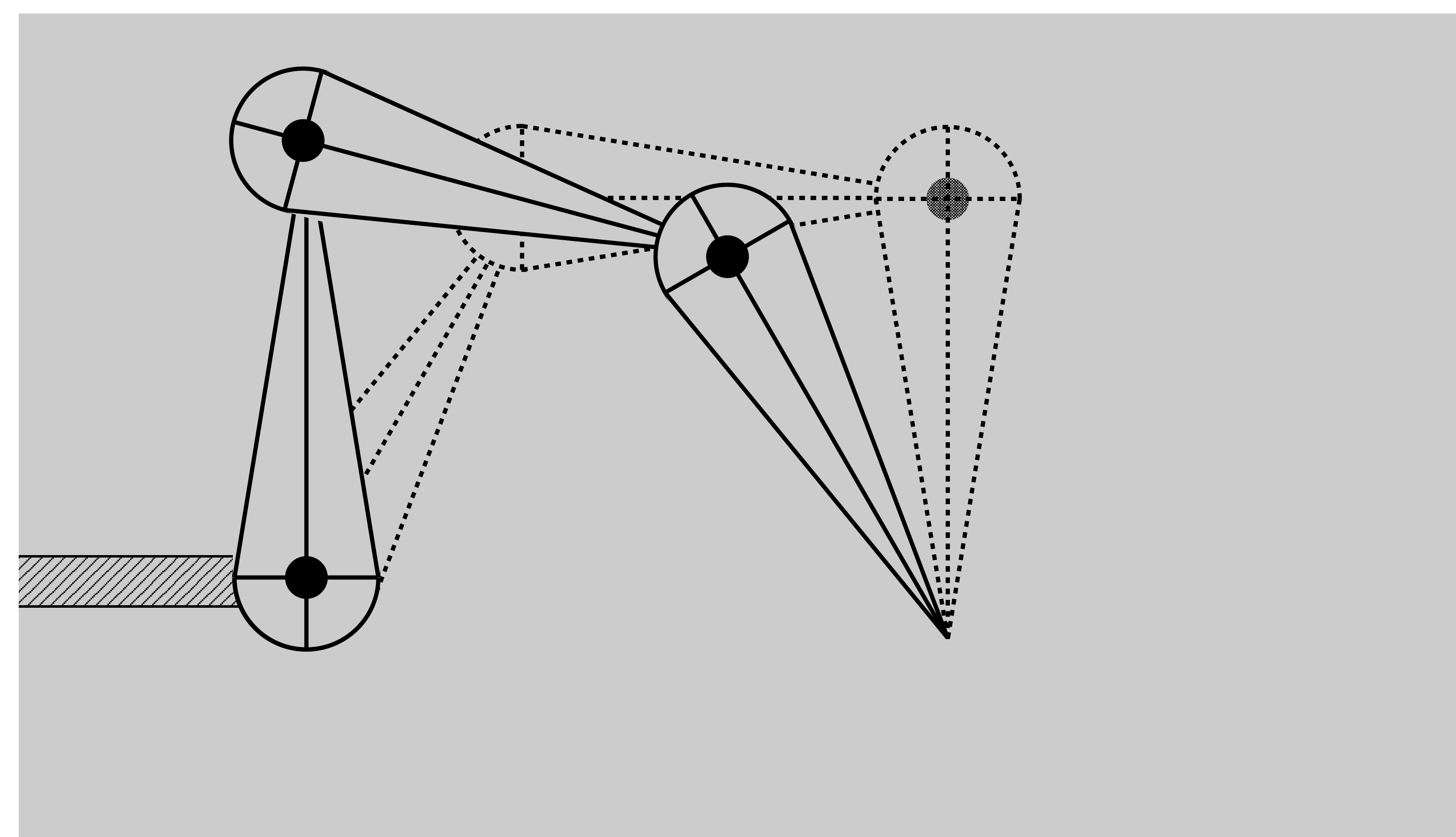
$$\theta_2 = \cos^{-1}\left(\frac{p_z^2 + p_x^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right)$$

$$\theta_1 = \frac{-p_z l_2 \sin(\theta_2) + p_x(l_1 + l_2\cos(\theta_2))}{p_x l_2 \sin(\theta_2) + p_z(l_1 + l_2\cos(\theta_2))}$$

Inverse kinematics

- Why is the problem hard?
  - Multiple solutions in configuration space (and these may not be nearby, causing jumps from frame-to-frame)
  - Solution may not be possible





Inverse kinematics

- Numerical solution to general N-link IK problem
  - Choose an initial configuration
  - Define an error metric (e.g. square of distance between goal and end effector's current position)
  - Apply *optimization method* to solve for joint angles given the desired (goal) end effector position

A few bits on optimization

(a commonly used tool in graphics)

Optimization problem in standard form

- **Can formulate most continuous optimization problems this way:**

"objective": how much does solution x cost?

$$(f_i : \mathbb{R}^n \to \mathbb{R},\ i = 0, \dots, m)$$

often (but not always) continuous, differentiable, ...

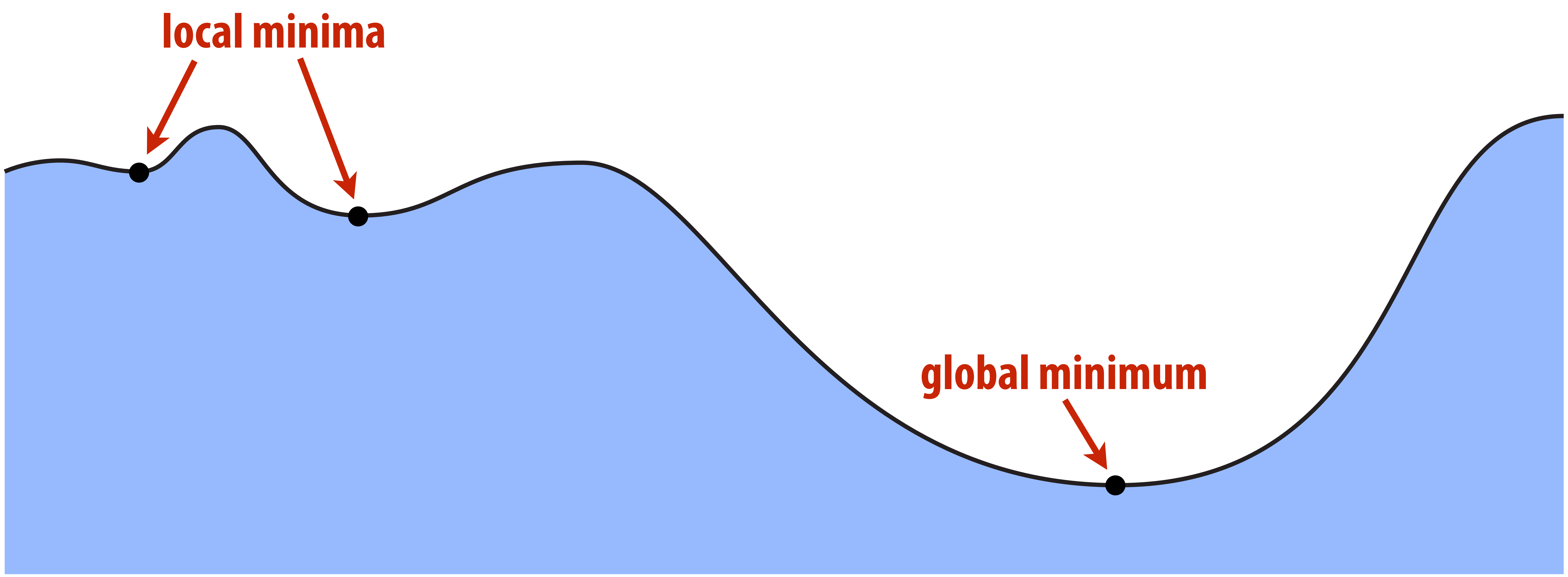
$$\begin{array}{ll} \displaystyle\min_{x \in \mathbb{R}^n} & f_0(x) \\ \text{subject to} & f_i(x) \le b_i,\ i = 1, \dots, m \end{array}$$

"constraints": what must be true about x? ("x is *feasible*")

- ***Optimal solution*** **x* has smallest value of f_0 among all feasible x**
- **Q: What if we want to *maximize* something instead?**
- **A: Just flip the sign of the objective!**
- **Q: What if we want *equality* constraints, rather than inequalities?**
- **A: Include two constraints: g(x) ≤ c and g(x) ≤ -c**

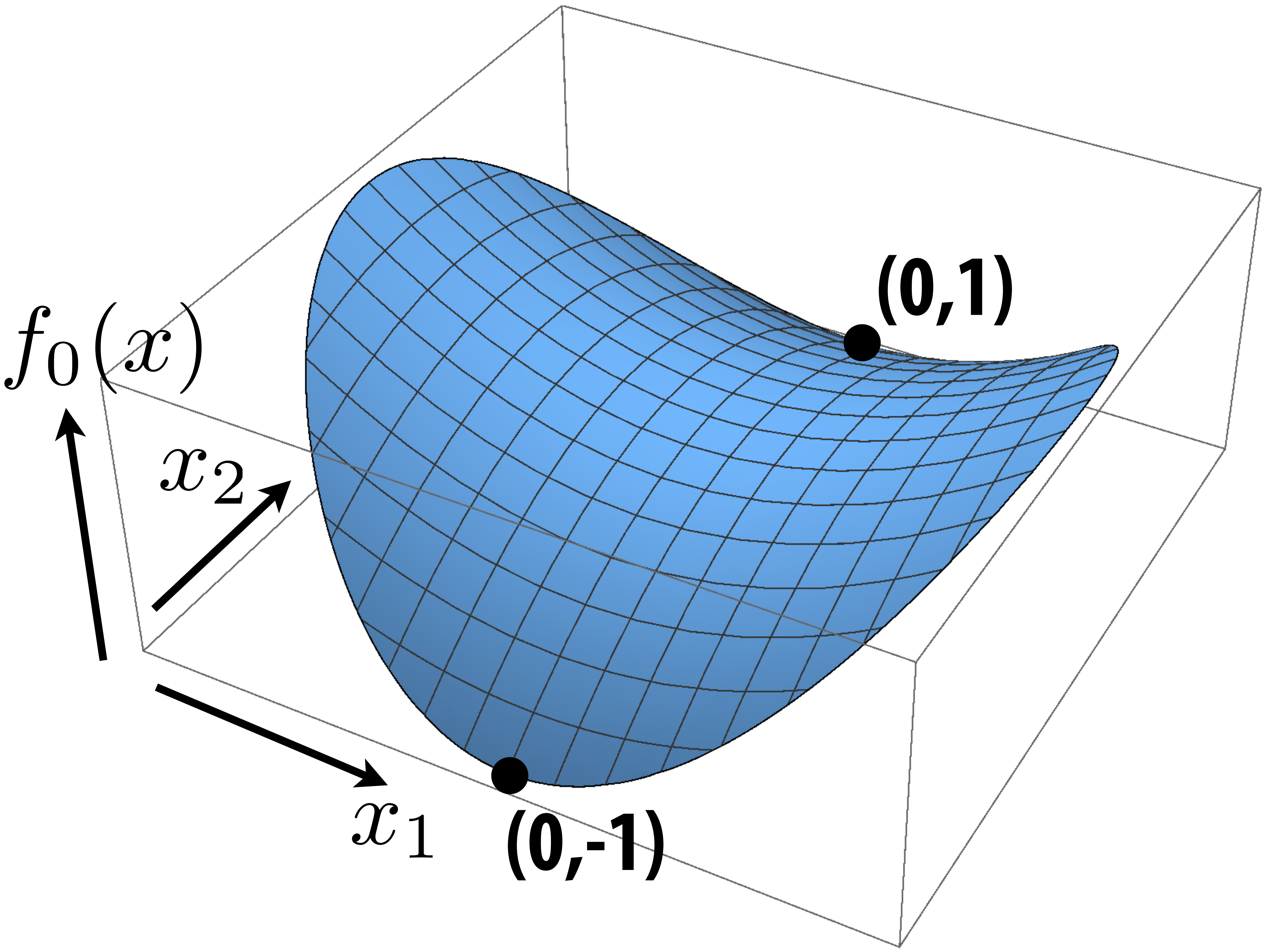
Local vs. global minima

- *Global* minimum is absolute best among all possibilities
- *Local* minimum is best "among immediate neighbors"



Philosophical question: does a local minimum *"solve"* the problem?

Optimization problem, visualized



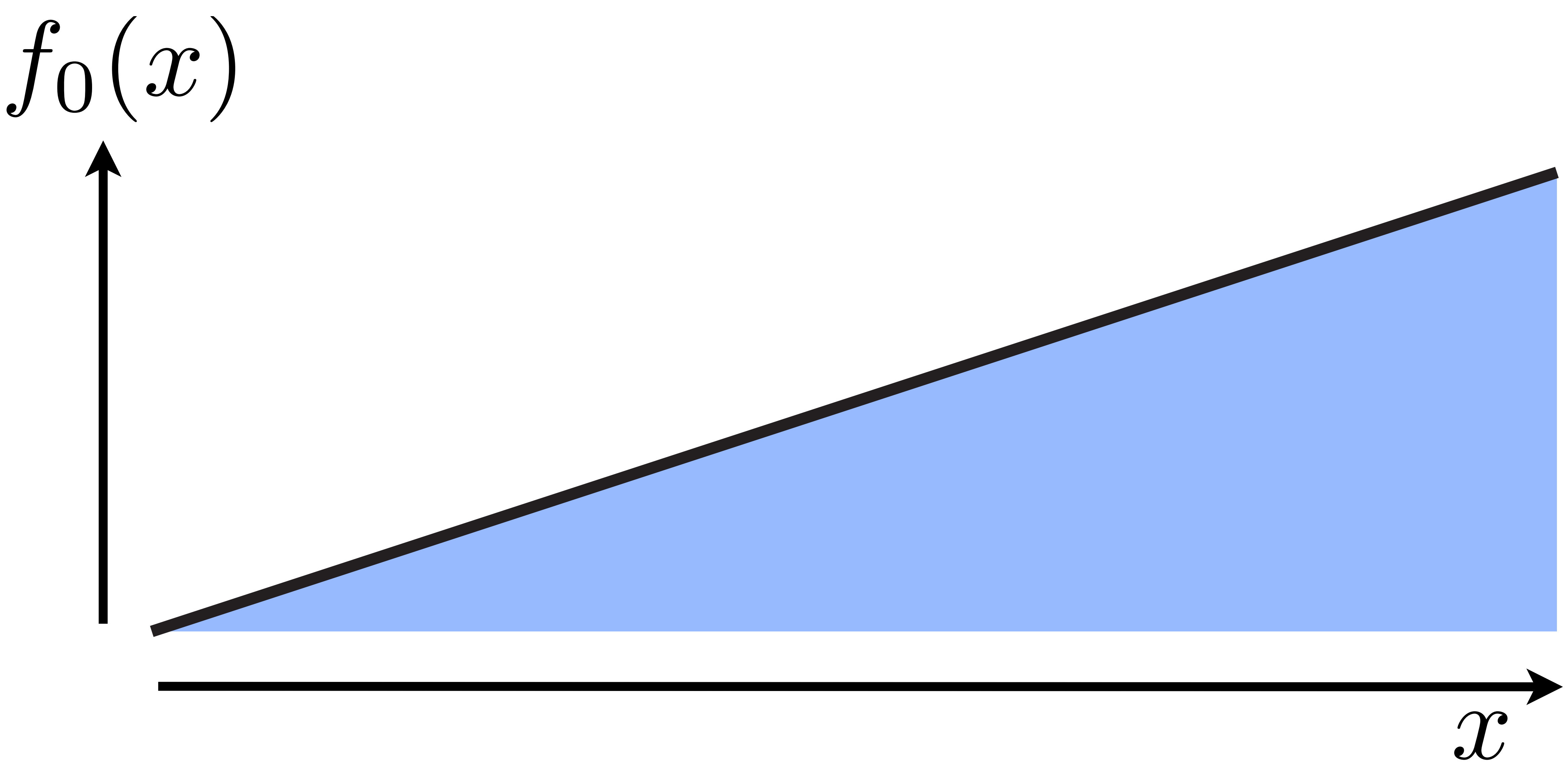
$$\min_{x \in \mathbb{R}^2} \quad x_1^2 - x_2^2$$
$$\text{s.t.} \quad x_1^2 + x_2^2 - 1 \leq 0$$

Q: Is this an optimization problem in standard form? **A: Yes**

Q: Where is the optimal solution? **A: There are two, (0,1), (0,-1)**

Existence and uniqueness of minimizers

- Already saw that (global) minimizer is not unique
- Does it always exist? Why?
- Just consider all possibilities and take the smallest one, right?



perfectly reasonable optimization *problem*

$$\min_{x \in \mathbb{R}} x$$

clearly has no *solution* (can always pick smaller x)

- WRONG! Not all objectives are bounded from below.
- It's like that old adage: *"no matter how good you are, there will always be someone better than you."*

Feasibility

- Ok, but suppose the objective is bounded from below
- Then we can just take the best feasible solution, right?

value of objective doesn't depend on x;
all feasible solutions are equally good

$$\begin{aligned}\min_{x\in\mathbb{R}^n} \quad & 0 \\ \text{subject to} \quad & f_i(x) \leq b_i,\ i = 1,\ldots,m\end{aligned}$$

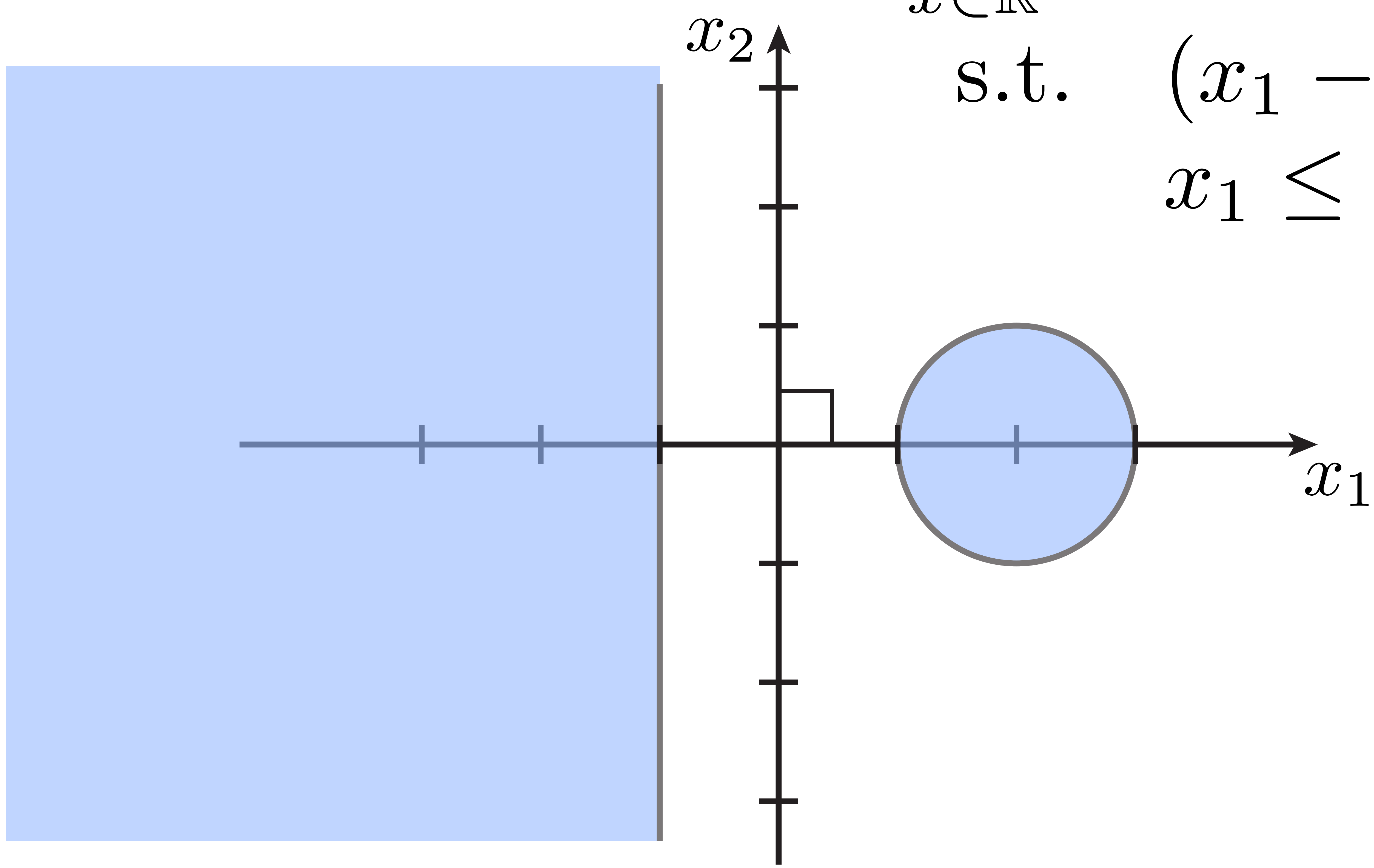
problem now is just finding a feasible solution—
which can be really hard (or impossible!)

- Not if there aren't any!
- Not all problems have solutions!

Feasibility - example

Q: Is this problem feasible?

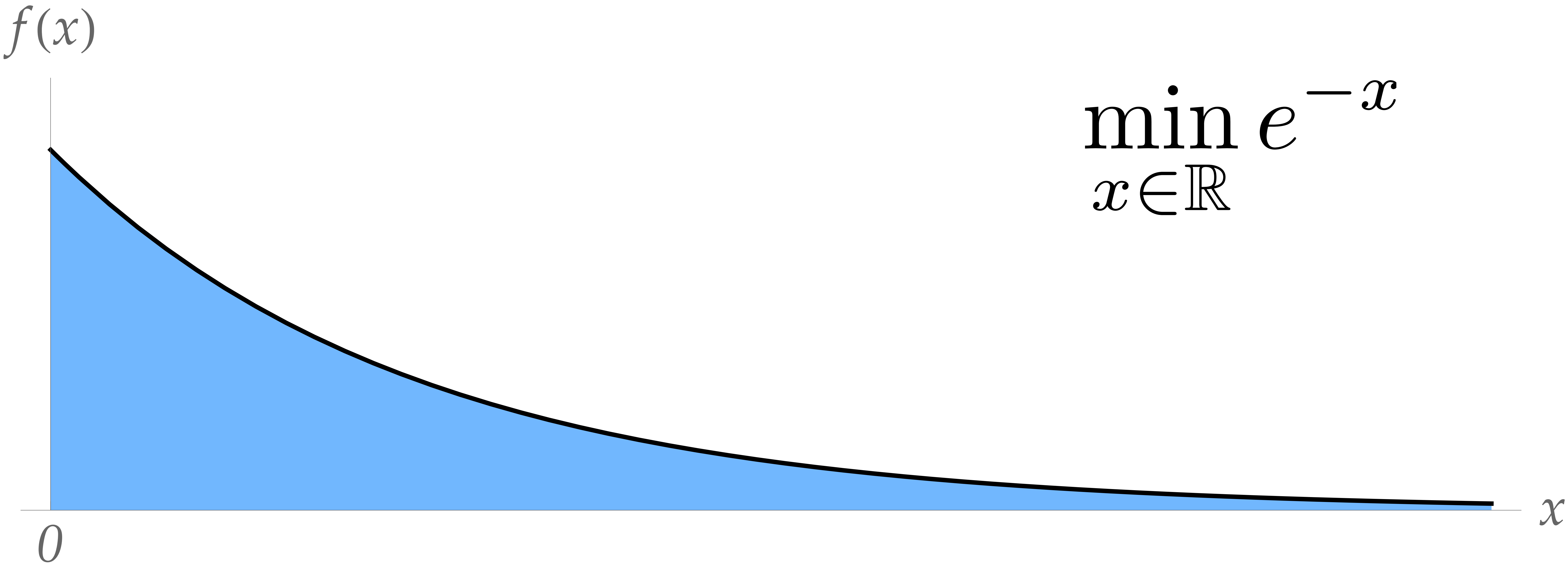
$$\min_{x \in \mathbb{R}^2} \quad \sin(x_1) + x_2^2$$
$$\text{s.t.} \quad (x_1 - 2)^2 + x_2^2 \leq 1,$$
$$x_1 \leq -1$$



A: No—the two sublevel sets (points where f_i(x) ≤ 0) have no common points, i.e., they do not overlap.

Existence and uniqueness of minimizers, cont.

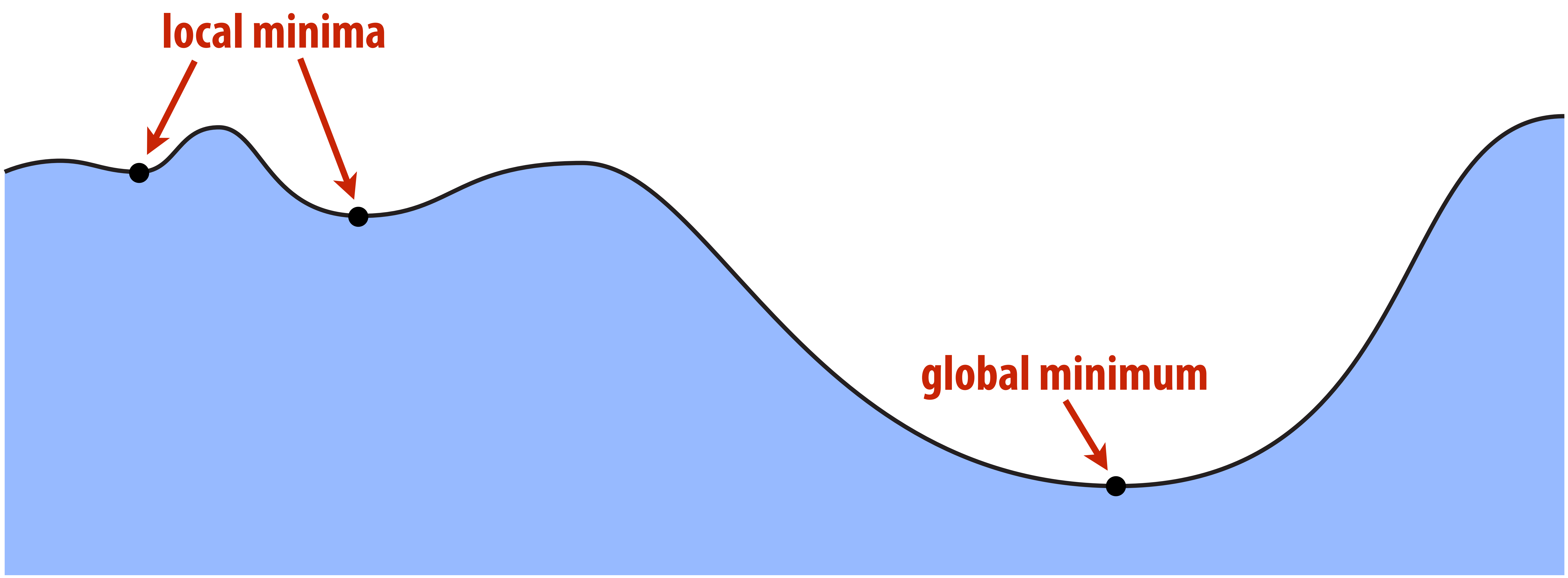
- Even being bounded from below is not enough:



- No matter how big x is, we never achieve the lower bound (0)

Characterization of minimizers

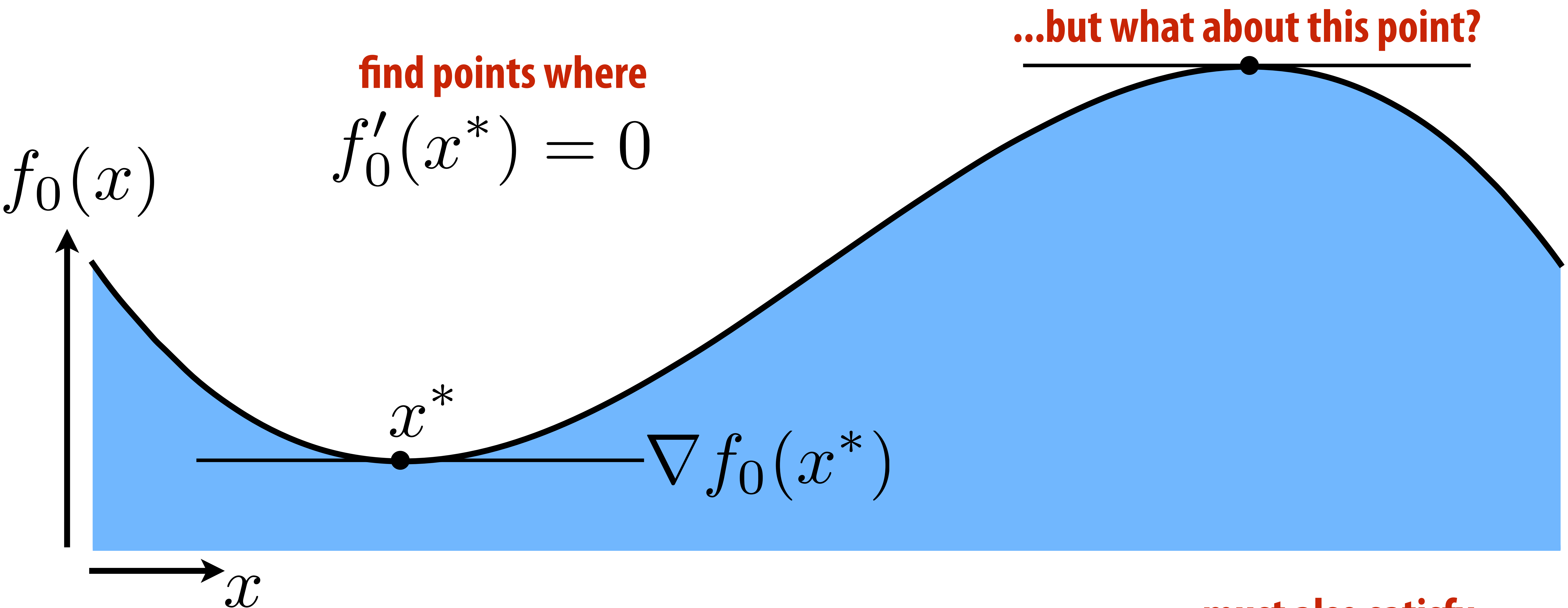
- Ok, so we have some sense of when a minimizer might *exist*
- But how do we know a given point x is a minimizer?



- Checking if a point is a global minimizer is (generally) hard
- But we can certainly test if a point is a local minimum (ideas?)
- (Note: a global minimum is also a local minimum!)

Characterization of local minima

- **Consider an objective f_0: R → R. How do you find a minimum?**
- **(Hint: you may have memorized this formula in high school!)**



- **Also need to check *second* derivative (how?)**
- **Make sure it's *positive***
- **Ok, but what does this all mean for more general functions f_0?**

must also satisfy

$$f_0''(x^*) \geq 0$$

Optimality conditions (unconstrained)

- In general, our objective is f0: $R^n \rightarrow R$
- How do we test for a local minimum?
- 1st derivative becomes *gradient*; 2nd derivative becomes *Hessian*

$$\nabla f := \begin{bmatrix} \partial f / \partial x_1 \\ \vdots \\ \partial f / \partial x_n \end{bmatrix}$$

GRADIENT
(measures "slope")

$$\nabla^2 f := \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial f}{\partial x_n^2} \end{bmatrix}$$

HESSIAN
(measures "curvature")

- Optimality conditions?

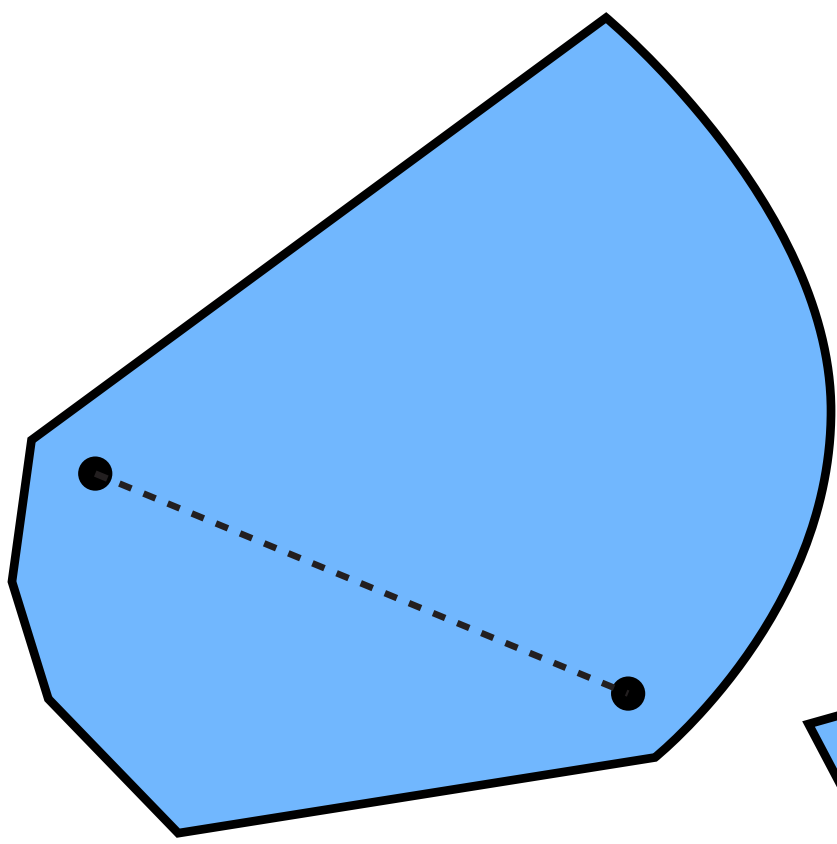
positive semidefinite (PSD)
($u^T A u \geq 0$ for all u)

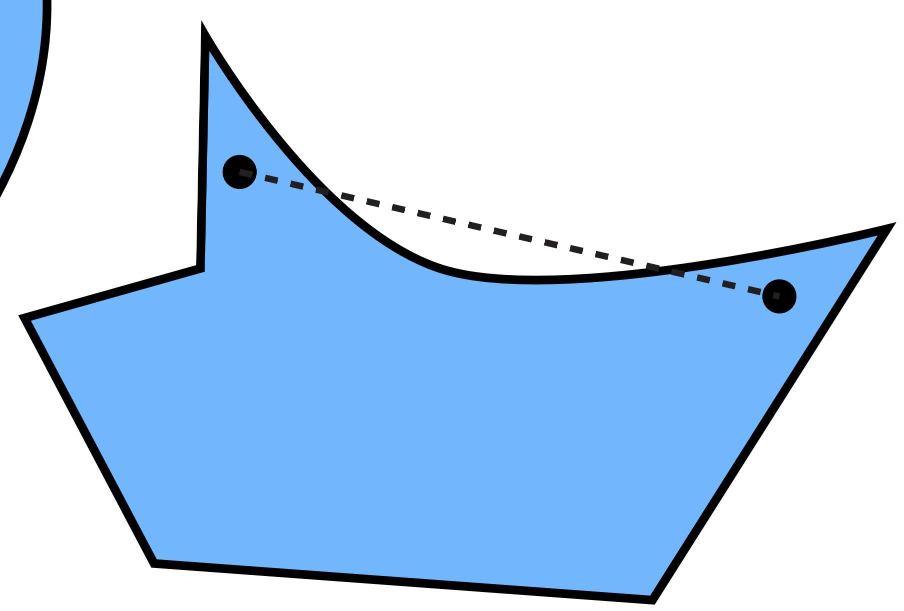
$$\nabla f_0(x^*) = 0 \qquad\qquad \nabla^2 f_0(x^*) \succeq 0$$

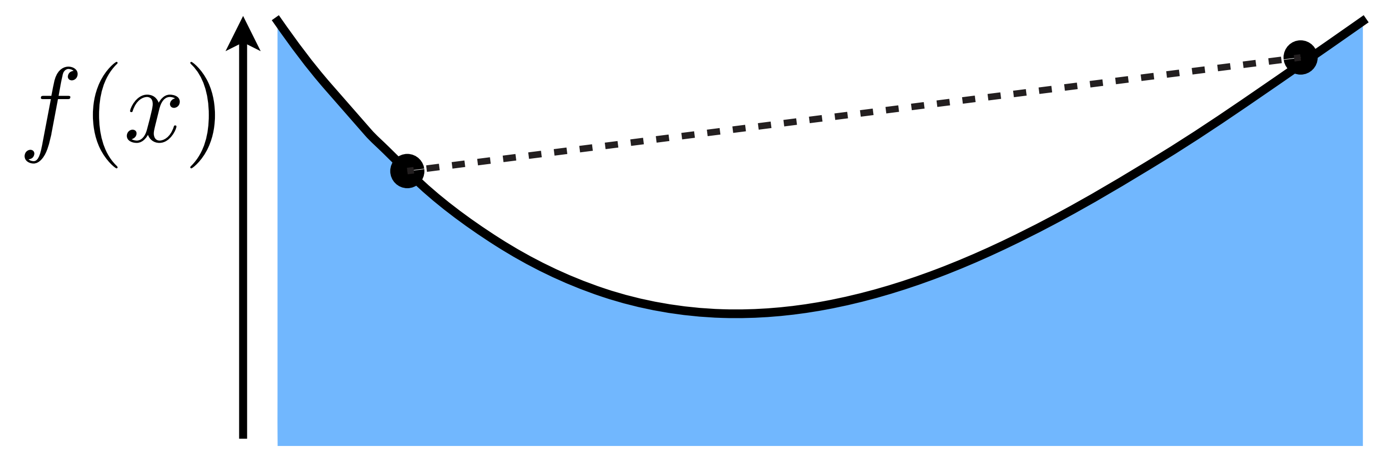
1st order   2nd order

Convex optimization

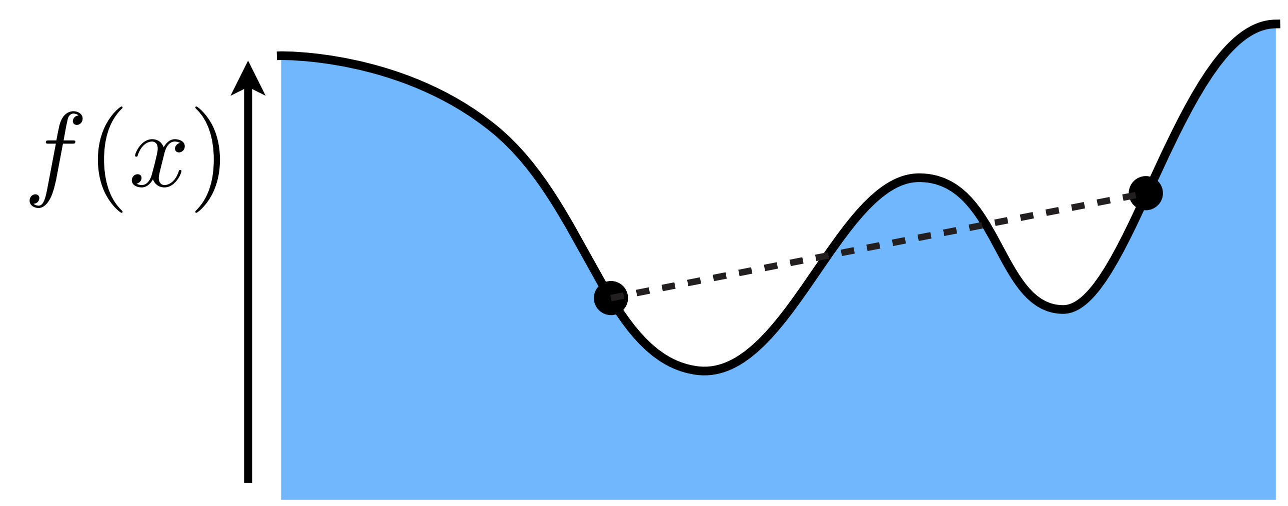
- Special class of problems that are almost always "easy" to solve (polynomial-time!)
- Problem is *convex* if it has a convex domain *and* convex objective


convex domain


noconvex domain



convex objective



nonconvex objective

- Why care about convex problems in graphics?
  - can make guarantees about solution (always the best)
  - doesn't depend on initialization (strong convexity)
  - often efficient to solve, but not always

Sadly, life is not usually that easy.
How do we solve optimization problems in general?

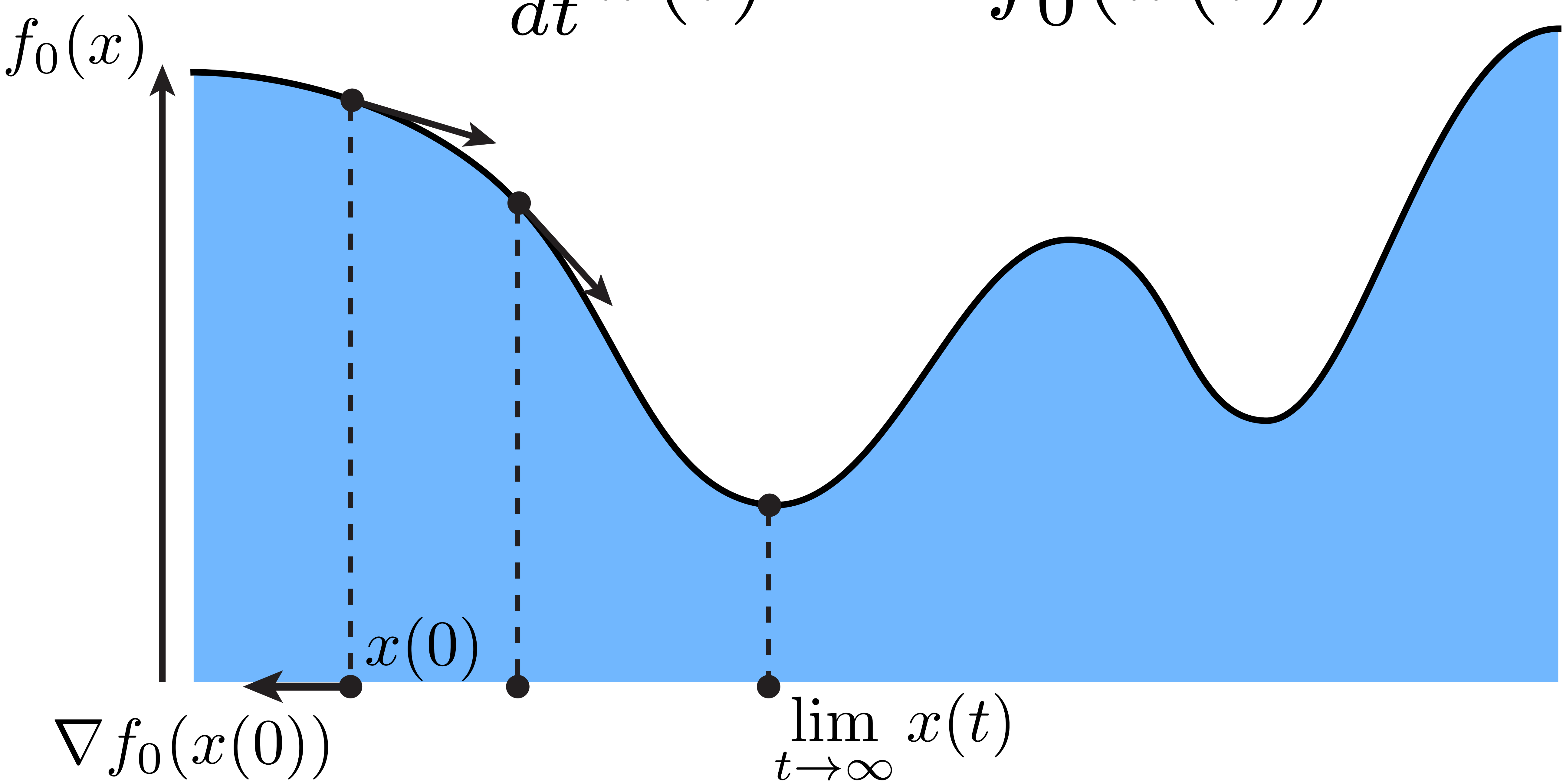
Descent methods

An idea as old as the hills:

Gradient descent (1D)

- **Basic idea: follow the gradient "downhill" until it's zero**
- **(Zero gradient was our 1st-order optimality condition)**

$$\frac{d}{dt}x(t) = -f_0'(x(t))$$



- **Do we always end up at a (global) minimum?**
- **How do we compute gradient descent in practice?**

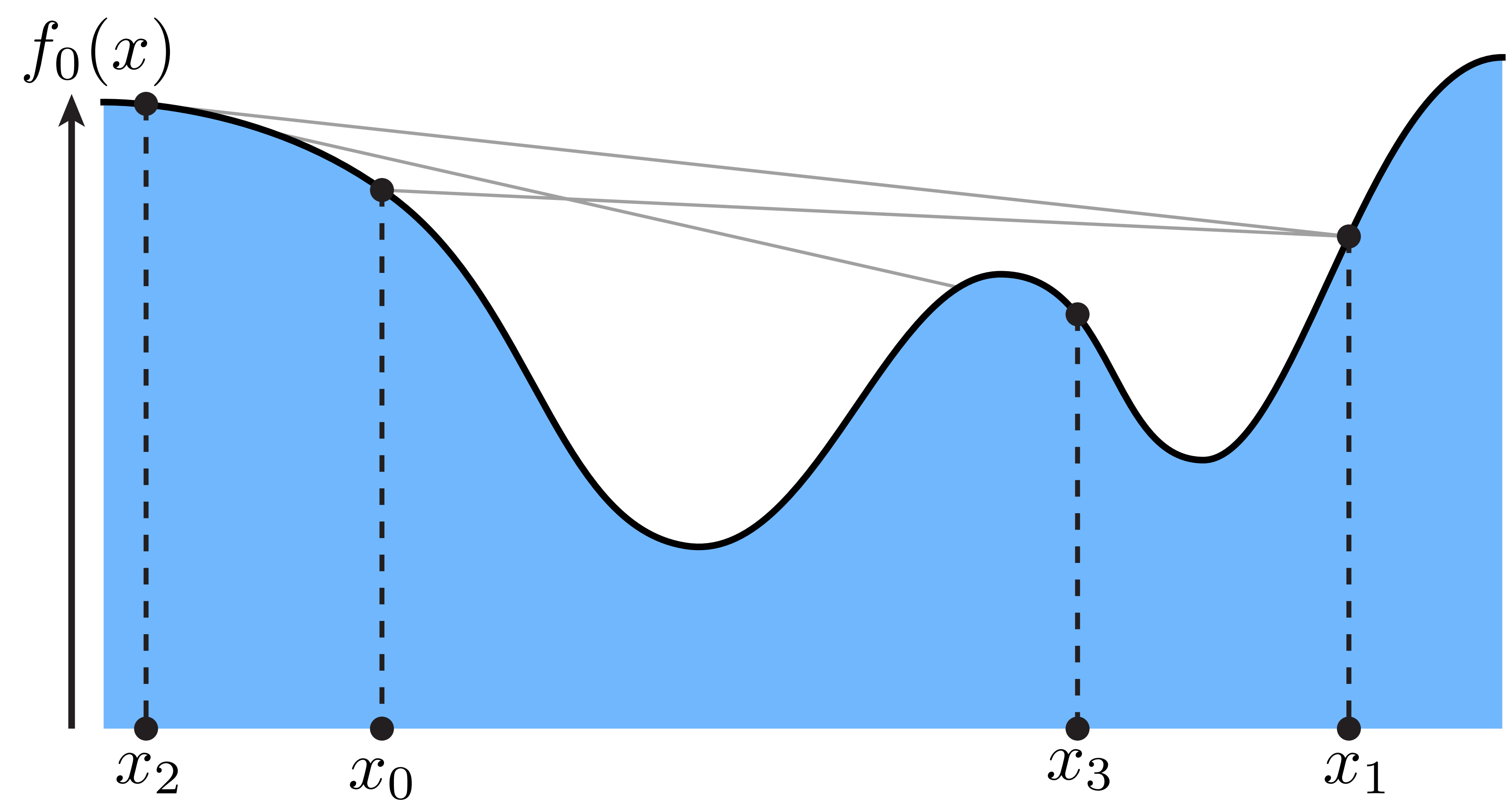
Gradient descent algorithm (1D)

- "Walk downhill"

$$x_{k+1} = x_k - \tau f_0'(x_k)$$

new estimate

step size



- Q: How do we pick the step size?
- If we're not careful, we'll go zipping all over the place; won't make any progress.

- Basic idea: use *"step control"* to determine step size based on value of objective and derivatives
- For now we will do something simple: make τ *small!*

Gradient descent algorithm (n-D)

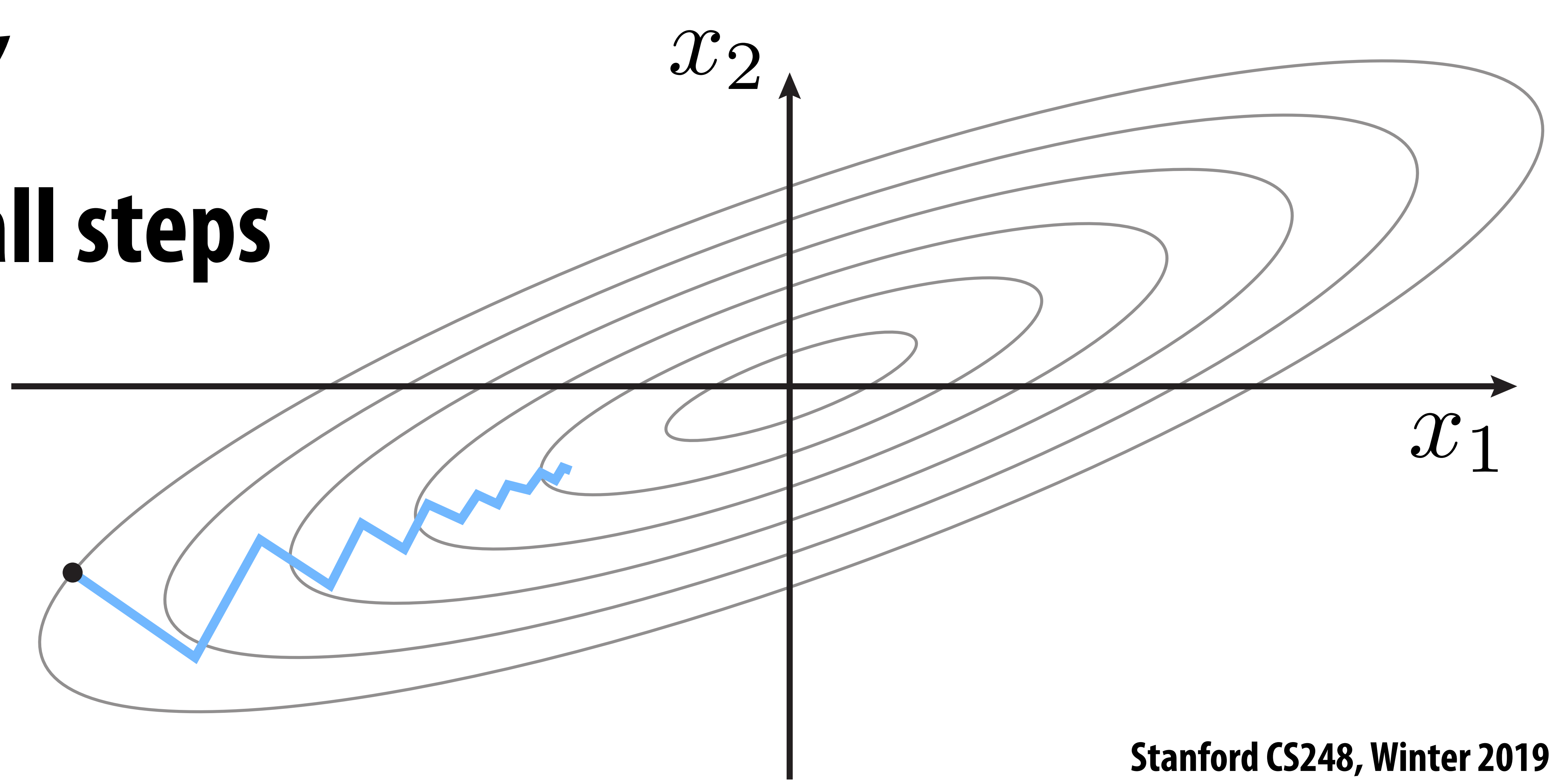
- Q: How do we write gradient descent equation in general?

$$\frac{d}{dt}x(t) = -\nabla f_0(x(t))$$

- Q: What's the corresponding discrete update?

$$x_{k+1} = x_k - \tau \nabla f_0(x_k)$$

- Basic challenge in nD:
  - solution can "oscillate"
  - takes many, many small steps
  - very slow to converge



Simple inverse kinematics algorithm

- What is the objective?
  - Distance from end effector position (given current joint parameters) to target position.

position of end effector (given θ)

$$f_0(\theta) = \|p_{\text{current}} - p_{\text{target}}\|^2$$

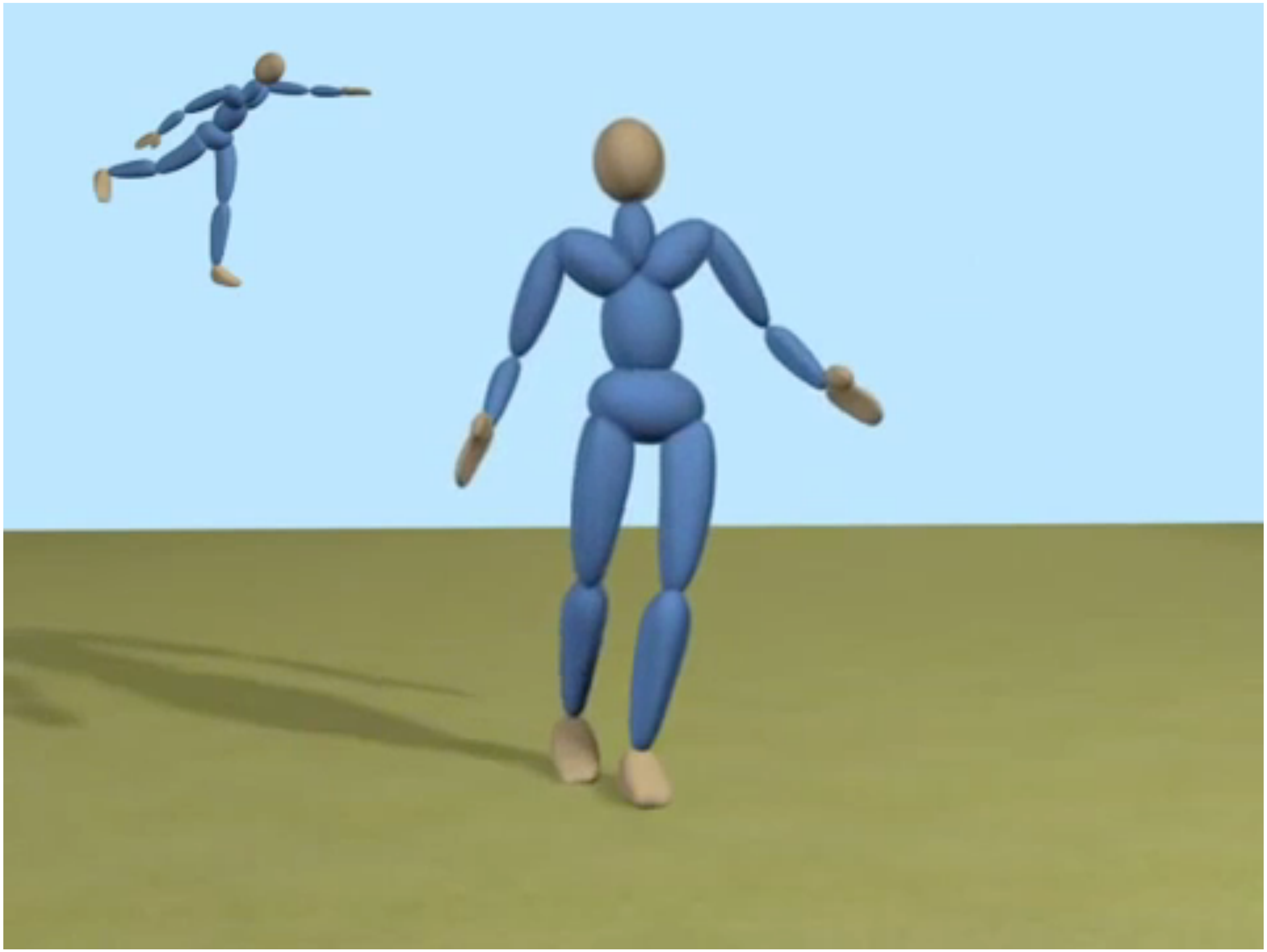
vector of joint angles for all joint (to optimize)

desired position

- Constraints?
  - Could limit range of motion of a joint

- How to optimize for joint angles:
  - Compute gradient of objective with respect to joint angles
  - Apply gradient descent

Many uses of optimization in animation (and graphics in general)



Sumit Jain, Yuting Ye, and C. Karen Liu, *"Optimization-based Interactive Motion Synthesis"*

Summary

- **Kinematics: how objects move, without regard to forces that create this movement**

- **Today: multiple ways of obtaining joint motion**
  - **Direct hand authoring of joint angles**
  - **Via measurement (motion capture)**
  - **As a result of solving for angles that yield a particular higher level result (inverse kinematics)**

- ***Acknowledgements: thanks to Keenan Crane, Ren Ng, Mark Pauly, Steve Marschner, Tom Funkhouser, James O'Brien for presentation resources***