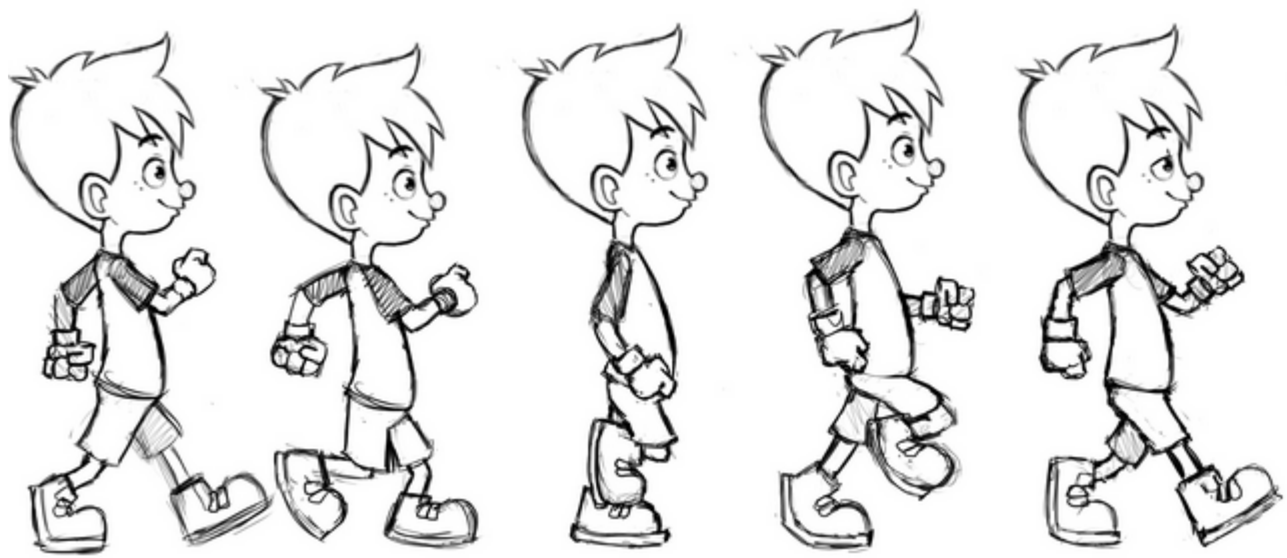
Character Animation and Skinning

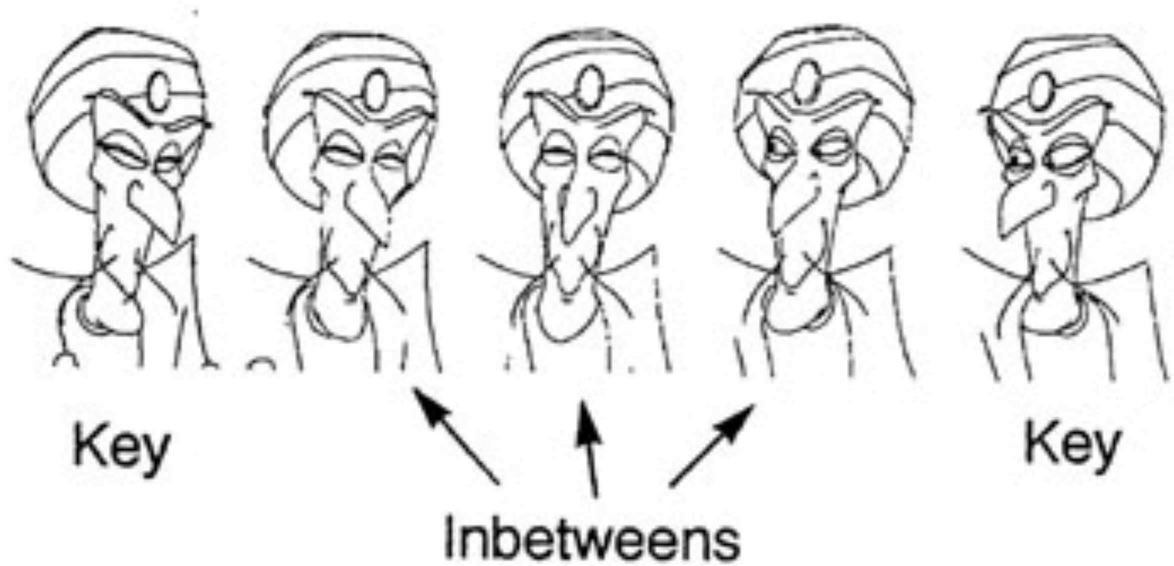
Animation

Motion over time



Traditional Character Animation

Lead animator draws sparse **key frames**



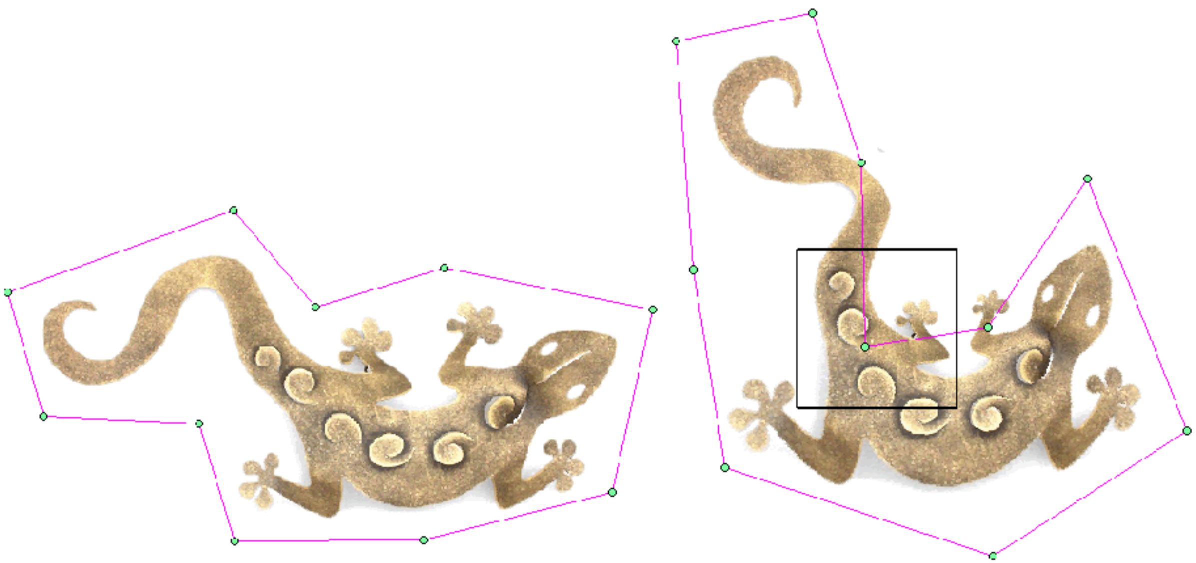
Secondary artists fill in (by hand) the intermediate frames: **in-betweening**

Computer Character Animation

How to in-between automatically on a 2D sprite?

Cage-Based Animation

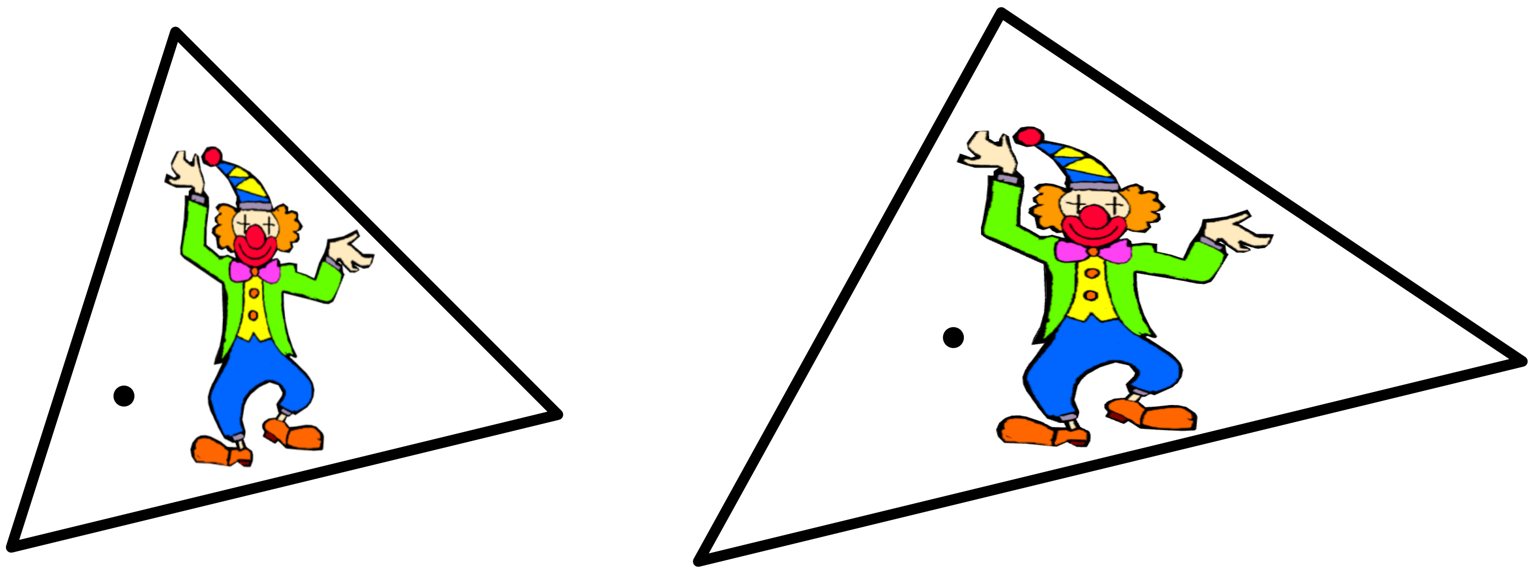
Surround object with **animation cage**



Moving the cage moves interior points

Simplest Cage: Triangle

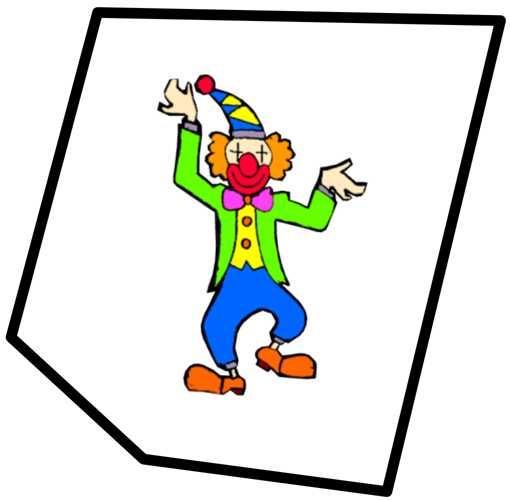
Use **barycentric interpolation**



Matches points' pixels between triangles

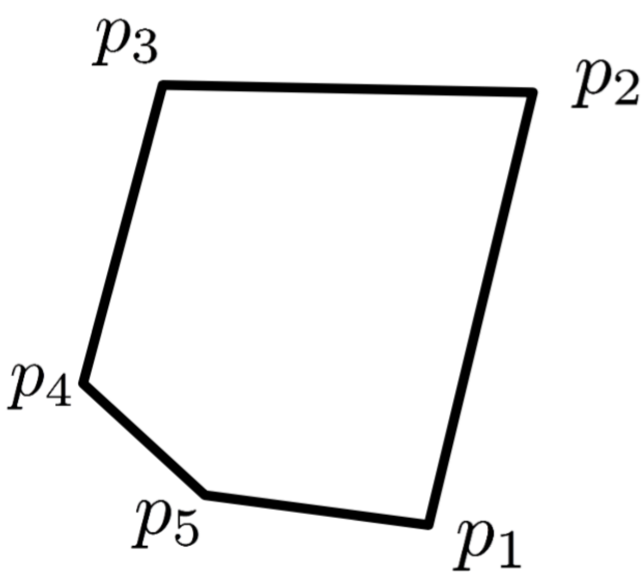
Polygonal Cages

Must generalize barycentric coordinates to arbitrary polygons



Many ways to do this: generalized barycentric coordinates **not** unique

Generalized Barycentric Coordinates



$$q = \sum \alpha_i p_i$$

Partition of unity:

$$1 = \sum \alpha_i$$

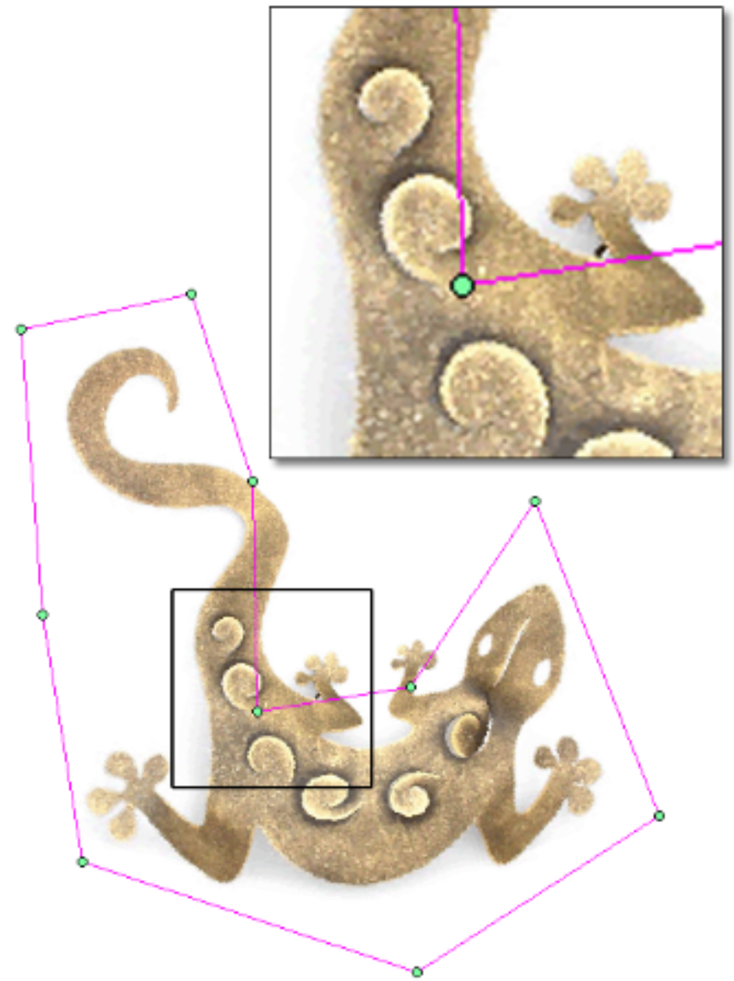
Reproduces the verts:

$$\alpha_i(p_j) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

Polygonal Cages

Other properties:

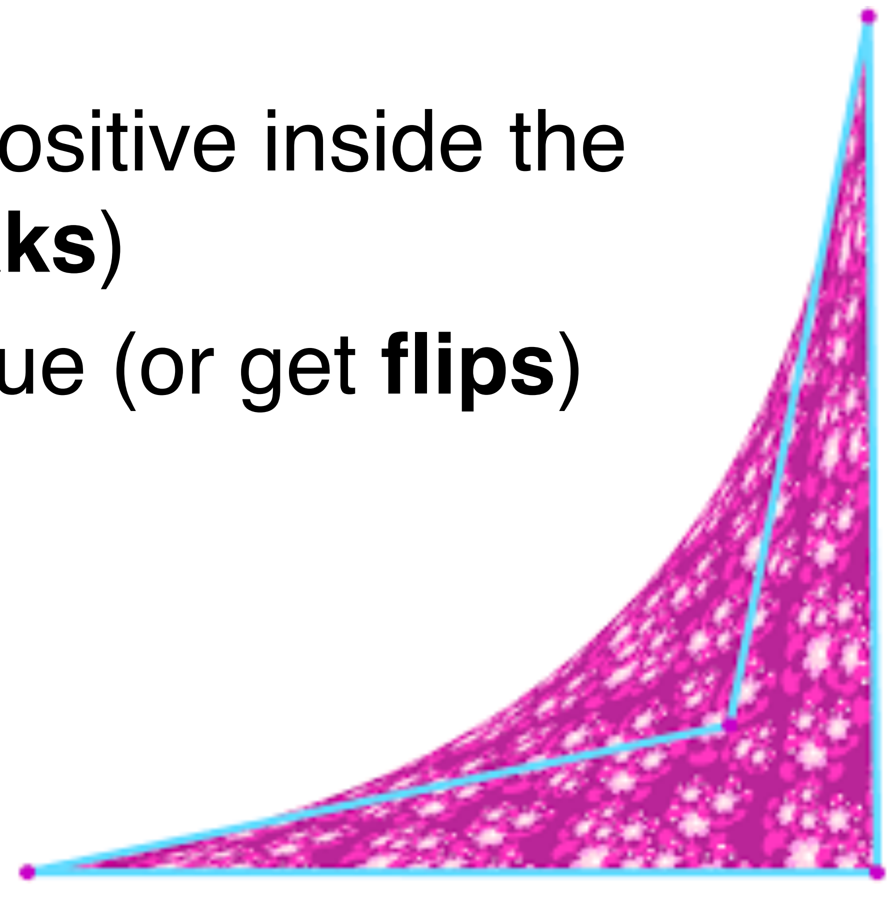
1. Weights must be positive inside the polygon (or get **leaks**)



Polygonal Cages

Other properties:

1. Weights must be positive inside the polygon (or get **leaks**)
2. Weights must unique (or get **flips**)



Polygonal Cages

Other properties:

1. Weights must be positive inside the polygon (or get **leaks**)
2. Weights must unique (or get **flips**)
3. Smooth
4. Easy to compute

Some Possible Schemes

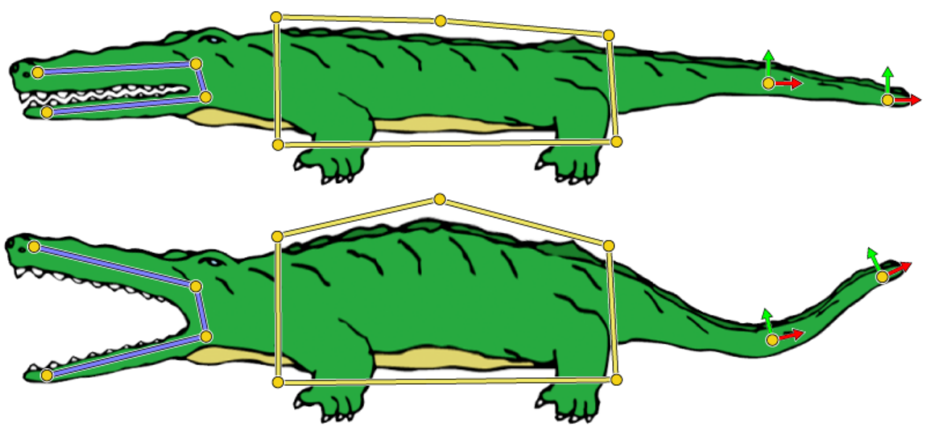
Wachspress Coordinates

Mean-value Coordinates

Green Coordinates

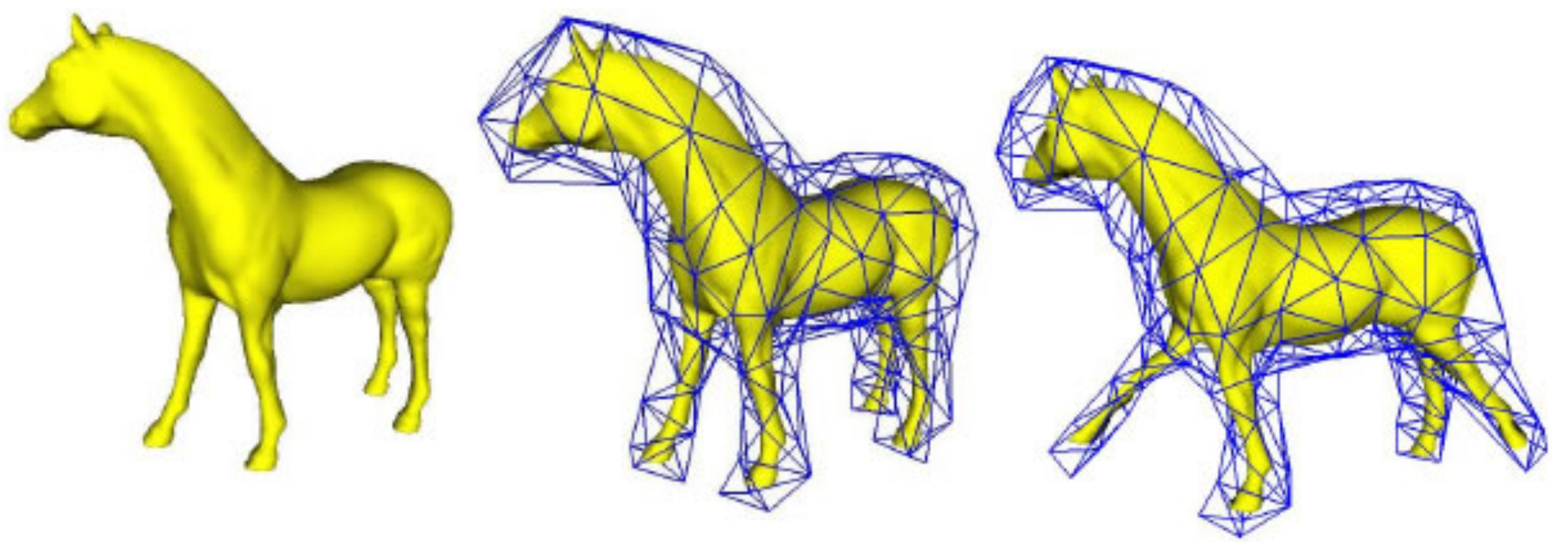
Bounded Biharmonic Weights

etc…



Cage-Based Animation

Extends to 3D from 2D naturally



Full control, but not intuitive

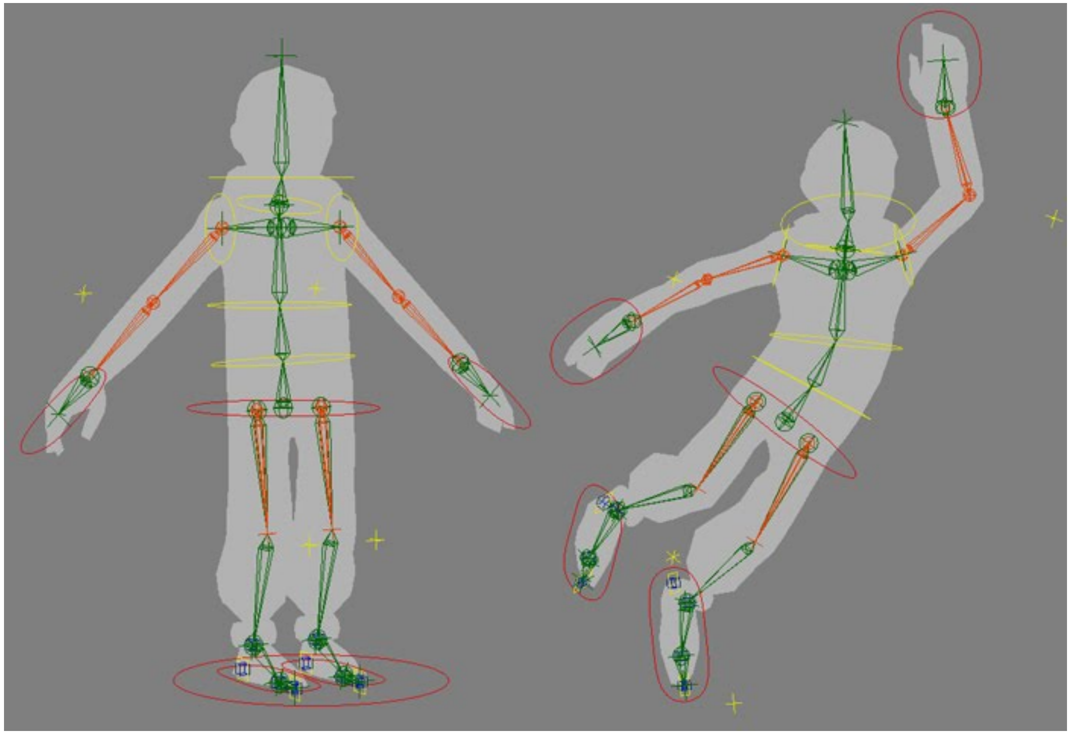
Handle-Based Animation

Pick special points (**handles**) on object

Moving handles moves nearby points

Character Rigs

Skeletons inside the geometry



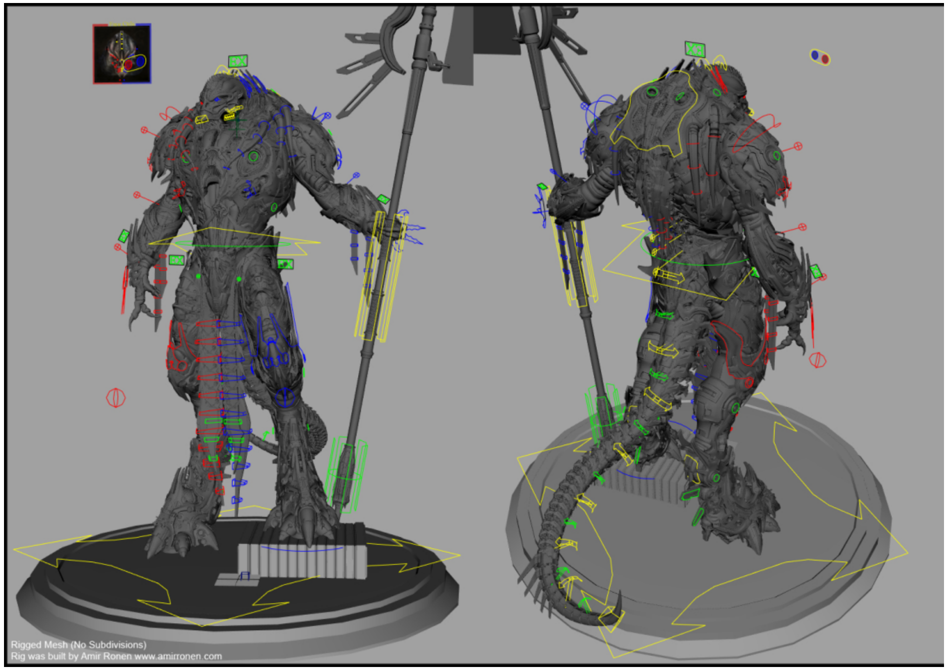
moving bones moves surrounding geometry

the industry standard for character animation

how to build rig?

Building a Rig

Usually done by hand using Maya etc.



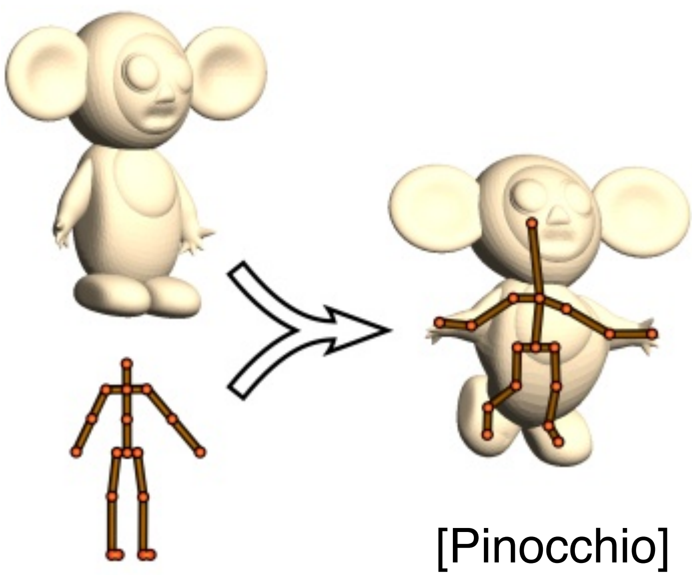
Expressiveness/complexity tradeoff

Rigging in Practice

https://www.youtube.com/watch?v=WxZz-yH-mKU

Building a Rig

Some automatic tools exist…



[Pinocchio]

[Mixamo]

Mixamo Demo

https://www.mixamo.com/

Automatic rigging can work well for humans/humanlike objects

- Assumes bipedal with standard placement and orientation of joints

Not so impressive for arbitrary characters…

https://www.youtube.com/watch?v=fG_ErhAeROU
(apologist edition)

Data Needed for Rigging

- Mesh data exists in world space in A-pose/T-pose
- Skeleton defines hierarchy of bone angles and lengths in A-pose
- Animation information represents changes in skeleton hierarchy

(Christoph Schoch)

Rigging Goal

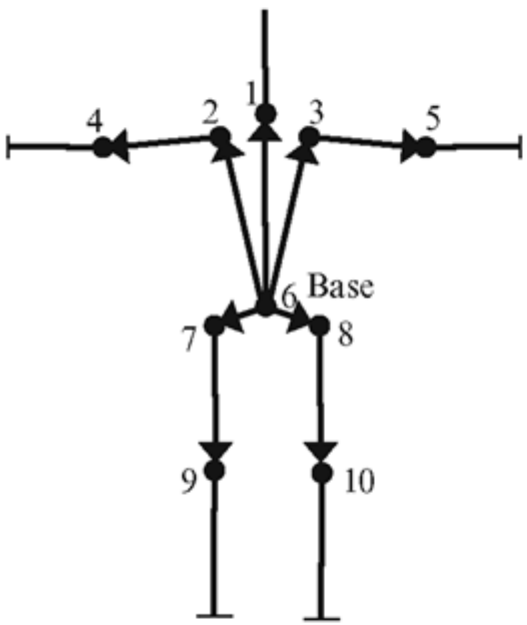
Take vertex data in initial pose **world** coordinates and convert to animation pose **world** coordinates

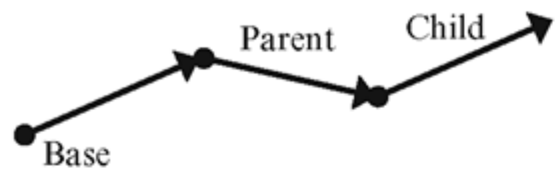
- Need to take **world** initial pose, apply **local** animation pose changes, then convert back to final **world** position

How to do this?

Representing a Rig

Tree of **bones** connected by **joints**





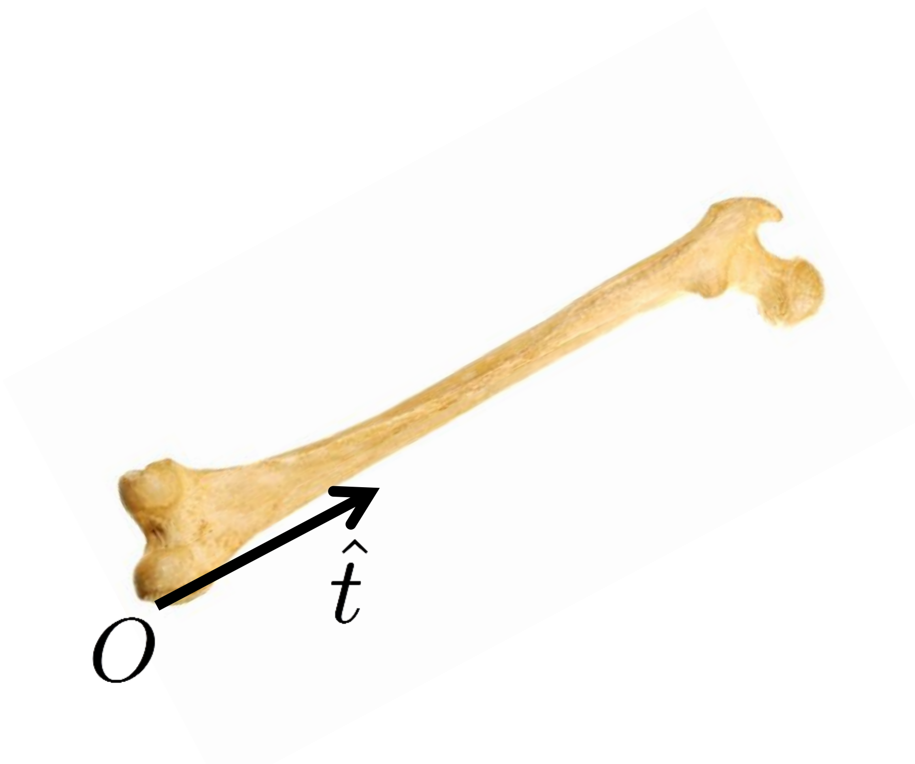
bones have two endpoints

- first attached to **parent**

Bone Local Coordinates

Origin O

One natural direction: **tangent** axis $\hat{t}$

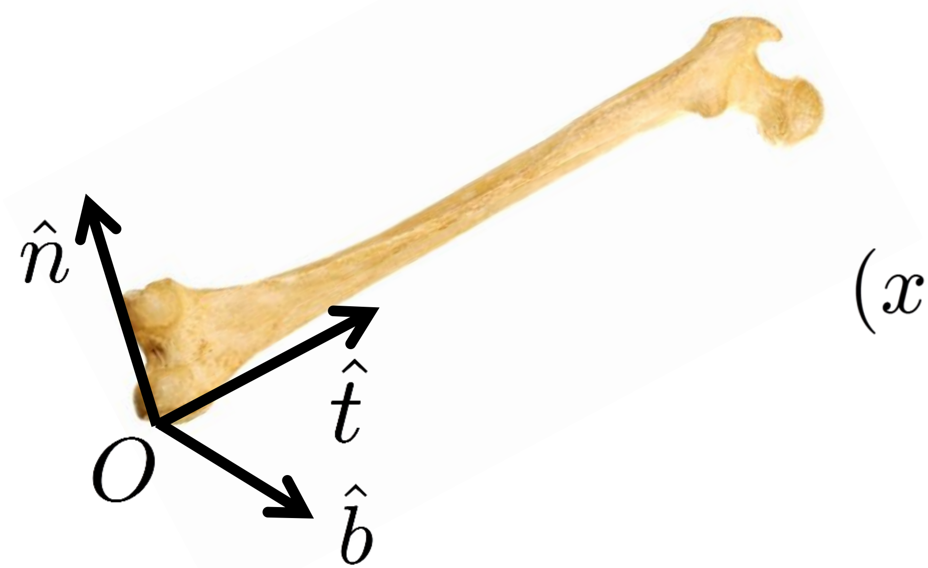


Bone Local Coordinates

Origin O

One natural direction: **tangent** axis $\hat{t}$

Two perpendicular directions: $\hat{n}, \hat{b}$



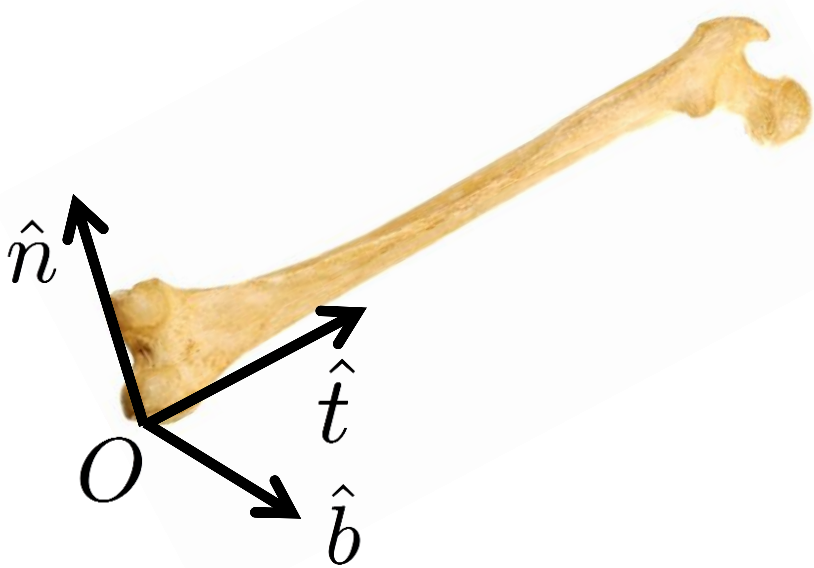
$$(x, y, z) = x\hat{t} + y\hat{n} + z\hat{b}$$

Bone Local Coordinates

Origin O

One natural direction: **tangent** axis $\hat{t}$

Two perpendicular directions: $\hat{n}, \hat{b}$



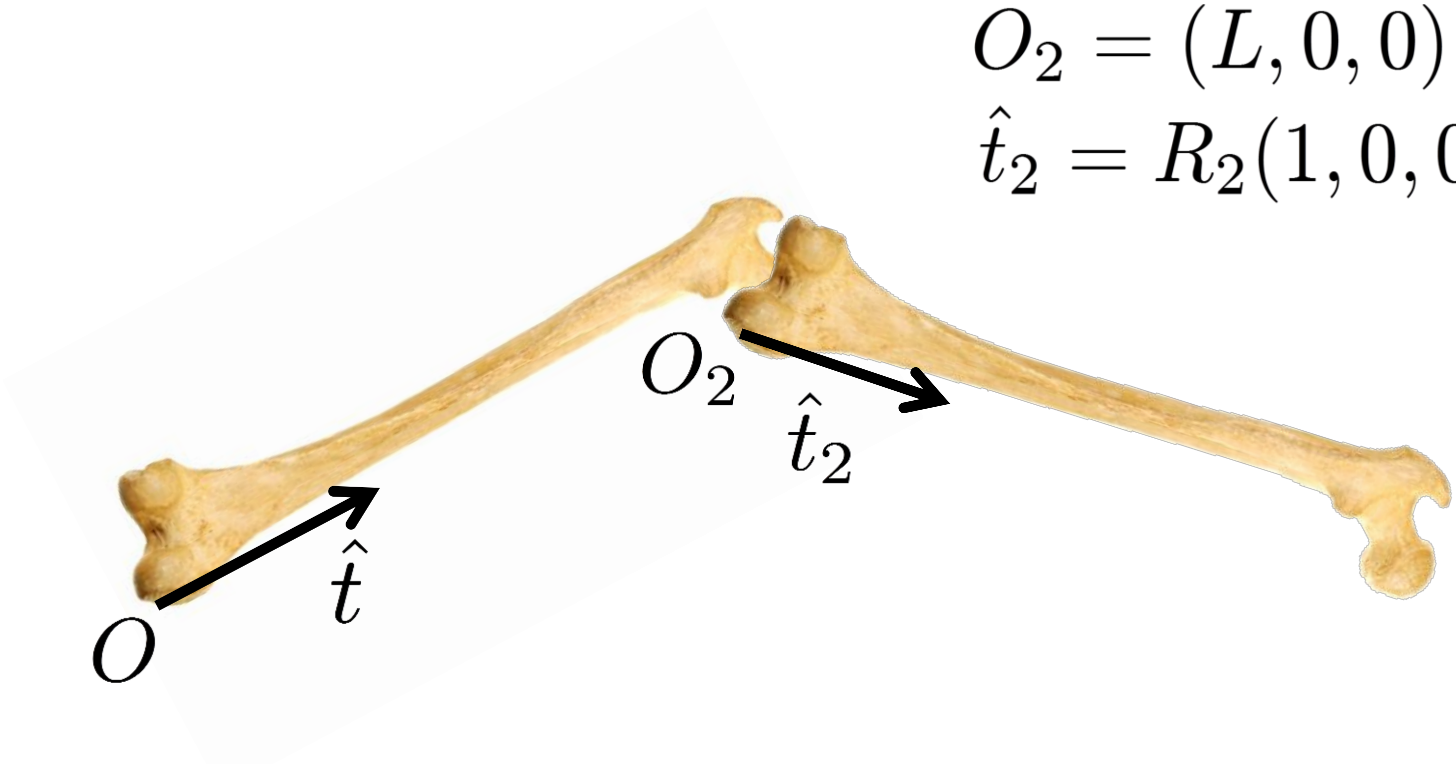
$$(x, y, z) = x\hat{t} + y\hat{n} + z\hat{b}$$

second endpoint: $(L, 0, 0)$

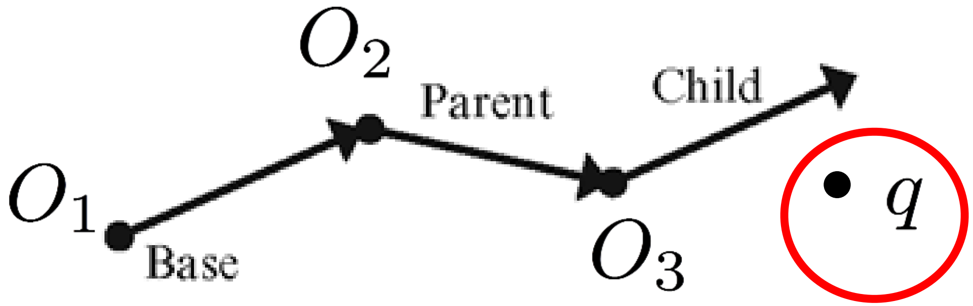
Bone Local Coordinates

Child bone can be expressed in terms of **parent** coordinate system

$$O_2 = (L, 0, 0) = T_2 O$$
$$\hat{t}_2 = R_2(1, 0, 0) = R_2 \hat{t}$$



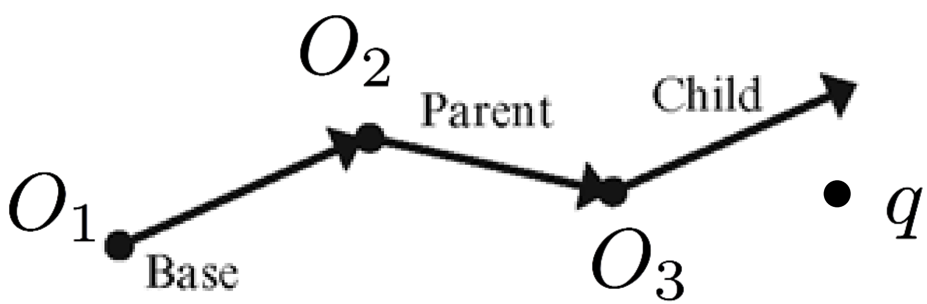
Bone to World Coordinates



In local coordinates:

$$q = (x, y, z) = O_3 + x\hat{t}_3 + y\hat{n}_3 + z\hat{b}_3$$

Bone to World Coordinates



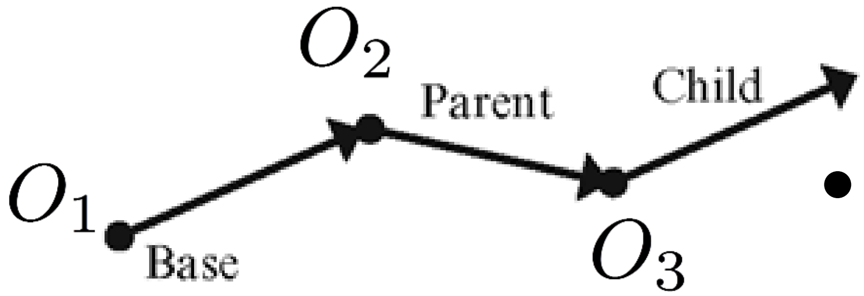
In local coordinates:

$$q = (x, y, z) = O_3 + x\hat{t}_3 + y\hat{n}_3 + z\hat{b}_3$$

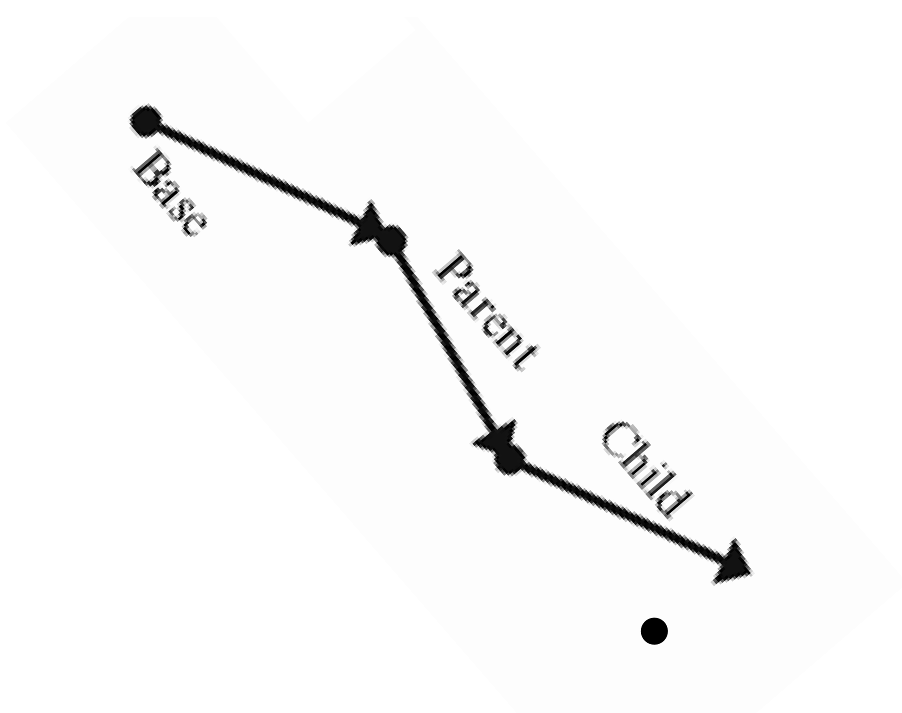
In world coordinates:

$$q = T_1 R_1 T_2 R_2 T_3 R_3 \begin{bmatrix} x \\ y \\ z \end{bmatrix} = M_3 \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Forward Kinematics



- $q = T_1 R_1 T_2 R_2 T_3 R_3 \begin{bmatrix} x \\ y \\ z \end{bmatrix}$



changing R_1 **also** changes child coordinate systems

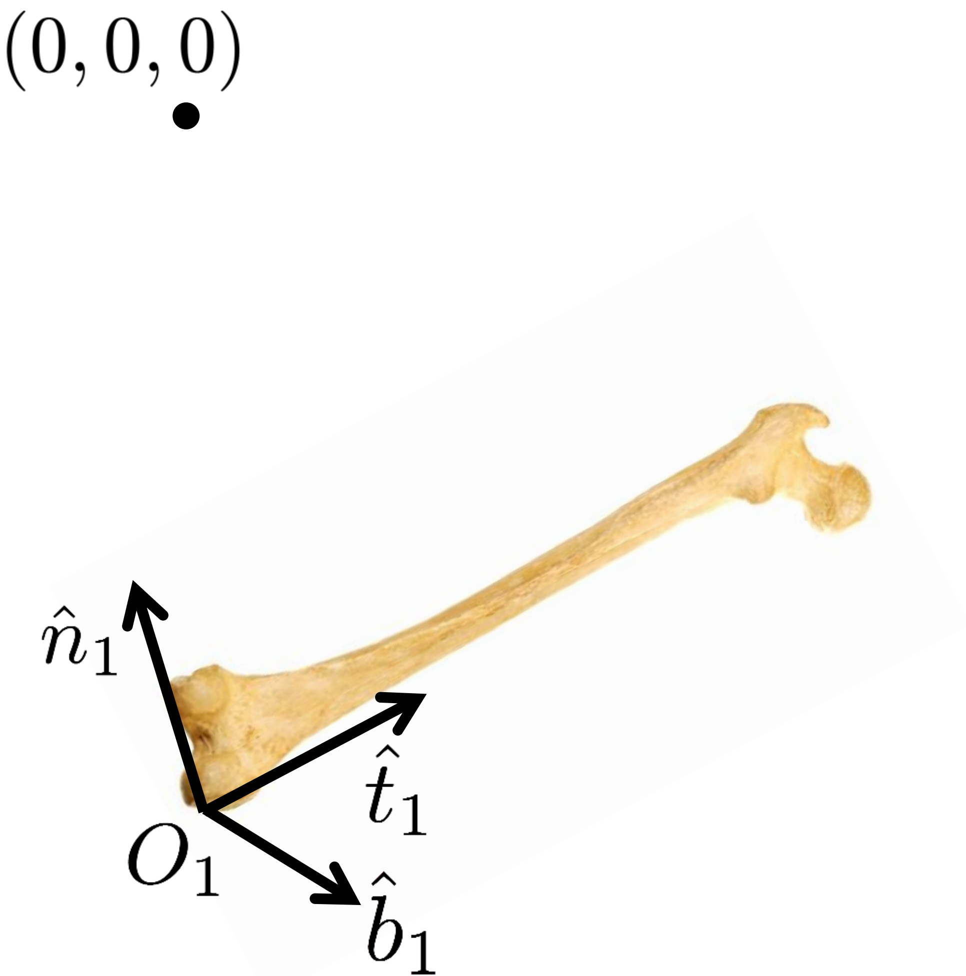
Bones or Joints?

Which works better? A hierarchy of bones or a hierarchy of joints? (i.e. what should we store in our tree?)

Bones or Joints?

- They accomplish the same thing!
- A tree of joints may be easier to construct initially but harder to reconstruct during traversal
- Either approach is fine -- just make sure you're consistent and you've thought through the math (I will focus on bone representation)
- ...but don't create hybrid trees with both object representations...

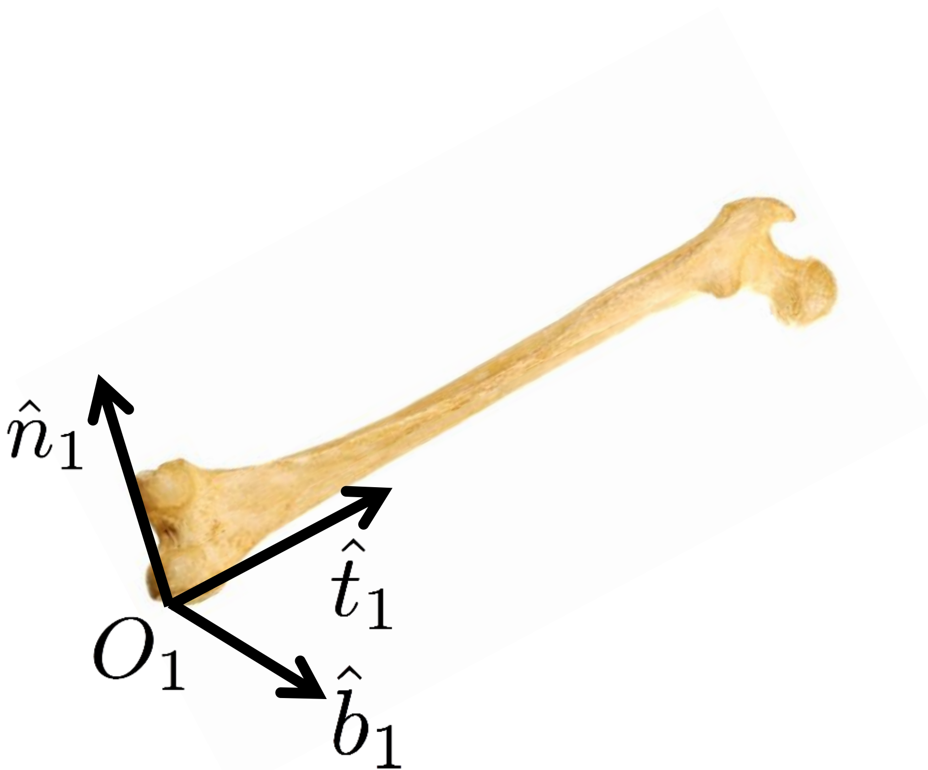
What About the Base?



What About the Base?

$(0, 0, 0)$

write origin & axes in **world coordinates**, then



$$T_1 = T_{O_1}$$

$$R_1 = \left[\begin{array}{c|c|c} \hat{t}_1 & \hat{n}_1 & \hat{b}_1 \end{array} \right]$$

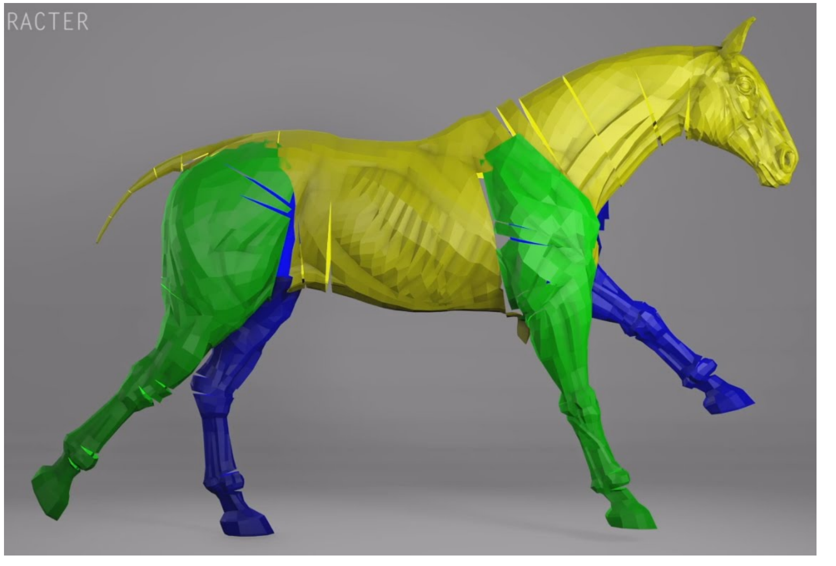
Additional Reading

https://www.gamedev.net/resources/_/technical/graphics-programming-and-theory/skinned-mesh-animation-using-matrices-r3577

Skinning

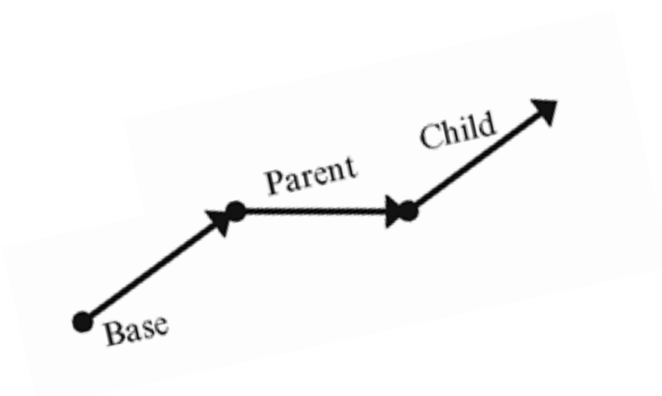
Moving bones moves the character

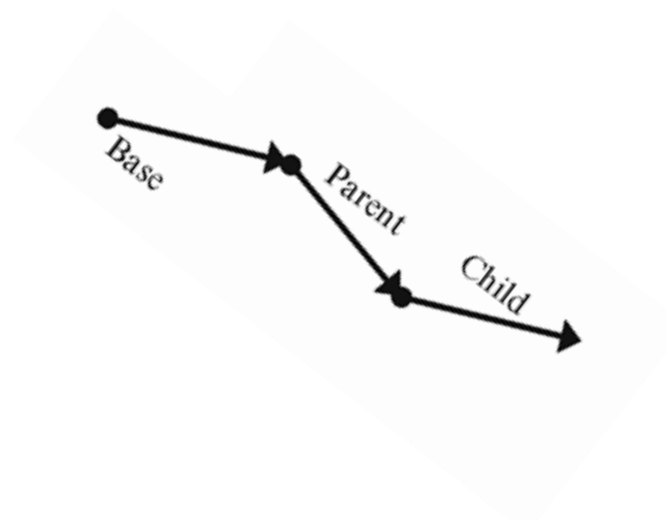
Closer bones have more influence



Nearest-Bone Skinning

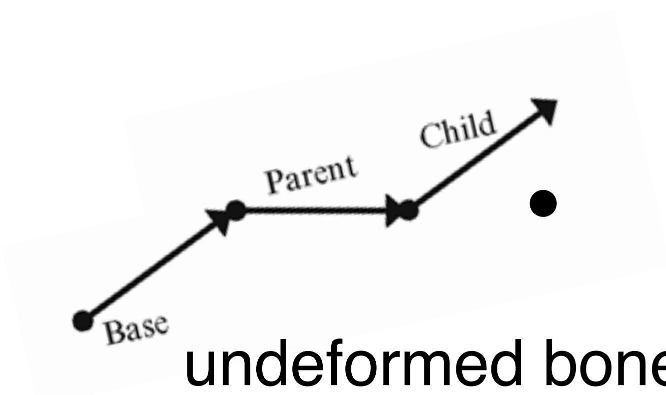
Given: **undeformed** (rest) skeleton and **deformed** skeleton



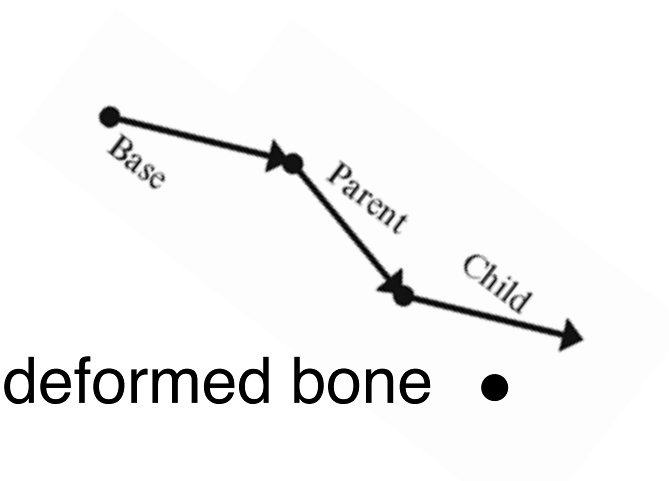


Coordinate Systems

world



undeformed bone

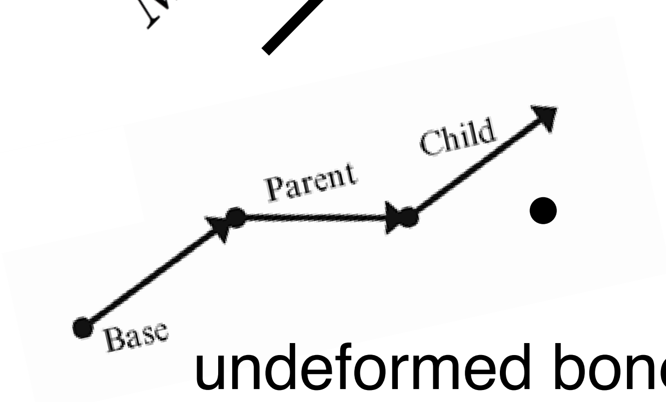


deformed bone

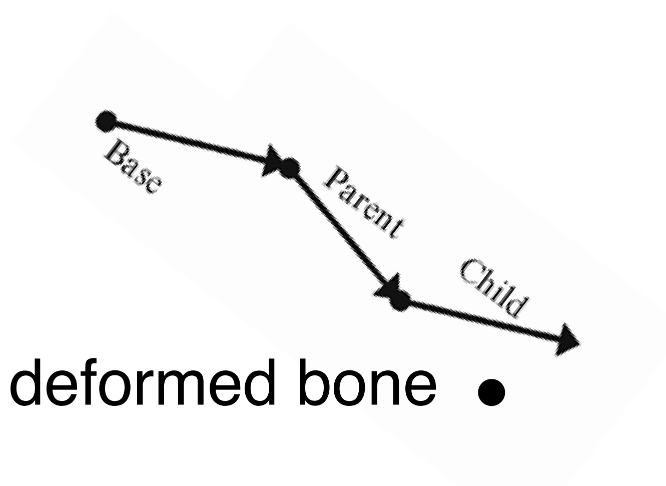
Coordinate Systems

$$M_3 = T_1 R_1 T_2 R_2 T_3 R_3$$

world

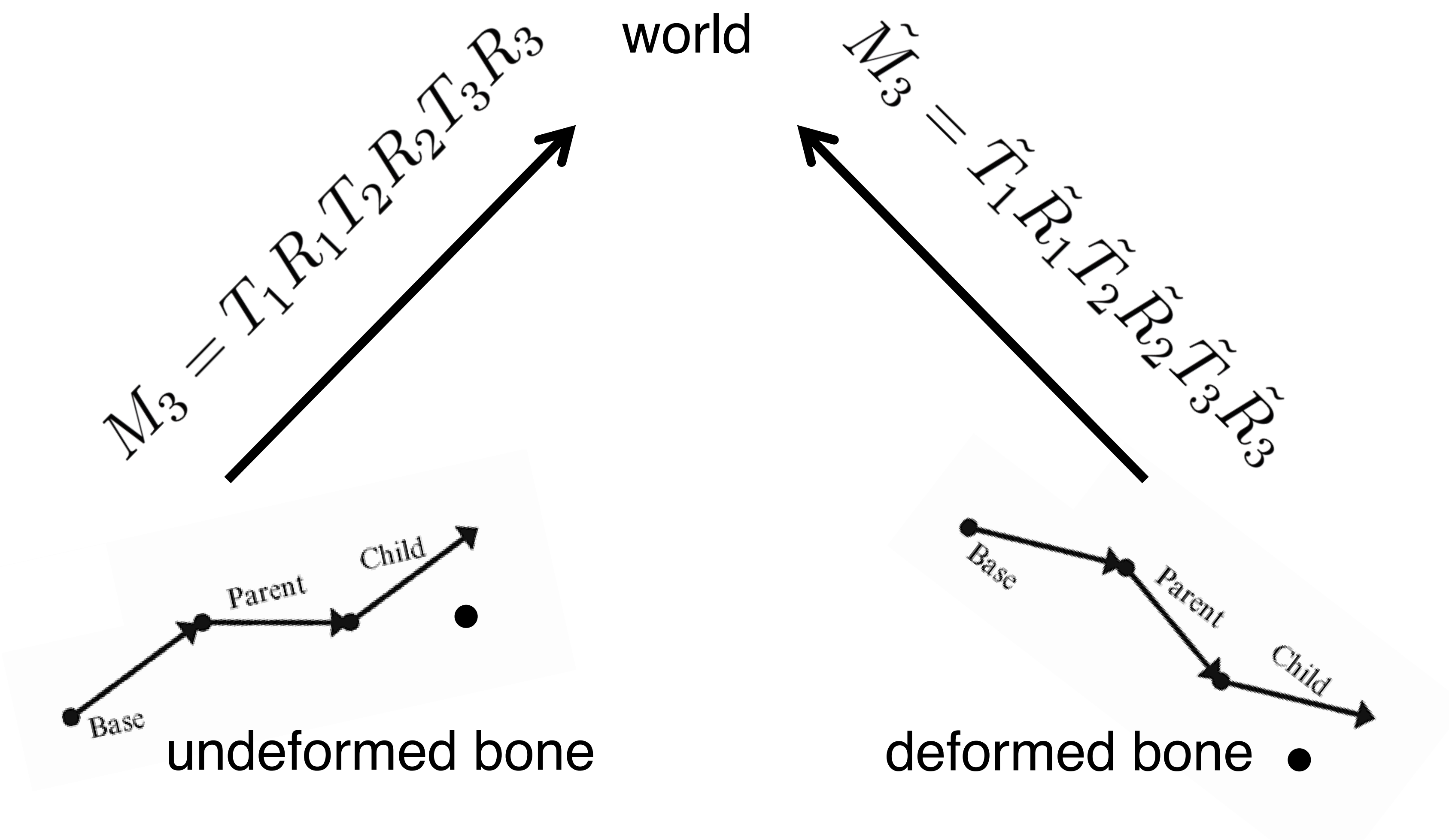


undeformed bone

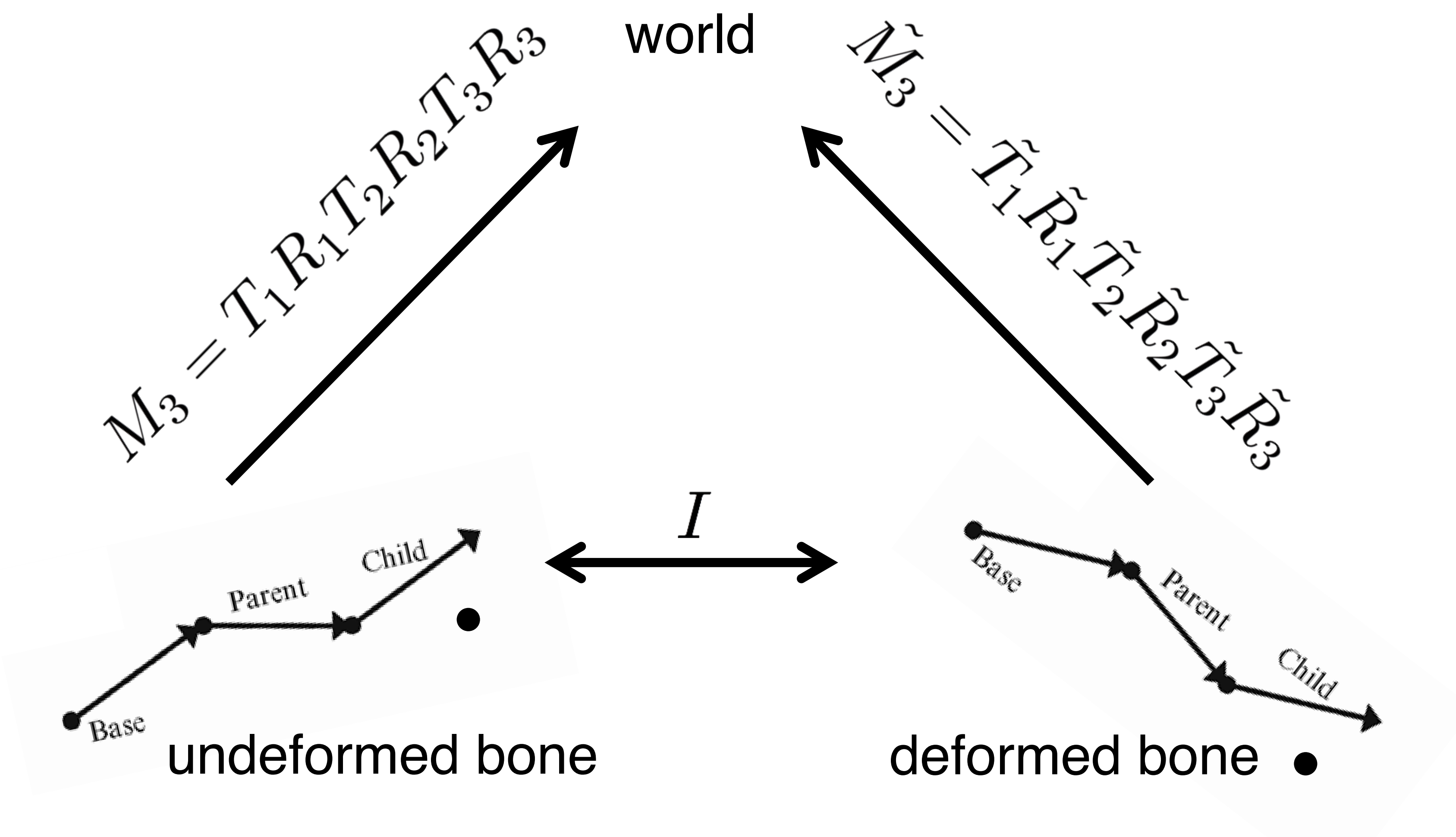


deformed bone

Coordinate Systems



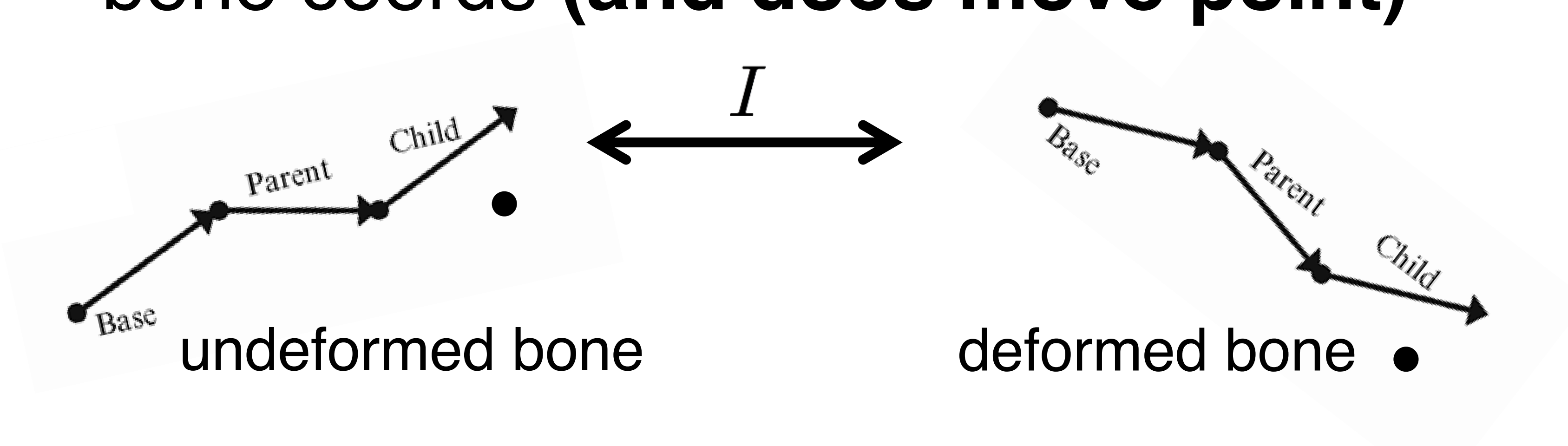
Coordinate Systems



Coordinate Systems

Key (and confusing) point:

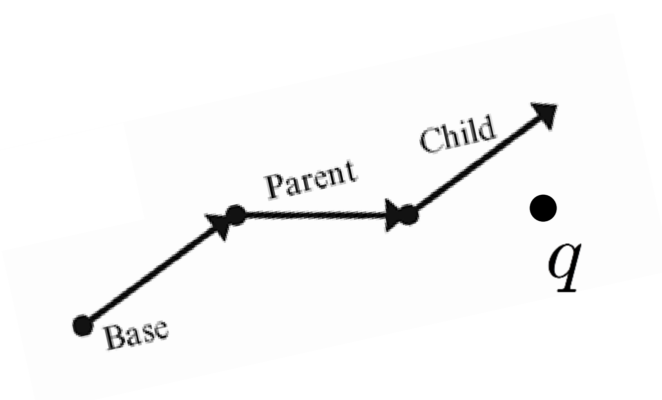
- M_3 maps from undeformed local to world coords **(doesn't move point)**
- **Identity** maps undeformed to deformed bone coords **(and does move point)**

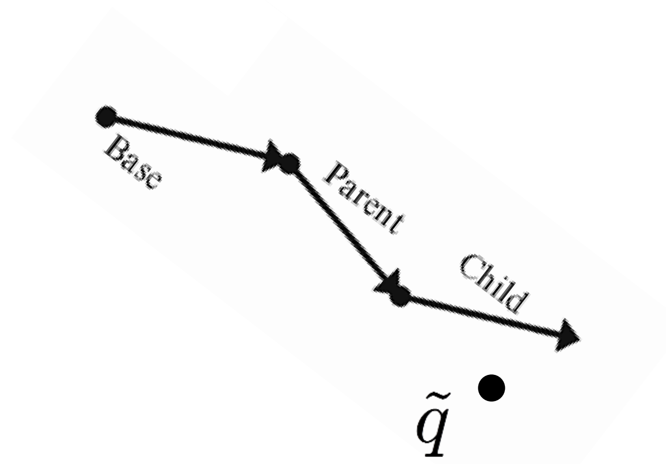


Nearest-Bone Skinning

Undeformed to deformed skin position (world coordinates):

$$\tilde{q} = \tilde{M}_3 M_3^{-1} q$$



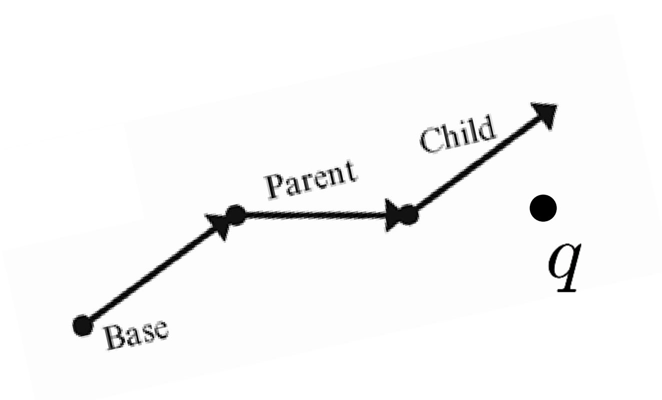


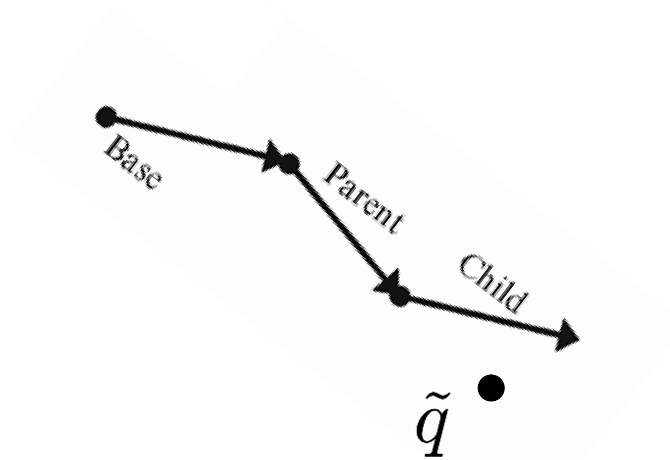
Nearest-Bone Skinning

Undeformed to deformed skin position (world coordinates):

$$\tilde{q} = \tilde{M}_3 M_3^{-1} q$$

changes during animation





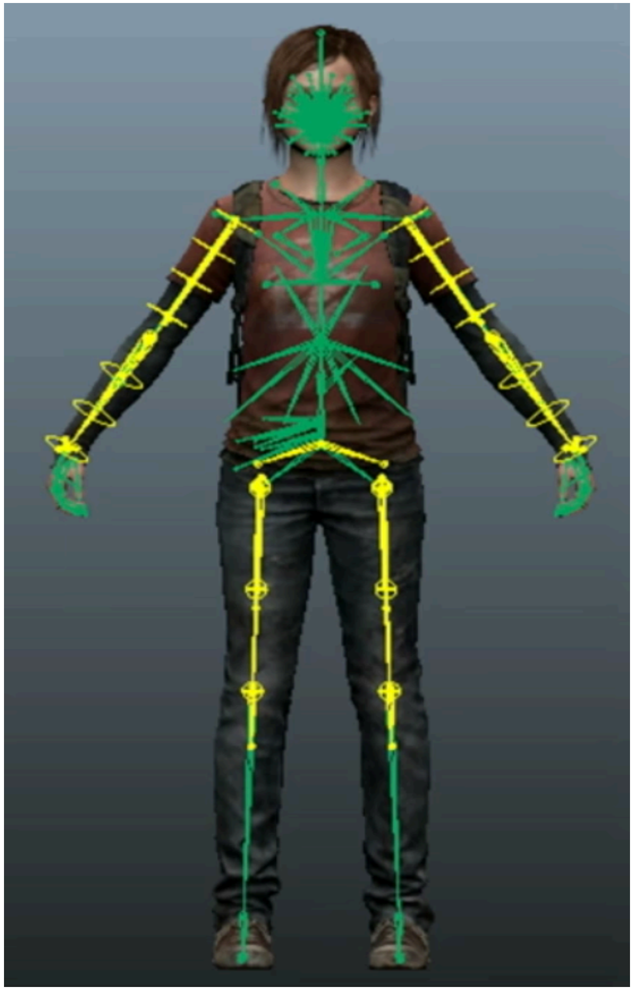
What about World Space Transforms?

- Accomplishes the same thing
  - Offset mapping not required
- Transformations to a parent bone must be applied explicitly to all children
  - Potentially inefficient
  - Potential for massive performance hit

Modern Rig Example

Hero Rig in Last of Us:

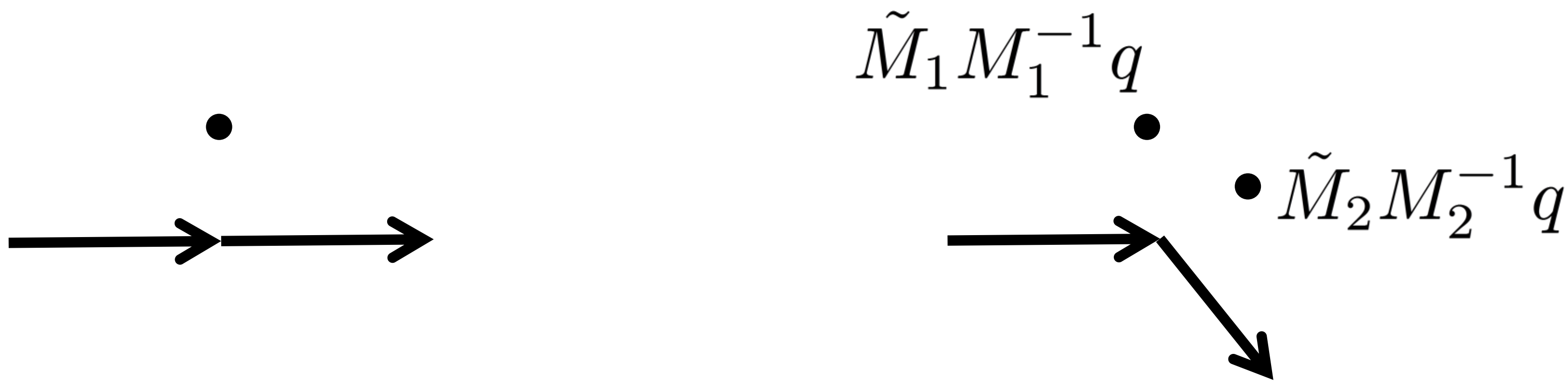
- 326 joints
- 85 runtime driven
- 241 animation sampled (baked)



https://youtu.be/myZcUvU8YWc

Problems with Nearest-Bone

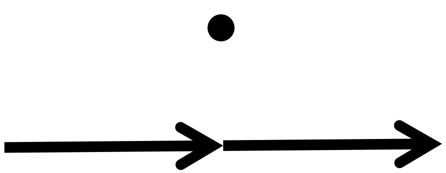
Which bone does point belong to?

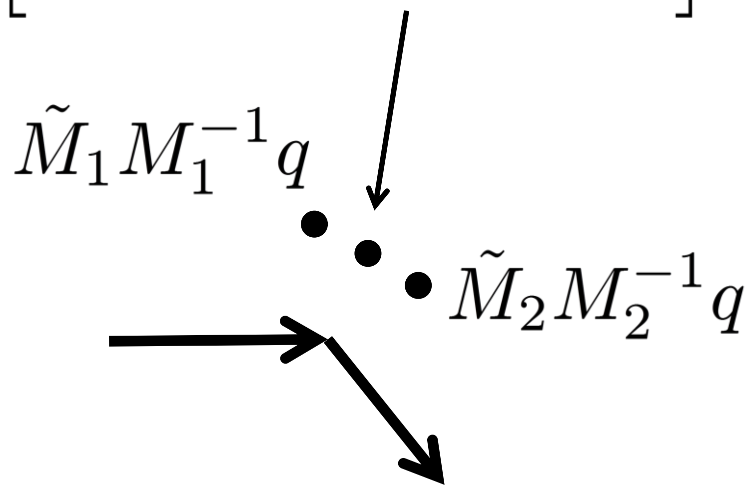


Problems with Nearest-Bone

Which bone does point belong to?

One solution: **average** $\left[\frac{1}{2}\tilde{M}_1 M_1^{-1} + \frac{1}{2}\tilde{M}_2 M_2^{-1}\right] q$



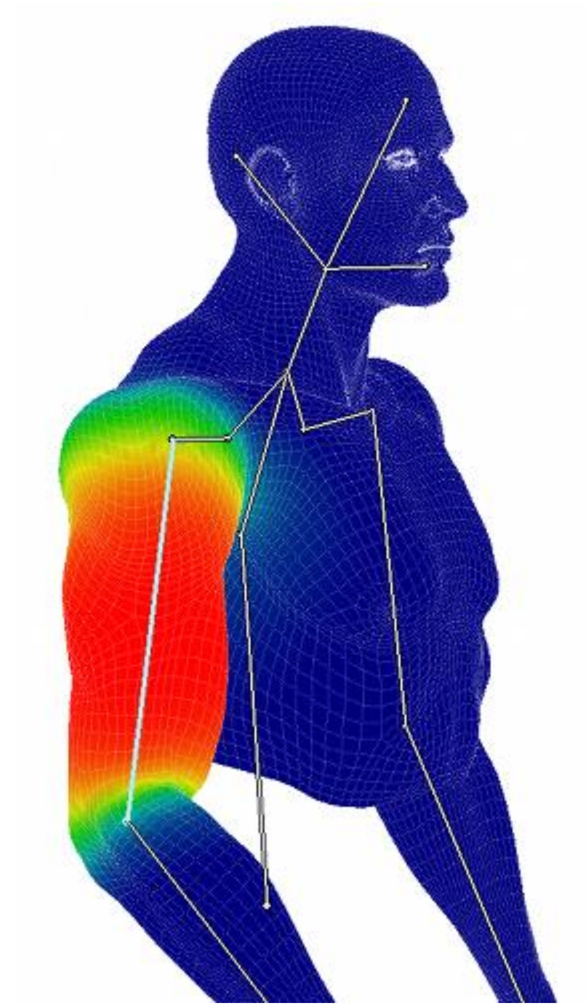


Linear-Blend Skinning

Each vertex feels **weighted average** of each bone's transformations

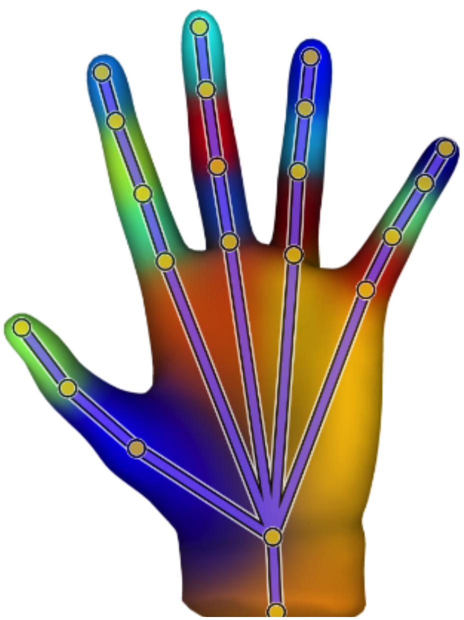
$$\tilde{q}_i = \sum_{\text{bones } j} w_{ij} \tilde{M}_j M_j^{-1} q_i$$

Nearby bones have higher weight



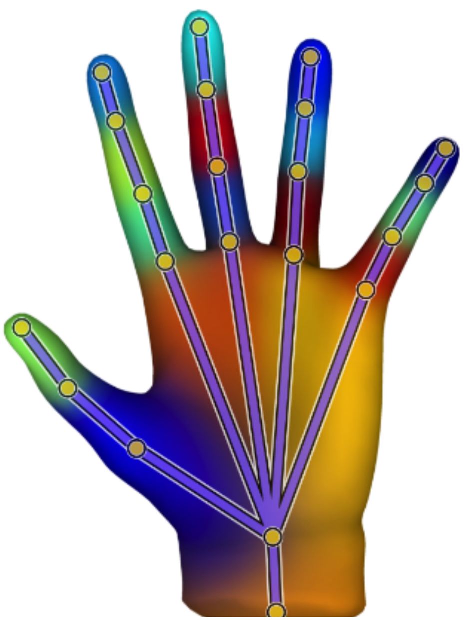
Linear-Blend Skinning

How to determine **skinning weights** w?



Linear-Blend Skinning

How to determine **skinning weights** w?

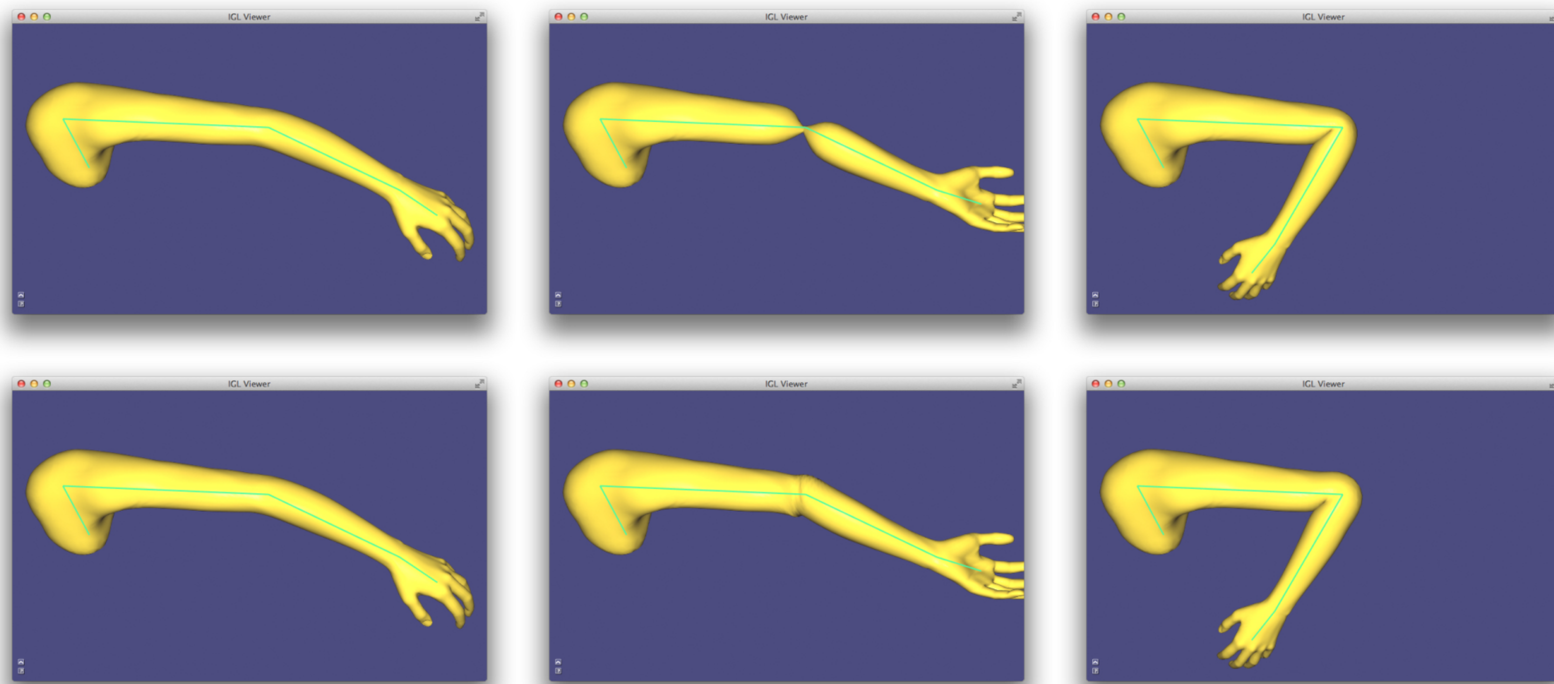


- Use only nearest bone
- Spatially blend the weights
- In practice: paint weights by hand

Painting Weights

https://www.youtube.com/watch?v=cuaXDkbg4QA

The “Arm Twist” Problem



(Why does this happen?)

Blending Transformations

Each individual bone undergoes a rigid transformation

- Combination rotation and translation

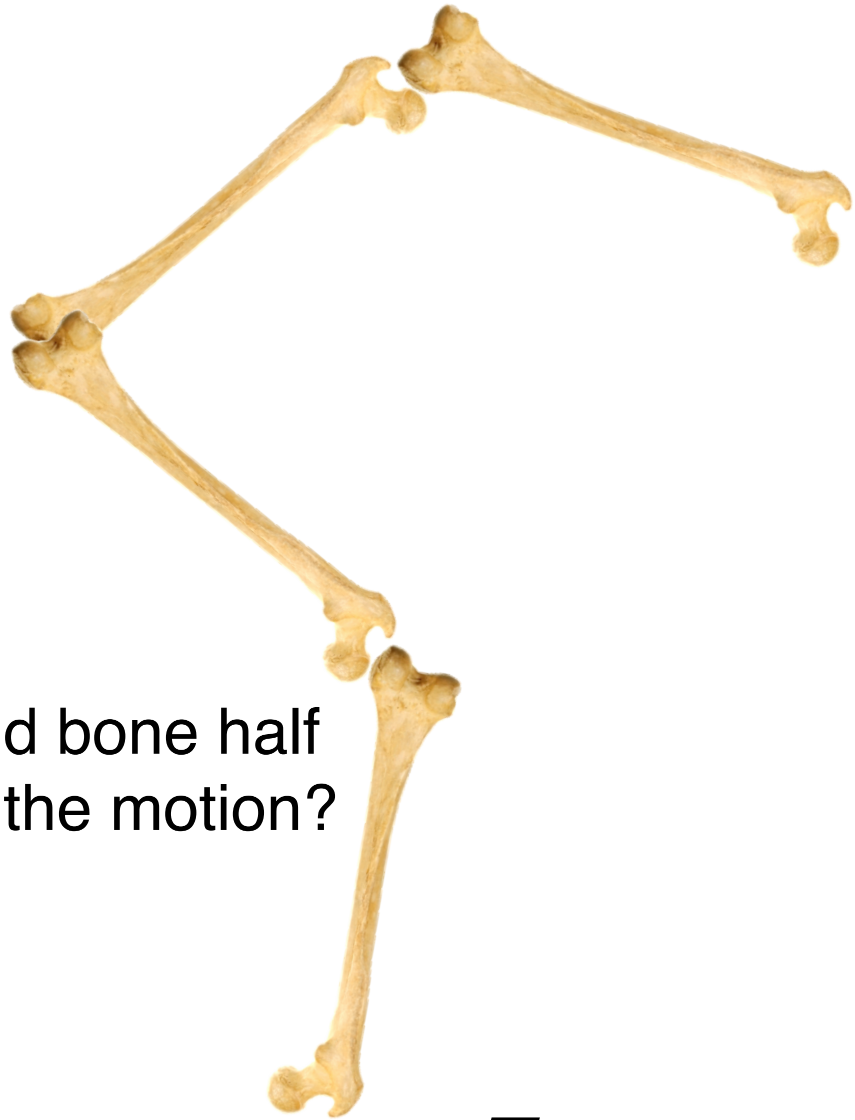
- Linear blend of rigid motions **not rigid**
  - Can introduce shear and scale

Separate Transforms: Problem

Blended transformations **not** coordinate-independent

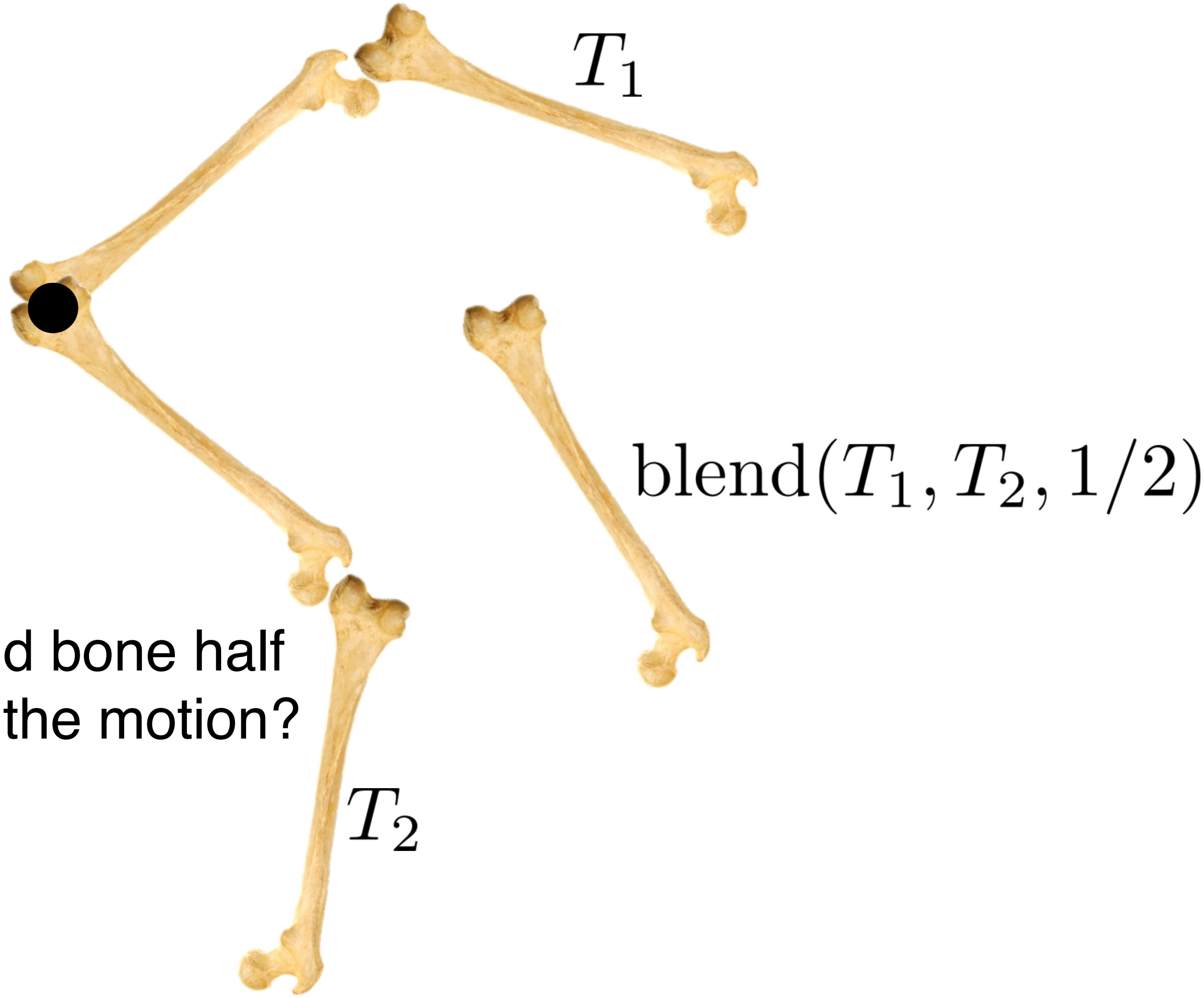
- Different origin positions in bone hierarchy result in different blends

Separate Transforms: Problem



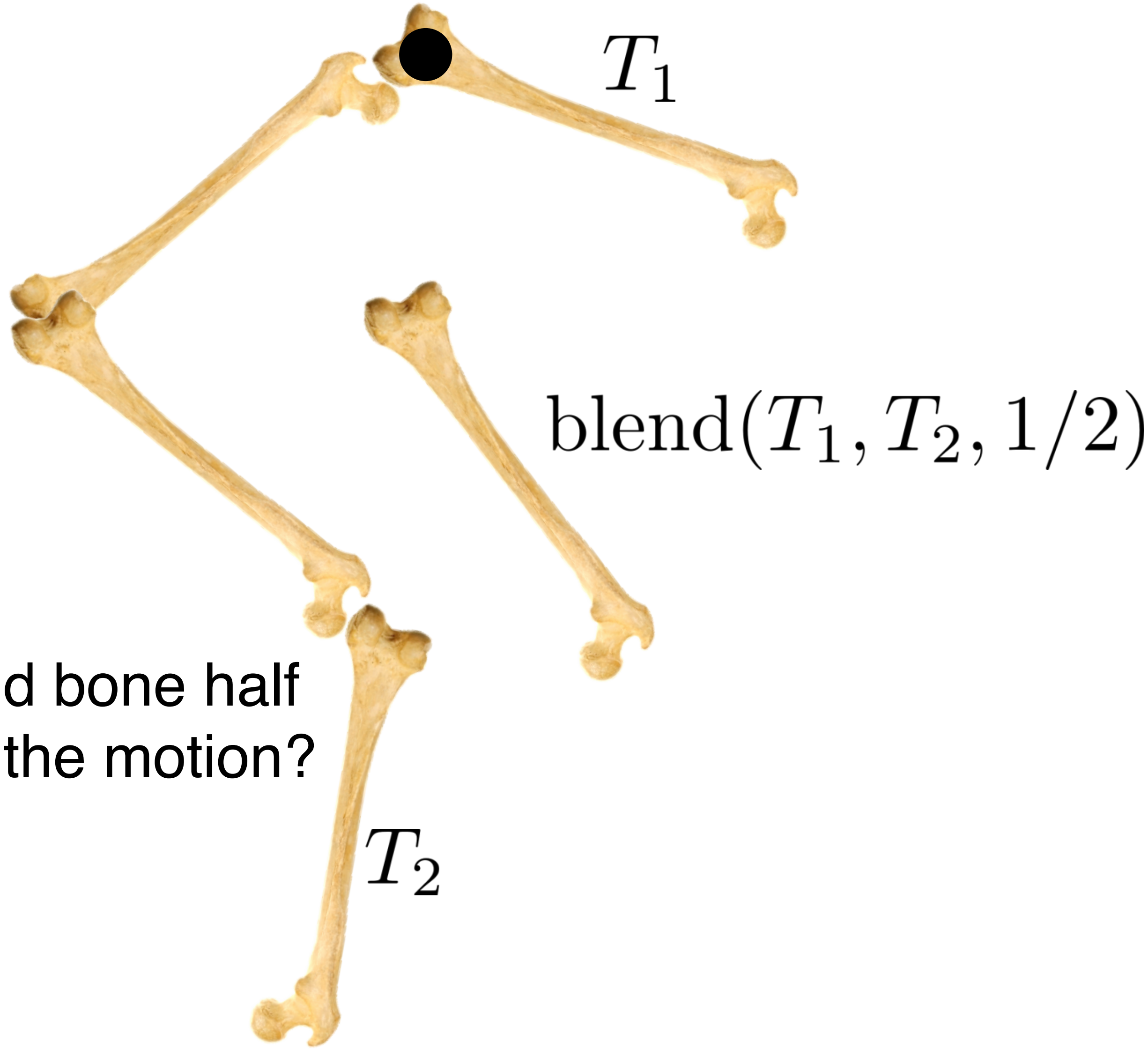
where is the child bone half way in between the motion?

Separate Transforms: Problem



where is the child bone half way in between the motion?

Separate Transforms: Problem



where is the child bone half way in between the motion?

Separate Transforms: Problem

Blended transformations **not** coordinate-independent

- Different origin positions in bone hierarchy result in different blends

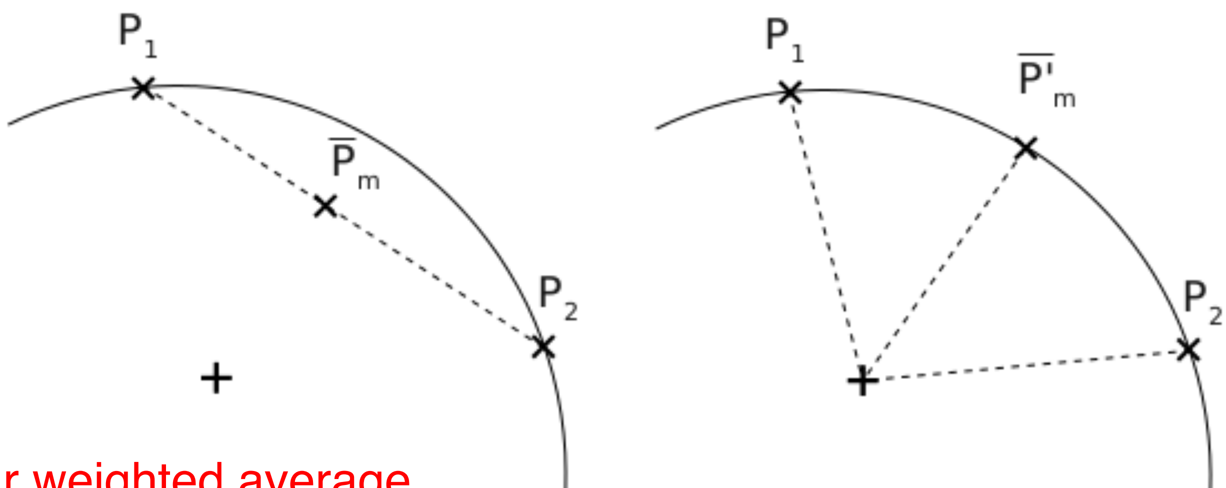
Must unify translation and rotation into single state

- Blend **centers of rotation**

Dual Quaternion Skinning

Prevents loss of volume during rigid motion

Normalize it to surface



Take linear weighted average

https://cs.gmu.edu/~jmlien/teaching/cs451/uploads/Main/dual-quaternion.pdf

Dual Quaternions for Rigid Bodies

- Expresses a rotation (encoded in real) and translation (encoded in dual)
- Dual unit is ε

$$\dot{q} = q_r + q_d \varepsilon$$

where
$$q_r = r$$
$$q_d = \frac{1}{2} t r$$
$$\varepsilon^2 = 0$$

Calculating the Dual Quaternion

- Rotation already encoded as a quaternion
  - Maps directly to qr
- Encode translation (X, Y, Z) into quaternion (t) then multiply by rotation to calculate q_d
  - Note t.w = 0

Quaternion multiplication reminder:

$$< w, v > < w', v' > = < ww' - v \cdot v', wv' + w'v + v \times v' >$$

Blending Dual Quaternions

Apply weighted average to dual quaternion then renormalize

$$\dot{\mathbf{q}} = \frac{\sum_{i=1}^{n} w_i \dot{\mathbf{q}}_i}{\| \sum_{i=1}^{n} w_i \dot{\mathbf{q}}_i \|}$$

Apply Dual Quaternions to Rigid Bodies

- Update vertex position and normals based on blended dual quaternions
  - Note: normals still need to be calculated in world space (i.e. use inverse transpose to handle non-uniform scales)

Blended vertex position:

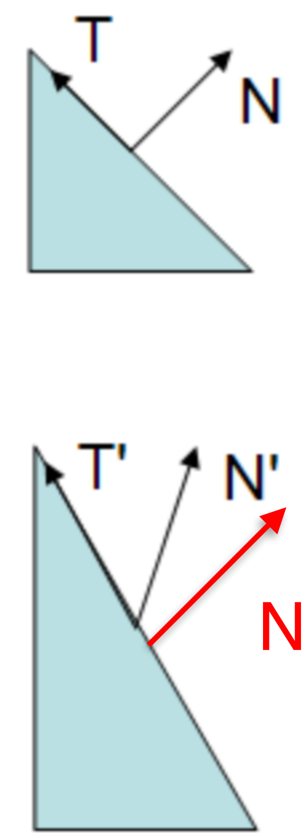
$$v' = v + 2(\overrightarrow{q_r} \times (\overrightarrow{q_r} \times v + q_{r.w}v)) + 2(q_{r.w}\overrightarrow{q_d} - q_{d.w}\overrightarrow{q_r} + \overrightarrow{q_r} \times \overrightarrow{q_d})$$

Blended normal position:

$$n' = n + 2\overrightarrow{q_r} \times (\overrightarrow{q_r} \times n + q_{r.w}n)$$

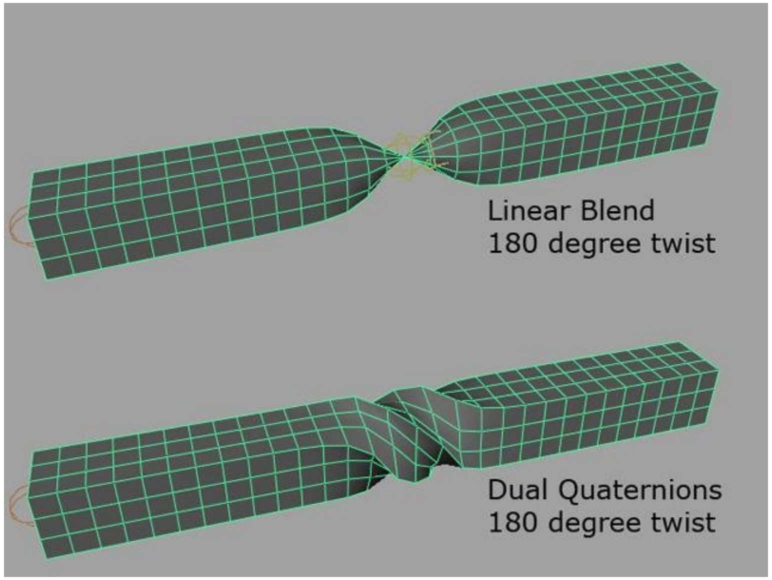
Side note: The "Normal Matrix"

- Matrix provided in fixed function pipeline
  - No longer available in shader pipeline
- Maintains correct direction of normals to surfaces regardless of non-uniform scales
- Full derivation here: http://www.lighthouse3d.com/tutorials/glsl-12-tutorial/the-normal-matrix/



Dual Quaternion Skinning

- No more arm twisting issues
- Less deformation
- The industry standard (used in Maya, etc)



https://www.cs.utah.edu/~ladislav/kavan08geometric/kavan08geometric.pdf

Animation Recap

Most common pipeline:

- build a 3D model of the character
- **rig** the 3D model (build a skeleton inside)
- **skin** the model (determine bone-skin weights)
- animate the bones by specifying **keyframes**; skin moves with them

Animation Recap

Most common pipeline:

- model, rig, skin, animate

Automatic approaches exist for each step

- not great, but getting better