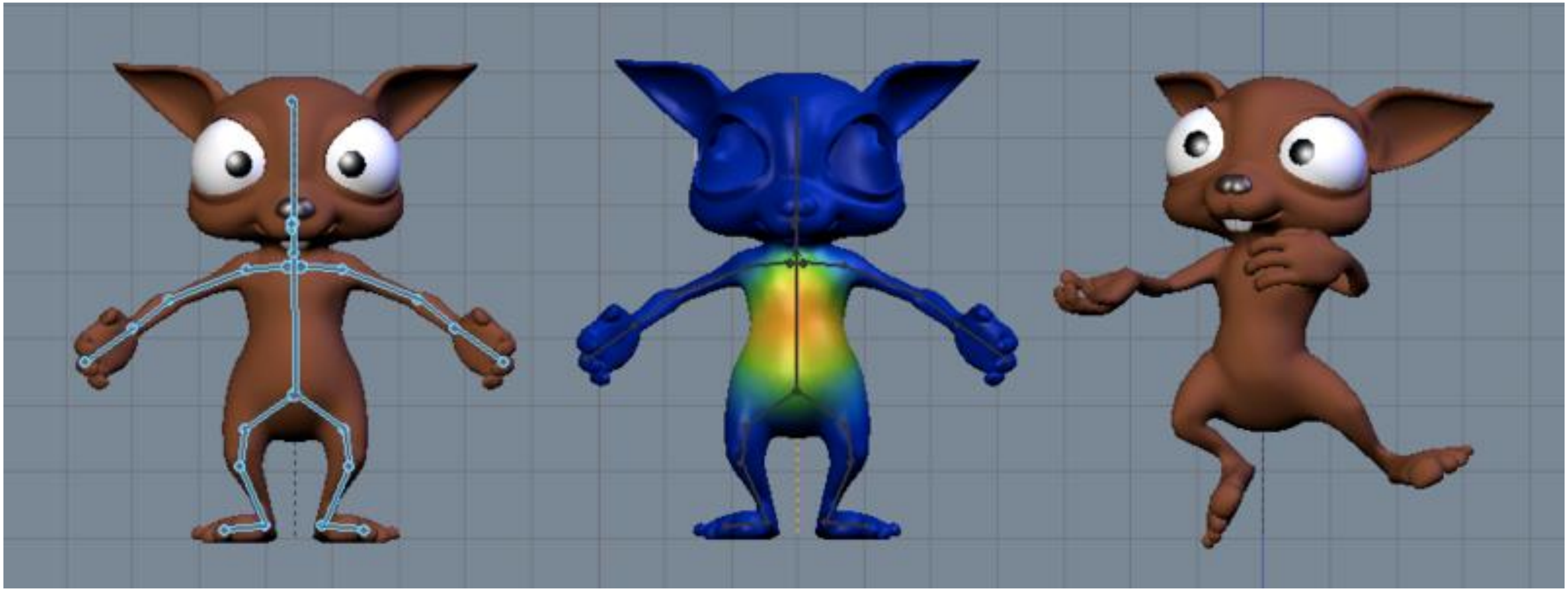
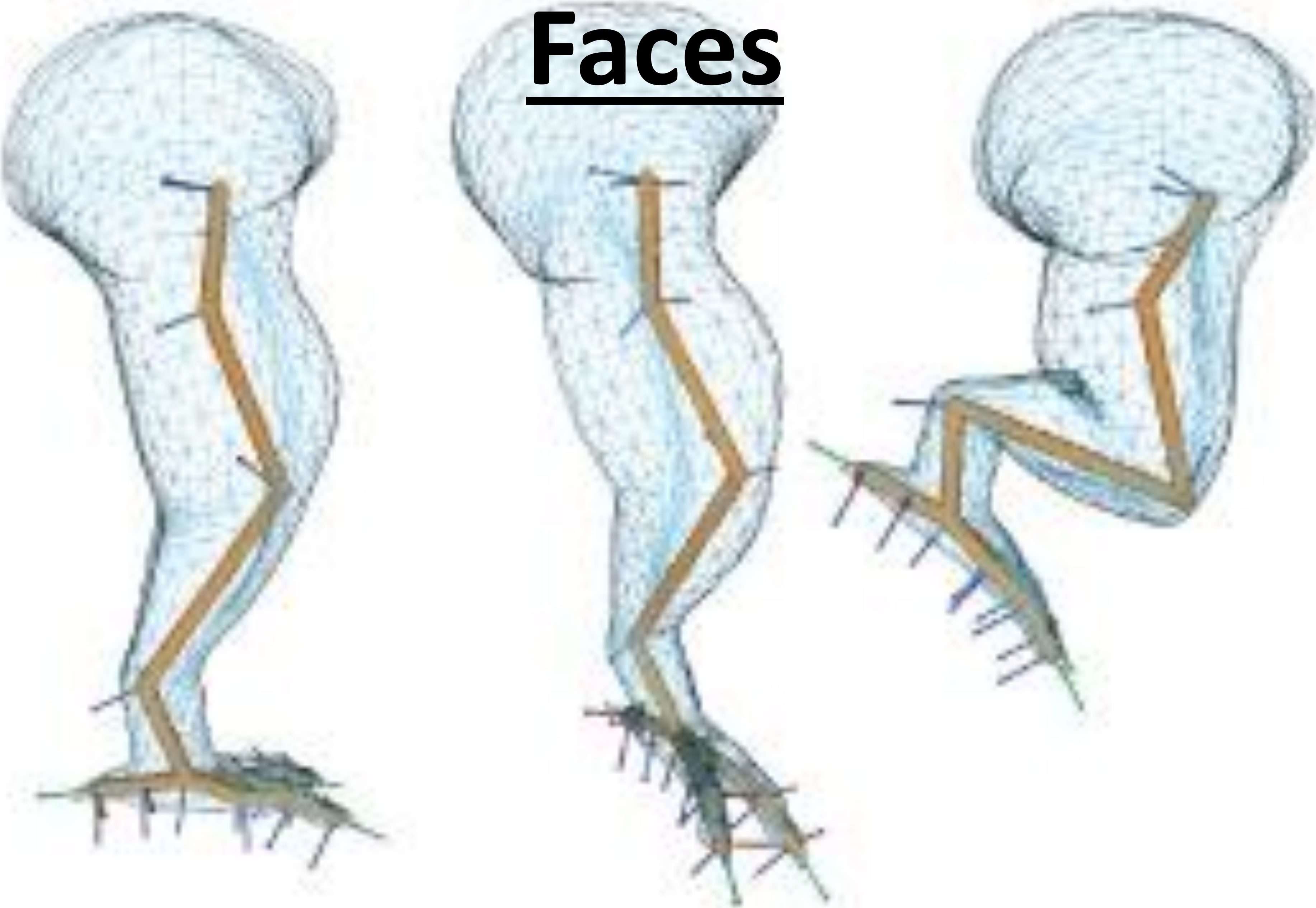
Skinning

Skinning (or Enveloping)

- Envelop the underlying skeleton with a surface representation (triangle mesh, implicit surface), or skin, that conveys the appearance of the character and deforms with the underlying skeleton

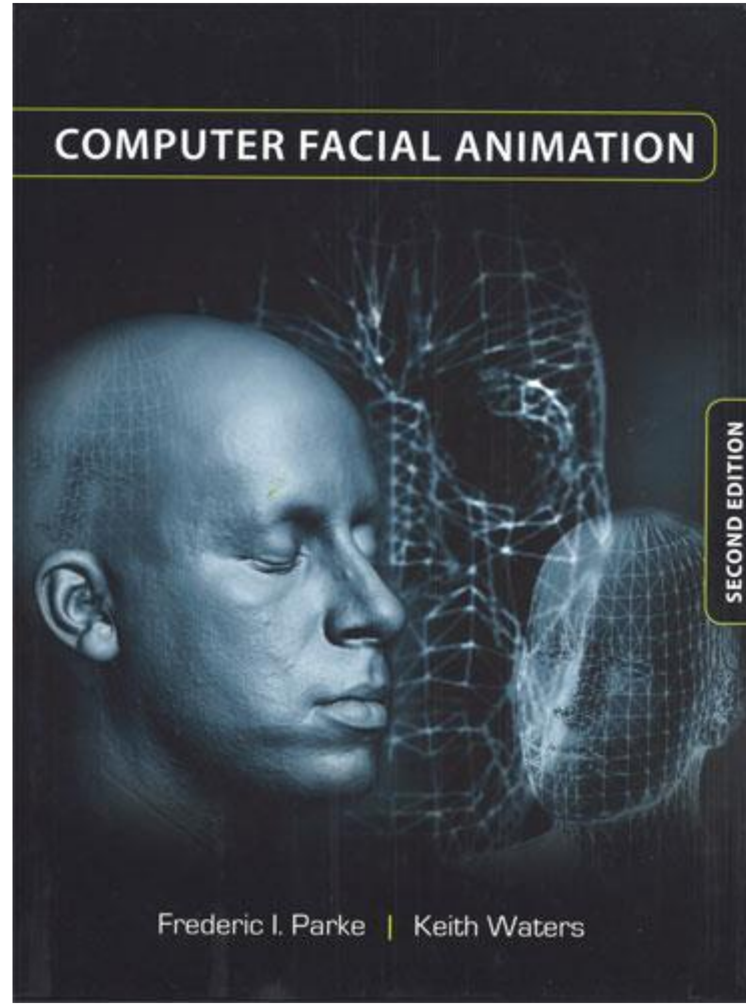


Faces

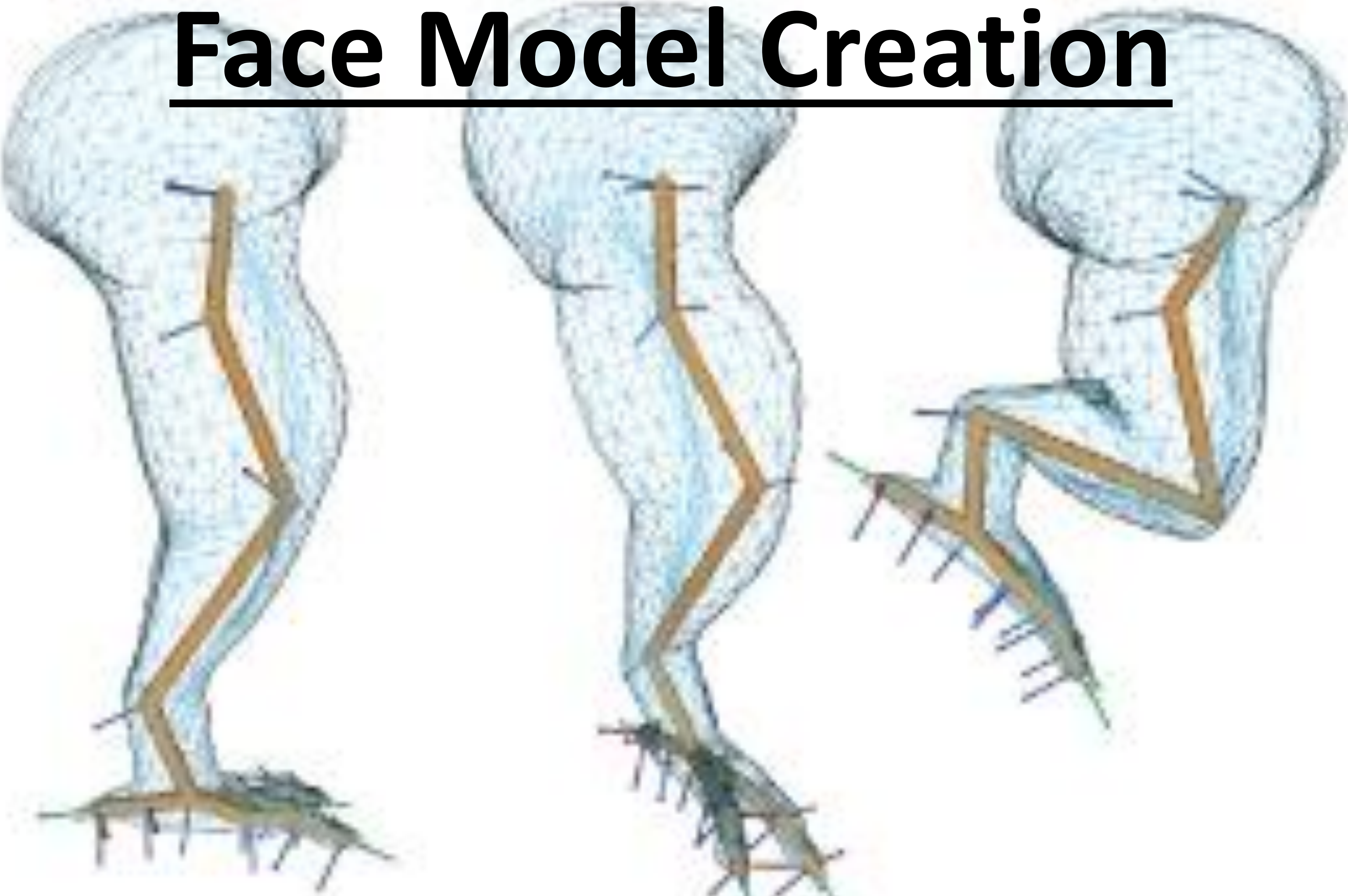


Facial Animation

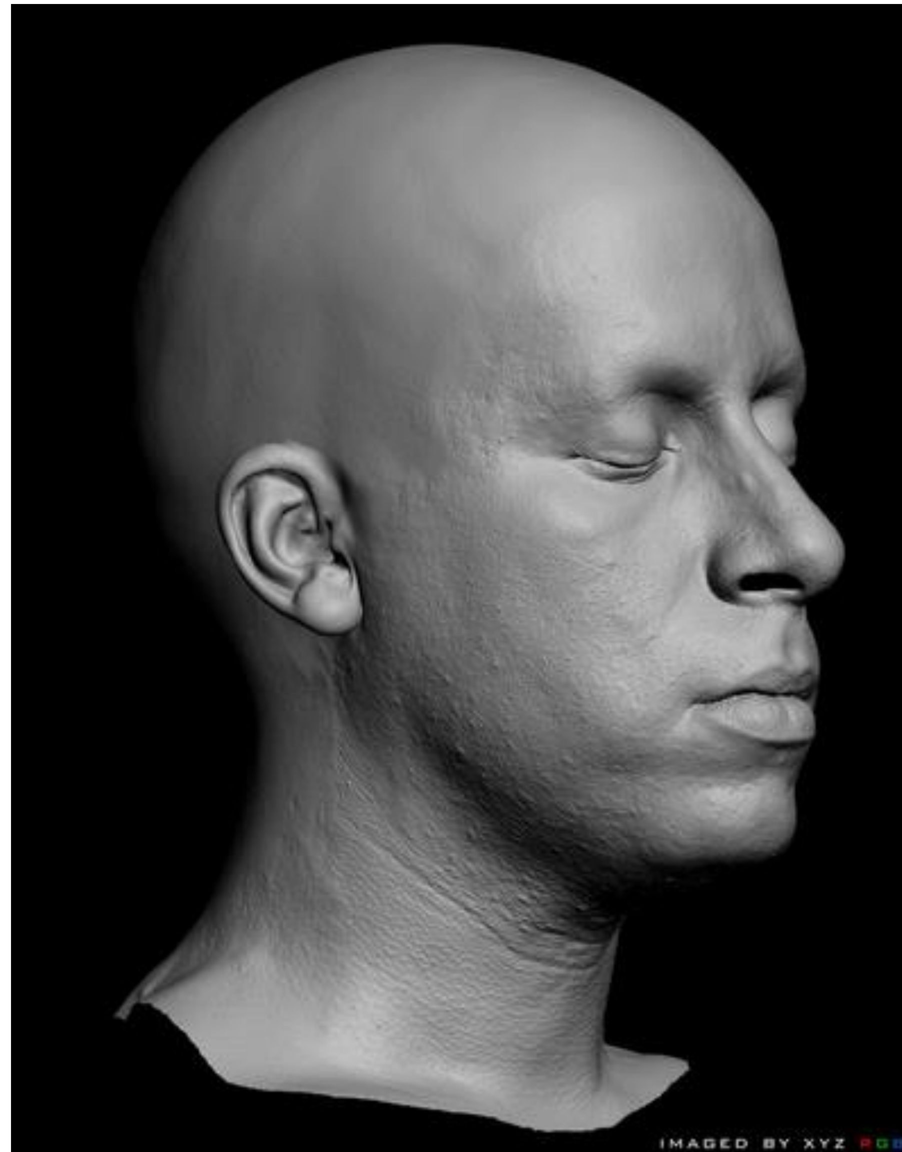
- Create a <u>neutral</u> resting shape for the face
- Then create a number of <u>key poses</u> for different expressions:
  - E.g. smile, frown, pucker, mouth open, jaw open
- Each shape is a deformed version of the skin in a target expression
- <u>Interpolating</u> between key shapes gives animation



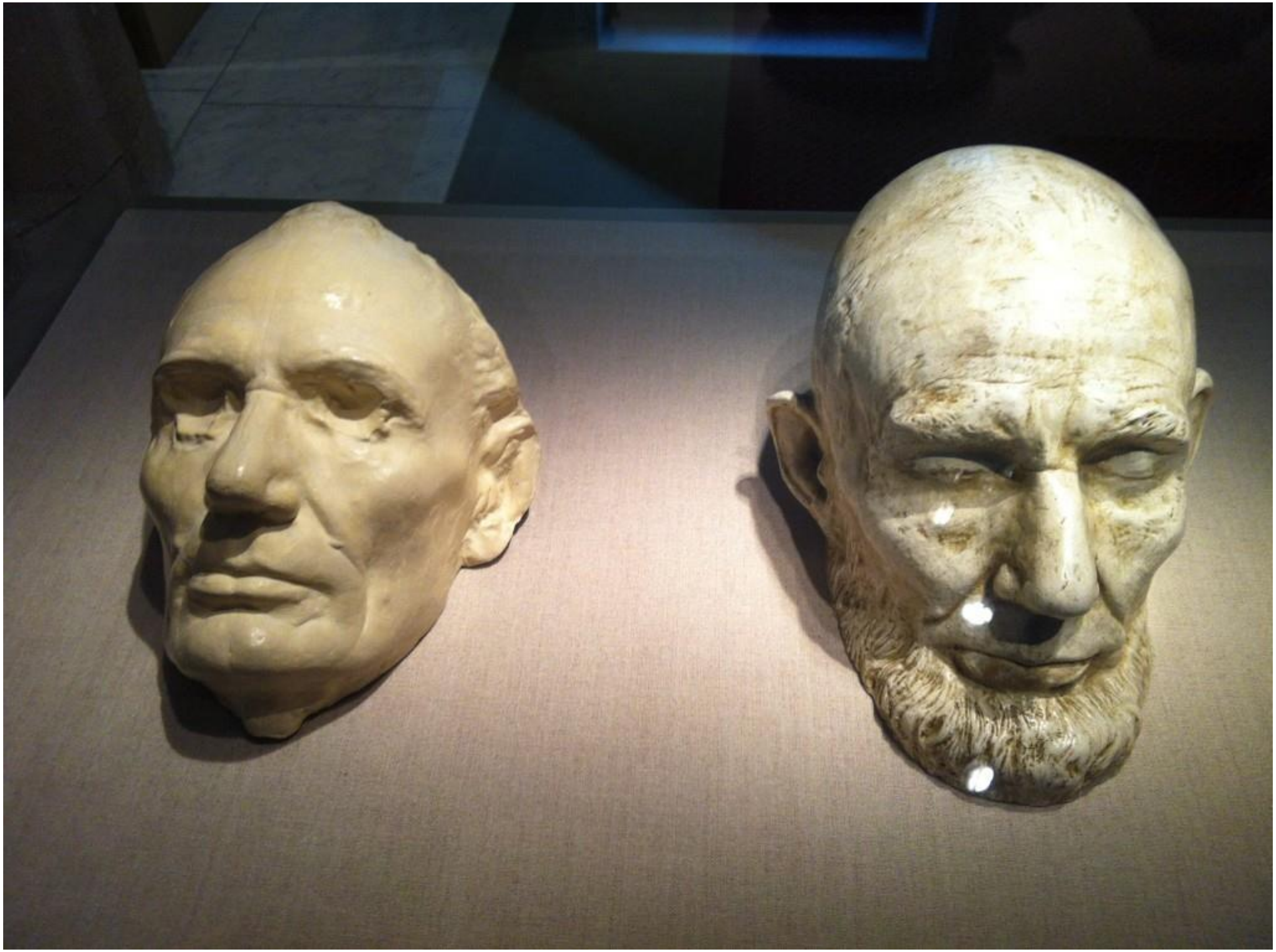
Face Model Creation



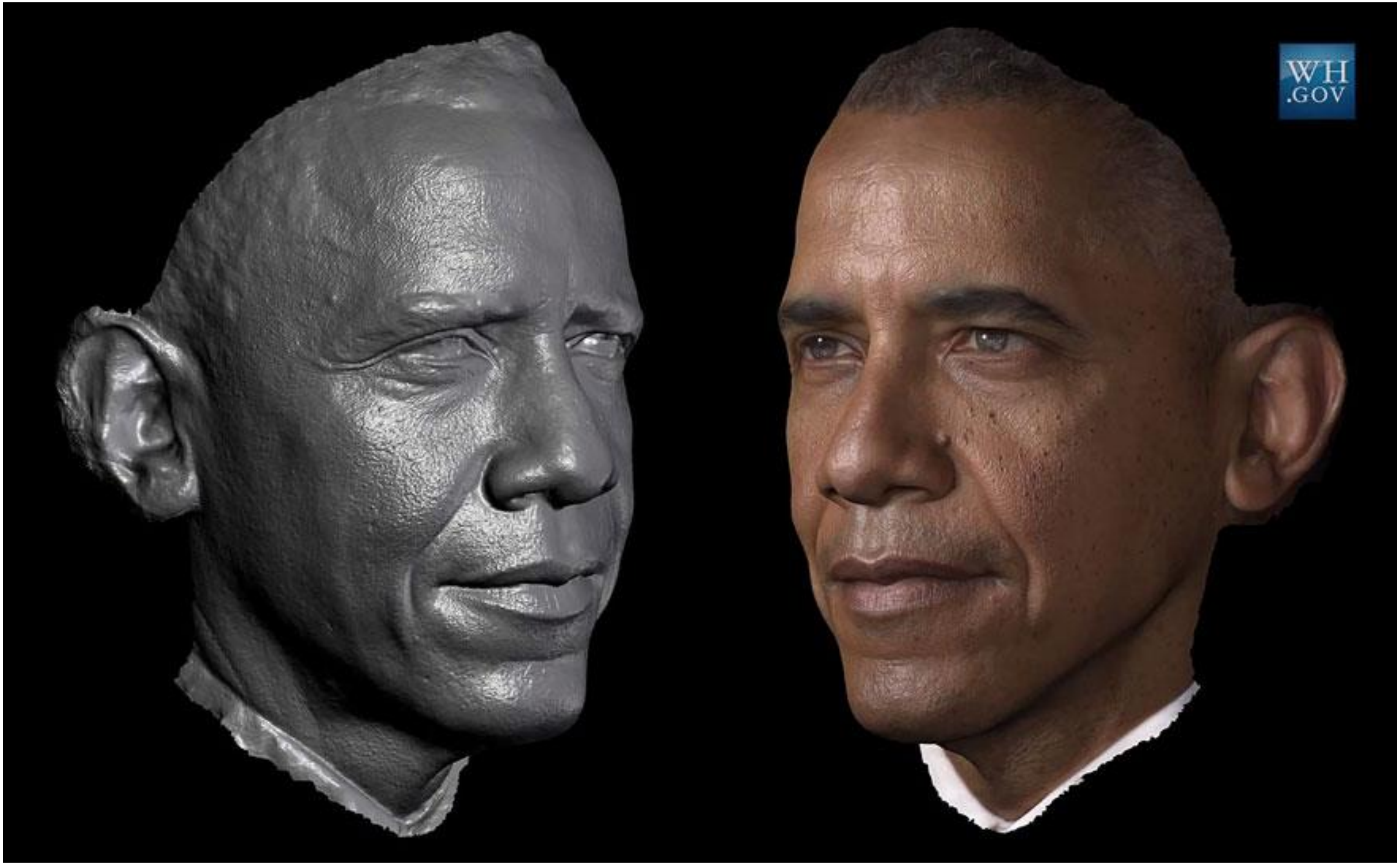
A Stanford Ph.D. student



Abraham Lincoln



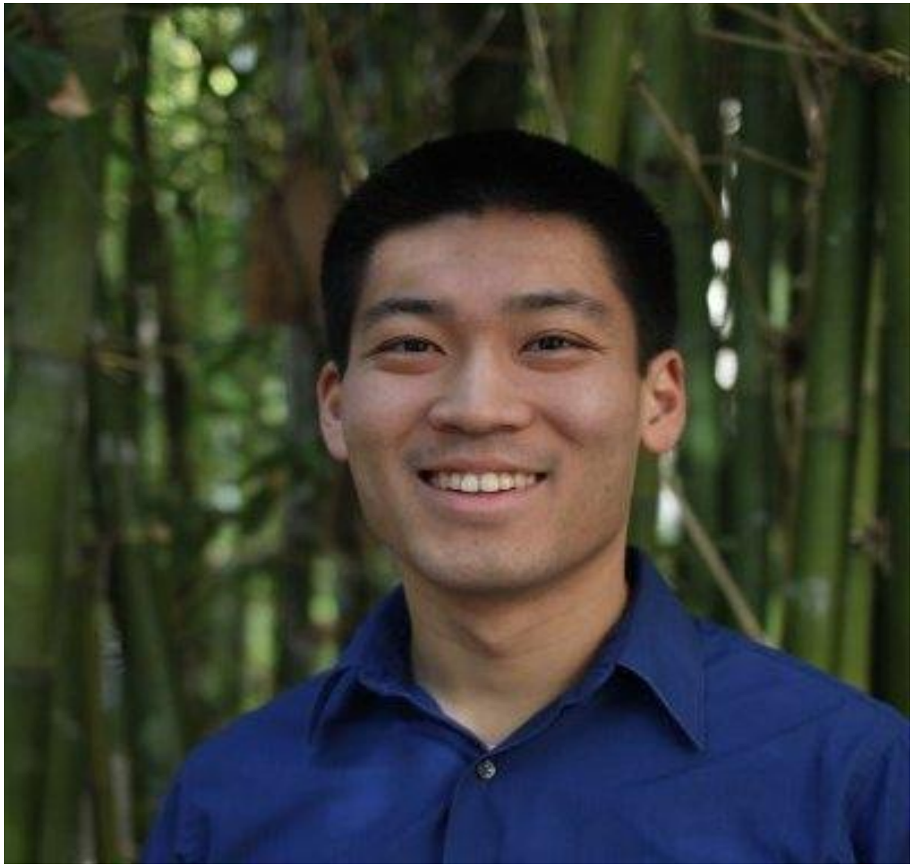
President Obama



https://youtu.be/4GiLAOtjHNo

Yoda

Kong: Skull Island (March 10, 2017)



Matthew Cong

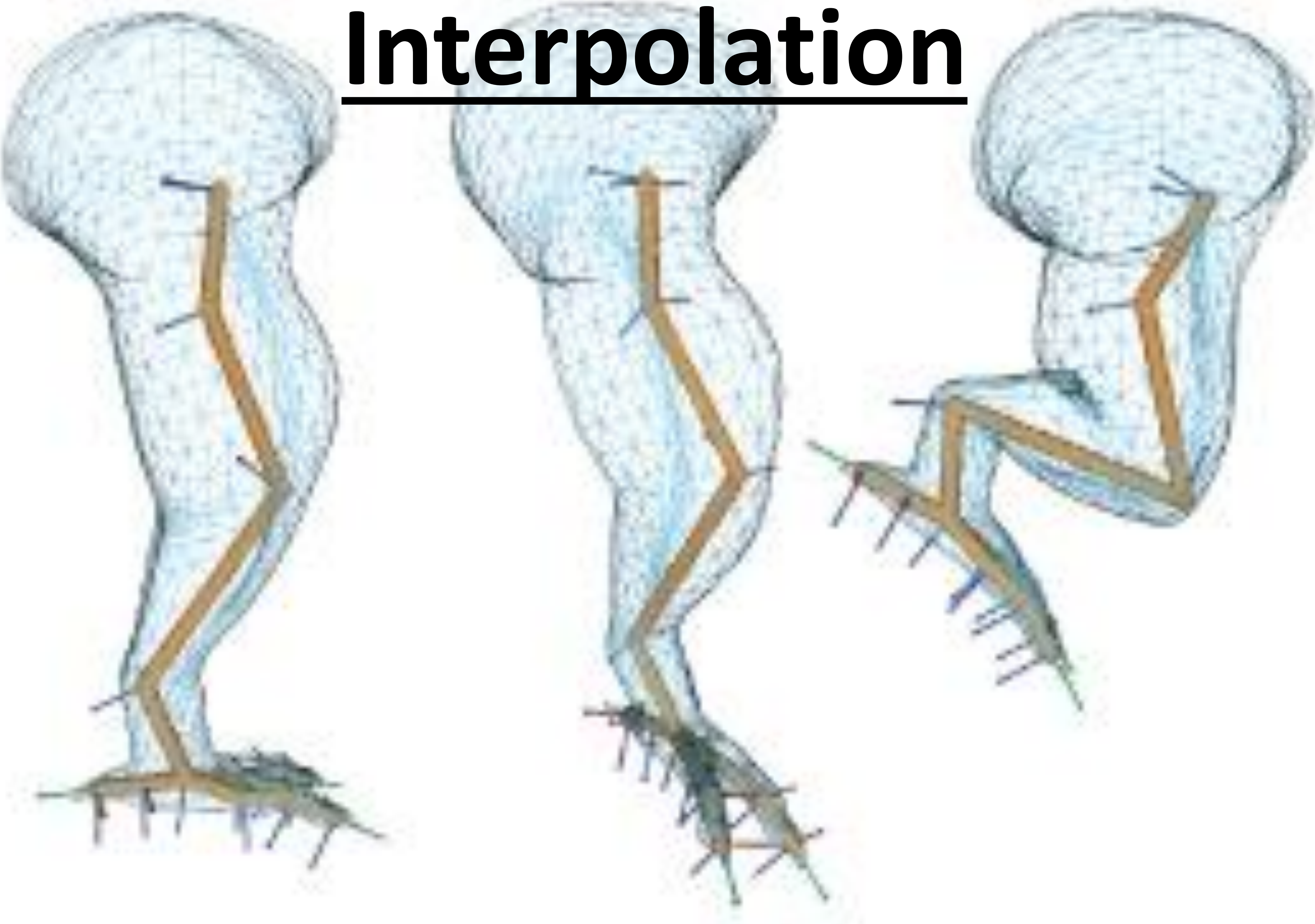
King Kong

https://www.youtube.com/watch?v=2onxgmKT1fw

Expression Shapes

- Besides scanning in the neutral pose, one needs to scan in shapes for every desired expression
- Alternatively, a modeler can deform vertices by hand to create various expression shapes

Interpolation



Degrees of Freedom

- Interpolation between various poses is carried out on a node by node basis
- Thus, the neutral shape and every expression shape is created with the same triangles
  - and with the triangle vertices corresponding in a one to one fashion
- This works/looks better if each vertex corresponds to a fixed position on the skin surface of the character
- If there are m vertices, then the i-th shape is be given by:

$$\begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{mi} \end{bmatrix}$$

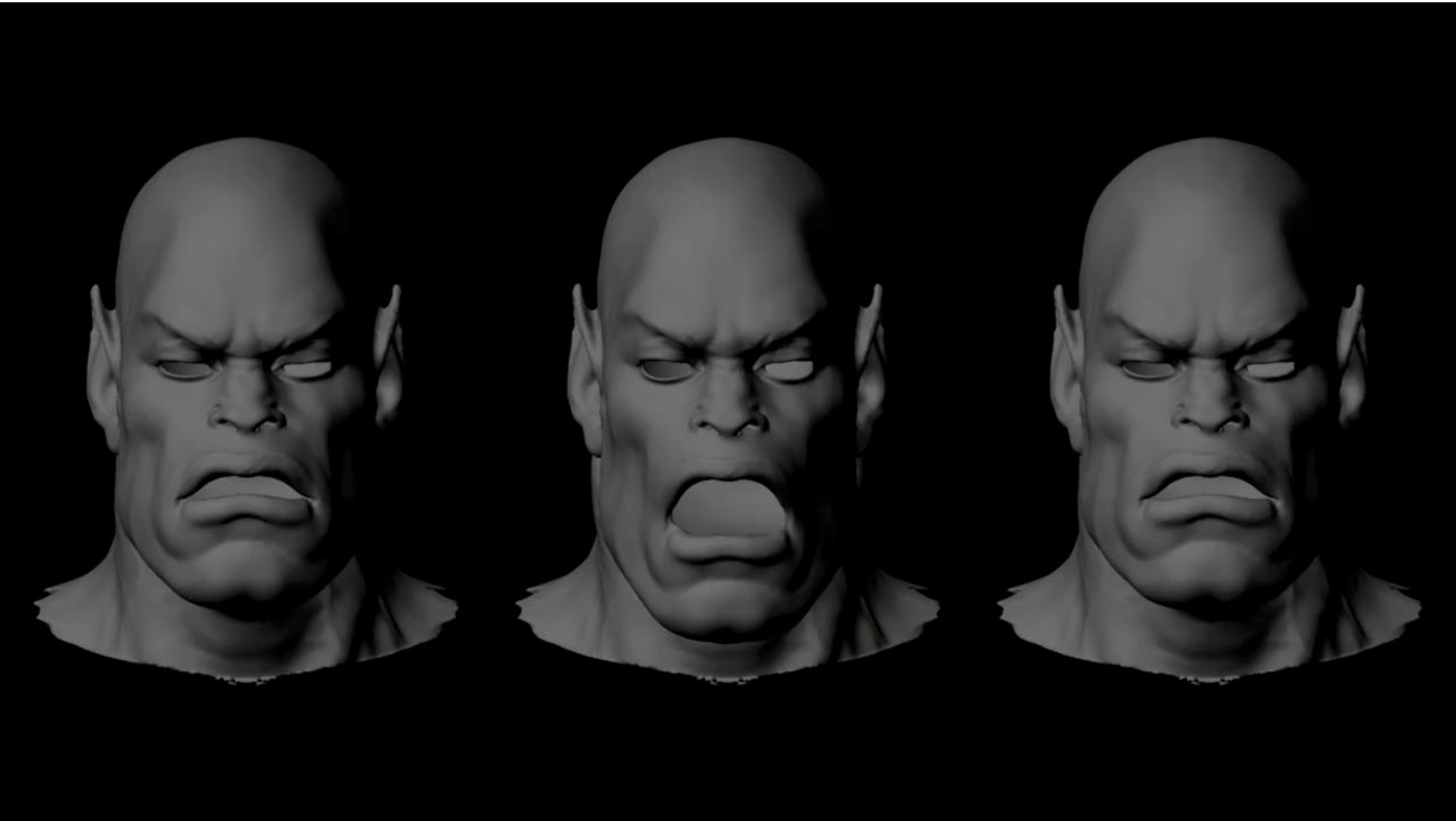
Interpolation

- Obtain a new shape by linearly interpolating between two key shapes

$$.28 \begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{m1} \end{bmatrix} + .72 \begin{bmatrix} x_{12} \\ x_{22} \\ \vdots \\ x_{m2} \end{bmatrix} = \text{resulting shape}$$

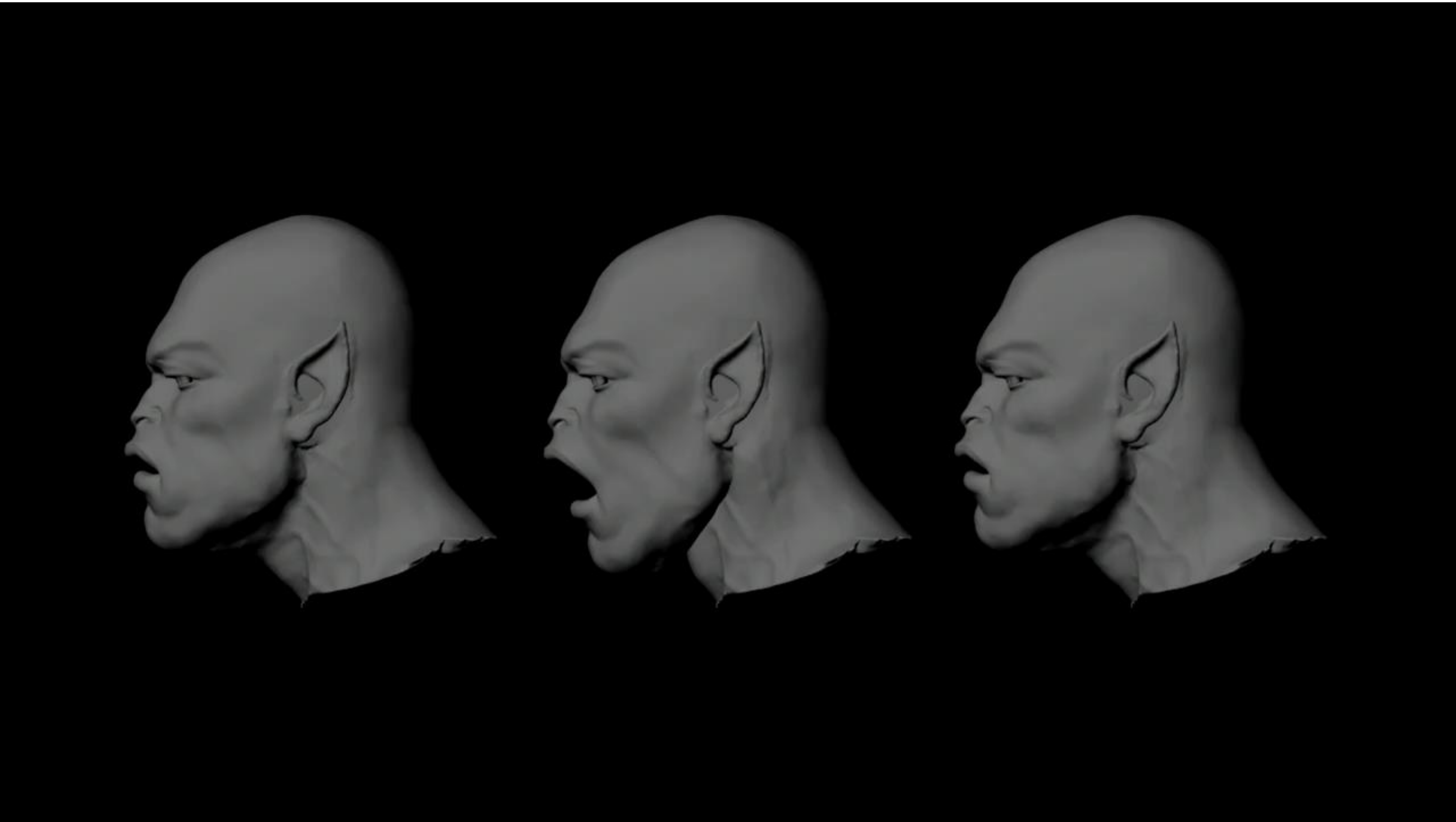
Animation

- Vary the interpolation weights (α,1-α) over time



Animation

- Vary the interpolation weights (α,1-α) over time



Shape Matrix

- Consider the case of n key shapes (with m vertices in each)
- Concatenate the n column vectors to form a shape matrix:

$$\begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{m1} \end{bmatrix}, \begin{bmatrix} x_{12} \\ x_{22} \\ \vdots \\ x_{m2} \end{bmatrix} \cdots \begin{bmatrix} x_{1n} \\ x_{2n} \\ \vdots \\ x_{mn} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}$$

- Note that one of the key shapes needs to be the face in a neutral/rest pose

Interpolation

- A new shape is computed by multiplying the shape matrix with a vector of interpolation weights:

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

- Every vector of interpolation weights $\vec{\alpha}$ gives a new set of vertex positions (i.e., a new shape) $\vec{x}$
- Animate the vector of interpolation weights $\vec{\alpha}$ in order to animate the shape of the face

Displacements

- Alternatively, one could construct a displacement matrix consisting of displacements from the neutral/rest pose

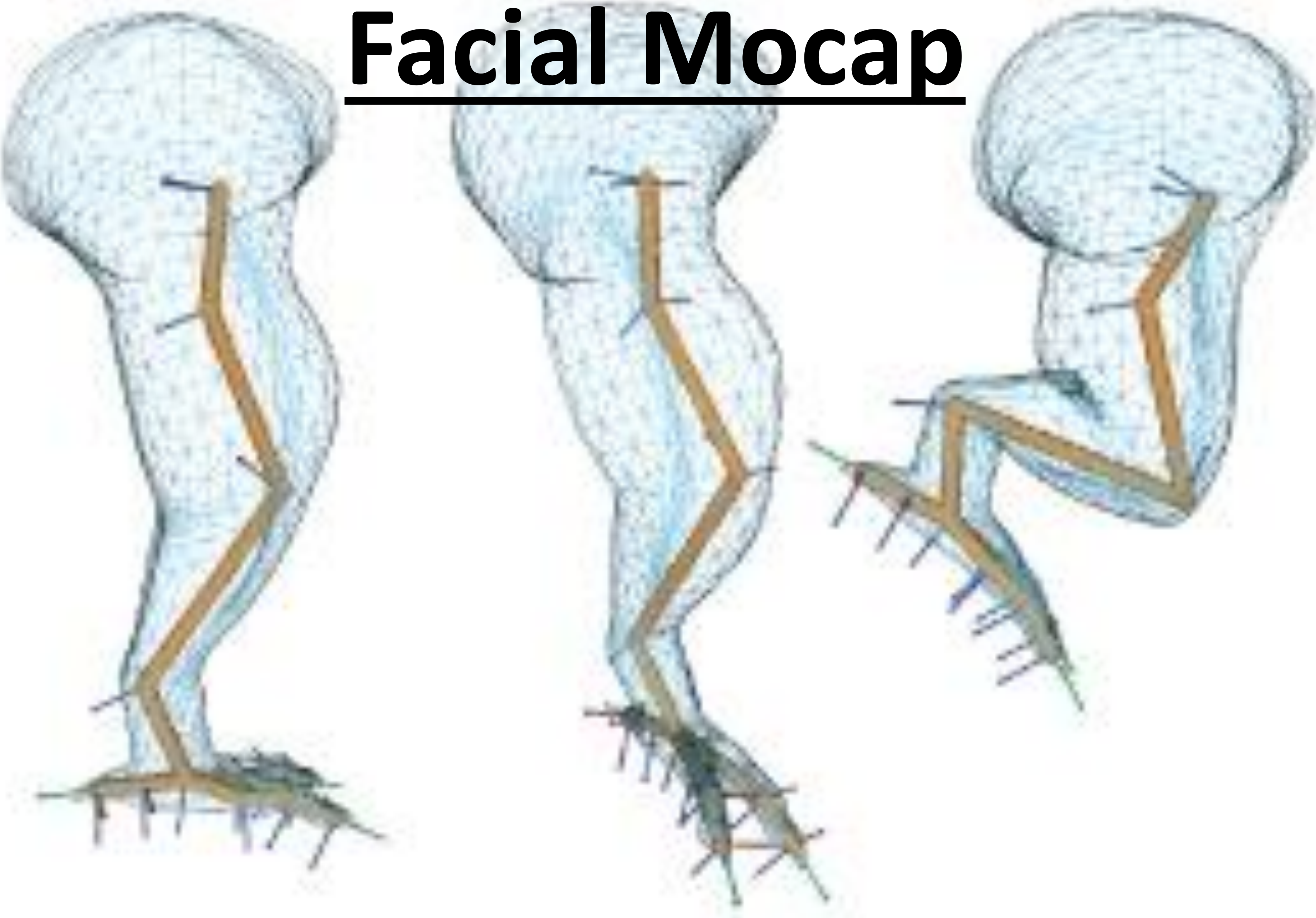
$$\begin{bmatrix} \delta x_{11} & \delta x_{12} & \cdots & \delta x_{1n-1} \\ \delta x_{21} & \delta x_{22} & \cdots & \delta x_{2n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \delta x_{m1} & \delta x_{m2} & \cdots & \delta x_{mn-1} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{n-1} \end{bmatrix} = \begin{bmatrix} \delta x_1 \\ \delta x_2 \\ \vdots \\ \delta x_m \end{bmatrix}$$

- In this case, the neutral shape $\overrightarrow{x_0}$ is not a column in the matrix (it would be a column of all zeroes)
- The result of the matrix multiplication is added to the neutral shape to obtain the new shape:

$$\vec{x} = \overrightarrow{x_0} + \overrightarrow{\delta x}$$

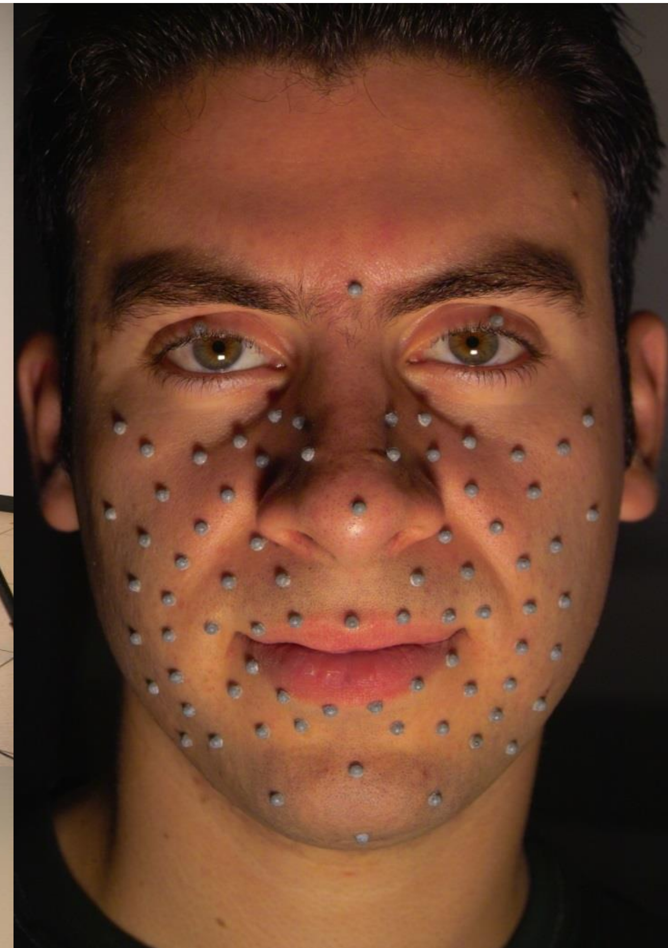
- The two approaches can be shown to be equivalent, if the weights have the property: $\sum_1^n \alpha_i = 1$

Facial Mocap

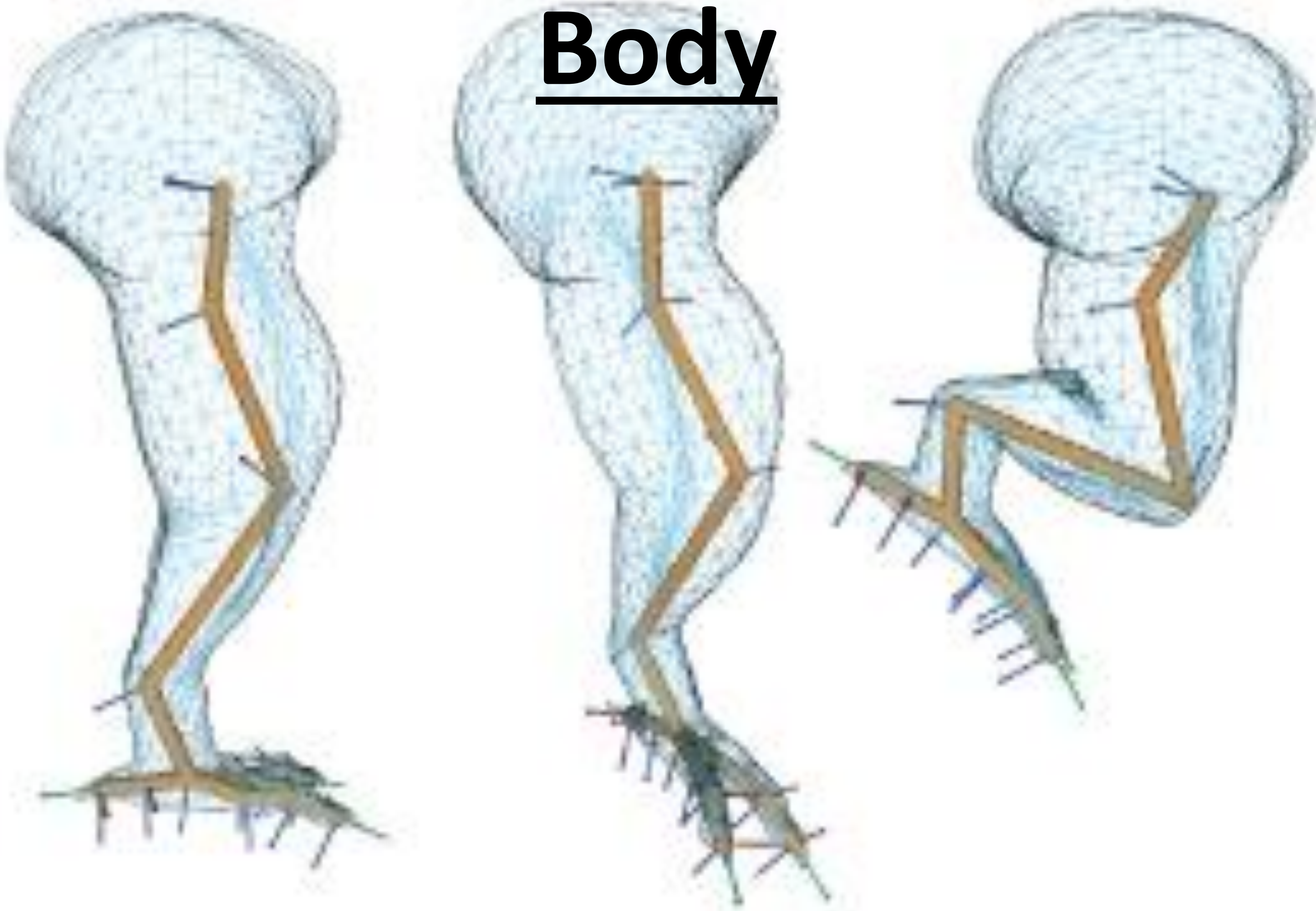


Facial Motion Capture

- Instead of animating $\vec{\alpha}$, one can compute $\vec{\alpha}$ via mocap
- Given a mocap frame, compute $\vec{\alpha}$ such that the resulting shape matches the mocap data as close as possible
  - E.g. add markers to the neutral shape; determine $\vec{\alpha}$ such that the displaced location of those markers agrees with the displaced mocap markers
- Increasing the number of shapes allows for the actor's performance to be more closely matched
  - An insufficient number of shapes can cause details in the actor's performance to be lost
- Finally, $\vec{\alpha}$ can be remapped to another creature
  - as long as the column vectors of the shape/displacement matrices have corresponding meanings from the actor shape matrix to the creature shape matrix



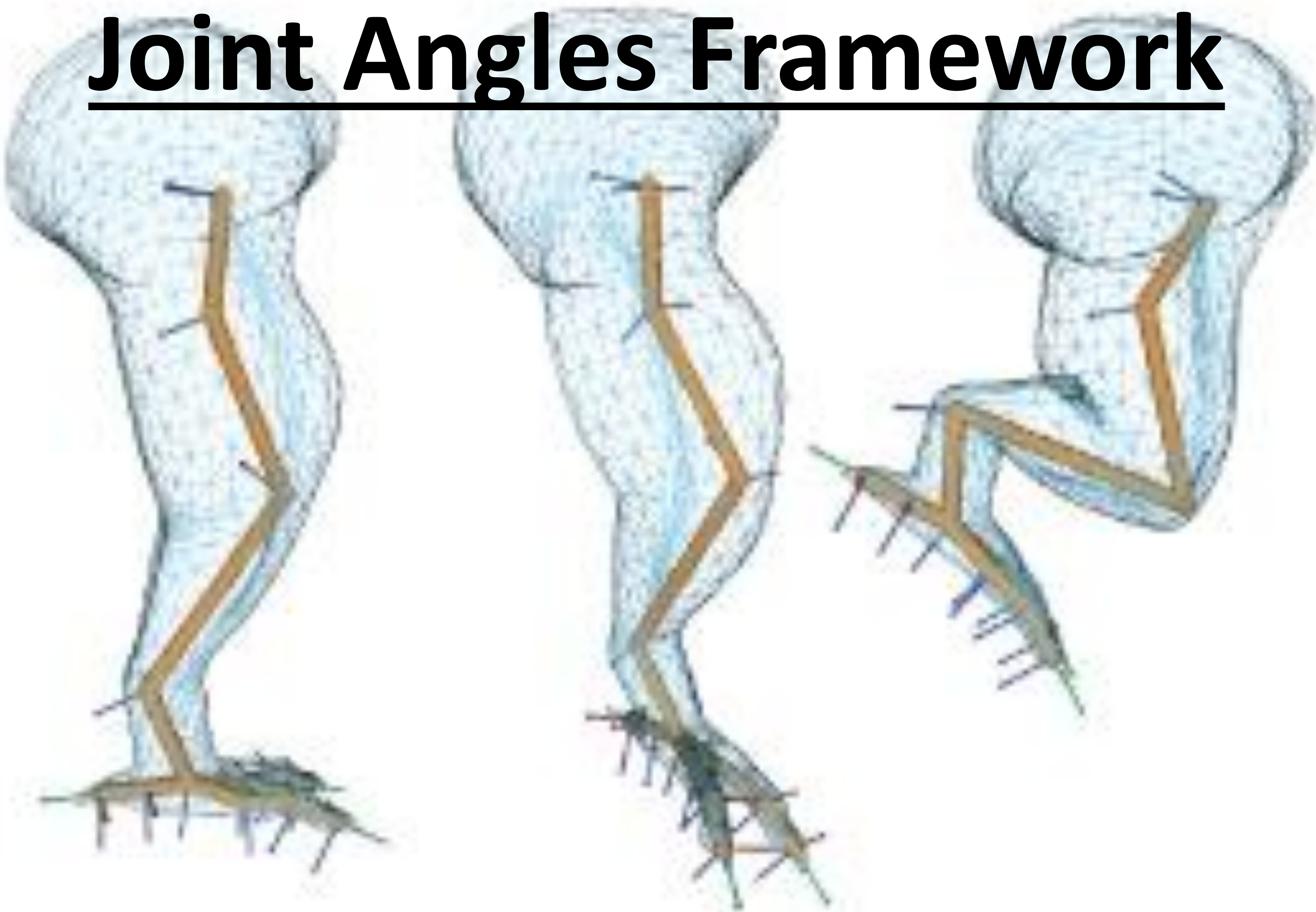
Body



A Different Approach

- A similar process could be carried out for the body
  - i.e. create a shape matrix and interpolate
- But, the shape of the body is highly dependent on the angles of joints, so one can bootstrap the interpolation weights $\vec{\alpha}$ from the joint angles
  - The joint angles do miss some shape information such as whether a muscle is being intentionally flexed
  - Note: the $\vec{\alpha}$ in facial animation can be bootstrapped in a similar fashion using the angle of the jaw joint and contractions of various facial muscles
- Many parts of the body are relatively disjoint from each other, so we expect the <u>displacement</u> matrix to be sparse (but the shape matrix is not sparse)
- Because of these considerations, we approach skinning the body in a slightly different manner
  - While noting that it still highly depends on shapes and interpolation

Joint Angles Framework



Summary

- Decompose the entire skin (for the whole character) into smaller pieces, and place a portion of the skin into the object space of each bone
  - The pieces may overlap, i.e. multiple bones may share the same skin vertices
- Given a set of joint parameters θ
- Let $T_i(\theta)$ represent the transformation that moves bone i from its object space to world space
- As the joint parameters change and the bones move in world space, calculate where the skin vertices are located in world space as well using $T_i(\theta)$
- Skin vertices which exist in the object space of multiple bones require some sort of interpolation or averaging

Rigid Skinning

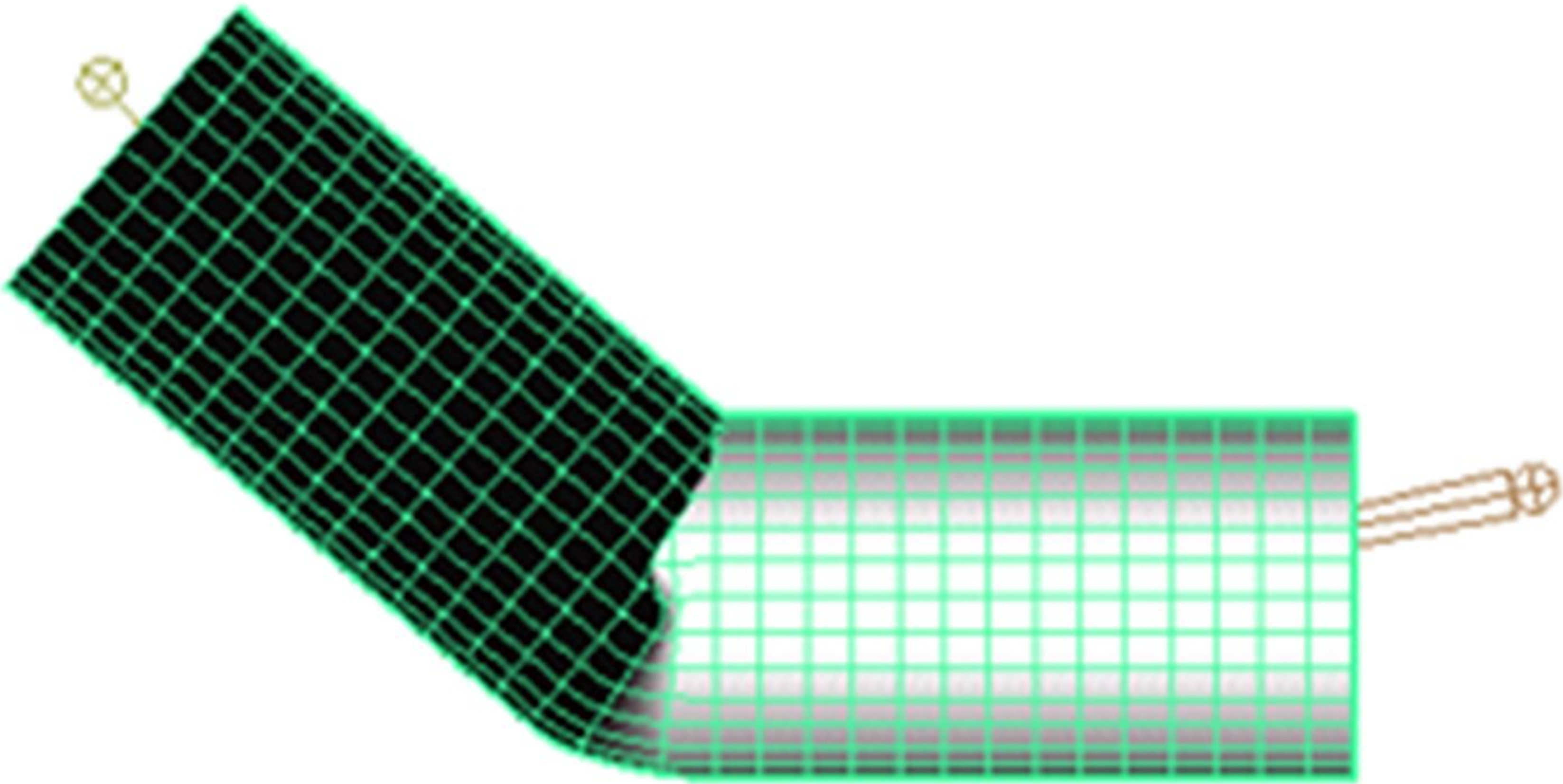
- Each skin vertex is assigned to exactly one bone
  - For example, the skin for the upper arm would be assigned to a different bone than the skin for the forearm
- Use the transform of the associated bone to position each vertex of the skin in world space:
  - Consider a vertex j with position v_j in the object space of the ith bone with transformation T_i
  - Then, the world space position of vertex j is given by

$$v_j' = T_i v_j$$

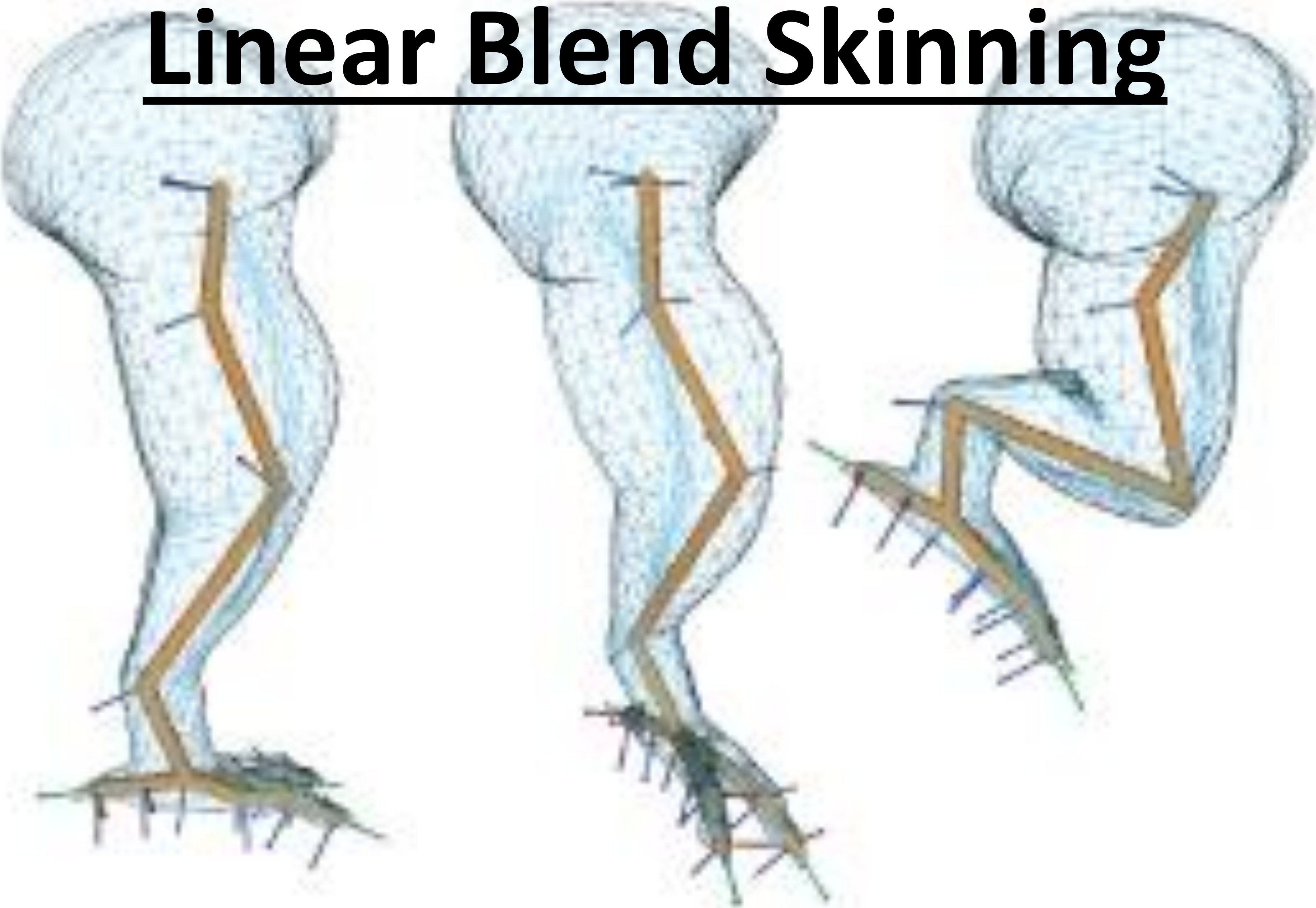
- As the skeleton moves, T_i changes and the vertex positions of the skin change as well

Rigid Skinning

- Unwanted discontinuities form along the boundaries where neighboring skin vertices are assigned to different bones



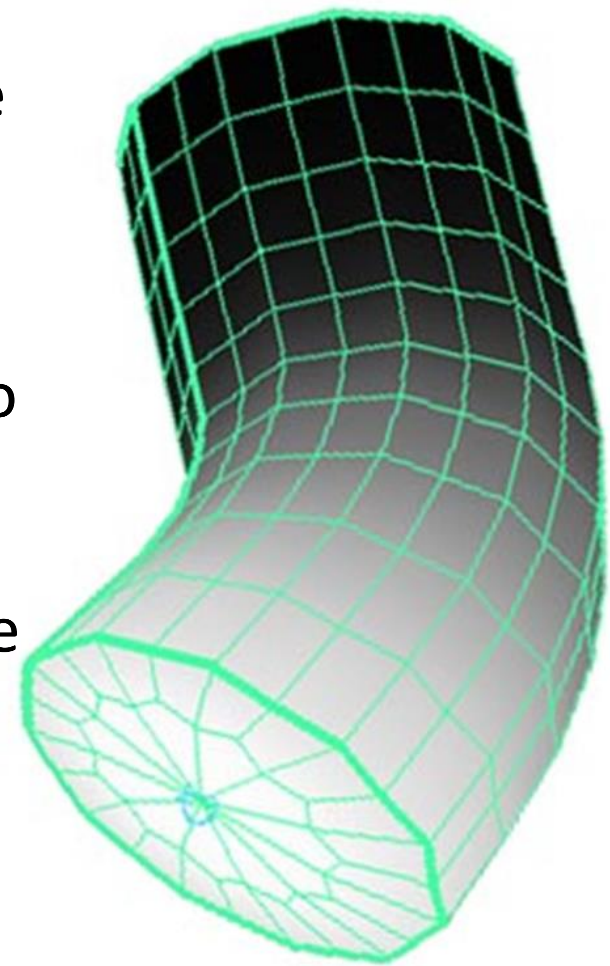
Linear Blend Skinning



Linear Blend Skinning

- Remove the discontinuity by linearly blending vertices near the joint
- Assign each skin vertex to more than one bone
  - Note: v_j^i will have different coordinates in different rigid body object spaces
- Each bone i to which vertex v_j belongs to is assigned a nonzero weight w_{ij}
- The world space position of the vertex is computed as the weighted average of the world space positions obtained from each bone via rigid skinning:

$$v_j' = \sum_i w_{ij} T_i v_j^i$$



Normals & Tangents

- Normal and tangent vectors of the surface mesh (important for rendering/collisions) are blended as well:

$$n'_j = \sum_i w_{ij} \, T_i^{-T} n_j^i$$

$$t'_j = \sum_i w_{ij} \, T_i t_j^i$$

- Normalize n'_j and t'_j if unit length is required

Weights

- Weights for a vertex should be sparse
  - E.g., if the angle of the elbow joint is changed, the skin for the leg shouldn't deform
  - Nonzero weights should be localized to nearby bones
- Sparse weights allow for fast evaluation
  - Typically at most four non-zero weights per vertex (at most four bones can deform a vertex)
- Weights should be smooth to avoid discontinuities
  - Often chosen with a smooth falloff based on distance to a particular bone
- Weights should be independent of mesh resolution
  - So that subdividing the mesh doesn't require recomputing weights
- Constrain the weights to be convex (i.e. $\sum_i w_{ij} = 1, w_{ij} \geq 0$) to avoid undesired scaling and extrapolation artifacts

Specifying Weights

- Manual Approach:
  - Hand-tune weights in order to obtain the best look
  - Intractable to individually modify the weights for each vertex in a large mesh
  - Various painting tools facilitate weight specification

- Automatic Approach:
  - Use an algorithm to calculate weights for each vertex and all its associated bones
    - E.g., based on a “distance” metric from vertices to bones
  - Automatically generated weights are often additionally modified by an artist for higher visual fidelity

Specifying Weights: Pinocchio

- System for automatically rigging and animating 3D characters
- Solves a Poisson equation (PDE!) for each bone with appropriate boundary conditions to obtain smoothly varying weights
- Can be used to rig and skin your own characters
- Available from MIT:

  http://www.mit.edu/~ibaran/autorig/pinocchio.html

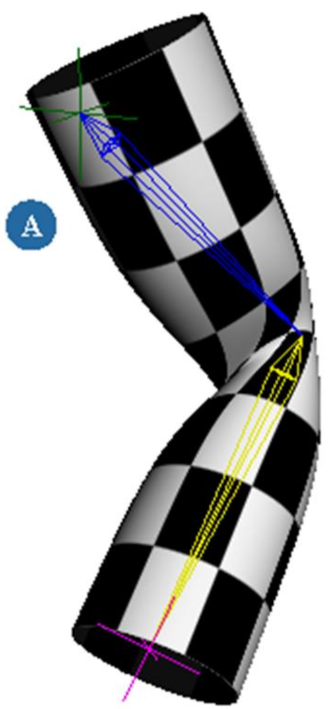
Specifying Weights: Geodesic Voxel Binding

- Automatic approach for specifying weights:

1. Voxelize the interior and boundary of the skin mesh for a rest pose
2. For each bone (left leg bone shown below), compute the geodesic distance from that bone to the center of each voxel using the Fast Marching Method (see rigid body lecture)
3. For each skin vertex, interpolate distance from the surrounding voxels
4. Use this distance in a falloff function to determine the weight for each vertex

Artifacts…

- Linear blend skinning has issues when the joint angles are large or when a bone undergoes a twisting motion
  - “bow tie” or “candy wrapper” effect
  - mesh loses volume
- Linearly blending the matrix representations of rigid body transformations does not (in general) result in a matrix that represents a rigid body transformation



Dual Quaternion Skinning

Dual Numbers

- A dual number has the form $\hat{a} = a_0 + \epsilon a_\epsilon$
  - where a_0 and a_ϵ are real numbers
  - and ϵ satisfies $\epsilon^2 = 0$
- Many arithmetic operations are defined for dual numbers
  - such as multiplication, division, conjugation, and the square root
- Dual numbers can be represented as 2x2 matrices:

$$\epsilon = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \text{ and } aI + b\epsilon = \begin{pmatrix} a & b \\ 0 & a \end{pmatrix}$$

- The sum and product of dual numbers can then be calculated with matrix addition and multiplication

Dual Quaternions

- A dual quaternion has the form $\hat{q} = q_0 + \epsilon q_\epsilon$
  - where q_0 and q_ϵ are standard quaternions
- If $q_\epsilon = 0$, the dual quaternion reduces to a standard quaternion q_0 (and represents a rotation)
- Dual quaternions can be used to represent a translation $\vec{t} = (t_x, t_y, t_z)$ as

$$\hat{t} = 1 + \frac{\epsilon}{2}\left(t_x \mathrm{i} + t_y \mathrm{j} + t_z \mathrm{k}\right)$$

- A rigid body transformation with rotation q_0 and translation $\hat{t}$, is represented by the dual quaternion

$$\hat{t} q_0 = \left(1 + \frac{\epsilon}{2}\left(t_x \mathrm{i} + t_y \mathrm{j} + t_z \mathrm{k}\right)\right) q_0$$

Skin Space

- Consider a skin space, where the full character skin is placed in its rest pose
- Each bone is placed into skin space and aligned with the character by a rigid transformation B_i
- A vertex v_j in skin space can be placed into the object space of a rigid body via

$$v_j^i = B_i^{-1} v_j$$

- Thus, the full formula for linear blend skinning is

$$v_j' = \sum_i w_{ij} T_i B_i^{-1} v_j$$

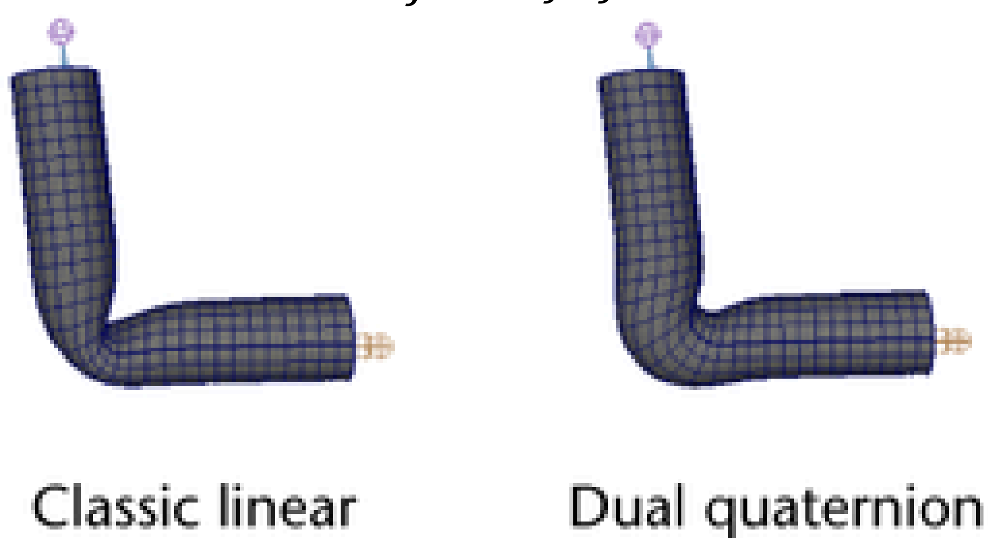
- The fact that $\sum_i w_{ij} T_i B_i^{-1}$ is not a rigid body transform leads to some of the issues with linear blend skinning

Dual Quaternion Skinning

- Convert each composite transformation matrix $T_i B_i^{-1}$ into a unit dual quaternion $\hat{q}_i$
- Then compute a normalized linearly blended dual quaternion $\hat{q}_j$ using the weights

$$\hat{q}_j = \frac{\sum_i w_{ij}\, \hat{q}_i}{\left\|\sum_i w_{ij}\, \hat{q}_i\right\|}$$

- This blended unit dual quaternion is guaranteed to represent a rigid body transformation
- Transform $\hat{q}_j$ back into a transformation matrix $\hat{T}_j$ and calculate the deformed skin vertex position as $v_j' = \hat{T}_j v_j$

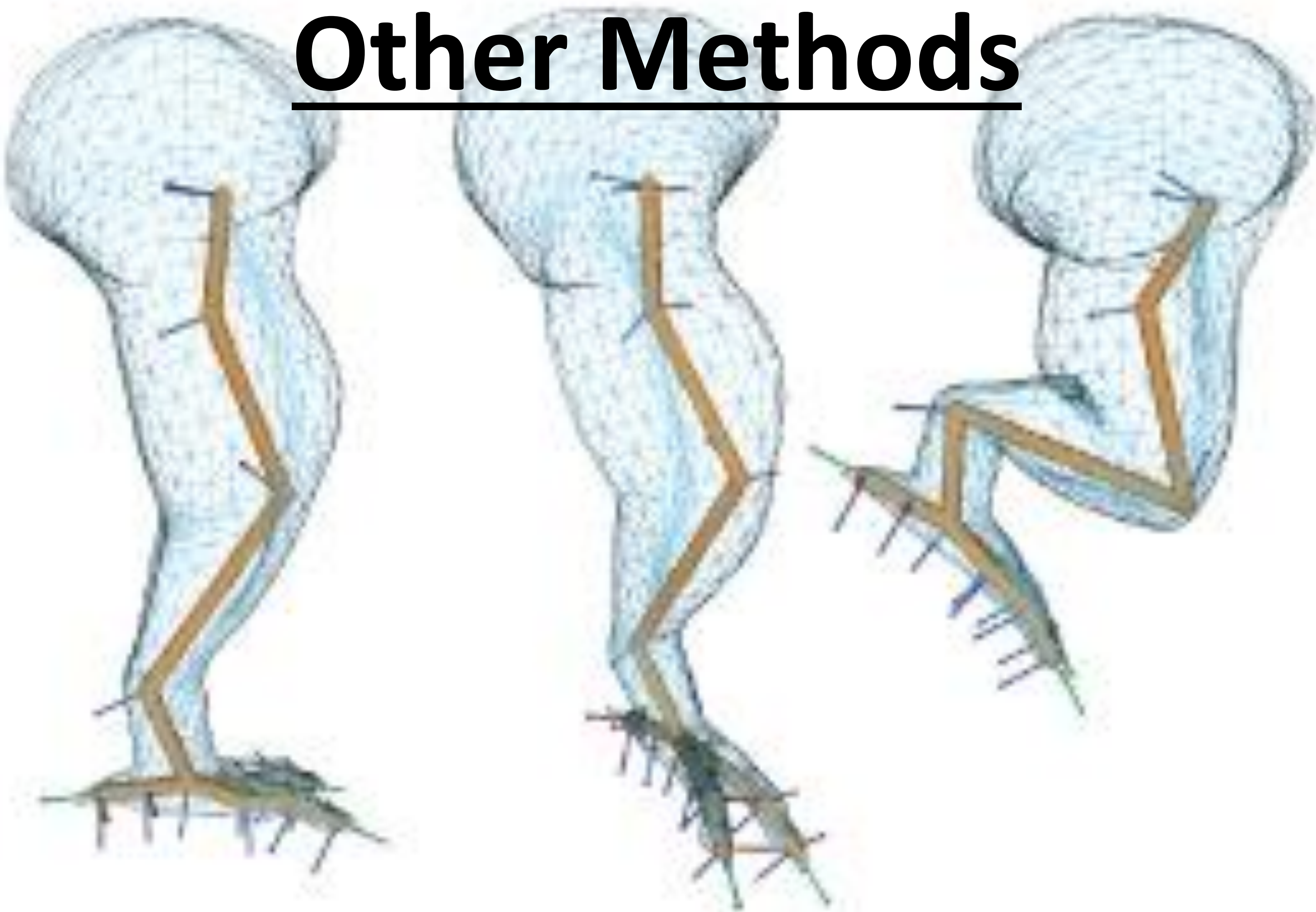


Question #1

LONG FORM:
- Summarize both face skinning and body skinning.
- Answer the short form questions.

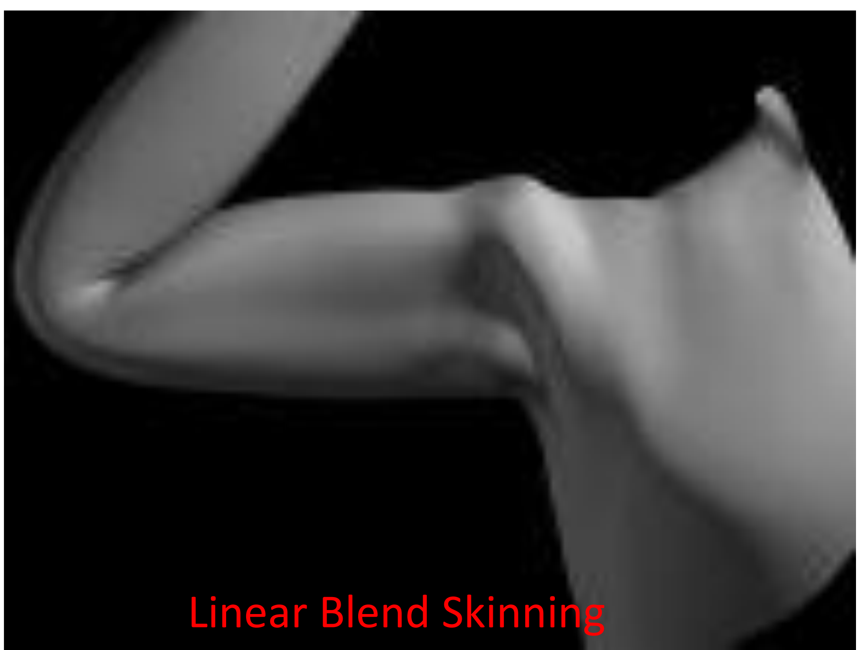
SHORT FORM:
- Pitch your game:
  - Start with a one sentence summary.
  - Why is it cool?
  - What makes it fun to play?
  - What makes it interesting technically?

Other Methods



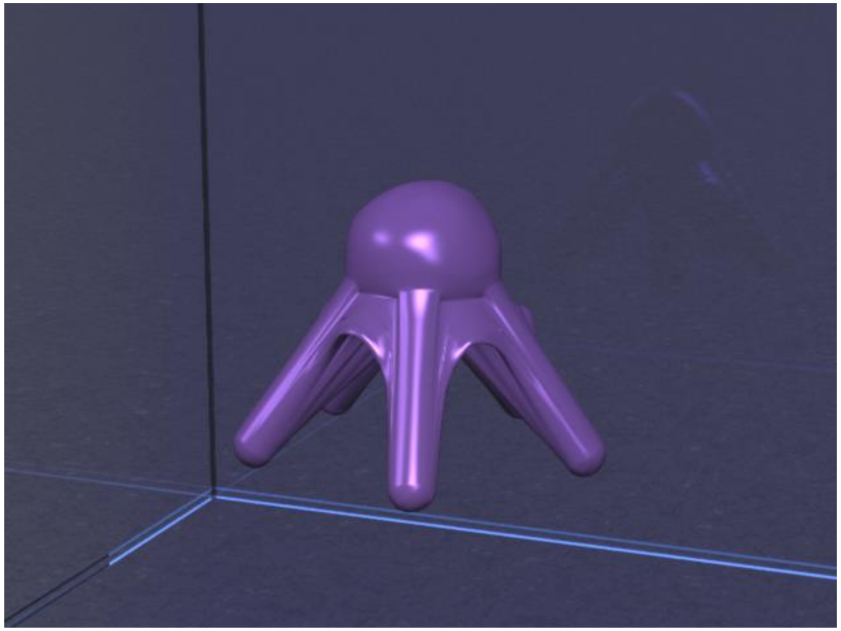
Pose Space Deformers

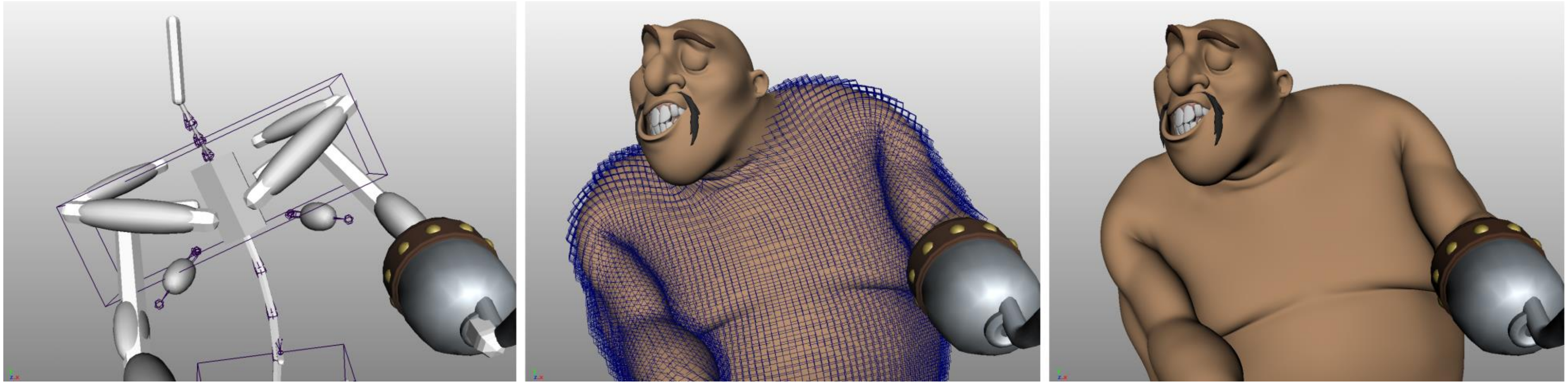
- Pose space deformation considers the entire skeleton including the joint parameters instead of only the locations of the bones in space
- Sculpt a deformed version of the skin for a number of different poses
  - More similar to faces in this sense…
- Perform non-uniform interpolation between sculpted poses to generate a new deformed skin for a non-sculpted pose
- Artist can tune the influence of each sculpted pose to nearby poses



Physics Based Skinning

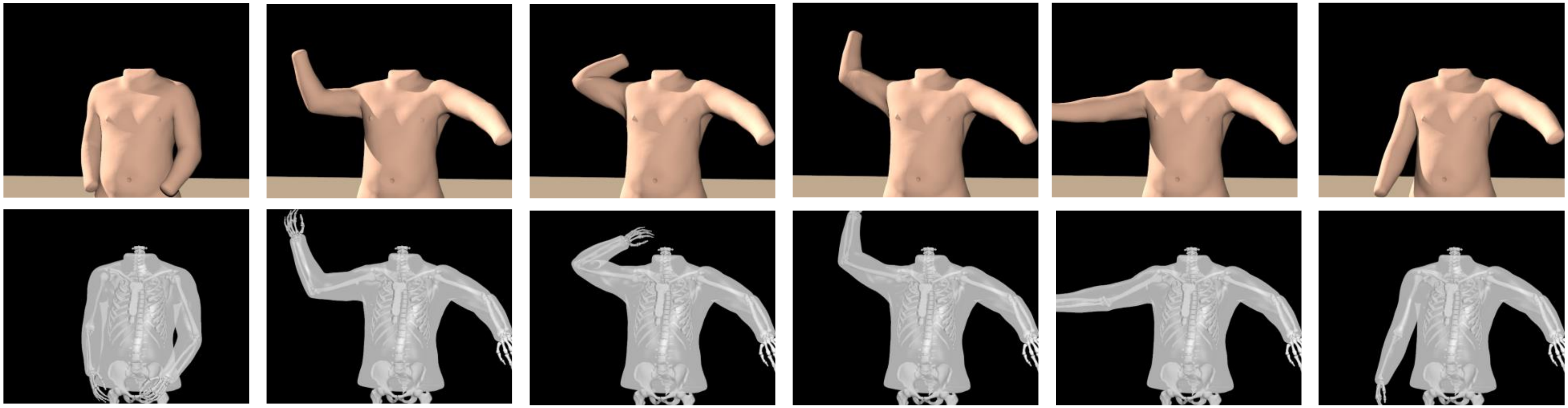
- Embed the skeleton into a volume (e.g. tetrahedral mesh) which can be simulated as a soft body flesh driven by the animated skeleton



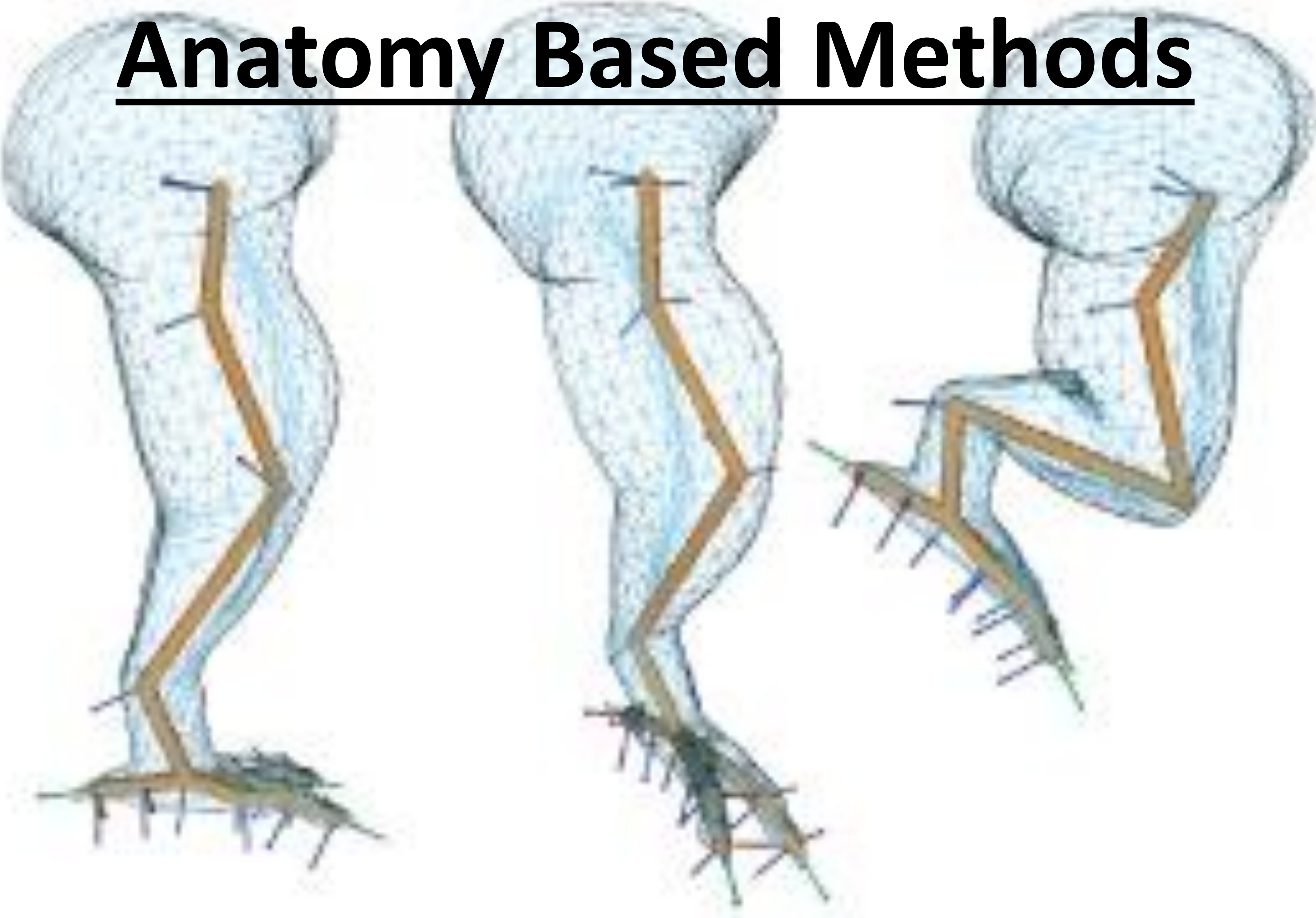


Quasistatics

- Each bone is treated as a kinematic rigid body
- A tetrahedralized volume is used for the flesh
  - mass/spring or finite elements
- The nodes inside/near the rigid body bones are constrained to move with them
- Simulation loop:
  - Move the bones of the skeleton to the desired configuration
  - Assume zero velocities and accelerations
  - Solve for the vertex positions of the surrounding tetrahedral flesh mesh such that it achieves force equilibrium
  - Resulting surface of the tetrahedral flesh mesh is the skin

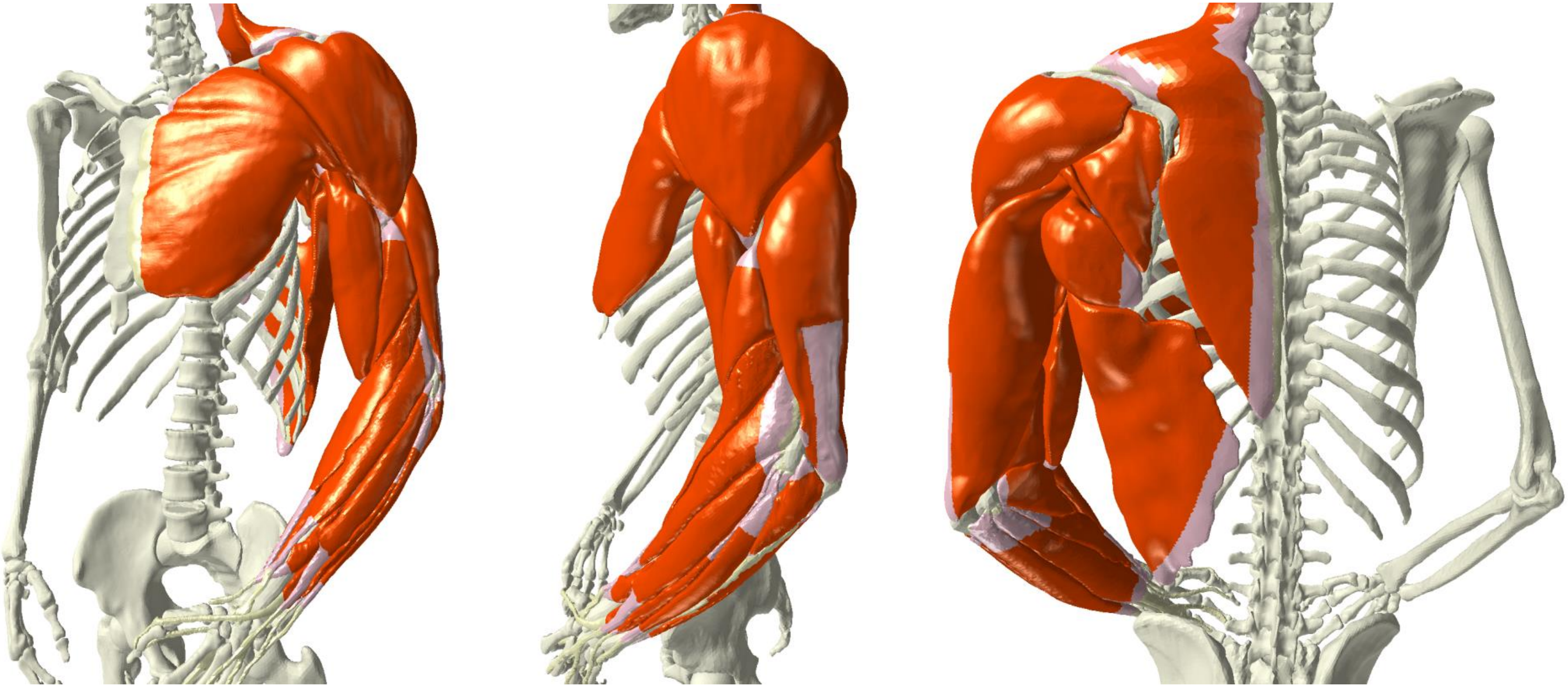


Anatomy Based Methods



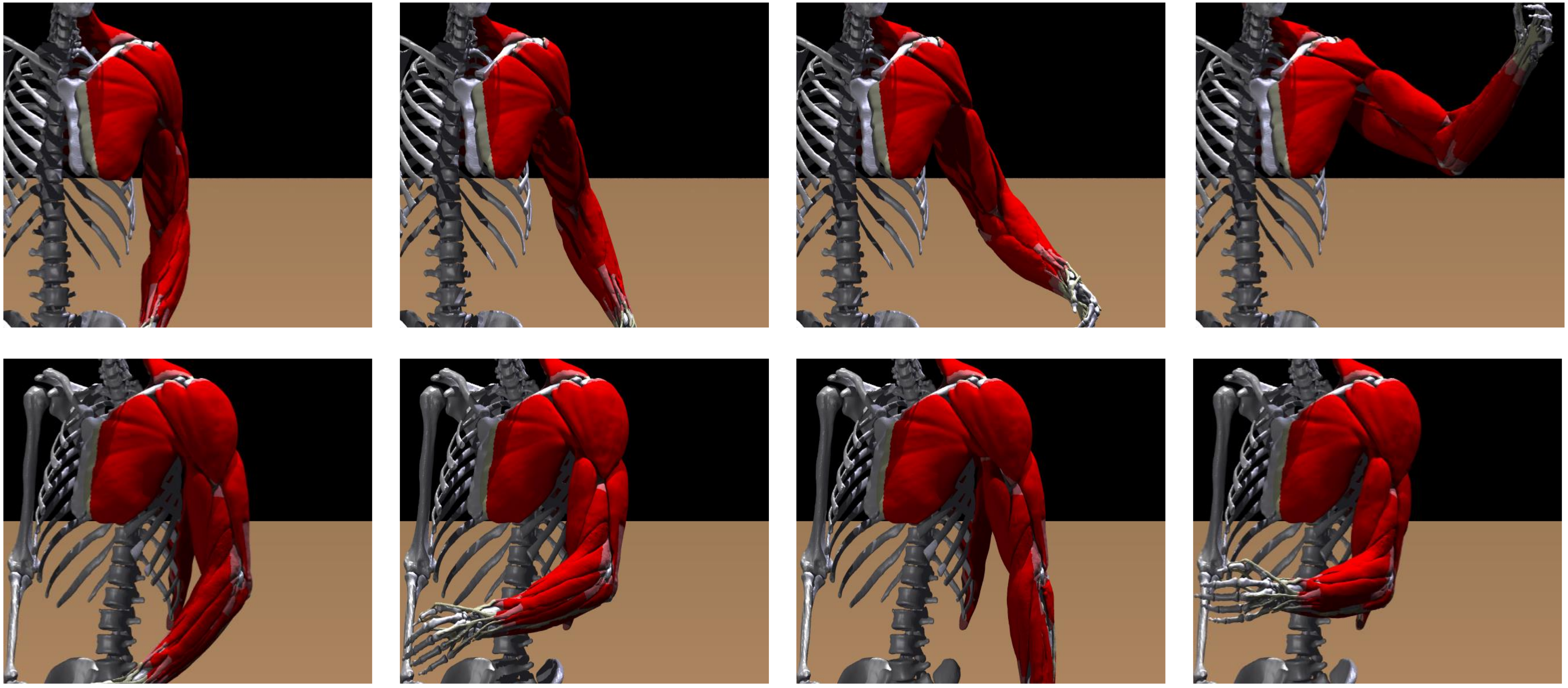
Muscles

- Can further improve the model by adding muscles which contract when activated and exert an internal force on the tetrahedral flesh mesh
- Can leverage existing datasets such as the NIH's Visible Human Project to obtain accurate muscle geometry

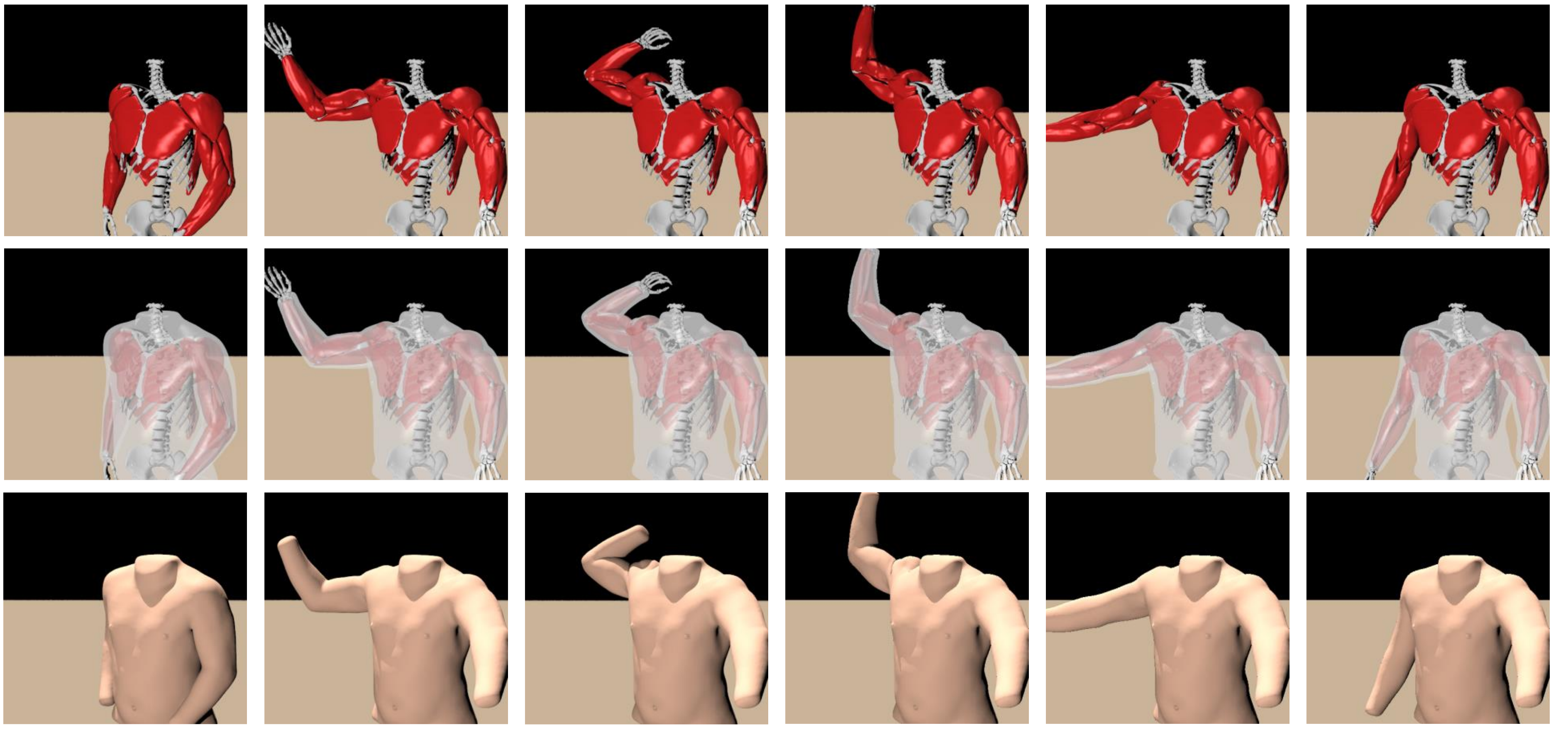


Muscles

- Animate the rigid body bones
- Solve an inverse problem to deduce muscle activations from bone motion
- Use the calculated muscle activations to simulate the tetrahedralized muscles (and the tetrahedralized flesh)

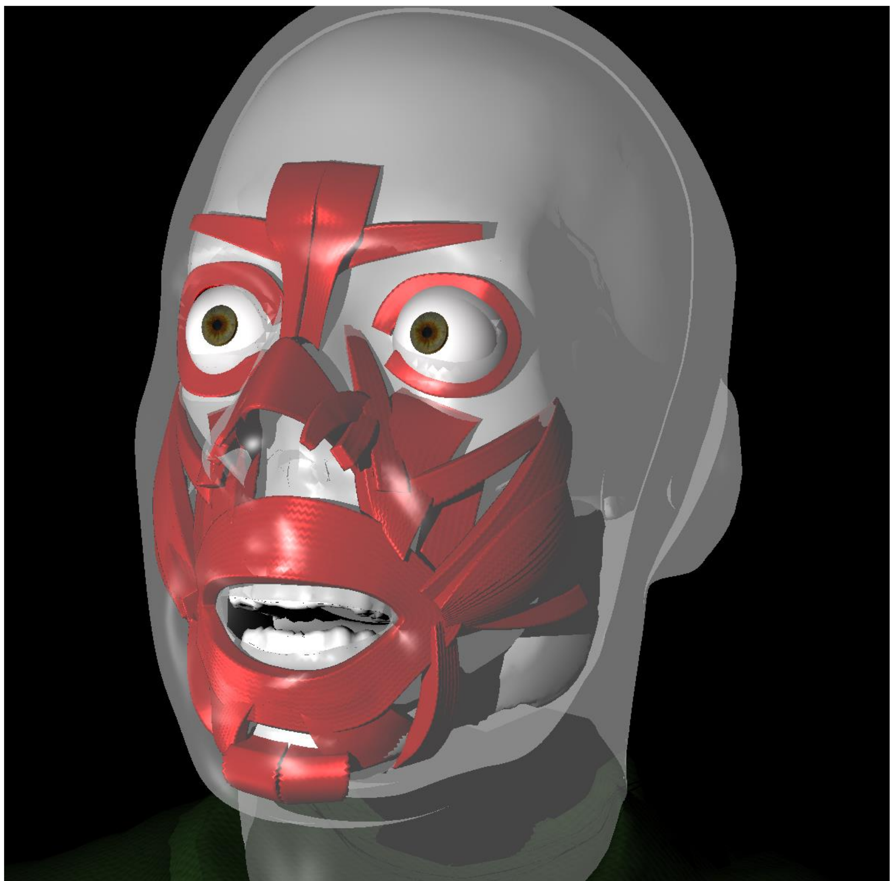
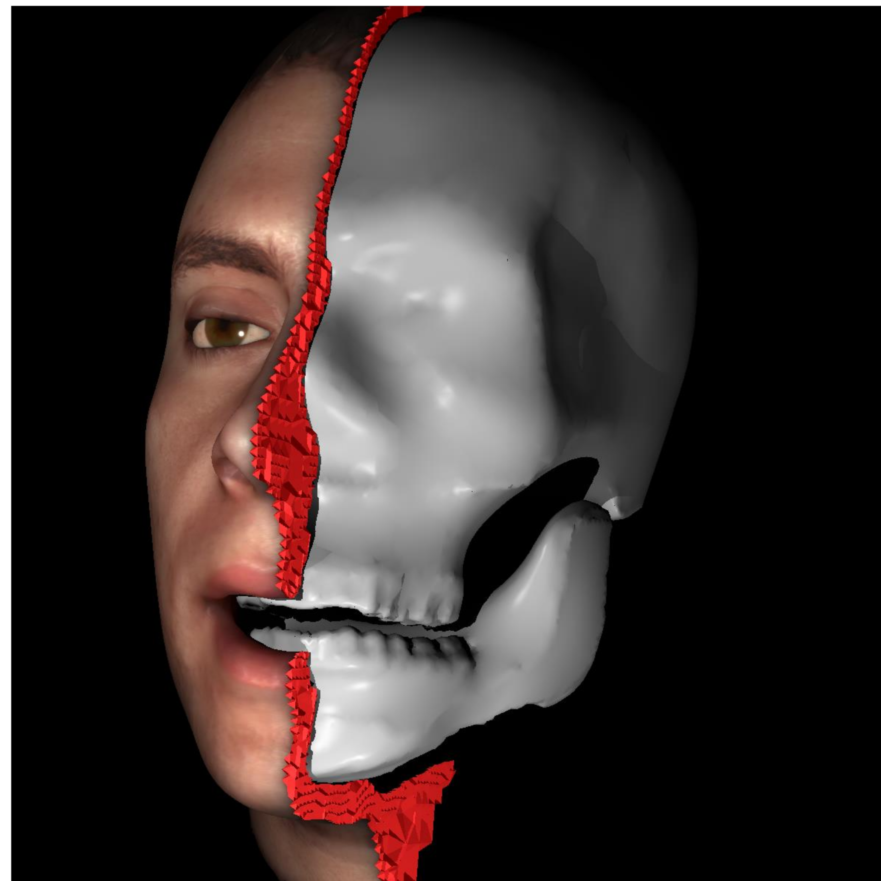


Muscles



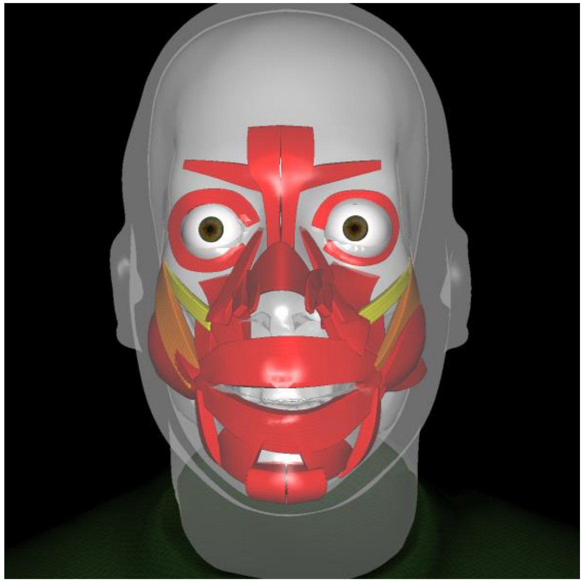
Anatomical Face Models

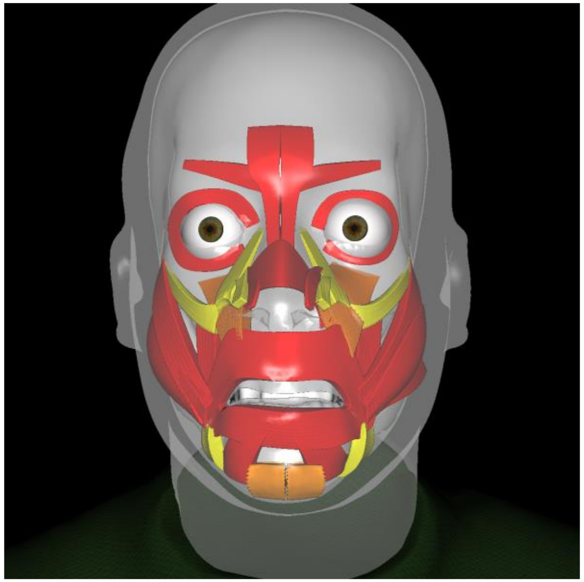
- Can add bones, muscles, and flesh for faces too…
- Animate the muscle activations and simulate the soft body flesh volume to obtain expressions

Anatomical Face Models

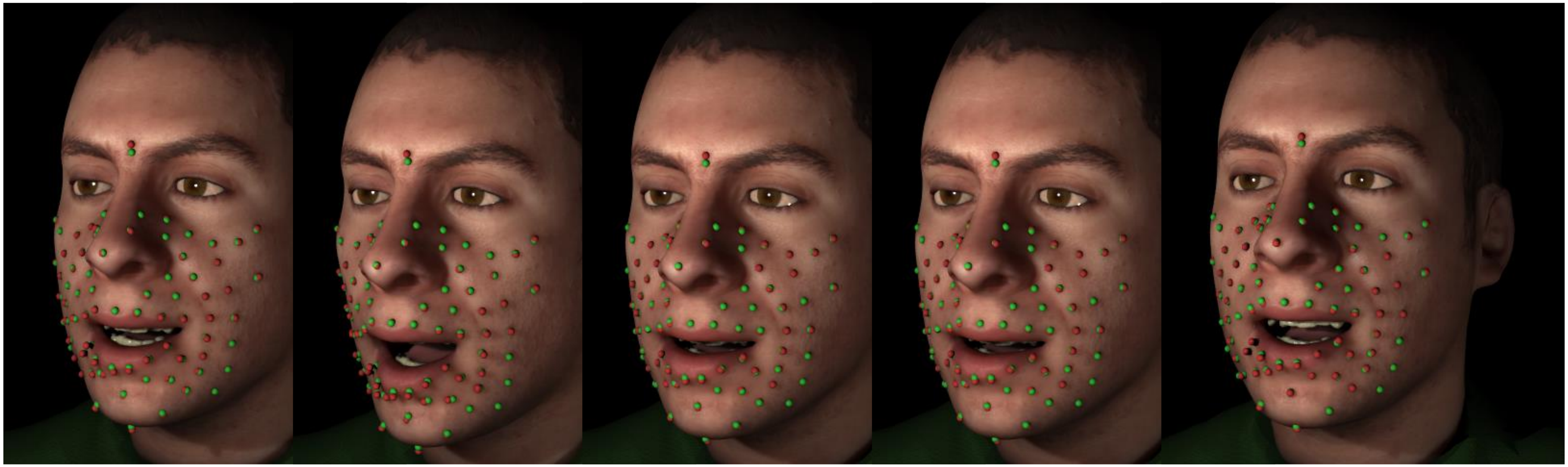
- Fully activated muscles are yellow and fully inactive are red





Estimating Muscle Activations

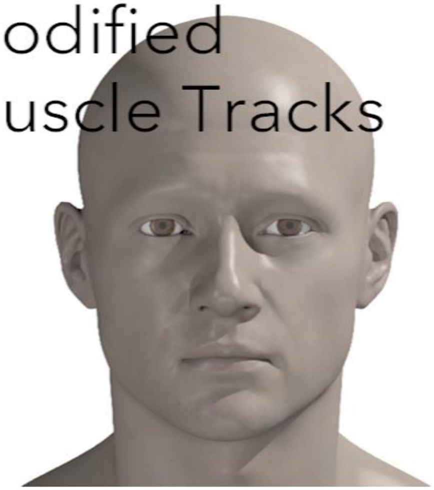
- Estimate muscle activations using motion capture
- For a given target facial shape (with target marker locations), solve an inverse problem to determine what muscle activations are required to match that shape (to match the markers)
- Facial expressions can also be modified by changing the joint angle of the jaw which causes the attached flesh to move and deform
- Can interpolate muscle activations and joint angles from different frames to obtain new physically valid facial expressions



Retargeting

Blendshape Performance

Modified Muscle Tracks



Modified Muscle Tracks

Retargeted Performance

Retargeted Muscles