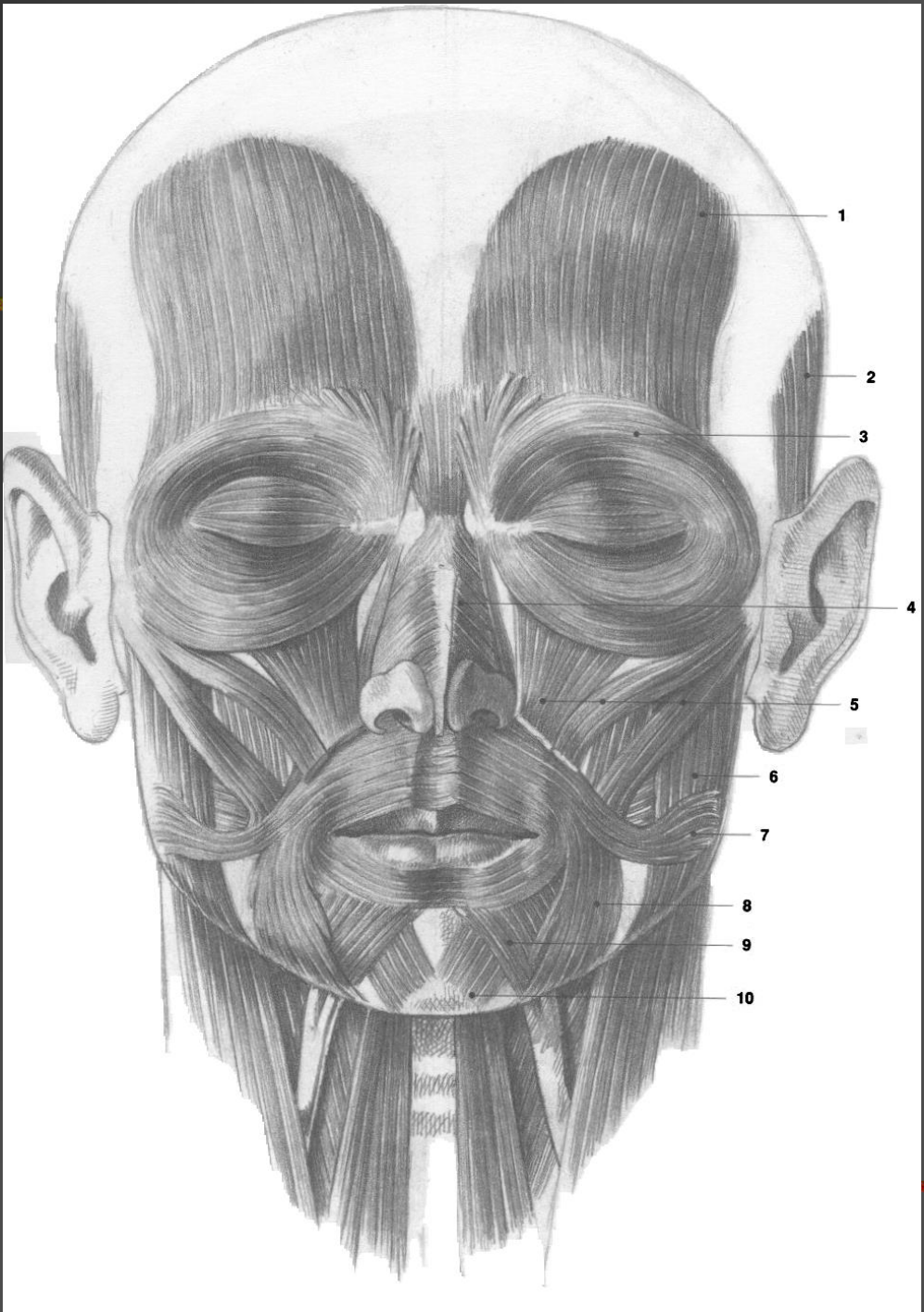
Facial Expressions & Rigging

CSE169: Computer Animation

Instructor: Steve Rotenberg

UCSD, Winter 2017

Facial Muscles



‘Universal’ Expression Groups

- Sadness
- Anger
- Happiness
- Fear
- Disgust
- Surprise

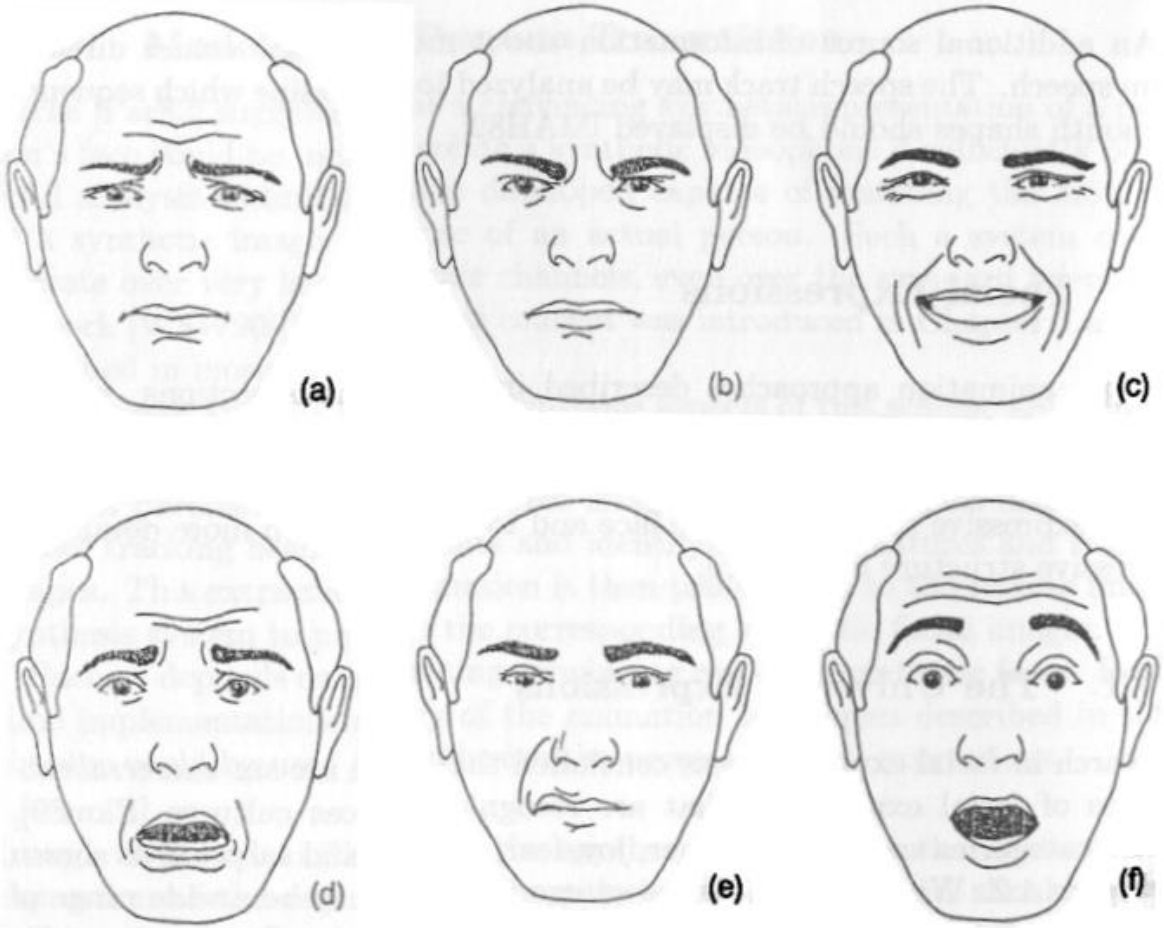


Figure 4.2.
The universal expressions: (a) sadness, (b) anger, (c) joy, (d) fear, (e) disgust, and (f) surprise.

FACS

- Facial Action Coding System (Ekman)
- Describes a set of ‘Action Units’ (AUs) that correspond to basic actions (some map to individual muscles, but other involve multiple muscles, or even joint motion)
- Examples:

1. Inner Brow Raiser	(Frontalis, Pars Medialis)
2. Outer Brow Raiser	(Frontalis, Pars Lateralis)
14. Dimpler	(Buccinator)
17. Chin Raiser	(Mentalis)
19. Tongue Out	
20. Lip Stretcher	(Risoris)
29. Jaw Thrust	
30. Jaw Sideways	
31. Jaw Clencher	

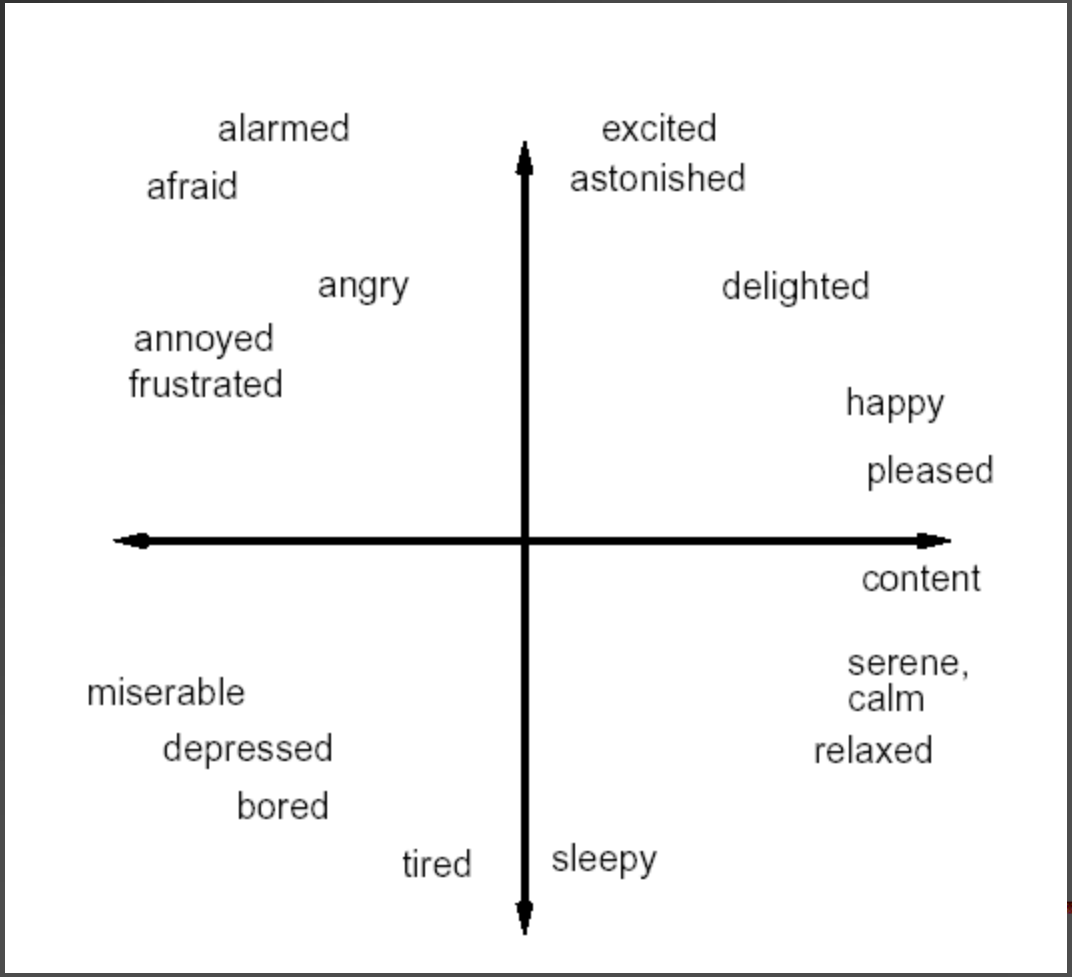
FACS

- Expressions are built from basic action units
- Happiness:

1. Inner Brow Raiser	(Frontalis, Pars Medialis)
6. Cheek Raiser	(Orbicularis Oculi, Pars Orbitalis)
12. Lip Corner Puller	(Zygomatic Major)
14. Dimpler	(Buccinator)

Emotional Axes

- Emotional states can loosely be graphed on a 2-axis system
- X=Happy/Sad
- Y=Excited/Relaxed



Facial Expression Reading

- Books
  - “The Artist’s Complete Guide to Facial Expression” (Faigin)
  - “The Expression of Emotions in Man and Animals” (Darwin)
  - “Computer Facial Animation” (Parke, Waters)
- Papers
  - “A Survey of Facial Modeling and Animation Techniques” (Noh)

Shape Interpolation

Bone Based Methods

- Using joints & skinning to do the jaw bone and eyeballs makes a lot of sense
- One can also use a pretty standard skeleton system to do facial muscles and skin deformations, using the blend weights in the skinning
- This gives quite a lot of control and is adequate for medium quality animation

Shape Interpolation Methods

- One of the most popular methods in practice is to use shape interpolation
- Several different key expressions are sculpted ahead of time
- The key expressions can then be blended on the fly to generate a final expression
- One can interpolate the entire face (happy to sad) or more localized zones (left eyelid, brow, nostril flare…)

Shape Interpolation

- Shape interpolation allows blending between several pre-sculpted expressions to generate a final expression
- It is a very popular technique, as it ultimately can give total control over every vertex if necessary
- However, it tends to require a lot of set up time
- It goes by many names:
  - Morphing
  - Morph Targets
  - Multi-Target Blending
  - Vertex Blending
  - Geometry Interpolation
  - etc.

Interpolation Targets

- One starts with a 3D model for the face in a neutral expression, known as the *base*
- Then, several individual *targets* are created by moving vertices from the base model
- The *topology* of the target meshes must be the same as the base model (i.e., same number of verts & triangles, and same connectivity)
- Each target is controlled by a DOF Φ_i that will range from 0 to 1

Morph Target DOFs

- We need DOFs to control the interpolation
- They will generally range from 0 to 1
- This is why it is nice to have a DOF class that can be used by joints, morph targets, or anything else we may want to animate
- Higher level code does not need to distinguish between animating an elbow DOF and animating an eyebrow DOF

Shape Interpolation Algorithm

- To compute a blended vertex position:

$$\mathbf{v}' = \mathbf{v}_{base} + \sum \phi_i \cdot \left(\Delta \mathbf{v}_i\right) \qquad \text{where } \Delta \mathbf{v}_i = \mathbf{v}_i - \mathbf{v}_{base}$$

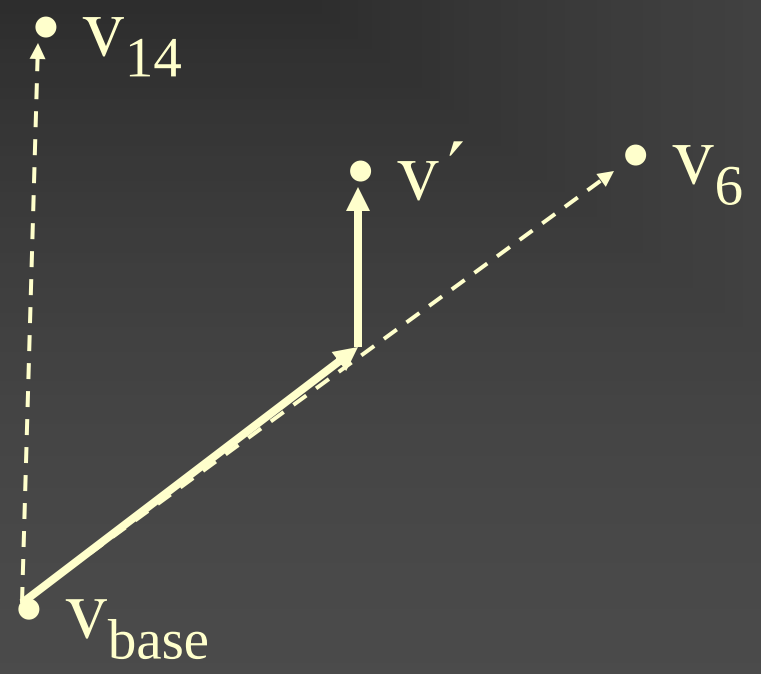
- The blended position is the base position plus a contribution from each target whose DOF value is greater than 0 (targets with a DOF value of 0 are essentially ‘off’ and have no effect)
- If multiple targets affect the same vertex, their results combine in a ‘reasonable’ way

Weighted Blending & Averaging

- Weighted sum: $x' = \sum_{i=0} w_i x_i$
- Weighted average: $\sum_{i=0} w_i = 1$
- Convex average: $0 \le w_i \le 1$
- Additive blend: $x' = x_0 + \sum_{i=1} w_i (x_i - x_0)$

$$= \left(1 - \sum_{i=1} w_i\right) x_0 + \sum_{i=1} w_i x_i$$

Additive Blend of Position



$\Phi_6=0.5$

$\Phi_{14}=0.25$

Normal Interpolation

- To compute the blended normal:

$$\mathbf{n}^* = \mathbf{n}_{base} + \sum \phi_i \cdot \left(\mathbf{n}_i - \mathbf{n}_{base}\right)$$

$$\mathbf{n}' = \frac{\mathbf{n}^*}{\left|\mathbf{n}^*\right|}$$

- Note: if the normal is going to undergo further processing (i.e., skinning), we might be able to postpone the normalization step until later

Shape Interpolation Algorithm

- To compute a blended vertex position:

$$\mathbf{v}' = \mathbf{v}_{base} + \sum \phi_i \cdot \left(\mathbf{v}_i - \mathbf{v}_{base}\right)$$

- The blended position is the base position plus a contribution from each target whose DOF value is greater than 0
- To blend the normals, we use a similar equation:

$$\mathbf{n}' = \mathbf{n}_{base} + \sum \phi_i \cdot \left(\mathbf{n}_i - \mathbf{n}_{base}\right)$$

- We won't normalize them now, as that will happen later in the skinning phase

Shape Interpolation and Skinning

- Usually, the shape interpolation is done in the skin's local space
- In other words, it's done *before* the actual smooth skinning computations are done

Smooth Skin Algorithm

- The deformed vertex position is a weighted average over all of the joints that the vertex is attached to. Each attached joint transforms the vertex as if it were rigidly attached. Then these values are blended using the weights:

$$\mathbf{v}'' = \sum w_i \mathbf{W}_i \cdot \mathbf{B}_i^{-1} \cdot \mathbf{v}'$$

- Where:
  - **v**'' is the final vertex position in world space
  - w_i is the weight of joint *i*
  - **v'** is the untransformed vertex position (output from the shape interpolation)
  - $\mathbf{B}_i$ is the binding matrix (world matrix of joint *i* when the skin was initially attached)
  - $\mathbf{W}_i$ is the current world matrix of joint *i* after running the skeleton forward kinematics
- Note:
  - **B** remains constant, so $\mathbf{B}^{-1}$ can be computed at load time
  - $\mathbf{B}^{-1} \cdot \mathbf{W}$ can be computed for each joint before skinning starts
- All of the weights must add up to 1:

$$\sum w_i = 1$$

Smooth Skinning Normals

- Blending normals is essentially the same, except we transform them as directions (x,y,z,0) and then renormalize the results

$$\mathbf{n}^* = \sum w_i \mathbf{W}_i \cdot \mathbf{B}_i^{-1} \cdot \mathbf{n}'$$

$$\mathbf{n}'' = \frac{\mathbf{n}^*}{\left|\mathbf{n}^*\right|}$$

Equation Summary

- Skeleton

$$\mathbf{L} = \mathbf{L}_{jnt}\left(\phi_1, \phi_2, \ldots, \phi_N\right)$$

$$\mathbf{W} = \mathbf{W}_{parent} \cdot \mathbf{L}$$

- Morphing

$$\mathbf{v}' = \mathbf{v}_{base} + \sum \phi_i \cdot \left(\mathbf{v}_i - \mathbf{v}_{base}\right)$$

$$\mathbf{n}' = \mathbf{n}_{base} + \sum \phi_i \cdot \left(\mathbf{n}_i - \mathbf{n}_{base}\right)$$

- Skinning

$$\mathbf{v}'' = \sum w_i \mathbf{W}_i \cdot \mathbf{B}_i^{-1} \cdot \mathbf{v}'$$

$$\mathbf{n}^* = \sum w_i \mathbf{W}_i \cdot \mathbf{B}_i^{-1} \cdot \mathbf{n}'$$

$$\mathbf{n}'' = \frac{\mathbf{n}^*}{\left|\mathbf{n}^*\right|}$$

Morph Target Storage

- Morph targets can take up a lot of memory. This is a big issue for video games, but less of a problem in movies.
- The base model is typically stored in whatever fashion a 3D model would be stored internally (verts, normals, triangles, texture maps, texture coordinates…)
- The targets, however, don’t need all of that information, as much of it will remain constant (triangles, texture maps…)
- Also, most target expressions will only modify a small percentage of the verts
- Therefore, the targets really only need to store the positions and normals of the vertices that have moved away from the base position (and the indices of those verts)

Morph Target Storage

- Also, we don't need to store the full position and normal, only the difference from the base position and base normal
- i.e., other than storing v_3, we store v_3-v_{base}
- There are two main advantages of doing this:
  - Fewer vector subtractions at runtime (saves time)
  - As the deltas will typically be small, we should be able to get better compression (saves space)

Morph Target Storage

- In a pre-processing step, the targets are created by comparing a modified model to the base model and writing out the ‘difference’
- The information can be contained in something like this:

```
class MorphTarget {
    int NumVerts;
    int Index [ ];
    Vector3 DeltaPosition [ ];
    Vector3 DeltaNormal [ ];
}
```

Colors and Other Properties

- In addition to interpolating the positions and normals, one can interpolate other per-vertex data:
  - Colors
  - Alpha
  - Texture coordinates
  - Auxiliary shader properties

Vascular Expression

- Vascular expression is a fancy term to describe blushing and other phenomena relating to the color change in the face
- Adding subtle changes in facial color that relate to skin distortion can help improve realism
- This can be achieved either by blending a color values with every vertex (along with the position and normal)
- Alternately, one could use a blush texture map controlled by a blended intensity value at each vertex

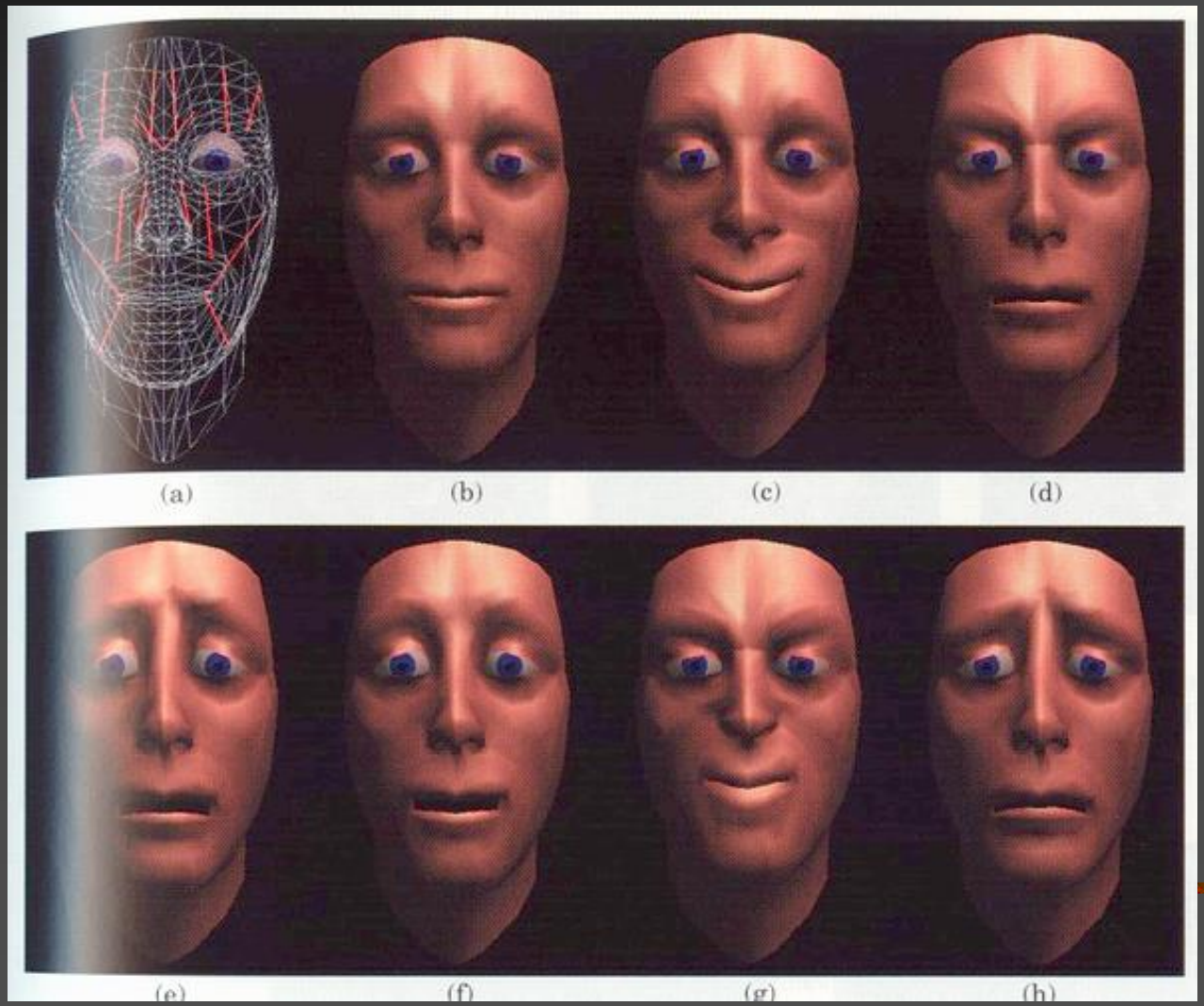
Wrinkles

- One application of auxiliary data interpolation is adding wrinkles
- Every vertex stores an auxiliary property indicating how wrinkled that area is
  - On the base model, this property would probably be 0 in most of the verts, indicating an unwrinkled state
  - Target expressions can have this property set at or near 1 in wrinkled areas
- When facial expressions are blended, this property is blended per vertex just like the positions and normals (but should be clamped between 0 and 1 when done)
- For rendering, this value is used as a scale factor on a wrinkle texture map that is blended with the main face texture
- Even better, one could use a wrinkle bump map or displacement map

Artificial Muscle Methods

- With this technique, muscles are modeled as deformations that affect local regions of the face
- The deformations can be built from simple operations, joints, interpolation targets, FFDs, or other techniques

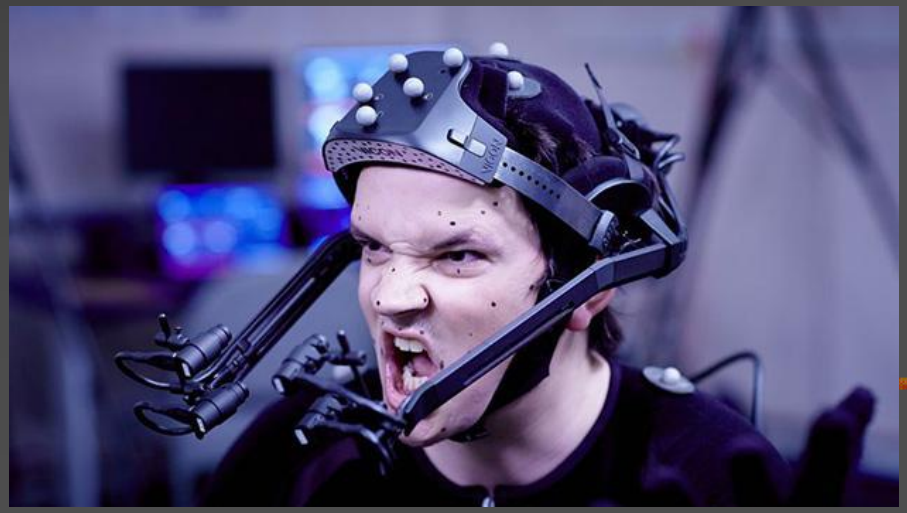
Artificial Muscles



Facial Features

- Key Facial Features
  - Deformable Skin
  - Hair
  - Eyes
  - Articulated Jaw (teeth…)
  - Tongue
  - Inside of mouth
- Each of these may require a different technical strategy

Motion Capture



Gollem

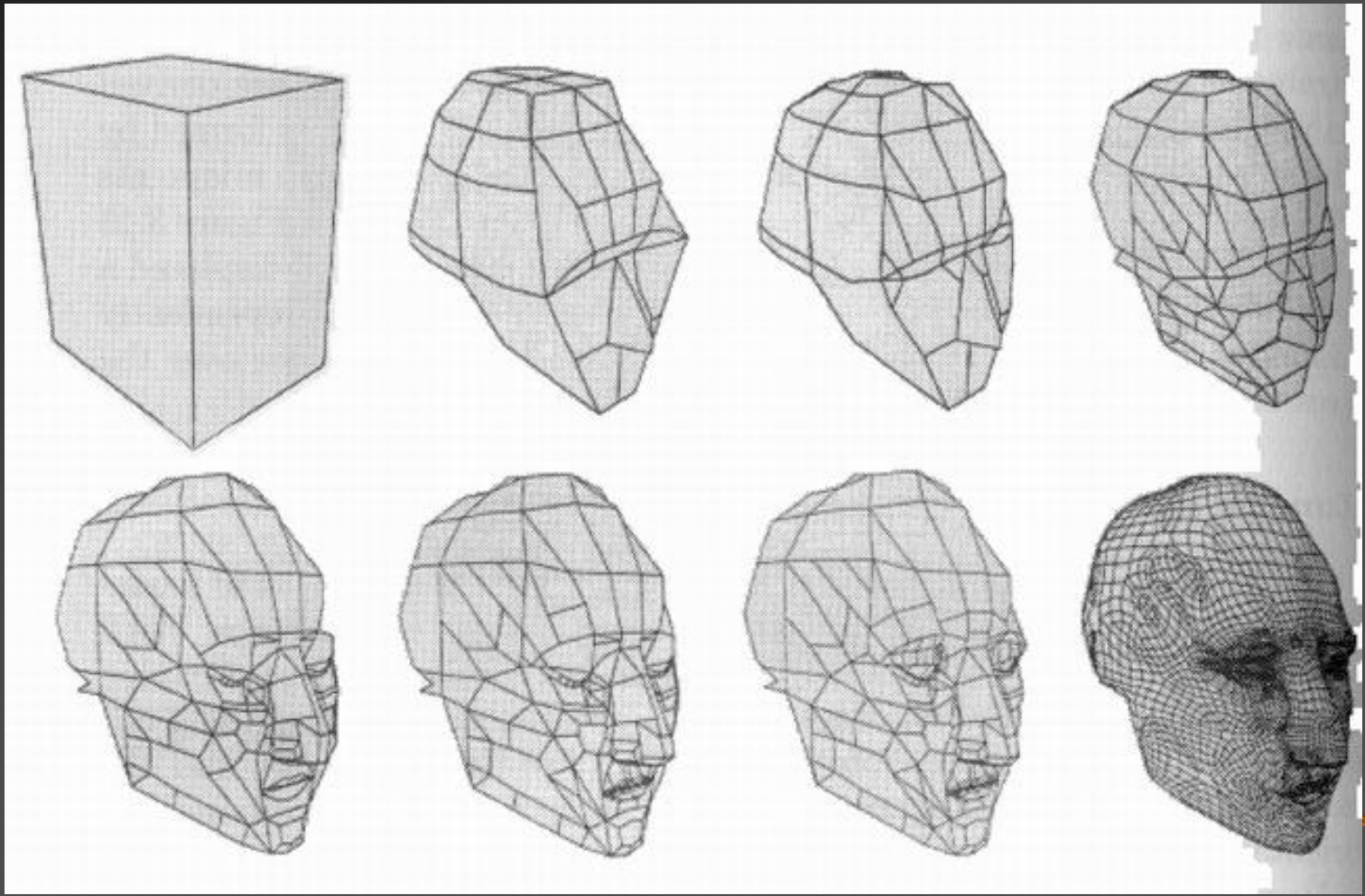
Image credits: (Above) 'Lord of the Rings - Return of the King' (Courtesy of New Line Productions © 2
New Line Productions. All Rights Reserved)

Facial Modeling

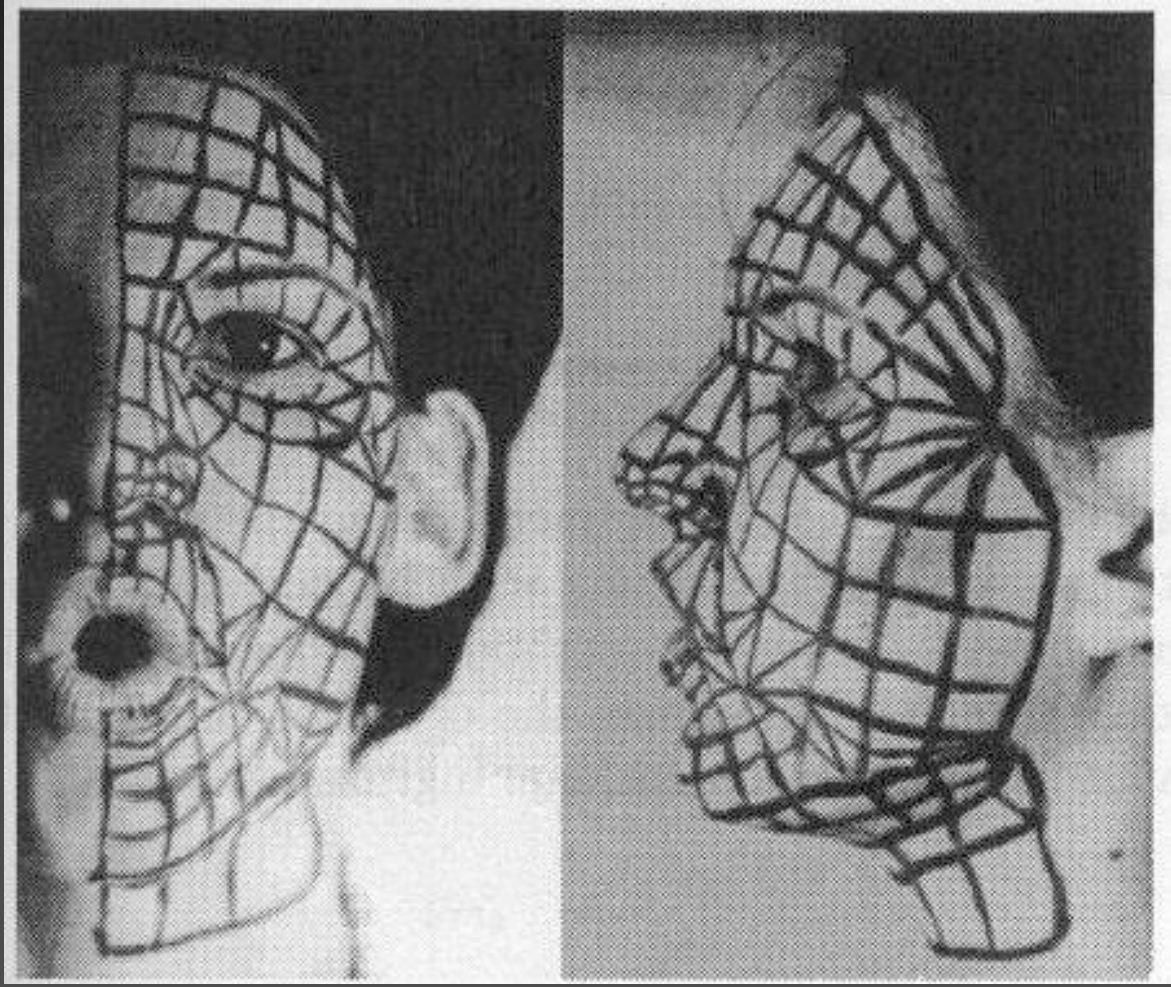
Facial Modeling

- Preparing the facial geometry and all the necessary expressions can be a lot of work
- There are several categories of facial modeling techniques
  - Traditional modeling (in an interactive 3D modeler)
  - Photograph & digitize (in 2D with a mouse)
  - Sculpt & digitize (with a 3D digitizer)
  - Scanning (laser)
  - Vision (2D image or video)

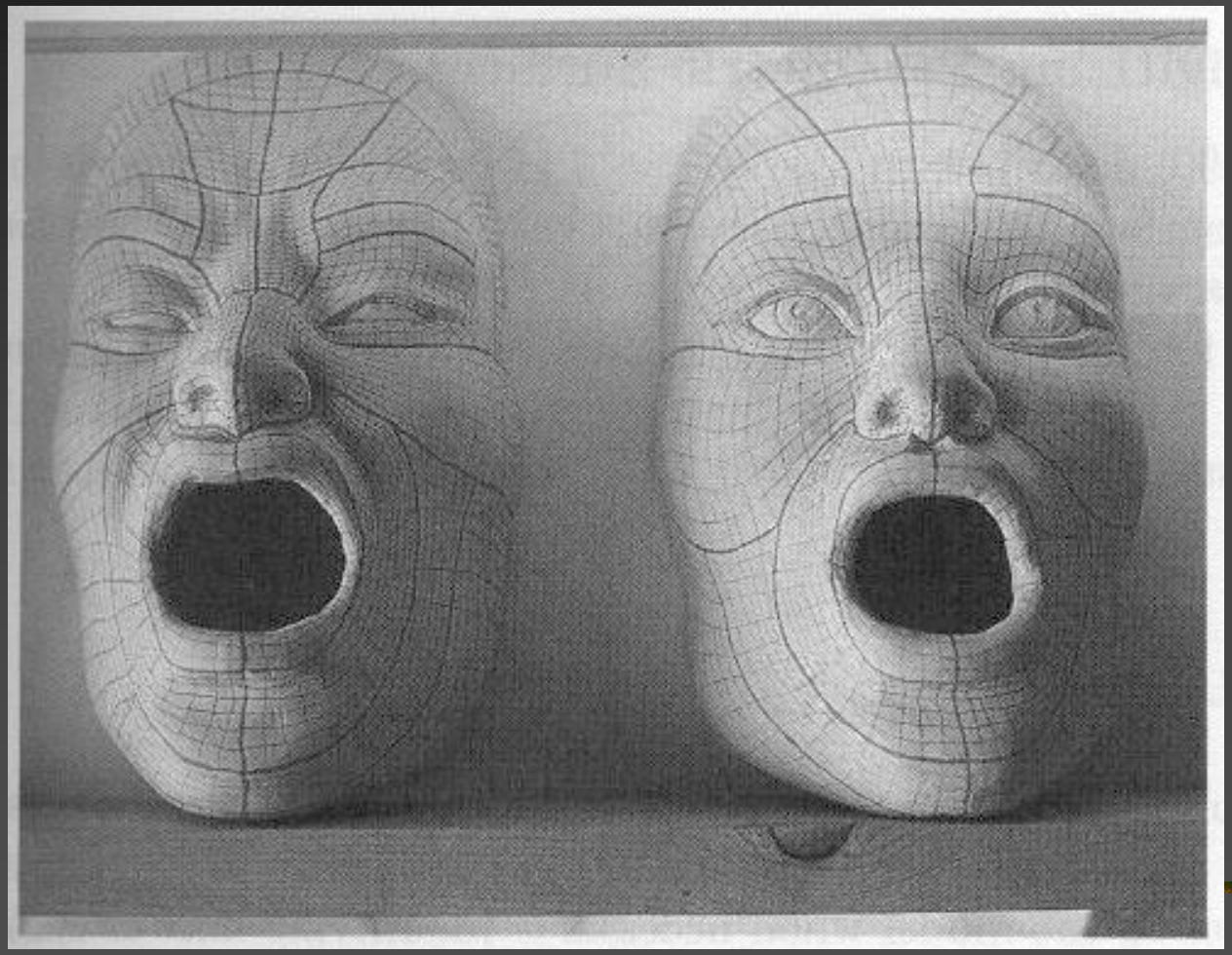
Traditional Modeling



Photograph & Digitize

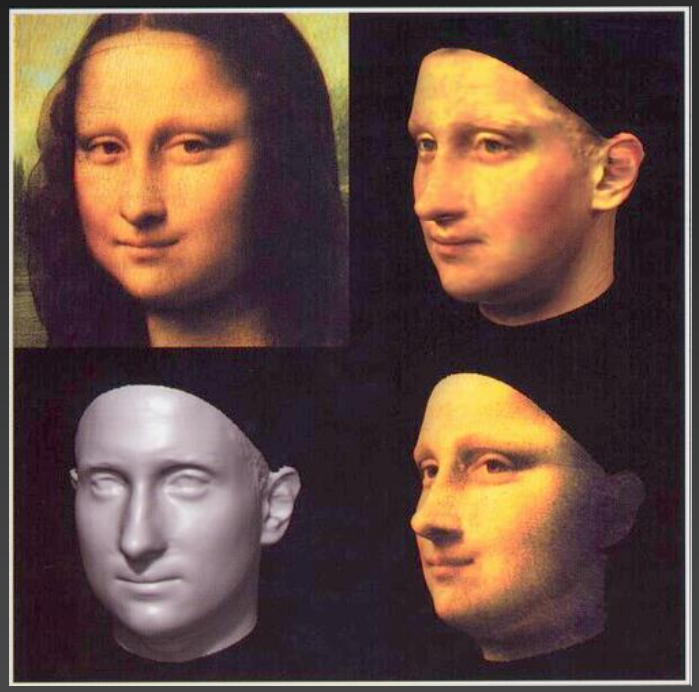


Sculpt & Digitize



Laser Scan

Computer Vision



(a) High-res geometry (b) Real-time hybrid map rendering (c) Offline SSS rendering

Project 2 Extra Credit

Textures in .skin file

- The modified version of the .skin file with texture information will have an array of 2D texture coordinates after the array of normals

```
texcoords [numverts] {
        [tx] [ty]
}
```

- If will also have a material definition that references a texture map. This will appear before the triangle array

```
material [mtlname] {
        texture [texname]
}
```

Morph File

```
positions [numverts] {
    [index] [x] [y] [z]
}
normals [numverts] {
    [index] [x] [y] [z]
}
```

Rigging

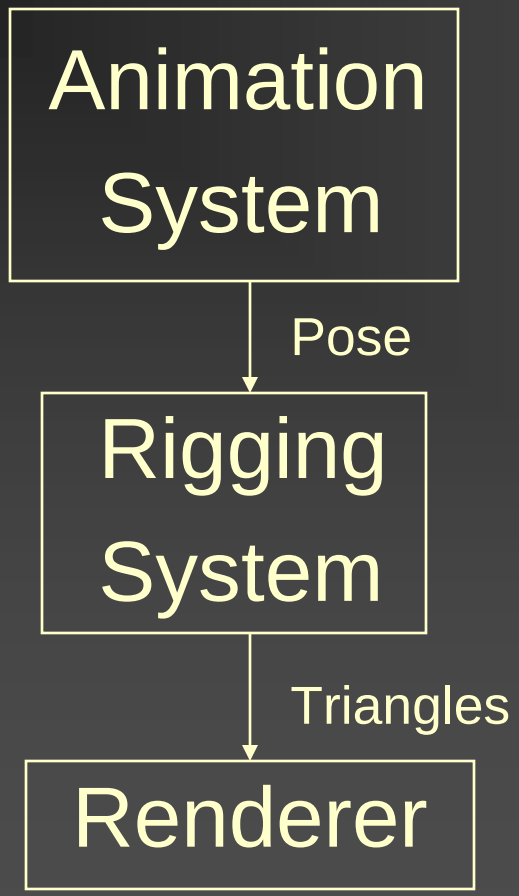
Rigging

- A rig is like a virtual puppet
- A rig contains several DOFs, each corresponding to an animatable parameter within the puppet
- DOFs can control:
  - Joint rotations, translations
  - Morph targets
  - Other things…
- Higher level animation code will specify values for the DOFs (i.e., pose the rig)

Rigging

- Ultimately, the rig takes DOF values from the animation system and generates the posed geometry of the character in world space
- This might involve:
  - Computing world joint matrices (posing the skeleton)
  - Interpolating verts in local space (morphing)
  - Transforming verts to world space (skinning)
- This geometry is then rendered through a rendering system (OpenGL…)

Rigging and Animation



Rig Data Flow

$$\mathbf{\Phi} = \begin{bmatrix} \phi_1 & \phi_2 & \cdots & \phi_N \end{bmatrix}$$

$$\Downarrow$$

$$\boxed{\text{Rigging System}}$$

$$\Downarrow$$

$$\mathbf{v}'', \mathbf{n}''$$

Skeleton, Morph, & Skin Data Flow

$$\begin{bmatrix}\phi_1 & \phi_2 & \cdots & \phi_M\end{bmatrix} \qquad \begin{bmatrix}\phi_{M+1} & \phi_{M+2} & \cdots & \phi_N\end{bmatrix}$$

$$\mathbf{L} = \mathbf{L}_{jnt}(\phi_1, \phi_2, \ldots, \phi_m)$$

$$\mathbf{W} = \mathbf{W}_{parent} \cdot \mathbf{L}$$

$$\mathbf{v}' = \mathbf{v}_{base} + \sum \phi_i \cdot (\mathbf{v}_i - \mathbf{v}_{base})$$

$$\mathbf{n}' = \mathbf{n}_{base} + \sum \phi_i \cdot (\mathbf{n}_i - \mathbf{n}_{base})$$

$$\mathbf{v}'' = \sum w_i \mathbf{W}_i \cdot \mathbf{B}_i^{-1} \cdot \mathbf{v}'$$

$$\mathbf{n}^* = \sum w_i \mathbf{W}_i \cdot \mathbf{B}_i^{-1} \cdot \mathbf{n}'$$

$$\mathbf{n}'' = \frac{\mathbf{n}^*}{|\mathbf{n}^*|}$$

$$\mathbf{v}'', \mathbf{n}''$$

Layered Approach

- We use a simple layered approach
  - Skeleton Kinematics
  - Shape Interpolation
  - Smooth Skinning
- Most character rigging systems are based on some sort of layered system approach combined with general purpose data flow to allow for customization

Equation Summary

- Skeleton

$$\mathbf{L} = \mathbf{L}_{jnt}\left(\phi_1, \phi_2, \ldots, \phi_N\right)$$

$$\mathbf{W} = \mathbf{W}_{parent} \cdot \mathbf{L}$$

- Morphing

$$\mathbf{v}' = \mathbf{v}_{base} + \sum \phi_i \cdot \left(\mathbf{v}_i - \mathbf{v}_{base}\right)$$

$$\mathbf{n}' = \mathbf{n}_{base} + \sum \phi_i \cdot \left(\mathbf{n}_i - \mathbf{n}_{base}\right)$$

- Skinning

$$\mathbf{v}'' = \sum w_i \mathbf{W}_i \cdot \mathbf{B}_i^{-1} \cdot \mathbf{v}'$$

$$\mathbf{n}^* = \sum w_i \mathbf{W}_i \cdot \mathbf{B}_i^{-1} \cdot \mathbf{n}'$$

$$\mathbf{n}'' = \frac{\mathbf{n}^*}{\left|\mathbf{n}^*\right|}$$

DOF Mapping & Expressions

DOF Types

- In addition to controlling joints and morph targets, DOFs can be extended to manipulate any *high level* parameter that the animator wants to control
- One could make DOFs to:
  - Turn the character green
  - Extend/flex all fingers in a hand simultaneously
  - Make the character's hair stand up
  - Morph the character from a man into a hairy monster
  - Control the intensity of a light
  - Control the creation rate of a particle system

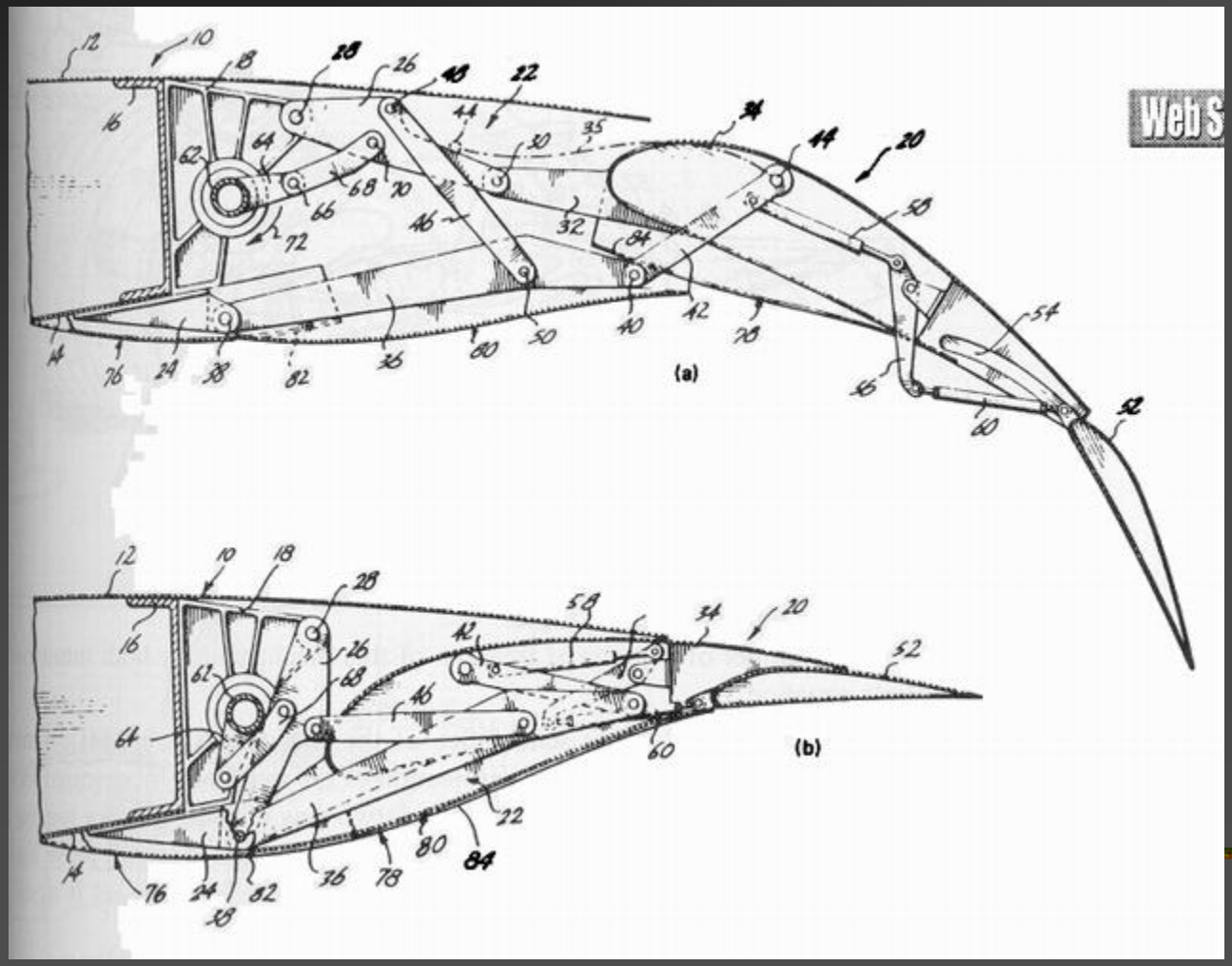
Full Body Morphing

- One can also rig up a DOF to morph an entire character (say from a human to a giant hairy monster)
- Morphing is made easier if the topology of the two characters matches (both skeleton & skin topology)
- To do this, one must interpolate a lot of data:
  - Skin positions & normals
  - Skin weights & attachment info
  - Bone offsets
  - Texture maps, other visual properties
  - Other stuff…

Grouping DOFs

- You can have one DOF control several properties. For example:
  - A DOF that makes all of the joints in a finger flex or extend simultaneously
  - Elbow DOF that controls both the elbow rotation and a morph target for the bicep deformation
  - Head DOF that rotates several vertebra in the neck
  - Retract DOF that controls a landing gear retraction
  - DOF to control a highly constrained mechanical system
  - Smile DOF that controls several individual muscles
- For flexibility, it's nice to be able to have the master DOF range from say 0…1 and allow each slave DOF to scale that number if necessary
- For more flexibility, you can use expressions…

Grouping DOFs



DOF Expressions

- For more flexibility, its nice to be able to run arbitrary expressions with DOF values
- An expression takes one or more DOFs as inputs and sets an external DOF as output
- An expression can literally be any mathematical expression:

DOF[27] = DOF[3] * 6.0 – sin (DOF[2]) + DOF[14]

- Rather than being hard-coded in C++, it's nice if expressions can be interpreted at runtime

DOF Mappings

- A rig can be implemented as an array of pointers to DOFs
- The DOF order is important and must be consistent between the rigging & animation systems
- The DOFs themselves exist as internal objects used in the skeleton, morph system, and in expressions
- Normally, we would have a single rig that controls all of the character's DOFs that we wish to animate
- Alternately, we could:
  - Have a rig that controls a subset of a character's DOFs
  - Have one rig that maps to more than one character
  - Have several different rigs for the same character to be used for different purposes

Minimalist Rigging

- It’s a good idea to use as few DOFs as possible when rigging a character
- Some reasons include:
  - Keeps the interface to controlling the character simpler. This makes the animator’s life easier.
  - Reduces the amount of animation data needed for playback. This is important in video games, as animation data tends to take up a lot of space