Modélisation 3D & Simulation

I. Kinematic modeling

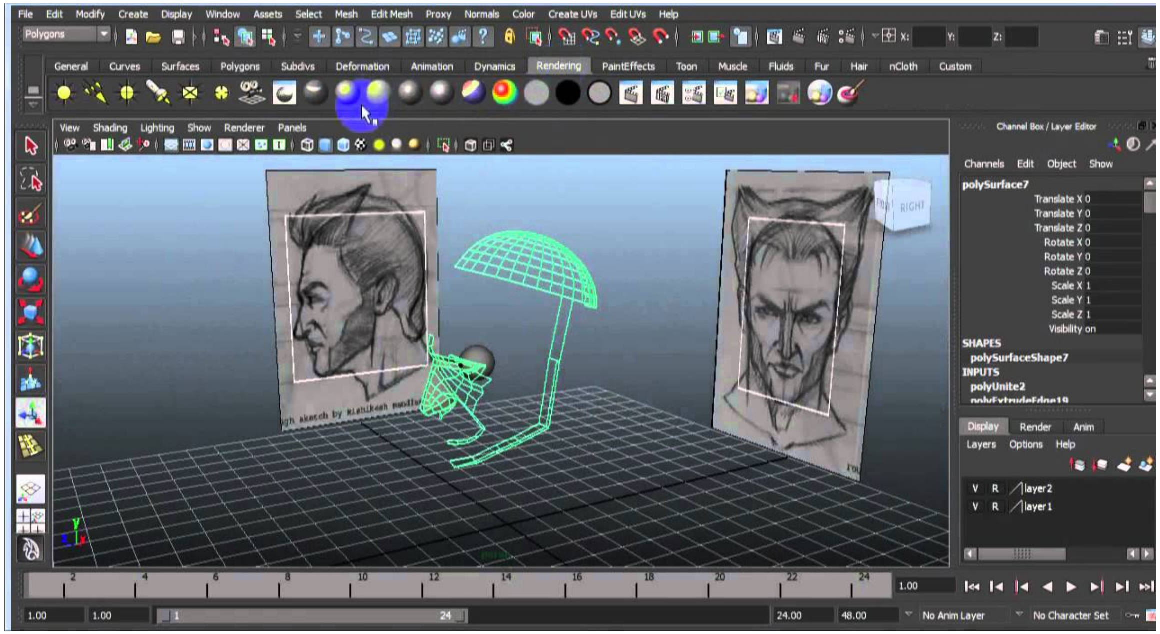
1. Facial modeling

Hyewon SEO

Equipe 'MLMS', ICube

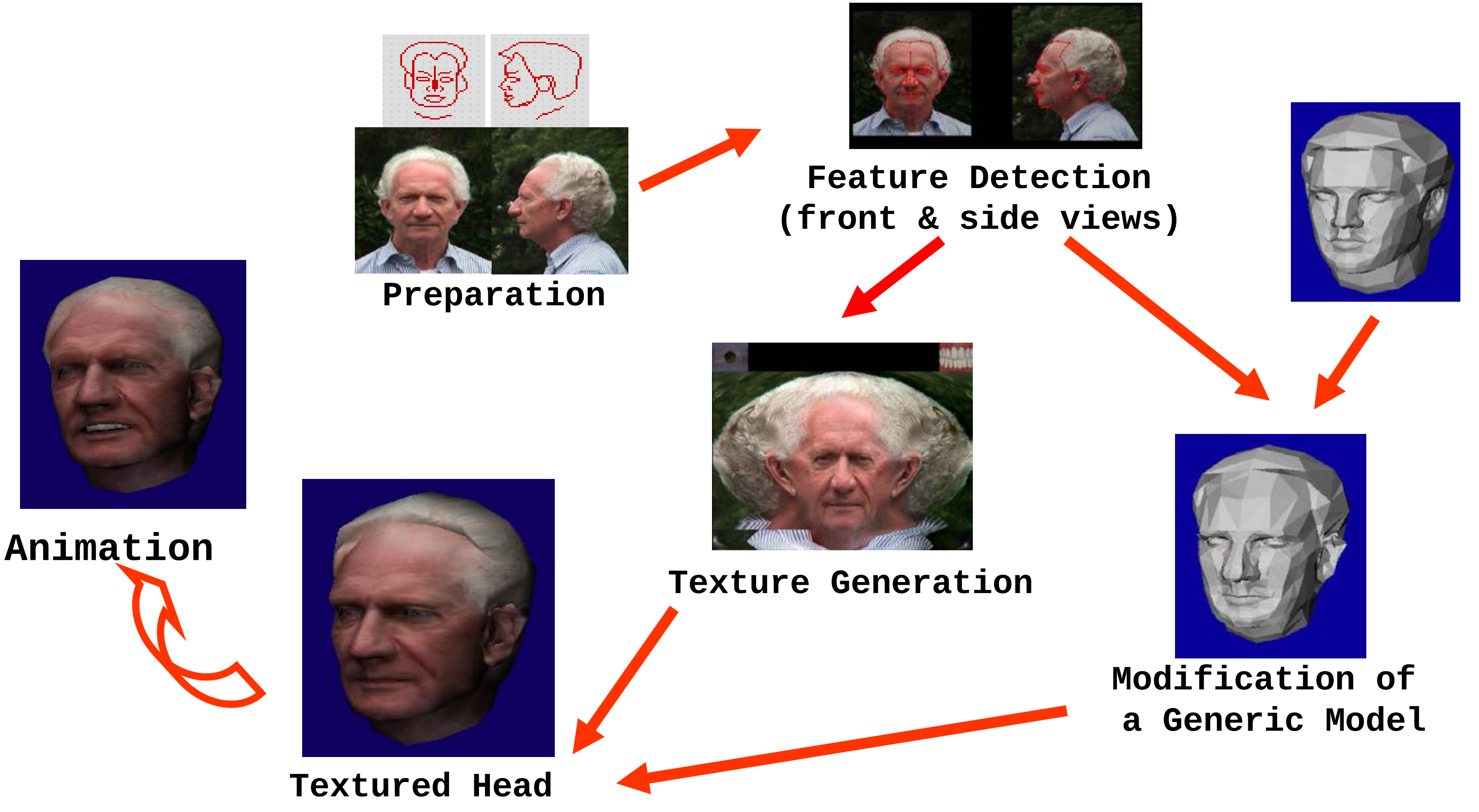
Facial shape modeling: static

- **By interactive design**
- **From existing CAD files**

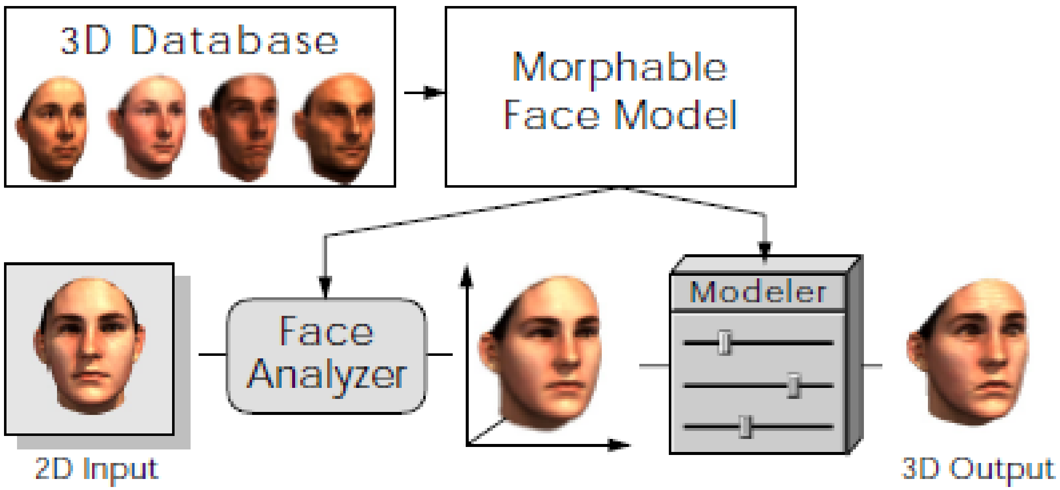


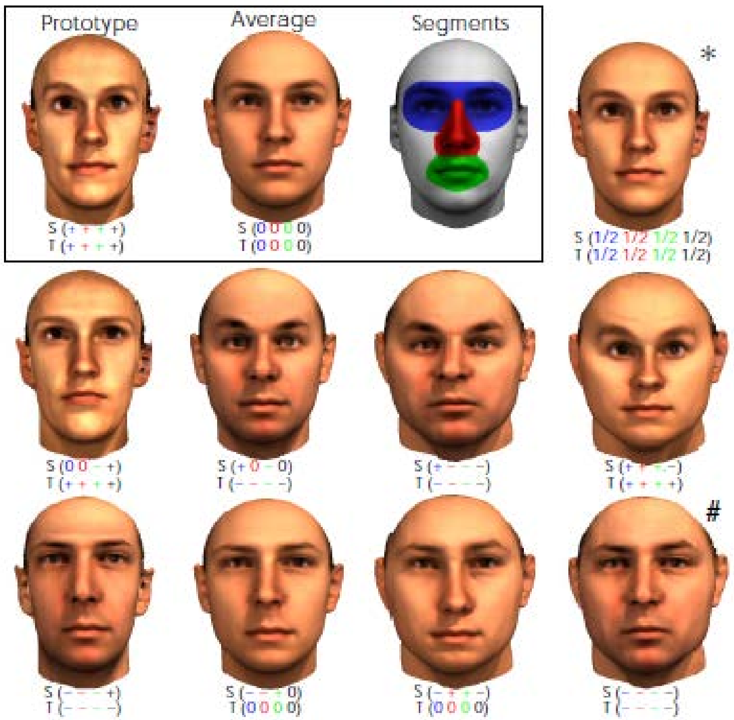
Facial shape modeling: static

- Using photos



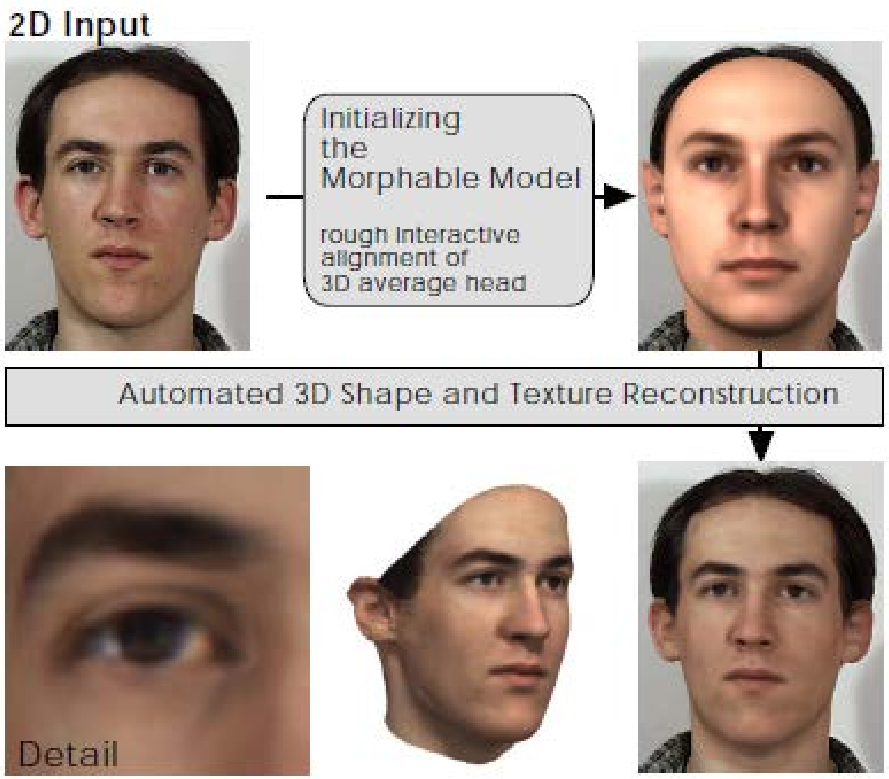
Facial shape modeling: static

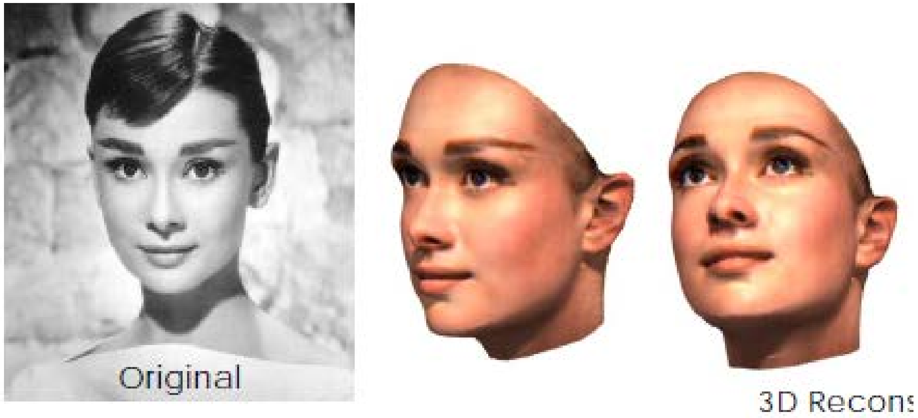




- **A morphable face model for the synthesis of 3D faces**

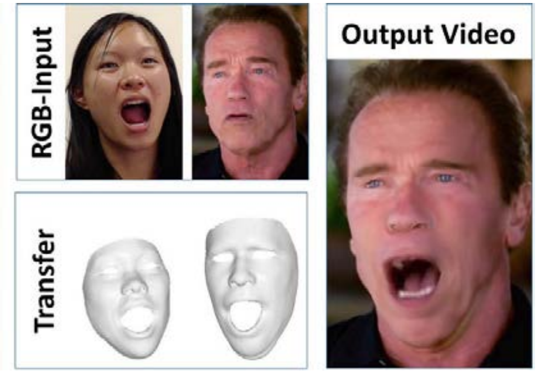
Facial shape modeling: static





Facial shape modeling: dynamic (Facial animation)

- Challenging and long-sought aspects of character animation



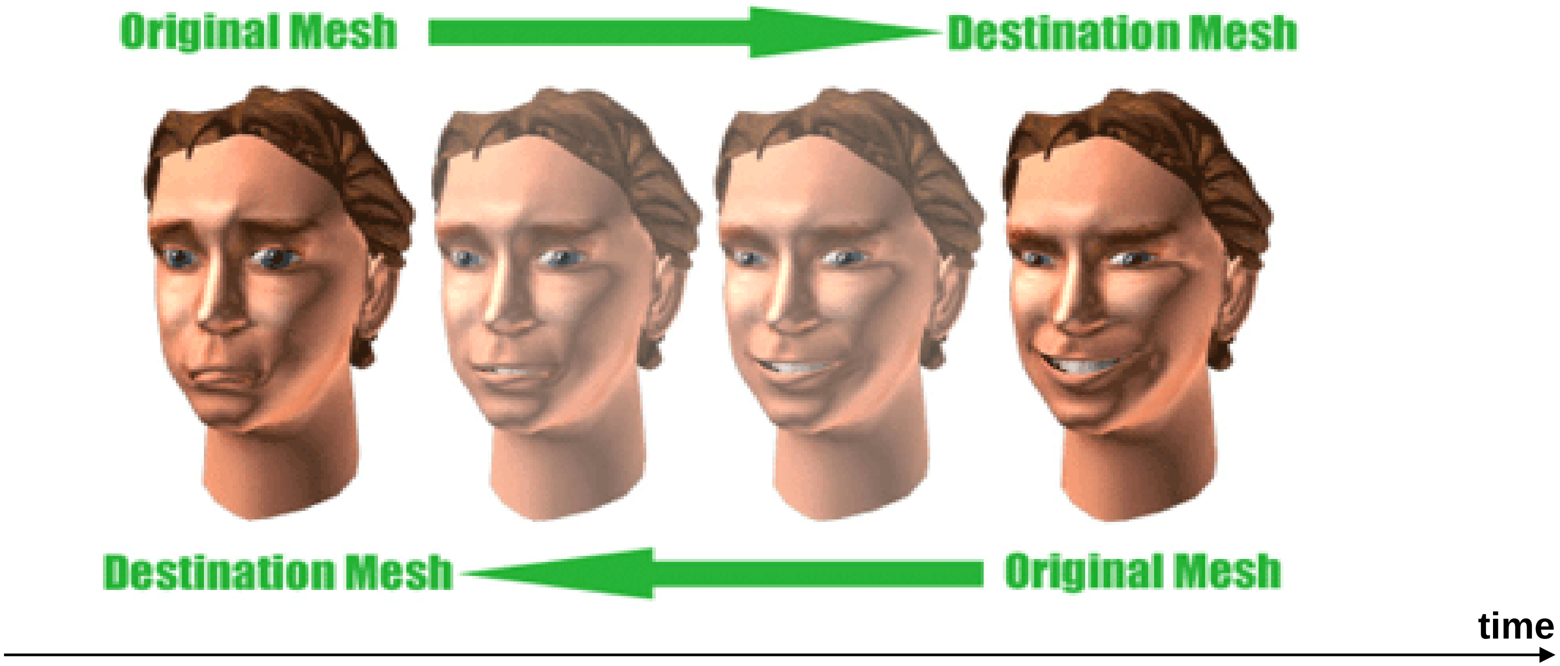
- Interactive design
  + Interpolation technique

- Facial motion capture:
  (video, marker-based mocap…)
  + Animation retargeting

- Data-driven generative methods
  Impressive results start to appear, recently

Keyframe based facial animation

- **Temporal interpolation, to generate an animation (sequence of poses)**



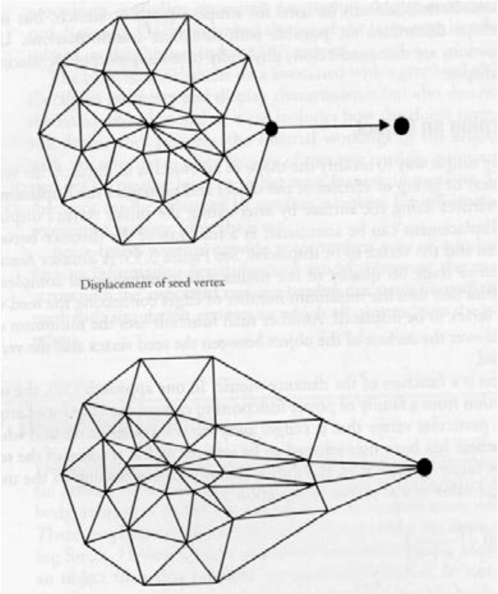
$$\mathbf{P} = (t\text{-}1)\ \mathbf{P}_0 + t\ \mathbf{P}_1 \text{ for } t \in [0,1]$$

Kinematic facial modeling

Distance-Based Vertex Displacement

- **Cannot define locations/trajectories of all vertices by hand..**
- **Work on *seed* vertices**

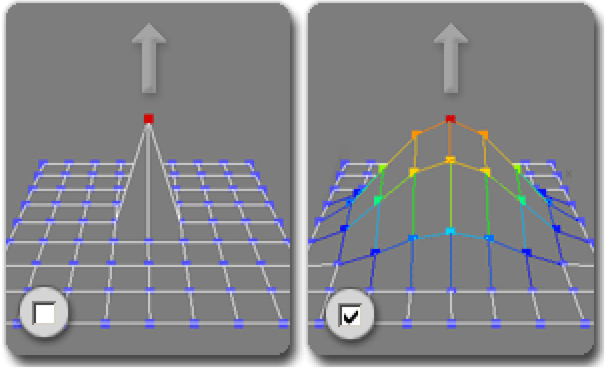
  Key, landmark, marker, feature, anchor, …
- **Effect nearby vertices**
  - **Usually by computation**
  - **With some controllable parameters**
- **A facial 'pose' can be generated in this way**



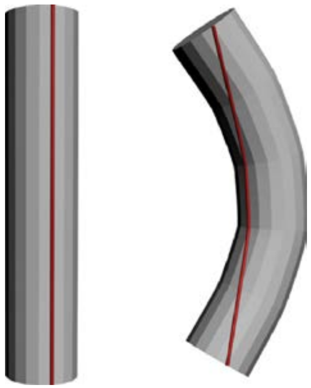
https://www.youtube.com/watch?v=TrgNKGjSyxA&index=2&list=PL8VqaFf6Kz-JbeYLvi7fgQvH-dHMhIHwT

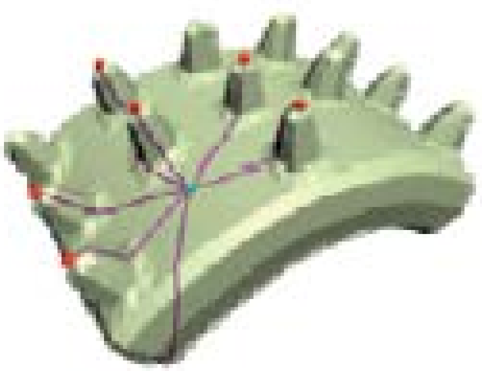
Facial poses by landmark displacements

Defining Distance-based Functions



- If vertex *i* is displaced by $\boxed{(dx, dy, dz)}$ units
  - Displace each neighbor, *j*, of *i* by $\boxed{(dx, dy, dz)} * f(i, j)$
- *f (i,j)* is typically a function of <u>distance</u>
  - Euclidean distance
  - (Smallest!) number of edges from *i* to *j*
  - Distance along surface, aka geodesic distance



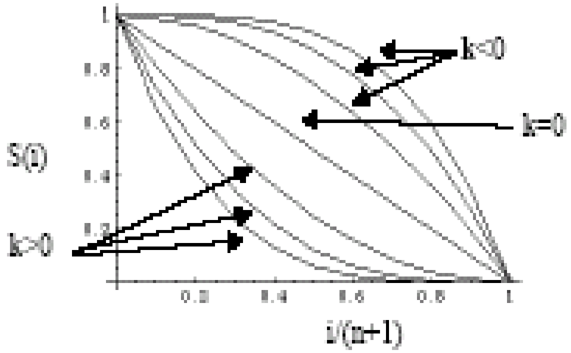


Vertex Displacement Function

$$f(d) = 1.0 - \left(\frac{d}{n+1}\right)^{k+1} ; k \geq 0$$

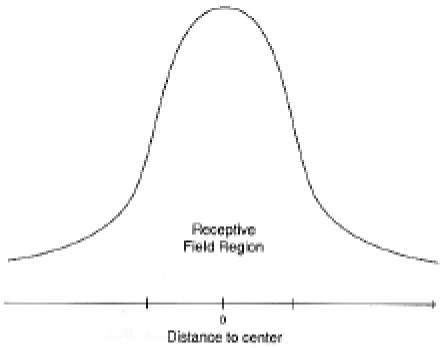
$$f(d) = \left(1.0 - \left(\frac{d}{n+1}\right)\right)^{-k+1} ; k < 0$$

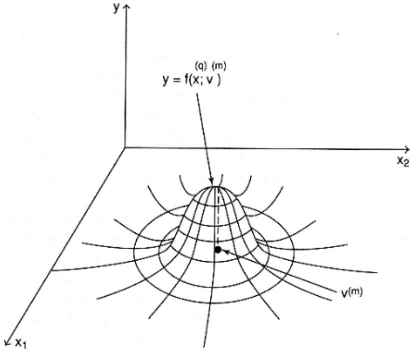
- ***d*** **is the (shortest) number of edges between *i* and *j***
- ***n*** **is the max number of edges affected**
- **(*k*=0) : linear; (*k*<0) : rigid; (*k*>0) : elastic**



Radial Basis Functions

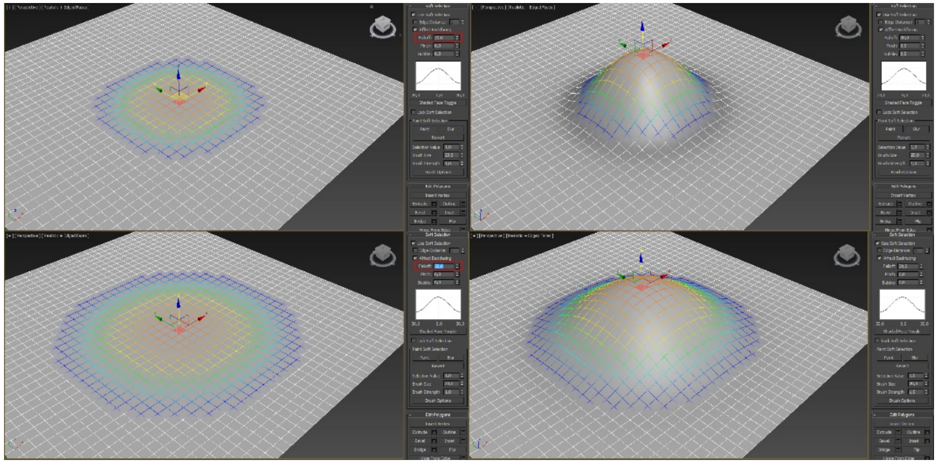
- **A radial basis function (RBF) is a real-valued function Φ whose value depends only on the distance from the origin**





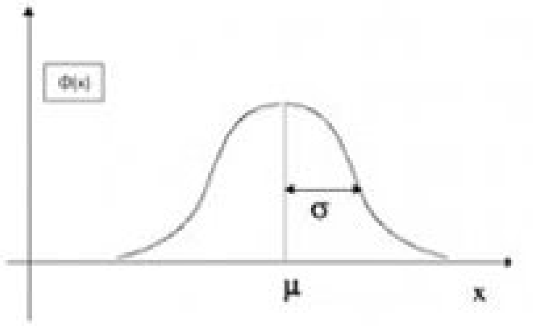
- **Commonly used RBFs**
  - **Gaussian function**

$$\phi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right)$$



RBF: Gaussian kernel function

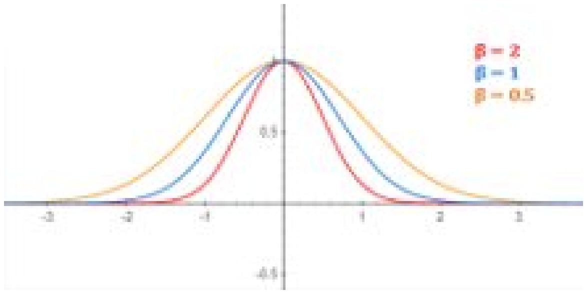
$$f(x) = \quad e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



$$\beta \overset{\text{let}}{=} \frac{1}{2\sigma^2}$$

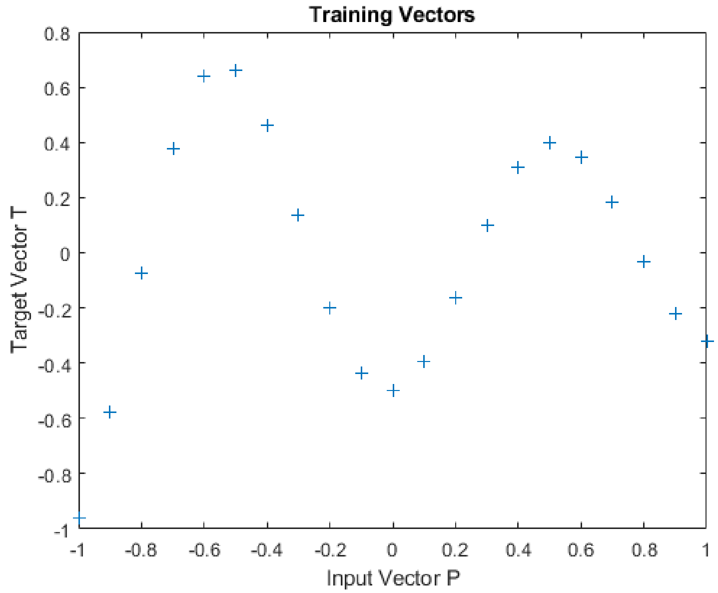
$$\Downarrow$$

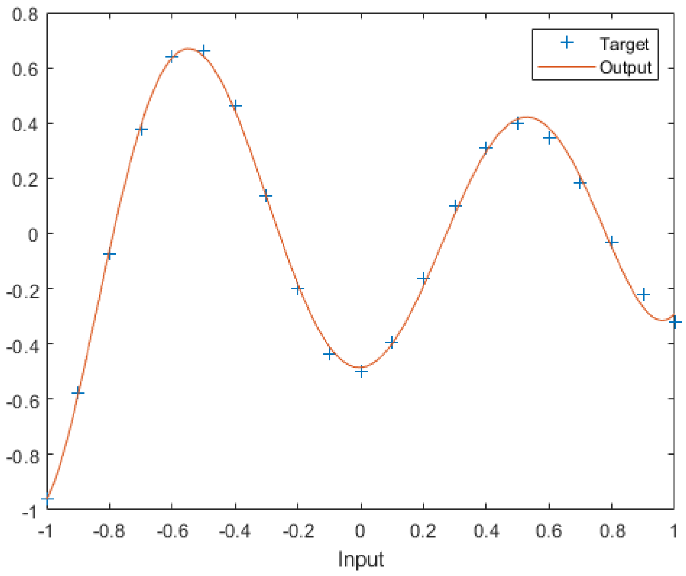
$$\varphi(x) = e^{-\beta\|x-\mu\|^2}$$



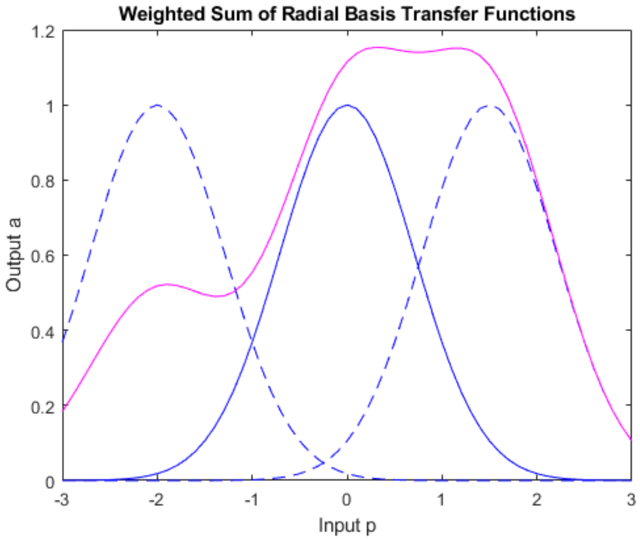
RBF

- Conventionally used for the curve fitting or scattered data interpolation
  - Given a set of sparse set of samples, we want to derive a continuous function that approximates the sample values





RBF interpolation



1. The function is assumed to be a linear combination of several basis functions:

$$f(\mathbf{x}) = \sum_{q=1}^{N} w_q \phi\left(\left\|\mathbf{x} - \mathbf{x}^q\right\|\right)$$

2a. From *N* samples of $f(\cdot)$, we can obtain *N* equations. Note: Each equation is composed of *N* terms.

$$f(\mathbf{x}^p) = \sum_{q=1}^{N} w_q \phi\left(\left\|\mathbf{x}^p - \mathbf{x}^q\right\|\right) = t^p$$

2b. The idea is to find the *N* “weights” w_q so that $f(\cdot)$ does an "exact" interpolation function of the data points.

RBF interpolation

3. Since an RBF is a 1-dimensional function, i.e. function output is a scalar, we need to have one RBF for each dimension: x, y, and z.

$$\begin{bmatrix} \boxed{f_x(\mathbf{x}^1)} \\ \vdots \\ f_x(\mathbf{x}^N) \end{bmatrix} = \begin{bmatrix} \phi_x^{11} & \cdots & \phi_x^{1N} \\ \vdots & \ddots & \vdots \\ \phi_x^{N1} & \cdots & \phi_x^{NN} \end{bmatrix} \begin{bmatrix} w_x^1 \\ \vdots \\ w_x^N \end{bmatrix} = \begin{bmatrix} \boxed{t_x^1} \\ \vdots \\ t_x^N \end{bmatrix}$$

$$\vdots$$

$$\begin{bmatrix} \boxed{f_z(\mathbf{x}^1)} \\ \vdots \\ f_z(\mathbf{x}^N) \end{bmatrix} = \begin{bmatrix} \phi_z^{11} & \cdots & \phi_z^{1N} \\ \vdots & \ddots & \vdots \\ \phi_z^{N1} & \cdots & \phi_z^{NN} \end{bmatrix} \begin{bmatrix} w_z^1 \\ \vdots \\ w_z^N \end{bmatrix} = \begin{bmatrix} \boxed{t_z^1} \\ \vdots \\ t_z^N \end{bmatrix}$$

4a. The weights are the solutions of the linear equations:

$$\sum_{q=1}^{N} \Phi_{pq} w_q = t^p$$

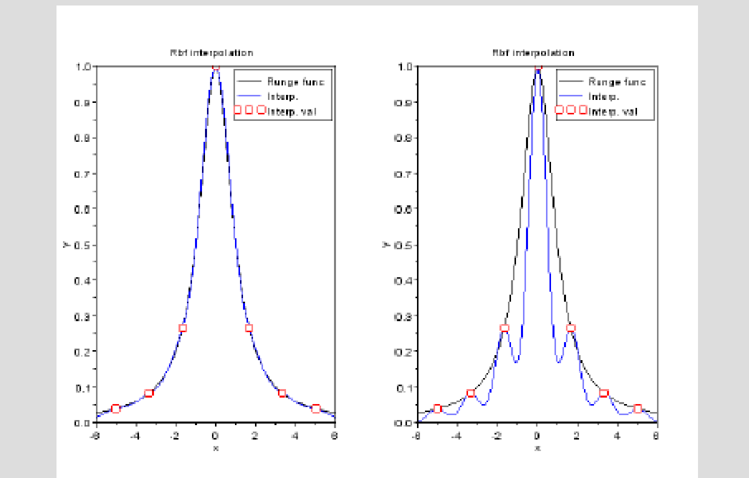
where: $\phi^{pq} = \exp\left(-\frac{\|\mathbf{x}^p - \mathbf{x}^q\|}{2\sigma^2}\right)$.

RBF interpolation

4b. This weight vector can be derived by defining vectors **t** and the square matrix **Φ:**

$$\Rightarrow \boldsymbol{\Phi}\,\mathbf{w} = \mathbf{t} \qquad \mathbf{w} = \boldsymbol{\Phi}^{-1}\,\mathbf{t}$$

NOTE: σ controls the fitting behavior of the approxmiation.



Optimal RBF with $\sigma \approx 0{,}523$ *(left), non optimal solution* $\sigma \approx 0{,}2$ *(right)*

RBF interpolation

- **RBF's have become a well-established tool for interpolation**
  - Produces high-quality meshes
  - Avoids the need for mesh connectivity information
  - The system of equations which needs to be solved is linear
  - The size of the linear system of equation is proportional to the number of nodes, not all vertices

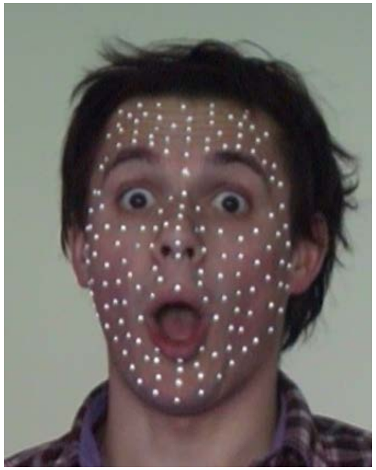
2D Warping by RBF

- **Displacement is computed as a function of distance to anchors**

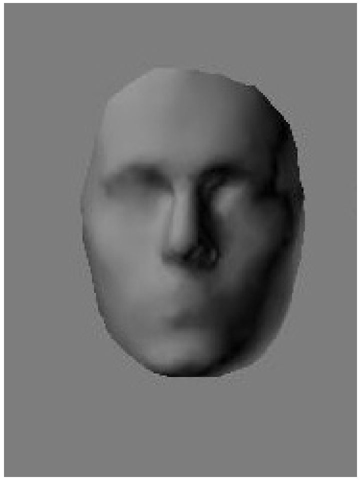
Face warping by RBF(3D case)

- **Given:**
  - **A face shape (a mesh)**
  - **Marker positions (from the mocap or by hand) on it**

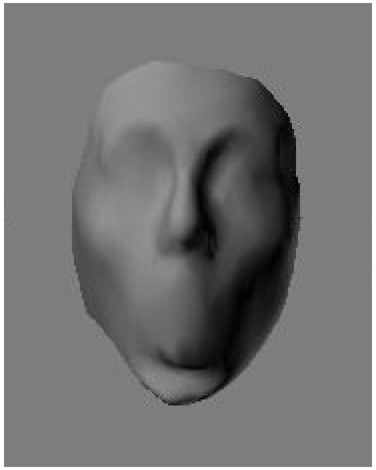
- **What we want to obtain:**
  - **A deformed mesh**



+

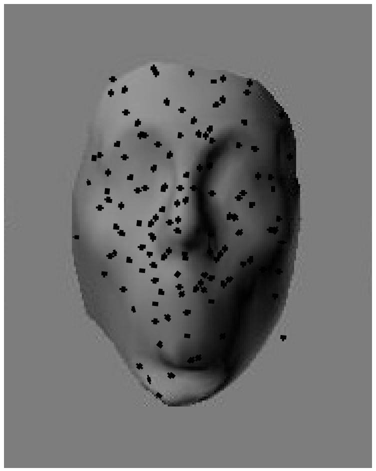


=

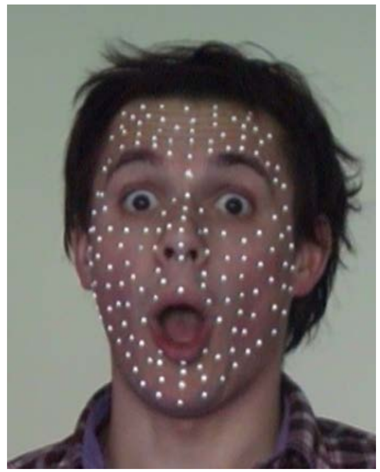


Performance based FA demo: https://www.youtube.com/watch?v=o_7CfWlkqm8

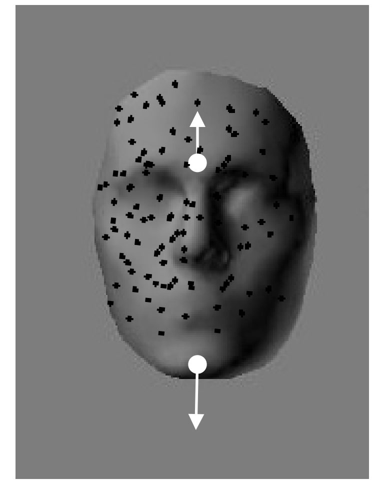
RBF based deformation: We assume..



= F(



) evaluated on



F(X,Y) = (dx, dy)

RBF based deformation: 2D example

X	Y	dX	dY
25	30	5	6
10	28	4	8
.	.	.	.

X	Y
25	30
10	28
.	.

Fx(X,Y)=dX

dX
5
4
.

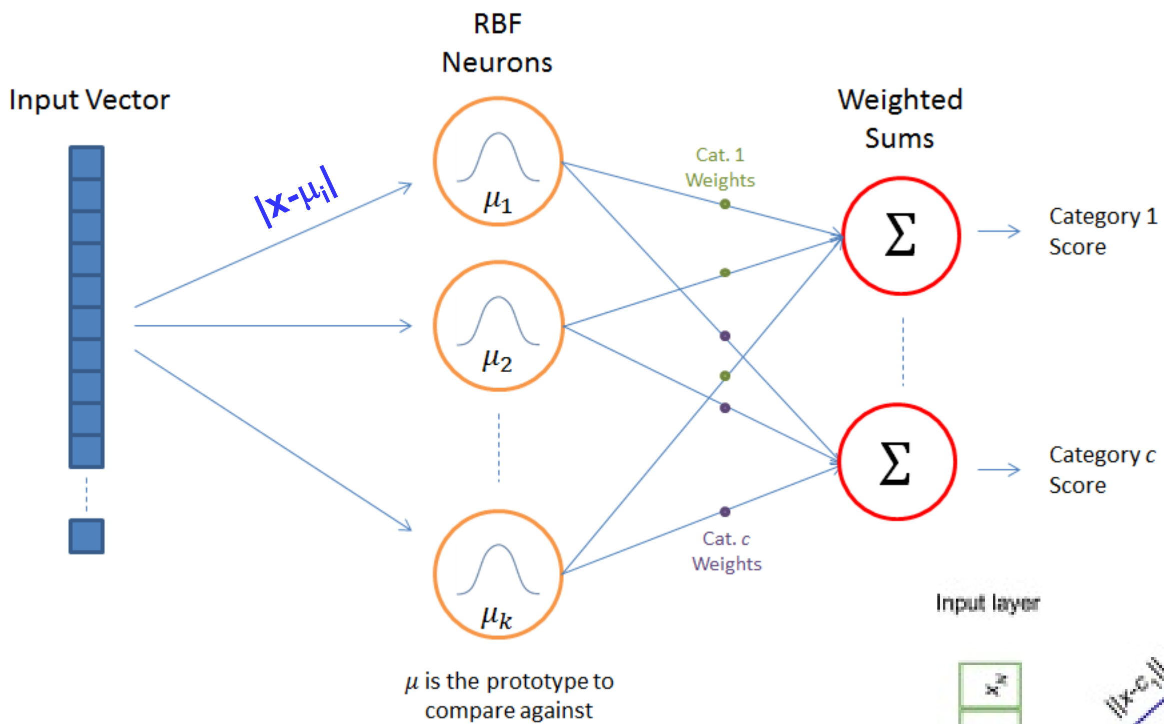
X	Y
25	30
10	28
.	.

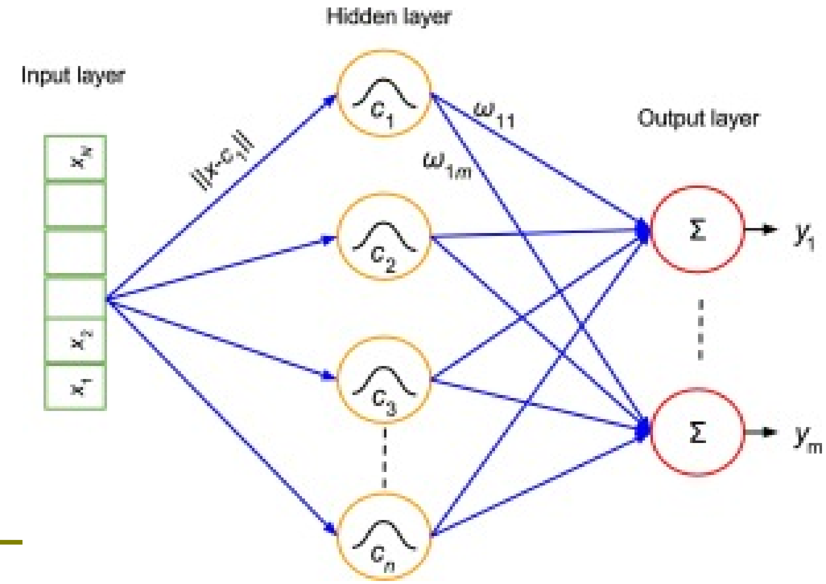
Fy(X,Y)=dY

dY
6
8
.

- **Once the functions are computed, we apply them to all mesh points**

RBF network


RBF Neurons
Input Vector
Weighted Sums
|x-μi|
Cat. 1 Weights
Category 1 Score
Category c Score
Cat. c Weights
μ1
μ2
μk
μ is the prototype to compare against


Hidden layer
Input layer
Output layer
||x-c1||
ω11
ω1m
c1
c2
c3
cn
x1
x2
xk
y1
ym

Summary

- Given example pairs $\{x_i, b_i\}$, $i=1,2,..,N$, find $f: R^d \rightarrow R$ s.t. $f(x_i)=b_i$.
- f takes smooth changes between examples.

  $f(x) = p_m(x) + \Sigma\, w_i \cdot \Phi(|x - x_i|)$

  p_m: low degree polynomial

  Φ: radial basis function

$$\Phi(r) = r^2 \log r$$

$$\Phi(r) = (r^2 + c^2)^{1/2}$$

$$\boxed{\Phi(r) = e^{-\frac{r^2}{2\sigma^2}}}$$

- w_i's are solved by $f(x_i)=b_i$ and (optionally) orthogonally conditions

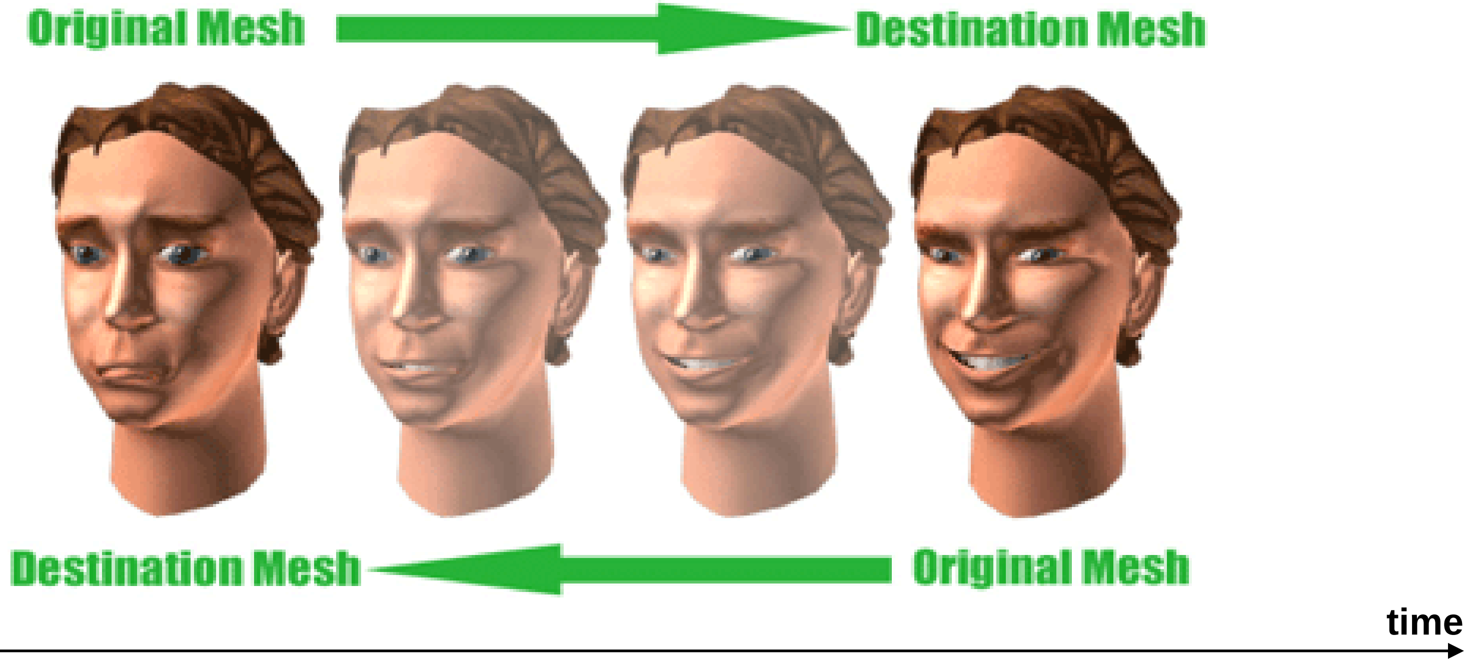
  $\Sigma\, w_i\, p(x_i) = 0$, for all $p \in \pi_m$,

  π_m : the space of all polynomials of degree at most *m*

- Function output is 1-dimensional: $f: R^n \text{-->} R$
  - $f(x) = (f_1(x), f_2(x))$ in case of 2D
  - $f(x) = (f_1(x), f_2(x), f_3(x))$ in case of 3D

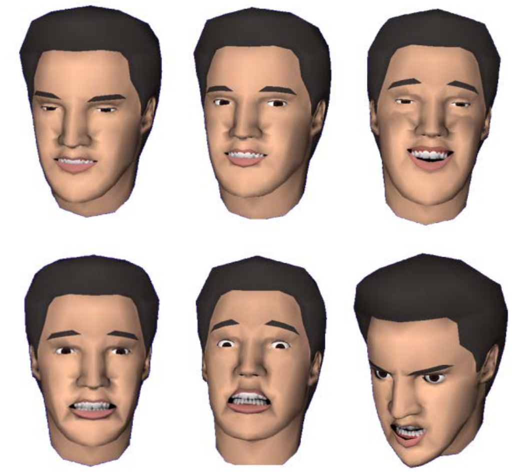
Keyframe based facial animation

- **Temporal interpolation, to generate an animation (sequence of poses)**

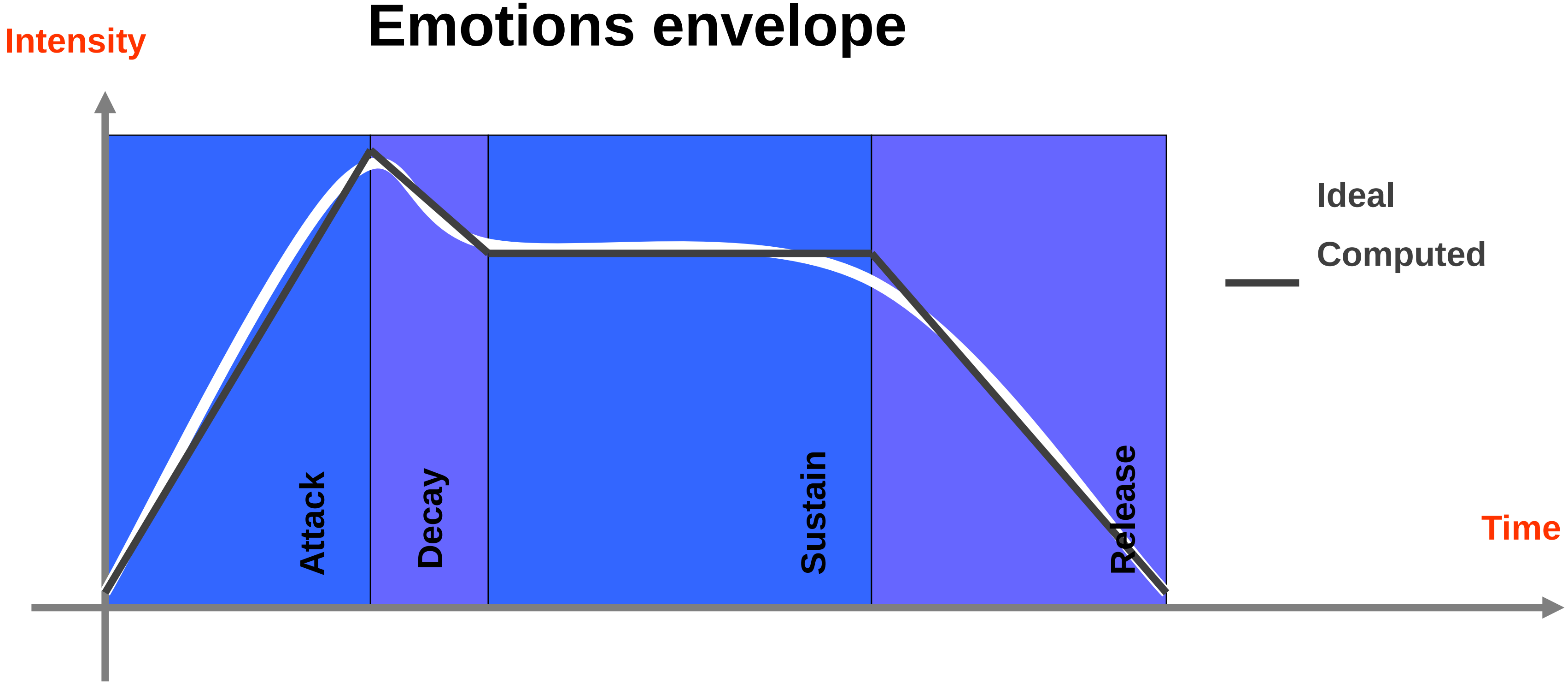


$$\mathbf{P} = (t\text{-}1)\,\mathbf{P}_0 + t\,\mathbf{P}_1 \text{ for } t \in [0,1]$$

Deformation at a high level



Emotions envelope

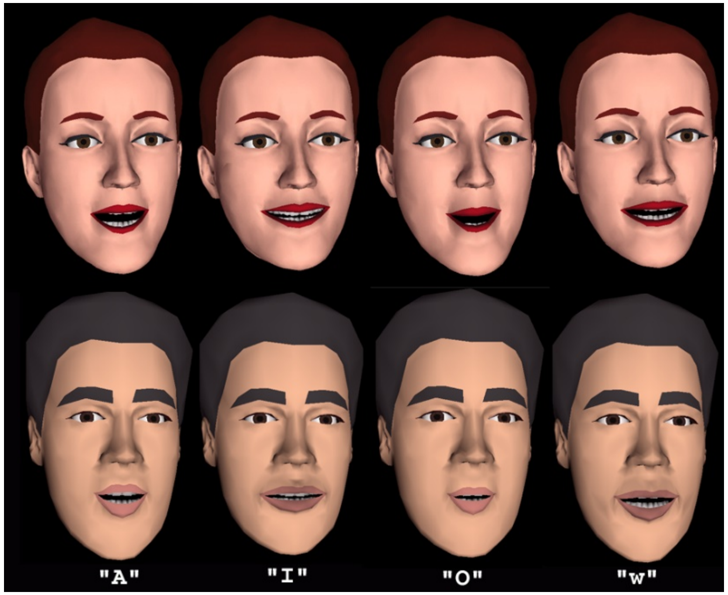


Technical elements for FA

- Region based deformation
- Head/eye movement
- Expression transfer
- Lip synchronization
- Personalized expression

https://www.youtube.com/watch?v=zGqfKY1rjkM
~facial muscle

https://www.youtube.com/watch?v=db_qxsvBaok
~Video as input



https://www.youtube.com/watch?v=vniMsN53ZPI (4:30)

https://www.youtube.com/watch?v=OhgmFyzovHM
~Video as input