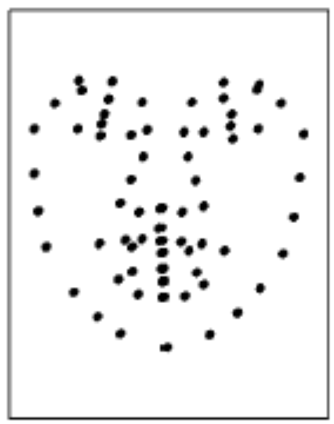
Active Appearances

- The material following is based on
  - T.F. Cootes, G.J. Edwards, and C.J. Taylor, “Active Appearance Models”, Proc. Fifth European Conf. Computer Vision, H. Burkhardt and B. Neumann, eds., vol. 2, pp. 484-498, 1998.
  - T.F. Cootes, G.J. Edwards, and C.J. Taylor, "Active appearance models," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 23, no. 6, pp. 681-- 685, June 2001.
- Authors’ focus was development of method for matching statistical models of appearance to [2D] images
- Applied to faces, 2D medical images
- Basic idea has since been extended to many applications in 2D & 3D medical imaging

Statistical Appearance Models

- Shape
  - In this case, 2D locations of key feature points
- “Texture”
  - I.e., patterns of intensities or colors across image patches
- Method to build: Identify key points; do deformable warp of points to common coordinate system; normalize intensities; read intensities into an intensity vector **G**

Labelled image



Points

Shape-free patch

$$\|\mathbf{G}\| = 1$$

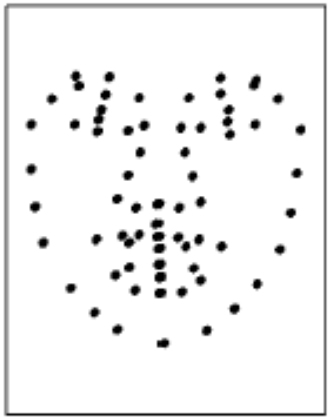
$$\sum \mathbf{G}_k = 0$$

Statistical Appearance Models

How might we do this?

- Shape
  - In this case, 2D locations of key feature points
- "Texture"
  - I.e., patterns of intensities or colors across image patches
- Method to build: Identify key points; do deformable warp of points to common coordinate system; normalize intensities; read intensities into an intensity vector **G**

Labelled image



Points

Shape-free patch

$$\|\mathbf{G}\| = 1$$

$$\sum \mathbf{G}_k = 0$$

Deformable warping from point cloud matches

- One answer might make use of what we learned in programming assignments
  - E.g., Determine some "nominal" location for each landmark point. E.g., pick some reference image or average multiple samples or do something else

$$\vec{\mathbf{x}}_k^{(nom)} = \frac{1}{N}\sum_j \vec{\mathbf{x}}_k^{(j)}$$

  - Then fit Bernstein polynomials to determine distortion.

$$\vec{\mathbf{x}}_k^{(nom)} = \sum_{s,t} \vec{\mathbf{c}}_{s,t} B_s(u_k) B_t(v_k)$$

  - Note: In this case, the coefficients will also parameterize the "shape"

Deformable warping from point cloud matches

- Another answer might use something like “thin plate splines” (e.g. Bookstein)

$$TPS(\vec{\mathbf{v}};\vec{\mathbf{a}},\mathbf{B},\mathbf{C},\mathbf{P}) = \vec{\mathbf{a}} + \mathbf{B} \bullet \vec{\mathbf{v}} + \sum_i \vec{\mathbf{c}}_i U(\left\| \vec{\mathbf{v}} - \vec{\mathbf{p}}_i \right\|)$$

$$\text{where } U(r) = r^2 \log\left(r\right)$$

- Thin plate splines are multidimensional analogues of 1-dimensional spline curves.

Thin Plate Splines Digression

- Some citations (from G. Donato and S. Belongie, "Approximation Methods for Thin Plate Spline Mappings and Principal Warps", 2002; http://www.cs.ucsd.edu/Dienst/UI/2.0/Describe/ncstrl.ucsd_cse/CS2003-0764)

[1] C. T. H. Baker. *The numerical treatment of integral equations*. Oxford: Clarendon Press, 1977.

[2] S. Belongie, J. Malik, and J. Puzicha. Matching shapes. In *Proc. 8th Int'l. Conf. Computer Vision*, volume 1, pages 454–461, July 2001.

[3] F. L. Bookstein. Principal warps: thin-plate splines and decomposition of deformations. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 11(6):567–585, June 1989.

[4] H. Chui and A. Rangarajan. A new algorithm for non-rigid point matching. In *Proc. IEEE Conf. Comput. Vision and Pattern Recognition*, pages 44–51, June 2000.

[5] M.H. Davis, A. Khotanzad, D. Flamig, and S. Harms. A physics-based coordinate transformation for 3-d image matching. *IEEE Trans. Medical Imaging*, 16(3):317–328, June 1997.

[6] F. Girosi, M. Jones, and T. Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7(2):219–269, 1995.

[7] M. J. D. Powell. A thin plate spline method for mapping curves into curves in two dimensions. In *Computational Techniques and Applications (CTAC95)*, Melbourne, Australia, 1995.

[8] A.J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *ICML*, 2000.

[9] G. Wahba. *Spline Models for Observational Data*. SIAM, 1990.

[10] Y. Weiss. Smoothness in layers: Motion segmentation using nonparametric mixture estimation. In *Proc. IEEE Conf. Comput. Vision and Pattern Recognition*, pages 520–526, 1997.

[11] C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference*, pages 682–688, 2001.

M-dimensional Thin Plate Spline Summary

Given

$$TPS(\vec{\mathbf{v}};\vec{\mathbf{a}},\mathbf{B},\mathbf{C},\mathbf{P}) = \vec{\mathbf{a}} + \mathbf{B}\bullet\vec{\mathbf{v}} + \sum_i \vec{\mathbf{c}}_i U(\left\|\vec{\mathbf{v}} - \vec{\mathbf{p}}_i\right\|)$$

where

$$U(r) = r^2 \log\left(r\right)$$

$$\vec{\mathbf{v}} = \left[v_1,\cdots,v_M\right]^T$$

$$\vec{\mathbf{p}}_i = \left[p_1,\cdots,p_M\right]_i^T$$

$$\mathbf{P} = \left[\vec{\mathbf{p}}_1,\cdots,\vec{\mathbf{p}}_N\right]^T$$

$$\mathbf{C} = \left[\vec{\mathbf{c}}_1,\cdots,\vec{\mathbf{c}}_N\right]$$

$$\mathbf{B} = \left[\vec{\mathbf{b}}_1,\cdots,\vec{\mathbf{b}}_M\right]$$

M-dimensional Thin Plate Spline Fitting

Given

$$\mathbf{V} = \left[\vec{\mathbf{v}}_1, \cdots, \vec{\mathbf{v}}_N\right] \quad \mathbf{F} = \left[\vec{\mathbf{f}}_1, \cdots, \vec{\mathbf{f}}_N\right]$$

find $\vec{\mathbf{a}}$, $\mathbf{B}$,$\mathbf{C}$ such that

$$\vec{\mathbf{f}}_i = TPS(\vec{\mathbf{v}}_i;\ \vec{\mathbf{a}}, \mathbf{B}, \mathbf{C}, \mathbf{V})$$

To do this, solve the linear system

$$\begin{bmatrix} \mathbf{K}_{[NxN]} & \vec{\mathbf{1}}_{[N\times 1]} & \mathbf{V} \\ \vec{\mathbf{1}}_{[1\times N]} & 0 & 0 \\ \mathbf{V}^T & 0 & \mathbf{0}_{[M\times M]} \end{bmatrix} \begin{bmatrix} \mathbf{C}^T \\ \vec{\mathbf{a}}^T \\ \mathbf{B}^T \end{bmatrix} = \begin{bmatrix} \mathbf{F}^T \\ 0 \\ \mathbf{0}_{[M\times 1]} \end{bmatrix}$$

where

$$\mathbf{K}_{i,j} = \mathbf{K}_{j,i} = U\left(\left\|\vec{\mathbf{v}}_i - \vec{\mathbf{v}}_j\right\|\right) \quad \text{with } U(r) = r^2 \log r$$

$$\mathbf{K}_{i,j} = \left(\vec{\mathbf{v}}_i - \vec{\mathbf{v}}_j\right) \bullet \left(\vec{\mathbf{v}}_i - \vec{\mathbf{v}}_j\right) \log\left(\sqrt{\left(\vec{\mathbf{v}}_i - \vec{\mathbf{v}}_j\right) \bullet \left(\vec{\mathbf{v}}_i - \vec{\mathbf{v}}_j\right)}\right)$$

TPS 2D case

Given a set of points $\vec{\mathbf{p}}_i = \left[x_i, y_i\right]$ and corresponding points $\vec{\mathbf{p}}_i{}^* = \left[x_i{}^*, y_i{}^*\right]$,

we want to find TPS parameters such that $\vec{\mathbf{p}}_i{}^* = TPS(\vec{\mathbf{p}}_i; \vec{\mathbf{a}}, \mathbf{B}, \mathbf{C}, \mathbf{P})$

To do this, we solve the least squares problem

$$\begin{bmatrix} 0 & \cdots & U_{1,k} & \cdots & U_{1,N} & 1 & x_1 & y_1 \\ \vdots & \ddots & & U_{ij} & & \vdots & \vdots & \vdots \\ \mathrm{U}_{\mathrm{k},1} & \cdots & 0 & \cdots & \mathrm{U}_{\mathrm{k,N}} & 1 & x_k & y_k \\ \vdots & U_{ij} & & \ddots & \vdots & \vdots & \vdots & \vdots \\ U_{N,1} & \cdots & U_{N,k} & \cdots & 0 & 1 & x_N & y_N \\ 1 & \cdots & 1 & \cdots & 1 & 0 & 0 & 0 \\ x_1 & \cdots & x_k & \cdots & x_N & 0 & 0 & 0 \\ y_1 & \cdots & y_k & \cdots & y_N & 0 & 0 & 0 \end{bmatrix} \bullet \begin{bmatrix} \vec{\mathbf{c}}_1 \\ \vdots \\ \vdots \\ \vdots \\ \vec{\mathbf{c}}_N \\ \vec{\mathbf{a}} \\ \vec{\mathbf{b}}_x \\ \vec{\mathbf{b}}_y \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{p}}_1{}^* \\ \vdots \\ \vec{\mathbf{p}}_k{}^* \\ \vdots \\ \vec{\mathbf{p}}_N{}^* \\ \vec{\mathbf{0}} \\ \vec{\mathbf{0}} \\ \vec{\mathbf{0}} \end{bmatrix}$$

where $\mathrm{U}_{\mathrm{i,j}} = \mathrm{U}_{\mathrm{j,i}} = \mathrm{U}(\left\| \vec{\mathbf{p}}_i - \vec{\mathbf{p}}_j \right\|)$

M-dimensional Thin Plate Spline Fitting

Define

$$\mathbf{L}_{[M+N+1\times M+N+1]} = \begin{bmatrix} \mathbf{K}_{[NxN]} & \vec{\mathbf{1}}_{[N\times 1]} & \mathbf{V} \\ \vec{\mathbf{1}}_{[1\times N]} & 0 & 0 \\ \mathbf{V}^T & 0 & \mathbf{0}_{[M\times M]} \end{bmatrix}$$

If there are many points, this matrix may be expensive to invert or even pseudo-invert. There are various methods to deal with this problem. These include

- Use a random sample of the $\vec{\mathbf{v}}_i$ to approximate the solution
- Use a random sample of the basis functions & all data to solve problem in least squares sense
- Use matrix approximation methods

See http://www.cs.ucsd.edu/Dienst/UI/2.0/Describe/ncstrl.ucsd_cse/CS2003-0764

Appearance models, con'd

Appearance model is defined by an instance parameter vector $\vec{\lambda}$, mean shape and texture $\mathbf{X}^{(avg)}$and $\mathbf{G}^{(avg)}$, and variation mode matrices $\mathbf{M}_{\mathbf{X}}$ and $\mathbf{M}_{\mathbf{G}}$. Thus, an instance (j) would be

$$\mathbf{G}^{(j)} = \mathbf{G}^{(avg)} + \mathbf{M}_{\mathbf{G}} \bullet \vec{\lambda}^{(j)} = \mathbf{G}^{(avg)} + \sum_{k=1}^{N_{\mathbf{G}}} \vec{\mathbf{M}}_{\mathbf{G}}^{(k)} \bullet \vec{\lambda}_k^{(j)}$$

$$\mathbf{X}^{(j)} = \mathbf{X}^{(avg)} + \mathbf{M}_{\mathbf{X}} \bullet \vec{\lambda}^{(j)} = \mathbf{X}^{(avg)} + \sum_{k=1}^{N_{\mathbf{X}}} \vec{\mathbf{M}}_{\mathbf{X}}^{(k)} \bullet \vec{\lambda}_k^{(j)}$$

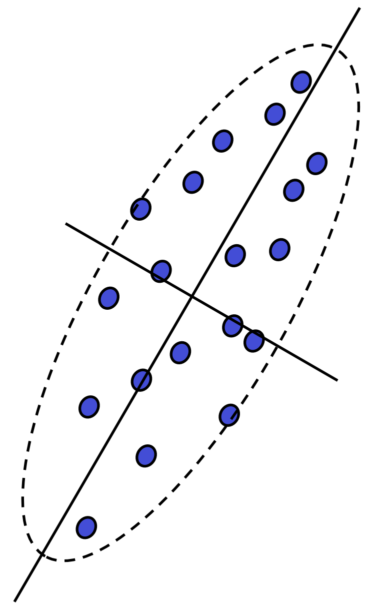
In fact, they created a multi-resolution hierarchy with models similar to the above at different resolutions.

Used PCA to determine the statistical parameters.

Digression: PCA

Suppose that you have a set of N vectors $\vec{\mathbf{a}}_i$ in an M dimensional space?

Is there a natural "coordinate system" for these vectors?



Digression: PCA

We proceed as follows

$$\vec{\mathbf{a}}^{(avg)} = \frac{\sum_i \vec{\mathbf{a}}_i}{N}; \quad \vec{\mathbf{b}}_i = \vec{\mathbf{a}}_i - \vec{\mathbf{a}}^{(avg)}; \quad \mathbf{B} = \left[\vec{\mathbf{b}}_1, \cdots \vec{\mathbf{b}}_N\right];$$

Then form the singular value decomposition

$$\mathbf{B} = \mathbf{U}\Sigma\mathbf{V}^T = \mathbf{U}\begin{bmatrix} \Sigma^{(N)} \\ \mathbf{0} \end{bmatrix}\mathbf{V}^T \quad \text{where } \Sigma^{(N)} = diag(\sigma_1, \cdots, \sigma_N)$$

Then we note that $\mathbf{M} = \mathbf{U}\Sigma^2\mathbf{U}^T$. Of course $\mathbf{U}$ is huge, but we have the following useful fact. We note that

$$\mathbf{B} = \left[\vec{\mathbf{u}}_1, \cdots, \vec{\mathbf{u}}_N, \vec{\mathbf{u}}_{N+1}, \cdots, \vec{\mathbf{u}}_M\right]\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_N \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \end{bmatrix}\mathbf{V}^T = \left[\vec{\mathbf{u}}_1, \cdots, \vec{\mathbf{u}}_N\right]\Sigma^{(N)}\mathbf{V}^T = \mathbf{U}^{(N)}\Sigma^{(N)}\mathbf{V}^T$$

Digression: PCA

This means that any column $\vec{\mathbf{b}}_k$ of **B** may be expressed as a linear combination of the first N columns of **U**

$$\mathbf{B} = \left[\vec{\mathbf{u}}_1, \cdots, \vec{\mathbf{u}}_N\right] \Sigma^{(N)} \mathbf{V}^T = \mathbf{U}^{(N)} \Sigma^{(N)} \mathbf{V}^T$$

$$\vec{\mathbf{b}}_k = \lambda_1^{(k)} \vec{\mathbf{u}}_1 + \cdots + \lambda_N^{(k)} \vec{\mathbf{u}}_N = \mathbf{U}^{(N)} \Lambda^{(k)}$$

where

$$\Lambda^{(k)} = \mathit{transpose}(\mathbf{U}^{(N)}) \vec{\mathbf{b}}_k$$

So

$$\vec{\mathbf{a}}_k = \vec{\mathbf{a}}^{(avg)} + \vec{\mathbf{b}}_k = \vec{\mathbf{a}}^{(avg)} + \lambda_1^{(k)} \vec{\mathbf{u}}_1 + \cdots + \lambda_N^{(k)} \vec{\mathbf{u}}_N$$

But often the last few values of the λ_k are small. If we ignore all but the first D values, we have

$$\vec{\mathbf{a}}_k \approx \vec{\mathbf{a}}^{(avg)} + \lambda_1^{(k)} \vec{\mathbf{u}}_1 + \cdots + \lambda_D^{(k)} \vec{\mathbf{u}}_D$$

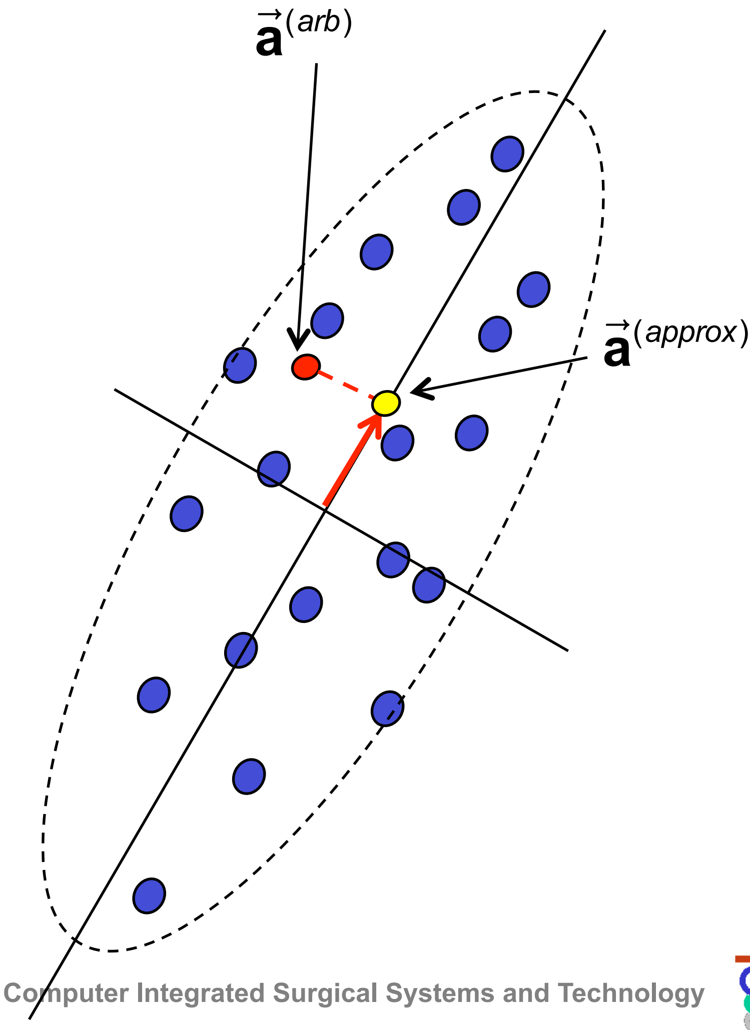
Digression: PCA

Suppose now that we have an arbitrary $\vec{\mathbf{a}}^{(arb)}$. We can approximate $\vec{\mathbf{a}}^{(arb)}$ as follows:

$$\vec{\mathbf{b}}^{(arb)} = \vec{\mathbf{a}}^{(arb)} - \vec{\mathbf{a}}^{(avg)}$$

$$\Lambda^{(arb)} = \mathit{transpose}(\mathbf{U}^{(D)})\vec{\mathbf{b}}^{(arb)}$$

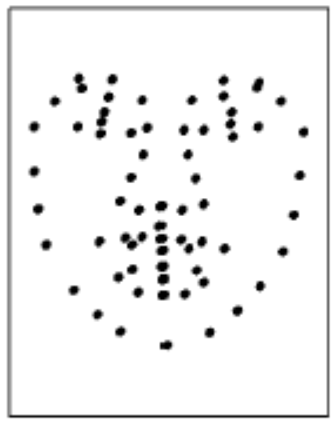
$$\vec{\mathbf{a}}^{(arb)} \approx \vec{\mathbf{a}}^{(avg)} + \lambda_1^{(arb)}\vec{\mathbf{u}}_1 + \cdots + \lambda_D^{(arb)}\vec{\mathbf{u}}_D$$



Training Set for 2001 paper

- 400 faces
- 68 points
- 10000 intensity values

Labelled image



Points

Shape-free patch

Complication

- How do you do PCA if shape and intensity may co-vary?

Answer : Form combined vector of shape and intensity variation

$$\mathbf{Y} = \begin{bmatrix} \mathbf{W}_{\mathbf{x}}\left(\mathbf{X} - \mathbf{X}^{(avg)}\right) \\ \mathbf{G} - \mathbf{G}^{(avg)} \end{bmatrix}$$

where $\mathbf{W}_{\mathbf{x}}$ is a diagonal matrix of weights. Then do PCA on **Y**.

Further complication

- How do you find the right weights to use?

Answer (from Cootes *et al. 1998)*:

The elements of $\mathbf{b}_s$ have units of distance, those of $\mathbf{b}_g$ have units of intensity, so they cannot be compared directly. Because $\mathbf{P}_g$ has orthogonal columns, varying $\mathbf{b}_g$ by one unit moves $\mathbf{g}$ by one unit. To make $\mathbf{b}_s$ and $\mathbf{b}_g$ commensurate, we must estimate the effect of varying $\mathbf{b}_s$ on the sample $\mathbf{g}$. To do this we systematically displace each element of $\mathbf{b}_s$ from its optimum value on each training example, and sample the image given the displaced shape. The RMS change in $\mathbf{g}$ per unit change in shape parameter b_s gives the weight w_s to be applied to that parameter in equation (5).

I.e., do PCA first on shape only and determine an appropriate $\mathbf{V_x}$. Then find an optimal $\vec{\lambda}^{(j)}$ for each training sample (j). Then vary the values of $\vec{\lambda}^{(j,k)}=\vec{\lambda}^{(j)}+\alpha\vec{\mathbf{e}}_k$ to create new shape models $\mathbf{X}^{(j,k)}$and determine the corresponding texture vectors $\mathbf{G}^{(j,k)}$. Then the weight

$$w_k = \sqrt{\frac{1}{N}\sum_j \left\|\mathbf{G}^{(j,k)} - \mathbf{G}^{(j)}\right\|^2} \,/\, \alpha.$$

 600.445 Fall 2004

Face modes

Fig. 2. First two modes of shape variation (±3 sd)

Shape

Fig. 3. First two modes of grey-level variation (±3 sd)

Intensity

Source: Cootes *et al. 1998*

Face modes

Fig. 4. First four modes of appearance variation (±3 sd)

Combined

Source: Cootes *et al. 1998*

Basic Algorithm

- Make an initial guess at model weights
- Create a model from weights
- Evaluate error
- Iteratively improve

Basic Iteration of the Method

1. Project the texture sample into the texture model frame using $\mathbf{g}_s = T_{\mathbf{u}}^{-1}(\mathbf{g}_{im})$.
2. Evaluate the error vector, $\mathbf{r} = \mathbf{g}_s - \mathbf{g}_m$, and the current error, $E = |\mathbf{r}|^2$.
3. Compute the predicted displacements, $\delta\mathbf{p} = -\mathbf{R}\mathbf{r}(\mathbf{p})$.
4. Update the model parameters $\mathbf{p} \rightarrow \mathbf{p} + k\delta\mathbf{p}$, where initially $k = 1$.
5. Calculate the new points, $\mathbf{X}'$ and model frame texture $\mathbf{g}'_m$.
6. Sample the image at the new points to obtain $\mathbf{g}'_{im}$.
7. Calculate a new error vector, $\mathbf{r}' = T_{\mathbf{u}'}^{-1}(\mathbf{g}'_{im}) - \mathbf{g}'_m$.
8. If $|\mathbf{r}'|^2 < E$, then accept the new estimate; otherwise, try at $k = 0.5$, $k = 0.25$, etc.

$$\mathbf{R} = \left(\frac{\partial \mathbf{r}}{\partial \mathbf{p}}^T \frac{\partial \mathbf{r}}{\partial \mathbf{p}}\right)^{-1} \frac{\partial \mathbf{r}}{\partial \mathbf{p}}^T .$$

Source: Cootes *et al. 2001*

Basic Iteration of the Method

1. Project the texture sample into the texture model frame using $\mathbf{g}_s = T_{\mathbf{u}}^{-1}(\mathbf{g}_{im})$.
2. Evaluate the error vector, $\mathbf{r} = \mathbf{g}_s - \mathbf{g}_m$, and the current error, $E = |\mathbf{r}|^2$.
3. Compute the predicted displacements, $\delta\mathbf{p} = -\mathbf{R}\mathbf{r}(\mathbf{p})$.
4. Update the model parameters $\mathbf{p} \rightarrow \mathbf{p} + k\delta\mathbf{p}$, where initially $k = 1$.
5. Calculate the new points, $\mathbf{X}'$ and model frame texture $\mathbf{g}'_m$.
6. Sample the image at the new points to obtain $\mathbf{g}'_{im}$.
7. Calculate a new error vector, $\mathbf{r}' = T_{\mathbf{u}'}^{-1}(\mathbf{g}'_{im}) - \mathbf{g}'_m$.
8. If $|\mathbf{r}'|^2 < E$, then accept the new estimate; otherwise, try at $k = 0.5$, $k = 0.25$, etc.

Note: simple sum of differences. What are some alternatives?

Source: Cootes *et al. 2001*

Results

Fig. 10. Reconstruction (left) and original (right) given original landmark points

Source: Cootes *et al. 1998*

Results

Initial	2 its	8 its	14 its	20 its	converged

Source: Cootes *et al. 1998* **Fig. 11.** Multi-Resolution search from displaced position

Results: Knee Example

- Trained on 30 knee MRI images
- With 42 landmark points

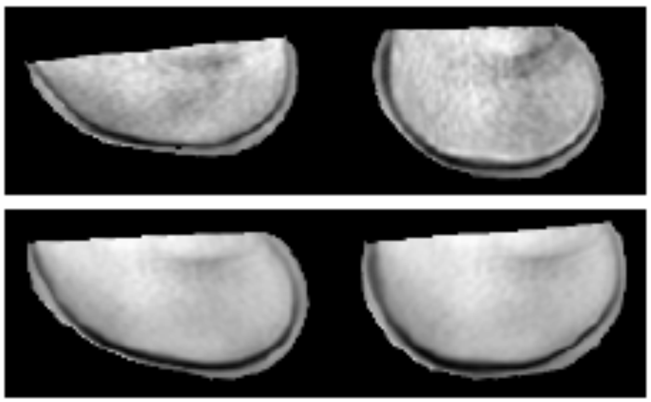


Fig. 12. First two modes of appearance variation of knee model

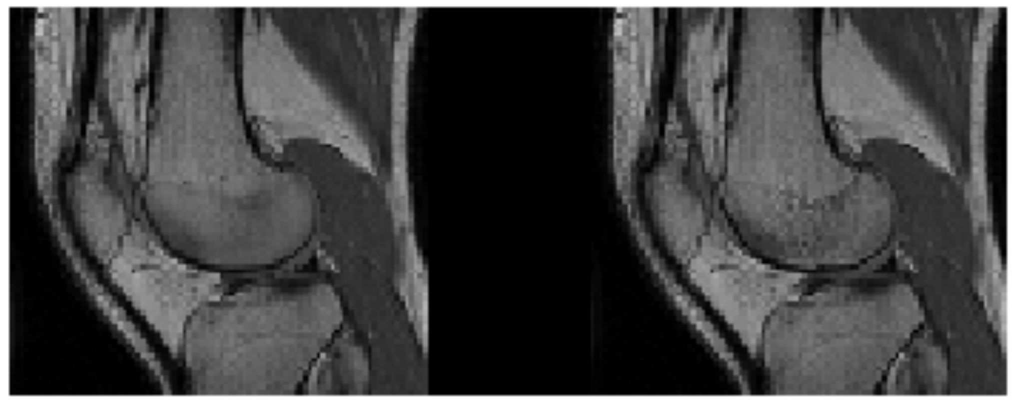


Fig. 13. Best fit of knee model to new image given landmarks

Source: Cootes *et al. 1998*

Results: Knee Example

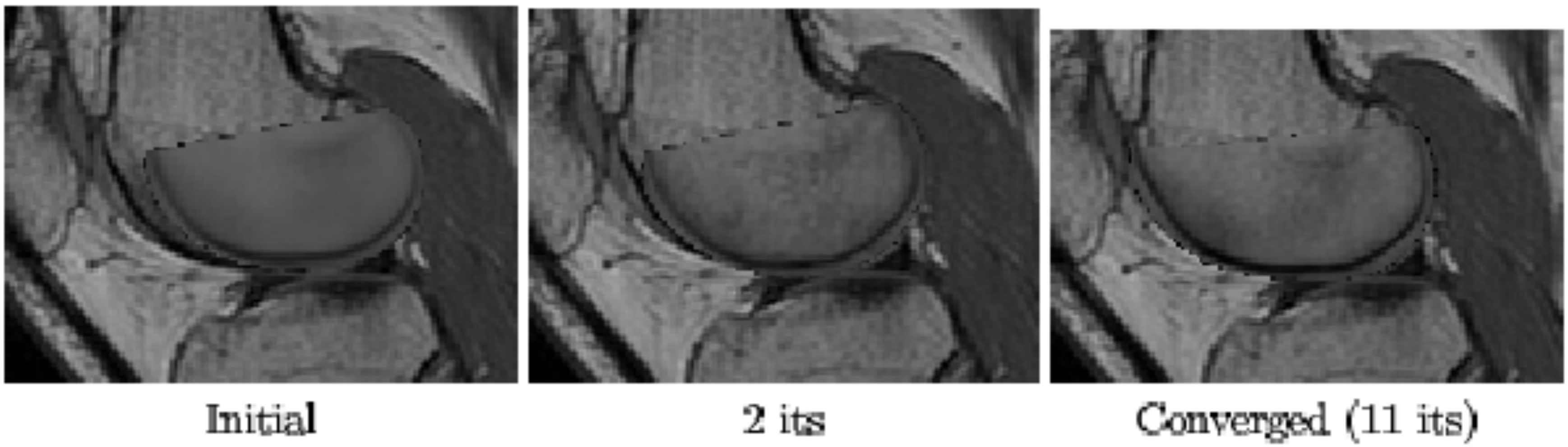


Fig. 14. Multi-Resolution search for knee

Source: Cootes *et al. 1998*