G22.3033-008, Spring 2010

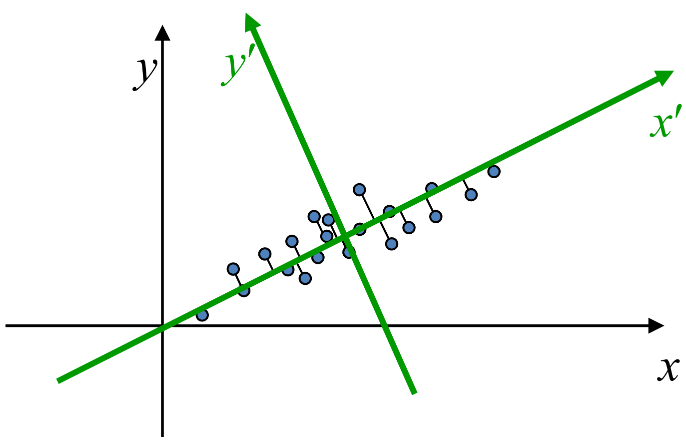
Geometric Modeling

Singular Value Decomposition

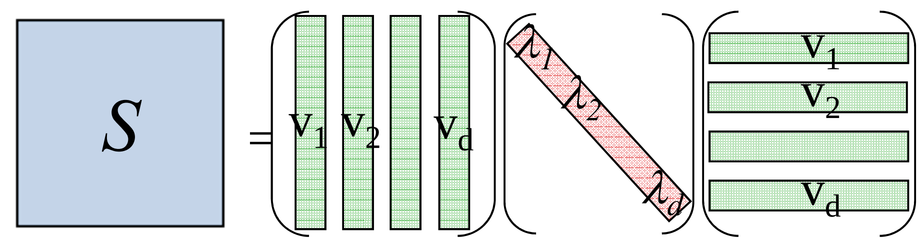
Computing RMSD rigid transform

Reminder: PCA

- Find principal components of data points
- Orthogonal directions that are dominant in the data (have high variance)



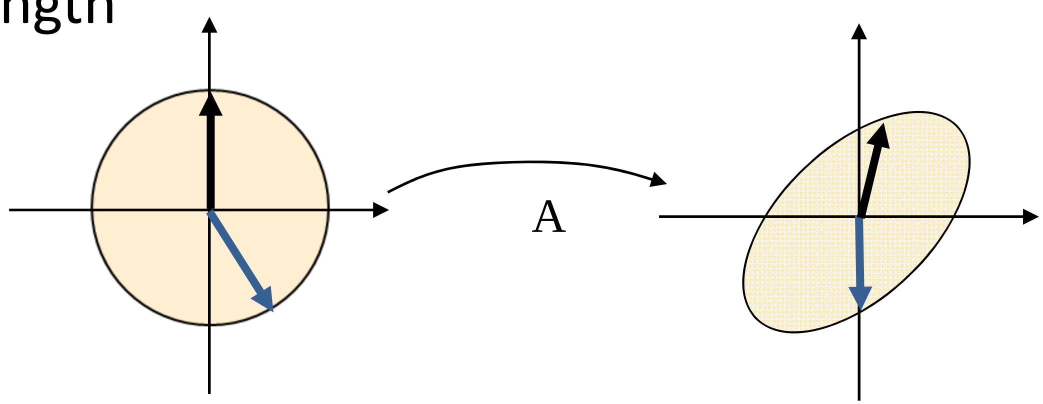
Scatter matrix $S = Y Y^T$



Singular Value Decomposition

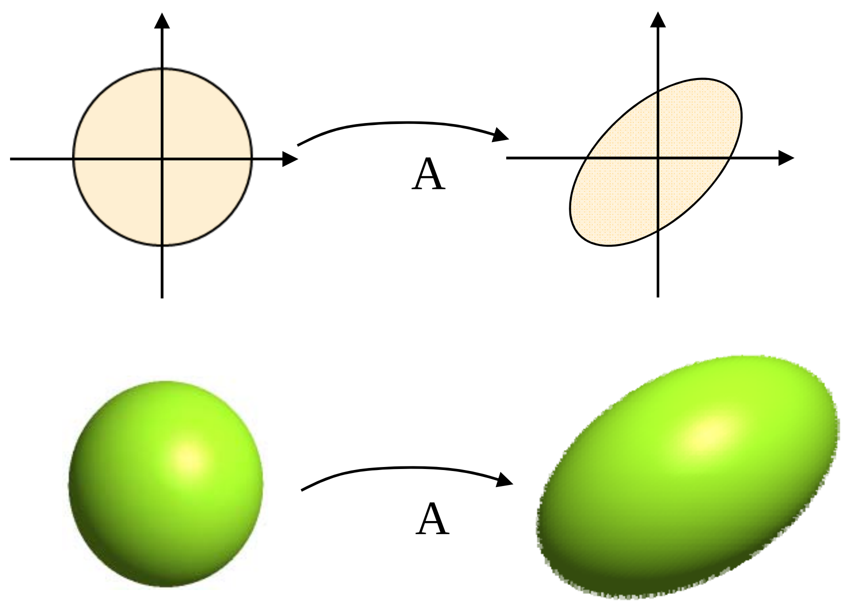
Geometric analysis of linear transformations

- We want to know what a linear transformation A does
- Need some simple and "comprehensible" representation of the matrix A
- Let's look what A does to some vectors
  - Since $\mathrm{A}(\alpha\mathbf{v}) = \alpha\mathrm{A}(\mathbf{v})$, it's enough to look at vectors $\mathbf{v}$ of <u>unit</u> length



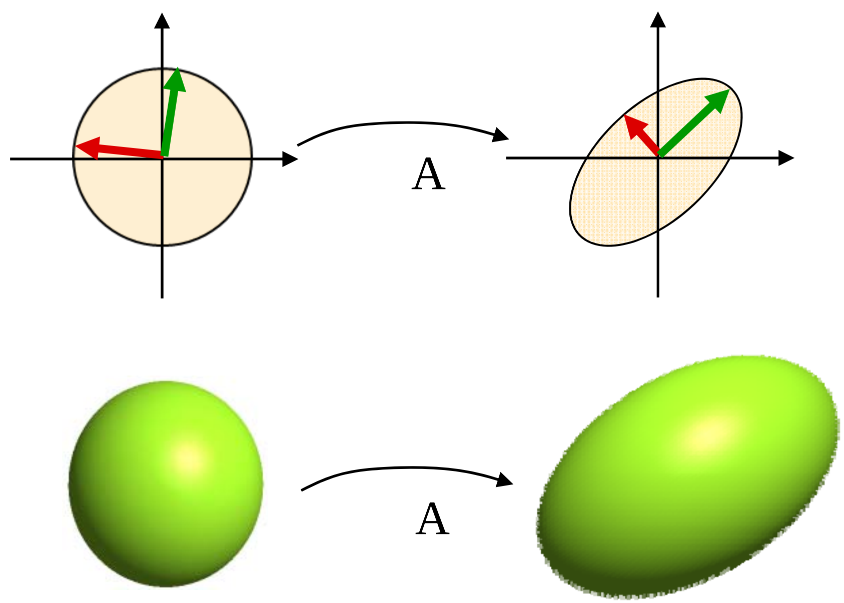
Geometric analysis of linear transformations

- A linear (non-singular) transform A always takes hyper-spheres to hyper-ellipses.



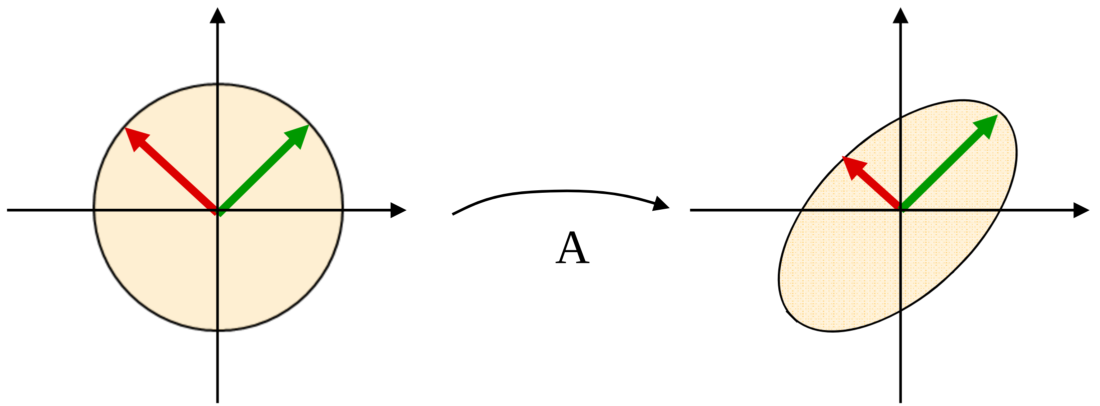
Geometric analysis of linear transformations

- Thus, one good way to understand what A does is to find which vectors are mapped to the "main axes" of the ellipsoid



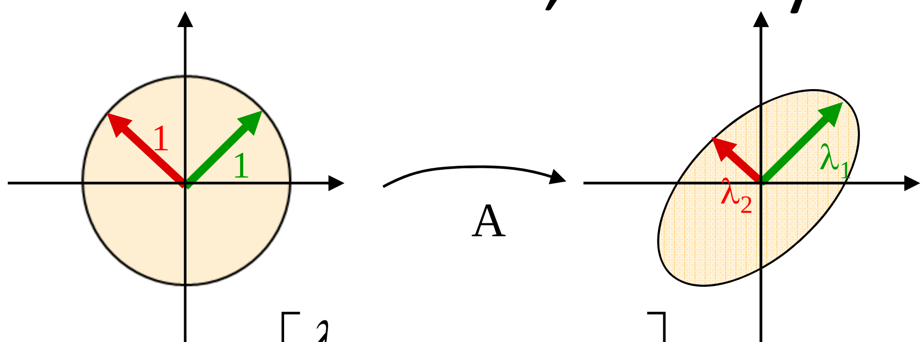
Geometric analysis of linear transformations

- If A is symmetric: $\boxed{\mathrm{A} = \mathrm{V}\,\mathrm{D}\,\mathrm{V}^{\mathrm{T}}, \quad \mathrm{V} \text{ orthogonal}}$
- The eigenvectors of A are the axes of the ellipse



Symmetric matrix: eigendecomposition

- In this case A is just a scaling matrix. The eigendecomposition of A tells us which orthogonal axes it scales, and by how much

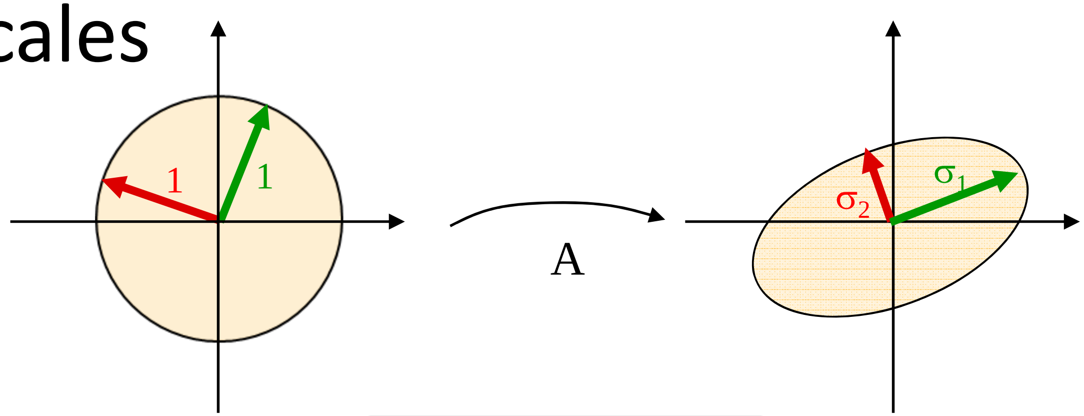


$$A = [\mathbf{v}_1\ \mathbf{v}_2 \dots \mathbf{v}_n] \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix} [\mathbf{v}_1\ \mathbf{v}_2 \dots \mathbf{v}_n]^T$$

$$\boxed{A\mathbf{v}_i = \lambda_i \mathbf{v}_i}$$

General linear transformations: Singular Value Decomposition

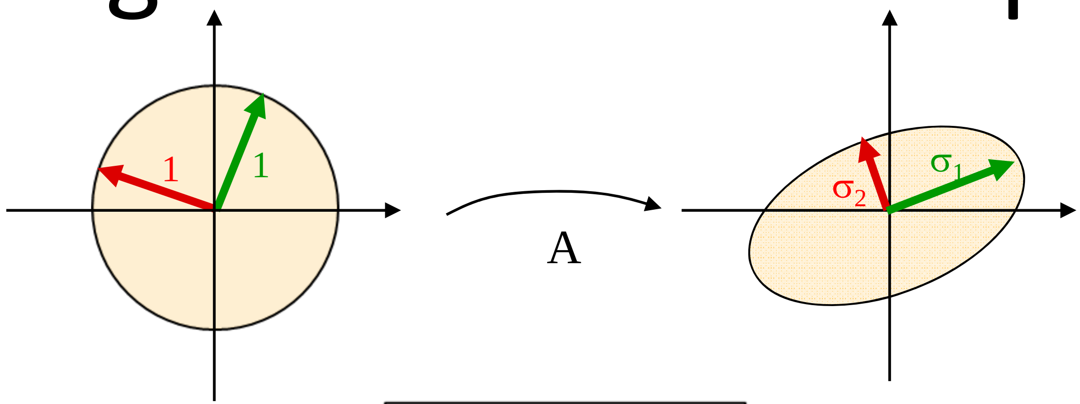
- In general A will also contain rotations, not just scales



$$\boxed{A = U\,\Sigma\,V^{T}}$$

$$A = \begin{bmatrix}\mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_n\end{bmatrix}\begin{bmatrix}\sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n\end{bmatrix}\begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2 & \dots & \mathbf{v}_n\end{bmatrix}^T$$

General linear transformations: Singular Value Decomposition



$$\boxed{A\,V = U\,\Sigma}$$

$$\overset{\textit{orthonormal}}{A\left[\mathbf{v}_1\ \mathbf{v}_2\ \ldots\ \mathbf{v}_n\right]} = \overset{\textit{orthonormal}}{\left[\mathbf{u}_1\ \mathbf{u}_2\ \ldots\ \mathbf{u}_n\right]}\begin{bmatrix}\sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n\end{bmatrix}$$

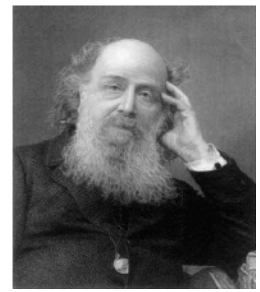
$$\boxed{A\mathbf{v}_i = \sigma_i\,\mathbf{u}_i\,,\quad \sigma_i \geq 0}$$

Some history

- SVD was discovered by the following people:

E. Beltrami
(1835 – 1900)

M. Jordan
(1838 – 1922)



J. Sylvester
(1814 – 1897)

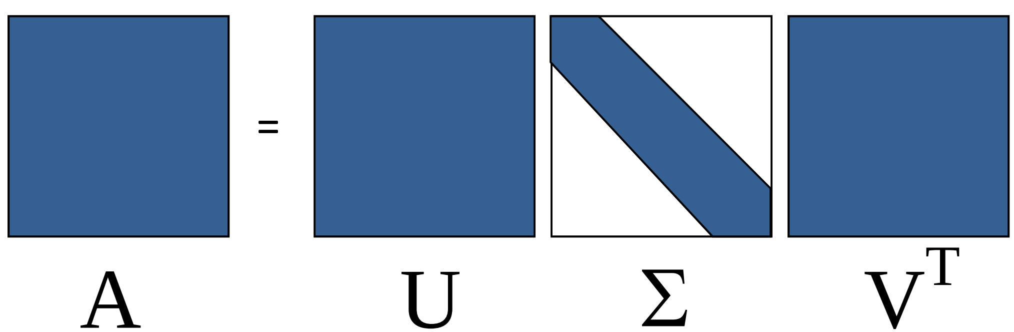
E. Schmidt
(1876-1959)

H. Weyl
(1885-1955)

SVD

- SVD exists for any matrix
- Formal definition:
  - For square matrices $\mathrm{A} \in R^{n \times n}$, there exist orthogonal matrices $\mathrm{U},\ \mathrm{V} \in R^{n \times n}$ and a diagonal matrix Σ, such that all the diagonal values σ_i of Σ are non-negative and

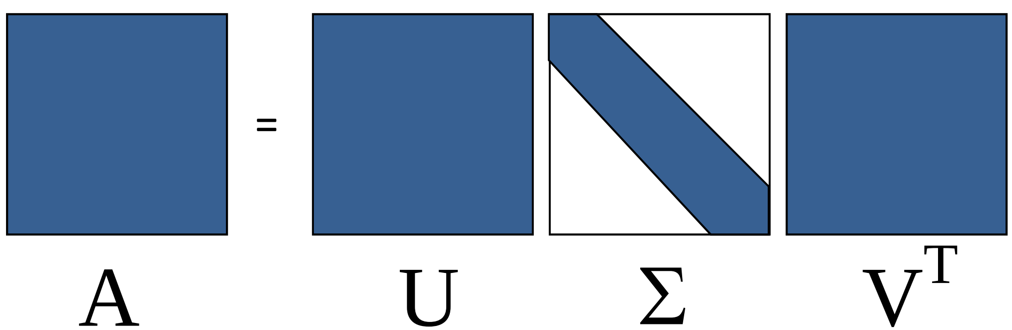
$$\boxed{\mathrm{A} = \mathrm{U}\,\Sigma\,\mathrm{V}^{\mathrm{T}}}$$



SVD

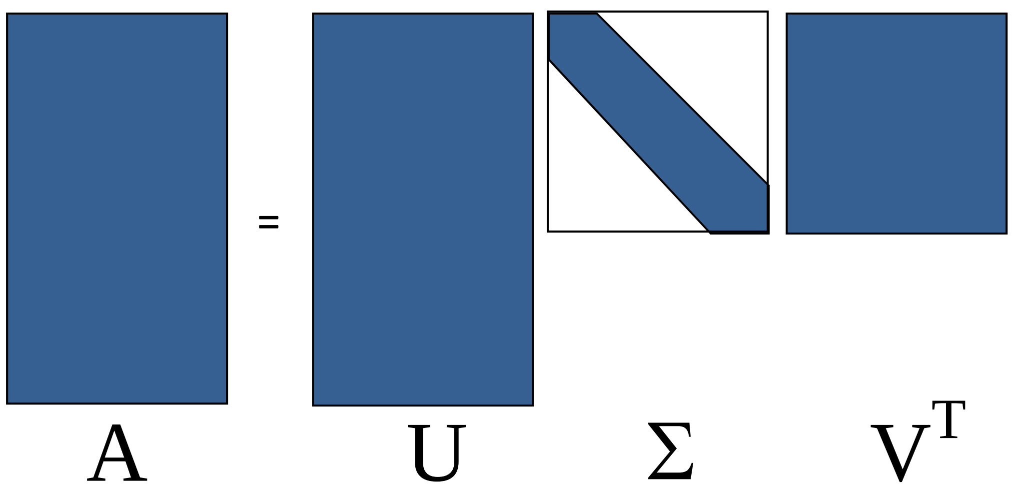
- The diagonal values of Σ are called the singular values. It is accustomed to sort them: $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$
- The columns of U ($\mathbf{u}_1, \ldots, \mathbf{u}_n$) are called the left singular vectors. They are the axes of the ellipsoid.
- The columns of V ($\mathbf{v}_1, \ldots, \mathbf{v}_n$) are called the right singular vectors. They are the preimages of the axes of the ellipsoid.

$$\boxed{A = U \, \Sigma \, V^T}$$



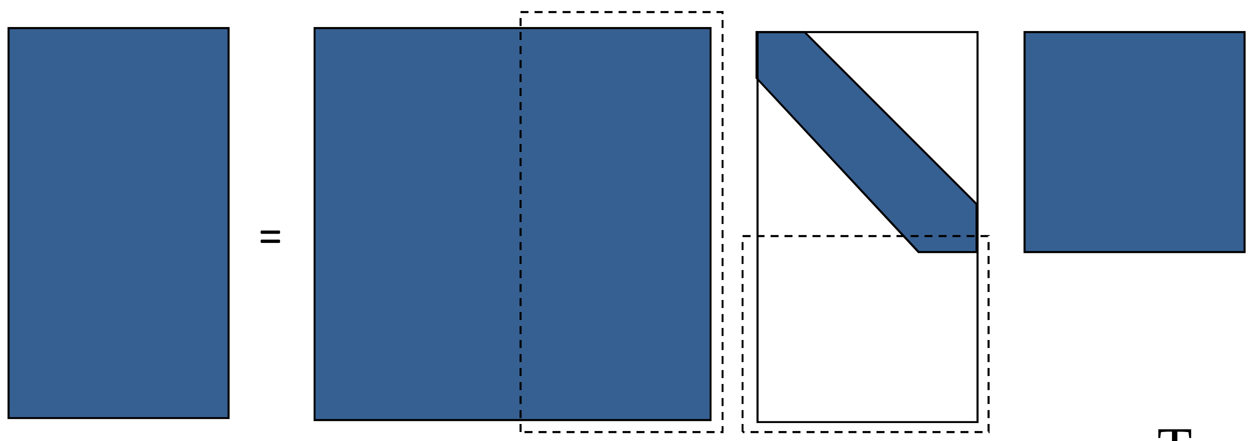
Reduced SVD

- For rectangular matrices, we have two forms of SVD. The reduced SVD looks like this:
  - The columns of U are orthonormal
  - Cheaper form for computation and storage



Full SVD

- We can complete U to a full orthogonal matrix and pad Σ by zeros accordingly



$$A \quad = \quad U \quad \Sigma \quad V^T$$

SVD

Applications

- There are stable numerical algorithms to compute SVD (albeit not cheap). Once you have it, you have many things:
  - Matrix inverse $\rightarrow$ can solve square linear systems
  - Numerical rank of a matrix
  - Can solve linear least-squares systems
  - PCA
  - Many more…

Matrix inverse and solving linear systems

- Matrix inverse

$$A = U\Sigma V^T$$

$$A^{-1} = \left(U\Sigma V^T\right)^{-1} = \left(V^T\right)^{-1}\Sigma^{-1}U^{-1} =$$

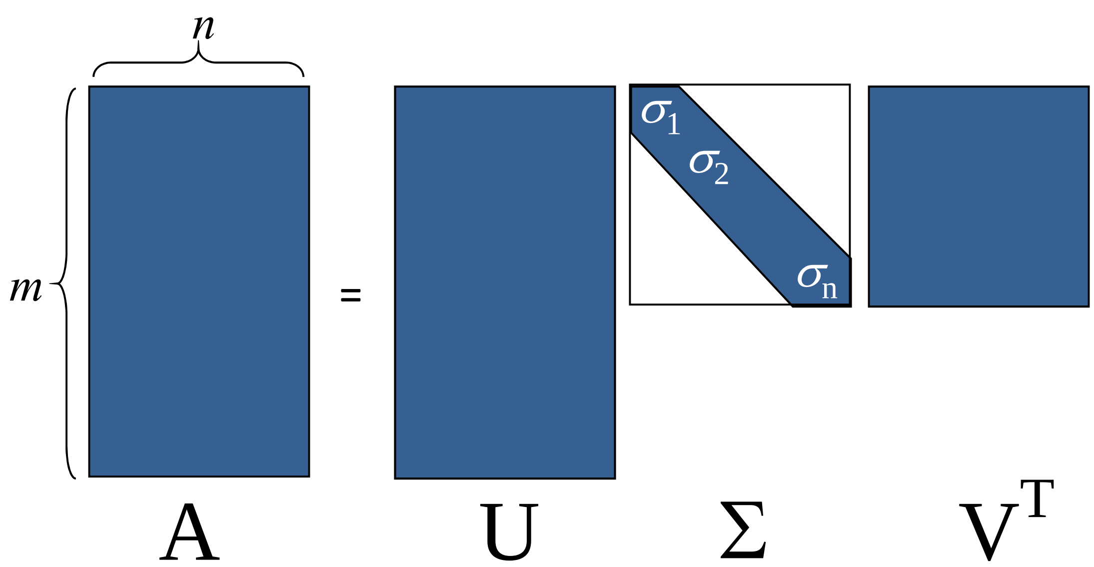
$$= V\begin{pmatrix} \frac{1}{\sigma_1} & & \\ & \ddots & \\ & & \frac{1}{\sigma_n} \end{pmatrix}U^T$$

- So, to solve $A\mathbf{x} = \mathbf{b}$

$$\mathbf{x} = V\Sigma^{-1}U^T\mathbf{b}$$

Matrix rank

- The rank of A is the number of non-zero singular values



Numerical rank

- If there are very small singular values, then A is close to being singular. We can set a threshold t, so that
  $\text{numeric\_rank}(\mathrm{A}) = \#\{\sigma_i \mid \sigma_i > \mathrm{t}\}$
- Using SVD is a numerically stable way! The determinant is not a good way to check singularity

PCA

- Construct the matrix X of the centered data points

$$\mathrm{X} = \begin{pmatrix} | & | & & | \\ \mathbf{p}_1' & \mathbf{p}_2' & \cdots & \mathbf{p}_n' \\ | & | & & | \end{pmatrix}$$

- The principal axes are eigenvectors of $\mathrm{S} = \mathrm{X}\mathrm{X}^{\mathrm{T}}$

$$\mathrm{S} = \mathrm{X}\mathrm{X}^{\mathrm{T}} = \mathrm{U}\begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_d \end{pmatrix}\mathrm{U}^{\mathrm{T}}$$

PCA

- We can compute the principal components by SVD of X:

$$X = U\Sigma V^T$$

$$XX^T = U\Sigma V^T(U\Sigma V^T)^T =$$

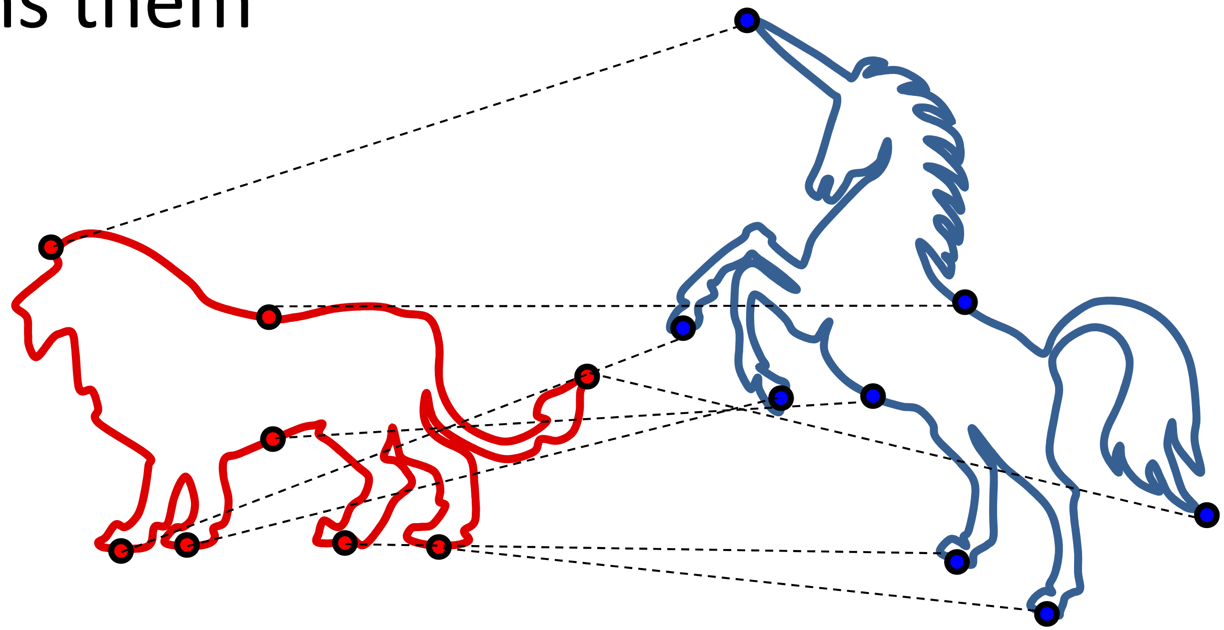
$$= U\Sigma V^T V\Sigma U^T = U\underline{\Sigma}^2 U^T$$

- Thus, the left singular vectors of X are the principal components! We sort them by the size of the singular values of X.

Least-squares rotation with SVD

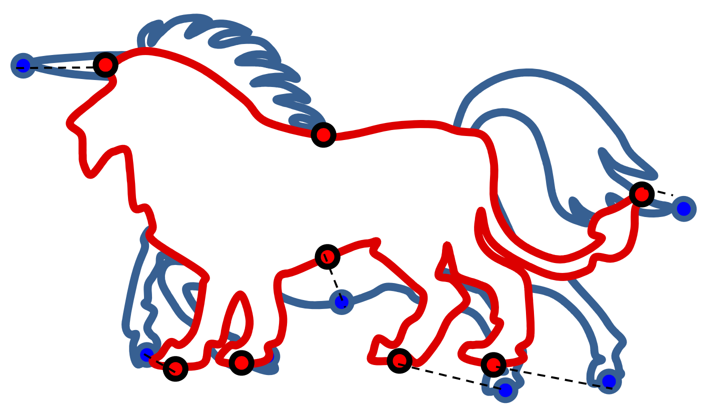
Shape matching

- We have two objects in correspondence
- Want to find the rigid transformation that aligns them



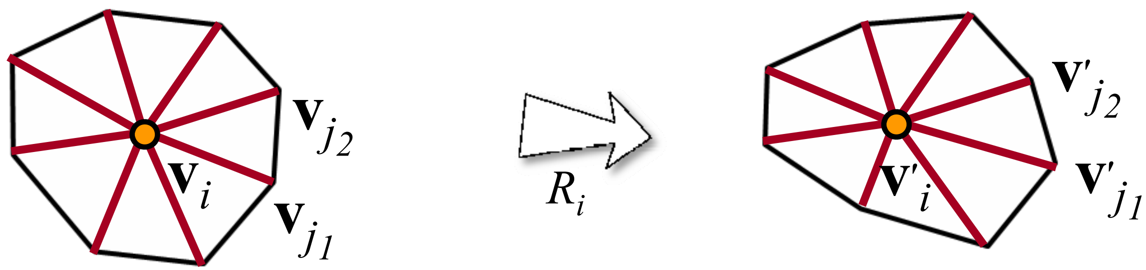
Shape matching

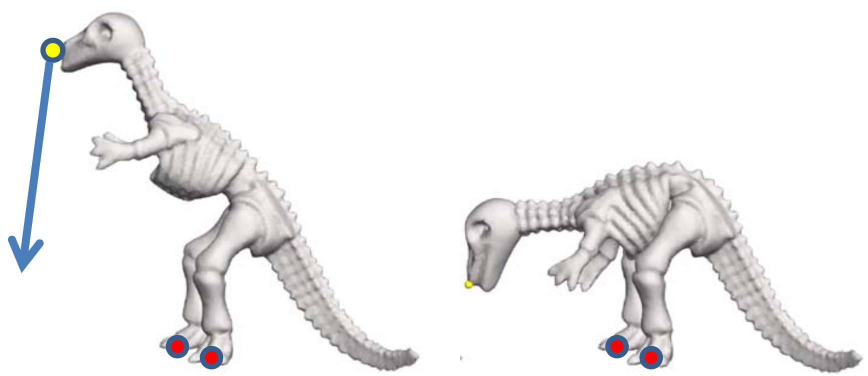
- When the objects are aligned, the lengths of the connecting lines are small



Optimal local rotation

- We will use this for mesh deformation





Shape matching – formalization

- Align two point sets

$$P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\} \text{ and } Q = \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}.$$

- Find a translation vector $\mathbf{t}$ and rotation matrix R so that

$$\sum_{i=1}^{n} \left\| \left(\mathrm{R}\mathbf{p}_i + \mathbf{t}\right) - \mathbf{q}_i \right\|^2 \quad \text{is minimized}$$

Shape matching – solution

- Solve translation and rotation separately
  - If $(\mathrm{R}, \mathbf{t})$ is the optimal transformation, then the point sets $\{\mathrm{R}\mathbf{p}_i + \mathbf{t}\}$ and $\{\mathbf{q}_i\}$ have the same centers of mass

$$\overline{\mathbf{p}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{p}_i \qquad \overline{\mathbf{q}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{q}_i$$

$$\overline{\mathbf{q}} = \frac{1}{n}\sum_{i=1}^{n}\left(\mathrm{R}\mathbf{p}_i + \mathbf{t}\right) = \mathrm{R}\left(\frac{1}{n}\sum_{i=1}^{n}\mathbf{p}_i\right) + \mathbf{t} = \mathrm{R}\overline{\mathbf{p}} + \mathbf{t}$$

$$\Downarrow$$

$$\boxed{\mathbf{t} = \overline{\mathbf{q}} - \mathrm{R}\overline{\mathbf{p}}}$$

Finding the rotation R

- To find the optimal R, we bring the centroids of both point sets to the origin

$$\mathbf{x}_i = \mathbf{p}_i - \overline{\mathbf{p}} \qquad \mathbf{y}_i = \mathbf{q}_i - \overline{\mathbf{q}}$$

- We want to find R that minimizes

$$\sum_{i=1}^{n} \left\| \mathrm{R}\mathbf{x}_i - \mathbf{y}_i \right\|^2$$

Finding the rotation R

$$\sum_{i=1}^{n} \left\| \mathrm{R}\mathbf{x}_i - \mathbf{y}_i \right\|^2 = \sum_{i=1}^{n} \left(\mathrm{R}\mathbf{x}_i - \mathbf{y}_i \right)^{\mathrm{T}} \left(\mathrm{R}\mathbf{x}_i - \mathbf{y}_i \right) =$$

$$= \sum_{i=1}^{n} \left(\mathbf{x}_i^{\mathrm{T}} \underbrace{\mathrm{R}^{\mathrm{T}}\mathrm{R}}_{\mathrm{I}} \mathbf{x}_i - \mathbf{y}_i^{\mathrm{T}} \mathrm{R}\mathbf{x}_i - \mathbf{x}_i^{\mathrm{T}} \mathrm{R}^{\mathrm{T}} \mathbf{y}_i + \mathbf{y}_i^{\mathrm{T}} \mathbf{y}_i \right)$$

These terms do not depend on R,
so we can ignore them in the minimization

Finding the rotation R

$$\min_{\mathrm{R}} \sum_{i=1}^{n} \left(-\mathbf{y}_i^{\mathrm{T}} \mathrm{R} \mathbf{x}_i - \mathbf{x}_i^{\mathrm{T}} \mathrm{R}^{\mathrm{T}} \mathbf{y}_i \right) = \max_{\mathrm{R}} \sum_{i=1}^{n} \left(\mathbf{y}_i^{\mathrm{T}} \mathrm{R} \mathbf{x}_i + \underbrace{\mathbf{x}_i^{\mathrm{T}} \mathrm{R}^{\mathrm{T}} \mathbf{y}_i}_{\text{this is a scalar}} \right)$$

$$\mathbf{x}_i^{\mathrm{T}} \mathrm{R}^{\mathrm{T}} \mathbf{y}_i = \left(\mathbf{x}_i^{\mathrm{T}} \mathrm{R}^{\mathrm{T}} \mathbf{y}_i \right)^{\mathrm{T}} = \mathbf{y}_i^{\mathrm{T}} \mathrm{R} \mathbf{x}_i$$

$$\Rightarrow \boxed{\operatorname*{argmax}_{\mathrm{R}} \sum_{i=1}^{n} \mathbf{y}_i^{\mathrm{T}} \mathrm{R} \mathbf{x}_i}$$

Finding the rotation R

$$\sum_{i=1}^{n} \mathbf{y}_i^{\mathrm{T}} \mathrm{R} \mathbf{x}_i = \mathrm{tr}\left(\mathrm{Y}^{\mathrm{T}} \mathrm{R} \mathrm{X}\right)$$

$$\boxed{\mathrm{tr}(\mathrm{A}) = \sum_{i=1}^{n} \mathrm{A}_{ii}}$$

$$\underset{\mathrm{Y}^{\mathrm{T}}}{\begin{bmatrix} - \mathbf{y}_1^{\mathrm{T}} - \\ - \mathbf{y}_2^{\mathrm{T}} - \\ \vdots \\ - \mathbf{y}_n^{\mathrm{T}} - \end{bmatrix}} \begin{bmatrix} \mathrm{R} \end{bmatrix} \underset{\mathrm{X}}{\begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \\ | & | & & | \end{bmatrix}} = \begin{bmatrix} - \mathbf{y}_1^{\mathrm{T}} - \\ - \mathbf{y}_2^{\mathrm{T}} - \\ \vdots \\ - \mathbf{y}_n^{\mathrm{T}} - \end{bmatrix} \begin{bmatrix} | & | & & | \\ \mathrm{R}\mathbf{x}_1 & \mathrm{R}\mathbf{x}_2 & \cdots & \mathrm{R}\mathbf{x}_n \\ | & | & & | \end{bmatrix}$$

Finding the rotation R

$$\sum_{i=1}^{n} \mathbf{y}_i^{\mathrm{T}} \mathrm{R} \mathbf{x}_i = \mathrm{tr}\left(\mathrm{Y}^{\mathrm{T}} \mathrm{R} \mathrm{X}\right)$$

$$\boxed{\mathrm{tr}(\mathrm{A}) = \sum_{i=1}^{n} \mathrm{A}_{ii}}$$

$$\begin{bmatrix} - \mathbf{y}_1^{\mathrm{T}} - \\ - \mathbf{y}_2^{\mathrm{T}} - \\ \vdots \\ - \mathbf{y}_n^{\mathrm{T}} - \end{bmatrix} \begin{bmatrix} | & | & & | \\ \mathrm{R}\mathbf{x}_1 & \mathrm{R}\mathbf{x}_2 & \cdots & \mathrm{R}\mathbf{x}_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} \mathbf{y}_1^{\mathrm{T}}\mathrm{R}\mathbf{x}_1 & & & \\ & \mathbf{y}_2^{\mathrm{T}}\mathrm{R}\mathbf{x}_2 & & \\ & & \ddots & \\ & & & \mathbf{y}_n^{\mathrm{T}}\mathrm{R}\mathbf{x}_n \end{bmatrix}$$

Finding the rotation R

- Find R that maximizes

$$\mathrm{tr}\left(\mathrm{Y}^{\mathrm{T}}\mathrm{R}\mathrm{X}\right) = \mathrm{tr}\left(\mathrm{R}\mathrm{X}\mathrm{Y}^{\mathrm{T}}\right) \quad (\text{because } \mathrm{tr}(\mathrm{AB}) = \mathrm{tr}(\mathrm{BA}))$$

- Let's do SVD on $\mathrm{S} = \mathrm{X}\mathrm{Y}^{\mathrm{T}}$

$$\mathrm{S} = \mathrm{X}\mathrm{Y}^{\mathrm{T}} = \mathrm{U}\Sigma\mathrm{V}^{\mathrm{T}}$$

$$\Downarrow$$

$$\mathrm{tr}\left(\mathrm{R}\mathrm{X}\mathrm{Y}^{\mathrm{T}}\right) = \mathrm{tr}\left(\underbrace{\mathrm{R}\mathrm{U}}\,\underbrace{\Sigma\mathrm{V}^{\mathrm{T}}}\right) = \mathrm{tr}\left(\Sigma\left(\underbrace{\mathrm{V}^{\mathrm{T}}\mathrm{R}\mathrm{U}}_{\text{orthogonal matrix}}\right)\right)$$

Finding the rotation R

- We want to maximize

$$\mathrm{tr}\left(\Sigma\left(\underbrace{V^{T} R U}_{\substack{\text{orthogonal matrix} \\ \text{all entries} \leq 1}}\right)\right)$$

$$\begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \sigma_3 \end{bmatrix} \begin{bmatrix} m_{11} & \cdots & \\ \vdots & m_{22} & \vdots \\ & \cdots & m_{33} \end{bmatrix}$$

$$\mathrm{tr}\left(\Sigma\left(V^{T} R U\right)\right) = \sum_{i=1}^{3} \sigma_i \, m_{ii} \leq \sum_{i=1}^{3} \sigma_i$$

Finding the rotation R

$$\mathrm{tr}\left(\Sigma\left(\mathrm{V}^{\mathrm{T}}\mathrm{RU}\right)\right) = \sum_{i=1}^{3} \sigma_i \, \mathrm{m}_{ii} \le \sum_{i=1}^{3} \sigma_i$$

- Our best shot is $\mathrm{m}_{ii} = 1$, i.e. to make $\mathrm{V}^{\mathrm{T}}\mathrm{RU} = \mathrm{I}$

$$\mathrm{V}^{\mathrm{T}}\mathrm{RU} = \mathrm{I}$$

$$\mathrm{RU} = \mathrm{V}$$

$$\boxed{\mathrm{R} = \mathrm{VU}^{\mathrm{T}}}$$

Summary of rigid alignment

- Translate the input points to the centroids

$$\mathbf{x}_i = \mathbf{p}_i - \overline{\mathbf{p}} \qquad \mathbf{y}_i = \mathbf{q}_i - \overline{\mathbf{q}}$$

- Compute the "covariance matrix"

$$\mathrm{S} = \mathrm{X}\mathrm{Y}^{\mathrm{T}} = \sum_{i=1}^{n} \mathbf{x}_i \mathbf{y}_i^{\mathrm{T}}$$

- Compute the SVD of S

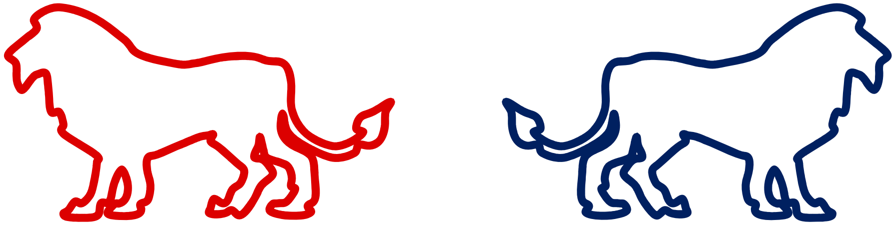
$$\mathrm{S} = \mathrm{U}\Sigma\mathrm{V}^{\mathrm{T}}$$

- The optimal orthogonal R is

$$\mathrm{R} = \mathrm{V}\mathrm{U}^{\mathrm{T}}$$

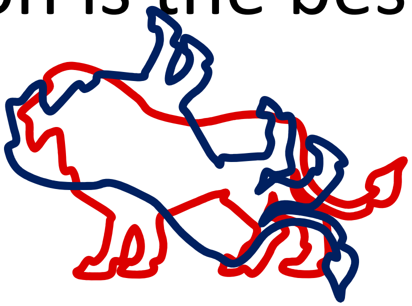
Sign correction

- It is possible that $\det(VU^T) = -1$: sometimes reflection is the best orthogonal transform



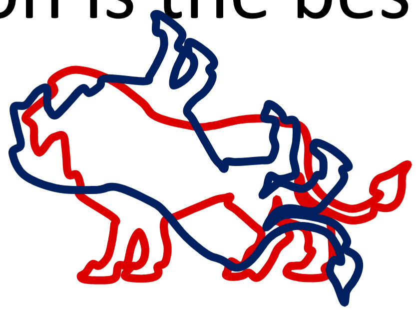
Sign correction

- It is possible that $\det(VU^T) = -1$: sometimes reflection is the best orthogonal transform



Sign correction

- It is possible that $\det(VU^T) = -1$: sometimes reflection is the best orthogonal transform

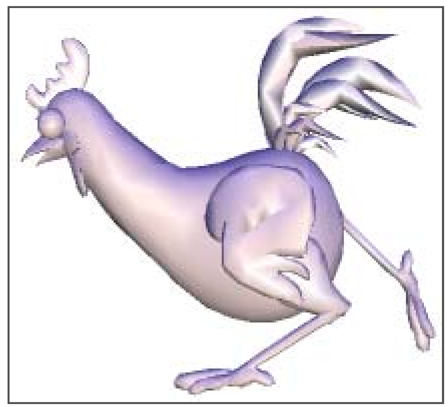


- To restrict ourselves to rotations only: take the last column of V (corresponding to the smallest singular value) and invert its sign.
- Why? See the PDF…

Complexity

- Numerical SVD is an expensive operation $O(\min(mn^2, nm^2))$
- We always need to pay attention to the dimensions of the matrix we're applying SVD to.

SVD for animation compression

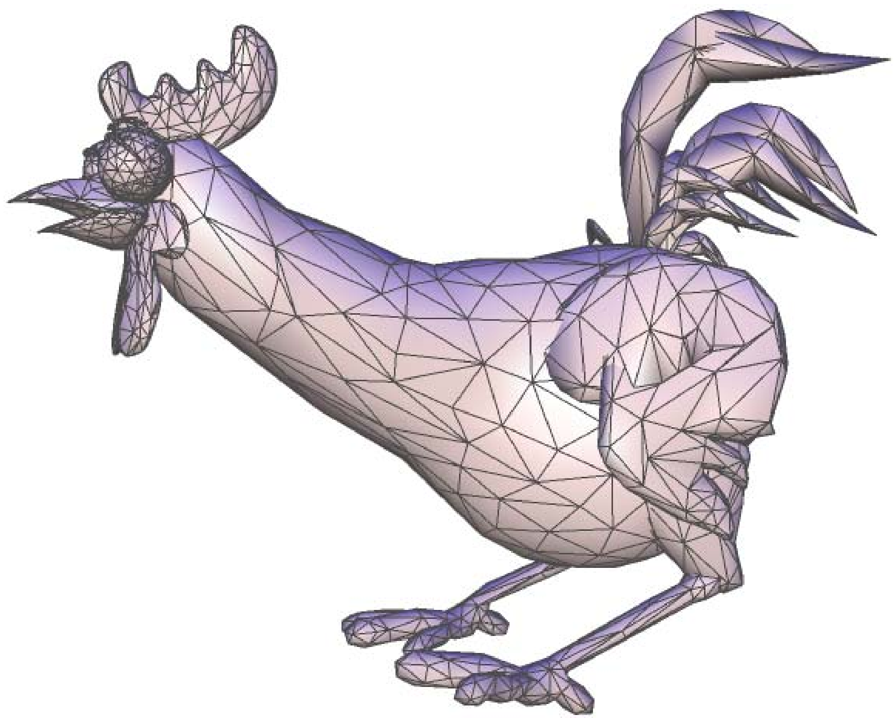


Chicken animation

See for instance:
Representing Animations by Principal Components, M. Alexa and W. Muller, Eurographics 2000
Compression of Soft-Body Animation Sequences, Z. Karni and C. Gotsman, Computers&Graphics 28(1): 25-34, 2004
Key Point Subspace Acceleration and Soft Caching, M. Meyer and J. Anderson, SIGGRAPH 2007

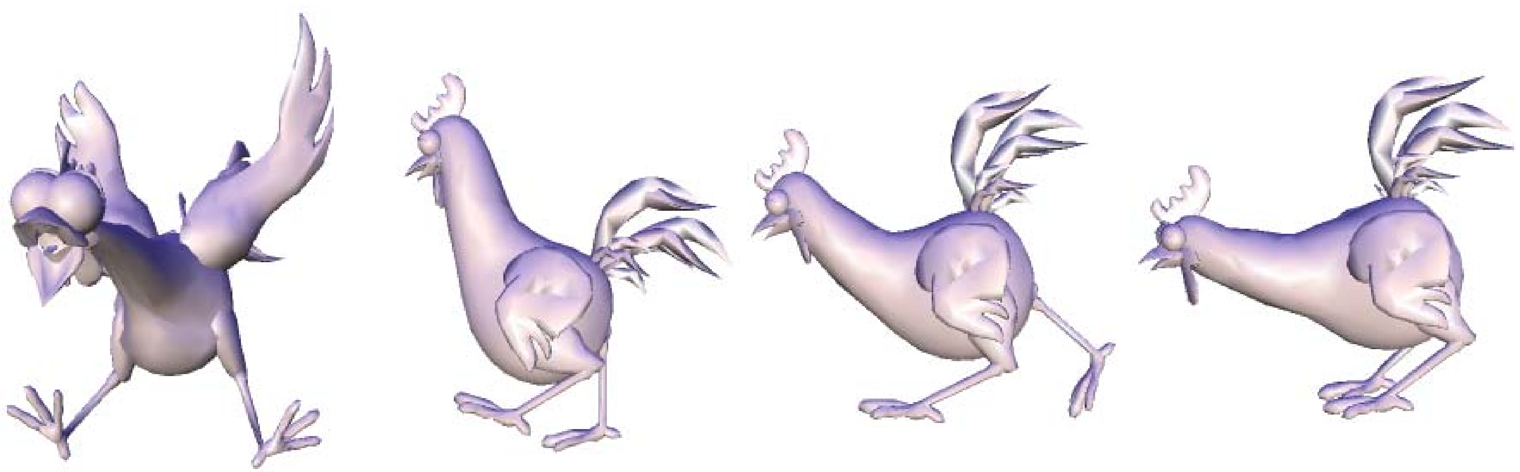
3D animations

- Each frame is a 3D model (mesh)



3D animations

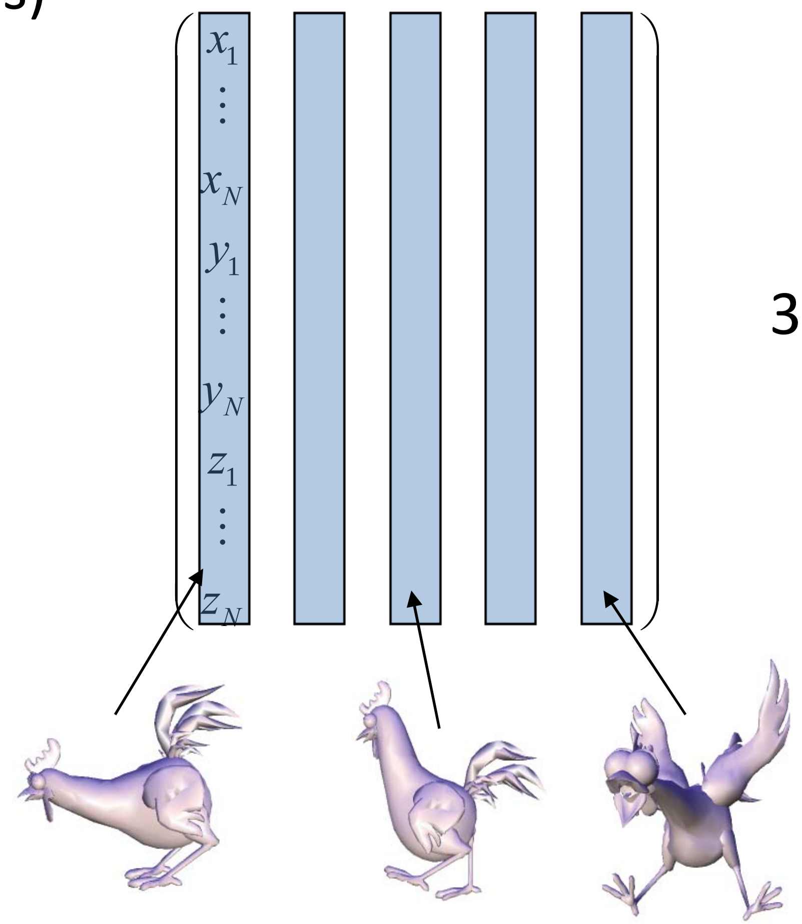
- Connectivity is usually constant (at least on large segments of the animation)
- The geometry changes in each frame $\rightarrow$ vast amount of data!



13 seconds, 3000 vertices/frame, 26 MB

Animation compression by dimensionality reduction

- The geometry of each frame is a vector in R^{3N} space (N = #vertices)

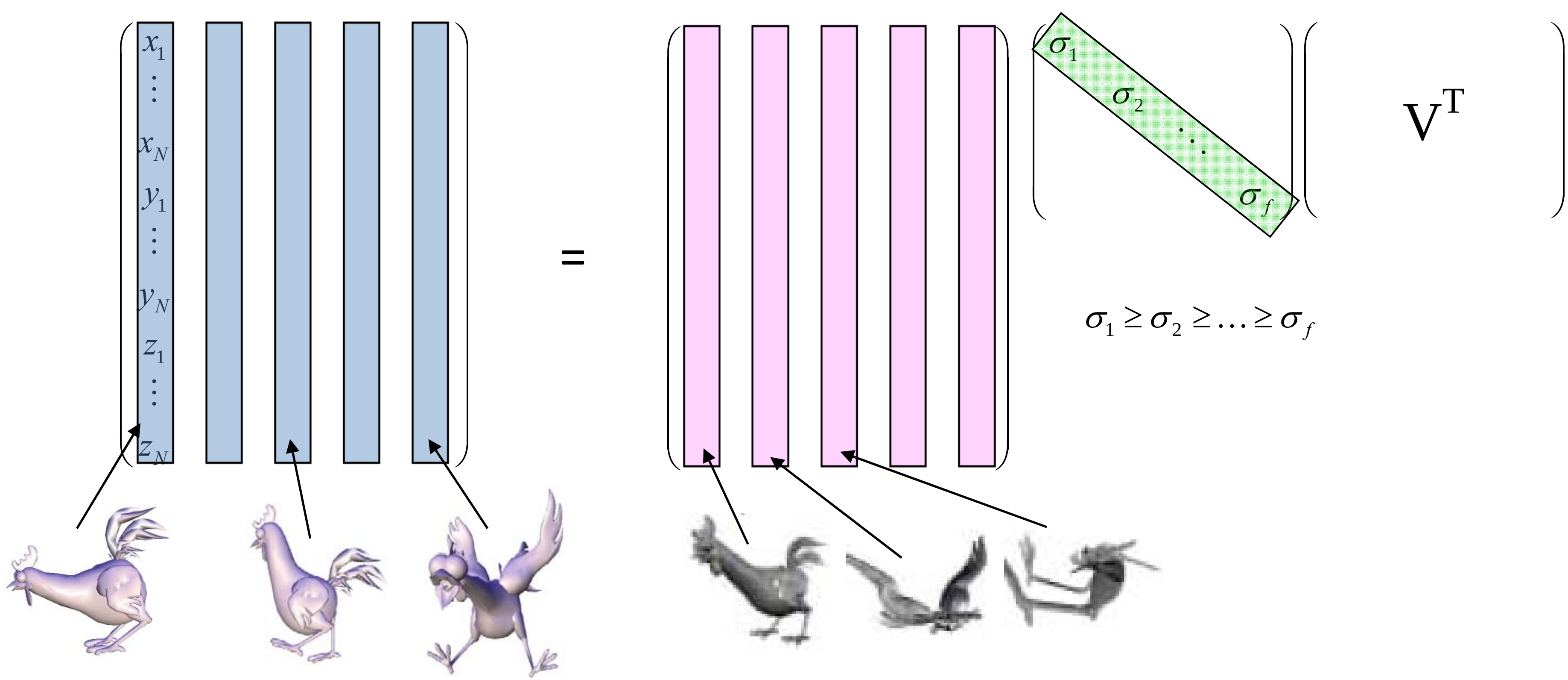


$3N \times \#f$

Animation compression by dimensionality reduction

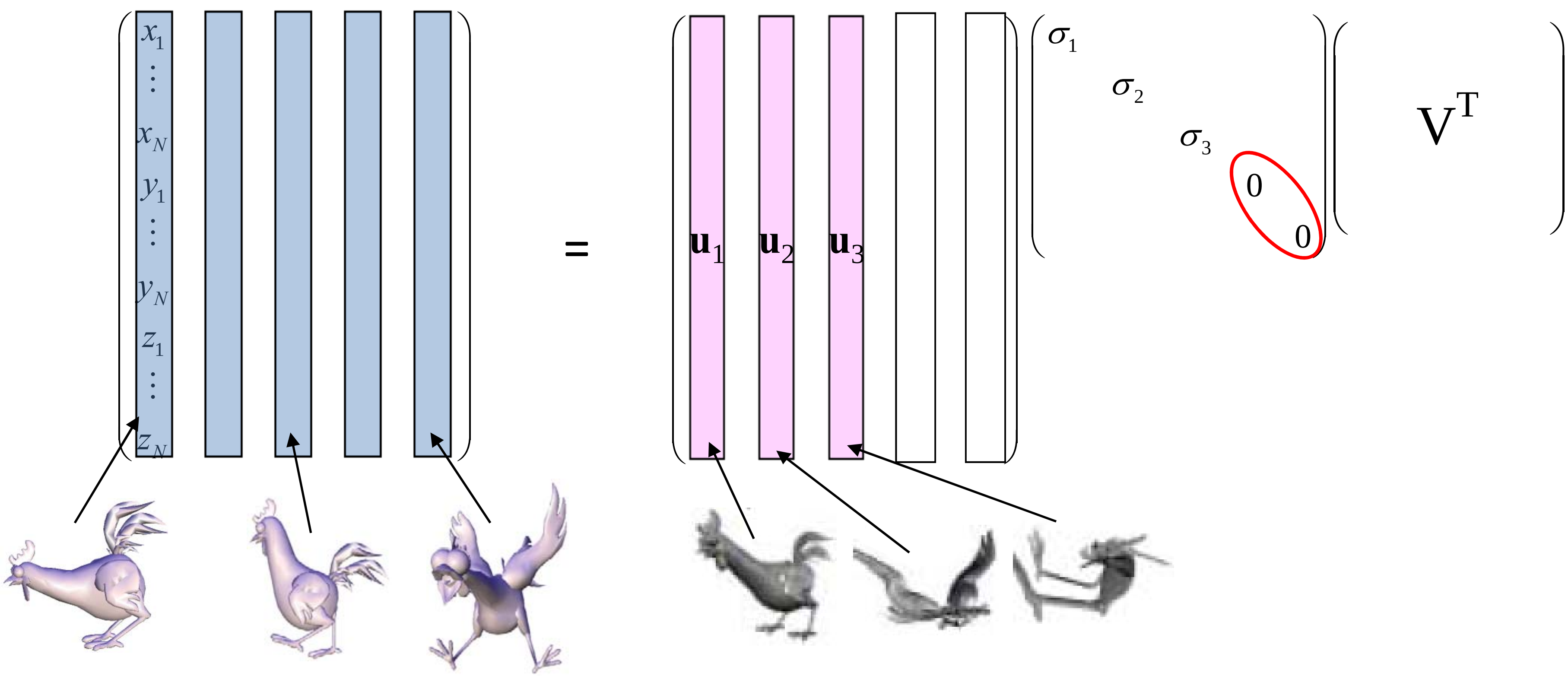
- Find a few vectors of R^{3N} that will best represent our frame vectors!

$U_{3N\times f}$ $\quad \Sigma_{f\times f}$ $\quad {V^T}_{f\times f}$



Animation compression by dimensionality reduction

- The first principal components are the important ones



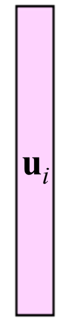
Animation compression by dimensionality reduction

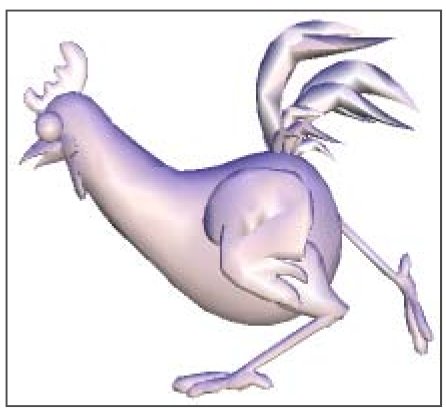
- Approximate each frame by linear combination of the first principal components
- The more components we use, the better the approximation
- Usually, the number of components needed is much smaller than f.

$$\begin{pmatrix} x_1 \\ \vdots \\ x_N \\ y_1 \\ \vdots \\ y_N \\ z_1 \\ \vdots \\ z_N \end{pmatrix} = \alpha_1 \mathbf{u}_1 + \alpha_2 \mathbf{u}_2 + \alpha_3 \mathbf{u}_3$$

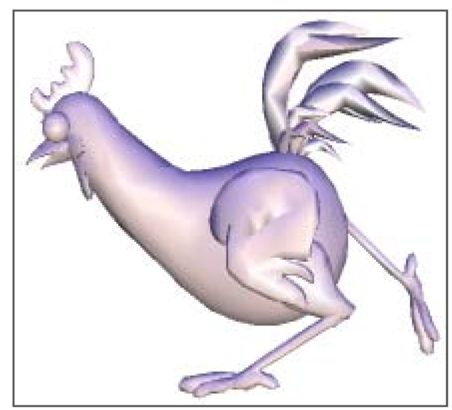
Animation compression by dimensionality reduction

- Compressed representation:
  - The chosen principal component vectors
  - Coefficients α_i for each frame





Animation with only
2 principal components



Animation with
4 out of 400 principal
components