# PCA & SVD

薛犇 吴润迪

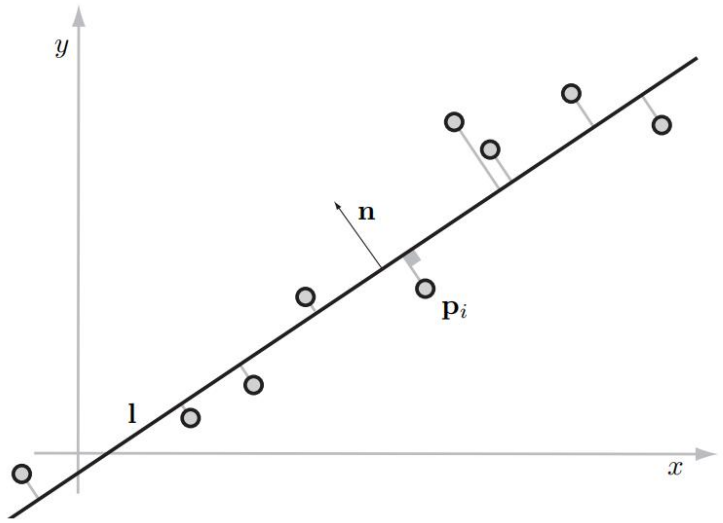
# Principal Component Analysis

- Start from a problem in least-square solution, fit one line
- Minimize the total distance from all points to

$$\min_{\mathbf{n},d} E(\mathbf{n},d) = \sum_{i=1}^{n} (\mathbf{n}^\top \mathbf{p}_i + d)^2, \quad \text{s.t. } \|\mathbf{n}\| = 1 .$$

- line **l :** $\mathbf{n}^\top \mathbf{x} + d = 0$
- Similar to previous solution:

$$\partial E(\mathbf{n},d)/\partial \mathbf{n} = 0 \qquad \partial E(\mathbf{n},d)/\partial d = 0$$



# Principal Component Analysis

$$
\begin{aligned}
& \partial E(\mathbf{n}, d)/\partial d = 0 \\
\Rightarrow \quad & 2\sum_i (\mathbf{n}^\top \mathbf{p}_i + d) = 0 \\
\Rightarrow \quad & nd = -\mathbf{n}^\top \sum_i \mathbf{p}_i \\
\Rightarrow \quad & d = -\mathbf{n}^\top \bar{\mathbf{p}}\,,
\end{aligned}
$$

- $\bar{\mathbf{p}} = \sum_i \mathbf{p}_i / n$ is the centroid of the given point set **P** .

$$\min_{\mathbf{n},d} E(\mathbf{n}, d) = \sum_{i=1}^{n} (\mathbf{n}^\top \mathbf{p}_i + d)^2, \quad \text{s.t. } \|\mathbf{n}\| = 1\,.$$

$$\min_{\mathbf{n}} E(\mathbf{n}) = \sum_{i=1}^{n} (\mathbf{n}^\top \mathbf{p}_i - \mathbf{n}^\top \bar{\mathbf{p}})^2 = \sum_{i=1}^{n} (\mathbf{n}^\top \tilde{\mathbf{p}}_i)^2, \quad \text{s.t. } \|\mathbf{n}\| = 1$$

# Principal Component Analysis

- Use Laplacian form:

$$\min_{\mathbf{n}} E(\mathbf{n}) = \sum_{i=1}^{n} (\mathbf{n}^\top \mathbf{p}_i - \mathbf{n}^\top \bar{\mathbf{p}})^2 = \sum_{i=1}^{n} (\mathbf{n}^\top \tilde{\mathbf{p}}_i)^2, \quad \text{s.t. } \|\mathbf{n}\| = 1$$

$$\Rightarrow \quad \min_{\mathbf{n}} \Big( \sum_{i} (\mathbf{n}^\top \tilde{\mathbf{p}}_i)^2 + \lambda(1 - \mathbf{n}^\top \mathbf{n}) \Big) = \min_{\mathbf{n}} (\mathbf{n}^\top C \mathbf{n} + \lambda(1 - \mathbf{n}^\top \mathbf{n}))$$

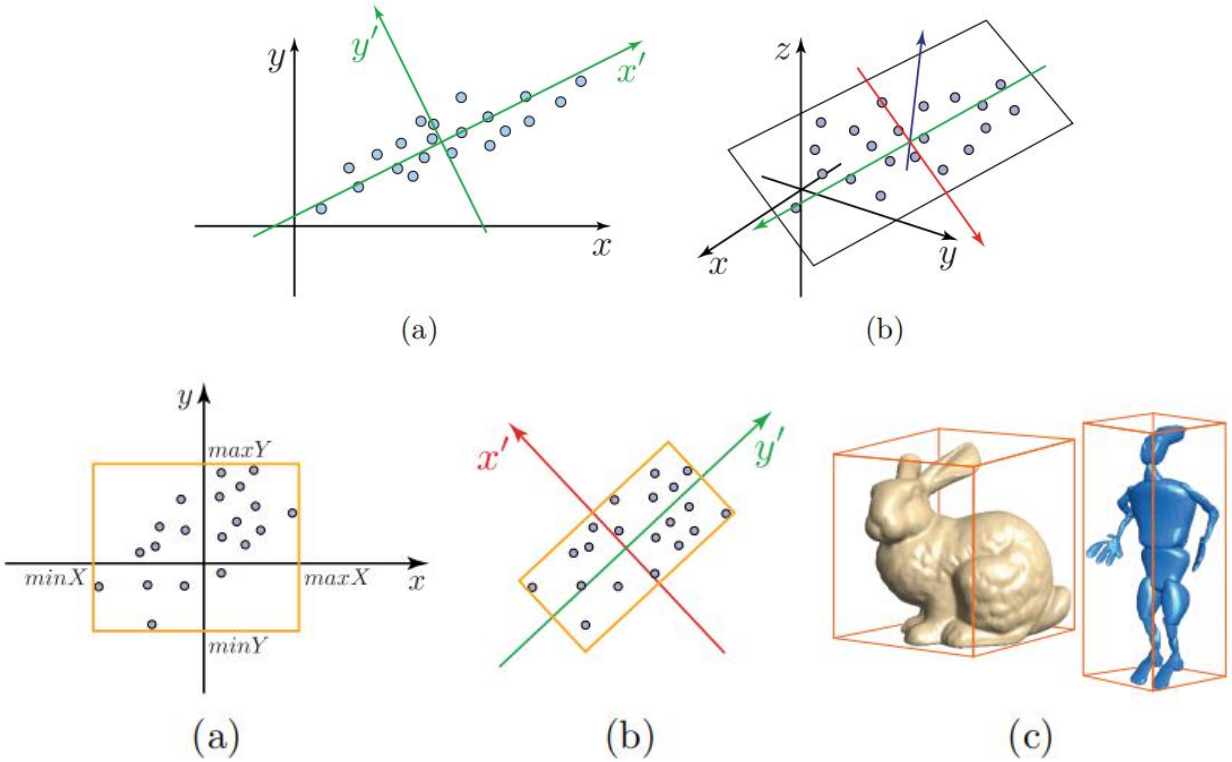
- $C = \sum_i \tilde{\mathbf{p}}_i \tilde{\mathbf{p}}_i^\top$ is the covariance matrix of the given point set.
- Setting the partial derivative of the above expression w.r.t **n** to zero, and using rules of matrix differentiation, we arrive at:

$$2C\mathbf{n} - 2\lambda\mathbf{n} = 0 \quad \Rightarrow \quad C\mathbf{n} = \lambda\mathbf{n}\,.$$

- Thus, **n** is one of the eigenvectors of covariance matrix $C$.

# Principal Component Analysis

- Principal component analysis finds orthogonal axes that represent the input points well in terms of linear structures



# PCA: Problem & Solution

- We need to find the directions that represents our data best.
- The variance of the projected points $\mathbf{x}'_i$ measures how far they are spread away from the center

$$\mathrm{var}(\ell) = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}'_i - \mathbf{m}\|^2.$$

- **m** is the mass(centroid of point $\mathbf{m} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$.
- Define the direction of line **l** by , and assume that **v** is normalized.
- So we can replace the projected points $\mathbf{x}'_i$ in the expression of covariance:

$$\|\mathbf{x}'_i - \mathbf{m}\| = \left|\frac{\langle \mathbf{v}, \mathbf{x}_i - \mathbf{m}\rangle}{\|\mathbf{v}\|}\right| = |\langle \mathbf{v}, \mathbf{y}_i\rangle| = |\mathbf{v}^\top \mathbf{y}_i|$$

# PCA: Problem & Solution

$$\begin{aligned}\mathrm{var}(\ell) &= \frac{1}{n}\sum_{i=1}^{n}\|\mathbf{x}_i' - \mathbf{m}\|^2 = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{v}^\top\mathbf{y}_i)^2 = \frac{1}{n}\|\mathbf{v}^\top Y\|^2 \\ &= \frac{1}{n}(\mathbf{v}^\top Y)(\mathbf{v}^\top Y)^\top = \frac{1}{n}\mathbf{v}^\top YY^\top\mathbf{v} = \frac{1}{n}\langle S\mathbf{v},\mathbf{v}\rangle\,.\end{aligned}$$

- **S** is the scatter(covariance) matrix:

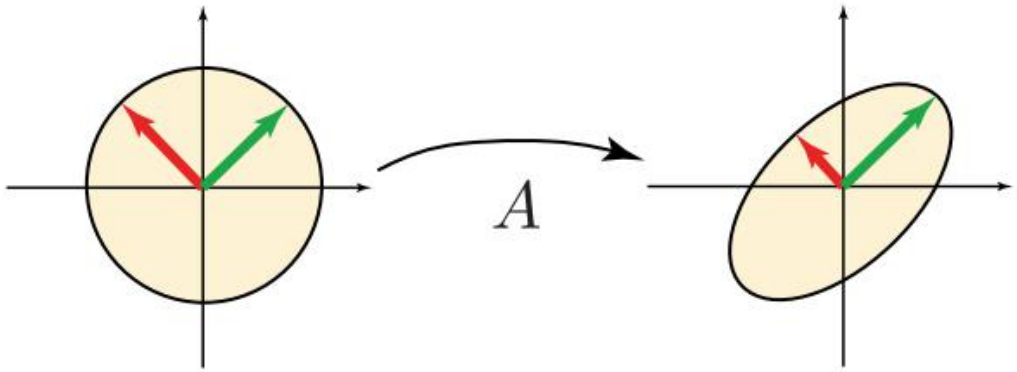
$$S = YY^\top \qquad Y = \begin{bmatrix} | & | & & | \\ \mathbf{y}_1 & \mathbf{y}_2 & \cdots & \mathbf{y}_n \\ | & \| & & | \end{bmatrix} \qquad \mathbf{y}_i = \mathbf{x}_i - \mathbf{m}$$

- Eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_d$ Eigenve $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d$
- The extremal values of the variance are, therefore:

$$\langle S\mathbf{v}_i, \mathbf{v}_i\rangle = \langle \lambda_i\mathbf{v}_i, \mathbf{v}_i\rangle = \lambda_i\langle \mathbf{v}_i, \mathbf{v}_i\rangle = \lambda_i$$

# Singular Value Decomposition

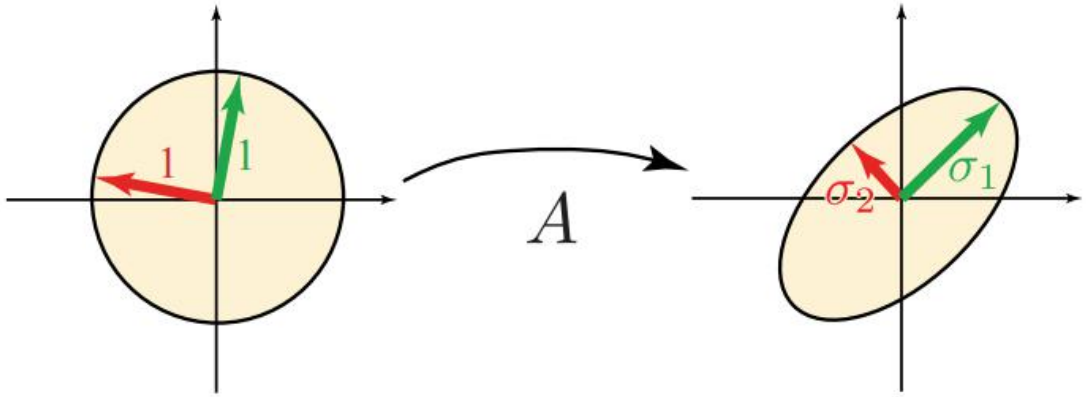
- Given a transformation A, let's see what it does geometrically:
- If we are lucky, th $A = V \Lambda V^{\top}$ , $V$ being orthogonal, meaning $A$ is symmetric, and the eigenvectors of $A$ are axes of the ellipse:




- The eigendecomposition of A tells us which orthogonal exes it scales.

# Singular Value Decomposition

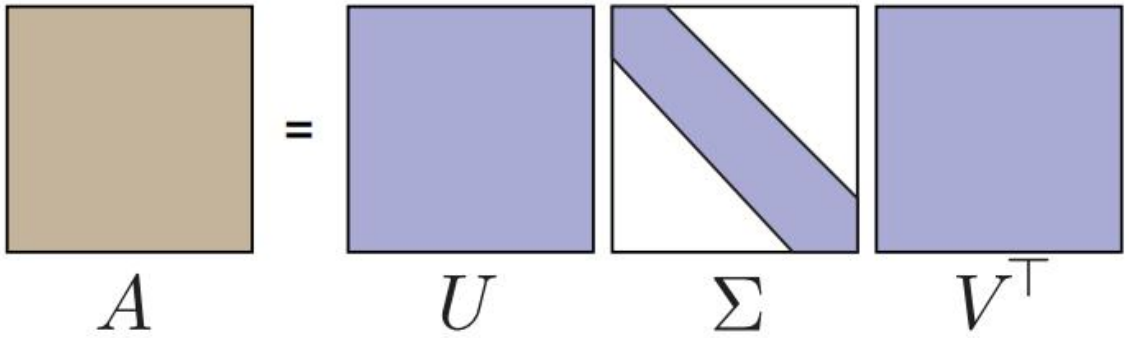
- However, in general, $A$ also contains rotations, not just scaling along orthogonal axes




- $A$ then looks like this:

$$A = (\mathbf{u}_1\ \mathbf{u}_2\ \cdots\ \mathbf{u}_n) \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix} (\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_n)^\top$$

# Singular Value Decomposition




- $\Sigma$ is a matrix with values on diagonal that are real and nonnegative
- $U$ and $V$ are orthogonal matrices
- SVD exists for any matrix of any dimension

# SVD: Algorithms

- Solutions:
  - Golub Kahan
  - Jacobi Rotation

# SVD: Algorithms - Golub Kahan

Pipeline:

1. Householder transformation, convert it to a bidiagonal matrix
2. QR algorithm to approximate the eigenvalue of this bidiagonal matrix

a **bidiagonal matrix** is a matrix with non-zero entries along the main diagonal and *either* the diagonal above or the diagonal below

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 4 & 0 & 0 \\ 0 & 3 & 3 & 0 \\ 0 & 0 & 4 & 3 \end{pmatrix} \quad \begin{pmatrix} 1 & 4 & 0 & 0 \\ 0 & 4 & 1 & 0 \\ 0 & 0 & 3 & 4 \\ 0 & 0 & 0 & 3 \end{pmatrix}$$

# SVD: Algorithms - Golub Kahan

- A Householder matrix P:

$$P = 1 - 2ww^T, |w|^2 = 1$$

$$\begin{aligned} P^2 =& (1 - 2w \cdot w^T) \cdot (1 - 2w \cdot w^T) \\ =& 1 - 4w \cdot w^T + 4w \cdot (w^T \cdot w) \cdot w^T \\ =& 1 \end{aligned}$$

- Therefore:

$$\begin{aligned} P =& P^{-1} \\ P^T =& P \end{aligned}$$

- $P$ is orthogonal

# SVD: Algorithms - Golub Kahan

- Change a form:

$$P = 1 - \frac{u \cdot u^T}{H}$$

$$H = \frac{1}{2}|u|^2$$

- Now **u** can be any vector. Assume **x** is the first column of matrix *A*

$$u = x + |x|e_0$$

$$e_0 = [1, 0, 0, \ldots .0]^T$$

# SVD: Algorithms - Golub Kahan

- We have

$$
\begin{aligned}
P \cdot x =& x - \frac{u}{H} \cdot (x + |x| e_0)^T \cdot x \\
=& x - \frac{2u \cdot (|x|^2 + |x| x_0)}{2|x|^2 + 2|x| x_0} \\
=& x - u \\
=& |x| e_0
\end{aligned}
$$

- So we can convert all elements in first column of A to 0 except the first one.
- And then we can iteratively convert A to a bidiagonal matrix.

# SVD: Algorithms - Golub Kahan (quote)

(1) 首先确定一个 Household 矩阵 $H_1 \in \mathbb{R}^{m\times m}$, 使得 $H_1 A$ 的第一列除第一个元素外, 其它分量都为零, 即

$$H_1 A = \begin{bmatrix} * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & * & * & \cdots & * \end{bmatrix}.$$

潘建瑜，华东师范大学数学系，《矩阵计算/数值线性代数》

# SVD: Algorithms - Golub Kahan (quote)

(2) 再确定一个 Household 矩阵 $\tilde{H}_1 \in \mathbb{R}^{(n-1)\times(n-1)}$, 把 $H_1 A$ 的第一行的第 3 至第 $n$ 个元素化为零, 即

$$H_1 A \begin{bmatrix} 1 & 0 \\ 0 & \tilde{H}_1 \end{bmatrix} = \begin{bmatrix} * & * & 0 & \cdots & 0 \\ 0 & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & * & * & \cdots & * \end{bmatrix}.$$

(3) 重复上面的过程, 直到把 $A$ 最终化为二对角矩阵.

潘建瑜，华东师范大学数学系，《矩阵计算/数值线性代数》

# SVD: Algorithms - Golub Kahan (quote)

$$A^TA = (U_1BV_1^T)^TU_1BV_1^T = V_1B^TBV_1^T,$$

即 $V_1^TA^TAV_1 = B^TB$. 由于 $B^TB$ 是对称三对角的, 所以这就相当于将 $A^TA$ 三对角化.

整个二对角化过程的运算量约为 $4mn^2 + 4m^2n - 4n^3/3$, 若不需要计算 $U_1$ 和 $V_1$, 则运算量约为 $4mn^2 - 4n^3/3$.

潘建瑜，华东师范大学数学系，《矩阵计算/数值线性代数》

# SVD: Algorithms - Golub Kahan (quote)

**二对角矩阵的奇异值分解**

设 $B \in \mathbb{R}^{n\times n}$ 是一个二对角矩阵 $B = \begin{bmatrix} a_1 & b_1 & & \\ & \ddots & \ddots & \\ & & \ddots & b_{n-1} \\ & & & a_n \end{bmatrix}$,

潘建瑜，华东师范大学数学系，《矩阵计算/数值线性代数》

# SVD: Algorithms - Golub Kahan (quote)

令 $T_{BB^T} = BB^T$, 则

$$T_{BB^T} = \begin{bmatrix} a_1^2 + b_1^2 & a_2 b_1 & & \\ a_2 b_1 & \ddots & \ddots & \\ & \ddots & a_{n-1}^2 + b_{n-1}^2 & a_n b_{n-1} \\ & & a_n b_{n-1} & a_n^2 \end{bmatrix}.$$

$T_{BB^T}$ 的特征值为 $B$ 的奇异值的平方, 且 $T_{BB^T}$ 的特征向量为 $B$ 的左奇异向量.

令 $T_{B^TB} = B^TB$, 则

$$T_{B^TB} = \begin{bmatrix} a_1^2 & a_1 b_1 & & \\ a_1 b_1 & a_2^2 + b_1^2 & \ddots & \\ & \ddots & \ddots & a_{n-1} b_{n-1} \\ & & a_{n-1} b_{n-1} & a_n^2 + b_{n-1}^2 \end{bmatrix}.$$

$T_{B^TB}$ 的特征值为 $B$ 的奇异值的平方, 且 $T_{B^TB}$ 的特征向量为 $B$ 的右奇异向量.

潘建瑜，华东师范大学数学系，《矩阵计算/数值线性代数》

# SVD: Algorithms - Golub Kahan

- In application: use QR algorithm to calculate the eigenvector
- Any real square matrix $A$ may be decompc $A = QR$
- $Q$ is an orthogonal matrix and $R$ is an upper triangular matrix
- Formally, let $A$ be a real matrix of which we want to compute the eigenvalues, and let $A_0 := A$. At the $k$-th step (starting with $k = 0$), we compute the QR decomposition $A_k = Q_k R_k$ where $Q_k$ is an orthogonal matrix (i.e., $Q^T = Q^{-1}$) and $R_k$ is an upper triangular matrix. We then form $A_{k+1} = R_k Q_k$. Note that

$$A_{k+1} = R_k Q_k = Q_k^{-1} Q_k R_k Q_k = Q_k^{-1} A_k Q_k = Q_k^{\mathsf{T}} A_k Q_k$$

- so all the $A_k$ are similar and hence they have the same eigenvalues. The algorithm is numerically stable because it proceeds by *orthogonal* similarity transforms.

# SVD: Algorithms - Golub Kahan (quote)

- Why we use QR?
- For common matrix, one iteration of QR decomposition cost O(n^3)
- But for tridiagonal matrix, time cost is O(n)

**对称 QR 迭代算法的运算量**

- 三对角化 $4n^3/3 + O(n^2)$, 若需计算特征向量, 则为 $8n^3/3 + O(n^2)$;
- 对 $T$ 做带位移的隐式 QR 迭代, 每次迭代的运算量为 $6n$;
- 计算特征值, 假定每个平均迭代 2 步, 则总运算量为 $12n^2$;
- 若要计算 $T$ 的所有特征值和特征向量, 则运算量为 $6n^3 + O(n^2)$;
- 若只要计算 $A$ 的所有特征值, 运算量为 $4n^3/3 + O(n^2)$;
- 若计算 $A$ 的所有特征值和特征向量, 则运算量为 $26n^3/3 + O(n^2)$;

# SVD: Algorithms - Jacobi Rotation (quote)

**基本思想:** 通过一系列的 Jacobi 旋转 将 $A$ 正交相似于一个对角矩阵:

$$A^{(0)} = A, \quad A^{(k+1)} = J_k^T A^{(k)} J_k, \quad k = 0, 1, \ldots,$$

且 $A^{(k)}$ 收敛到一个对角矩阵, 其中 $J_k$ 为 Jacobi 旋转, 即 Givens 变换:

$$J_k = G(i_k, j_k, \theta_k) = \begin{array}{c} \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccc} & & i_k & & & j_k & \\ \end{array} \\ \left[\begin{array}{c|ccc|c} I & & & & \\ \hline & \cos\theta_k & \cdots & -\sin\theta_k & \\ & \vdots & \ddots & \vdots & \\ & \sin\theta_k & \cdots & \cos\theta_k & \\ \hline & & & & I \end{array}\right] \end{array} \begin{array}{l} \\ \\ i_k \\ \\ j_k \\ \\ \end{array}$$

潘建瑜，华东师范大学数学系，《矩阵计算/数值线性代数》

# SVD: Algorithms - Jacobi Rotation

- Givens Transformation, Consider:

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

$$c \leftarrow a/r$$
$$s \leftarrow -b/r.$$

$$r = \sqrt{a^2 + b^2}$$

# SVD: Algorithms - Jacobi Rotation (quote)

**引理** 设 $A \in \mathbb{R}^{2\times 2}$ 是对称矩阵, 则存在 Givens 变换 $G \in \mathbb{R}^{2\times 2}$ 使得 $G^T AG$ 为对角阵. (板书)

为了使得 $A^{(k)}$ 收敛到一个对角矩阵, 其非对角元素必须趋向于 0.

记 $\mathrm{off}(A)$ 为所有非对角元素的平方和, 即

$$\mathrm{off}(A) = \sum_{i\neq j} a_{ij}^2 = \|A\|_F^2 - \sum_{i=1}^{n} a_{ii}^2 ,$$

我们的目标就是使得 $\mathrm{off}(A)$ 尽快趋向于 0.

**引理** 设 $A = [a_{ij}]_{n\times n} \in \mathbb{R}^{n\times n}$ 是对称矩阵, $\hat{A} = [\hat{a}_{ij}]_{n\times n} = J^T AJ$, $J = G(i, j, \theta)$, 其中 $\theta$ 的选取使得 $\hat{a}_{ij} = \hat{a}_{ji} = 0$, 则

$$\mathrm{off}(\hat{A}) = \mathrm{off}(A) - 2a_{ij}^2.$$

(板书)

潘建瑜，华东师范大学数学系，《矩阵计算/数值线性代数》

# SVD: Algorithms - Jacobi Rotation (quote)

**算法 1.1** Jacobi 迭代算法

1: Given a symmetric matrix $A \in \mathbb{R}^{n\times n}$
2: **if** eigenvectors are desired **then**
3: set $J = I$ and $shift = 1$
4: **end if**
5: **while** not converge **do**
6: choose an index pair $(i, j)$ such that $a_{ij} \neq 0$
7: $\tau = (a_{ii} - a_{jj})/(2a_{ij})$
8: $t = \text{sign}(\tau)/(|\tau| + \sqrt{1+\tau^2})$ % 计算 $\tan\theta$
9: $c = 1/\sqrt{1+t^2}, s = c \cdot t$ % 计算 Givens 变换
10: $A = G(i, j, \theta)^T A G(i, j, \theta)$ % 实际计算时不需要做矩阵乘积
11: **if** $shift = 1$ **then**
12: $J = J \cdot G(i, j, \theta)$
13: **end if**
14: **end while**

**$a_{ij}$ 的选取问题**

一种直观的选取方法就是使得 $a_{ij}$ 为所有非对角元素中绝对值最大的一个, 这就是经典 Jacobi 算法.

潘建瑜，华东师范大学数学系，《矩阵计算/数值线性代数》

# SVD: Algorithms - Jacobi Rotation (quote)

可以证明, 经典 Jacobi 算法至少是线性收敛的.

**定理** 对于经典 Jacobi 算法 1.2, 有

$$\mathrm{off}(A^{(k+1)}) \leq \left(1 - \frac{1}{N}\right) \mathrm{off}(A^{(k)}), \quad N = \frac{n(n-1)}{2}.$$

故 $k$ 步迭代后, 有

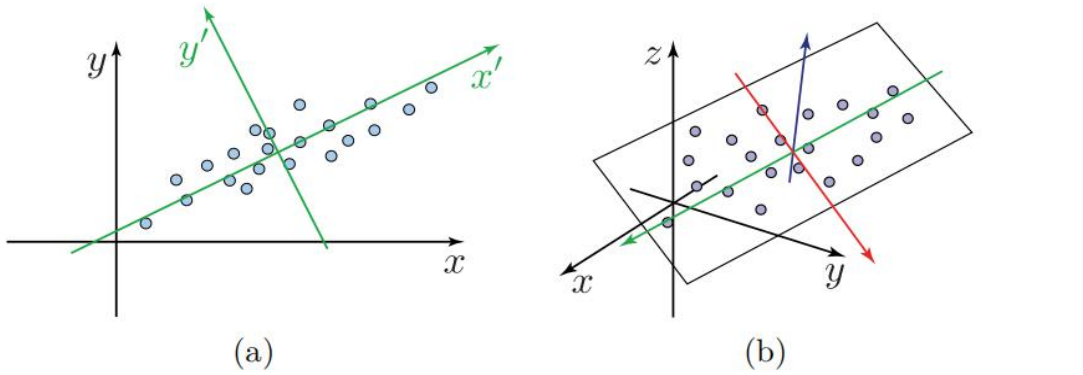
$$\mathrm{off}(A^{(k)}) \leq \left(1 - \frac{1}{N}\right)^k \mathrm{off}(A^{(0)}) = \left(1 - \frac{1}{N}\right)^k \mathrm{off}(A).$$

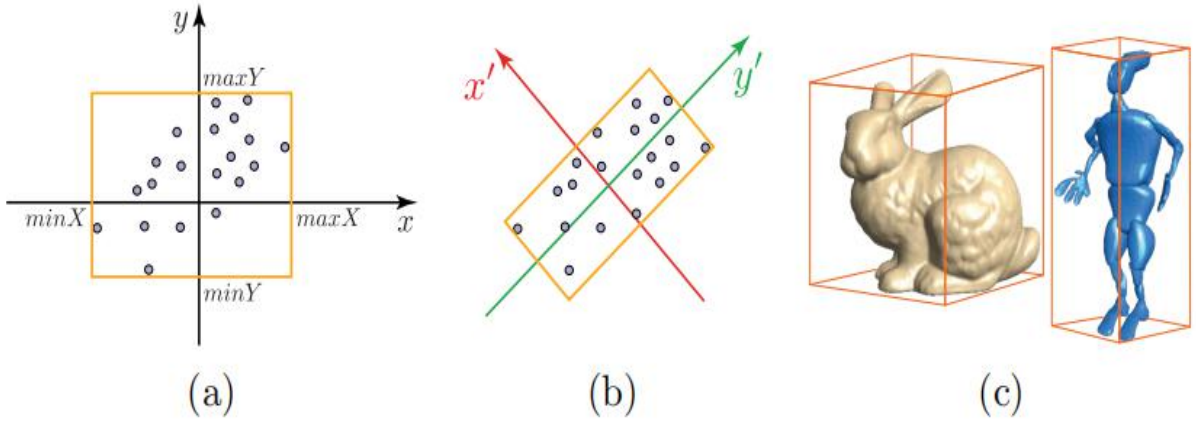
(板书)

# Application

- Find bounding box
- Approximate normal of point cloud
- Shape matching
- 3D animation compression by principal components
- TensorTextures: multilinear image-based rendering

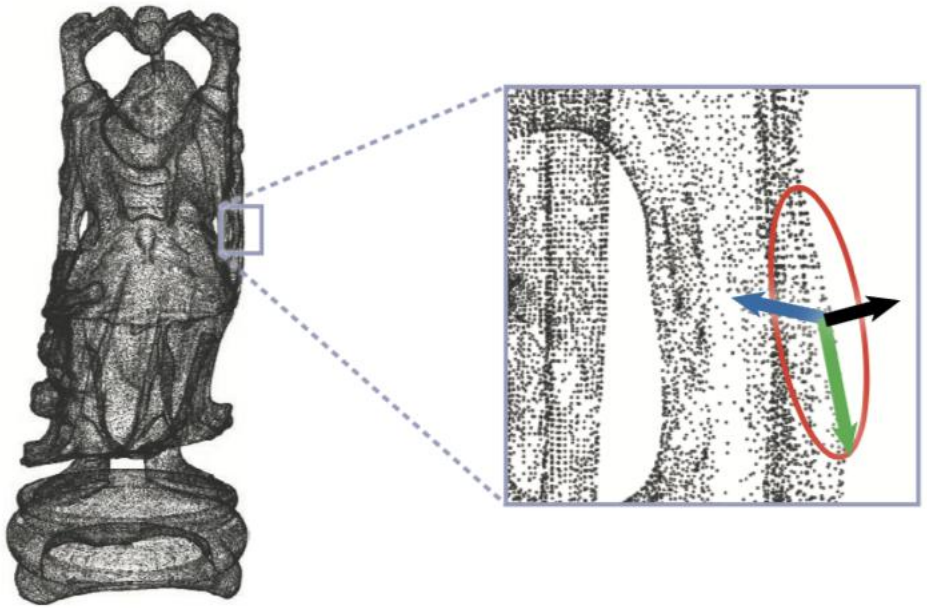
# Find bounding box



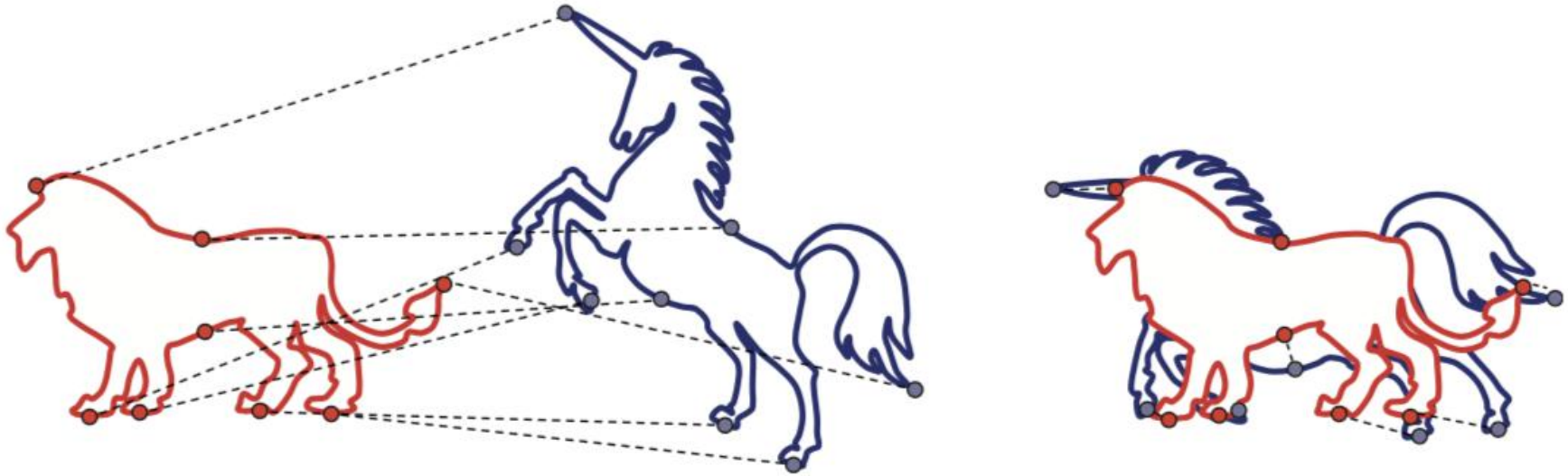
# Approximate normal of point cloud

- Compute the PCA of its local neighboring points
- The direction of the normal is simply the direction of the third eigenvector, corresponding to the smallest eigenvalue.



# Shape matching

- Align two shapes in correspondence by a rigid transformation



# 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 **t** and rotation matrix **R** such that:

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

# Shape matching - solution

- It turns out that we can solve for the translation and rotation separately.
- If **(R, t)** is the optimal transformation, then the points $\{p_i\}$ and $\{Rq_i + t\}$ have the same centers of mass.

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

# Shape matching - solution

- To find the optimal **R**, we bring the centers of both point sets to the origin:

$$p_i \leftarrow p_i - p, \qquad q_i \leftarrow q_i - p$$

- Now we want to find **R** that minimizes

$$\sum_{i=1}^{n} \|\mathbf{p}_i - R\mathbf{q}_i\|^2.$$

# Shape matching - solution

- Given the orthogonality of **R**, we have $R^T R = I$ and hence

$$\begin{aligned}\sum_{i=1}^{n} \|\mathbf{p}_i - R\,\mathbf{q}_i\|^2 &= \sum_{i=1}^{n} (\mathbf{p}_i - R\,\mathbf{q}_i)^\top (\mathbf{p}_i - R\,\mathbf{q}_i) \\ &= \sum_{i=1}^{n} \mathbf{p}_i{}^\top \mathbf{p}_i - \mathbf{p}_i{}^\top R\,\mathbf{q}_i - \mathbf{q}_i{}^\top R\,\mathbf{p}_i + \mathbf{q}_i{}^\top R^\top R\,\mathbf{q}_i \\ &= \sum_{i=1}^{n} \mathbf{p}_i{}^\top \mathbf{p}_i - \mathbf{p}_i{}^\top R\,\mathbf{q}_i - \mathbf{q}_i{}^\top R\,\mathbf{p}_i + \mathbf{q}_i{}^\top \mathbf{q}_i\,.\end{aligned}$$

- where the first and last terms do not depend on **R** and the middle two terms are scalars.

# Shape matching - solution

- Thus we have

$$\min_R \sum_{i=1}^{n} (-\mathbf{p}_i{}^\top R\,\mathbf{q}_i - \mathbf{q}_i{}^\top R\,\mathbf{p}_i) = \max_R \sum_{i=1}^{n} (\mathbf{p}_i{}^\top R\,\mathbf{q}_i + \mathbf{q}_i{}^\top R\,\mathbf{p}_i)$$

- Since $\boldsymbol{p_i^T R q_i} = (\boldsymbol{p_i^T R q_i})^T = \boldsymbol{p_i^T R^T q_i}$, we actually have

$$\operatorname*{argmax}_R \sum_{i=1}^{n} \mathbf{p}_i{}^\top R\,\mathbf{q}_i$$

$$\sum_{i=1}^{n} \mathbf{p}_i{}^\top R\,\mathbf{q}_i = \mathrm{Trace}\left(\sum_{i=1}^{n} R\,\mathbf{q}_i\mathbf{p}_i{}^\top\right) = \mathrm{Trace}\left(R\sum_{i=1}^{n} \mathbf{q}_i\mathbf{p}_i{}^\top\right)$$

# Shape matching - solution

- Let

$$H = \sum_{i=1}^{n} \mathbf{q}_i \mathbf{p}_i^\top$$

- So we want to find **R** that minimizes

$$\mathrm{Trace}(RH)$$

Theorem: if $M$ is symmetric positive definite (all eigenvalues of $M$ are positive) and $B$ is any orthogonal matrix then

$$Trace(M) \geq Trace(BM)$$

# Shape matching - solution

- Now we only need to find **R** such that **RH** is symmetric positive definite. Then we know for sure *Trace*(**RH**) is maximal.

- Compute SVD of **H**

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

- Define $R = VU^\top$

$$RH = (VU^\top)(U\Sigma V^\top) = V\Sigma V^\top$$

- which is symmetric and its eigenvalues are positive.

# Shape matching - solution

- Summary

Translate the input points to the centroids:

$$\mathbf{p}'_i = \mathbf{p}_i - \mathbf{p} \qquad \mathbf{q}'_i = \mathbf{q}_i - \mathbf{q}$$

Compute the "covariance matrix"

$$H = \sum_{i=1}^{n} \mathbf{q}'_i \mathbf{p}'^T_i$$

Compute the SVD of $H$:

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

The optimal rotation is

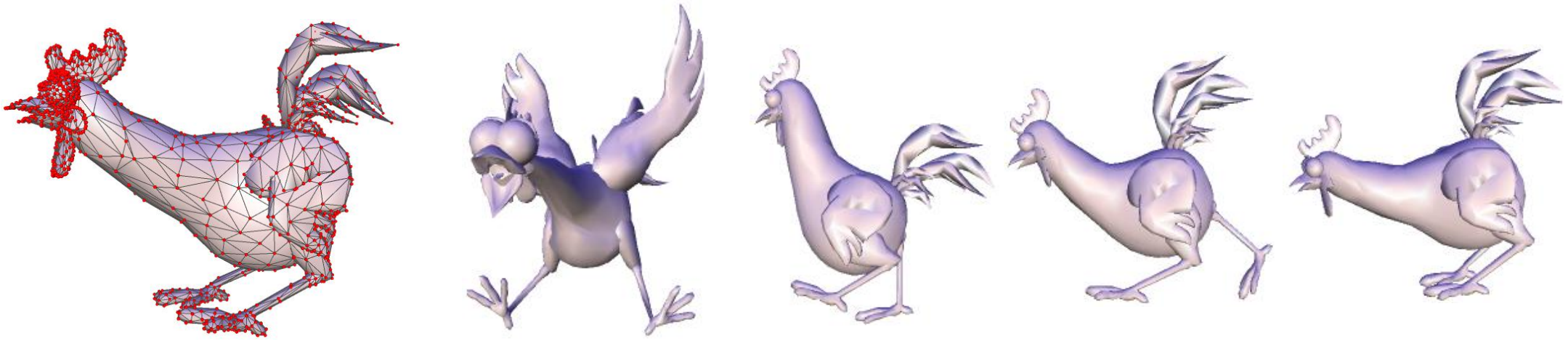
$$R = V U^T$$

The translation vector is

$$\mathbf{t} = \mathbf{p} - R\mathbf{q}$$

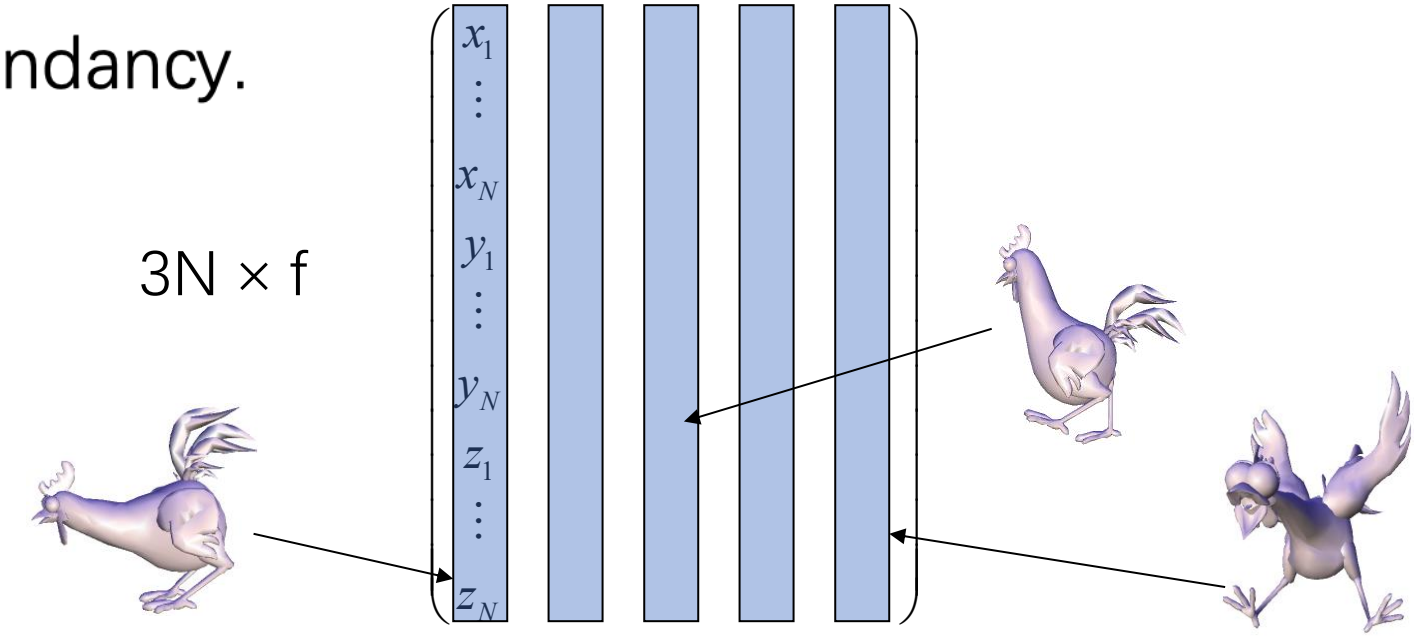
# 3D animations compression

- Each frame is a 3D-model
- Geometry – 3D coordinates of the vertices
- When the number of vertices is huge, how to represent it?



# 3D animations compression

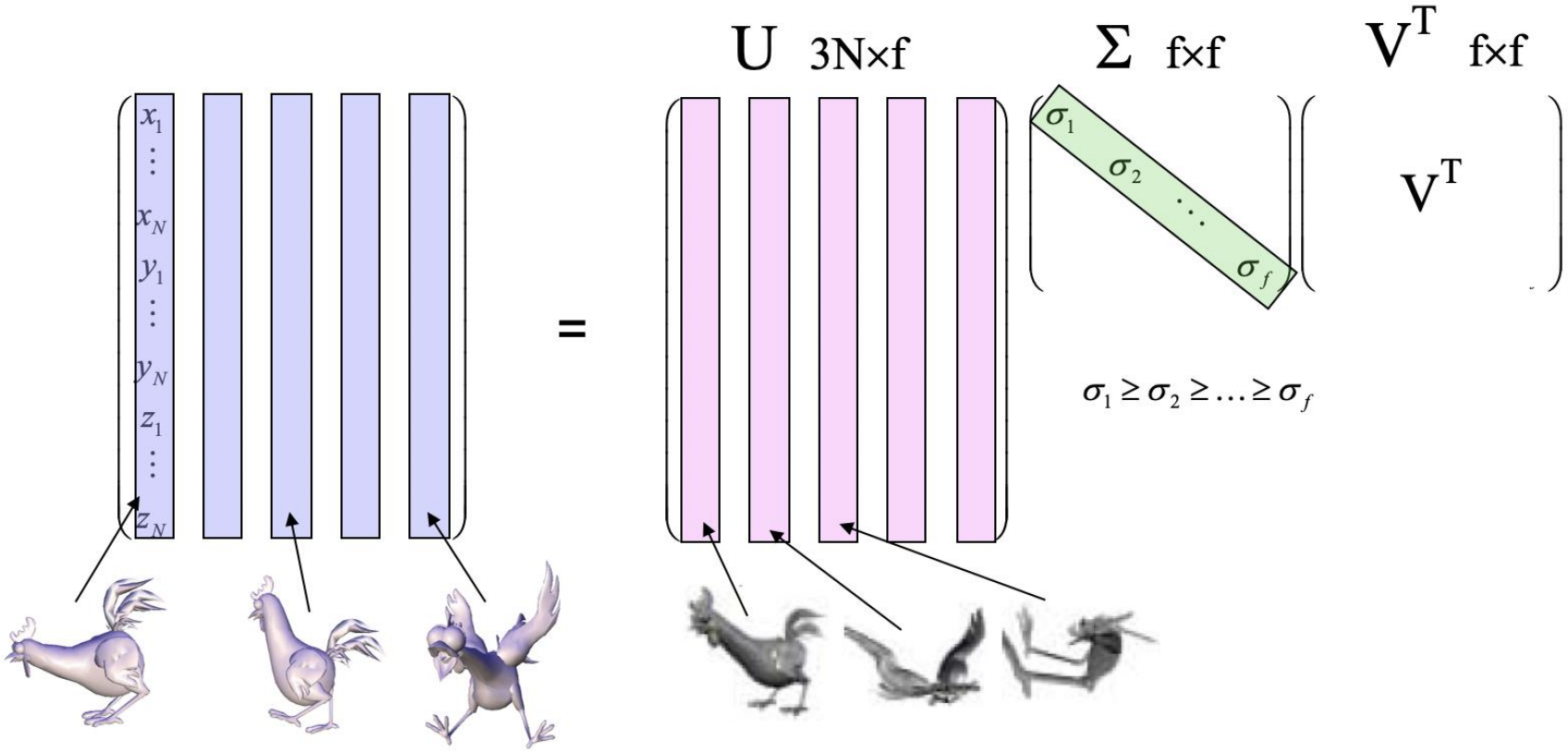
- Only describe key frames $B_i$.
- Additional frames are generated by interpolation between two consecutive key frames. $A(t) = \sum_i a_i(t) \cdot B_i$.
- Still there is some redundancy.



# 3D animations compression

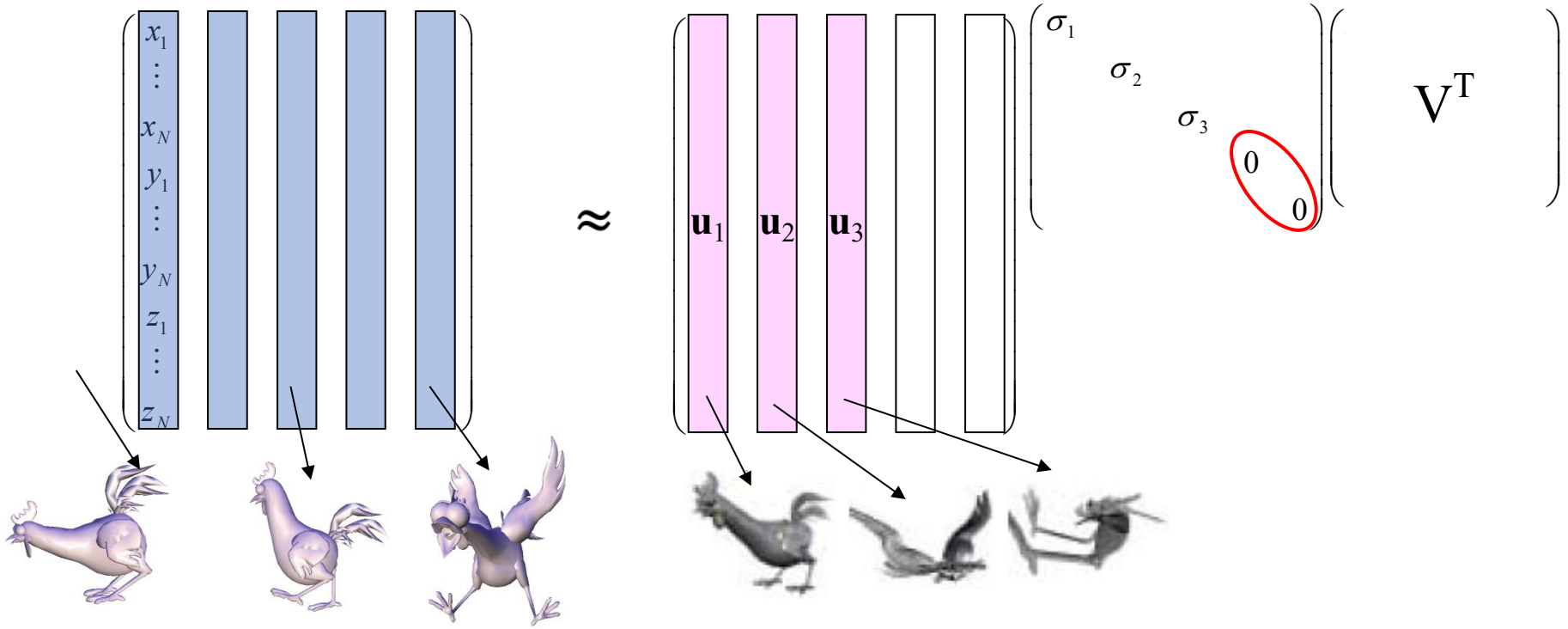
- Find a few new vectors in $R^{3N}$ that can best describe our frames.

-



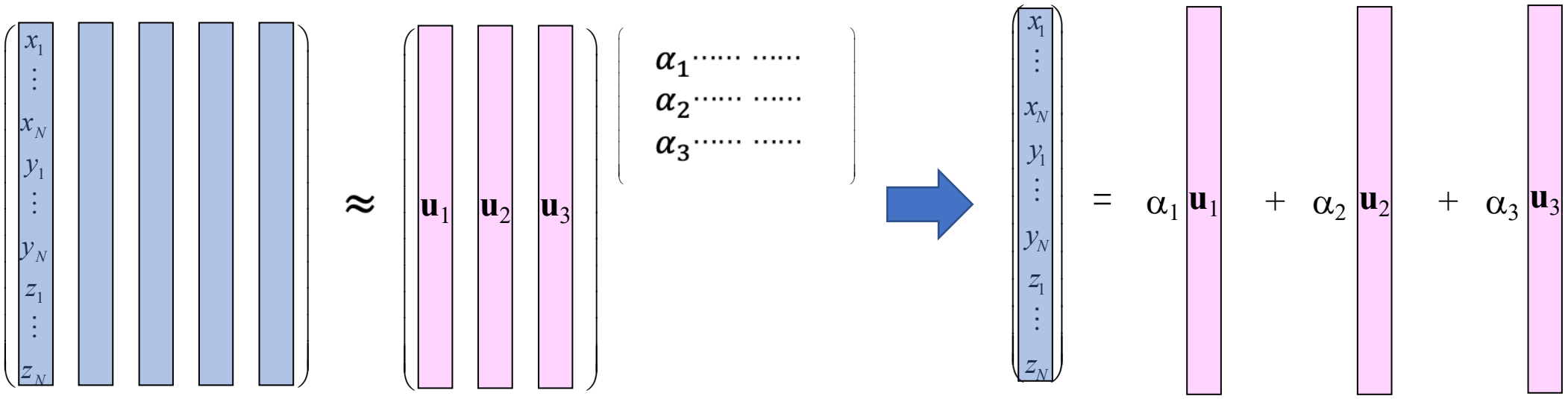
# 3D animations compression

- Taking the first **k** principle components
- According to the property of SVD, it is the best rank **k** approximation of the original matrix.



# 3D animations compression

- Approximate each frame by linear combination of the first a few principal components.



# TensorTexture: purely image-based rendering

- A framework that learns a parsimonious model of the bidirectional texture function (BTF) from observational data.
- BTF is a function of six variables $(x, y, \theta_v, \phi_v, \theta_i, \phi_i)$, where $(x, y)$ are surface parametric (texel) coordinates and where $(\theta_v, \phi_v)$ is the view direction and $(\theta_i, \phi_i)$ is the illumination direction.

TensorTextures: Multilinear Image-Based Rendering, M. Alex O. Vasilescu and Demetri Terzopoulos

# TensorTexture: Tensor and N-mode SVD

- A ***tensor*** is a higher order generalization of a vector (1st-order tensor) and a matrix (2nd-order tensor).

$$\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$$

- The *mode-n* vectors are the column vectors of matrix $A(n) \in R^{I_n \times (I_1 \times I_2 \ldots I_{n-1} \times I_{n+1} \ldots I_N)}$ that results from *flattening* the tensor A

- The *mode*-n *product* of a tensor $A \in R^{I_1 \times I_2 \ldots I_n \ldots \times I_N}$ and a matrix $M \in R^{J_n \times I_n}$ is denoted by $A \times_n M$. The result can be expressed in terms of *flattened* matrices as $B(n) = MA(n)$.



TensorTextures: Multilinear Image-Based Rendering, M. Alex O. Vasilescu and Demetri Terzopoulos

# TensorTexture: Tensor and N-mode SVD

- The **N-*mode SVD*** is a generalization of the SVD that orthogonalizes these N spaces and decomposes the tensor as the mode-n product of the N orthogonal spaces:

$$\mathcal{D} = \mathcal{Z} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \ldots \times_n \mathbf{U}_n \ldots \times_N \mathbf{U}_N$$

- Tensor $Z$, known as the *core tensor*, is analogous to the diagonal singular value matrix in conventional matrix SVD.
- $U_n$ contains the orthonormal vectors spanning the column space of the matrix $D(n)$ that results from the mode-n flattening of $D$
- SVD can be rewritten as $\mathbf{D} = \mathbf{S} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2$



TensorTextures: Multilinear Image-Based Rendering, M. Alex O. Vasilescu and Demetri Terzopoulos

# TensorTexture: Tensor and N-mode SVD

- Calculate N-mode SVD

Our **N-mode SVD algorithm** for decomposing $\mathcal{D}$ according to equation (1) is as follows:

1. For $n = 1, \ldots, N$, compute matrix $\mathbf{U}_n$ in (1) by computing the SVD of the flattened matrix $\mathbf{D}_{(n)}$ and setting $\mathbf{U}_n$ to be the left matrix of the SVD.[3]

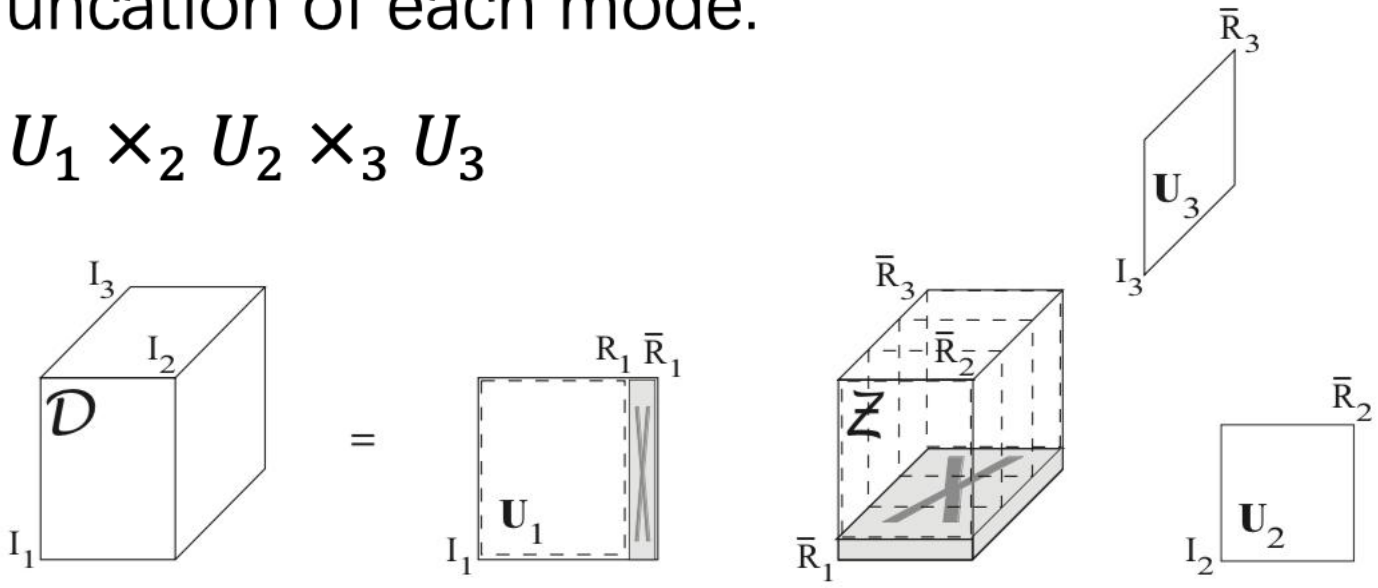
2. If it is needed, solve for the core tensor as follows:

$$\mathcal{Z} = \mathcal{D} \times_1 \mathbf{U}_1^T \times_2 \mathbf{U}_2^T \ldots \times_n \mathbf{U}_n^T \ldots \times_N \mathbf{U}_N^T.$$

TensorTextures: Multilinear Image-Based Rendering, M. Alex O. Vasilescu and Demetri Terzopoulos

# TensorTexture: Tensor and N-mode SVD

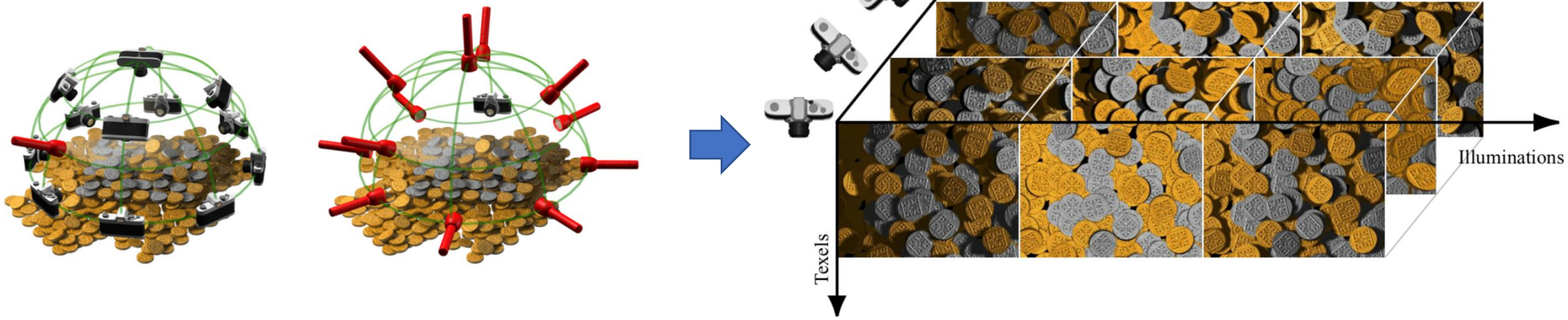
- Analogous to the optimal dimensional reduction in PCA resulting from the truncation of eigenvectors
- Here it admits similar scheme and offers more control, enabling a tailored truncation of each mode.
- $D = Z \times_1 U_1 \times_2 U_2 \times_3 U_3$




TensorTextures: Multilinear Image-Based Rendering, M. Alex O. Vasilescu and Demetri Terzopoulos

# TensorTexture: multilinear rendering

- Given an ensemble of images of a textured surface, we define an image data tensor $D \in R^{T \times I \times V}$
- $V, I$:number of different view and illumination conditions
- $T$: the number of texels in each texture image




$V = 37 \quad I = 21 \quad T = 240 \times 320 \times 3 = 230400$

TensorTextures: Multilinear Image-Based Rendering, M. Alex O. Vasilescu and Demetri Terzopoulos

# TensorTexture: multilinear rendering

- Apply the N-mode SVD algorithm to decompose this tensor $D \in R^{230400\times21\times37}$.

$$\mathcal{D} = \mathcal{Z} \times_1 \mathbf{U}_{\text{texel}} \times_2 \mathbf{U}_{\text{illum}} \times_3 \mathbf{U}_{\text{view}}$$

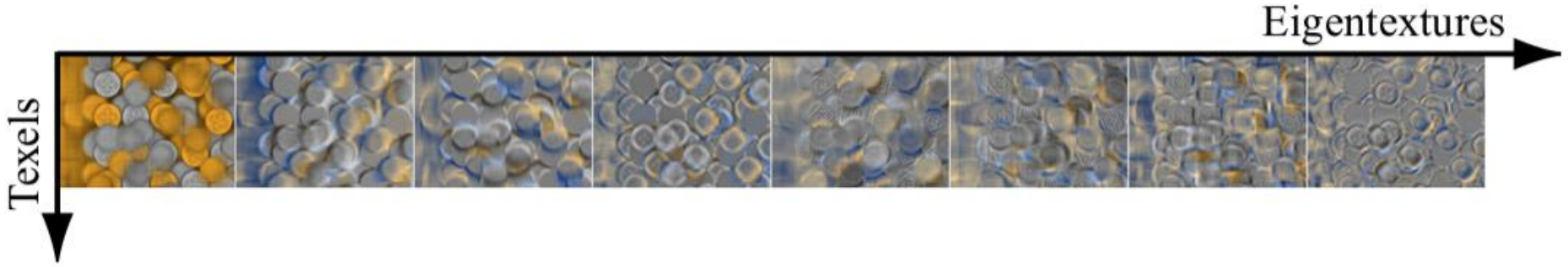
- Analogous to the standard SVD $A = UDV^T$, where the column vectors of $U$ span the view space,
- The column vectors of the 37 × 37 mode matrix $U_{view}$ span the view space. The rows of $U_{view}$ encode an illumination and texel invariant representation for each of the different views.



TensorTextures: Multilinear Image-Based Rendering, M. Alex O. Vasilescu and Demetri Terzopoulos

# TensorTexture: multilinear rendering

- The column vectors of the 230400 × 777 mode matrix $U_{texel}$ span the texel space and are, in fact, the PCA eigenvectors (i.e. "eigentextures" ) since they were computed by performing an SVD on the matrix D(texel) obtained by mode-1 flattening the data tensor D

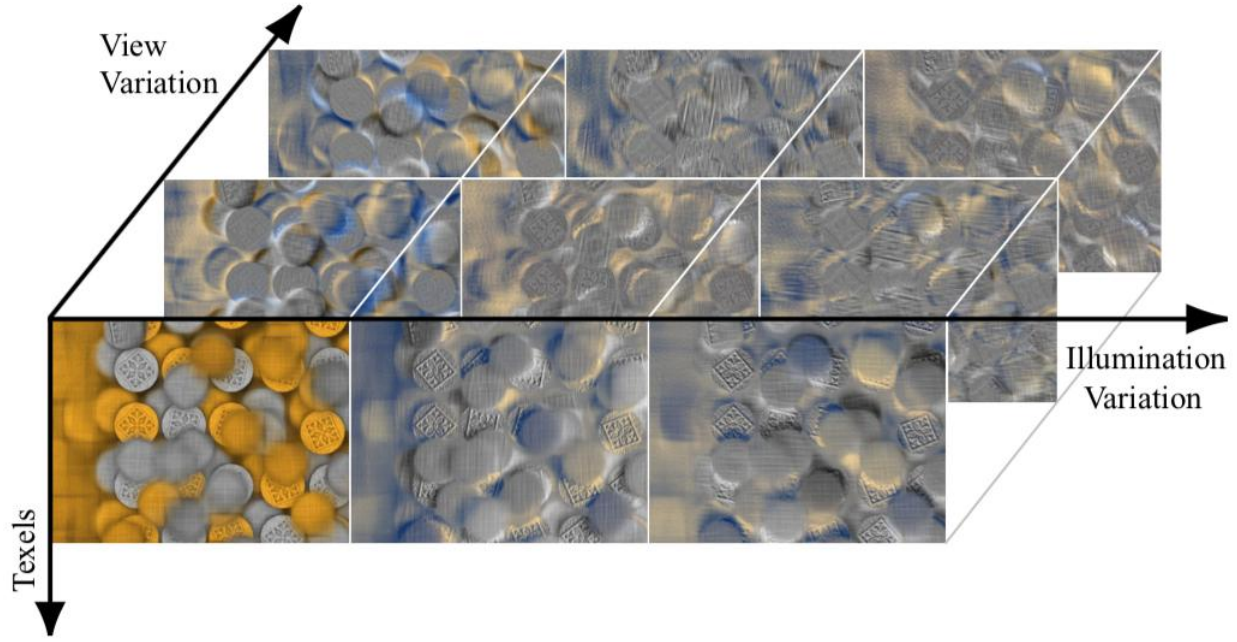



TensorTextures: Multilinear Image-Based Rendering, M. Alex O. Vasilescu and Demetri Terzopoulos

# TensorTexture: multilinear rendering

- TensorTextures models how the appearance of a textured surface varies with view and illumination:

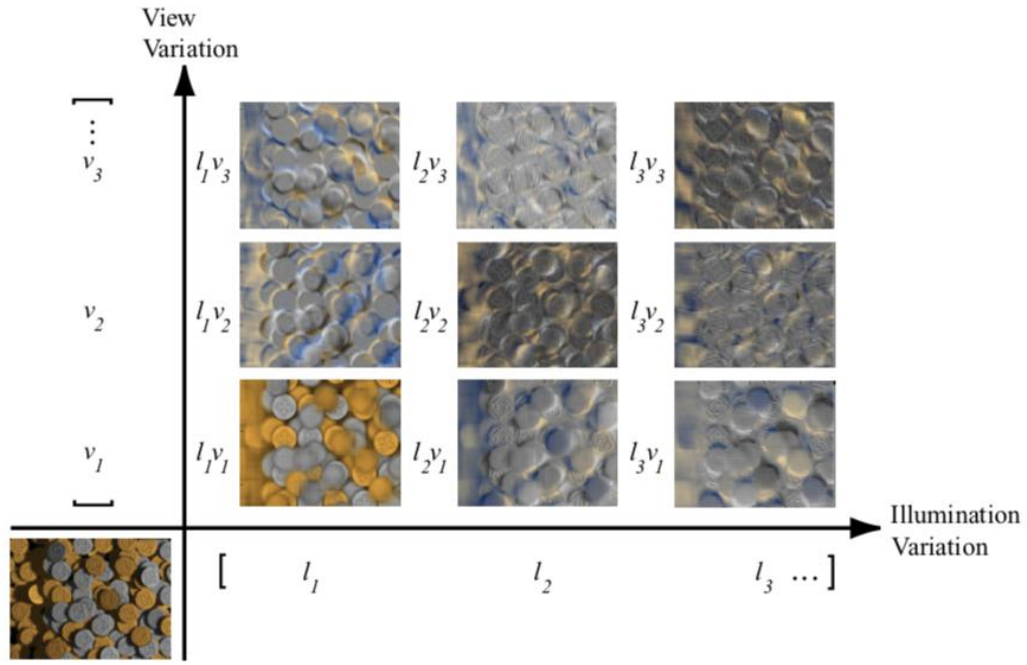
$$\begin{aligned} \mathcal{T} &= \mathcal{Z} \times_1 \mathbf{U}_{\text{texel}} \\ &= \mathcal{D} \times_2 \mathbf{U}_{\text{illum}}^T \times_3 \mathbf{U}_{\text{view}}^T \end{aligned}$$




TensorTextures: Multilinear Image-Based Rendering, M. Alex O. Vasilescu and Demetri Terzopoulos

# TensorTexture: multilinear rendering

- It decomposes the image ensemble into 37 × 21 basis vectors of the same dimension, and represents each image by two coefficient vectors, one of length 37 to encode the view and the other of length 21 to encode the illumination.




TensorTextures: Multilinear Image-Based Rendering, M. Alex O. Vasilescu and Demetri Terzopoulos

# TensorTexture: multilinear rendering

- TensorTexture basis leads to a straightforward rendering algorithm. To render an image **d**, compute

$$\mathbf{d} = \mathcal{T} \times_2 \mathbf{l}^T \times_3 \mathbf{v}^T$$

- where **v** and **l** are the view and illumination representation.

- Given a novel view direction, find the three nearest observed view and represent the novel view as a convex combination of them.



TensorTextures: Multilinear Image-Based Rendering, M. Alex O. Vasilescu and Demetri Terzopoulos

# Reference


- https://pdfs.semanticscholar.org/6d92/8a2f8fbf93812b03d077508dfa9c7dbda4d1.pdf. 3d animation compression.

- https://dl.acm.org/citation.cfm?id=1015725 TensorTexture.

- https://www.google.com.hk/search?safe=active&biw=1220&bih=738&ei=Uf74W62hJcrYvgSXgamYDg&q=svd+for+graphics&oq=svd+for+&gs_l=psy-ab.3..35i39l2j0i20i263j0l7.46.4126..5499...8.0..0.490.5002.4-11.....0....1..gws-wiz.....6..0i67j0i131.4hKaXXRsPVw SVD and Its Applications

- https://www.cs.tau.ac.il/~dcor/Graphics/cg-slides/3d_geometry_lesson2.pdf 3d geometry for computer graphics