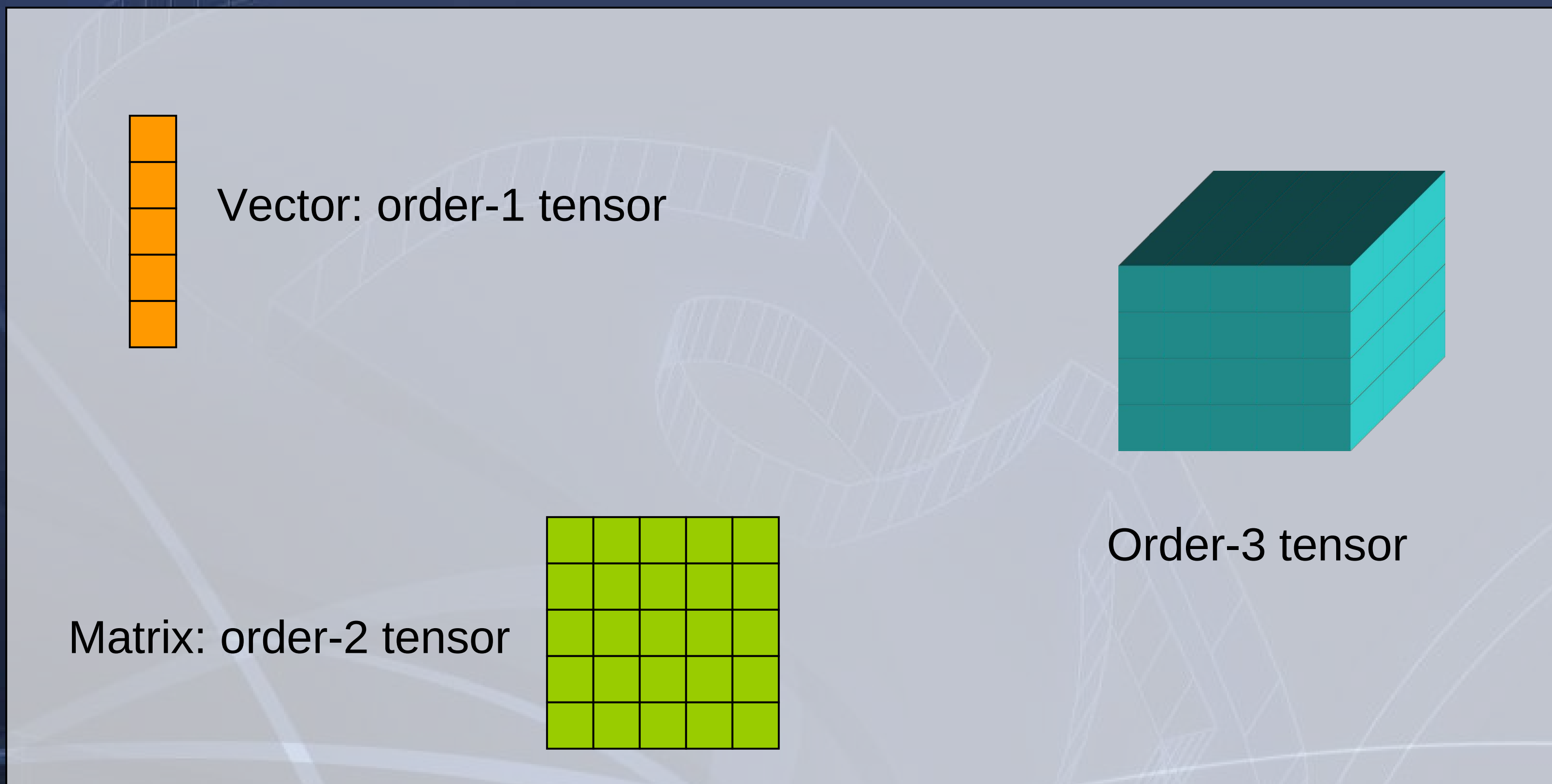
Tensors and Component Analysis

Musawir Ali

Tensor: Generalization of an n-dimensional array



Reshaping Tensors

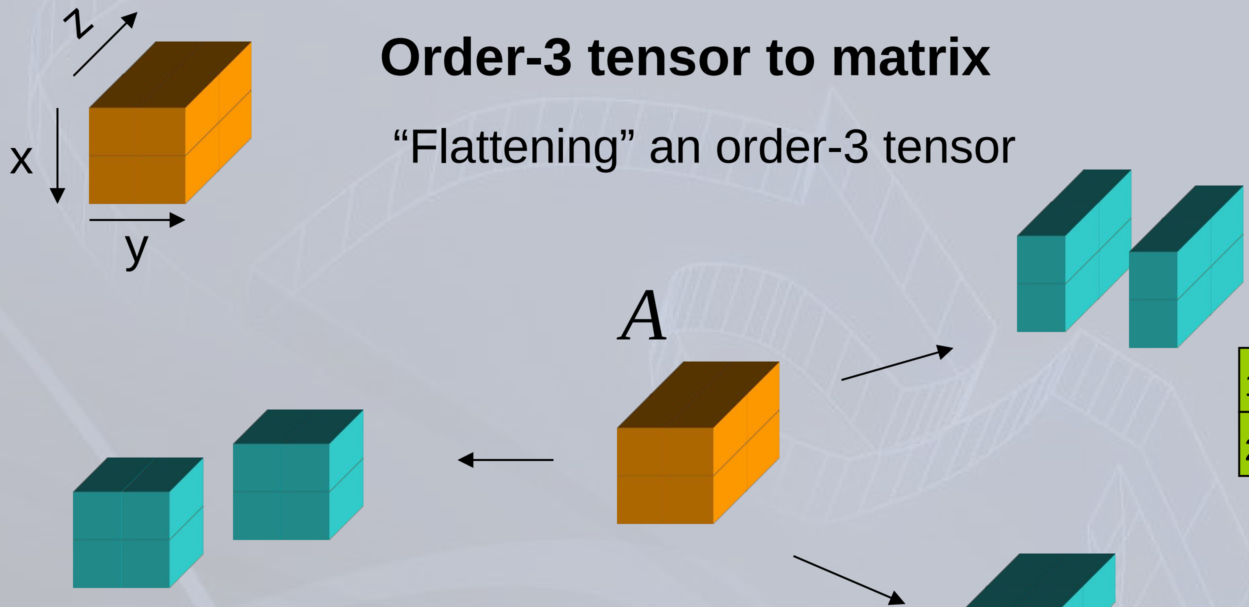
Matrix to vector

"Vectorizing" a matrix

Reshaping Tensors

Order-3 tensor to matrix

"Flattening" an order-3 tensor



$A_{(1)}$

1,1,1	1,1,2	1,2,1	1,2,2
2,1,1	2,1,2	2,2,1	2,2,2

$A_{(2)}$

1,1,1	2,1,1	1,1,2	2,1,2
1,2,1	2,2,1	1,2,2	2,2,2

$A_{(3)}$

1,1,1	1,2,1	2,1,1	2,2,1
1,1,2	1,2,2	2,1,2	2,2,2

n-Mode Multiplication

Multiplying Matrices and order-3 tensors

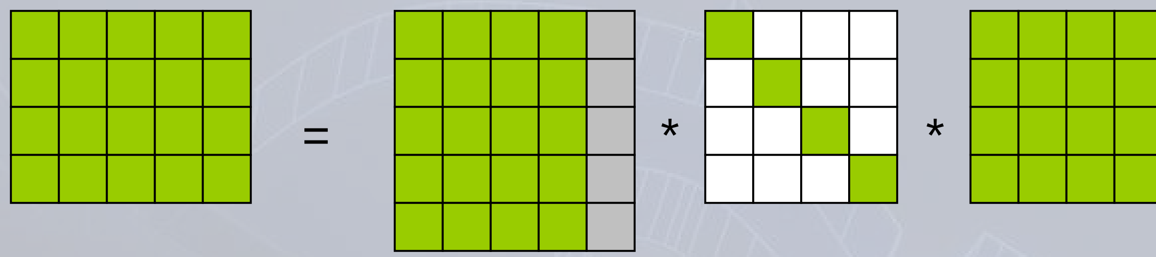
$$\mathcal{A} \times_n M = M A_{(n)}$$

Multiplying a tensor $\mathcal{A}$ with matrix M

$\mathcal{A}$ is flattened over dimension n (permute dimensions so that the dimension n is along the columns, and then flatten), and then regular matrix multiplication of matrix M and flattened tensor $A_{(n)}$ is performed

Singular Value Decomposition (SVD)

SVD of a Matrix



M = U S V^T

U and V are orthogonal matrices, and S is a diagonal matrix consisting of singular values.

Singular Value Decomposition (SVD)

SVD of a Matrix: observations

$$M = U\ S\ V^T$$

Multiply both sides by M^T

Multiplying on the left

$$M^TM = (U\ S\ V^T)^T\ U\ S\ V^T$$

$$M^TM = (V\ S\ U^T)\ U\ S\ V^T$$

$$U^TU = I$$

$$M^TM = V\ S^2\ V^T$$

Multiplying on the right

$$MM^T = U\ S\ V^T\ (U\ S\ V^T)^T$$

$$MM^T = U\ S\ V^T\ (V\ S\ U^T)$$

$$V^TV = I$$

$$MM^T = U\ S^2\ U^T$$

Singular Value Decomposition (SVD)

SVD of a Matrix: observations

$M^TM = V\ S^2\ V^T$ ⟵ diagonalizations ⟶ $MM^T = U\ S^2\ U^T$

Diagonalization of a Matrix: (finding eigenvalues)

$A = W\ \Lambda\ W^T$

where:

- A is a square, symmetric matrix
- Columns of W are eigenvectors of A
- Λ is a diagonal matrix containing the eigenvalues

Therefore, if we know U (or V) and S, we basically have found out the eigenvectors and eigenvalues of MM^T (or M^TM) !

Principal Component Analysis (PCA)

What is PCA?

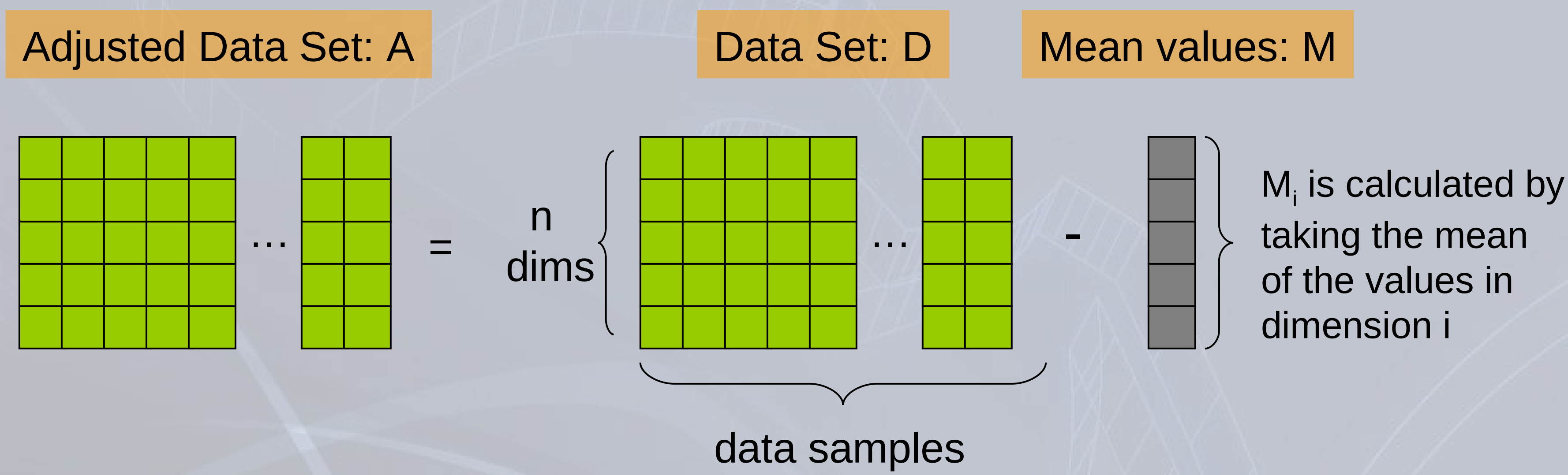
- Analysis of n-dimensional data
- Observes correspondence between different dimensions
- Determines principal dimensions along which the variance of the data is high

Why PCA?

- Determines a (lower dimensional) basis to represent the data
- Useful compression mechanism
- Useful for decreasing dimensionality of high dimensional data

Principal Component Analysis (PCA)

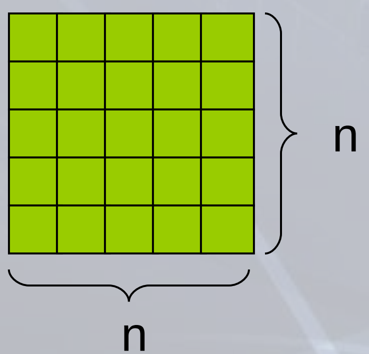
Steps in PCA: #1 Calculate Adjusted Data Set



Principal Component Analysis (PCA)

Steps in PCA: #2 Calculate Co-variance matrix, C, from Adjusted Data Set, A

Co-variance Matrix: C



C_{ij} = cov(i,j)

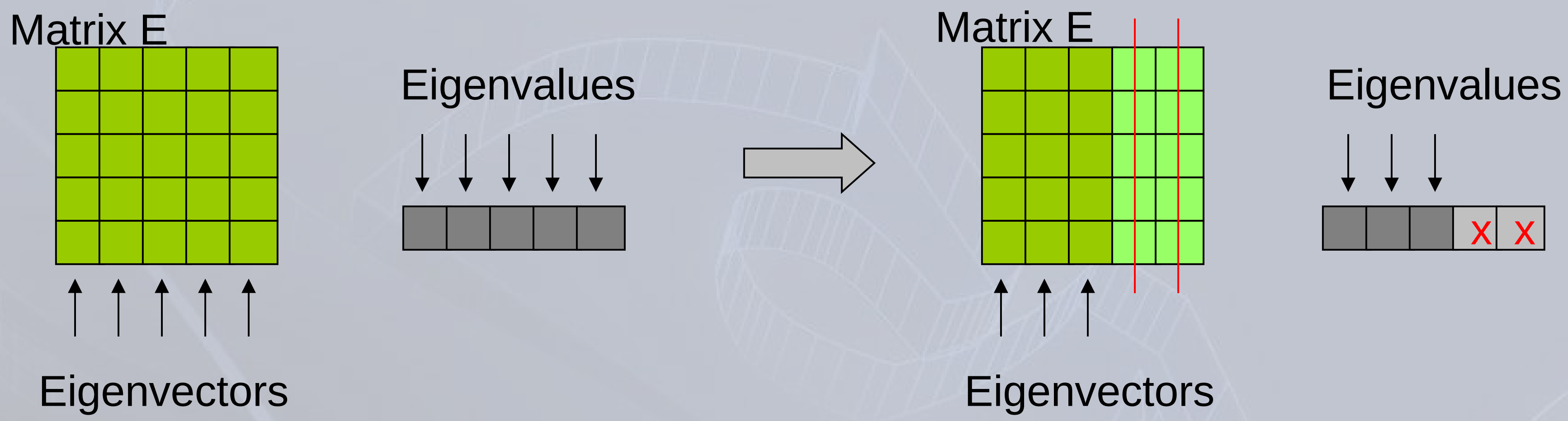
$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)}$$

Note: Since the means of the dimensions in the adjusted data set, A, are 0, the covariance matrix can simply be written as:

$$C = (A\,A^T) / (n-1)$$

Principal Component Analysis (PCA)

Steps in PCA: #3 Calculate eigenvectors and eigenvalues of C



If some eigenvalues are 0 or very small, we can essentially discard those eigenvalues and the corresponding eigenvectors, hence reducing the dimensionality of the new basis.

Principal Component Analysis (PCA)

Steps in PCA: #4 Transforming data set to the new basis

$F = E^T A$

where:

- F is the transformed data set
- E^T is the transpose of the E matrix containing the eigenvectors
- A is the adjusted data set

Note that the dimensions of the new dataset, F, are less than the data set A

To recover A from F:

$(E^T)^{-1}F = (E^T)^{-1}E^T A$

$(E^T)^T F = A$

$EF = A$

* E is orthogonal, therefore $E^{-1} = E^T$

PCA using SVD

Recall: In PCA we basically try to find eigenvalues and eigenvectors of the covariance matrix, C. We showed that $C = (AA^T) / (n-1)$, and thus finding the eigenvalues and eigenvectors of C is the same as finding the eigenvalues and eigenvectors of AA^T

Recall: In SVD, we decomposed a matrix A as follows:

$$A = U S V^T$$

and we showed that:

$$AA^T = U S^2 U^T$$

where the columns of U contain the eigenvectors of AA^T and the eigenvalues of AA^T are the squares of the singular values in S

Thus SVD gives us the eigenvectors and eigenvalues that we need for PCA

N-Mode SVD

Generalization of SVD

Order-2 SVD: $\longrightarrow$ $D = U\,S\,V^T$

Written in terms of mode-n product: $D = S \times_1 U \times_2 V$

By definition of mode-n multiplication:

$D = U\,(\,V\,S_{(2)}\,)_{(1)}$

$D = U\,(\,V\,S\,)^T$

$D = U\,S^T\,V^T$

$D = U\,S\,V^T$

Note: $S^T = S$ since S is a diagonal matrix

Note: $D = S \times_1 U \times_2 V = S \times_2 V \times_1 U$

N-Mode SVD

Generalization of SVD

Order-3 SVD: $\longrightarrow$ $\boldsymbol{D} = \boldsymbol{Z} \times_1 U_1 \times_2 U_2 \times_3 U_3$

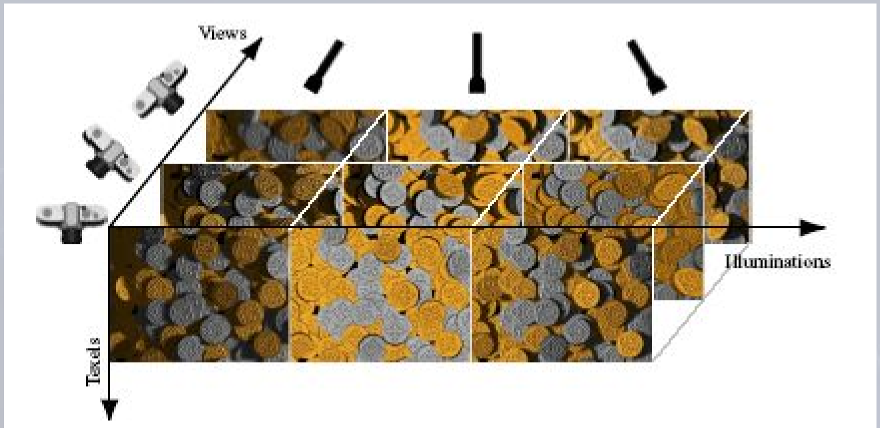
- $\boldsymbol{Z}$, the core tensor, is the counterpart of S in Order-2 SVD
- U_1, U_2, and U_3 are known as mode matrices

Mode matrix U_i is obtained as follows:

- perform Order-2 SVD on $D_{(i)}$, the matrix obtained by flattening the tensor $\boldsymbol{D}$ on the i'th dimension.
- U_i is the leftmost (column-space) matrix obtained from the SVD above

PCA on Higher Order Tensors

Bidirectional Texture Function (BTF) as a Order-3 Tensor



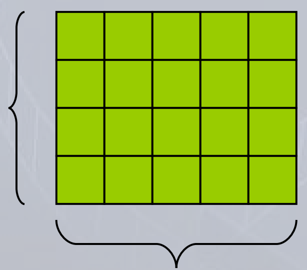
Changing illumination, changing view, and changing texels (position) along respective dimensions of the tensor

$$\boldsymbol{D} = \boldsymbol{Z} \times_1 U_t \times_2 U_i \times_3 U_v$$

PCA on Higher Order Tensors

Performing PCA on a Order-3 Tensor

texels



view + illumination

$\xrightarrow{\text{PCA}}$

U_t

- The matrix, U_t (shown above), resulting from the PCA, consists of the eigenvectors (eigentextures) along its columns

- Transform data to new basis:

  $F = U_t^T A$

where A is the data in original basis

- Retrieving/Rendering image:

  $\boldsymbol{T} = \boldsymbol{Z} \times_1 U_t$

  $\text{Image} = \boldsymbol{T} \times_1 F$

TensorTextures

A cheaper equivalent of PCA: TensorTextures

PCA: $\boldsymbol{T} = \boldsymbol{Z} \times_1 U_t$ ⟺ TensorTextures: $\boldsymbol{T} = \boldsymbol{D} \times_2 U_i^T \times_3 U_v^T$

Recall: $\boldsymbol{D} = \boldsymbol{Z} \times_1 U_t \times_2 U_i \times_3 U_v$

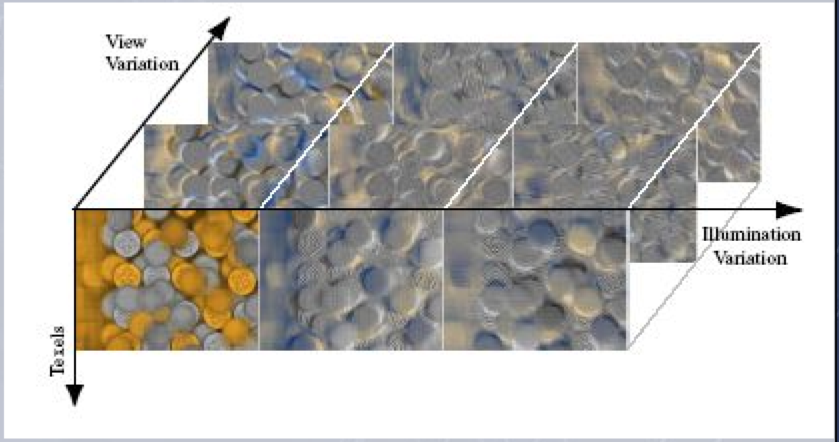
$\boldsymbol{D} = U_v (U_i (U_t Z_{(1)})_{(2)})_{(3)}$

$\boldsymbol{D} = (U_v U_i) (U_t Z_{(1)})_{(2)}$

$(U_v U_i)^{-1} D_{(2)} = U_t Z_{(1)}$

$(U_i^T U_v^T) D_{(3)} = Z \times_1 U_t$

$\boldsymbol{D} \times_2 U_i^T \times_3 U_v^T = Z \times_1 U_t$



TensorTextures

Advantages of TensorTextures over PCA

PCA: $\boldsymbol{T} = \boldsymbol{Z} \times_1 U_t$

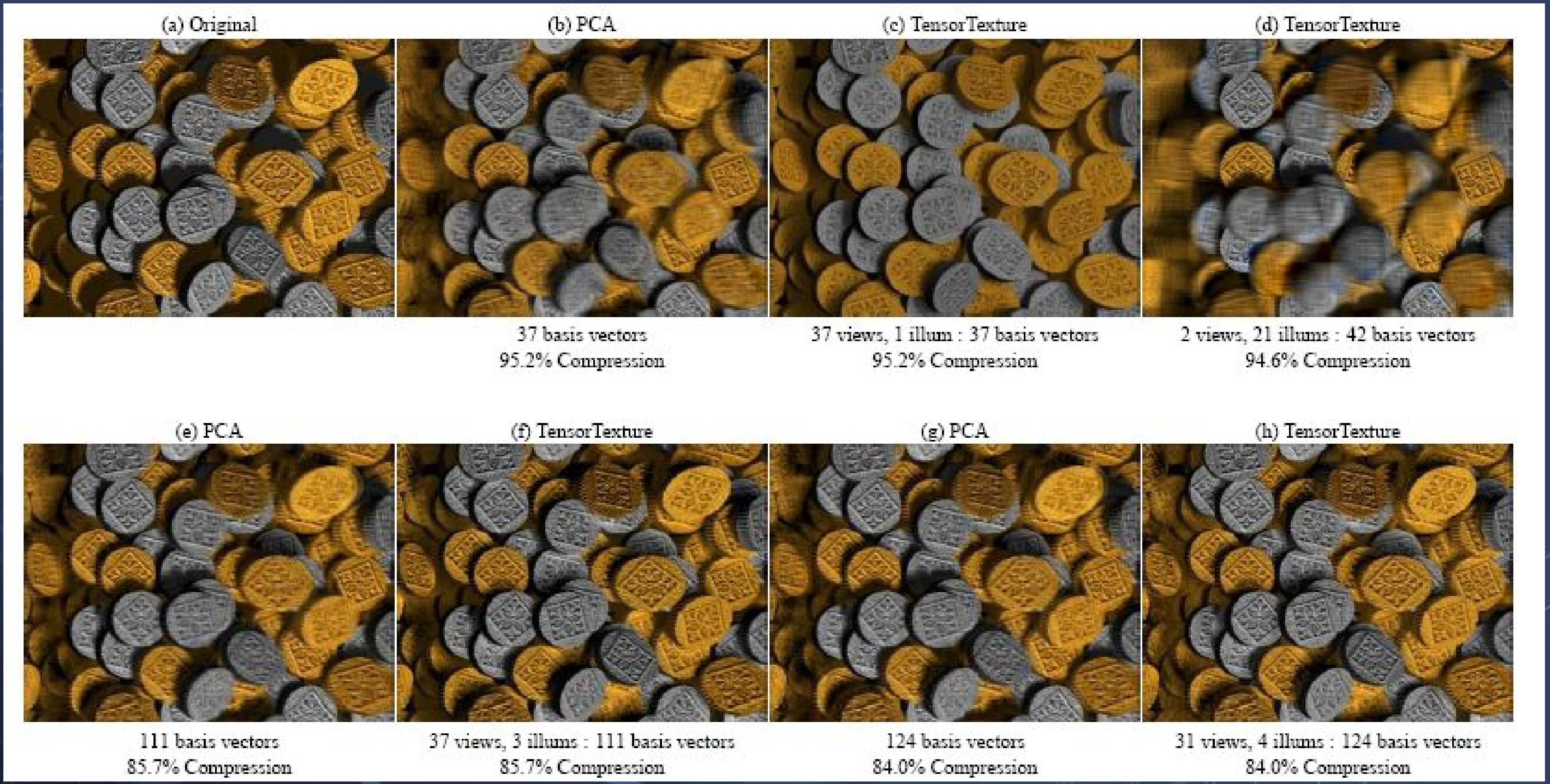
TensorTextures: $\boldsymbol{T} = \boldsymbol{D} \times_2 U_i^T \times_3 U_v^T$

- Independent compression control of illumination and view in TensorTextures, whereas in PCA, the illumination and view are coupled together and thus change/compression of eigenvectors will effect both parameters.

- Images represented using fewer coefficients in TensorTextures.
    - PCA: (v * i) basis vectors, each of size t
    - TensorTextures: (v + i) basis vectors, of size v and I

where v = # of view directions, i = # of illumination conditions, and t = # of texels in each image used in the creation of the tensor

TensorTextures Compression vs PCA compression



Notice that PCA has one set of basis vectors where as TensorTextures has two: one for illumination, and one for view. This enables selective compression to a desired level of tolerance of perceptual error