Tutorial: Tensor Approximation in Visualization and Graphics

Implementation Examples in Scientific Visualization

Renato Pajarola, **Susanne K. Suter**, and Roland Ruiters

University of Zurich[UZH]

VISUALIZATIONAND MULTIMEDIALAB

universitätbonn

Institute of Computer Science II
Computer Graphics

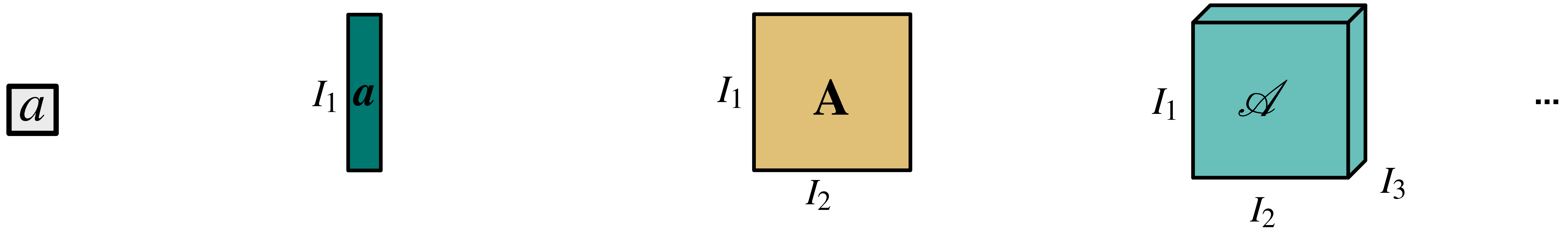
Outline

- Tensor classes in MATLAB and vmmlib
  - Downloads:
    - http://www.sandia.gov/~tgkolda/TensorToolbox
    - https://github.com/VMML/vmmlib
  - Typical tensor operations
  - Toy examples (see folder: vmmlib_ta_demo)
  - Test dataset (see folder: vmmlib_ta_demo)
- GPU-based tensor reconstruction

Typical TA Operations

- Create a tensor (memory mapping)
- Unfolding
- TTM
  - core generation vs. reconstruction
- Create tensor models (Tucker, CP)
- Algorithms (HOSVD, HOOI, HOPM)

Tensor: A Multidimensional Array



a scalar	1^{st} order tensor or vector	2^{nd} order tensor or matrix	3^{rd} order tensor or volume
	$i_1 = 1, \ldots, I_1$	$i_2 = 1, \ldots, I_2$	$i_3 = 1, \ldots, I_3$

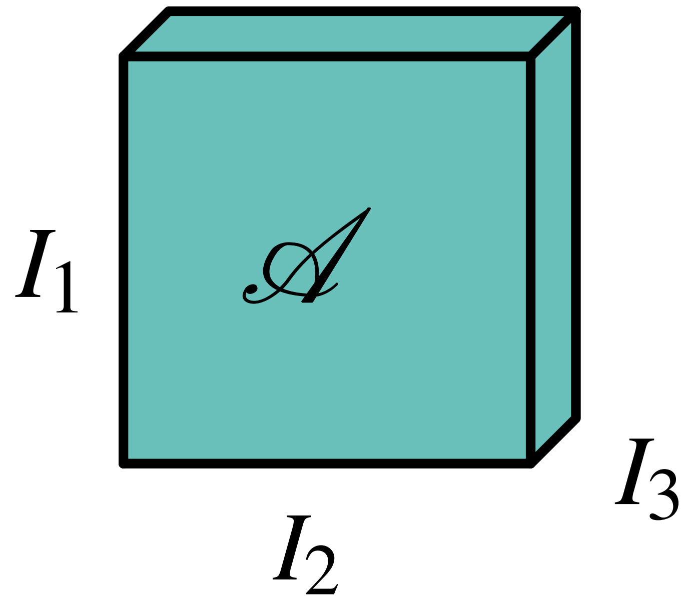
- MATLAB: N-way tensor
  - M = ones(4,3,2); (A 4 x 3 x 2 array)
  - A = tensor(M,[2 3 4]); (M has 24 elements)
  - A = tenones([3 4 2]);
  - A = tenrand([4 3 2]);
- For details on the MATLAB tensor toolbox see toolbox documentation

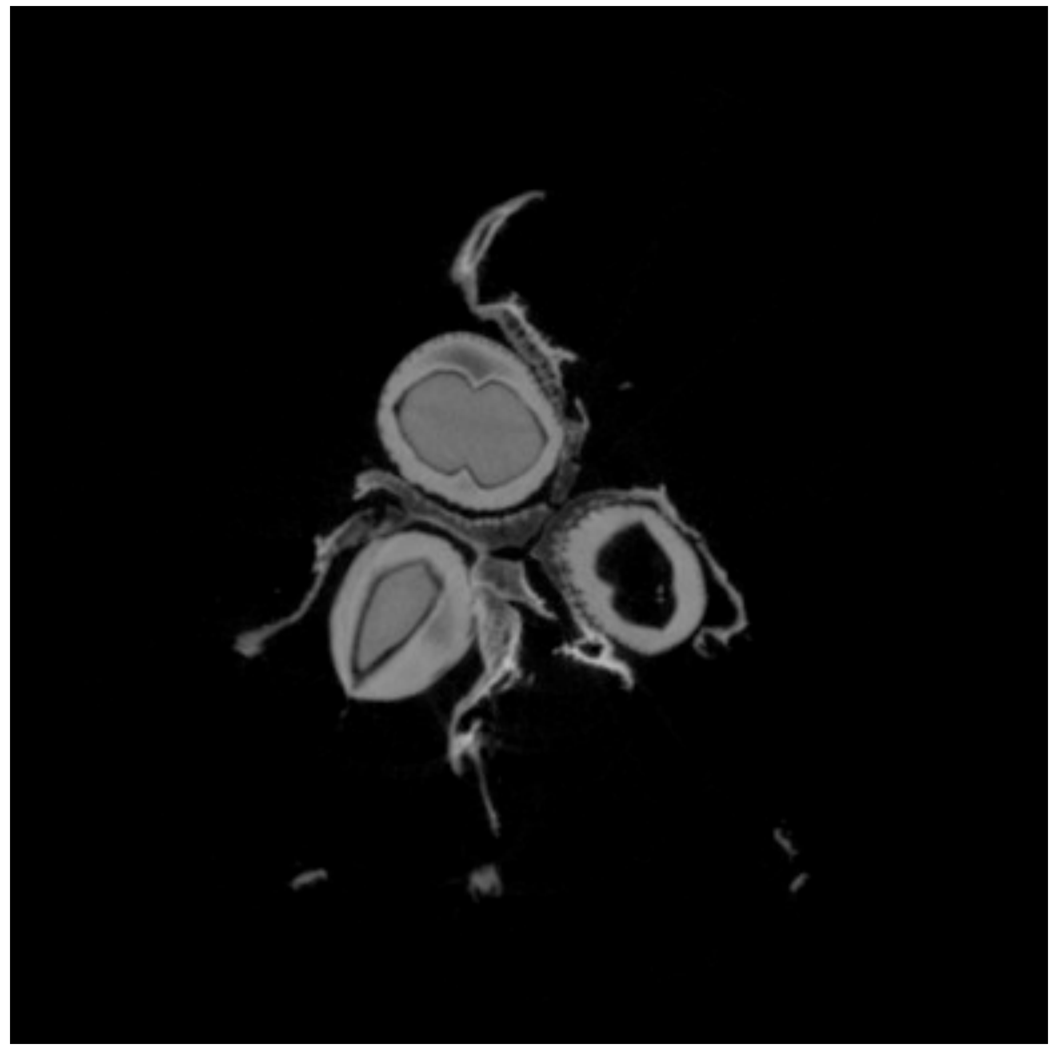
```
X is a tensor of size
2 x 3 x 4

X(:,:,1) =            X(:,:,3) =

   1   1   1             1   1   1

   1   1   1             1   1   1

X(:,:,2) =            X(:,:,4) =

   1   1   1             1   1   1

   1   1   1             1   1   1
```

VISUALIZATION AND MULTIMEDIA LAB

universität bonn

Test Dataset: Hazelnut

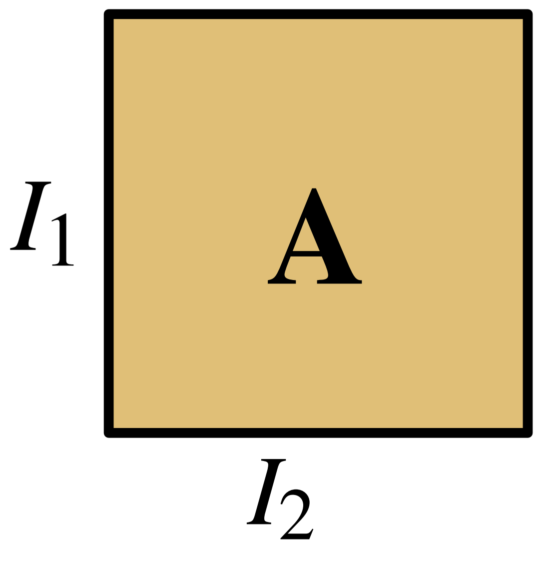




- A microCT scan of a dried hazelnut (acquired at the UZH)
- $I_1 = I_2 = I_3 = 512$
- Values: unsigned char (8bit)

A Matrix in vmmlib

[vmmlib]



$i_2 = 1, \ldots, I_2$

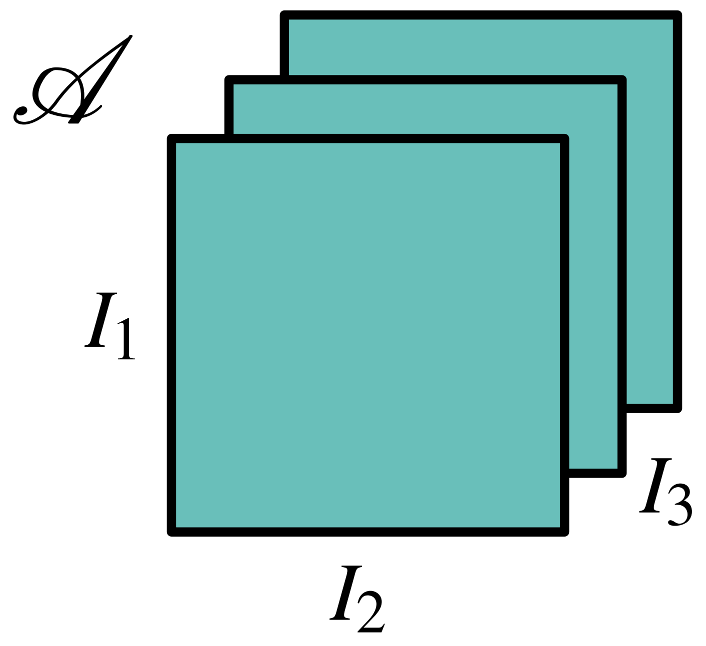
```
matrix< I1, I2, Type >

matrix< 4, 3, unsigned char > m;
```

```
Example:
(0, 1, 2)
(3, 4, 5)
(6, 7, 8)
(9, 10, 11)
```

- I_1 (M) rows
- I_2 (N) columns
- The matrices are per default column-major ordered
- A matrix is an array of I_2 (N) columns, where each column is of size I_1 (M)

A Tensor3 in vmmlib

[vmmlib]



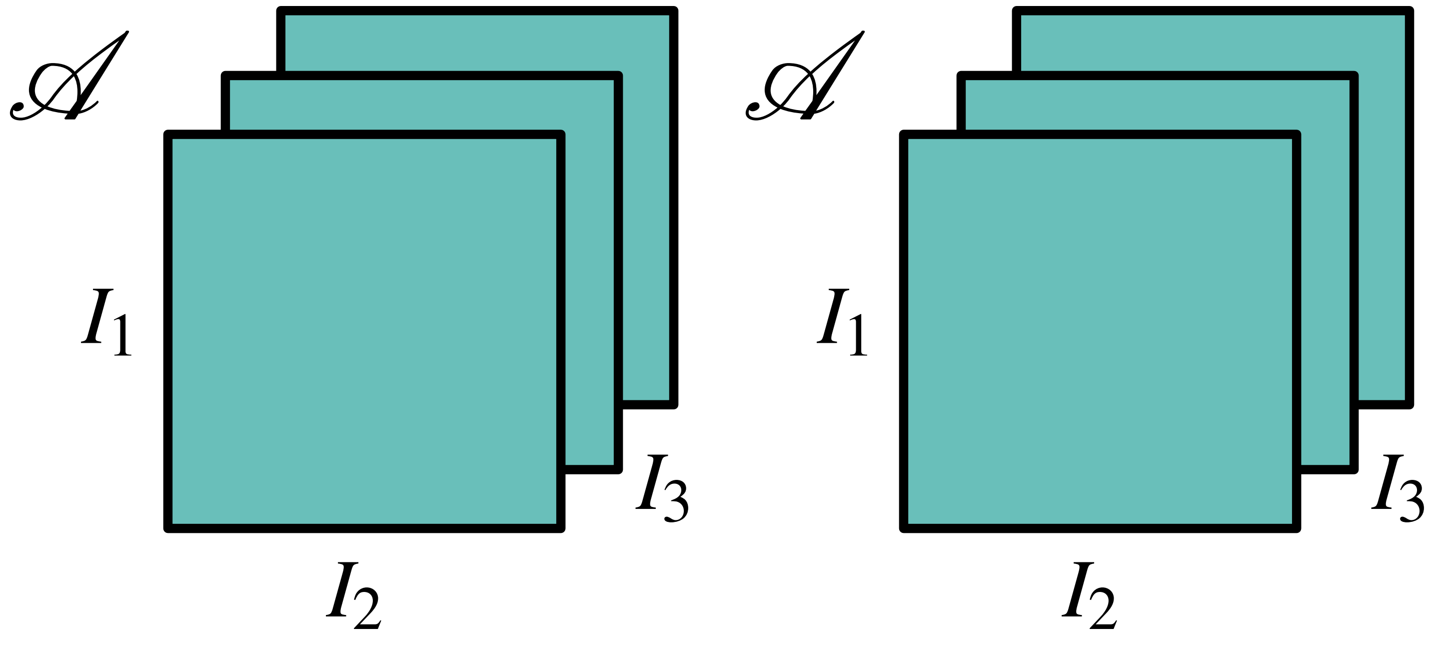

```
tensor3< I1, I2, I3, Type >

tensor3< 4, 3, 2, unsigned char > t3;
```

```
Example:
(0, 1, 2)
(3, 4, 5)
(6, 7, 8)
(9, 10, 11)
 ***
(12, 13, 14)
(15, 16, 17)
(18, 19, 20)
(21, 22, 23)
 ***
```

- A tensor3 A in vmmlib is an array of I_3 matrices each of size I_1 times I_2
- The matrices are per default column-major ordered
- For each tensor3, the explicit size and the type of the values is requested
- A tensor3 is internally allocated and deallocated as pointer while the matrices are not

A Tensor4 in vmmlib

[vmmlib]



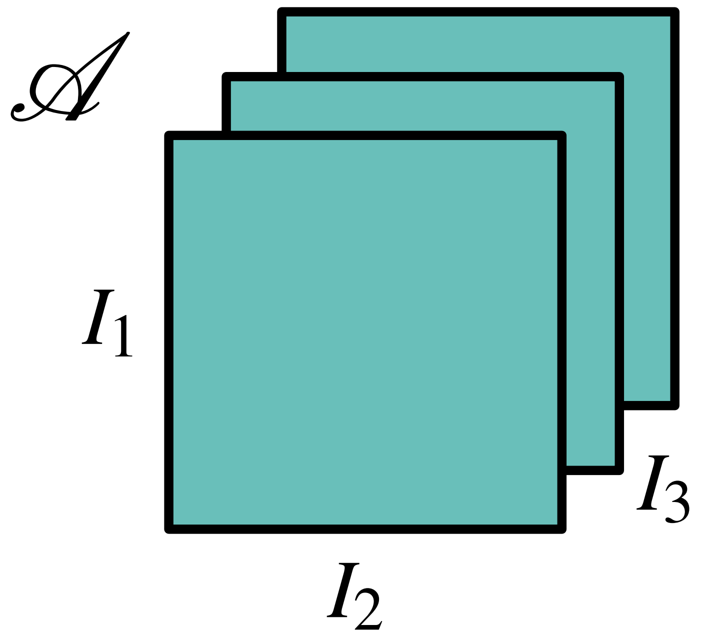
```
tensor4< I1, I2, I3, I4, Type >

tensor4< 4, 3, 2, 2, unsigned char > t4;
```

```
Example:
(0, 1, 2)
(3, 4, 5)
(6, 7, 8)
(9, 10, 11)
 ***
(12, 13, 14)
(15, 16, 17)
(18, 19, 20)
(21, 22, 23)
 ***
----
(24, 25, 26)
...
```

- A tensor4 in vmmlib is an array of I_4 tensor3s
- For each tensor4, the explicit size and the type of the values is requested

Large Data Tensors (in vmmlib)

[vmmlib]

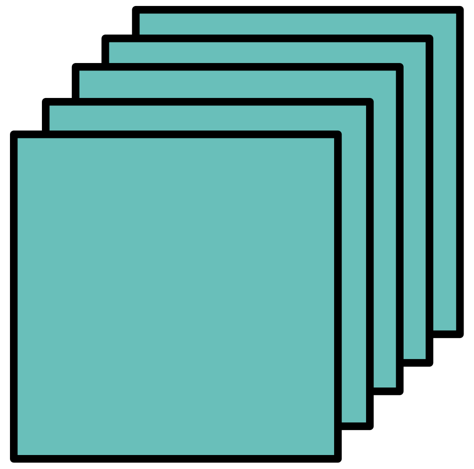


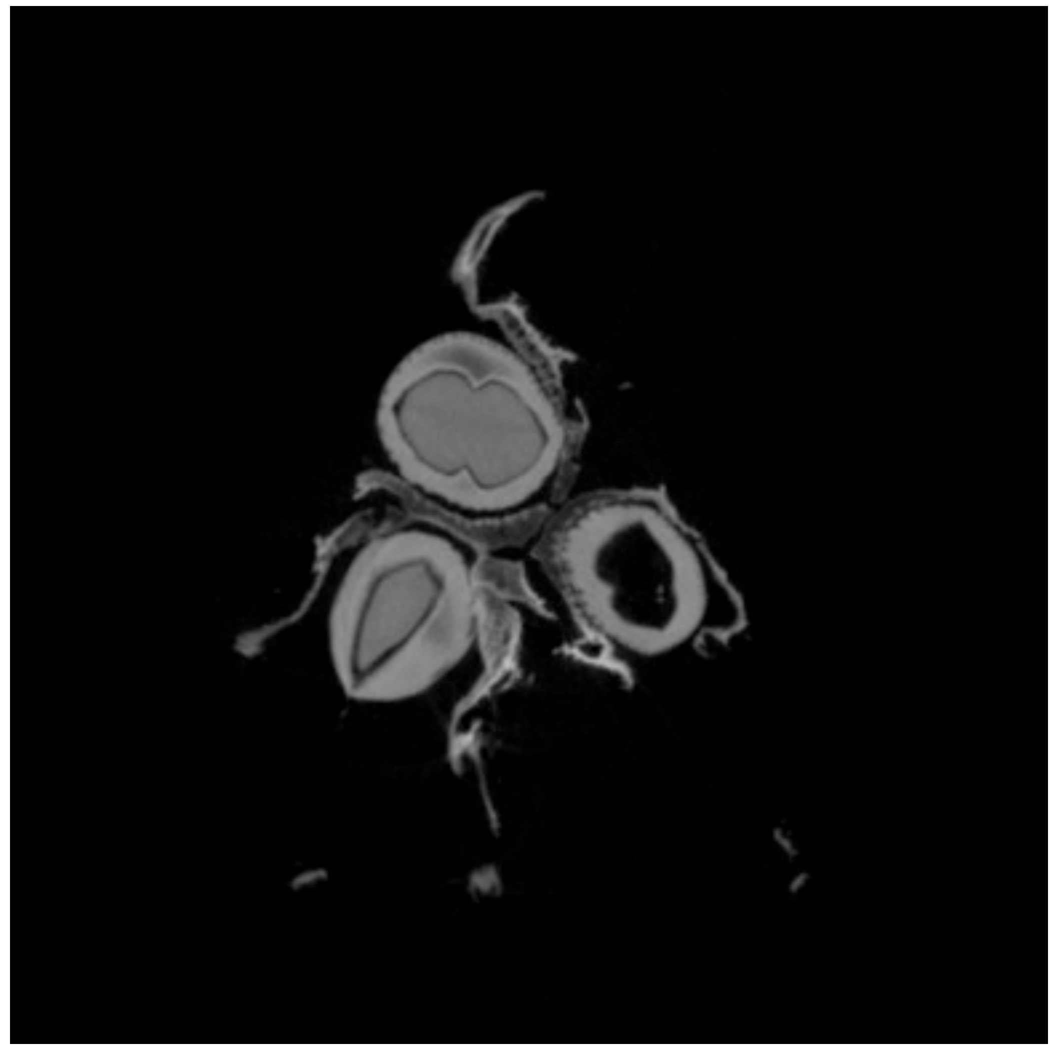

```
const size_t d = 512;
typedef tensor3< d,d,d, unsigned char > t3_512u_t;
typedef t3_converter< d,d,d, unsigned char > t3_conv_t;
typedef tensor_mmapper< t3_512u_t, t3_conv_t > t3map_t;

std::string in_dir = "./dataset";
std::string file_name = "hnut512_uint.raw";
t3_512u_t t3_hazelnut;
t3_conv_t t3_conv;

t3map_t t3_mmap( in_dir, file_name, true, t3_conv ); //true -> read-only
t3_mmap.get_tensor( t3_hazelnut );
```

Get Slices of a Tensor3

[vmmlib]

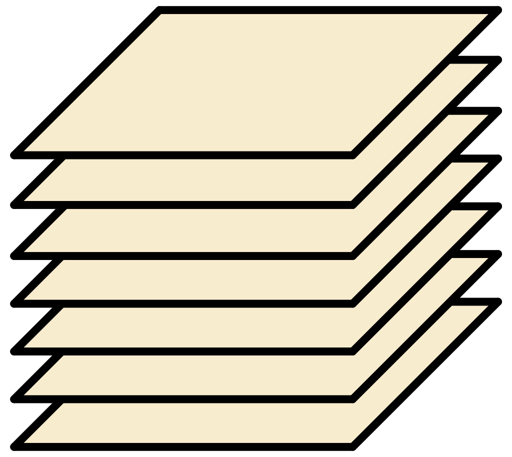
frontal slices

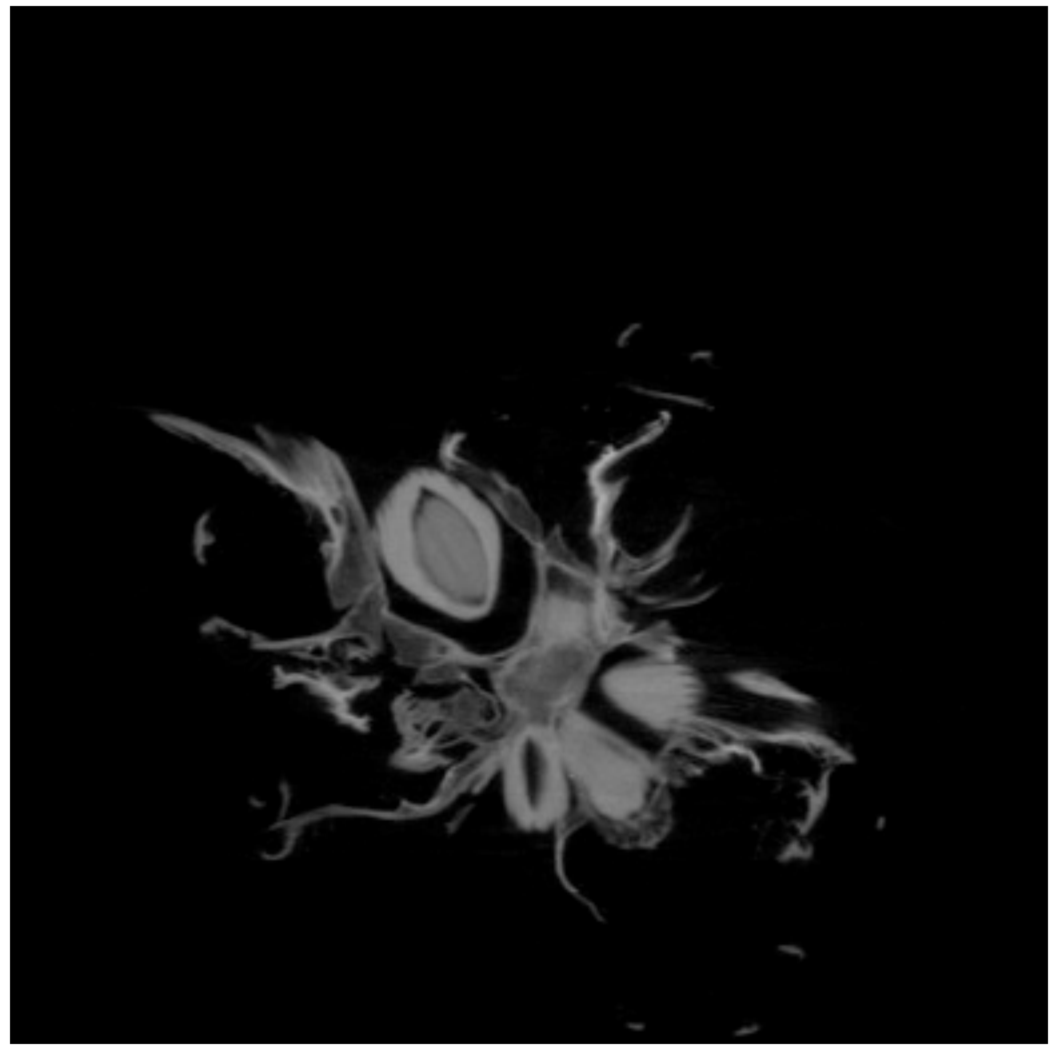




```
matrix< 512, 512, values_t > slice;
t3.get_frontal_slice_fwd( 256, slice );
```

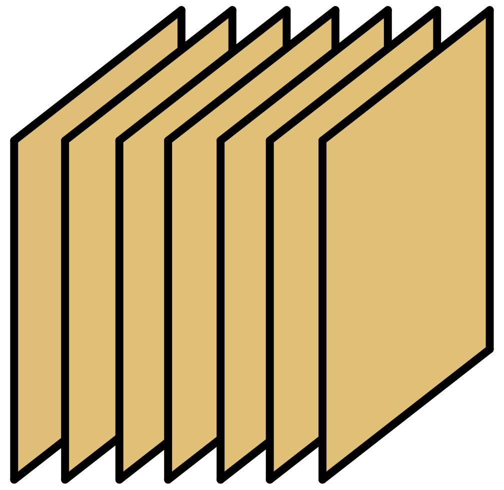
horizontal slices

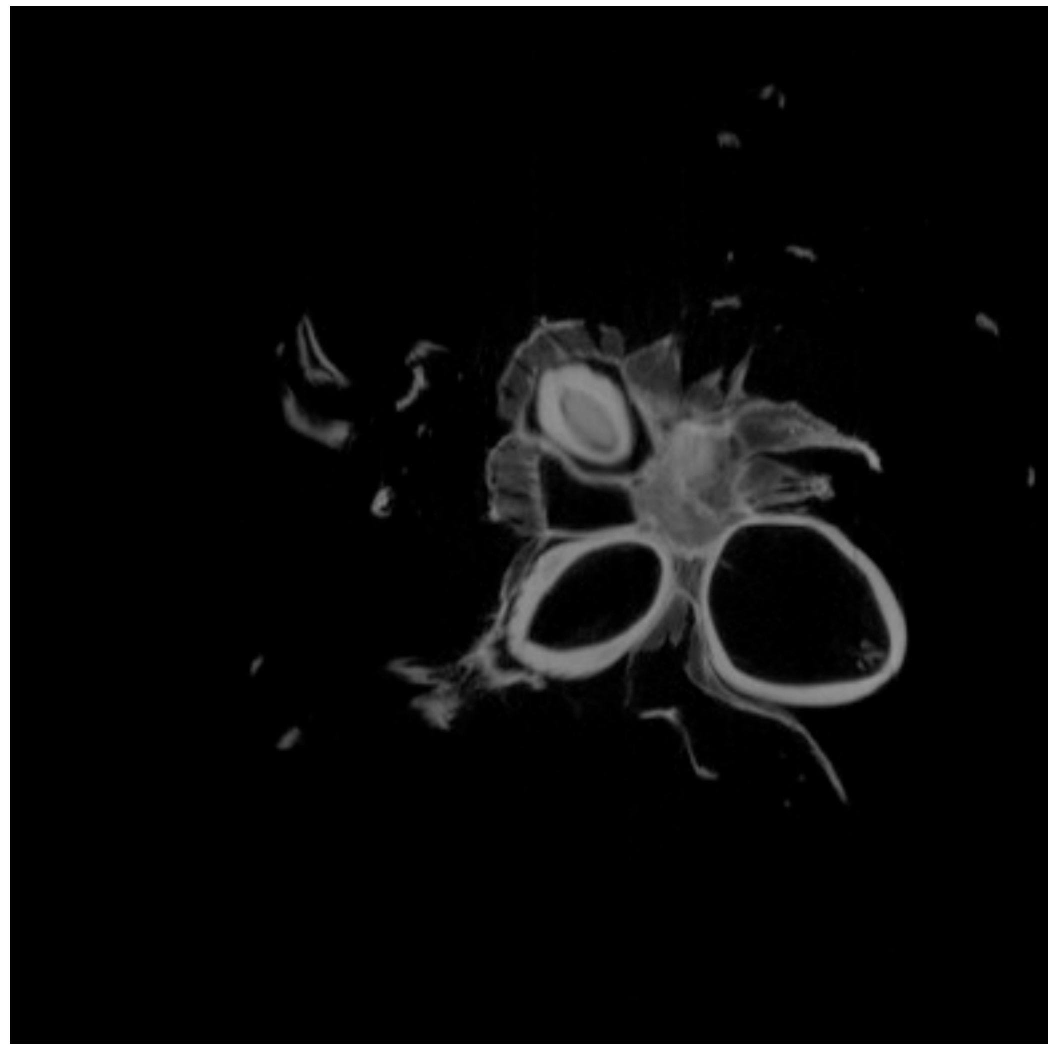




```
matrix< 512, 512, values_t > slice;
t3.get_horizontal_slice_fwd( 256, slice );
```

lateral slices





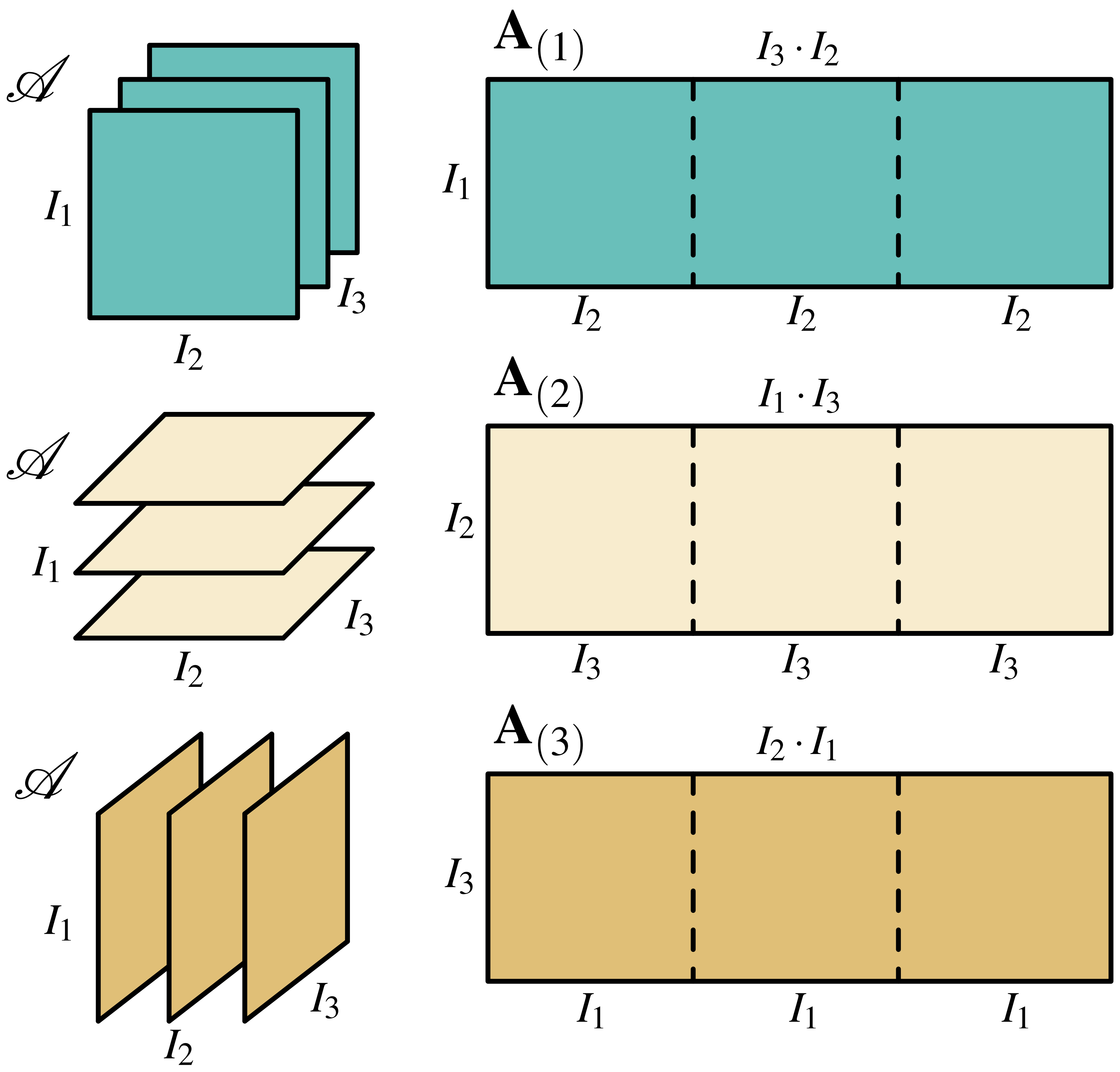
```
matrix< 512, 512, values_t > slice;
t3.get_lateral_slice_fwd( 256, slice );
```

Forward Tensor Unfolding (Matricization)

[vmmlib]

Forward Cyclic Unfolding

after [Kiers, 2000]



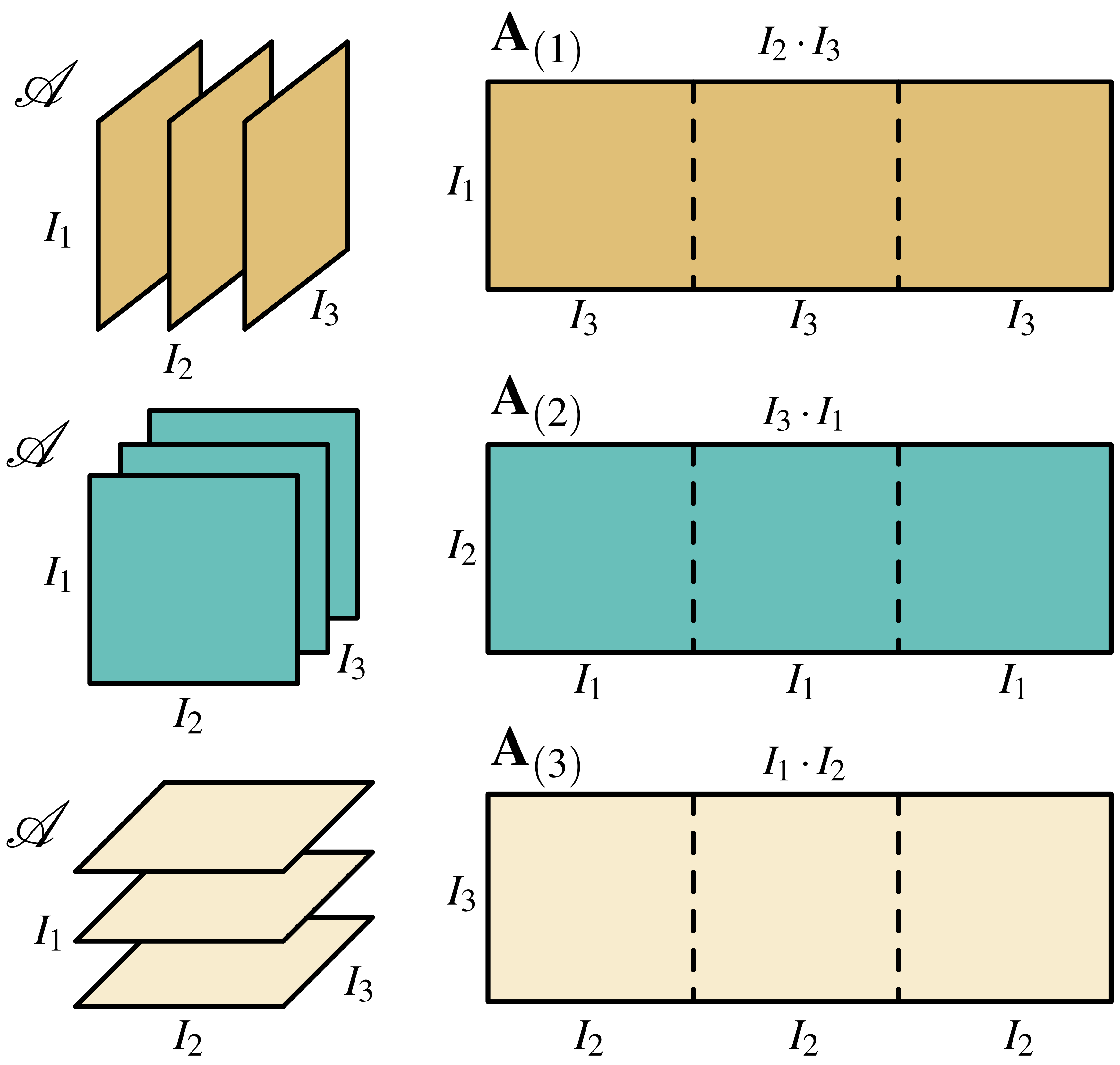

```
tensor3< I1, I2, I3, values_t > t3
matrix< I1, I3*I2, values_t > unf_front_fwd;
t3.frontal_unfolding_fwd( unf_front_fwd );

        forward unfolded tensor (frontal)
        (0, 1, 2, 12, 13, 14)
        (3, 4, 5, 15, 16, 17)
        (6, 7, 8, 18, 19, 20)
        (9, 10, 11, 21, 22, 23)

matrix< I2, I1*I2, values_t > unf_horiz_fwd;
t3.horizontal_unfolding_fwd( unf_horiz_fwd );

         forward unfolded tensor (horizontal)
         (0, 12, 3, 15, 6, 18, 9, 21)
         (1, 13, 4, 16, 7, 19, 10, 22)
         (2, 14, 5, 17, 8, 20, 11, 23)

matrix< I3, I2*I1, values_t > unf_lat_fwd;
t3.lateral_unfolding_fwd( unf_lat_fwd );

         forward unfolded tensor (lateral)
         (0, 3, 6, 9, 1, 4, 7, 10, 2, 5, 8, 11)
         (12, 15, 18, 21, 13, 16, 19, 22, 14, 17, 20, 23)
```

Backward Tensor Unfolding (Matricization)

[vmmlib]

Backward Cyclic Unfolding

after [De Lathauer et al., 2000a]




```
tensor3< I1, I2, I3, values_t > t3
matrix< I1, I2*I3, values_t > unf_lat_bwd;
t3.lateral_unfolding_bwd( unf_lat_bwd );
```

```
backward unfolded tensor (lateral)
(0, 12, 1, 13, 2, 14)
(3, 15, 4, 16, 5, 17)
(6, 18, 7, 19, 8, 20)
(9, 21, 10, 22, 11, 23)
```

```
matrix< I2, I3*I1, values_t > unf_front_bwd;
t3.frontal_unfolding_bwd( unf_front_bwd );
```

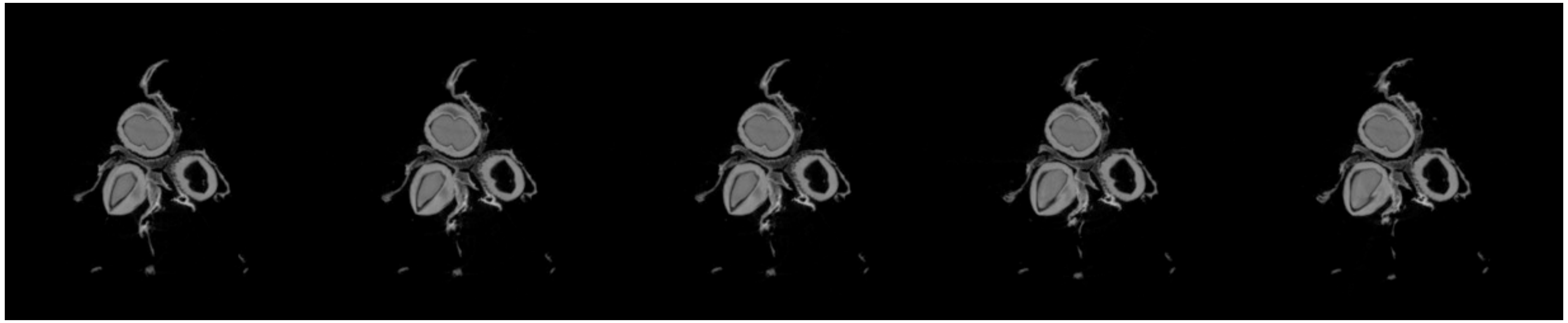
```
backward unfolded tensor (frontal)
(0, 3, 6, 9, 12, 15, 18, 21)
(1, 4, 7, 10, 13, 16, 19, 22)
(2, 5, 8, 11, 14, 17, 20, 23)
```

```
matrix< I3, I1*I2, values_t > unf_horiz_bwd;
t3.horizontal_unfolding_bwd( unf_horiz_bwd );
```

```
backward unfolded tensor (horizontal)
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
(12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23)
```

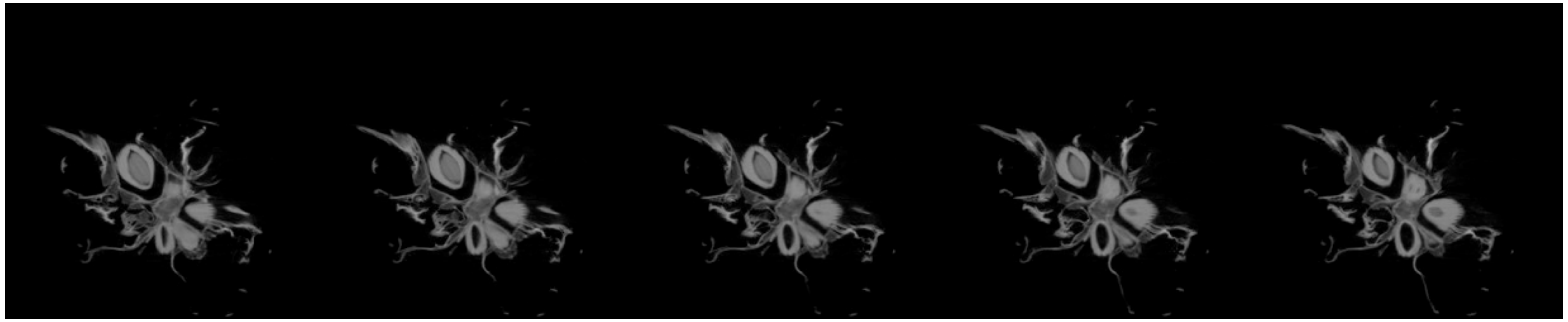
Example Unfoldings along the Modes 1, 2, and 3

mode-1 unfolding …



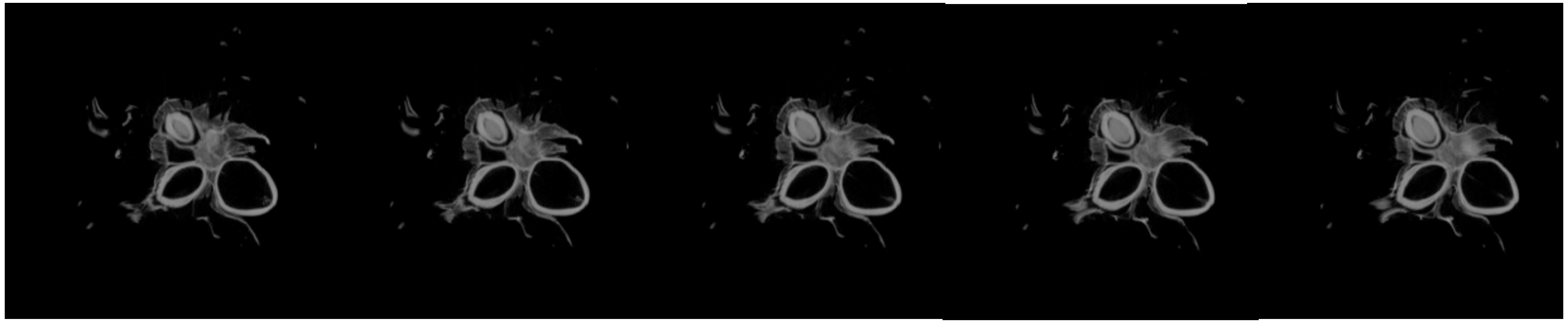
…

mode-1 unfolding …



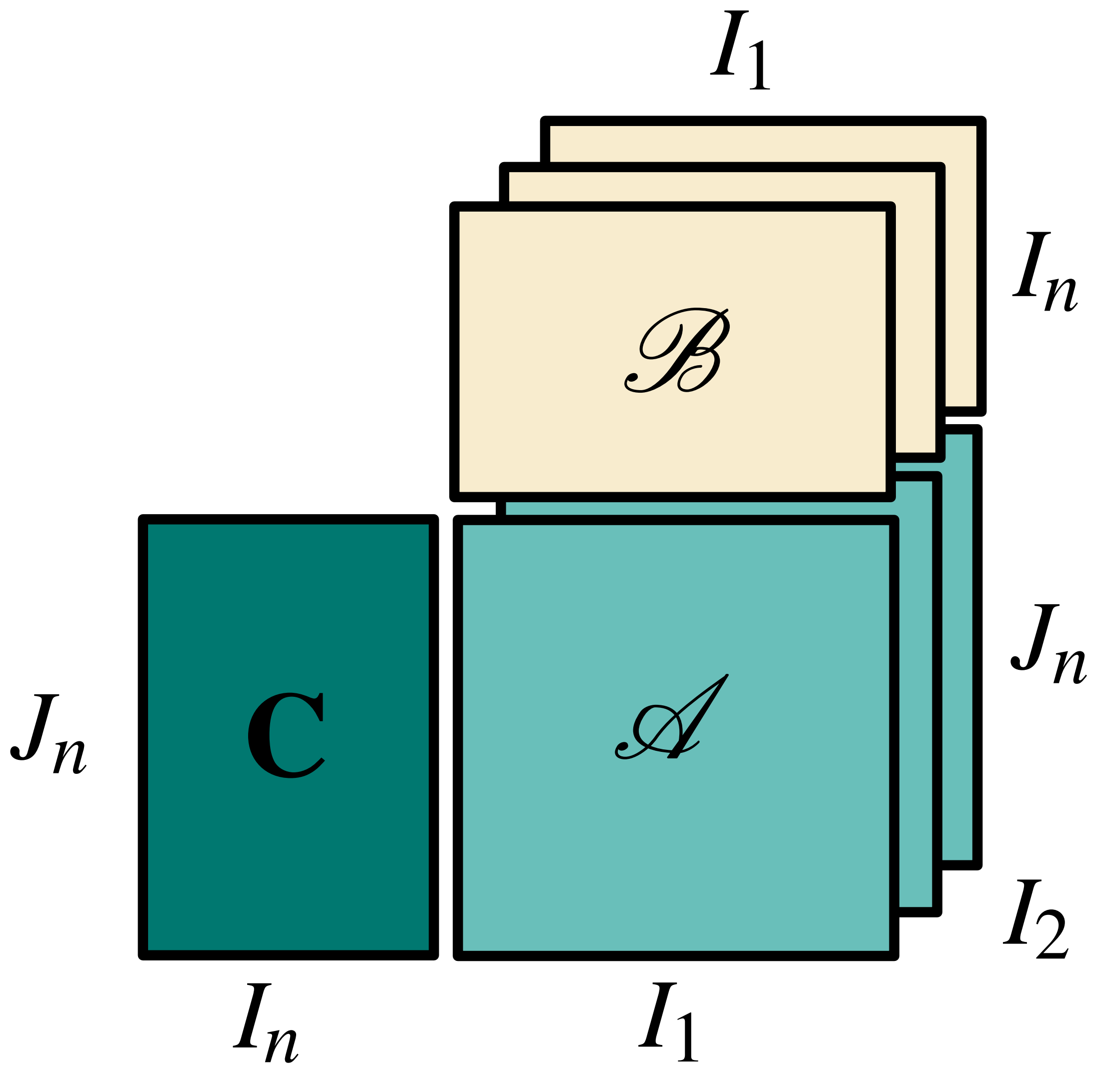
…

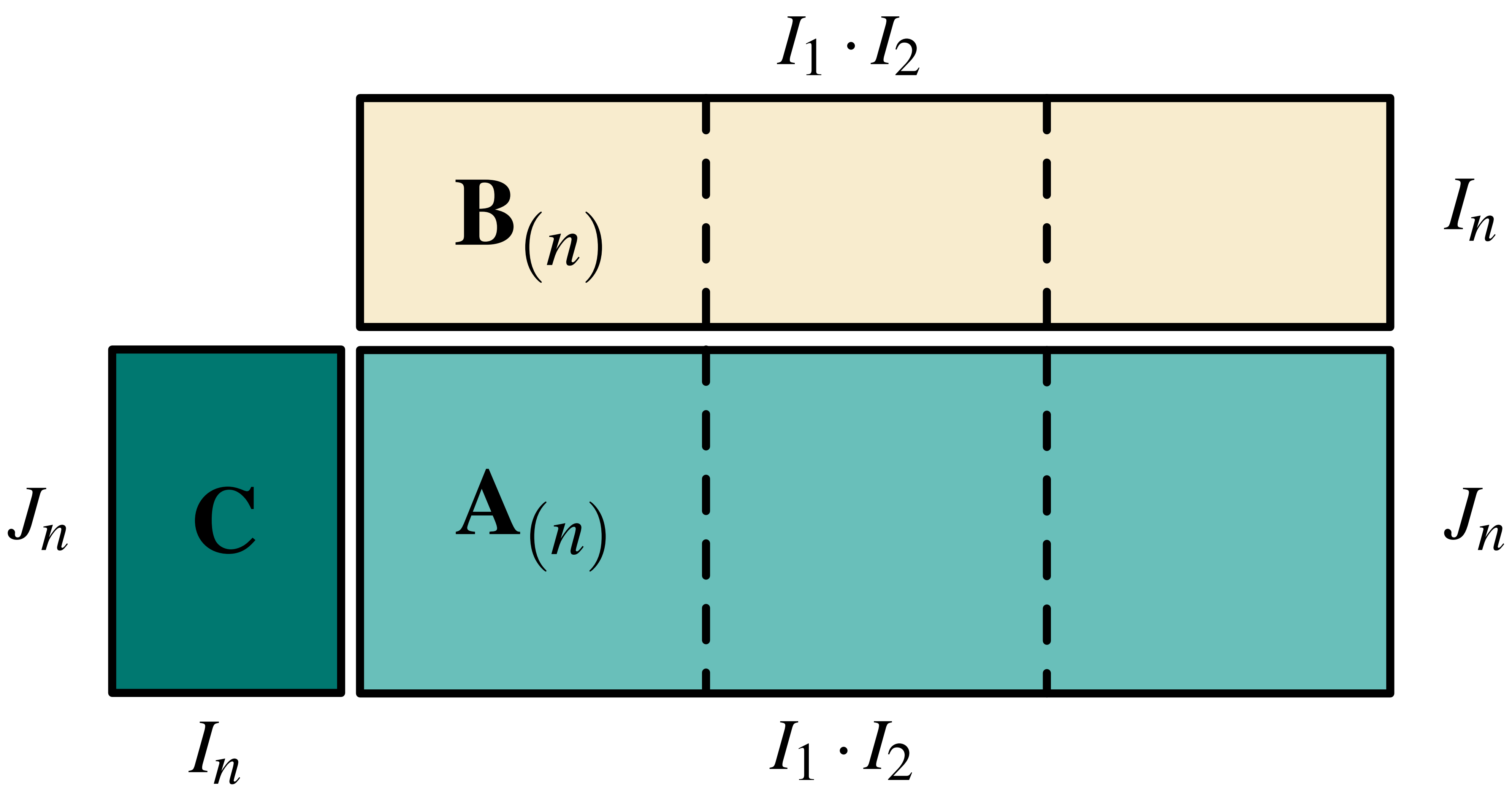
mode-3 unfolding …



…

Tensor Times Matrix Multiplication





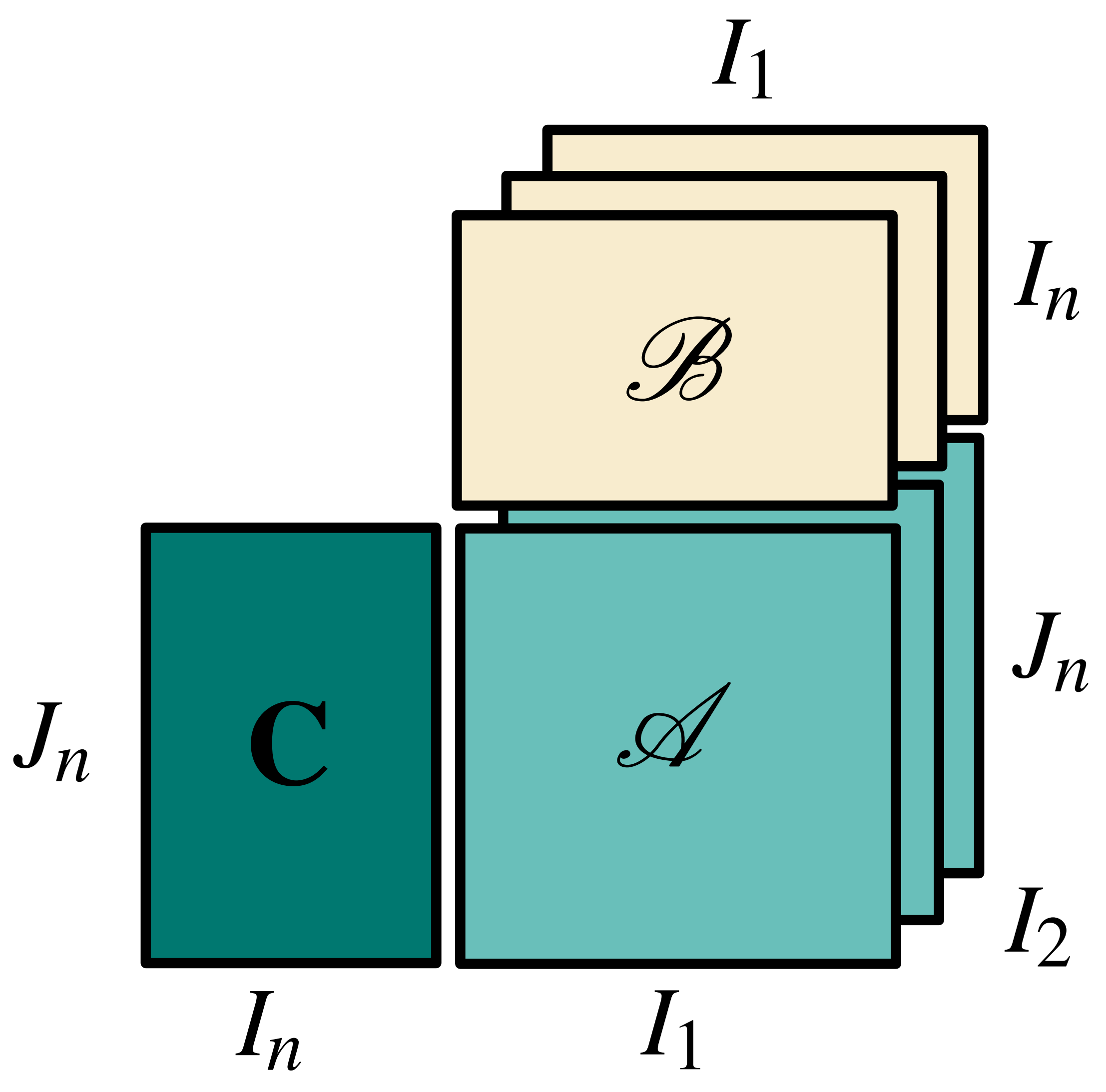
$\mathscr{A} = \mathscr{B} \times_n \mathbf{C}$ $\Leftrightarrow$ $\mathbf{A}_{(n)} = \mathbf{C}\mathbf{B}_{(n)}$

n-mode product $(\mathscr{B} \times_n \mathbf{C})_{i_1 \ldots i_{n-1} j_n i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} b_{i_1 i_2 \ldots i_N} \cdot c_{j_n i_n}$

[De Lathauer et al., 2000a]

Tensor Times Matrix Multiplications

[vmmlib]

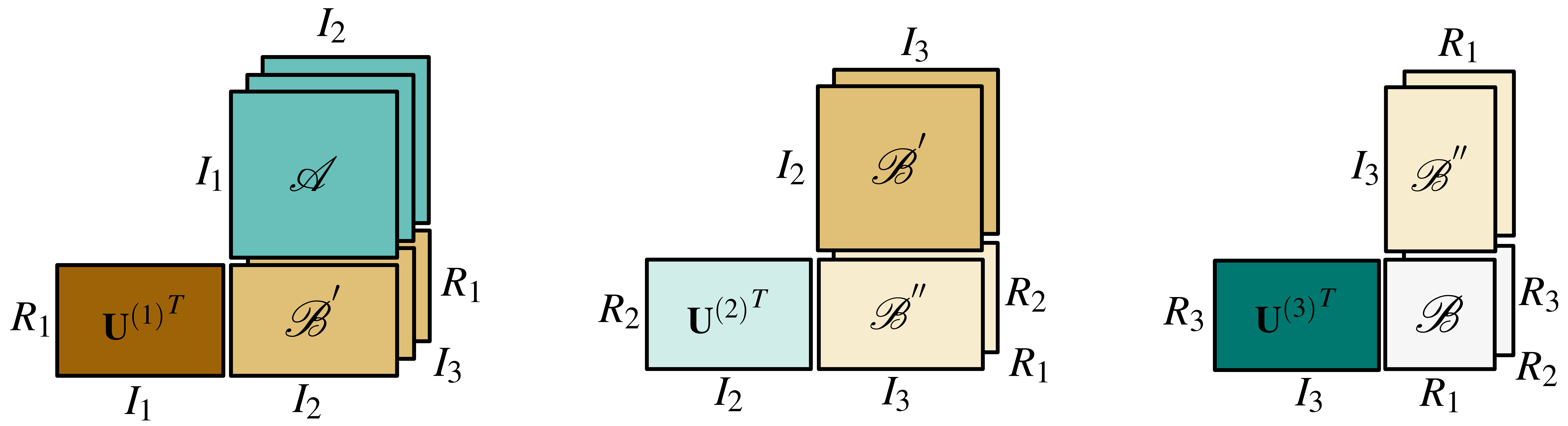



```
t3_ttm::multiply_frontal_fwd(    tensor3_b, matrix_c1, tensor3_a1 );
t3_ttm::multiply_horizontal_fwd( tensor3_b, matrix_c2, tensor3_a2 );
t3_ttm::multiply_lateral_fwd(    tensor3_b, matrix_c3, tensor3_a3 );

t3_ttm::full_tensor3_matrix_multiplication(
                        tensor3_b,
                        matrix_c1,
                        matrix_c2,
                        matrix_c3,
                        tensor3_a
                        );
```

- The T3_TTM is implemented using openMP and BLAS for the parallel matrix-tensor_slice multiplications.
- The full TTM multiplication includes three TTMs: first a TTM along frontal slices, then a TTM along horizontal slices, and finally a TTM along lateral slices.
- Since the tensor3 is an array consisting of frontal slices (matrices), we start first with the frontal slice multiplication. This is optimized for tensors with In > Jn (For example, Tucker core generation). If you have a situation, where Jn > In (for example Tucker reconstruction), you could rearrange the order of the modes of the TTM multiplications such that the most expensive TTM (the one of the largest tensor) is performed along frontal slices.

Example TTMs: Core Computation

[vmmlib]



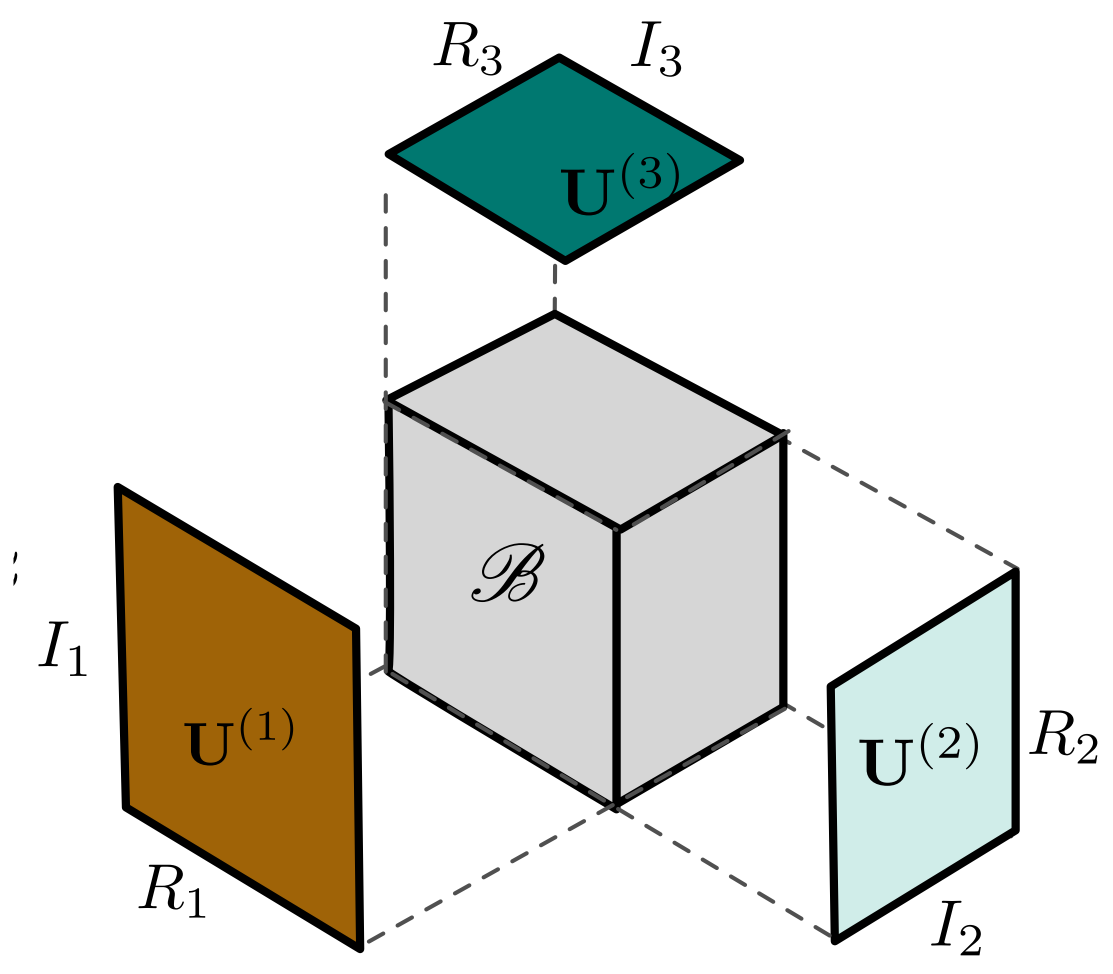
$$\mathscr{B} = \mathscr{A} \times_1 \mathbf{U}^{(1)^{(-1)}} \times_2 \mathbf{U}^{(2)^{(-1)}} \times_3 \cdots \times_N \mathbf{U}^{(N)^{(-1)}} \xrightarrow[\text{factor matrices}]{\text{orthogonal}} \mathscr{B} = \mathscr{A} \times_1 \mathbf{U}^{(1)^T} \times_2 \mathbf{U}^{(2)^T} \times_3 \cdots \times_N \mathbf{U}^{(N)^T}$$

- Three consecutive TTM multiplication (along modes 1,2,3)
- For orthogonal matrices, use the transposes of the three factor matrices (otherwise the (pseudo)-inverses)
- `t3_ttm::full_tensor3_matrix_multiplication( A, U1_t, U2_t, U3_t, B );`

Tucker3 Tensor

[vmmlib]

```
typedef tucker3_tensor< R1, R2, R3, I1, I2, I3, T_value, T_coeff > tucker3_t;
```



- Define input tensor size (I_1,I_2,I_2)
- Define multilinear rank (R_1,R_2,R_3)
- Define value type and coefficient value type
- Internally always computes with floating point values
- Stores the three factor matrices (I_n x R_n) and the core tensor (R_1,R_2,R_3)
- ALS:
  - if not converged (fit does not improve anymore, tolerance 1e-04)
  - the ALS stops latest after 10 iteration
- Reconstruction

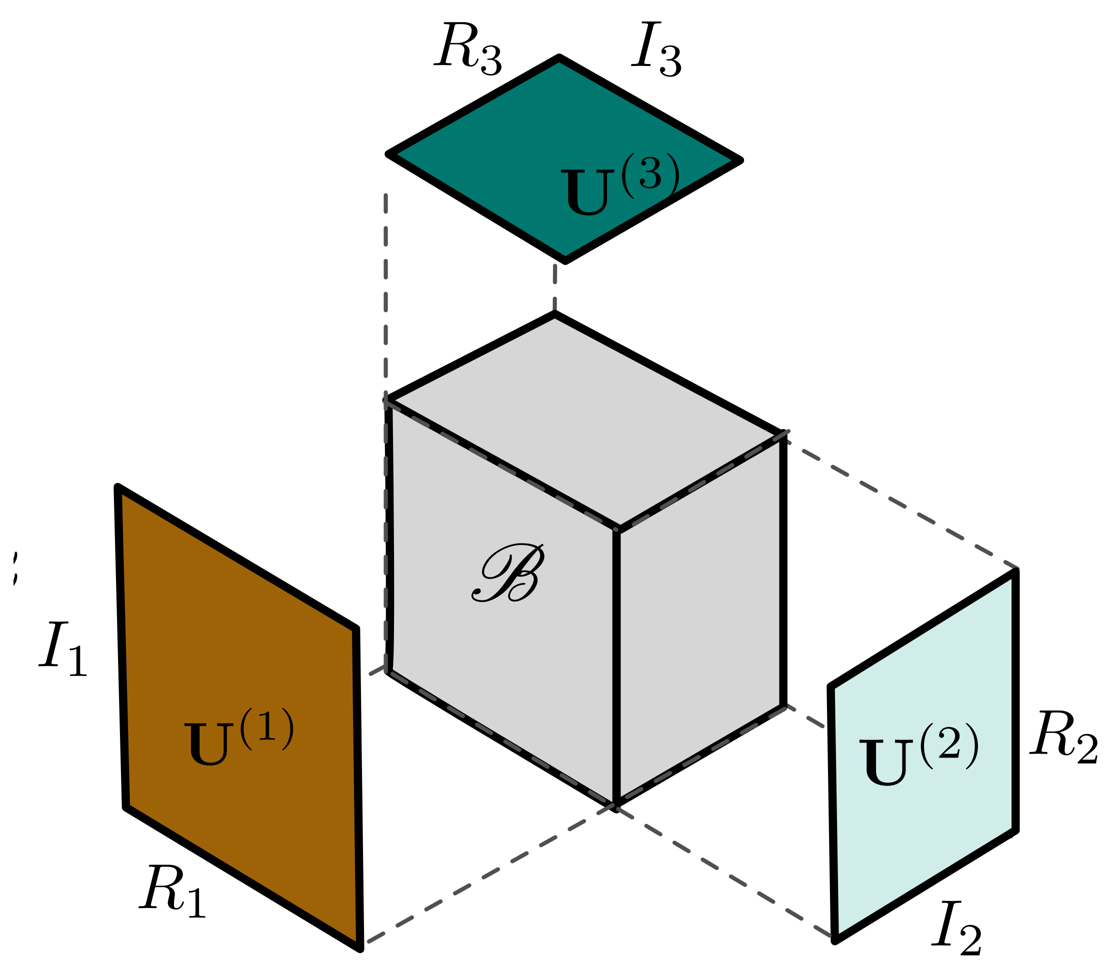
VISUALIZATIONAND MULTIMEDIALAB

universitätbonn

Example Code Tucker3 Tensor

[vmmlib]

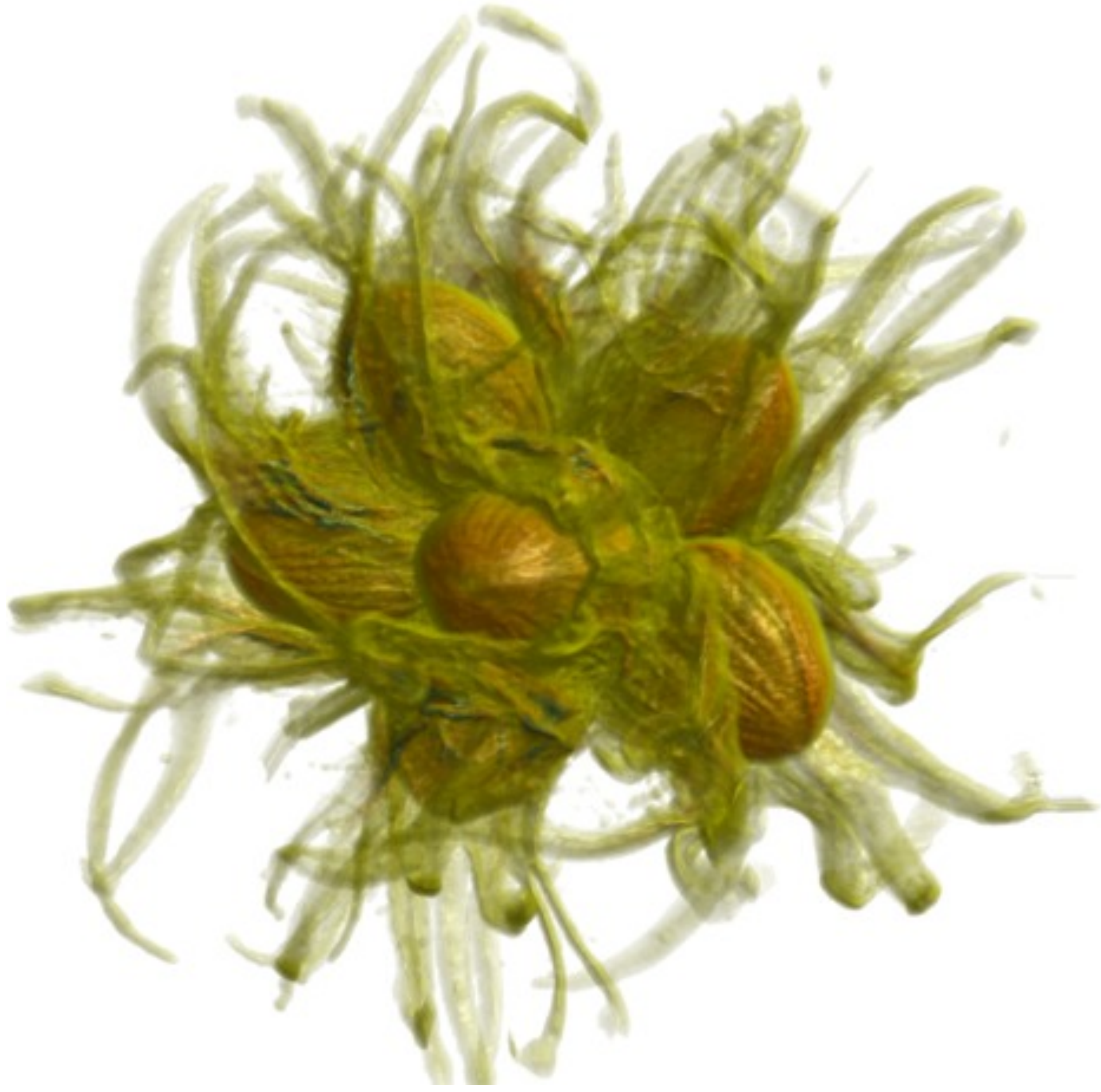
```
typedef tensor3< I1, I2, I3, values_t > t3_t;
t3_t t3; //after initializing a tensor3, the tensor is still empty
t3.fill_increasing_values(); //fills the empty tensor with the values 0,1,2,3...

typedef tucker3_tensor< R1, R2, R3, I1, I2, I3, values_t, float > tucker3_t;
tucker3_t tuck3_dec; //empty tucker3 tensor

//choose initialization of Tucker ALS (init_hosvd, init_random, init_dct)
typedef t3_hooi< R1, R2, R3, I1, I2, I3, float > hooi_t;

//Example for initialization with init_rand
tuck3_dec.tucker_als( t3, hooi_t::init_random());

//Example for initialization with init_hosvd
tuck3_dec.tucker_als( t3, hooi_t::init_hosvd());

//Reconstruction
t3_t t3_reco;
tuck3_dec.reconstruct( t3_reco );

//Reconstruction error (RMSE)
double rms_err = = t3.rmse( t3_reco );
```

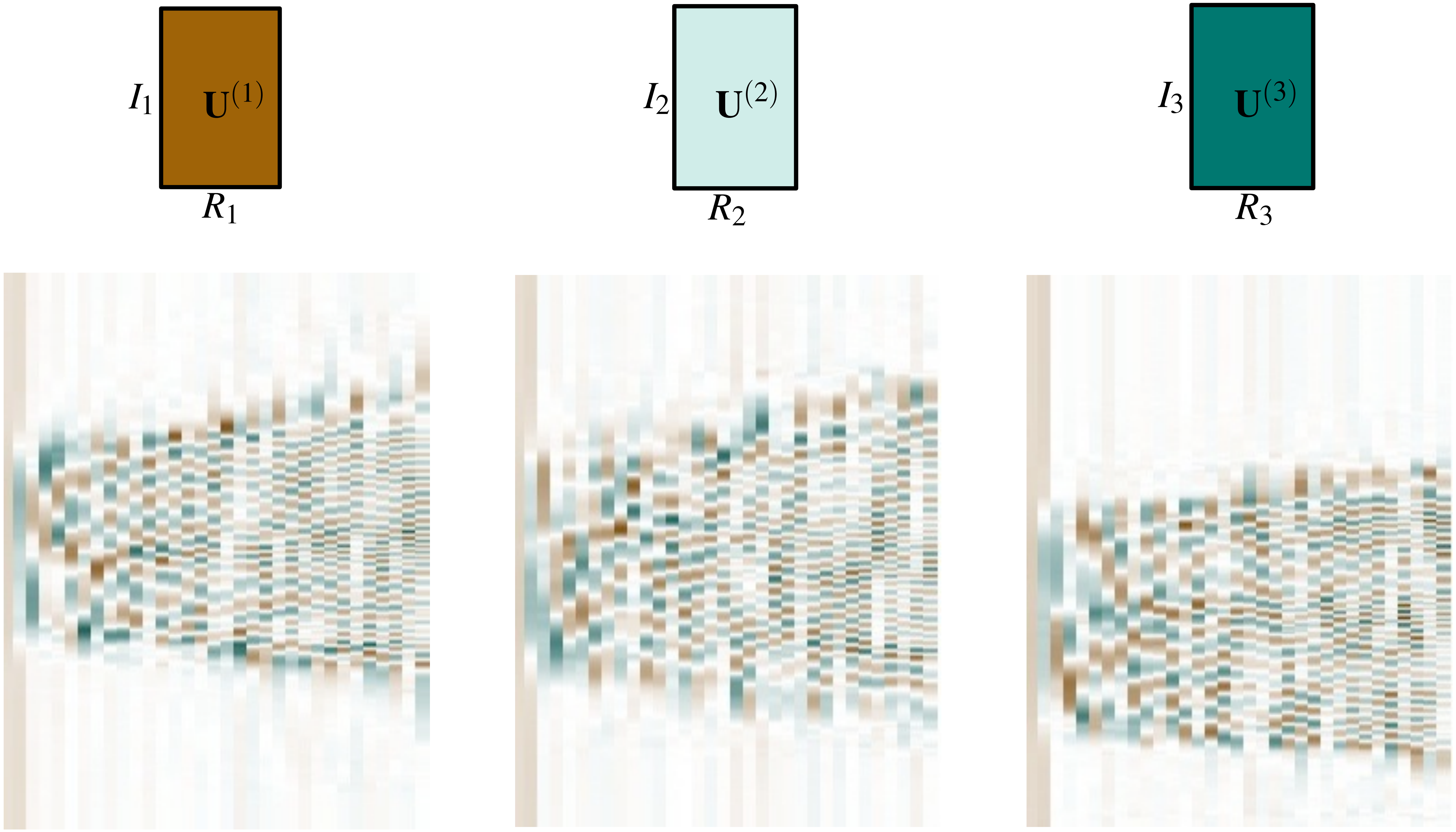


VISUALIZATIONANDMULTIMEDIALAB

universitätbonn

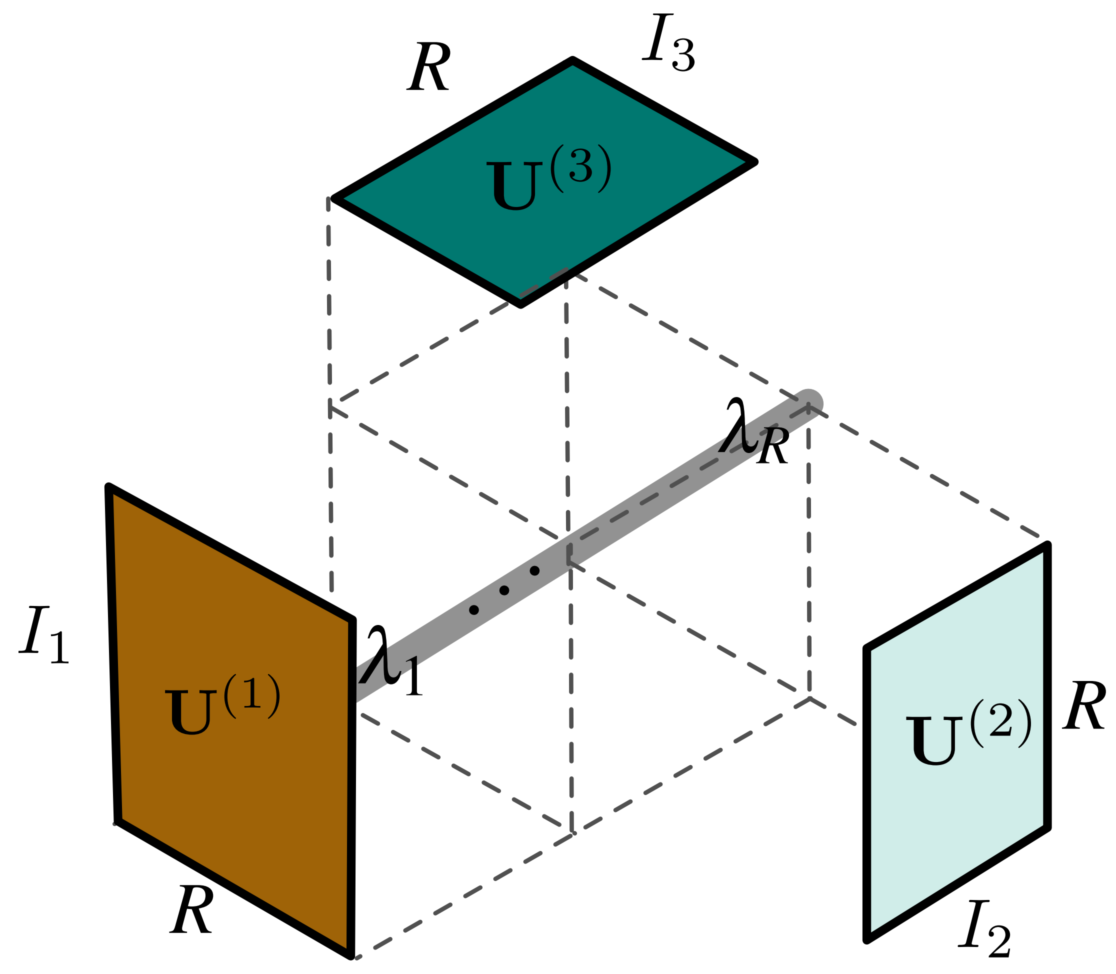
Example Tucker3 Factor Matrices





CP3 Tensor

[vmmlib]

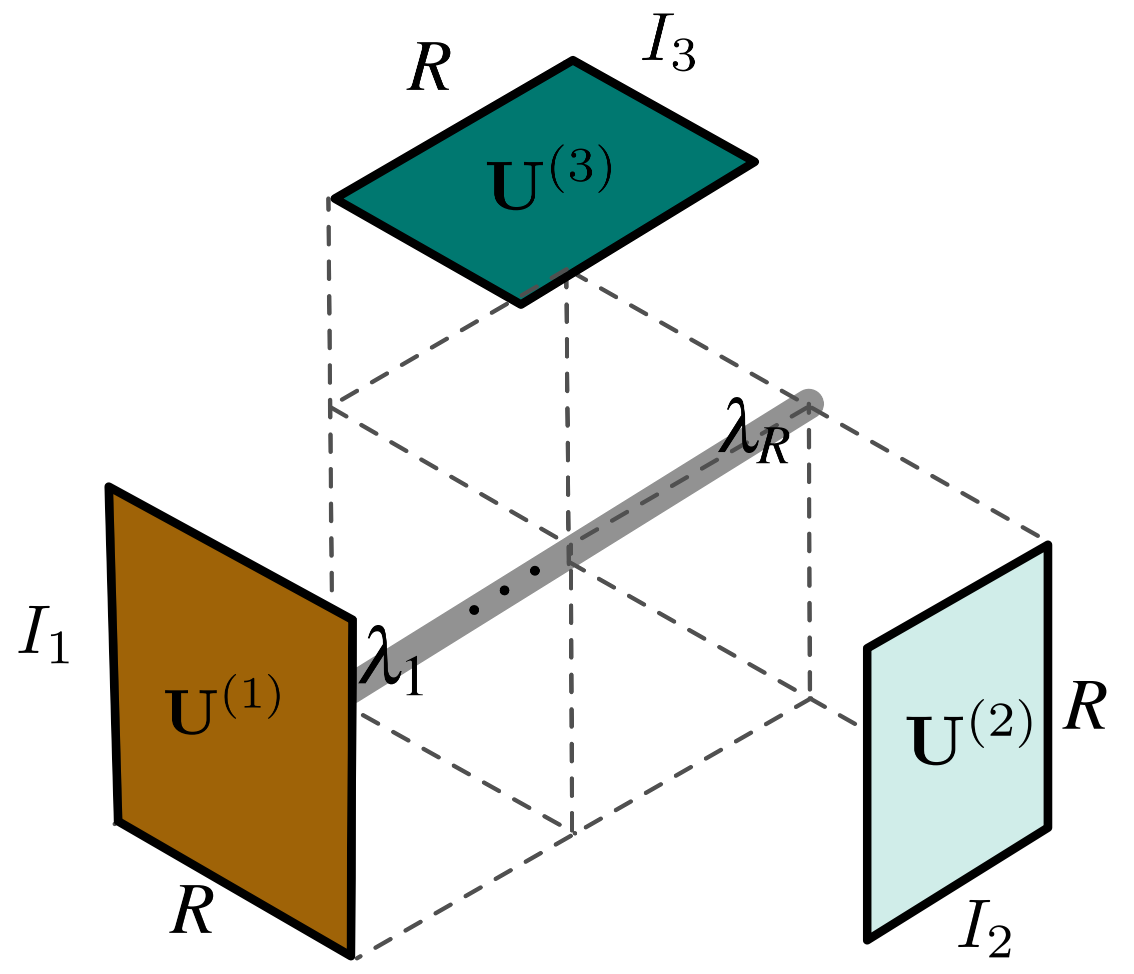


- Define input tensor size (I_1,I_2,I_2)
- Define rank R
- Define value type and coefficient value type
- Internally always computes with floating point values
- Stores three factor matrices each of size (I_n x R) and the lambdas R
- ALS:
  - if not converged (fit does not improve anymore, tolerance 1e-04)
  - set number of maximum CP ALS iterations
- Reconstruction

Code Example CP3 Tensor

[vmmlib]

```
typedef cp3_tensor< r, a, b, c, values_t, float > cp3_t;
typedef t3_hopm< r, a, b, c, float > t3_hopm_t;

cp3_t cp3_dec;

//Decomposition or CP ALS
//choose initialization of Tucker ALS (init_hosvd, init_random)

int max_cp_iter = 20;
cp3_dec.cp_als( t3, t3_hopm_t::init_random(), max_cp_iter );

//Reconstruction
t3_t t3_cp_reco;
cp3_dec.reconstruct( t3_cp_reco );

//Reconstruction error (RMSE)
rms_err = t3.rmse( t3_cp_reco ) ;
```

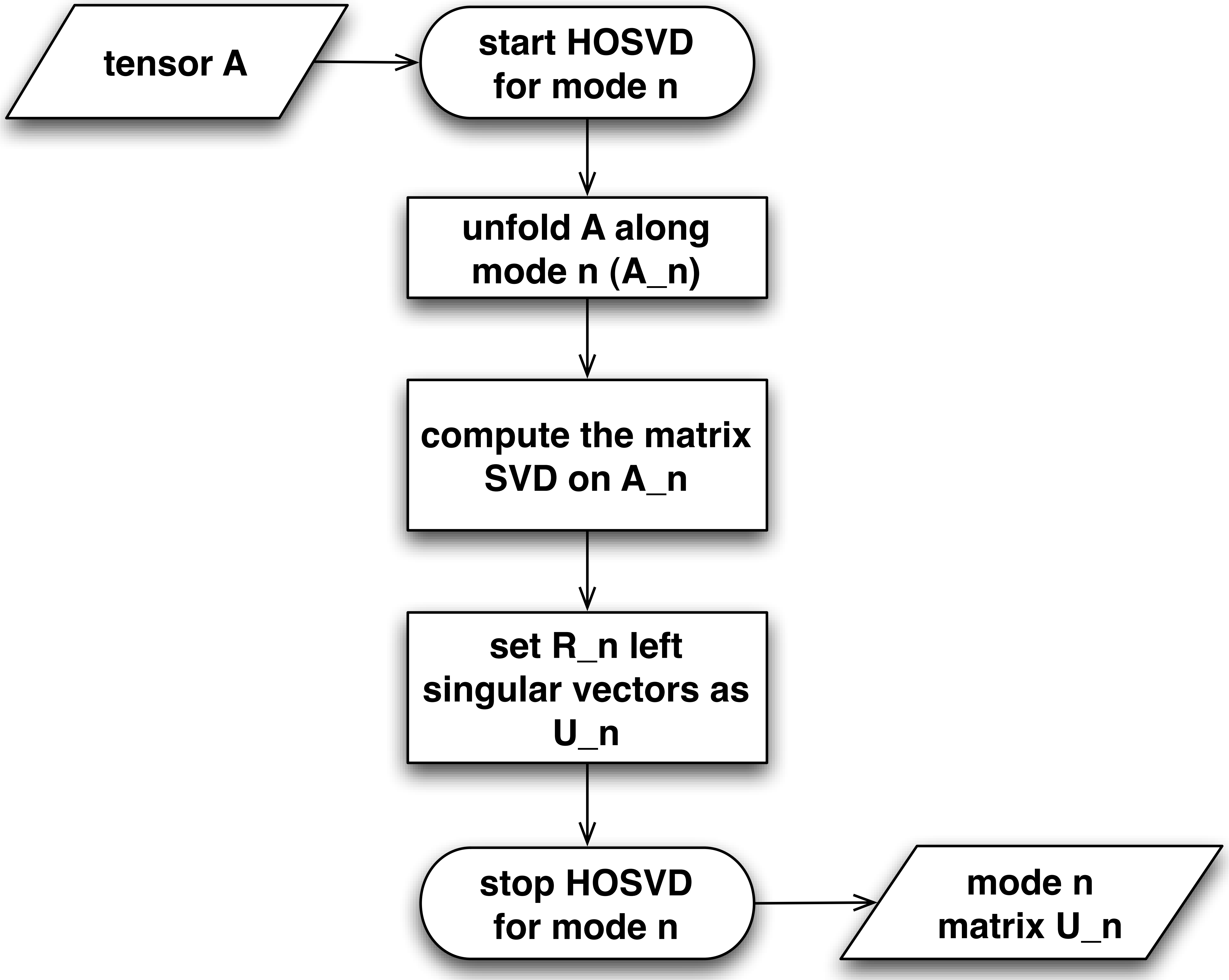


VISUALIZATIONAND MULTIMEDIALAB

universitätbonn

Higher-order SVD (HOSVD)

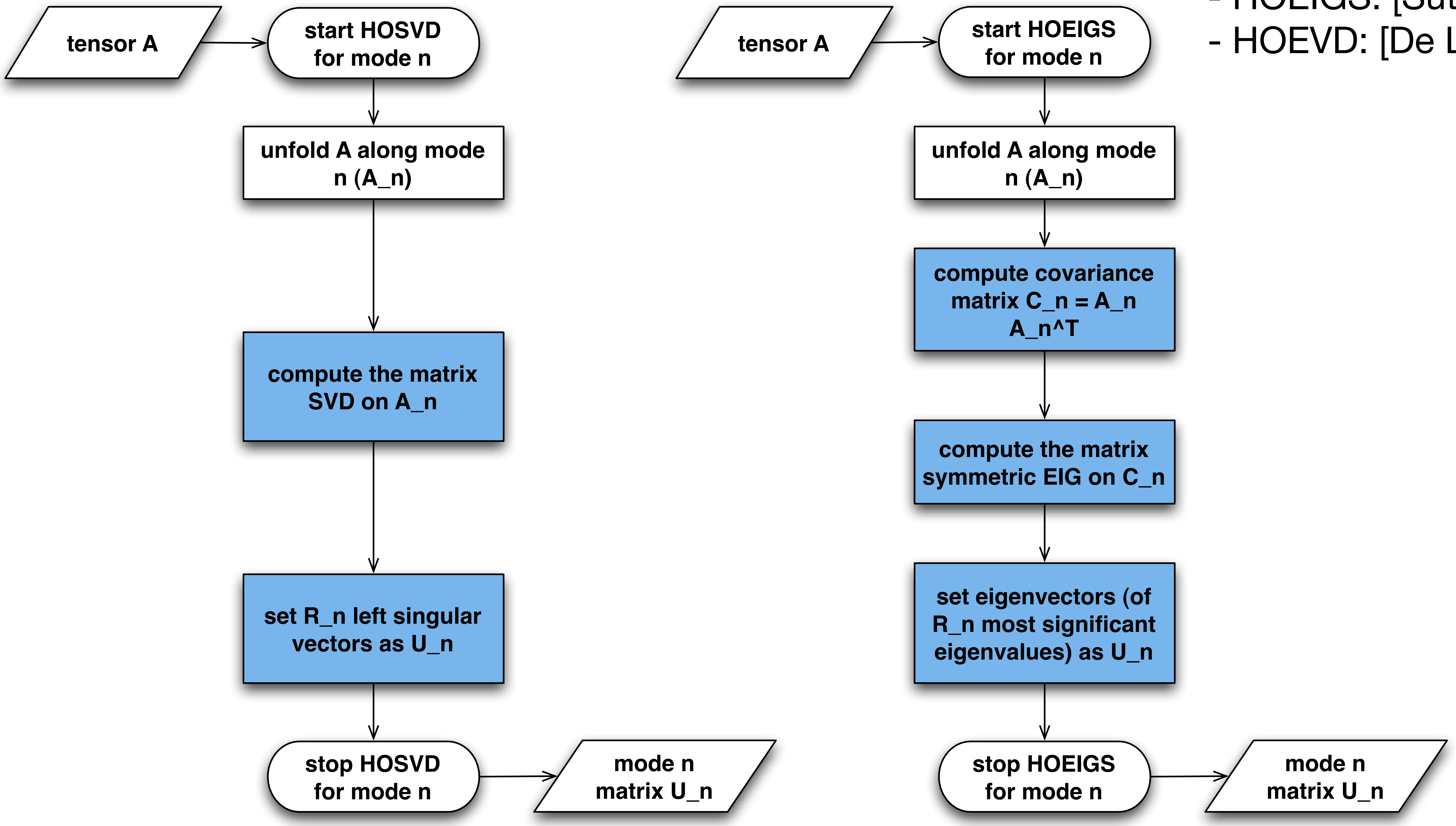
[De Lathauwer et al., 2000a]
[vmmlib]



HOSVD vs. HOEIGS (HOEVD)

[De Lathauwer et al., 2000a]

Higher-order symmetric eigenvalue decomposition
- HOEIGS: [Suter et al.]
- HOEVD: [De Lathauwer et al., 2000a]



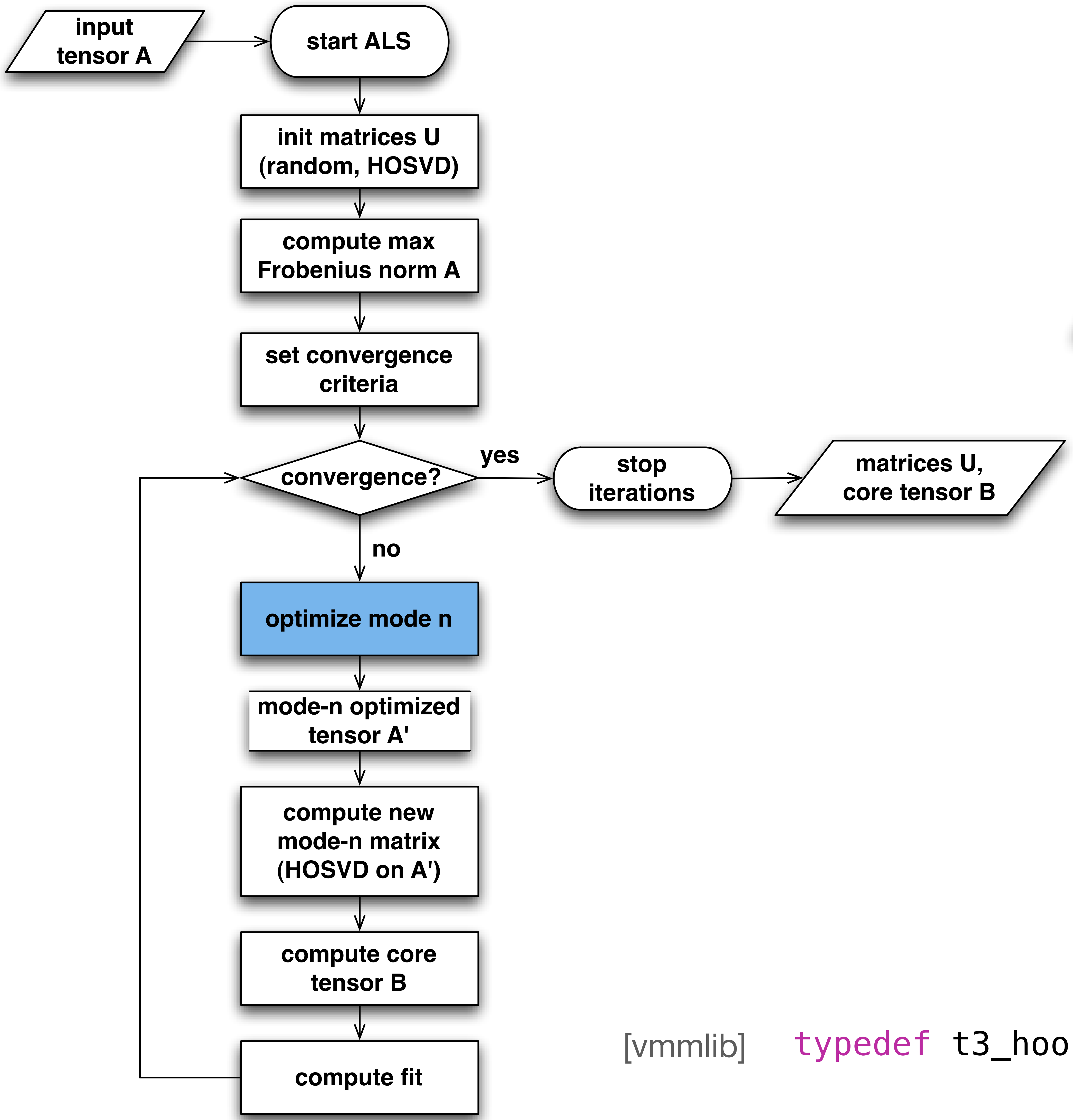
[vmmlib]
```
typedef t3_hosvd< R1, R2, R3, I1, I2, I3 > t3_hosvd_t;
//HOSVD modes: eigs_e or svd_e
```

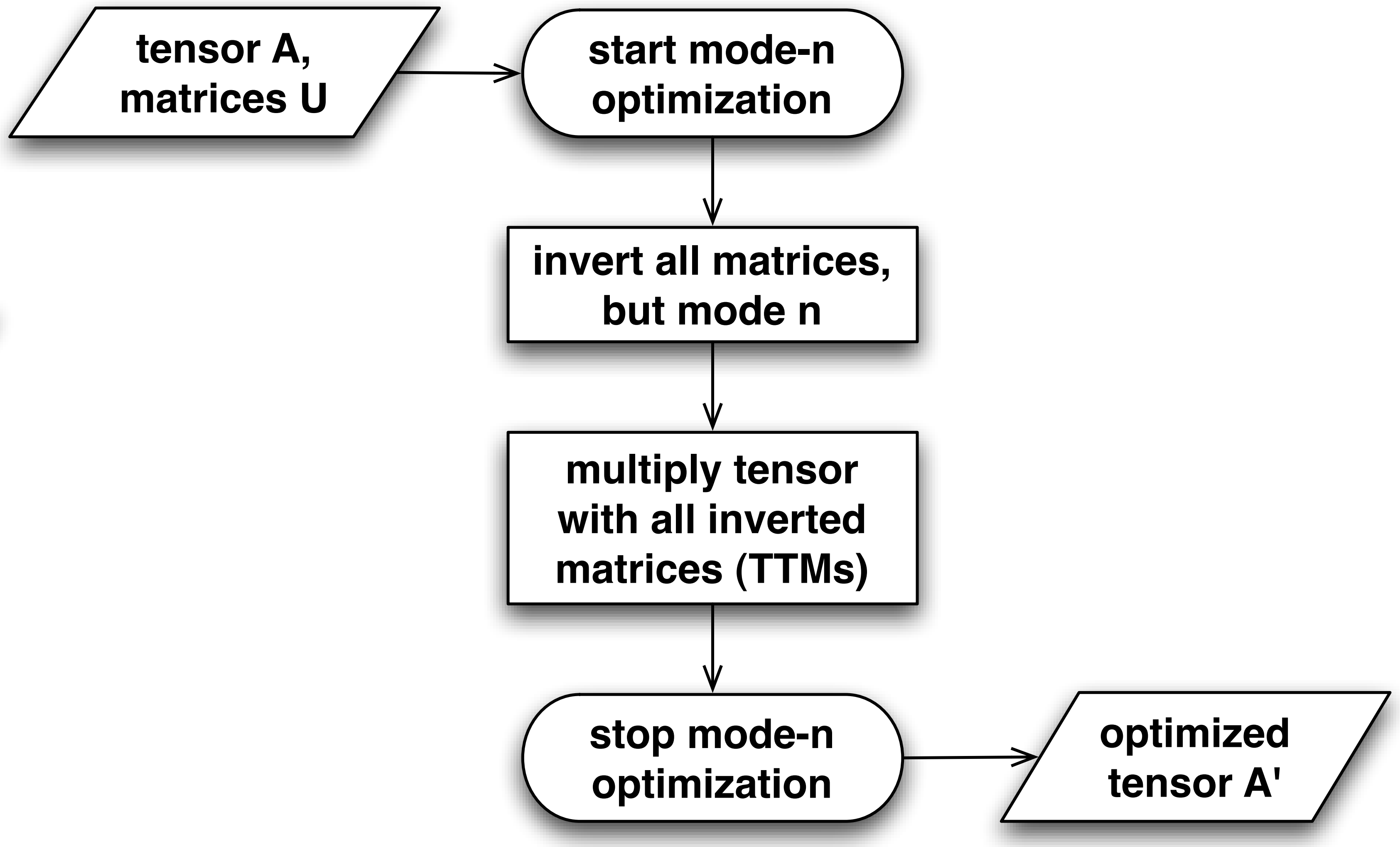

VISUALIZATION AND MULTIMEDIA LAB

universität bonn

Higher-order Orthogonal Iteration (HOOI)

[De Lathauwer et al., 2000b]

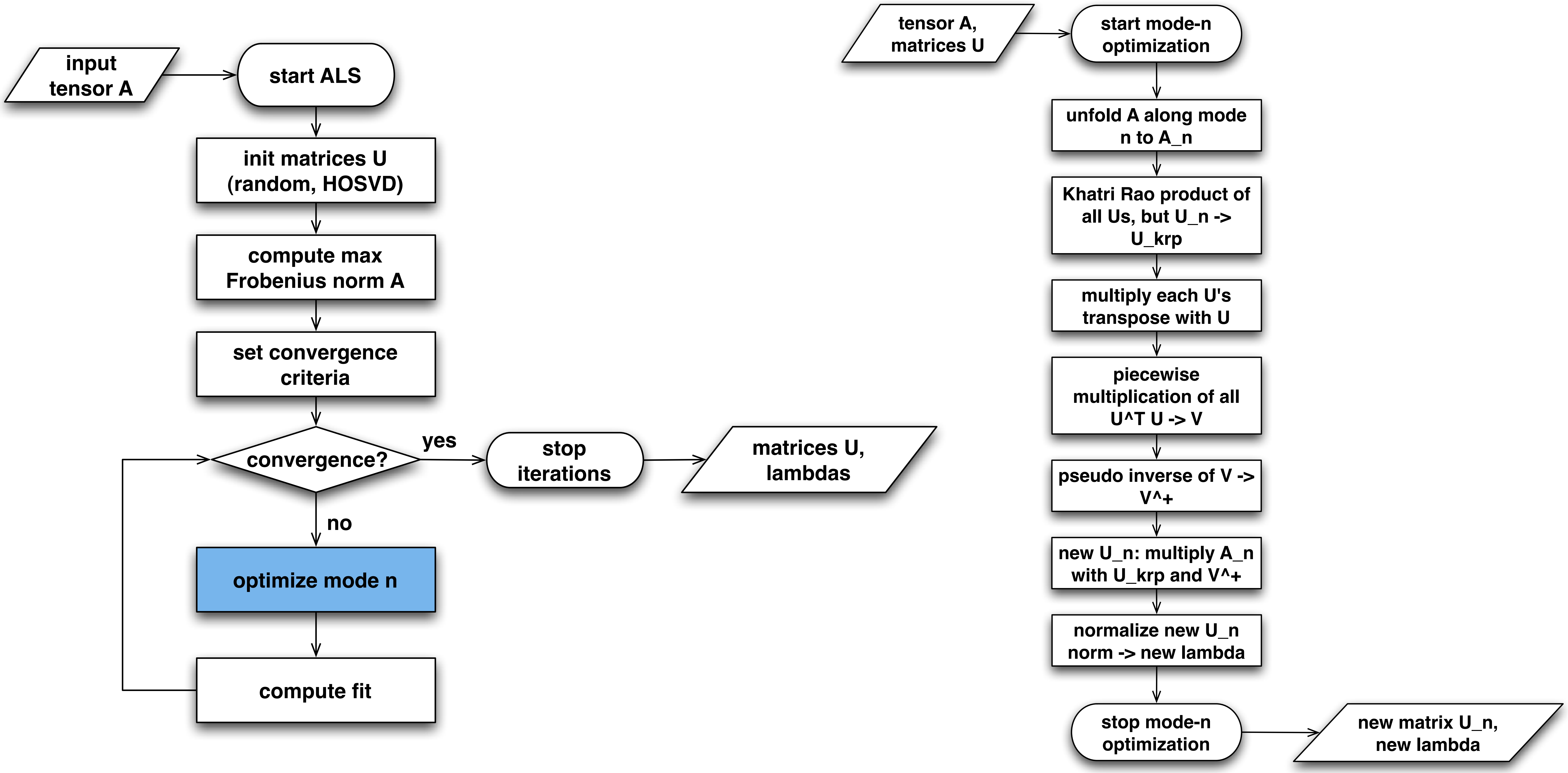




[vmmlib] `typedef t3_hooi< R1, R2, R3, I1, I2, I3 > t3_hooi_t;`

Higher-order Power Method (HOPM)

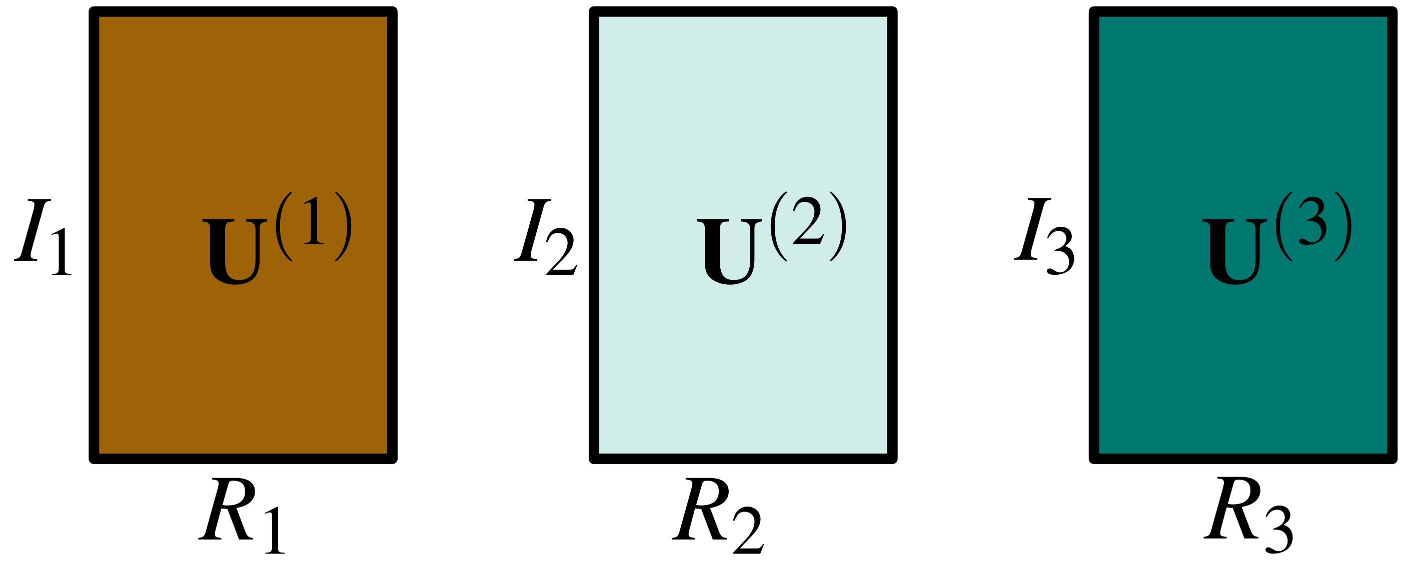
[De Lathauwer et al., 2000b]



[vmmlib] `typedef t3_hopm< R, I1, I2, I3 > t3_hopm_t;`

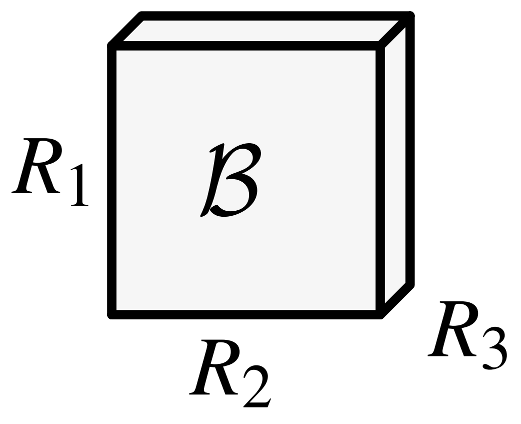
Tucker Tensor-specific Quantization

[Suter et al., 2011]



- Factor matrices quantization
  - values between [-1,1]
  - linear quantization

$$\tilde{x}_{\mathbf{U}} = (2^{Q_{\mathbf{U}}} - 1) \cdot \frac{x - x_{min}}{x_{max} - x_{min}}$$



- Core tensor quantization
  - many small values; few large values
  - logarithmic quantization

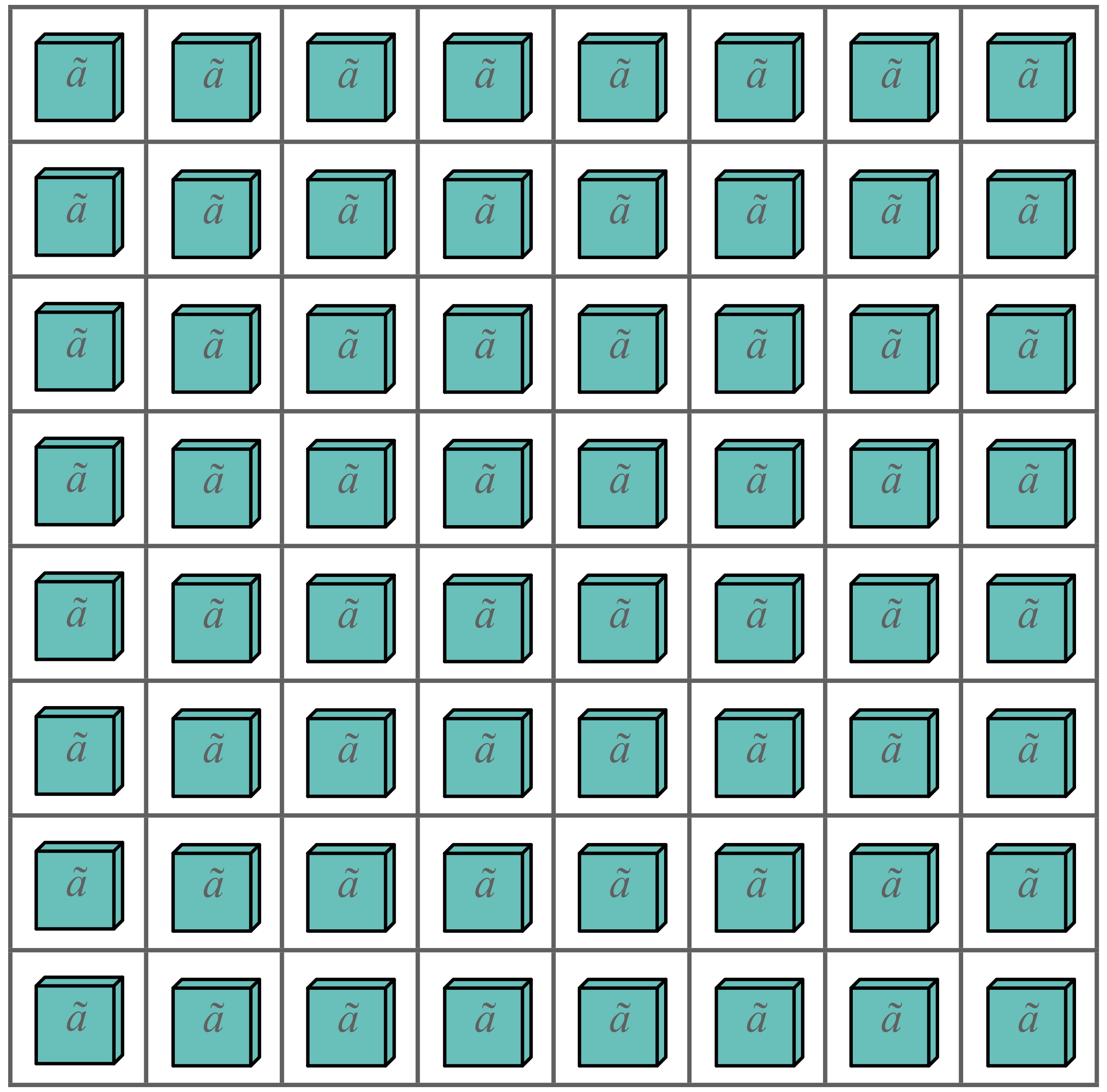
$$|\tilde{x}_{\mathscr{B}}| = (2^{Q_{\mathscr{B}}} - 1) \cdot \frac{\log_2(1 + |x|)}{\log_2(1 + |x_{max}|)}$$

[vmmlib]

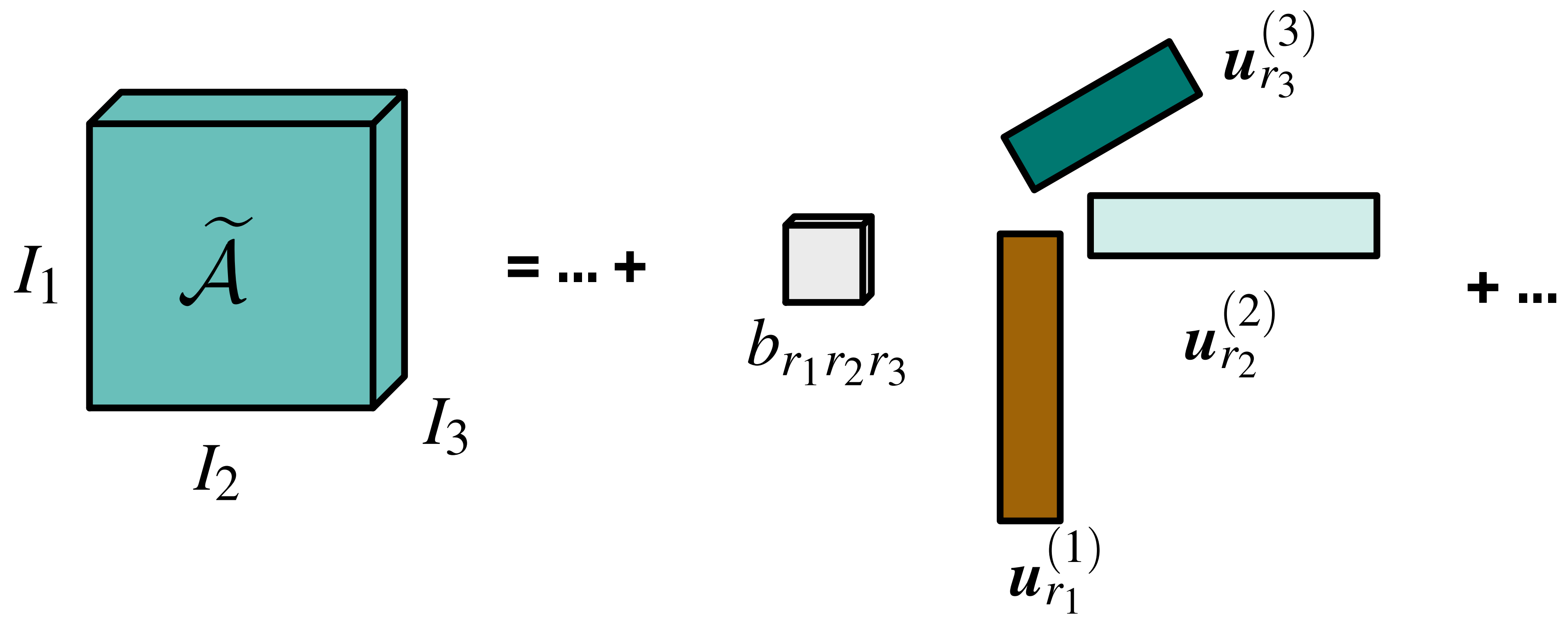
```
typedef qtucker3_tensor< R1, R2, R3, I1, I2, I3, T_value, T_coeff > qtucker3_t;
```

Parallel Tensor Reconstruction

[Suter et al., 2011]



parallel computing grid per brick



$$\widetilde{a}_{i_1 i_2 i_3} = \sum_{r_1} \sum_{r_2} \sum_{r_3} b_{r_1 r_2 r_3} \cdot u^{(1)}_{i_1 r_1} \cdot u^{(2)}_{i_2 r_2} \cdot u^{(3)}_{i_3 r_3}$$

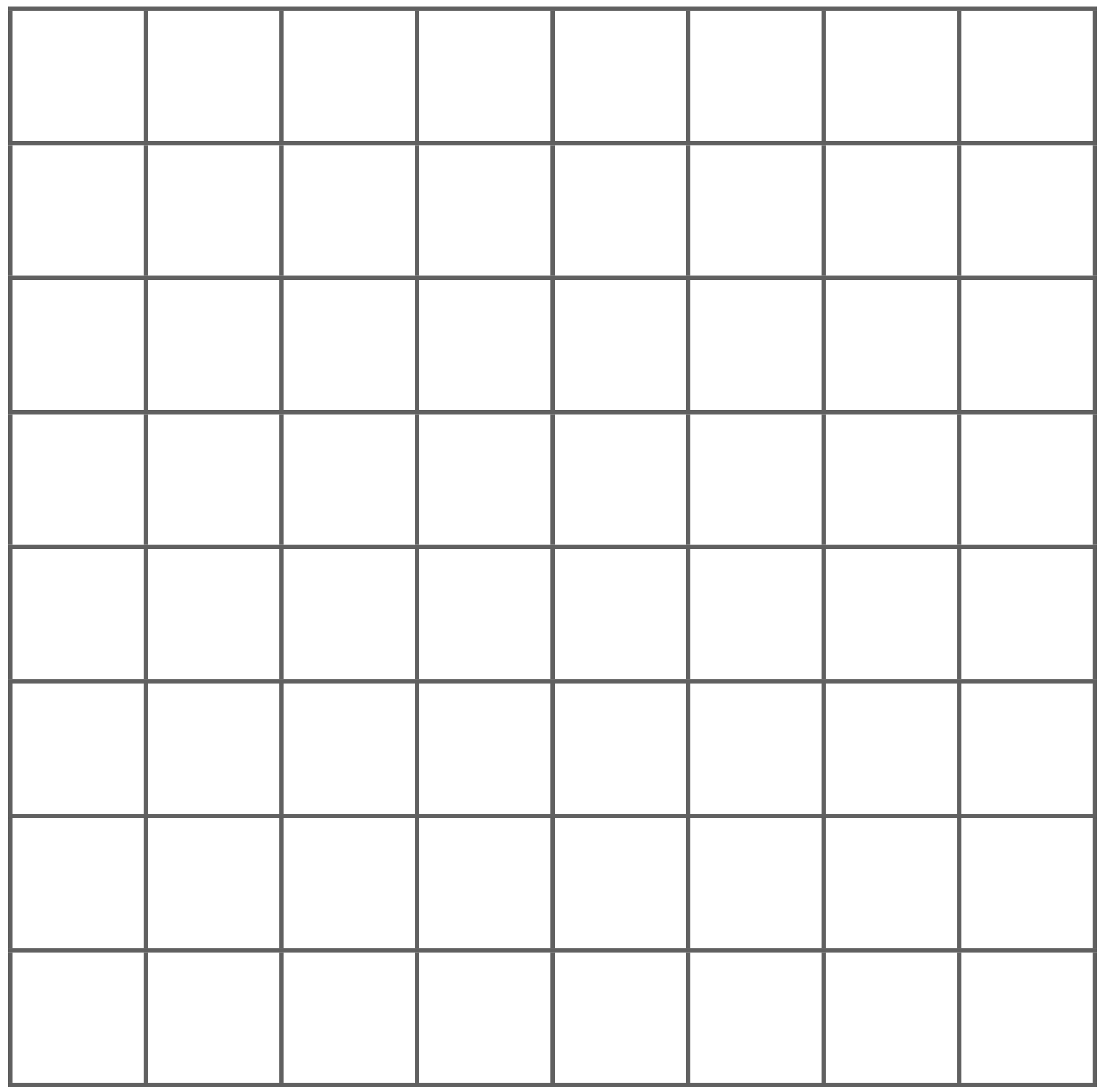
↑

triple-for-loop

computational cost per voxel is <u>cubic</u>: O(R^3)

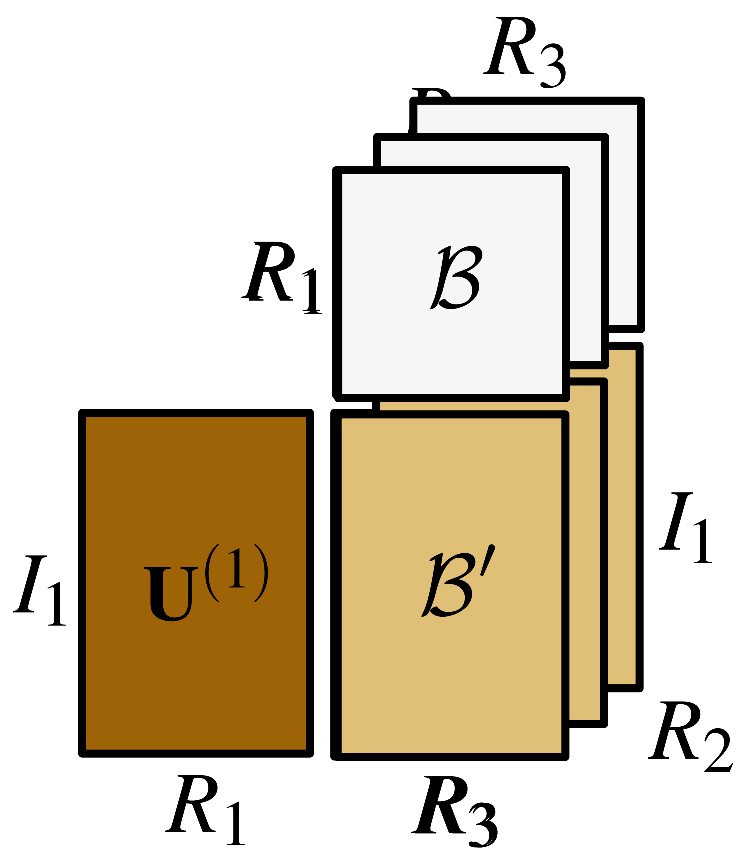
Faster Parallel Tensor Reconstruction

[Suter et al., 2011]



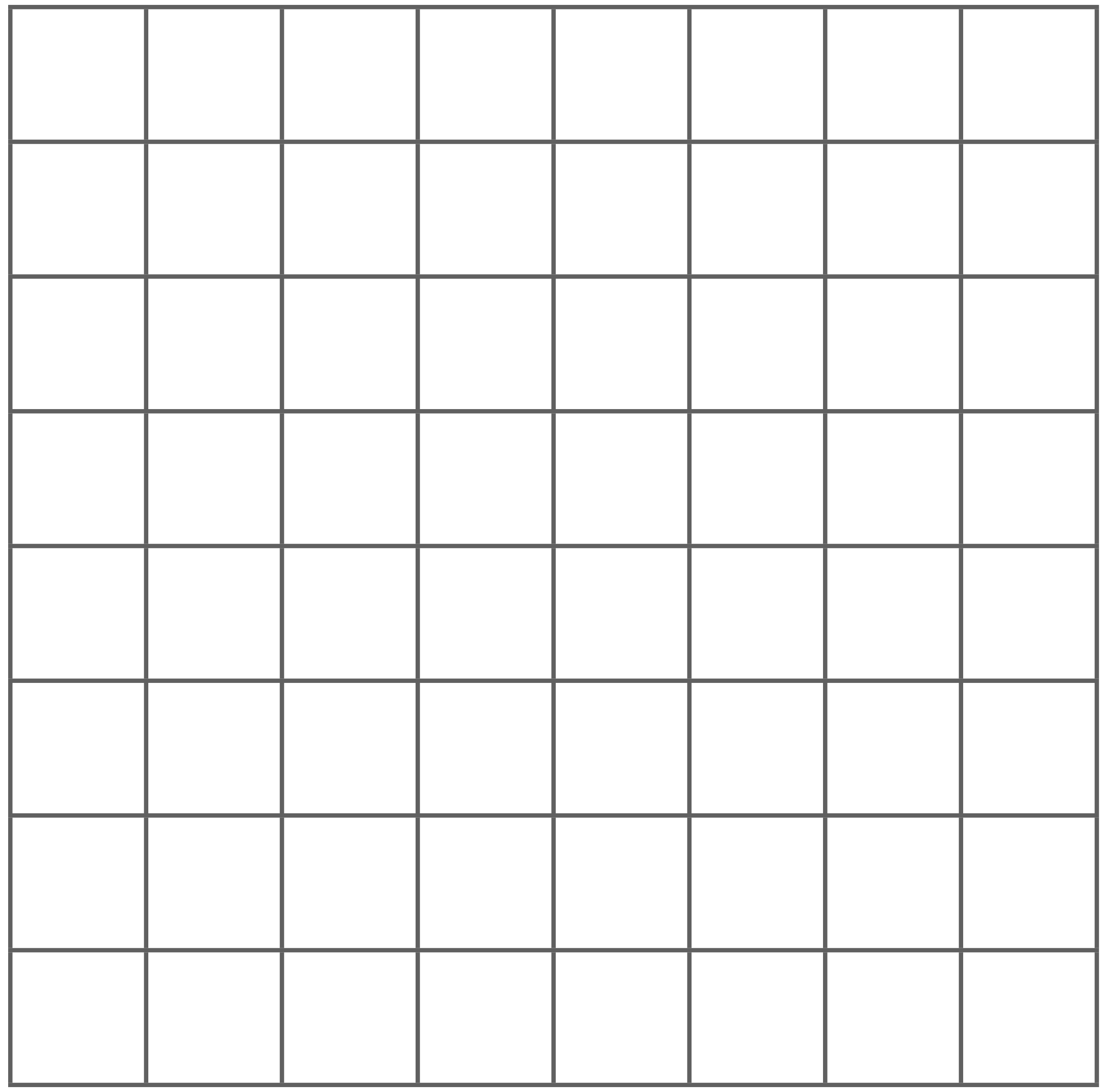
parallel computing grid per brick

tensor times matrix (TTM) multiplication or n-mode product

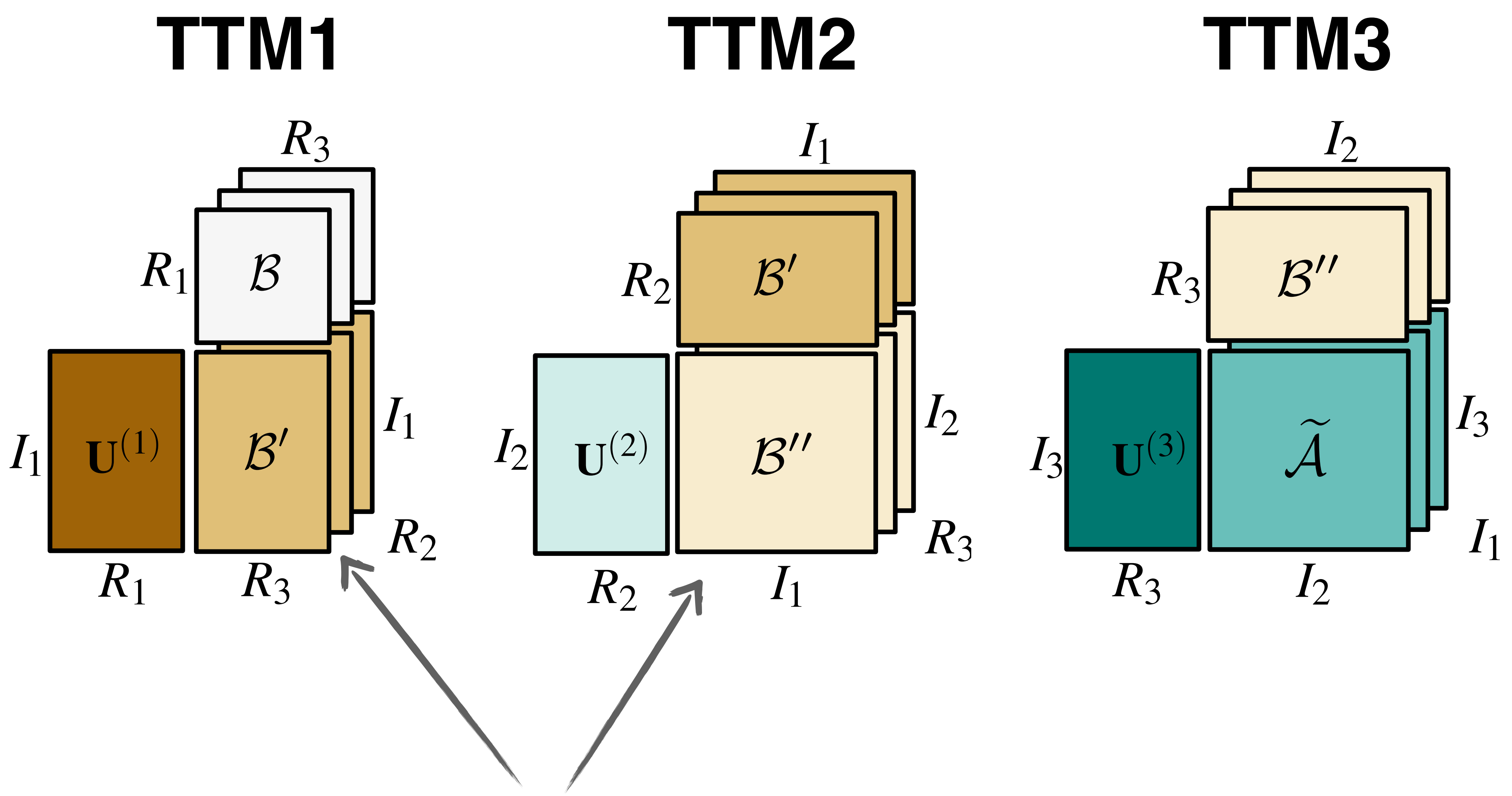


Faster Parallel Tensor Reconstruction

[Suter et al., 2011]



parallel computing grid per brick



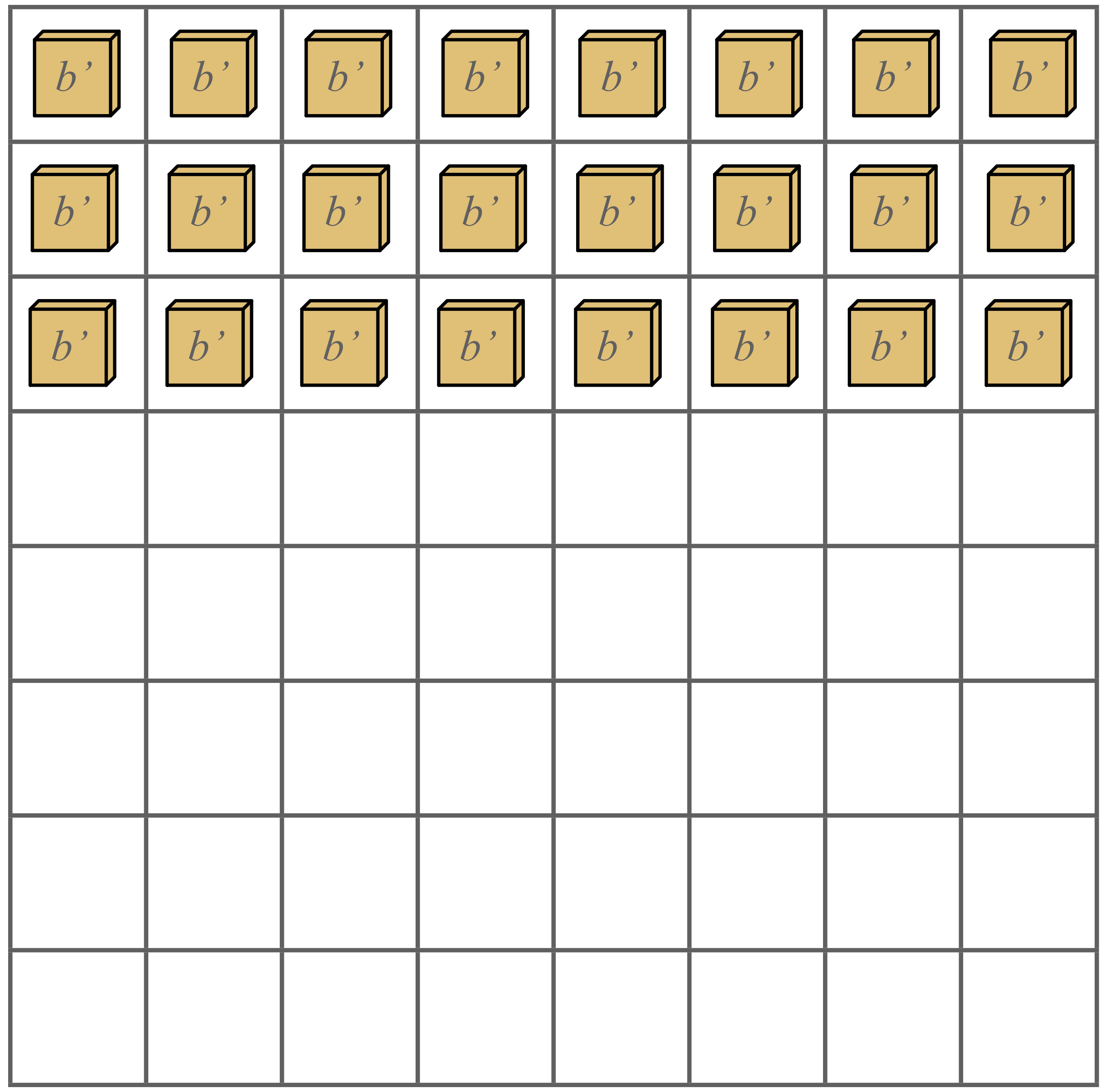
store intermediate results (B' and B'')

$\tilde{a}$ **computational cost per voxel is linear: O(R)**

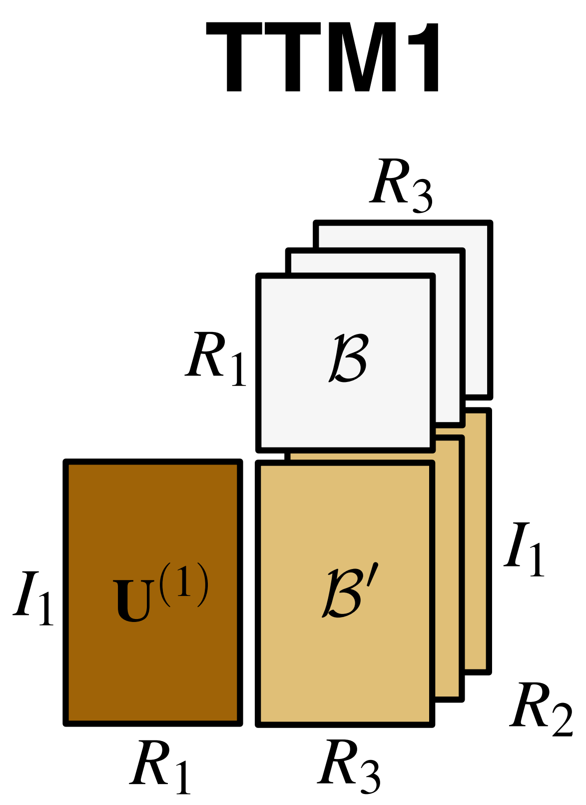
universitätbonn

Compute Intermediate Tensor B’

[Suter et al., 2011]

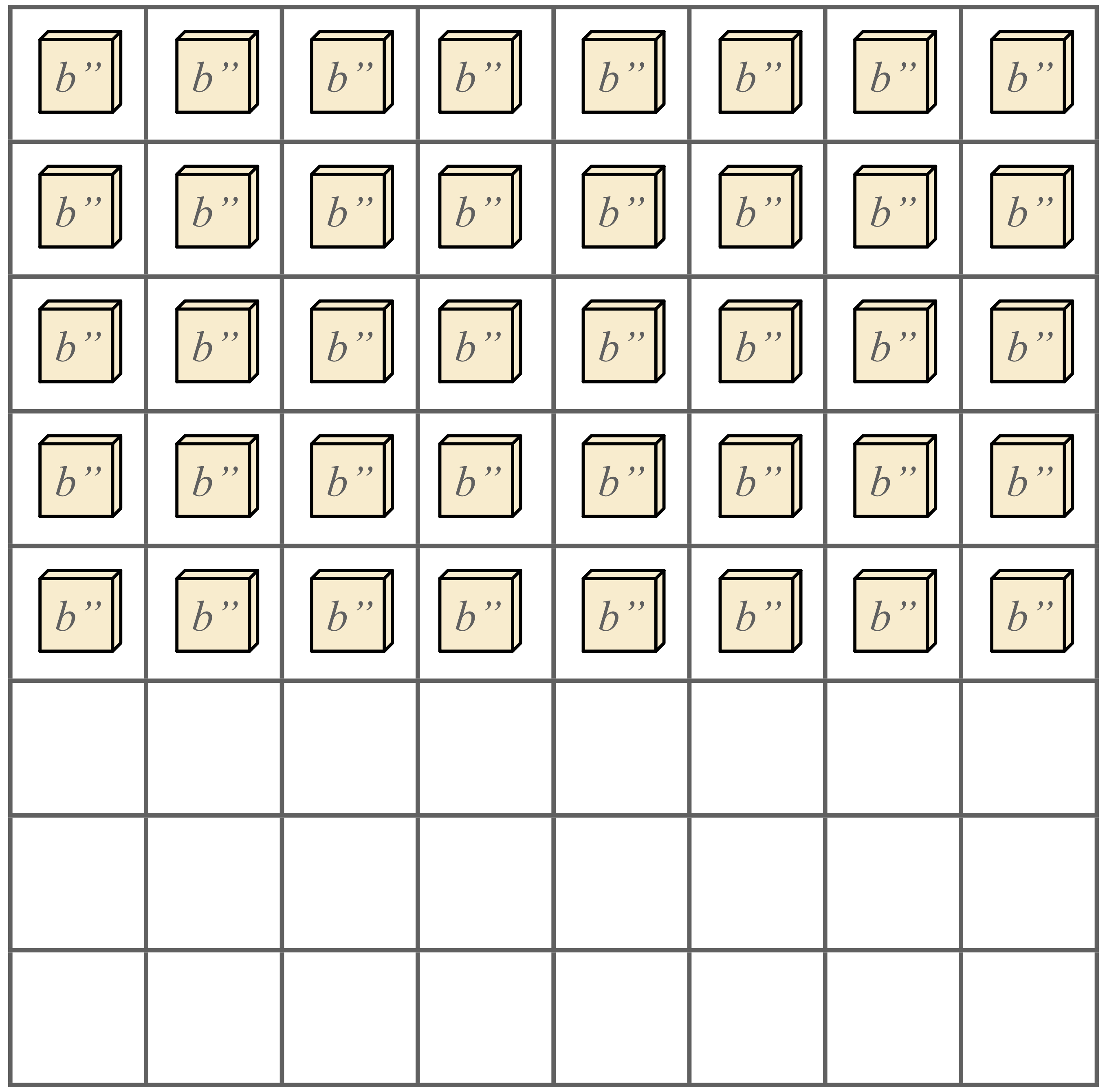


parallel computing grid per brick

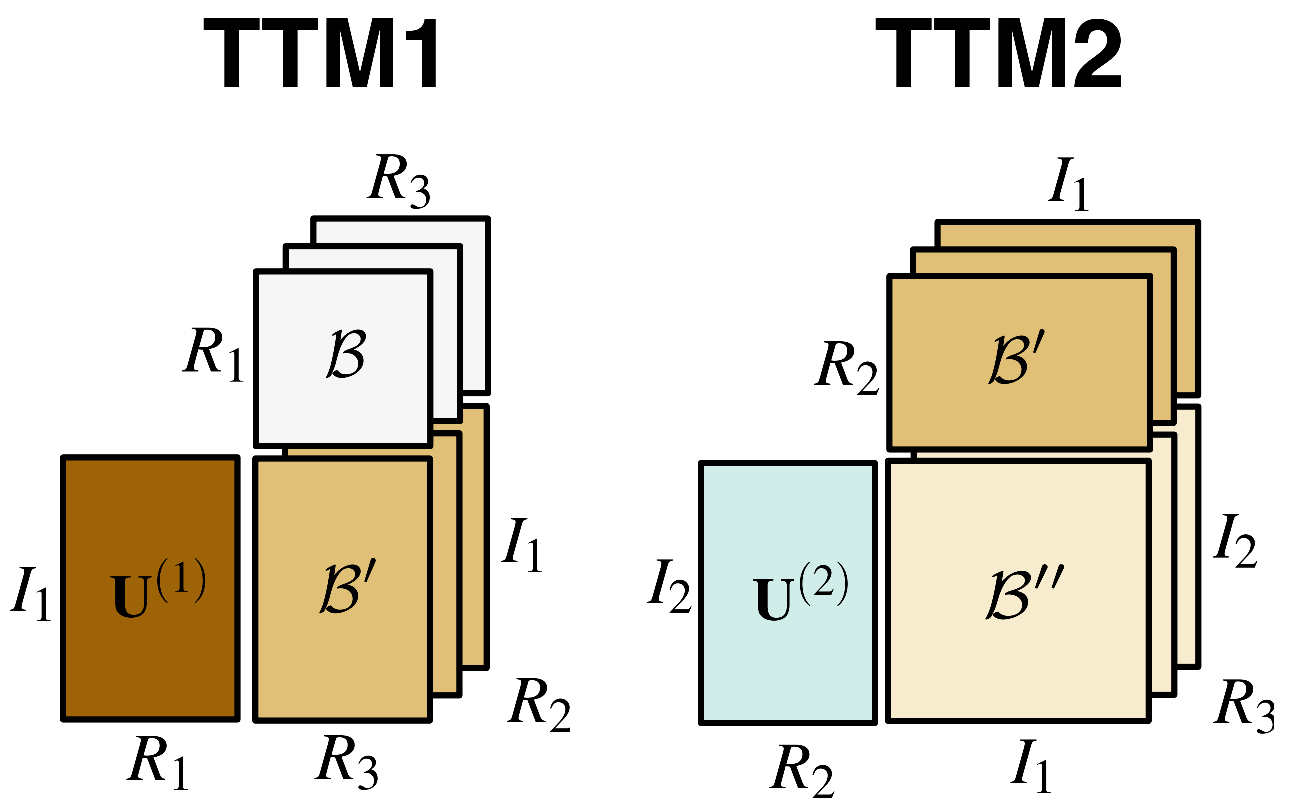


Compute Intermediate Tensor B”

[Suter et al., 2011]

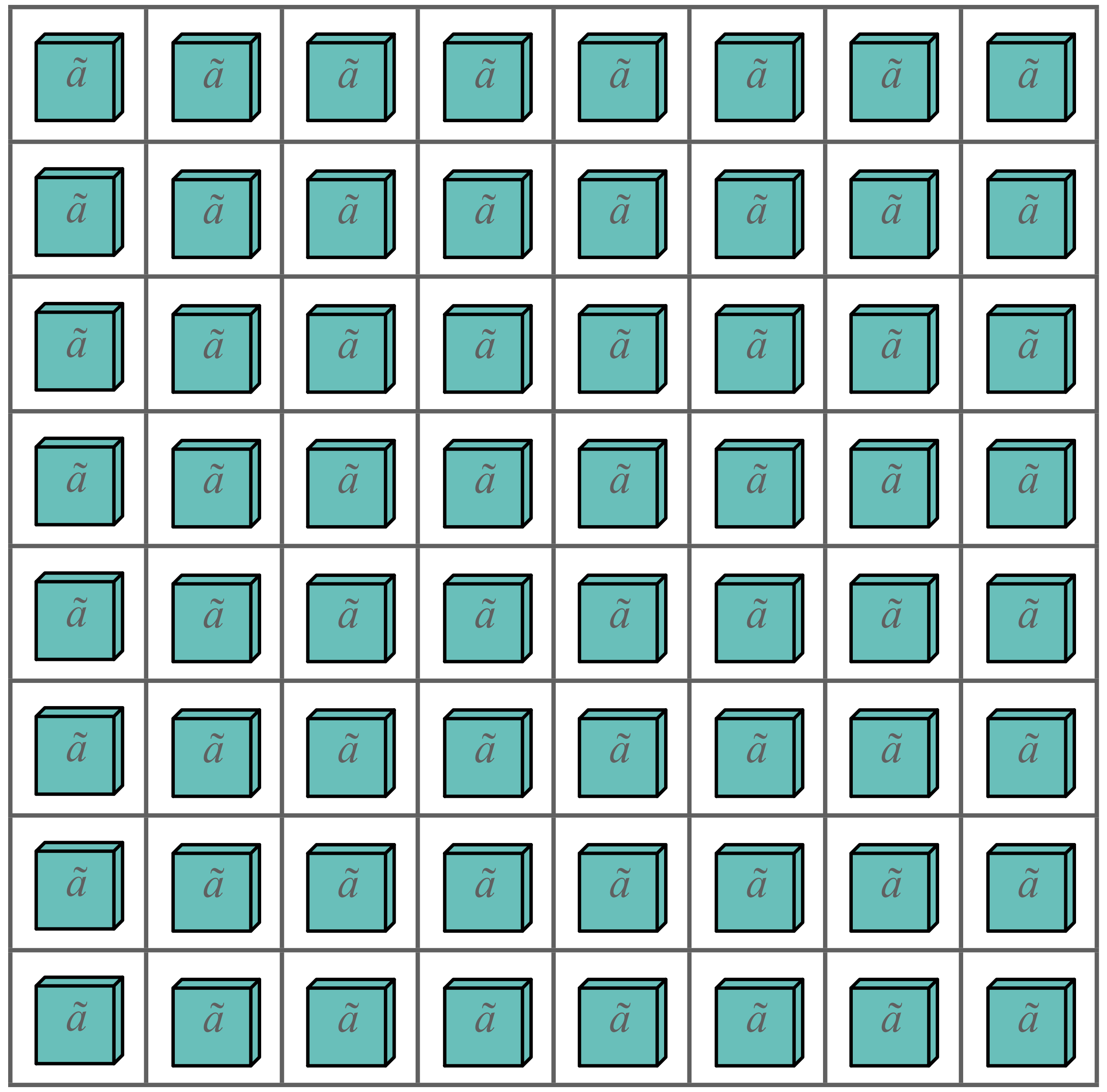


parallel computing grid per brick

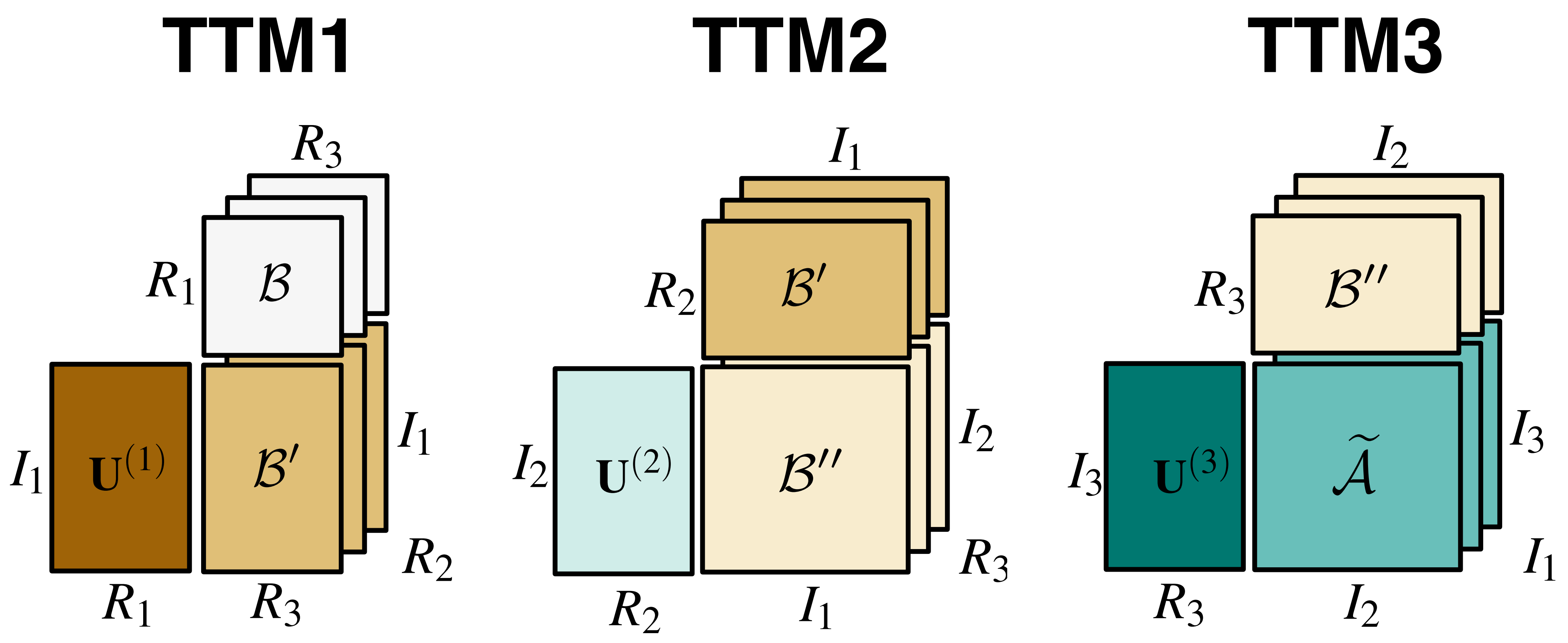


Compute Approximated Tensor Ã

[Suter et al., 2011]

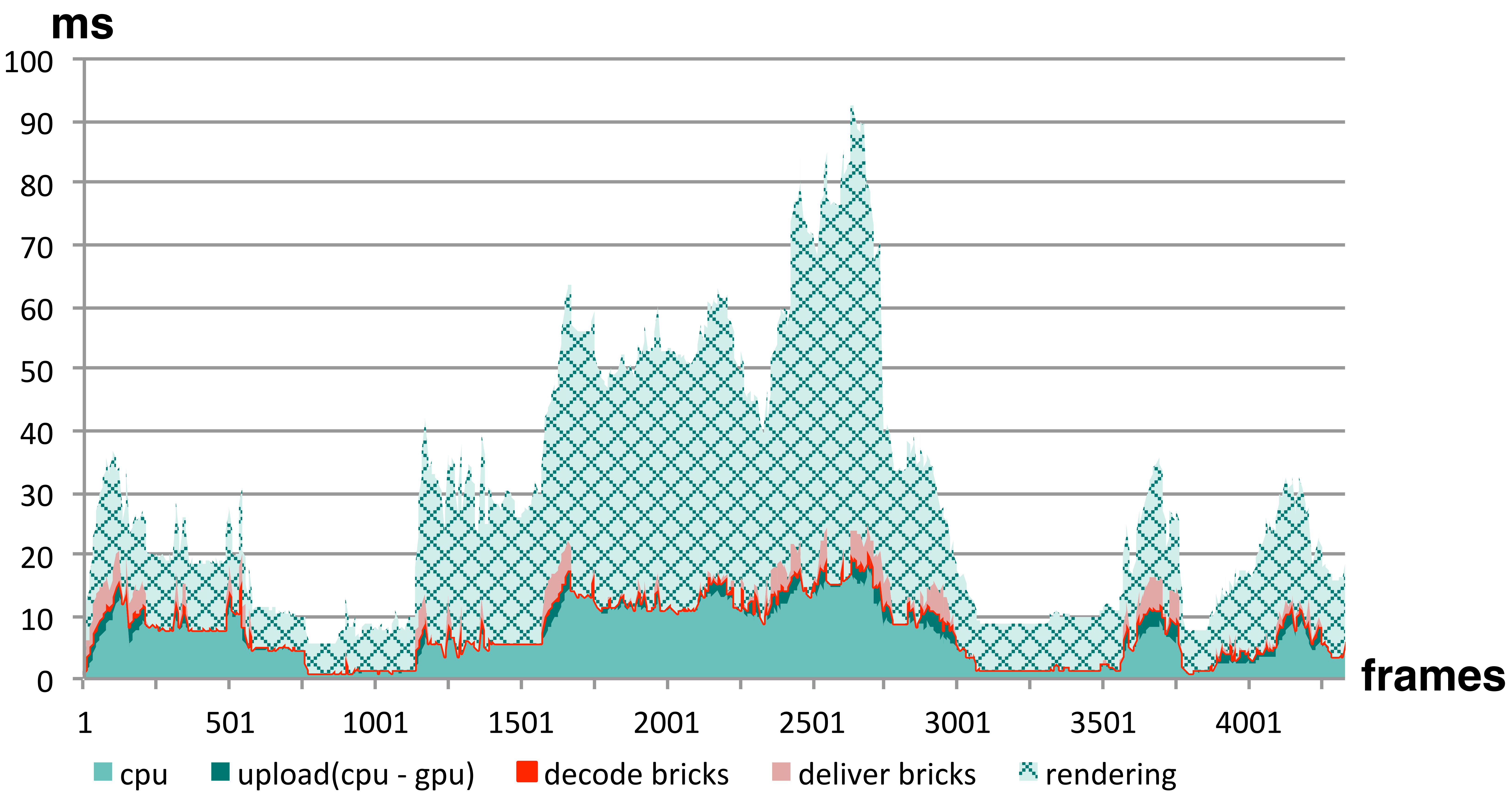


parallel computing grid per brick

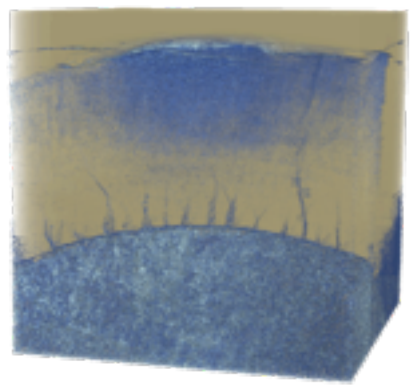


Reconstruction Performance

[Suter et al., 2011]

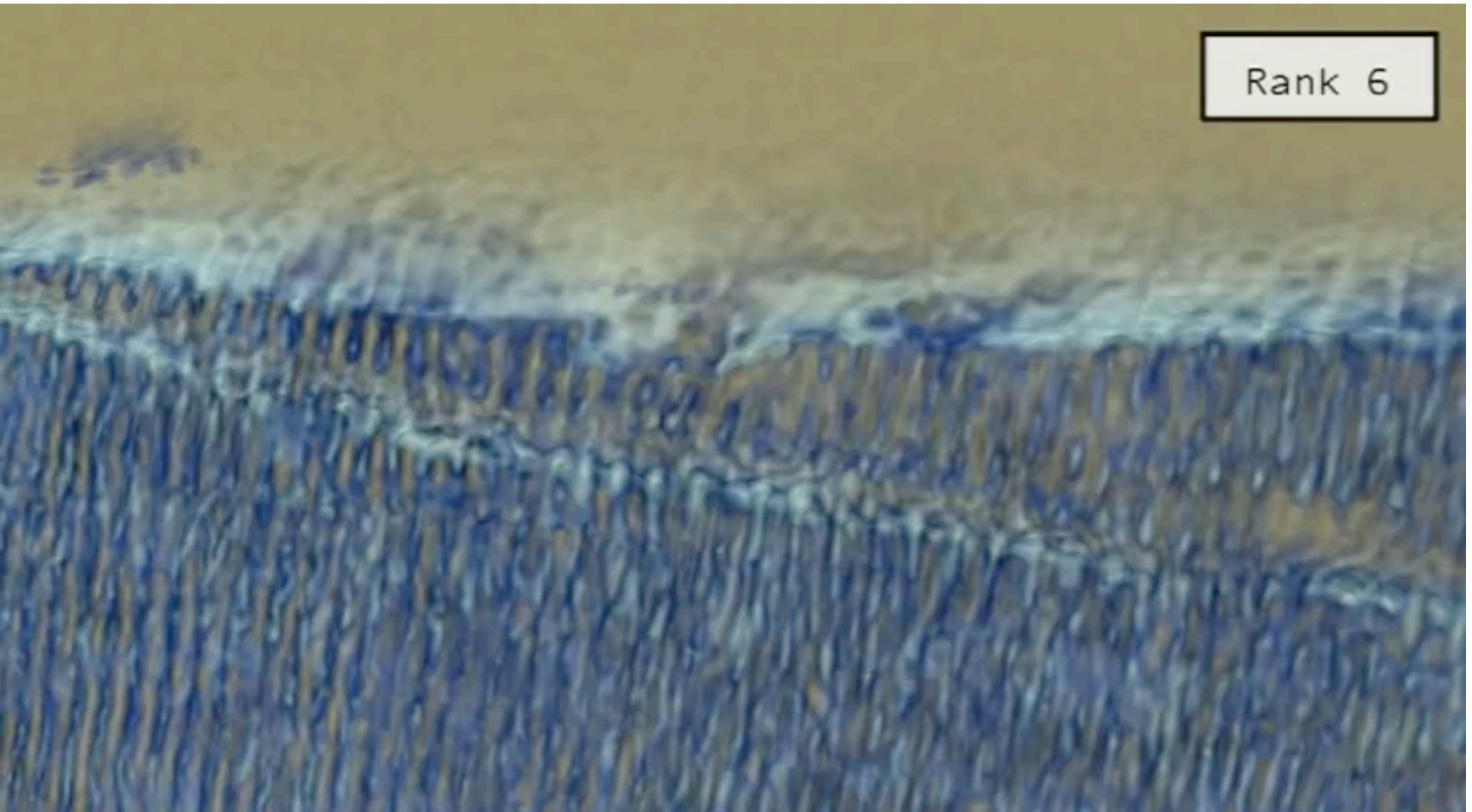


2048^3



- Intel Core 2 E8500 3.2GHz Linux PC, 4GB memory
- NVIDIA GeForce GTX 480, 1.5GB memory

great ape molar (17GB -> 5.5 GB)

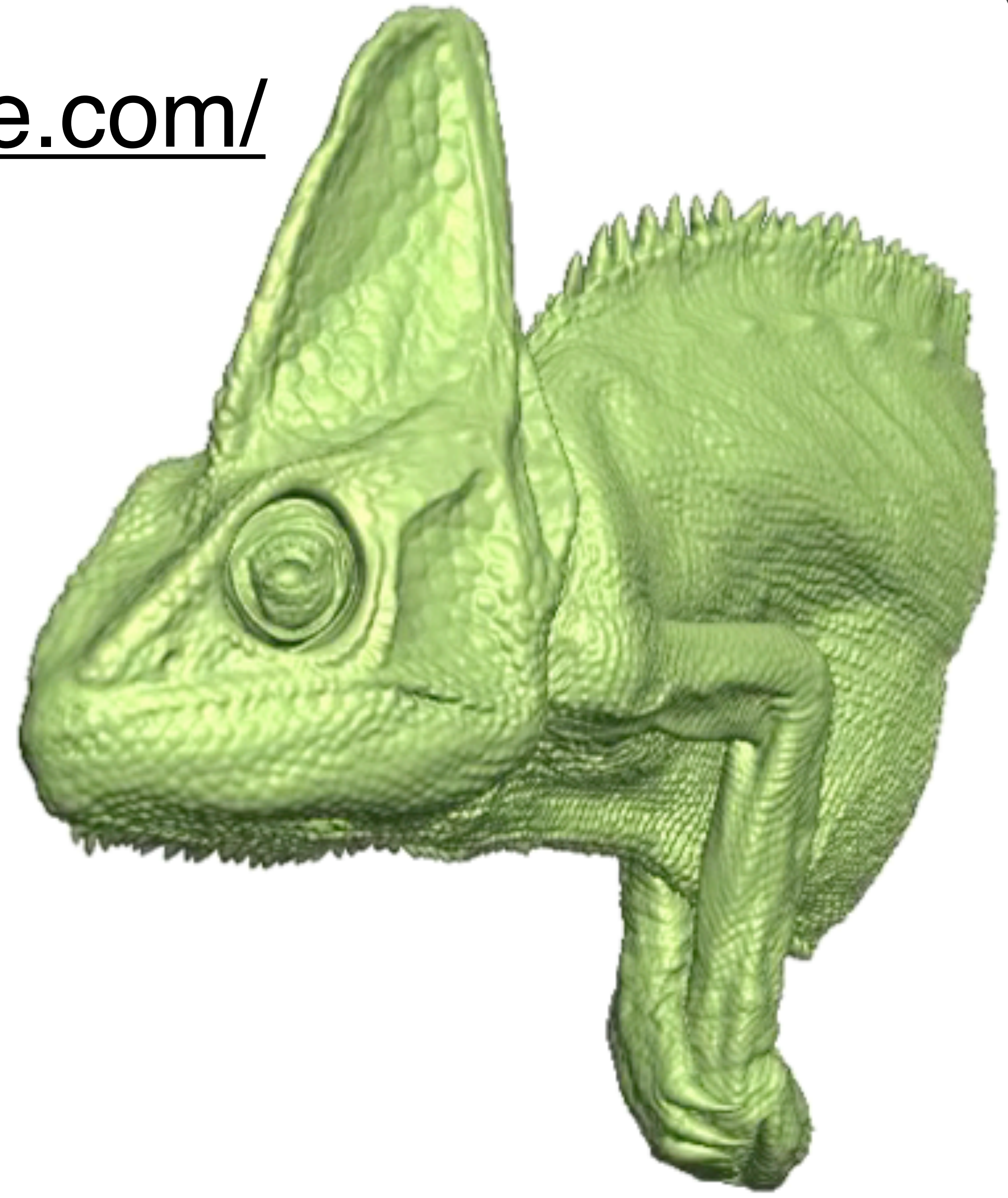


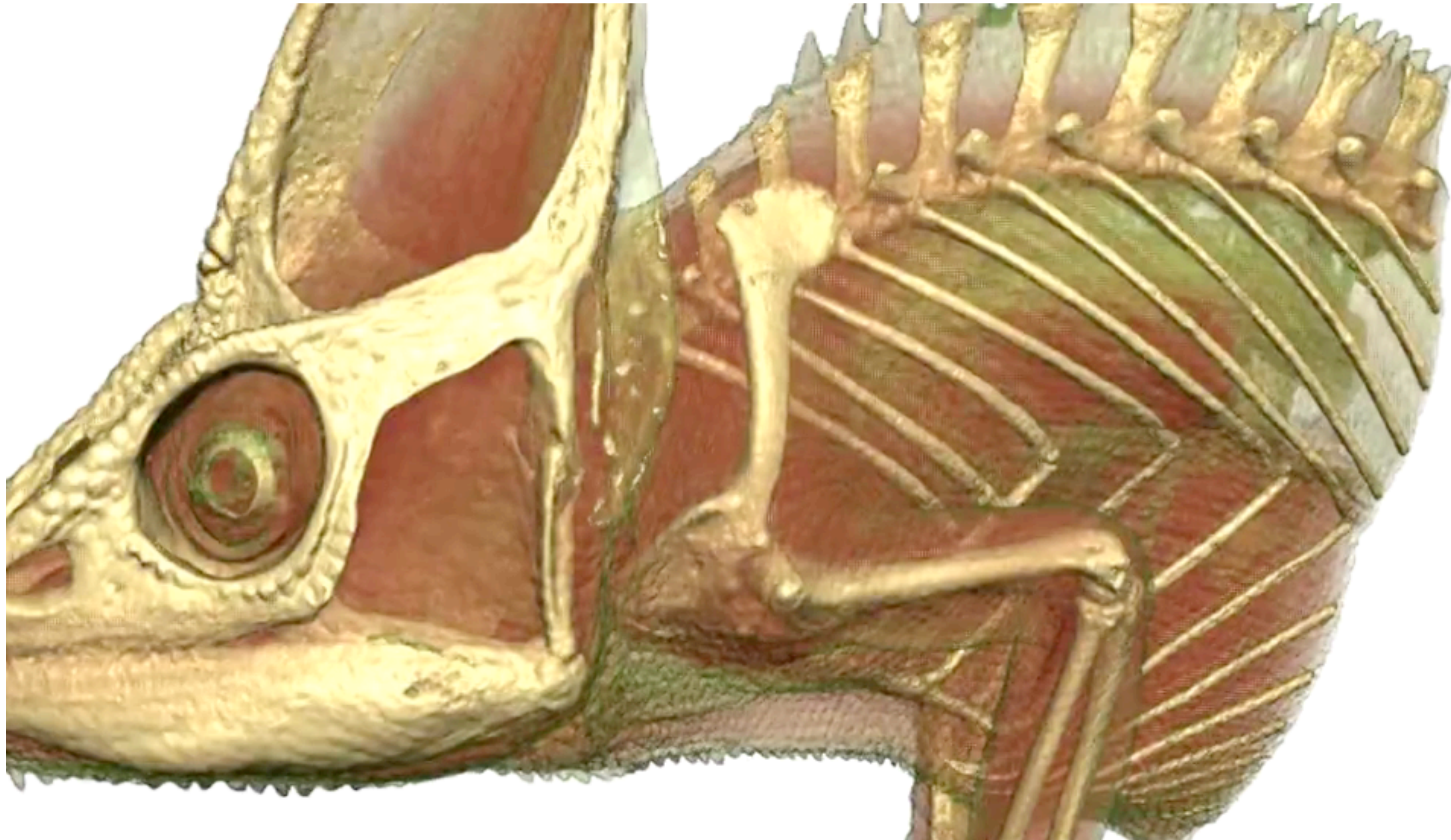
demo videos

http://www.youtube.com/user/VMMLuzh

[Suter et al., 2011]

chameleon (2 GB -> 230 MB)





Overview of TA in Visualization