Tutorial: Tensor Approximation in Visualization and Graphics

Implementation Examples in Scientific Visualization

Renato Pajarola, Susanne K. Suter, and Roland Ruiters
Tutorial Continued...

- **Tuesday May. 7 from 9:00 to 10:40**
- **Location: Room B.1**
  - Implementation Examples in Scientific Visualization (Suter, 25min)
  - Graphics Applications (Ruiters, 30min)
  - Clustering and Sparsity (Ruiters, 25min)
  - Summary/Outlook (Pajarola, 10min)
Outline

• Part 1: Typical decomposition algorithms/operations
• Part 2: GPU-based tensor reconstruction
Typical TA Operations

tensor approximation

- tensor decomposition
- real-time tensor reconstruction

\[ A = U(1)U(2)U(3) \]

\[ B = R(1)R(2)R(3) \]
Tensor Decomposition

• Create factor matrices
  ‣ higher-order SVD (HOSVD)
    - tensor unfolding
  ‣ alternating least-squares (ALS) algorithms
    - higher-order orthogonal iteration (HOOI)
    - higher-order power method (HOPM)
• Generate core tensor
  ‣ tensor times matrix (TTM) multiplications
Tensor Reconstruction

- Realtime (!) reconstruction
  - tensor times matrix (TTM) multiplications
Tensor: A Multidimensional Array

0\textsuperscript{th}-order tensor

\[ a \]

1\textsuperscript{st}-order tensor

\[ I_1 a \]

\( i_1 = 1, \ldots, I_1 \)

2\textsuperscript{nd}-order tensor

\[ I_1 A \]

\[ I_2 \]

\( i_2 = 1, \ldots, I_2 \)

3\textsuperscript{rd}-order tensor

\[ I_1 \mathcal{A} \]

\[ I_2 \]

\[ I_3 \]

\( i_3 = 1, \ldots, I_3 \)

\[ \ldots \]
SVD Extension to Higher Orders

PARAFAC (parallel factors) [Harshman, 1970]

CANDECOMP (CAND) (canonical decomposition) [Caroll & Chang, 1970]

Tucker

Three-mode factor analysis (3MFA/Tucker3) [Tucker, 1964+1966]

Higher-order SVD (HOSVD) [De Lathauwer et al., 2000a]

CP

coefficients

basis matrices $U^{(n)}$

core tensor $\mathcal{B}$

orthonormal matrices preserved

rank-R decomposition preserved

$\begin{array}{c}
I_1 \\
I_2 \\
I_3 \\
A
\end{array}$

$\begin{array}{c}
U^{(1)} \\
U^{(2)} \\
U^{(3)} \\
R_1 \\
R_2 \\
R_3
\end{array}$

$\begin{array}{c}
R
\end{array}$

$\begin{array}{c}
R
\end{array}$
Part 1:
Typical Decomposition Algorithms and Operations
Downloads

• MATLAB tensor toolbox
  ▸ http://www.sandia.gov/~tgkolda/TensorToolbox

• vmmlib: C++ library for vectors, matrices, and tensor approximation
  ▸ http://vmml.github.io/vmmlib/

• Tensor tutorial notes
  ▸ http://vmml.ifi.uzh.ch/links/TutorTensorAprox.html
Test Dataset: Hazelnut

- A microCT scan of dried hazelnuts
- $I_1 = I_2 = I_3 = 512$
- Values: unsigned char (8bit)
- [Link to the dataset](http://vmml.ifi.uzh.ch/research/datasets.html)
Higher-order SVD (HOSVD)

1. Start HOSVD for mode n
2. Unfold tensor $\mathcal{A}$ along mode n ($\mathcal{A}_n$)
3. Compute the matrix SVD on $\mathcal{A}_n$
4. Set $R_n$ left singular vectors as $U^{(n)}$
5. Stop HOSVD for mode n
6. Mode n matrix $U^{(n)}$

Slices of a Tensor3

- **Frontal slices**
- **Horizontal slices**
- **Lateral slices**

```
[vvmlib] matrix< 512, 512, values_t > slice;

t3.get_frontal_slice_fwd( 256, slice );

t3.get_horizontal_slice_fwd( 256, slice );

t3.get_lateral_slice_fwd( 256, slice );
```
Tensor Unfolding (Matricization)

forward cyclic unfolding

\[ A \]
\[ \mathbf{A}_{(1)} \]
\[ I_1 \]
\[ I_2 \cdot I_2 \]
\[ I_3 \cdot I_2 \]

\[ \mathbf{A}_{(2)} \]
\[ I_1 \cdot I_3 \]
\[ I_2 \cdot I_3 \]
\[ I_3 \cdot I_3 \]

\[ \mathbf{A}_{(3)} \]
\[ I_1 \cdot I_2 \]
\[ I_2 \cdot I_1 \]
\[ I_3 \cdot I_2 \]

backward cyclic unfolding

\[ A \]
\[ \mathbf{A}_{(1)} \]
\[ I_1 \]
\[ I_2 \cdot I_2 \]
\[ I_3 \cdot I_2 \]

\[ \mathbf{A}_{(2)} \]
\[ I_1 \cdot I_3 \]
\[ I_2 \cdot I_3 \]
\[ I_3 \cdot I_3 \]

\[ \mathbf{A}_{(3)} \]
\[ I_1 \cdot I_2 \]
\[ I_2 \cdot I_2 \]
\[ I_3 \cdot I_2 \]


Tensor Unfolding Example

mode-1 unfolding

mode-2 unfolding

mode-3 unfolding
Higher-order SVD (HOSVD)

Large Data Tensors (in vmmlib)

```cpp
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 );
```
Optimize Factor Matrices

- Higher-order orthogonal iteration
  - optimize factor matrix of mode n
  - keep factor matrices of all other modes fixed
  - generate optimized data tensor
    - project original data tensor on the inverted factor matrices of all other modes
  - receive optimized mode-n factor matrix
    - apply HOSVD to the optimized tensor

Optimize Mode-n Factor Matrix

tensor $\mathcal{A}$, matrices $U^{(n)}$

start mode-n optimization

invert all matrices, but mode n

multiply tensor with all inverted matrices (TTMs)

stop mode-n optimization

optimized tensor $P_n$
Higher-order Orthogonal Iteration (HOOI)

1. **Start HOOI ALS**
   - Input tensor \( \mathbf{A} \)

2. **Init matrices** \( \mathbf{U}^{(n)} \)
   - Random, HOSVD

3. **Compute max Frobenius norm**

4. **Set convergence criteria**

5. **Convergence?**
   - Yes: **Stop iterations**
   - No: **Mode-n optimization**

6. **Mode-n optimized tensor** \( \mathbf{P}_n \)

7. **Compute new mode-n matrix** (HOSVD on \( \mathbf{P}_n \))

8. **Compute core tensor** \( \mathbf{B} \)

9. **Compute fit**
Tensor Times Matrix Multiplication

\[ \mathbf{A}(n) = \mathbf{C} \mathbf{B}(n) \]

\[ \mathcal{A} = \mathcal{B} \times_n \mathbf{C} \]

**n-mode product**
[De Lathauwer et al., 2000a]

\[ (\mathcal{B} \times_n \mathbf{C})_{i_1 \ldots i_{n-1} i_n i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} b_{i_1 i_2 \ldots i_{n-1} i_{n+1} \ldots i_N} \cdot c_{j_n i_n} \]
Example TTM: Core Computation

- Three consecutive TTM multiplications
- For orthogonal matrices, use the transposes of the three factor matrices (otherwise the (pseudo)-inverses)
Part 2: GPU Reconstruction

Parallel Tensor Reconstruction

parallel computing grid per brick

\[ \tilde{a}_{i_1 i_2 i_3} = \sum_{r_1} \sum_{r_2} \sum_{r_3} b_{r_1 r_2 r_3} \cdot u_{i_1 r_1}^{(1)} \cdot u_{i_2 r_2}^{(2)} \cdot u_{i_3 r_3}^{(3)} \]

\( \uparrow \)

computational cost per voxel is cubic: \( O(R^3) \)
Faster Parallel Tensor Reconstruction

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

parallel computing grid per brick
Faster Parallel Tensor Reconstruction

parallel computing grid per brick

store intermediate results (B’ and B’”)

computational cost per voxel is linear: $O(R)$
Compute Intermediate Tensor B’

parallel computing grid per brick
Compute Intermediate Tensor $B''$

parallel computing grid per brick
Compute Approximated Tensor $\tilde{A}$

parallel computing grid per brick
Reconstruction Performance

- 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)
Conclusion

• Critical implementation steps

• Tensor decomposition
  ‣ initial decomposition or a large input tensor
    – memory mapping
  ‣ tensor times matrix (TTM) multiplications
    – parallel matrix matrix multiplications
  ‣ higher-order SVD

• Tensor reconstruction
  ‣ GPU implementation of TTM