Visual Data Processing in the Tensor Compressed Domain

Rafael Ballester-Ripoll and Renato Pajarola

rballester@ifi.uzh.ch, *pajarola@ifi.uzh.ch*

18th January 2016

Universität Zürich[UZH]

VISUALIZATIONAND MULTIMEDIALAB

Table of Contents

VISUALIZATIONAND
MULTIMEDIALAB

Section 1

Tensor Methods for Interactive Visualization

Introduction

- Large-scale interactive visualization: **high rank** data over regular grids
  - Computer tomography, simulations, etc.
  - We tolerate (and encourage) approximations: $\|\mathscr{A} - \widetilde{\mathscr{A}}\| \leq \epsilon$

Introduction

- Large-scale interactive visualization: **high rank** data over regular grids
  - Computer tomography, simulations, etc.
  - We tolerate (and encourage) approximations: $\|\mathscr{A} - \widetilde{\mathscr{A}}\| \leq \epsilon$

- Asymmetric pipeline:
  - Slow decomposition is acceptable (offline stage)
  - But **fast reconstruction** is critical (online stage). We use parallel algorithms for graphics cards

Introduction

- Large-scale interactive visualization: **high rank** data over regular grids
  - Computer tomography, simulations, etc.
  - We tolerate (and encourage) approximations: $\|\mathscr{A} - \widetilde{\mathscr{A}}\| \leq \epsilon$
- Asymmetric pipeline:
  - Slow decomposition is acceptable (offline stage)
  - But **fast reconstruction** is critical (online stage). We use parallel algorithms for graphics cards
- In volume rendering: data sets of size I^3, with I large (e.g. 2048).
  - Possible added dimension(s): features (RGB color, X-ray density), time, etc.

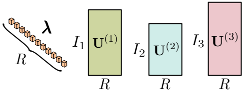
VISUALIZATIONAND
MULTIMEDIALAB

Useful Models

- We use multilinear methods that work for several models

Useful Models

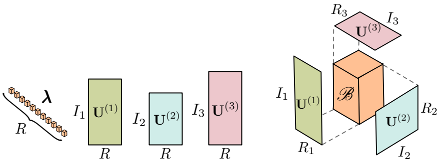
- We use multilinear methods that work for several models



CP

Useful Models

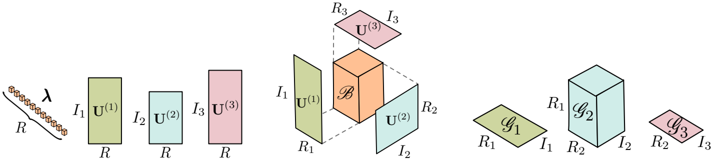
- We use multilinear methods that work for several models



CP Tucker

Useful Models

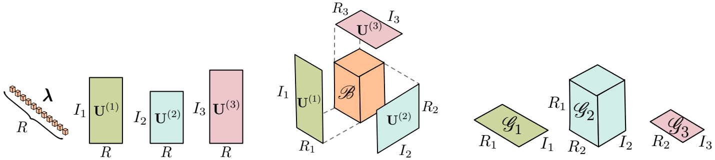
- We use multilinear methods that work for several models



CP Tucker Tensor Train

Useful Models

- We use multilinear methods that work for several models



CP   Tucker   Tensor Train

- In the interactive visualization literature, **Tucker/HOSVD** has been the most common
  - CP has expensive reconstruction (usually high rank $R \gg I$)
  - Tensor Train is quite recent; until now mostly applied for large dimensionality problems

Compression Quality

- The Tucker decomposition has competitive compression accuracy
  - Its bases (factor matrices) are learned (data-dependent)
  - For 3+ dimensions, they only take a small fraction of the total memory

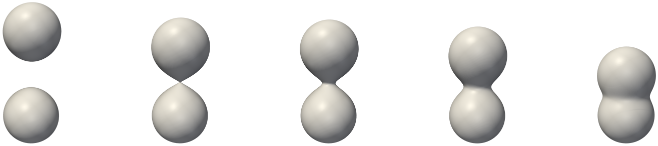
[Wu et al., 2008]
[Suter et al., 2011]
[Ballester-Ripoll and Pajarola, 2015]

Smooth Feature Compression

- At high compression rates, tensor decompositions are good at **preserve visual features**

Smooth Feature Compression

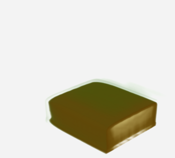
- At high compression rates, tensor decompositions are good at **preserve visual features**
- One way to see it: **isosurfaces**
  - For example, spheres are isosurfaces of multivariate Gaussians (rank-1)



Smooth isosurfaces from a rank-1 function

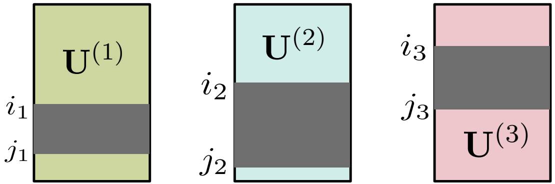
Smooth Feature Compression

- **Different ranks select different features at different scales**
  - Example: bonsai (256^3), from 1 to 256 Tucker ranks



Spatial Selectivity

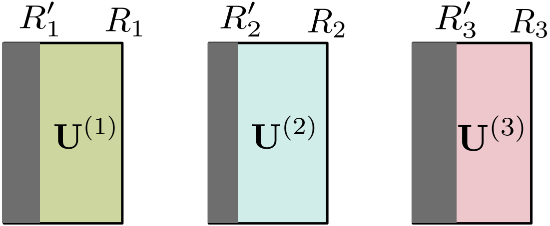
- To reconstruct the subregion $[i_1, j_1] \times [i_2, j_2] \times [i_3, j_3]$:



- Several axis-aligned spatial transformations are possible: translation, stretching, projection, etc.

Rank Selectivity

- Rank selection for interactive level-of-detail [Suter et al., 2013]: Tucker core from $\mathbb{R}^{R_1 \times R_2 \times R_3}$ to $\mathbb{R}^{R'_1 \times R'_2 \times R'_3}$

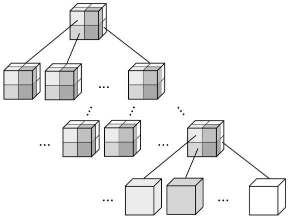


Multiresolution Tucker Compression

- 3D Tucker rank-R reconstruction cost: $I^3R + I^2R^2 + IR^3$
- $O(R)$ operations per voxel result

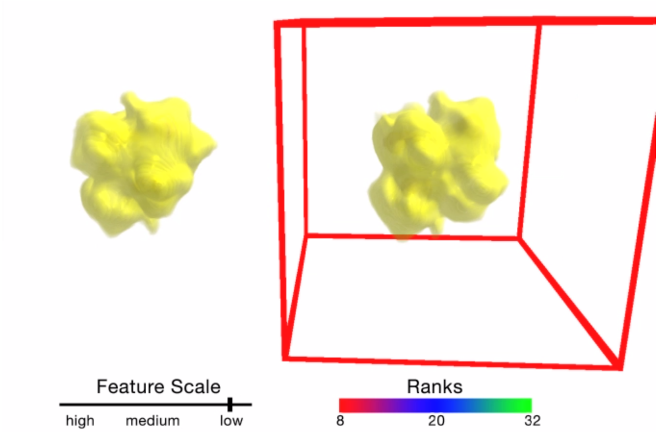
Multiresolution Tucker Compression

- 3D Tucker rank-R reconstruction cost: $I^3R + I^2R^2 + IR^3$
- $O(R)$ operations per voxel result
- Octree: partition I^3 volume into bricks of size B^3 [Suter et al., 2013]:

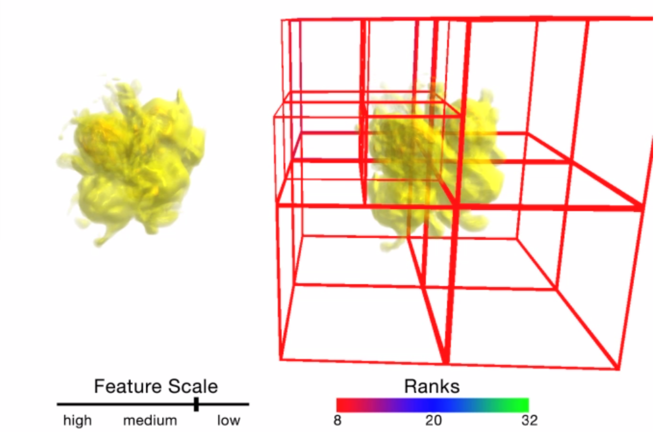


- One Tucker core per brick: **speeds up** reconstruction by a factor $\approx I/B$

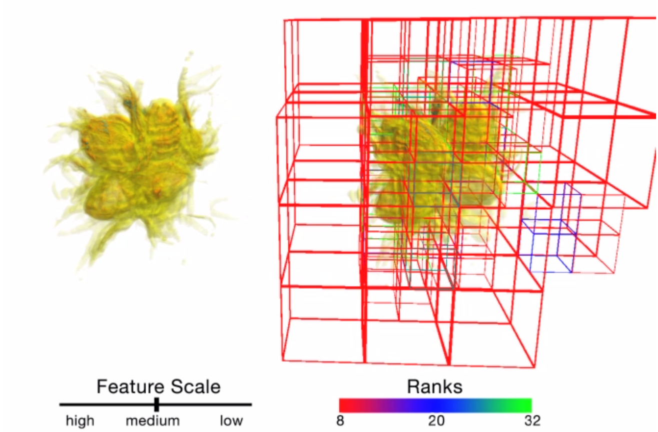
Multiresolution Tucker Visualization



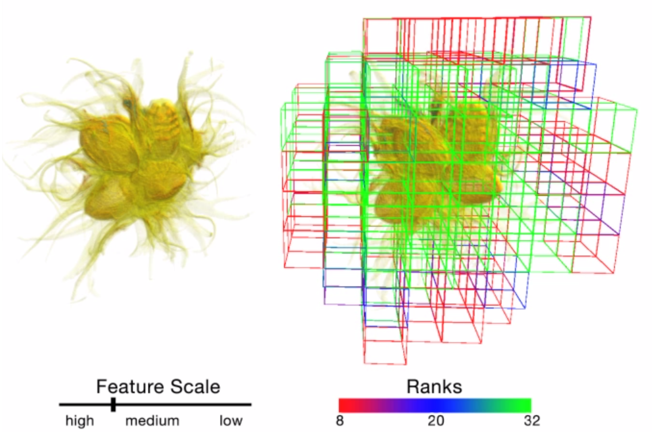
Multiresolution Tucker Visualization



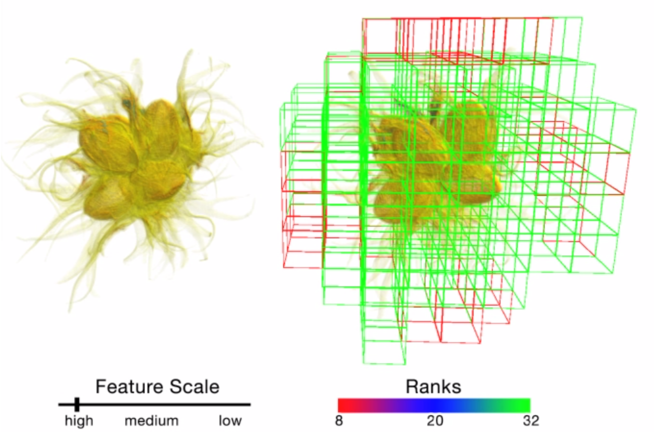
Multiresolution Tucker Visualization



Multiresolution Tucker Visualization



Multiresolution Tucker Visualization



Section 2

Multiresolution Filtering

Multiresolution Filtering

- Problem: filter over different resolution levels
  - Lower resolution requires downsampling the filter kernel

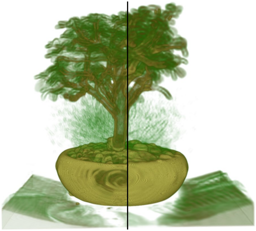
Multiresolution Filtering

- Problem: filter over different resolution levels
  - Lower resolution requires downsampling the filter kernel
- Example: 3D Sobel operator for edge detection
  - Size $3 \times 3 \times 3$, **cannot be downsampled**!

$$
h_z = \begin{matrix}
\begin{bmatrix} -1 & -2 & -1 \\ -2 & -4 & -2 \\ -1 & -2 & -1 \end{bmatrix} \\
\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\
\begin{bmatrix} +1 & +2 & +1 \\ +2 & +4 & +2 \\ +1 & +2 & +1 \end{bmatrix}
\end{matrix}
$$

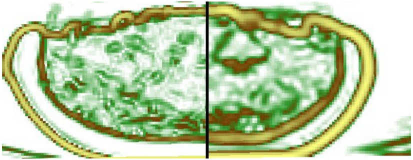
Multiresolution Filtering

- Problem: filter over different resolution levels
  - Lower resolution requires downsampling the filter kernel
- Example: 3D Sobel operator for edge detection
  - Size $3 \times 3 \times 3$, cannot be downsampled!
- With naive filtering, **artifacts appear**



Multiresolution Filtering

- Problem: filter over different resolution levels
  - Lower resolution requires downsampling the filter kernel
- Example: 3D Sobel operator for edge detection
  - Size $3 \times 3 \times 3$, cannot be downsampled!
- With naive filtering, **artifacts appear**



Tensor Convolution

- Separable linear transforms can be computed on the corresponding factors
  - Convolution theorem. Cosine and Fourier transforms, separable wavelets, etc.

VISUALIZATIONAND
MULTIMEDIALAB

Tensor Convolution

- Separable linear transforms can be computed on the corresponding factors
  - Convolution theorem. Cosine and Fourier transforms, separable wavelets, etc.

- **We can filter via the Tucker factors**
  - Long-known property, already in 2D for the SVD

Example: Sobel

- The 3D Sobel operator is a combination of 3 rank-1 filters:

$$\widehat{\mathscr{A}}(\boldsymbol{i}) = \sqrt{(\mathscr{A} * h_x)(\boldsymbol{i})^2 + (\mathscr{A} * h_y)(\boldsymbol{i})^2 + (\mathscr{A} * h_z)(\boldsymbol{i})^2}$$

with

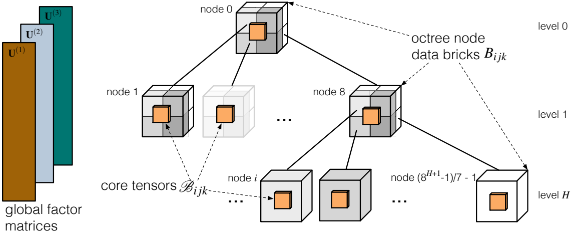
$$\begin{cases} h_x = \boldsymbol{u} \circ \boldsymbol{v} \circ \boldsymbol{v} \\ h_y = \boldsymbol{v} \circ \boldsymbol{u} \circ \boldsymbol{v} \\ h_z = \boldsymbol{v} \circ \boldsymbol{v} \circ \boldsymbol{u} \end{cases}$$

and

$$\boldsymbol{u} = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, \boldsymbol{v} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$$

Multiresolution Filtering

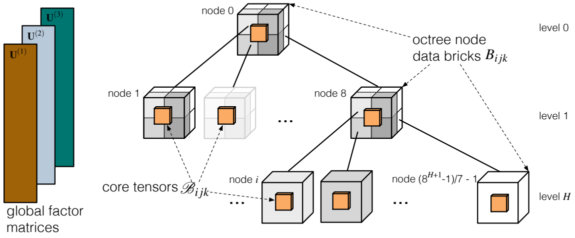
- Idea: keep global-resolution matrices



VISUALIZATIONAND MULTIMEDIALAB

Multiresolution Filtering

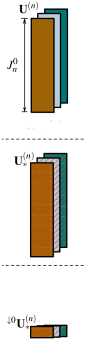
- Idea: keep global-resolution matrices



- How?
  - We decompose the whole volume once $\rightarrow$ global factors
  - Then we compress octree bricks by projection with the corresponding global factor chunks $\rightarrow$ Tucker cores

Filter + Decompression

- Decompressing a filtered brick in low resolution:
  - We first convolve the tall global matrices
  - Then we downscale them as needed
  - After reconstruction, we get the desired resolution

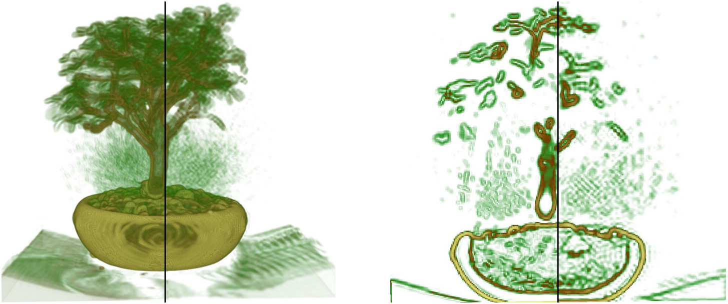


Filter + Decompression

- Filtering cost: $O(NKR)$. **Negligible** compared to reconstruction ($O(I^N R)$)
- And we have to reconstruct anyway for rendering
- GPU implementation (in collaboration with David Steiner):
  - C++ CUDA filtering and decompression
  - For a 2048^3 volume: under 1 ms for a rank-2 filter (difference of Gaussians) of size 20

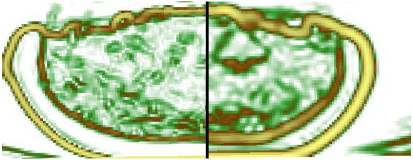
Result

- Smoother response across different levels of detail [Ballester-Ripoll et al., 2016]

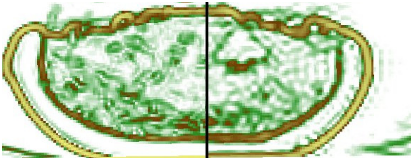


Sobel on two resolution levels

Result (Close-up)



Naive filtering



Multiresolution Tucker

Section 3

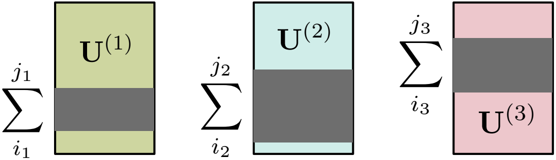
Integrals and Summed Area Tables

Integrals in the Compressed Domain

- Sum over a region $[i_1, j_1] \times [i_2, j_2] \times [i_3, j_3]$:

$$\sum_{i_1}^{j_1} \sum_{i_2}^{j_2} \sum_{i_3}^{j_3} \mathscr{A}$$

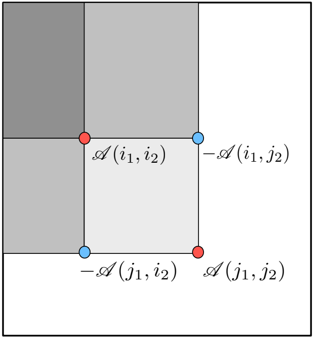
via the sum of Tucker factor rows:



Alternative: Summed Area Tables

- A SAT is just a precomputed definite integral
- E.g., a 2D sum over $[i_1, j_1] \times [i_2, j_2]$ is

$$\mathscr{A}(j_1, j_2) - \mathscr{A}(j_1, i_2) - \mathscr{A}(i_1, j_2) + \mathscr{A}(i_1, i_2)$$



VISUALIZATION AND MULTIMEDIA LAB

Alternative: Summed Area Tables

- Instead of computing sums of rows, it's better to use a SAT. We can either
  - Compress a SAT instead of the original

  or
  - Via factor matrix manipulation: compute the accumulative sum, columnwise

Alternative: Summed Area Tables

- Instead of computing sums of rows, it's better to use a SAT. We can either
  - Compress a SAT instead of the original

  or
  - Via factor matrix manipulation: compute the accumulative sum, columnwise

- For reconstruction, we **subtract the relevant rows**

$$\Rightarrow \boldsymbol{u}^{(n)} := \mathbf{U}^{(n)}(j_n, :) - \mathbf{U}^{(n)}(i_n, :)$$

Alternative: Summed Area Tables

- Instead of computing sums of rows, it's better to use a SAT. We can either
  - Compress a SAT instead of the original

  or
  - Via factor matrix manipulation: compute the accumulative sum, columnwise

- For reconstruction, we **subtract the relevant rows**

$$\Rightarrow \boldsymbol{u}^{(n)} := \mathbf{U}^{(n)}(j_n, :) - \mathbf{U}^{(n)}(i_n, :)$$

- Only 2 rows must be visited per dimension (instead of K)
  - $O(NR)$ operations

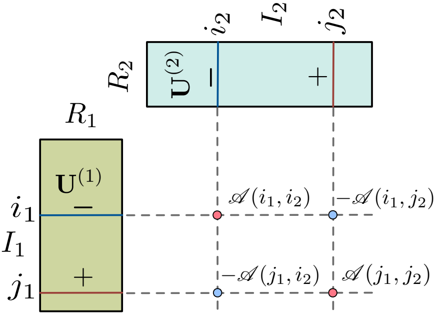
VISUALIZATIONAND MULTIMEDIALAB

SAT Query (2 Dimensions)

- 2D SAT: sum over $[i_1, j_1] \times [i_2, j_2]$ is

$$\mathscr{A}(j_1, j_2) - \mathscr{A}(j_1, i_2) - \mathscr{A}(i_1, j_2) + \mathscr{A}(i_1, i_2)$$

- We subtract rows in $\mathbf{U}^{(1)}$ and $\mathbf{U}^{(2)}$: the signs are combined to give the correct formula:



SAT Query (N Dimensions)

- SAT for N dimensions [Tapia, 2011]:

$$\sum_{i_1}^{j_1-1} \cdots \sum_{i_N}^{j_N-1} f(x_1, ..., x_N) = \sum_{p \in \{0,1\}^N} (-1)^{N-\|p\|_1} \cdot \mathscr{A}[p]$$

SAT Query (N Dimensions)

- SAT for N dimensions [Tapia, 2011]:

$$\sum_{i_1}^{j_1-1} \cdots \sum_{i_N}^{j_N-1} f(x_1, ..., x_N) = \sum_{p \in \{0,1\}^N} (-1)^{N-\|p\|_1} \cdot \mathscr{A}[p]$$

- In the Tucker factors, $\mathscr{B}$ times a sequence of vectors (i.e. tensor **contraction**) produces the desired scalar:

$$\mathscr{B} \times_1 \boldsymbol{u}^{(1)} \times_2 ... \times_N \boldsymbol{u}^{(N)}$$

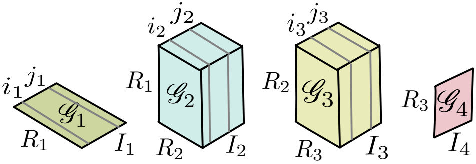
where

$$\boldsymbol{u}^{(n)} := \mathbf{U}^{(n)}(j_n, :) - \mathbf{U}^{(n)}(i_n, :)$$

VISUALIZATIONAND MULTIMEDIALAB

SAT Query (Tensor Train)

- For the Tensor Train, we have to subtract core slices instead of matrix rows



- Subtraction more expensive ($O(NR^2)$ operations)

VISUALIZATIONAND MULTIMEDIALAB

Time Costs

	Without SAT	**With SAT**
Uncompressed	$O(K^N)$	2^N
Tucker	$O(NKR) + O(R^N)$	$O(NR) + O(R^N)$
Tensor Train	$O(NKR^2) + O(R^2)$	$O(NR^2) + O(R^2)$

- Tensor Train is always **linear** with N
  - By using SAT we save a factor K

VISUALIZATION AND MULTIMEDIA LAB

Space Costs

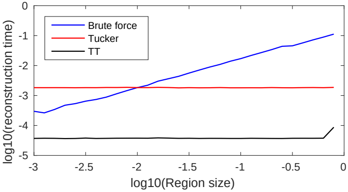
Uncompressed	$O(I^N)$
Tucker	$O(NIR) + O(R^N)$
Tensor Train	$O(NIR^2)$

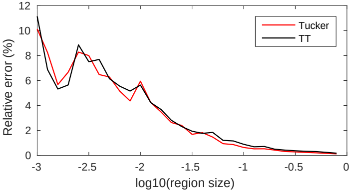
- For the same accuracy, Tensor Train's R often gets larger than Tucker's

Application: Histograms

- We used this to retrieve histograms [Ballester-Ripoll et al., 2016]
- Main principle: integral histogram [Porikli, 2005]
  - It uses one SAT per color bin. All SATs are stacked along an extra dimension
  - The look-up for a region gives a vector: the histogram over that area
  - **Expensive storage**. Bonsai example: 256^3 voxels and 64 color bins $\rightarrow 4$ gigabytes
  - **Highly compressible**

Some Results (I)





- Rectangular regions from 0.1% to 100% of the input size
- 256^3 voxels and 64 bins $\rightarrow$ tensor train reconstruction under 0.1ms (MATLAB implementation)
- $\approx$ 1MB (16x reduction compared to the original Bonsai)

Some Results (II)

- **Non-rectangular queries are possible**: just filter in the compressed domain
  - Unlike many histogram look-up algorithms

Some Results (II)

- **Non-rectangular queries are possible**: just filter in the compressed domain
  - Unlike many histogram look-up algorithms

- We use the identity

$$f * h = f' * \int h$$

where h is the input, $\int h$ is the SAT, and f defines the target region

Some Results (II)

- **Non-rectangular queries are possible**: just filter in the compressed domain
  - Unlike many histogram look-up algorithms

- We use the identity

$$f * h = f' * \int h$$

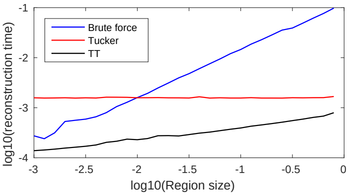
where h is the input, $\int h$ is the SAT, and f defines the target region

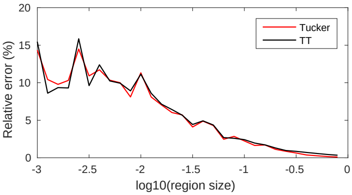
- For non-separable regions: use tensor dot product formulas (Tucker, TT)
  - Speed highly dependent on the region's rank

VISUALIZATIONAND MULTIMEDIALAB

Some Results (II)

- Results for Gaussian regions:





Conclusions

- Tensor algorithms in real-time applications need very efficient reconstruction
  - Achieved via parallelization, data partitioning, etc.
- Operating in the tensor bases is a powerful trick
  - Usually very fast, flexible, and matching our needs
- Tensor decompositions are a promising framework for interactively visualizing data sets
  - Especially, with increasing size and number of dimensions

Rafael Ballester-Ripoll
rballester@ifi.uzh.ch

Renato Pajarola
pajarola@ifi.uzh.ch

http://www.ifi.uzh.ch/vmml.html

Universität Zürich[UZH]

VISUALIZATIONAND
MULTIMEDIALAB