4D Medical Image Processing with CUDA

Anders Eklund, PhD, Mats Andersson, PhD, Hans Knutsson, PhD

Linköping University
Wanderine Consulting
Sweden

Agenda

4D Data ?

fMRI – functional magnetic resonance imaging

- Preprocessing
- Non-parametric fMRI analysis
- Real-time fMRI

True 4D Image Denoising

- Adaptive filtering in 4D
- Non-separable 4D convolution

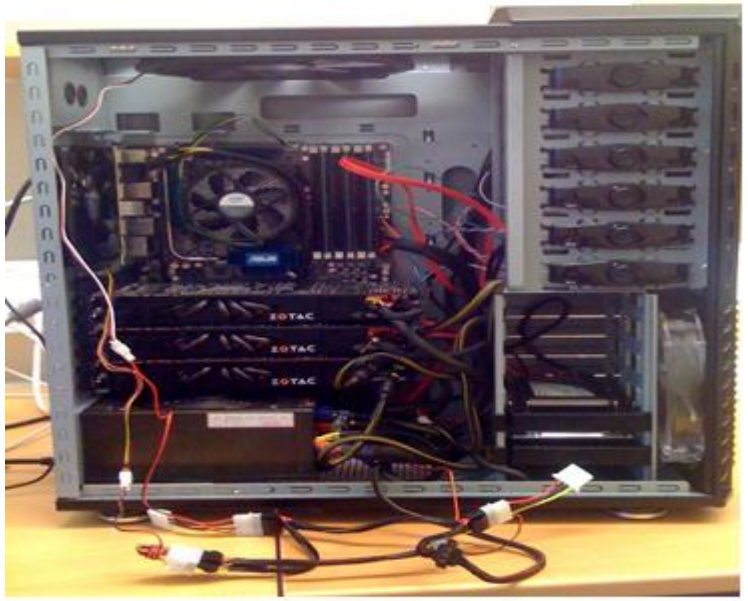
Hardware

Intel Xeon 2.4 GHz, 24 GB

1 x Nvidia GTX 580, 3.0 GB

2 x Nvidia GTX 480, 1.5 GB

1500 W power supply



Software

CUDA 4.0 combined with Matlab mex-files

Linux Fedora 14

MeVisLab (for visualization)

4D Data?

A number of medical imaging modalities can collect 4D data

Several volumes (3D) over time (1D)

Three spatial dimensions, one temporal dimension

Magnetic resonance imaging (MRI)

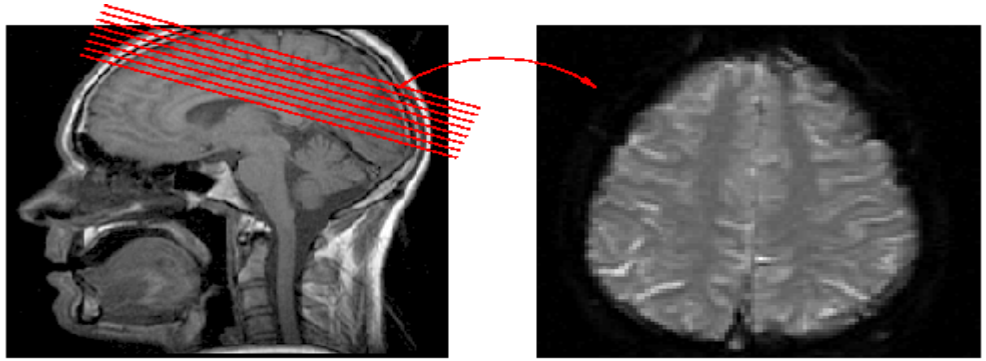
Computed tomography (CT)

Ultrasound (US)

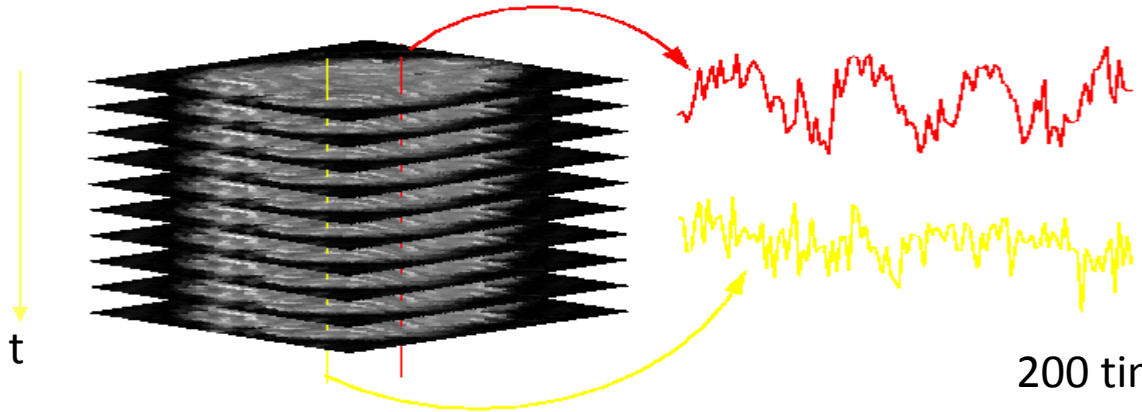
fMRI

- Functional magnetic resonance imaging
- Estimate brain activity from magnetic resonance images of the brain
- Active neurons consume more oxygen, alters the magnetic properties of the blood

- 4D data ~ 64 x 64 x 30 x 200
- Spatial resolution ~ 3 x 3 x 3 mm
- Temporal resolution ~ 0.5 Hz



64 x 64



200 time points

Preprocessing

Slice timing correction, compensate for time differences between slices

Motion correction, compensate for head movement

Spatial smoothing, use information from neighbouring voxels

Detrending, remove unwanted time trends

All these preprocessing steps can be performed in parallel

Eklund et al., fMRI analysis on the GPU – Possibilities and challenges, Computer Methods and Programs in Biomedicine, 2012

Motion correction

Volume registration of ~200 volumes to a reference volume

Intensity based registration can induce false positives,
due to intensity fluctuations caused by the fMRI signal

We use phase based volume registration, quadrature filters

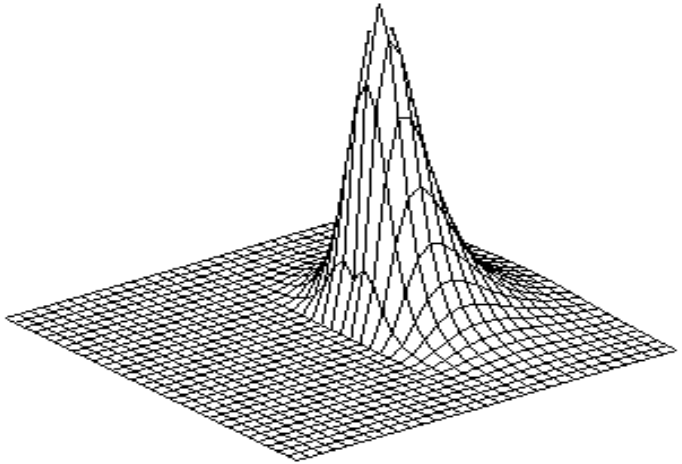
Local phase is invariant to image intensity

64 x 64 x 22 x 200 dataset, motion correction in 1 second
(5 ms per volume)

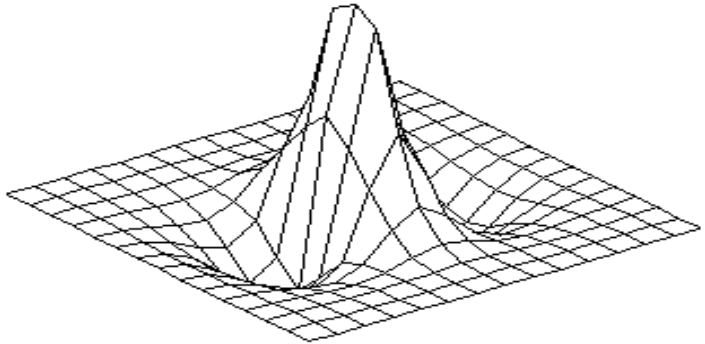
Eklund et al., Phase Based Volume Registration Using CUDA, ICASSP, 2010

Quadrature filters

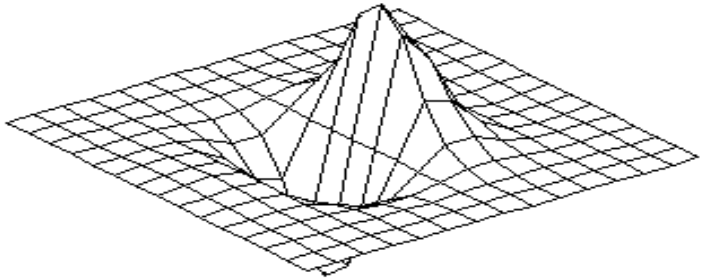
Frequency domain



Spatial domain

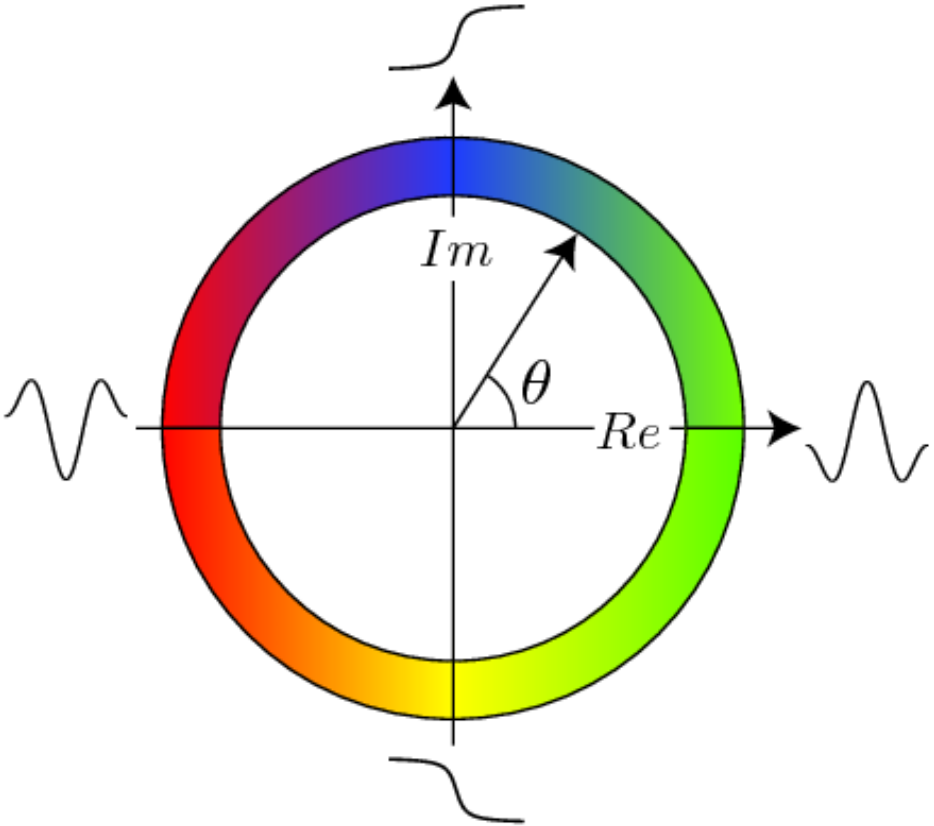


Real
(even)



Imag
(odd)

Local phase



Phase based optical flow

$$\nabla\varphi^T \boldsymbol{v} - \Delta\varphi = 0,$$

$$\boldsymbol{v}(\boldsymbol{x}) = \boldsymbol{B}(\boldsymbol{x})\boldsymbol{p}$$

$$\boldsymbol{v}(\boldsymbol{x}) = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + \begin{bmatrix} p_4 & p_5 & p_6 \\ p_7 & p_8 & p_9 \\ p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} 1 & 0 & 0 & x & y & z & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & x & y & z & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & x & y & z \end{bmatrix}}_{\boldsymbol{B}} \boldsymbol{p}.$$

Eklund et al., Phase Based Volume Registration Using CUDA, ICASSP, 2010

Phase based optical flow

$$\epsilon^2 = \sum_k \sum_i \left(\nabla \varphi_k(\boldsymbol{x}_i)^T \boldsymbol{v}(\boldsymbol{x}_i) - \Delta \varphi_k(\boldsymbol{x}_i) \right)^2 .$$

$$\boldsymbol{v}(\boldsymbol{x}) = \boldsymbol{B}(\boldsymbol{x})\boldsymbol{p}$$

$$\epsilon^2 = \sum_k \sum_i c_{ki} (\nabla \varphi_k(\boldsymbol{x}_i)^T \boldsymbol{B}(\boldsymbol{x}_i)\boldsymbol{p} - \Delta \varphi_k(\boldsymbol{x}_i))^2 .$$

$$\frac{\partial \epsilon^2}{\partial \boldsymbol{p}} = 2 \sum_k \sum_i c_{ki} \boldsymbol{B}_i^T \nabla \varphi_{ki} (\nabla \varphi_{ki}^T \boldsymbol{B}_i \boldsymbol{p} - \Delta \varphi_{ki}) .$$

$$\underbrace{\sum_k \sum_i c_{ki} \boldsymbol{B}_i^T \nabla \varphi_{ki} \nabla \varphi_{ki}^T \boldsymbol{B}_i}_{\boldsymbol{A}} \boldsymbol{p} = \underbrace{\sum_k \sum_i c_{ki} \boldsymbol{B}_i^T \nabla \varphi_{ki} \Delta \varphi_{ki}}_{\boldsymbol{h}} .$$

for each iteration

Convolve the current volume with 3 quadrature filters (x,y,z)
(non-separable 3D convolution, 7 x 7 x 7 filters)

Calculate phase differences, phase gradients and certainties

Setup the equation system, by summing over all voxels

Calculate a movement field from the parameter vector

Rotate and translate the current volume,
use texture memory for fast linear interpolation

end

Processing times

Preprocessing of 64 x 64 x 22 x 200 dataset

SPM8	240 s
Matlab / C	36 s
OpenMP	12 s
CUDA	1 s

Eklund et al., fMRI analysis on the GPU – Possibilities and challenges, Computer Methods and Programs in Biomedicine, 2012

Parametric fMRI analysis

The most common approach to fMRI analysis is to apply the General Linear Model (GLM) to each voxel timeseries

$Y = XB + e$

Test if there is a significant difference between activity and rest

Apply t-test or F-test, use parametric null distribution

Multiple testing

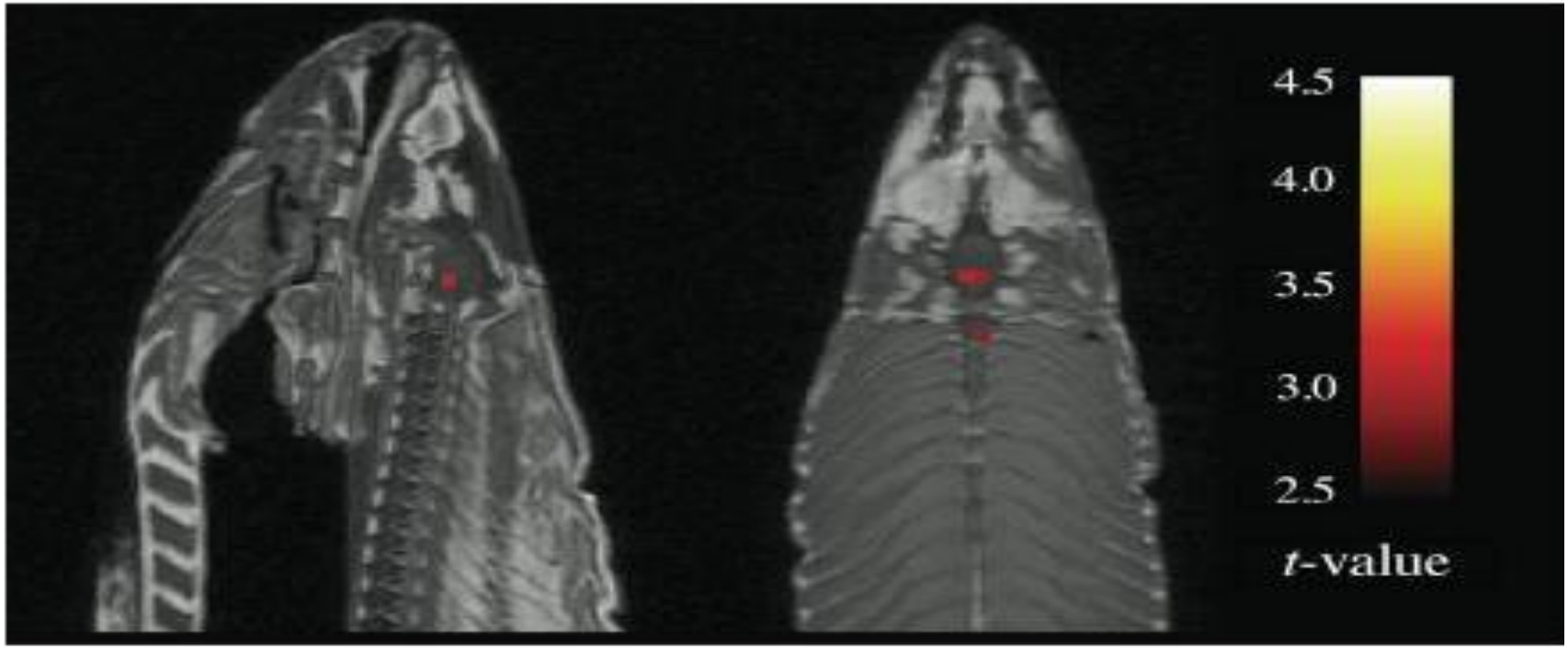
One major problem in fMRI is the large number of tests

Test 20 000 brain voxels for activity

p = 0.05, 1000 false positives (on average)

Bonferroni correction, random field theory,
false discovery rate

Significant brain activity found in dead salmon (Bennett et al. 2010)



Non-parametric fMRI analysis

Parametric fMRI analysis relies on several assumptions (normality, independence etc.)

The assumptions become more critical due to the multiple testing

More advanced detection statistics do not have a known parametric null distribution

Eklund et al., Fast random permutation tests enable objective evaluation of methods for single subject fMRI analysis, International Journal of Biomedical Imaging, 2011

Non-parametric fMRI analysis

Non-parametric methods are based on a lower number of assumptions, but are much more computationally demanding

Random permutation test

Empirically estimate the null distribution,
by analyzing ~ 10 000 similar datasets

Fisher ~ 1930, now we have the computational power…

Non-parametric fMRI analysis

Generate datasets by permuting timeseries (single subject fMRI) or by permuting activity maps (multi subject fMRI)

Can solve the problem of multiple testing by estimating the ***maximum*** null distribution

Only save the maximum test value in each permutation

Permuting timeseries in CUDA

A random permutation of a timeseries can result in irregular memory access patterns

Want to keep the spatial structure, apply the same permutation to all timeseries (permute the volumes)

If the data is stored as (x,y,z,t), permute chunks of voxels

(Do not store data as (t,x,y,z))

Permuting timeseries in CUDA

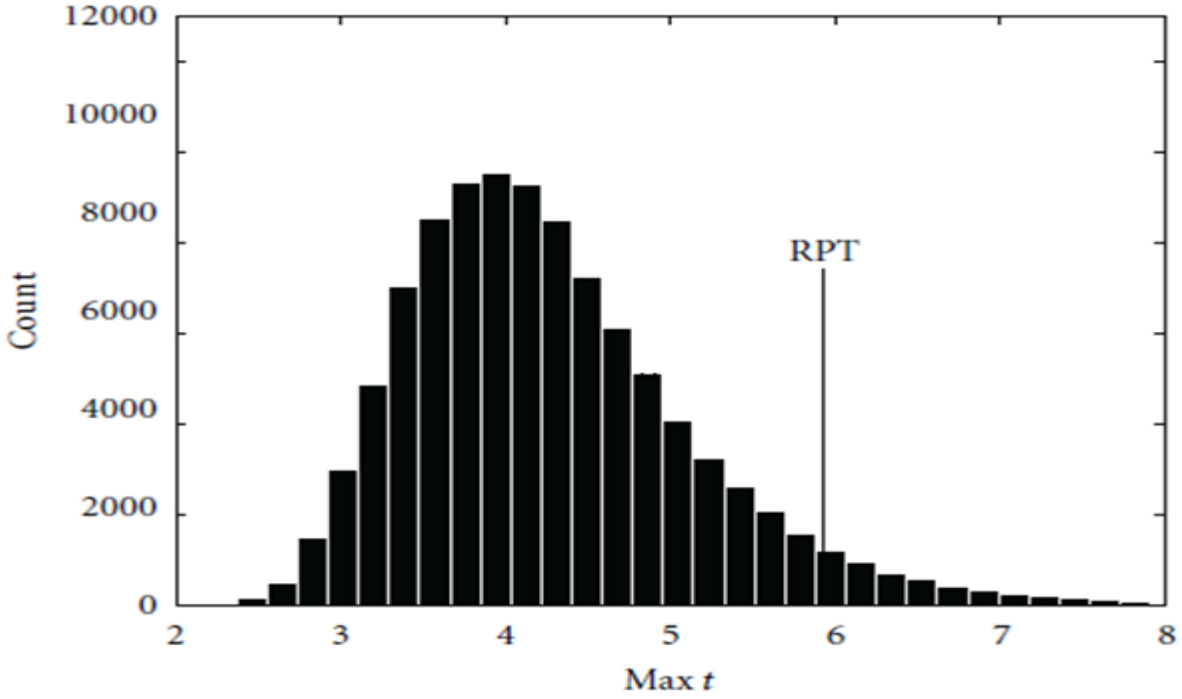
32 threads along x, 8 threads along y = 256 threads per block

Store permutation vector in constant memory

Each GPU thread loops over time for one voxel

```
for (int t = 0; t < DATA_T; t++)
        index = x + y * DATA_W +  z * DATA_W * DATA_H +
        c_Permutation_Vector[t] * DATA_W * DATA_H * DATA_D;
```

Non-parametric fMRI analysis



Estimated null distribution of ***maximum*** t

Multi-GPU
Random permutation test

Data

Preprocessing
3333 Permutations

Preprocessing
3333 Permutations

Preprocessing
3333 Permutations

Processing times

10 000 permutations of
64 x 64 x 22 x 200 dataset

Matlab / C	40 h
OpenMP	6 h
CUDA, 1 x GTX 480	6 min
CUDA, 3 x GTX 480	2 min

Verifying the random permutation test

How do we know that the random permutation test works correctly?

Analyzed 1484 rest datasets (85 GB)
10 000 permutations per dataset (850 TB)

Used a familywise significance threshold of 5%
Found significant activity in ~5% of the rest datasets

SPM8	~100 years
Multi-GPU	~10 days

Eklund et al., Does Parametric fMRI Analysis with SPM Yield Valid Results?
- An Empirical Study of 1484 Rest Datasets, NeuroImage, 2012

Non-parametric fMRI analysis

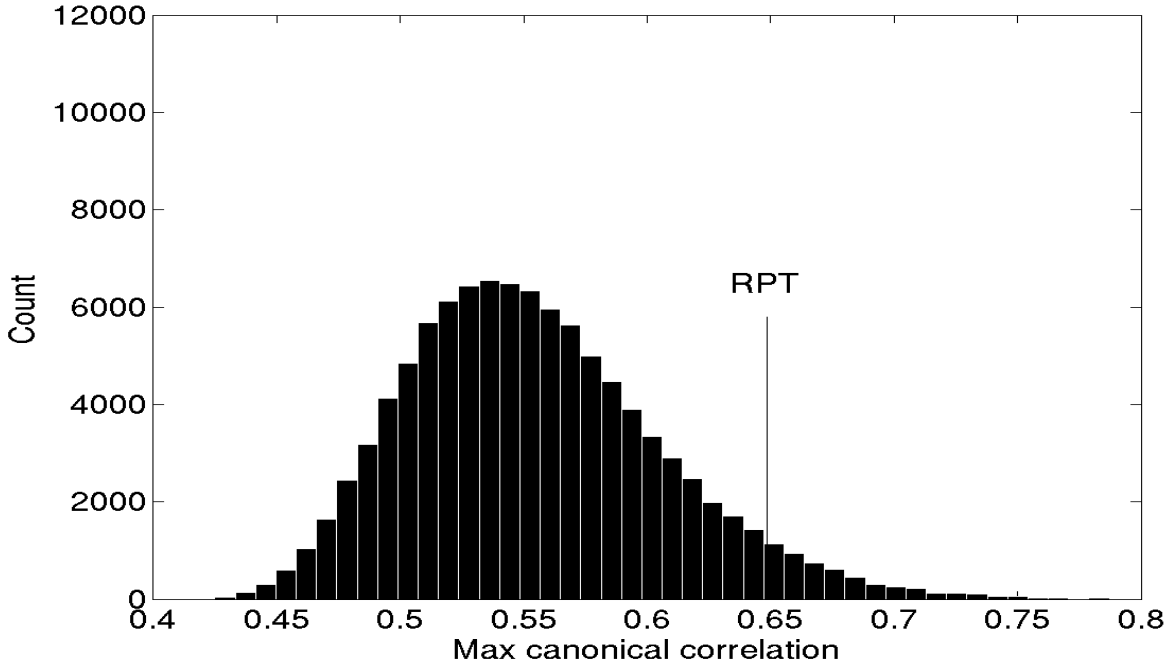
We like to do fMRI analysis by using
restricted canonical correlation analysis (RCCA)

RCCA does not have any parametric null distribution

Use the random permutation test to
estimate the null distribution

Use the estimated null distribution to
calculate significance thresholds and p-values

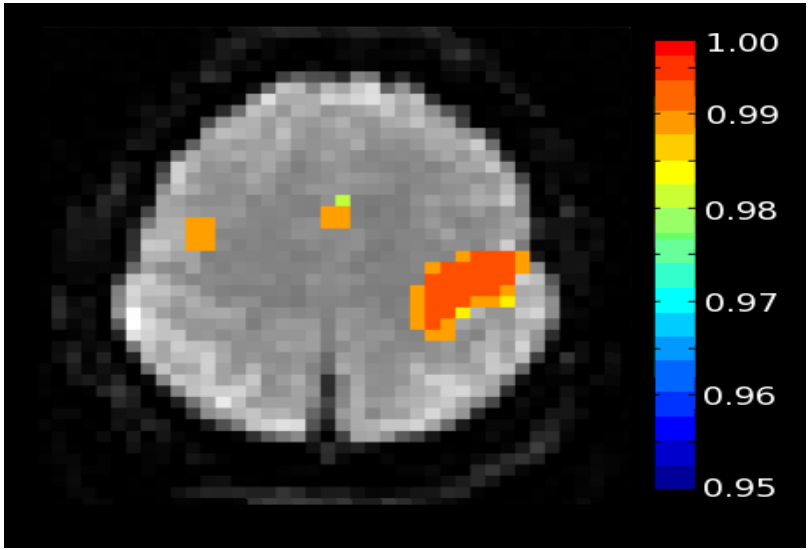
Non-parametric fMRI analysis



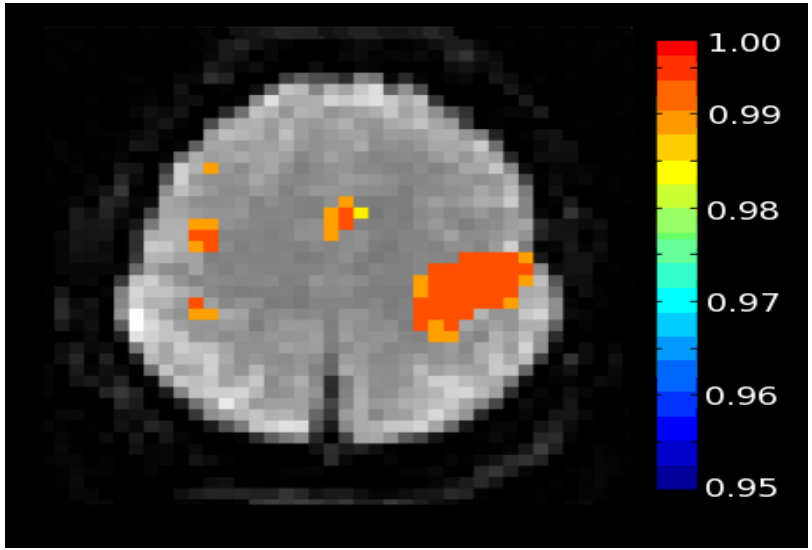
Estimated null distribution of maximum canonical correlation

GLM vs RCCA
Corrected p-values (1 – p)
Thresholded at corrected p = 0.05

GLM



RCCA



Real-Time fMRI

In real-time fMRI, the analysis is performed
while the subject is in the MR scanner

Look at your own brain activity, learn to suppress pain

Interactive brain mapping

Brain computer interfaces

Using the GPU, more advanced fMRI analysis can be done in real-time

Eklund et al., Using real-time fMRI to control a dynamical system by brain activity classification, MICCAI, 2009

Real-Time fMRI

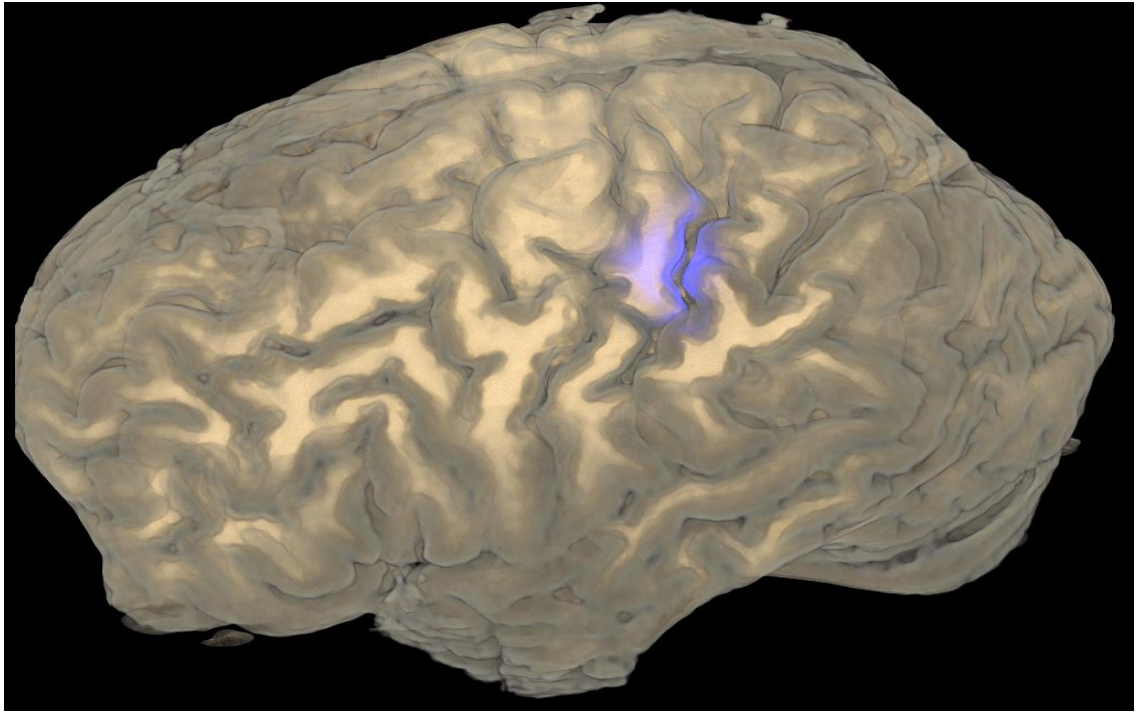
More advanced visualization in real-time

Treat the low resolution fMRI signal as a light source in the high resolution anatomical volume

Local ambient occlusion for shadow effects

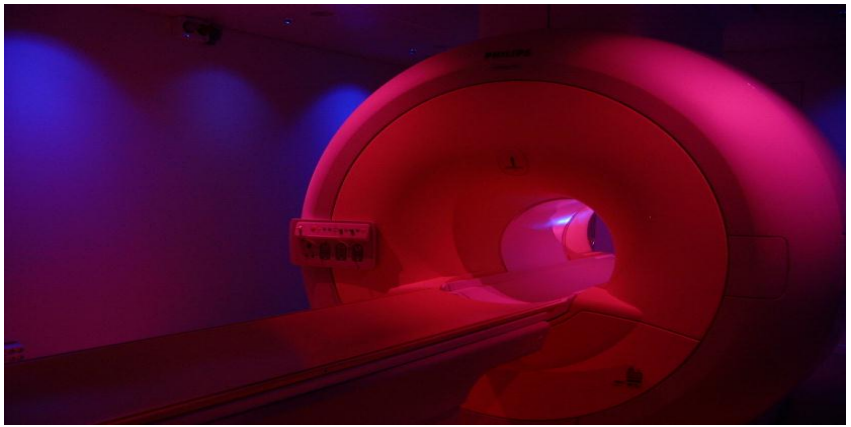
Nguyen et al., Concurrent volume visualization of real-time fMRI, IEEE Volume Graphics, 2010

Real-Time fMRI



Nguyen et al., Concurrent volume visualization of real-time fMRI, IEEE Volume Graphics, 2010

Linköping



Real-time fMRI

30 miles
Webcam interface

Norrköping

Advanced visualization
15 meter dome

Conclusions fMRI

The GPU speeds up the preprocessing,
required for future increase in temporal and spatial resolution

Non-parametric fMRI analysis becomes practibable

More advanced analysis in real-time

More advanced visualization

Analysis of large datasets (functional connectomes project 85 GB)

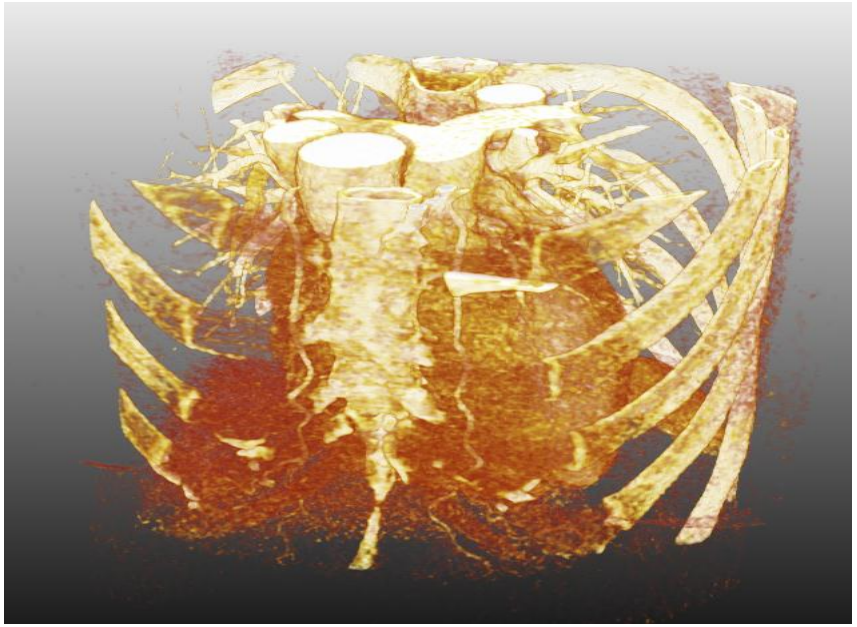
Eklund et al., fMRI analysis on the GPU – Possibilities and challenges,
Computer Methods and Programs in Biomedicine, 2012

True 4D Image Denoising on the GPU

Image denoising is common in medical imaging, to improve the image quality

For CT data, lower the radiation, keep the same image quality

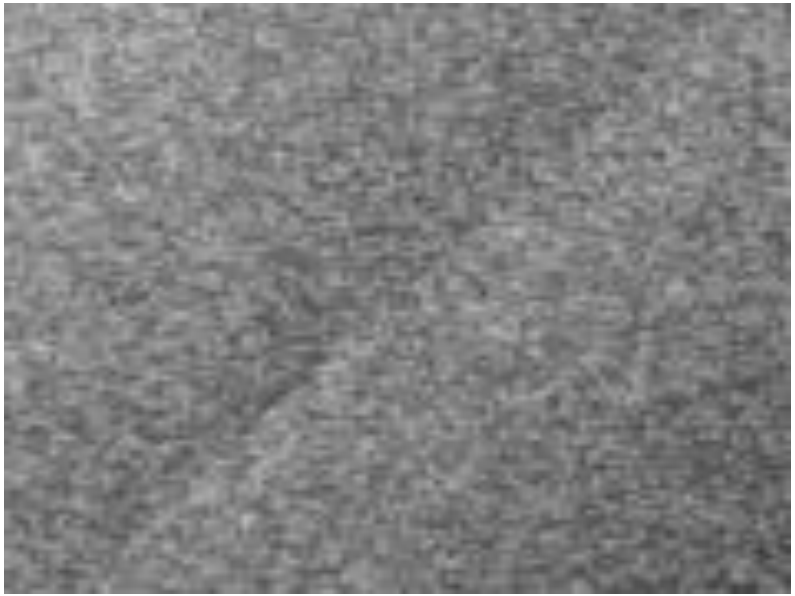
4D CT heart dataset of the resolution 512 x 512 x 448 x 20 (9 GB as floats)



Eklund et al., True 4D Image Denoising on the GPU, International Journal of Biomedical Imaging, 2011

Why 4D Image Denoising?

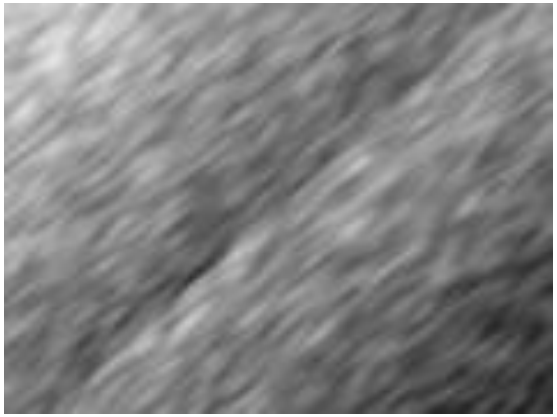
Original



Degraded

Eklund et al., True 4D Image Denoising on the GPU,
International Journal of Biomedical Imaging, 2011

Why 4D Image Denoising?



2D Denoising

3D Denoising

4D Denoising

Eklund et al., True 4D Image Denoising on the GPU,
International Journal of Biomedical Imaging, 2011

Adaptive filtering / Steerable filters

Adaptive filtering for image denoising, Knutsson et al.

- 2D 1981

- 3D 1992

- 4D 2011

Adaptive filtering / Steerable filters

Estimate the local structure tensor in each pixel / voxel / time voxel, use quadrature filters or monomial filters

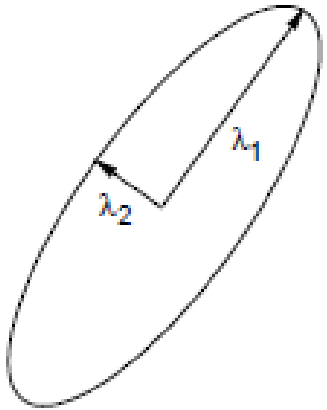
The tensor contains information about the local structure

In 2D, orientation of lines and edges

Knutsson, Representing Local Structure Using Tensors,
Scandinavian Conference on Image Analysis, 1989

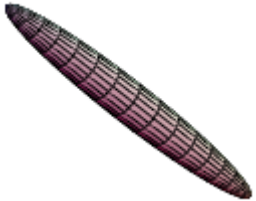
The local structure tensor in 2D

$$\mathbf{T} = \lambda_1 \hat{\mathbf{e}}_1 \hat{\mathbf{e}}_1^T + \lambda_2 \hat{\mathbf{e}}_2 \hat{\mathbf{e}}_2^T$$

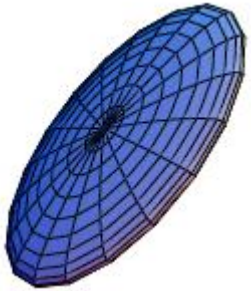


CF Westin, A Tensor Framework for Multidimensional Signal Processing, 1994

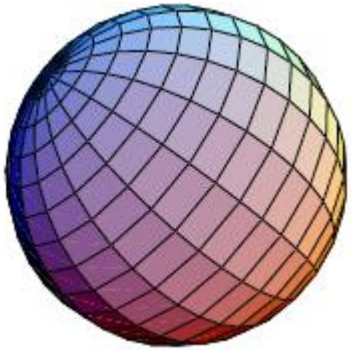
The local structure tensor in 3D



Plane



Line



Isotropic

CF Westin, A Tensor Framework for Multidimensional Signal Processing, 1994

The local structure tensor in 4D

Requires 12 complex valued quadrature filters in 4D
(24 real valued filters, non-separable)

Monomial filters, sufficient to use 14 real valued filters in 4D
(recently developed by Knutsson et al.)

Knutsson et al., Representing Local Structure Using Tensors II,
Scandinavian Conference on Image Analysis, 2011

Monomial filters

First order monomial filters in 4D (odd),

x, y, z, t

Second order monomial filters in 4D (even),

xx, xy, xz, xt, yy, yz, yt, zz, zt, tt

Knutsson et al., Representing Local Structure Using Tensors II,
Scandinavian Conference on Image Analysis, 2011

Adaptive filtering / Steerable filters

Apply a number of denoising filters, one isotropic lowpass filter and a number of anisotropic highpass filters in different directions

Use the local structure tensor to calculate weights for the filter responses of the highpass filters

Lowpass filter the data, put back highpass information along well defined orientations

Effect: Smooth along edges and lines, but not perpendicular to them

Adaptive filtering in 4D

Requires 14 filters to estimate the local structure tensor,
spatial support 7 x 7 x 7 x 7 time voxels

Requires 11 filters to do the actual denoising,
spatial support 11 x 11 x 11 x 11 time voxels

The local structure tensor in 4D

$$\mathbf{T} = \begin{pmatrix} t_1 & t_2 & t_3 & t_4 \\ t_2 & t_5 & t_6 & t_7 \\ t_3 & t_6 & t_8 & t_9 \\ t_4 & t_7 & t_9 & t_{10} \end{pmatrix} = \begin{pmatrix} xx & xy & xz & xt \\ xy & yy & yz & yt \\ xz & yz & zz & zt \\ xt & yt & zt & tt \end{pmatrix}$$

t1 = mfr1 * mfr1 + mfr5 * mfr5 + mfr6 * mfr6 + mfr7 * mfr7 + mfr8 * mfr8
t2 = mfr1 * mfr2 + mfr5 * mfr6 + mfr6 * mfr9 + mfr7 * mfr10 + mfr8 * mfr11
t3 = mfr1 * mfr3 + mfr5 * mfr7 + mfr6 * mfr10 + mfr7 * mfr12 + mfr8 * mfr13
t4 = mfr4 * mfr1 + mfr5 * mfr8 + mfr6 * mfr11 + mfr7 * mfr13 + mfr8 * mfr14
t5 = mfr2 * mfr2 + mfr6 * mfr6 + mfr9 * mfr9 + mfr10 * mfr10 + mfr11 * mfr11
t6 = mfr2 * mfr3 + mfr6 * mfr7 + mfr9 * mfr10 + mfr10 * mfr12 + mfr11 * mfr13
t7 = mfr2 * mfr4 + mfr6 * mfr8 + mfr9 * mfr11 + mfr10 * mfr13 + mfr11 * mfr14
t8 = mfr3 * mfr3 + mfr7 * mfr7 + mfr10 * mfr10 + mfr12 * mfr12 + mfr13 * mfr13
t9 = mfr3 * mfr4 + mfr7 * mfr8 + mfr10 * mfr11 + mfr12 * mfr13 + mfr13 * mfr14
t10 = mfr4 * mfr4 + mfr8 * mfr8 + mfr11 * mfr11 + mfr13 * mfr13 + mfr14 * mfr14

Memory demanding!

Non-separable 4D convolution with CUDA

Two approaches can be used

Spatial filtering (convolution)

Fast Fourier Transform (FFT) based filtering,
multiplication between the filter and
the signal in the frequency domain

Spatial filtering

The filters are Cartesian non-separable

512 x 512 x 448 x 20 x 11 x 11 x 11 x 11 x 11 =

375 000 000 000 000 multiply add

11 filter responses for 512 x 512 x 448 x 20 = 103 GB

Size of global memory for Nvidia GTX 580 = 3 GB (standard 1.5 GB)

Also need to store the 10 tensor components…

Spatial filtering

For 2D convolution, use the shared memory

Load pixels into shared memory, apply filters, write results back to global memory

(Examples in CUDA SDK)

Spatial filtering

48 KB of shared memory, 11 x 11 x 11 x 9 float values

Do not get any valid filter responses
for filters of size 11 x 11 x 11 x 11 !

11 filters of size 11 x 11 x 11 x 11 = 644 KB,
do not fit in constant memory (64 KB) !

No support for 4D textures

Spatial filtering

Non-separable 4D convolution requires 8 loops
(loop over x,y,z,t for the data and for the filter)

Our approach, use an optimized non-separable 2D convolver

Load 64 x 64 float values (x,y) into shared memory (16 KB),
three thread blocks = 48 KB

Spatial filtering

Load 11 x 11 filter values into constant memory, for 11 filters
(5.3 KB, smaller than constant memory cache of 8 KB)

Apply the 11 filters at the same time, unroll loops with Matlab script

Loop over z and t on the CPU (for data and filter),
increment the filter responses inside the kernel

Four loops on CPU, four loops on GPU

FFT based filtering

No direct support for 4D FFT's in CUDA

The FFT is cartesian separable, apply batches of 1D FFT's

Batches of 1D FFT’s are applied along the first direction of the data

Problem, need to flip the data order between each FFT,
(x,y,z,t) -> (y,z,t,x), (y,z,t,x) -> (z,t,x,y), (z,t,x,y) -> (t,x,y,z)

FFT based filtering

Slower to change the order of data than to perform 1D FFT…

CUDA 4.0 supports batches of 2D FFT's,
sufficient with one flip instead of three

Apply 2D FFT along x,y, flip data, apply 2D FFT along z,t

Multiplication between filter and data

Inverse 4D FFT

Processing times

Spatial filtering

CPU 2 days One GPU 27 minutes

FFT based filtering

CPU 1 hour One GPU 9 minutes

Multi-GPU

Data

Slices 1-20

Slices 21-40

Slices 41-60

Spatial vs FFT based filtering

FFT based filtering is, in general, faster
but is more memory demanding

CPU FFT more efficient due to larger memory

GPU FFT not as optimized as CPU FFT
(especially for sizes that are not a power of 2)

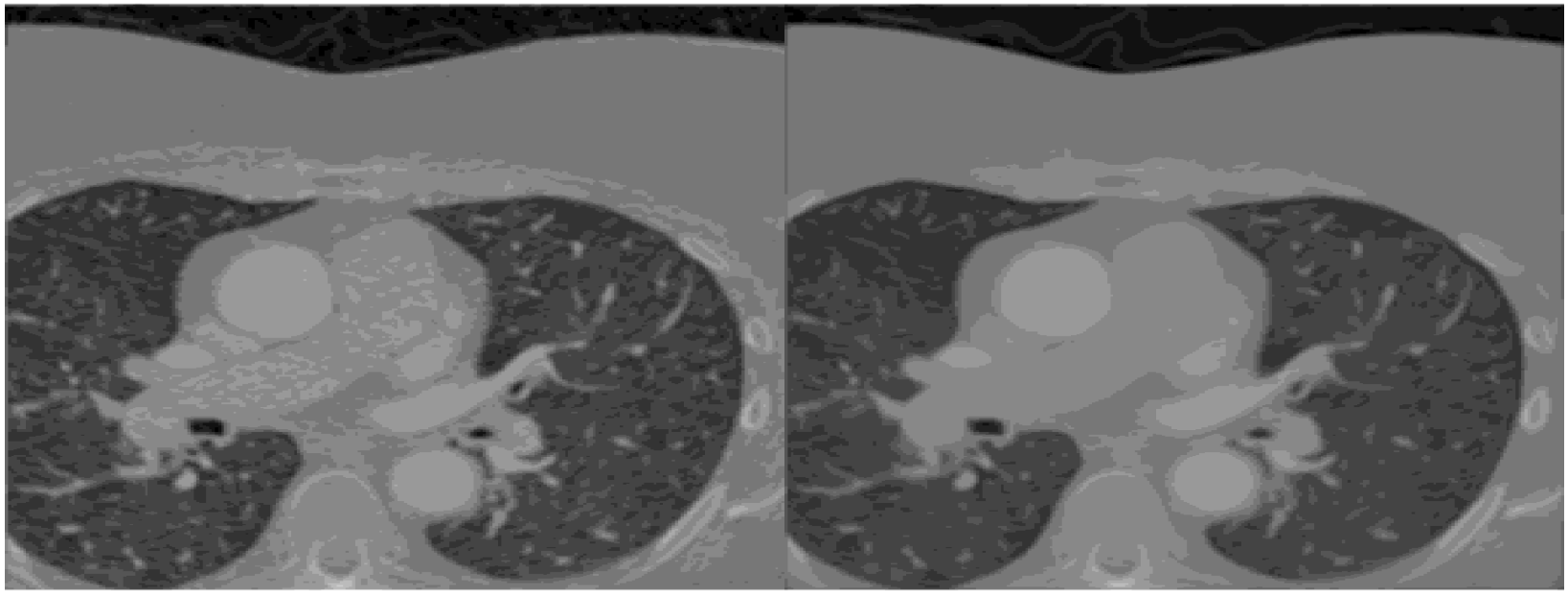
Spatial vs FFT based filtering

Spatial filtering can handle larger datasets
(512 x 512 x 512 x 100)

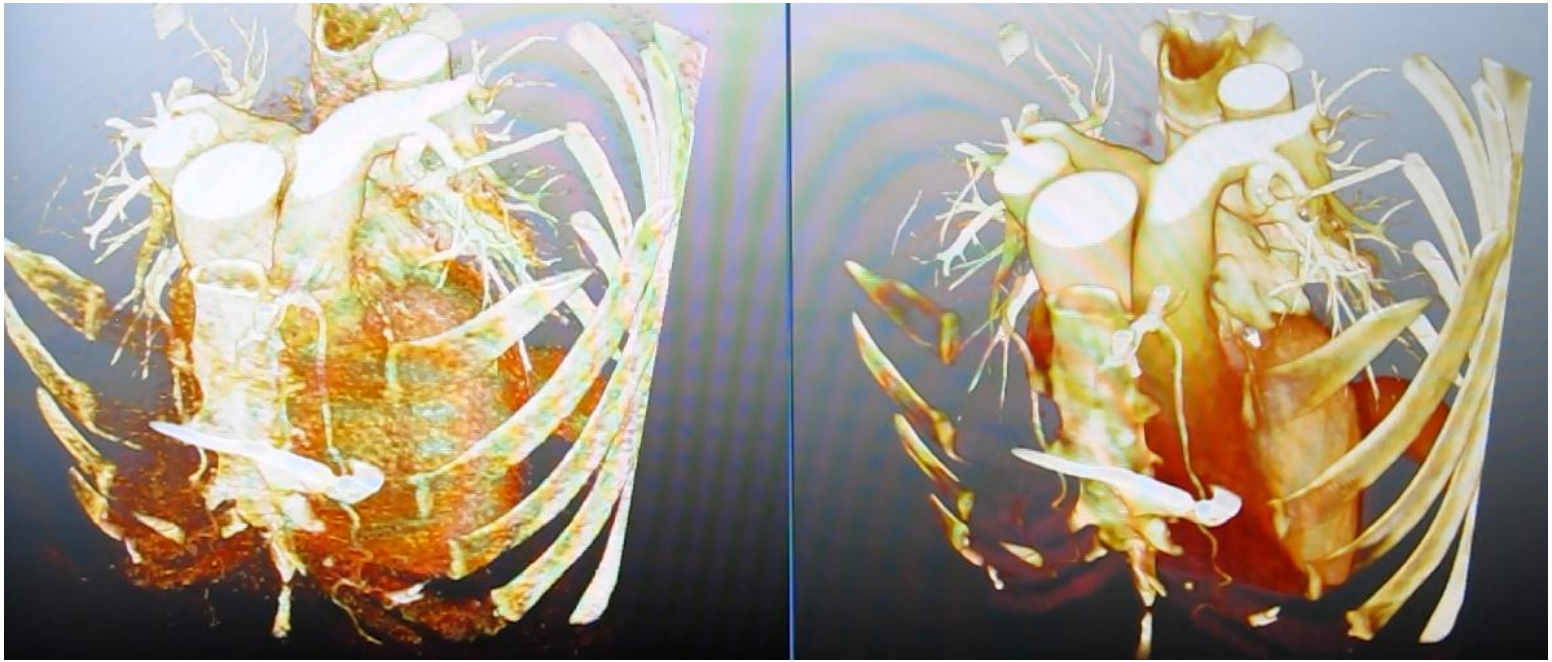
Easier to denoise a single slice / volume with spatial filtering

Filter networks, apply several small filters, instead of one large

Denoising results



Denoising results



Wishlist to Nvidia

4D textures

Direct support for 4D FFT's, fftshift function

Global memory, 1.5 GB => 32 – 128 GB

Shared memory, 48 KB => 1 MB per MP

Registers, 32 768 => 524 288 per MP

Constant memory, 64 KB => 1 MB

An image processing library with
support for 3D and 4D convolution (floats, not integers!)

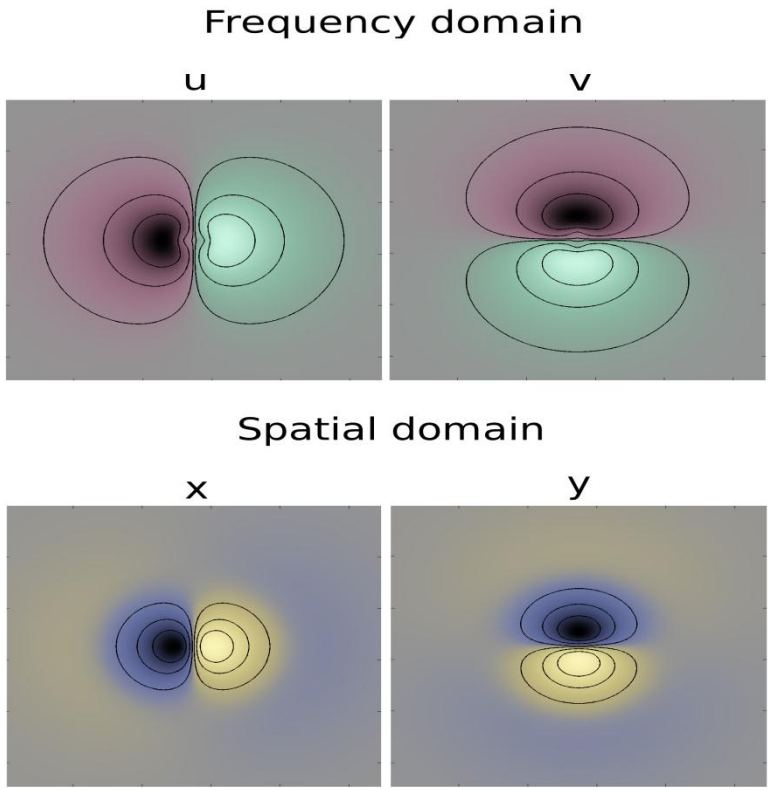
Questions?

anders.eklund@liu.se

CMIV

Linköpings universitet

First order monomial filters in 2D



Second order monomial filters in 2D