# Volume Data Representation and Processing Algorithms

A collection of ideas

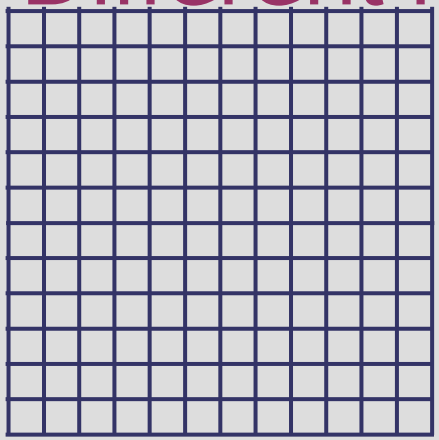
Miloš Šrámek

# Overview

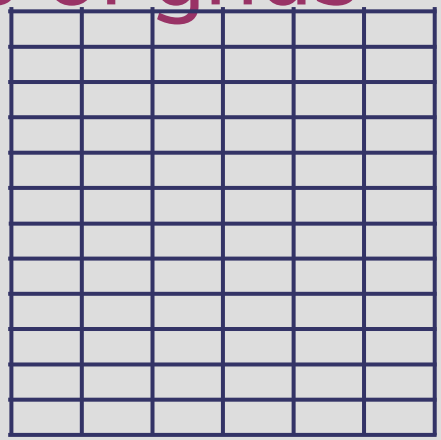
- F3d format and tools for volume data storage and processing
- The problem
- Algorithms for processing of volume data
- Different memory representations
- In-memory and  stream processing
- A paper proposal

# The f3d Format

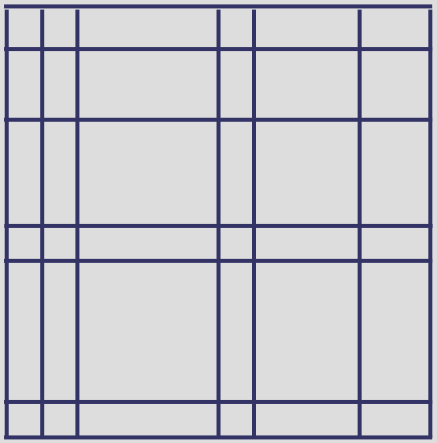
- Support for different kinds of voxels
  - Single- and multiband volumes
  - 8, 16, 32 bit integers, signed and unsigned, floats
- Different kinds of grids



**Cartesian**



**regular**



**rectilinear**

# The f3d Format

- Self descriptive
  - Stores all necessary data attributes
- Data compression
- Cross platform compatibility
  - Little/Big endian
  - Size of voxel representation
- Availability
  - Covered by a non-restrictive license
- Ease of usage

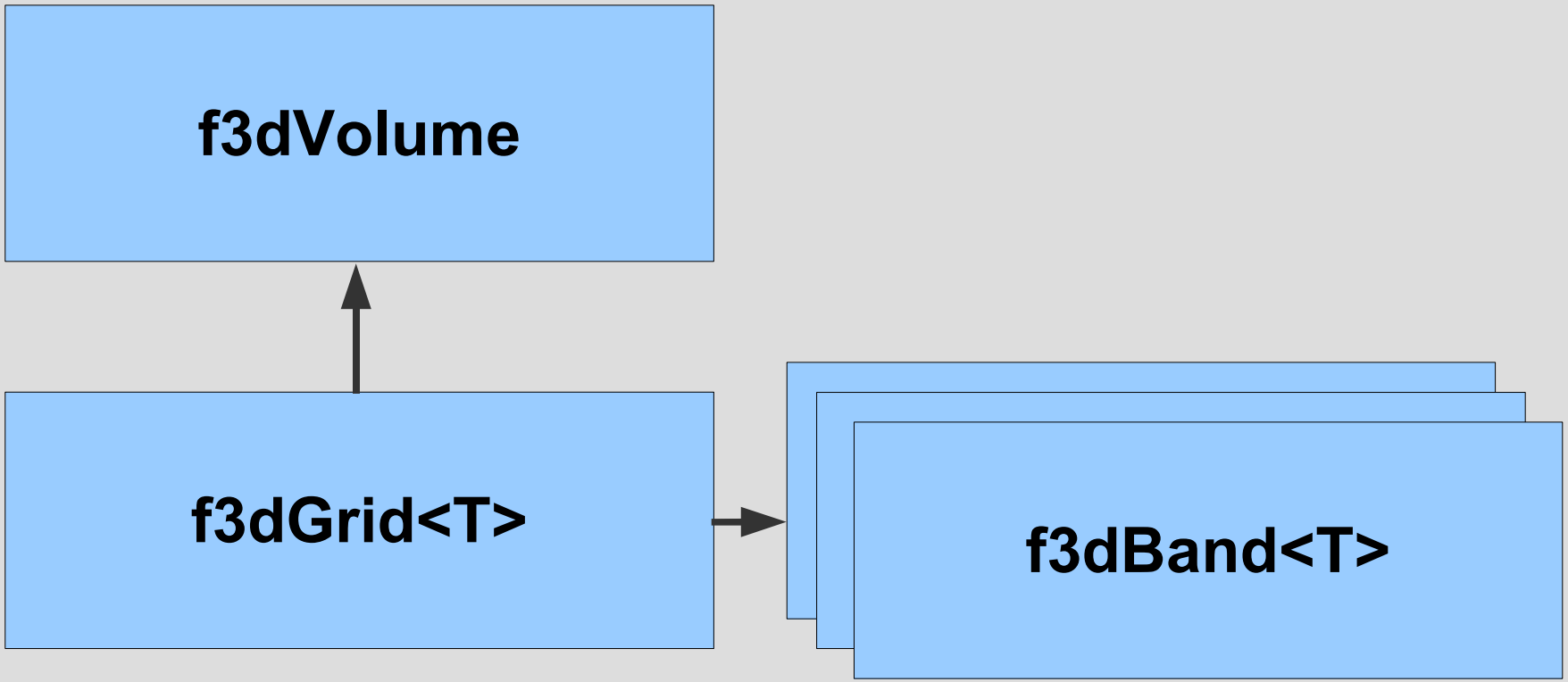
# Basic C Library

- Reading:
  - f3dReadHeader
  - f3dReadSlice
  - f3dReadGrid, f3dReadCubGrid
- Writing: analogous
- Other:
  - f3dSetHdrComment, f3dDelHdrComment
  - f3dHostType
  - f3dVoxelSize

# Extended C++ Class Library

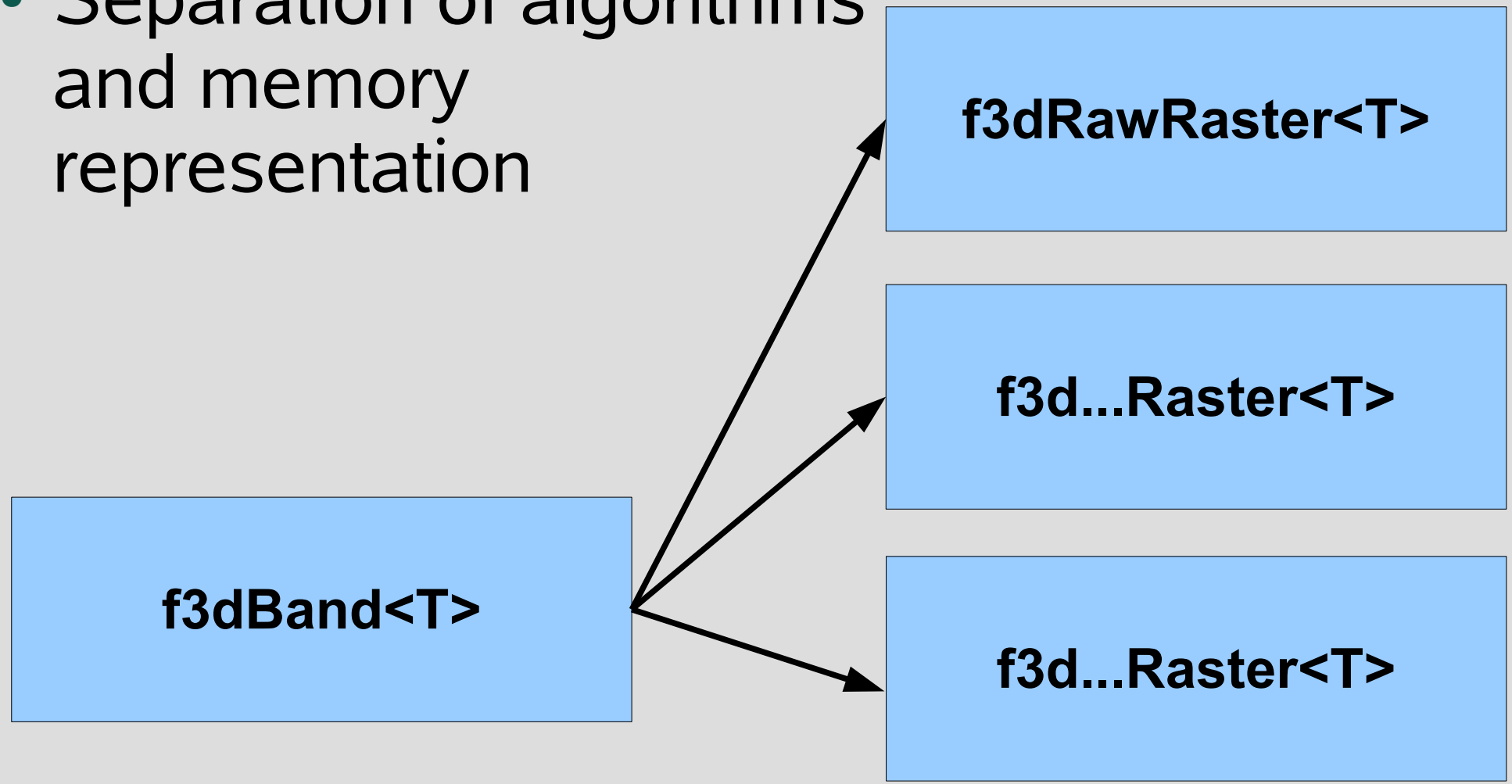
- Basic access to data voxels
- File storage and retrieval
- 3D data processing
  - filters, segmentation, rendering, interpolation, transformations, gradient, ...
- Typeless data processing

# Class Hierarchy



# Memory Independence

- Separation of algorithms and memory representation



# f3d Tools

- f3dclass functionality implemented in the form of UNIX filters:
  - `f3dprog [sw] [input.f3d] > output.f3d`
  - Filter concatenation is possible:
  - `tool1 [sw] in.f3d |...| toolN [sw] > out.f3d`
- f3dview – simple slice viewer
- f3dvr – simple hardware-based renderer

# f3d Tools: Point Operators

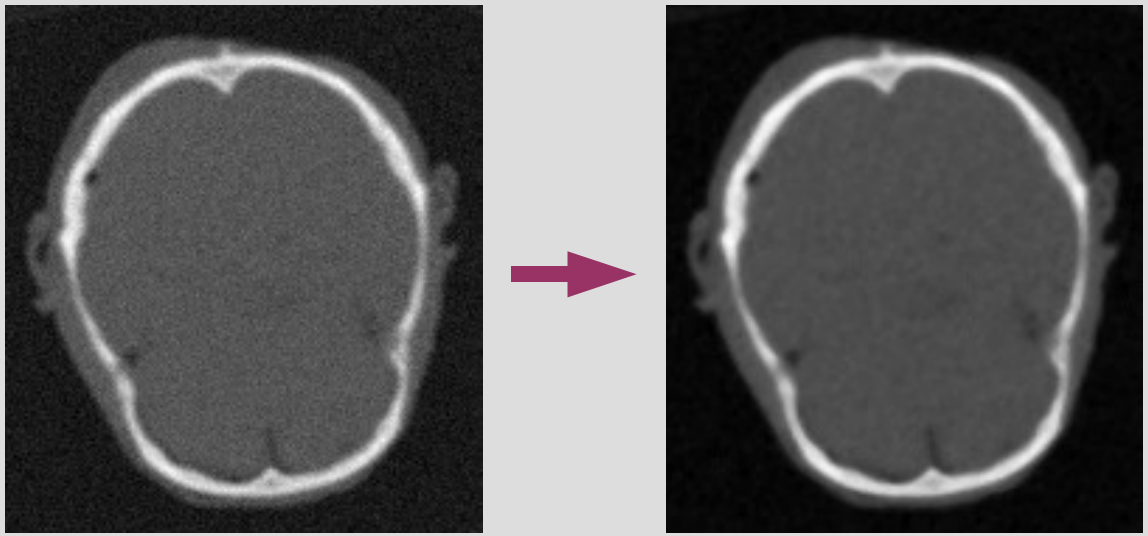
- f3d2f3d (voxel type conversion), f3dinvert, f3dthresh, f3dbit , f3darith, f3dmask
- Example: Delete bone from CT data:

```
f3dthresh -lo 90 in.f3d | f3dmask -i in.f3d > out.f3d
```

# f3d Tools: Local Operators (Filters)

- Data processing in a local voxel neighborhood
- Order filters: f3dmax, f3dmin, f3dmedian
- Example: Noise removal by the median filter

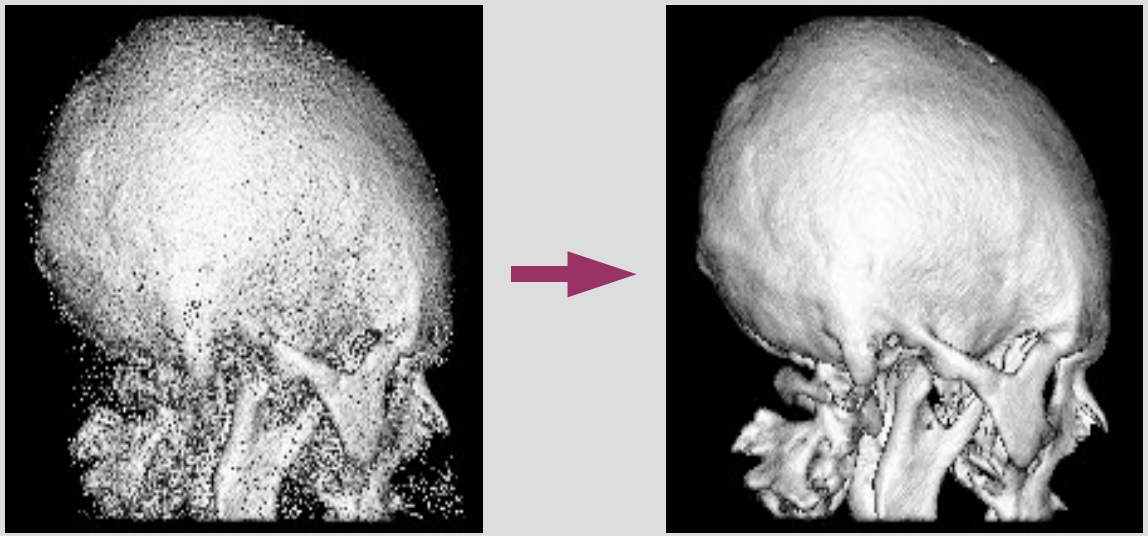
```
f3dmedian -k 3 in.f3d > out.f3d
```



# f3d Tools: Local Operators (Filters)

- Data processing in a local voxel neighborhood
- Order filters: f3dmax, f3dmin, f3dmedian
- Example: Noise removal by the median filter

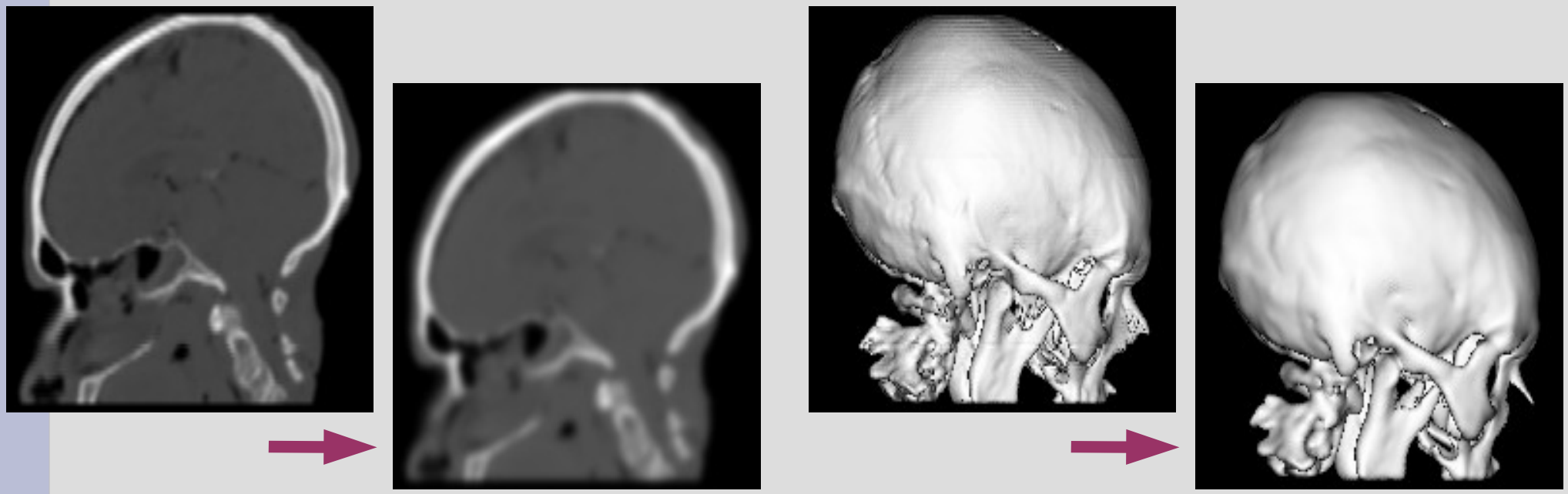
```
f3dmedian -k 3 in.f3d | f3drender -lo 90 -r 4> out.png
```



# f3d Tools: Local Operators Gaussian filtering

- Convolution by a Gaussian with different widths
- Example (data smoothing):

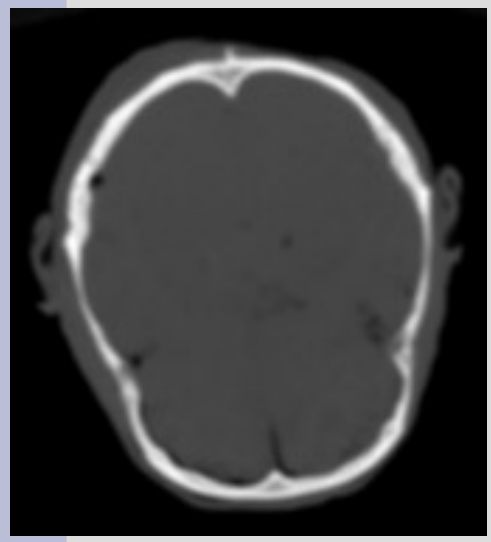
```
f3dgauss -w 2 in.f3d > out.f3d
```

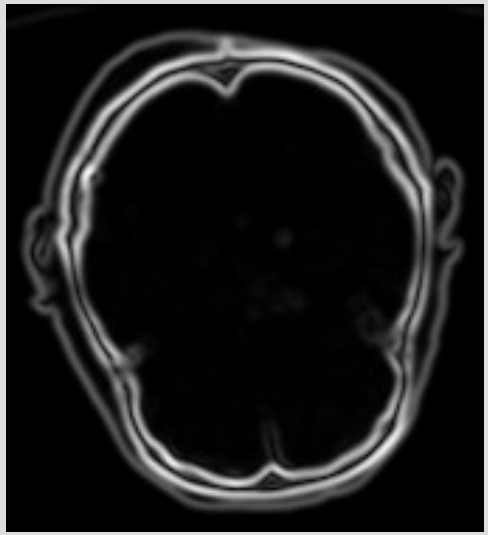


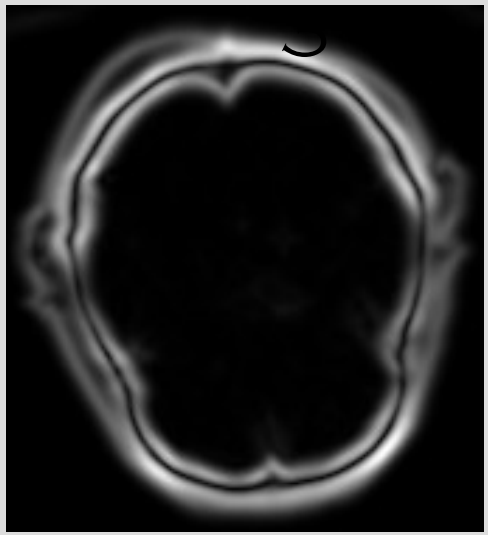
# f3d Tools: Local Operators
# Gradient Magnitude Estimation

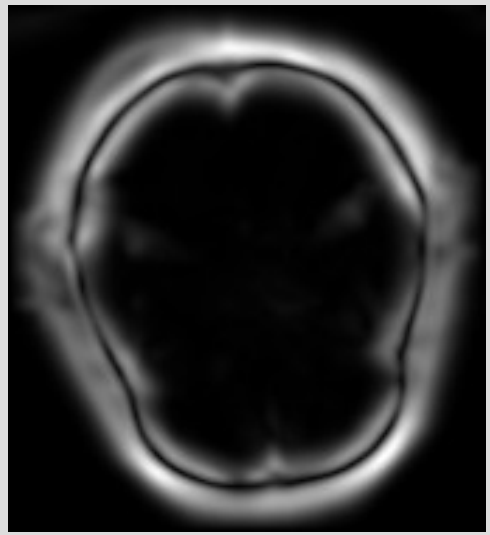
- Magnitude of the Gabor filter applied to all 3 directions, edge detection
- Example :

```
f3dgradmag -w 3 in.f3d > out.f3d
```





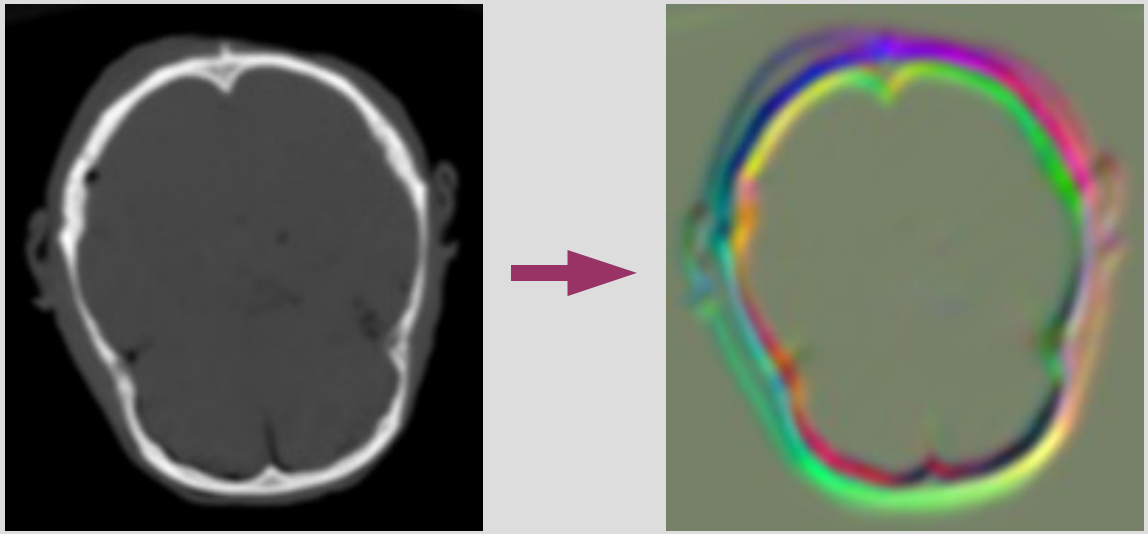



# f3d Tools: Local Operators
# Gradient Estimation

- Convolution by a Gabor filter in all directions
- Example :

```
f3dgrad -w 3 in.f3d > out.f3d
```



# The Problem

- Volume data is often too big to fit into memory and its processing is too slow
- Possible solutions:
  - Alternative memory layouts
  - Stream processing
- Task:
  - Design different raster types  with common API for different tasks:
    - Interactive (GUI) vs. batch processing
    - Sequential vs. direct access processing
    - High speed vs. low memory requirements

# Algorithm Classes and Iterators

- Algorithms
  - Single voxel operations
  - Local operations
    - Unbuffered
    - Buffered
  - Global operations
- Iterators (data representation hiding)
  - Single voxel access sequential iterator
  - Local voxel access sequential iterator
  - Direct access iterator

# Single Voxel Operations

- Sequential access to just the current voxel
  - Write only (creating volumes)
  - Read only (maximum value search)
  - Read/Write – point operations

```
typename RASTER::VoxelIterator it;

for(it = raster->dataStartVox(F3D_RW);
                        it != raster->dataEndVox(); ++it) {
        *it = OP(*it);
}
```

# Local Operators (non-separable, buffered)

- Data access in local neighborhood required
- Data filtering – input must remain unchanged, therefore buffering is required:

```
typename RASTER::BufferIterator iit;
typename RASTER::VoxelIterator oit;

for(iit = raster->dataStartBuf(F3D_READ, N), oit = raster->dataStartVox(F3D_WRITE);
                iit != raster->dataEndBuf(); ++iit, ++oit)
{
    float val(0);
    int i,j,k;
    for(i = -N/2; i <= N/ 2; i++)
        for(j = -N/2; j <= N/2; j++)
            for(k = -N/2; k <= N/2; k++)
                val += iit.get(i, j, k) * kernel[i + N/2][j + N/2][k + N/2];
    *oit = T(val);
}
```

# Local Operators (unbuffered)

- In some algorithms we have to use the new value stored earlier (e.g., distance transforms

```
typename RASTER::VoxelIterator it;

for(it = raster->dataStartVox(F3D_RW, N); it != raster->dataEndVox(); ++it)
{
        float val(0);
        int i,j,k;
        for(i = -N/2; i <= N/ 2; i++)
                for(j = -N/2; j <= N/ 2; j++)
                        for(k = -N/2; k <= N/2; k++)
                                val = ... it.get(i, j, k) ...
        *it = T(val);
}
```

# Local Operators (separable, buffered)

- Separable filters:

$$F(x, y, z) = X(x) * Y(y) * Z(z)$$

- Complexity 3n (non-separable: $n^3$)
- Special iterator is required
  - Combination of BufferIterator and VoxelIterator

# Global operations

- Sometimes can be implemented as unbuffered local operators
- Usually require direct access to data (Fourier transform, rendering etc)
- DAIterator should be written

# Different memory layouts

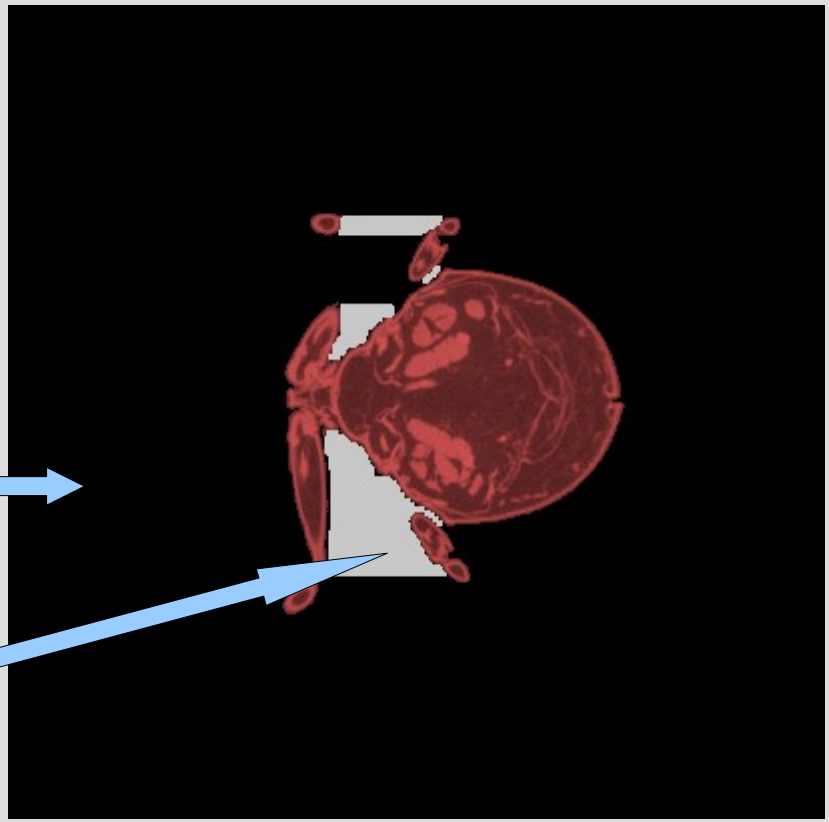
- Different *f3dXXRaster* implementations required
- Common interface provided by iterators
- Current status: Partial implementation of
  - f3dRawRaster – the trivial one
  - f3dPackRaster – some compression
  - f3dSwappedRaster – intermediate file storage
  - f3dRLRaster, f3dBlockRaster – modified RL compression, blocking
- Tasks: full implementation, verification, performance evaluation, new raster designs

# f3dRawRaster

- Data fully represented in a linear array
- Sequential access: ptr++
- Direct access is easy
- Problems with large data

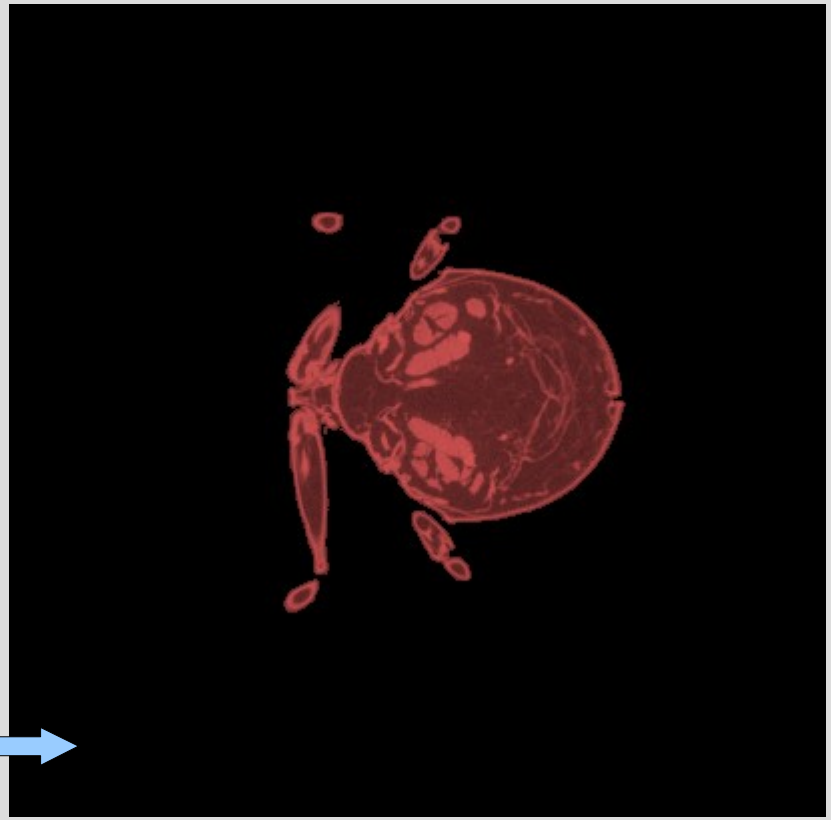
# f3dPackRaster

- Object-of-interest is usually in the middle:
  - Segmentation required
  - Per-row storage: skip N – represent – skip M
- Direct access is easy
- Useful for interactive processing
- Used in iseg



# f3dRLRaster

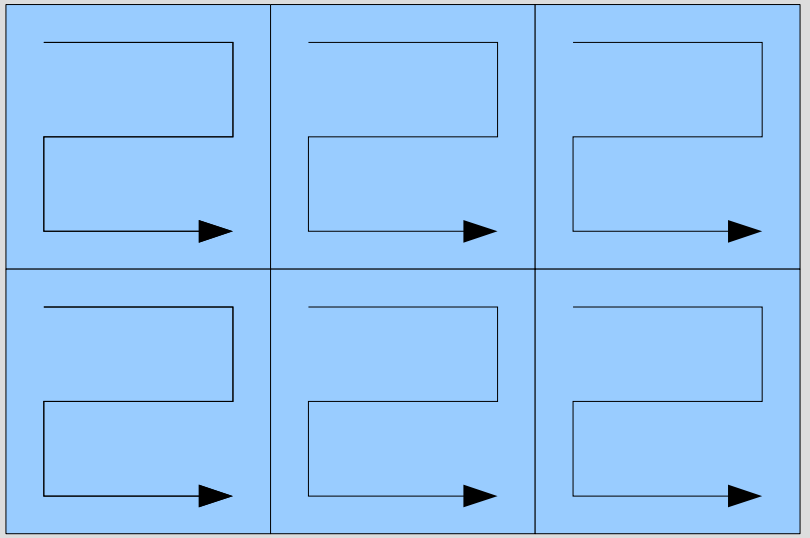
- Object-of-interest is usually in the middle:
  - Segmentation required
  - Per-row storage: skip – represent ....
- Direct access no more so easy
- Different possibilities, data dependent
- Used by Pavol in vxtRL



Unrepresented Background

# f3dBlockRaster

- Volume represented as an array of linearly organized blocks
- Suitable for random access (rendering)
- Not all blocks represented
- Block overlap
- 2(3)-level hierarchy, BSP, octree?



# f3dSwappedRaster

- To be used in operations, which require lot of data in memory, bur still not all
- Buffering numerous slices, the least important stored on a HDD
- Suitable for direct access

# F3dStreamRaster (1)

- Voxel and local operations require just few slices in memory
- Processing reorganized:

```
LoadVolume()
For all slices
{
        process slice ();
}
WriteVolume()
```

```
For all slices
{
        LoadSlice()
        process slice ();
        WriteSlice()
}
```

# F3dStreamRaster (2)

- Just single-run tools possible:

```
tool in.f3d > out.f3d
```

- They can be, however, concatenated:

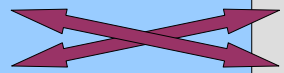
```
tool1 in.f3d | tool2 | ... toolN > out.f3d
```

- Advantages: small memory footprint, natural parallelization
- Disadvantages: requires code modifications, not everything can be streamed

# F3dStream (3)

- Minimal modification required:

```
f3dVolume *s = f3dLoadRawVolume(fin);
s->addComm(cmdline.c_str());
s->addGNoise(sigma, band);
s->save(stdout);
```



```
f3dVolume *s = f3dLoadStreamVolume(fin);
s->addComm(cmdline.c_str());
s->save(stdout);
s->addGNoise(sigma, band);
```

- Some cases: temporal storage of intermediate results in a file
- Individual approach required
- Not everything can be implemented with small footprint(3D FFT?)

# What to do next?

- Implement, evaluate and write a paper
  - Vis'06, deadline March 2006
- Related work
  - vtk/itk (everything is a filter), openvl (iterators), out-of-core techniques
- Implement missing representations
- Comparison: speed & memory requirements
  - Find suitable data sets
- Write the paper