3D Gauss Filtering using SSE

Progress report

vasko.anton@gmail.com

Content

- SSE
- Gauss3
- SSE optimization of gauss3
- Results
- Conclusion

Quick info about SSE

- Intel’s extension to the CPU’s instruction set
- Currently available also on AMD’s CPUs
- 8 registers 128 bit wide
- New instructions for parallel calculations (e.g. add, sub, mul, div, sqrt …)
- Working with cache (prefetching, non-temporal store/read)

Gauss3

- Gauss filter of length 3 (has 3 nonzero coefficients), e.g. for w=0.25
- Separable filter =>
- Filtering in 3 dimensions: X, Y and Z
- Symmetric filter $[f_0, f_1, f_2]=[f_0, f_1, f_0]$

How is it done in f3d

```
for each dimension D=X,Y,Z
  for each slice S
    for each line L (in dimension D) of slice S
    {
        copy line L to temporary buffer
        filter line in temporary buffer
        copy filtered line back
    }
```

Possible optimizations

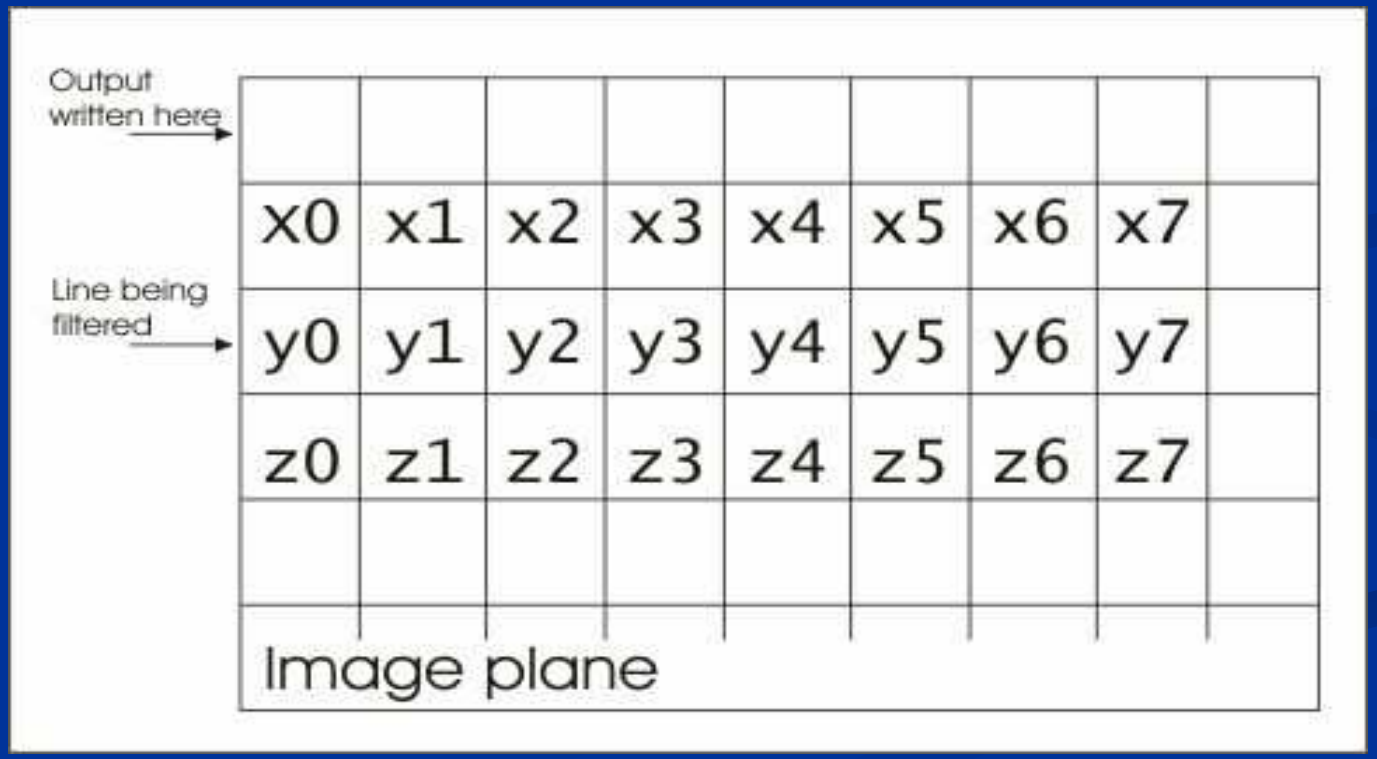
1. Gauss is symmetric – we can save 3 multiplication for every filtered voxel:

   $f_0*x_0 + f_1*x_1 + f_0*x_2 = f_0*(x_0+x_2) + f_1*x_1$

2. Make it as cache friendly as possible => avoid copying

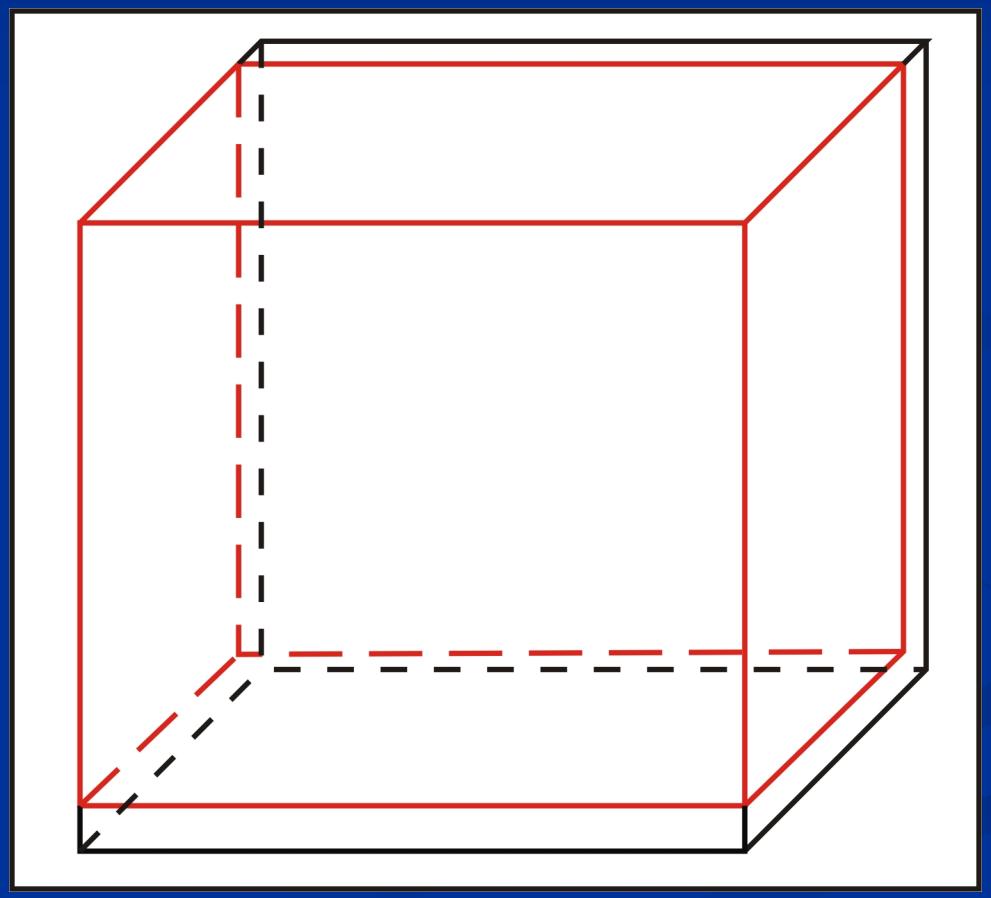
Avoiding copying

- For X and Y direction – add additional line
- For Z direction – add additional slice



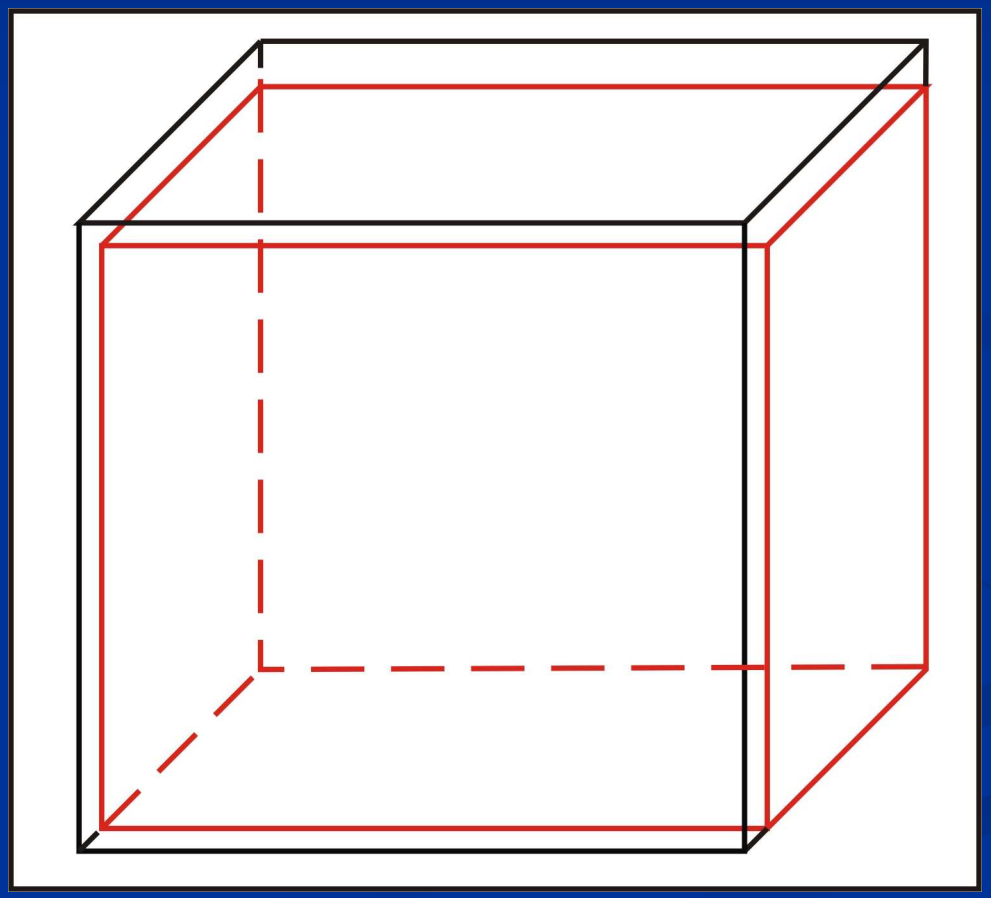
How does it work (1)

- Before filtering



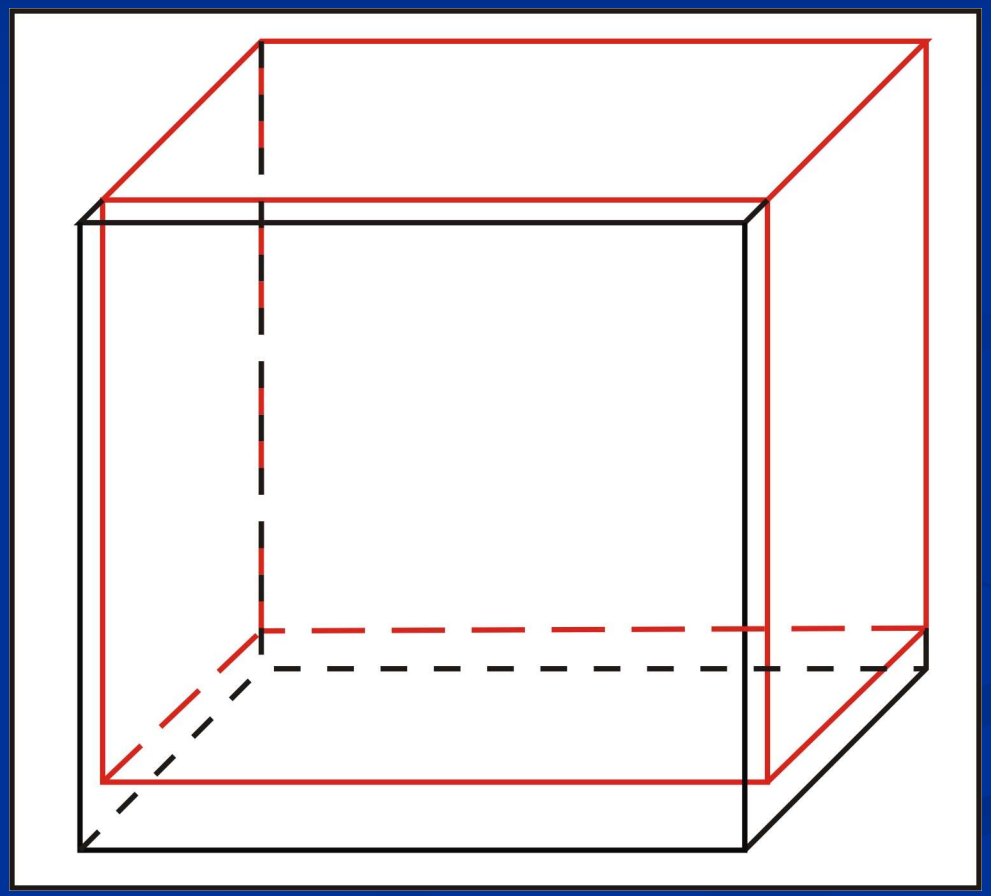
How does it work (2)

- After filtering in X direction



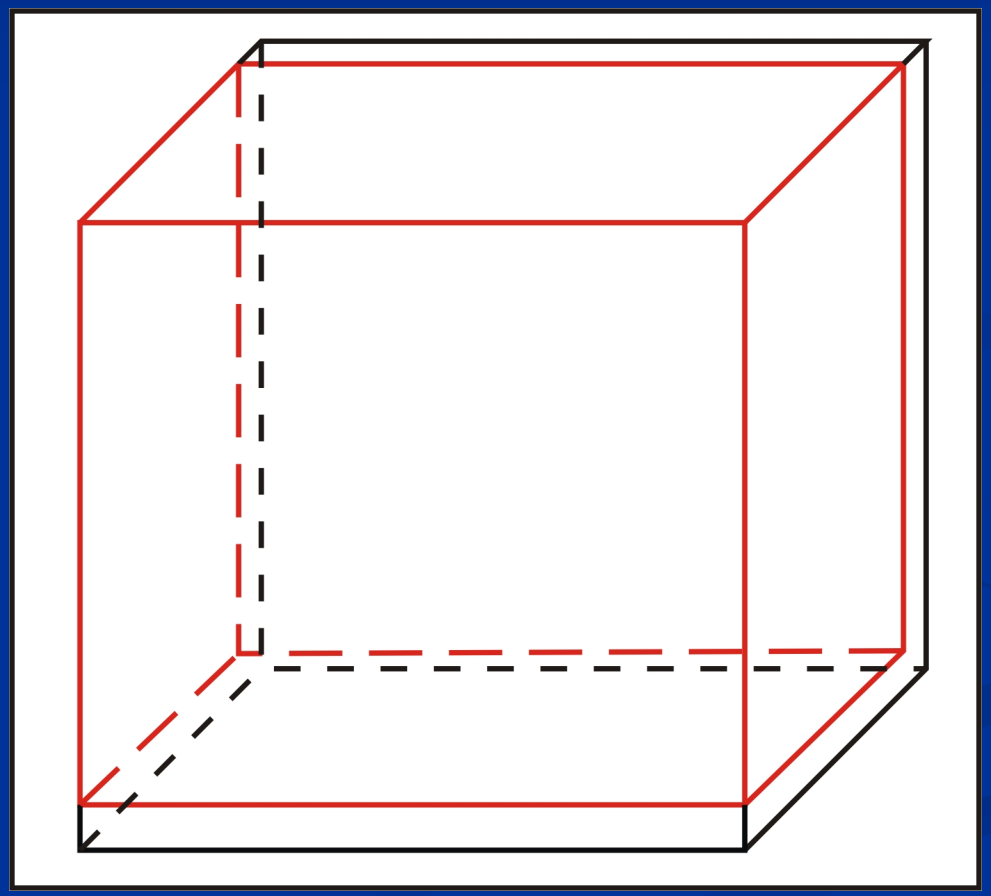
How does it work (3)

- After filtering in Y direction



How does it work (4)

- After filtering in Z direction



Further optimizations

- Filtering in X and Y direction can be done in one pass – reusing cache data
- Neighboring voxels can be filtered in parallel (using SSE)

- Omitting assembler details ☺ (see my previous presentation)

Results

- Measured on Intel Pentium 4 1,6 GHz with 1 GB RAM

	256x128x64	256x256x256	1024x512x64
non-opt	1,10s	8,30s	99,70s
SSE	0,03s	0,30s	3,20s
speedup	36,67	27,67	31,16

Practical limits

- Not general optimization of filtering – currently there are optimized only gauss3, gauss5, gauss7
- Only floating-point data (or convert it on the fly – it is still faster and more precise!)
- Volume dimensions – multiple of 4
- Additional memory is required –

  3 lines/slice + 3 slices

Changes in f3d

- not many
- new raster - f3dSIMDRaster
- assembler implementation of gauss filtering for gauss3, gauss5 and gauss7
- changed f3dBand.cpp (gauss filtering)

Usage in f3d (old way)

```
f3dVolume* v;
v=f3dLoadRawVolume(filename);
v->gaussFilter(sigmax, sigmay, sigmaz);
v->save(filename);
```

Usage in f3d (new way)

```
f3dVolume* v;
v = f3dLoadSIMDVolume(filename);
v->gaussFilter(sigmax, sigmay, sigmaz);
v->save(filename);
```

Conclusion

- My intention (due to motivation article) – SSE can speedup things (theoretically max 4 times) and assembler a little bit
- The praxis has shown that SSE speeds up things “only” 2-3 times
- The biggest impact (when processing huge data) has the change of the algorithm with cache optimization in mind

Future work

- optimization (for f3d) of:

1. gabor filtering
2. general filtering
3. converting routines
4. any other calculation expensive things

Thanks for your attention !

vasko.anton@gmail.com