Coupling Rotor Dynamics with a Parallel Airflow Simulation

Johannes Hofmann, DLR
Melven Röhrig-Zöllner, DLR

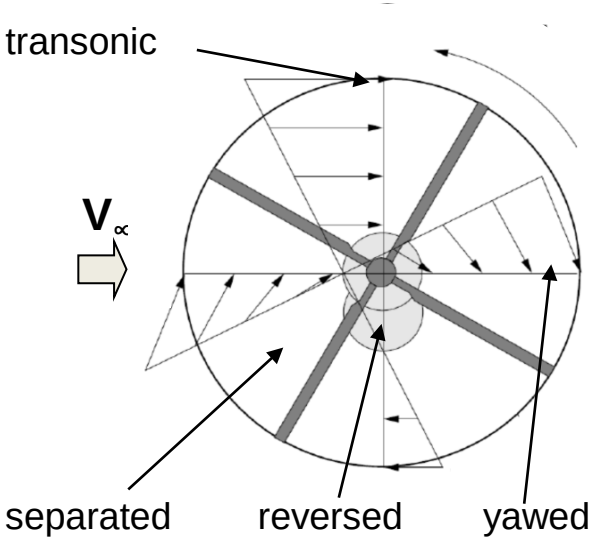
20.6.2016

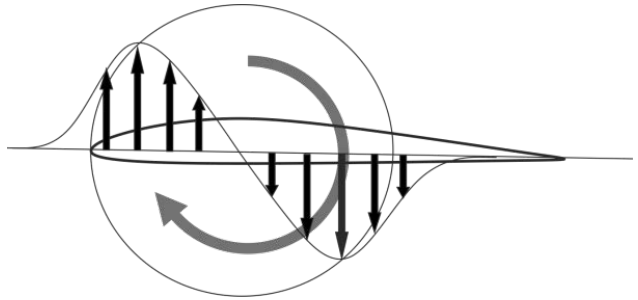
Outline

- What is Freewake and why do we do it?
- Freewake code of the DLR Institute of Flight Systems
- Modernization and port to GPGPU calculations
- Coupling schemes
- Variable rotor speed
- Algorithmic improvements
- Multiple rotors
- Timings
- Conclusion, future work

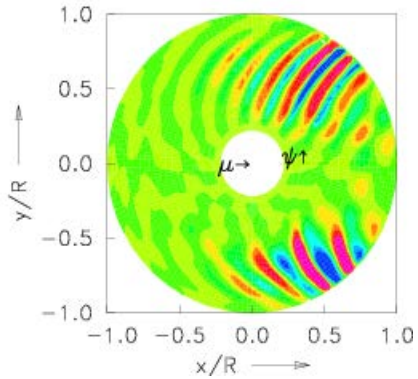
Blade tip vortices on the DLRs Bo105 visualized by BOS
(Background Oriented Schlieren Method)

Why is the rotor wake so important?





Blade vortex interaction (BVI)



BVI locations

Forward flight leads to:

- Different flow velocities at the advancing and retreating side
- Reversed, yawed, separated and transonic flow
- Wake encounters from preceding blades

Influence on

- Noise
- Vibration
- High power-consumption

- Stability
- Performance
- Flight mechanics

Comparison of Wake Models

	Prescribed wake	Free-wake	CFD
Method:	precalculated geometry and influence coefficients	Induced velocities on 2D-grid in 3D-space, vortex aging explicitly modeled	Navier-Stokes equations in every cell of the 3D-mesh
Cost:	1	10^3	10^{6-9}

Prescribed wake

- extremely fast results with good accuracy for conventional cases
- limits: constant RPM, identical blades/motions, steady state conditions

CFD

- very good representation of complex flow patterns
- vortex conservation difficult → of paramount importance for helicopter rotors
- needs extremely fine grids, higher-order discretization
- huge computational cost

Free-wake

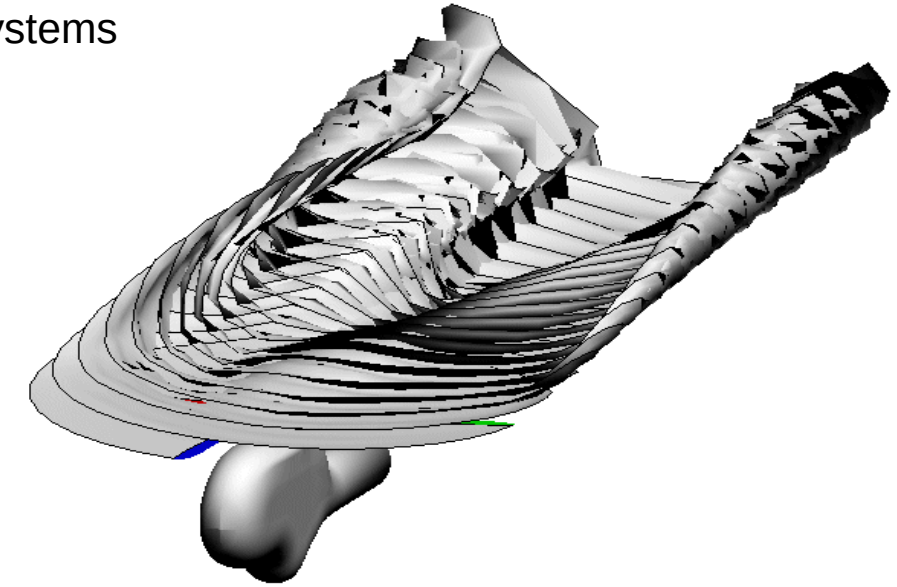
- fast compared to CFD
- explicit vortex-tracing (no numerical dissipation)
- still: most codes need clusters to run on → cumbersome, expensive

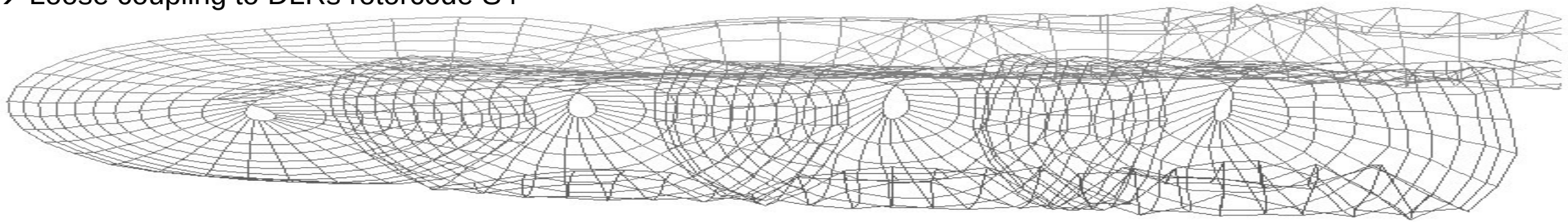
The Initial Code

- Freewake code developed at DLR by van der Wall and Roth 1994-1996
- Modeling based on experimental results from wind tunnel test
- Code designed for massively parallel calculations on distributed memory systems
- Parallelized via MPI (message passing interface)
- Design performance: cycle time of 10min on 200 nodes
- Initial geometry via prescribed wake
- Standard grid:
    - Vortex element length $\Delta\psi$=10°
    - Integration time step $\Delta\psi$=1°
    - 4 revolutions
    - 4 blades
    - 11 Trailed vortices

    → $n_P = 4 \cdot 4 \cdot 11 \cdot \frac{360}{10} = 6336$

→ Loose coupling to DLRs rotorcode S4





Port to GPGPU and Modernization

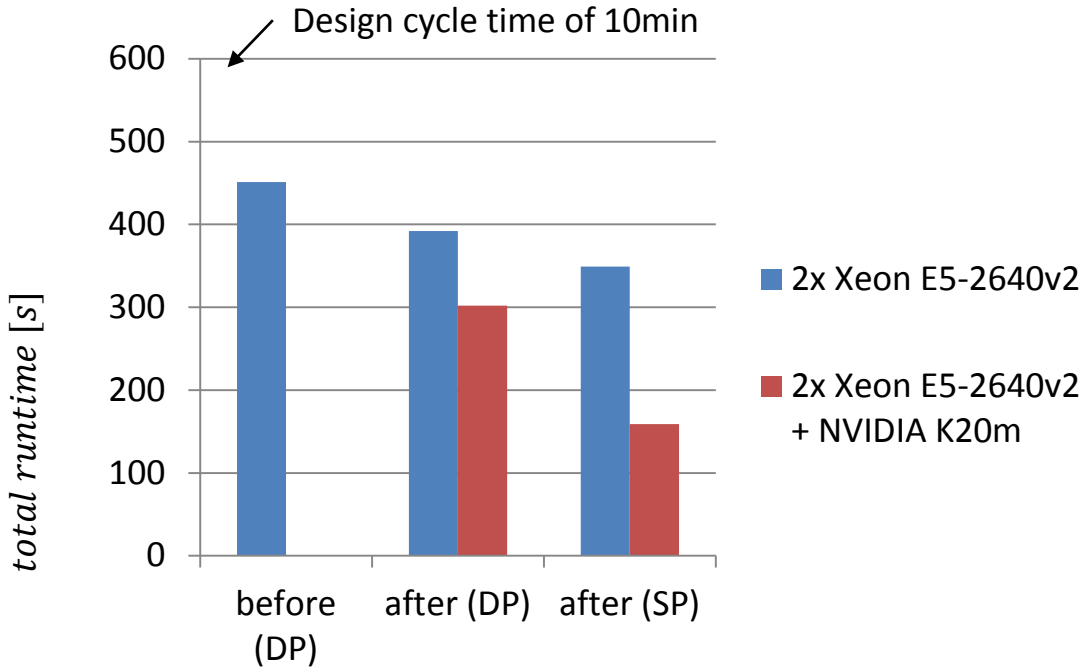
- Successfully ported the Freewake simulation to GPUs using OpenACC
  - original numerical method not modified
  - refactored & restructured a lot of code
  - results verified on CPU and GPU

→ Porting complex algorithms to GPUs is difficult

  - branches in loops hurt (much more than for CPUs)
- Loop restructuring may also improve the CPU performance
  - (SIMD vectorization on modern CPUs)

- Stumbled upon several OpenACC PGI-compiler bugs (all fixed very fast)

- Freewake on a workstation with reasonable cycle times → Goal achieved!

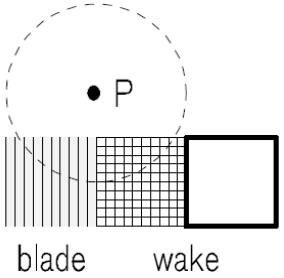


Free-wake Methodology

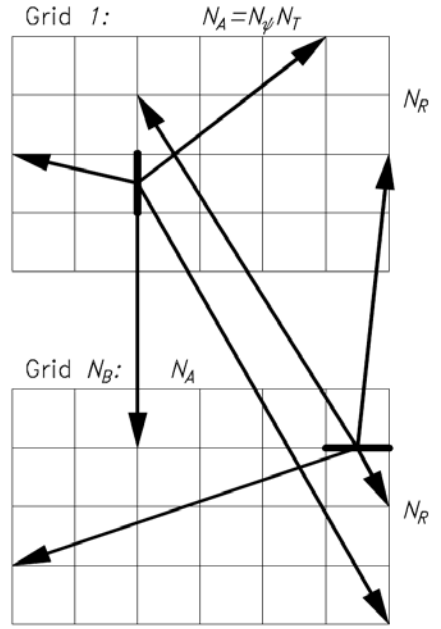
- Basic formulation:
  - Influence from every vortex segment onto every mesh-node via the Biot-Savart law:

$$d\overrightarrow{v_i} = \frac{\Gamma}{4\pi} \cdot \frac{\vec{l} \times d\vec{s}}{|\vec{l}|^3}$$

- Near range:
  - Adaptive grid re-meshing → finer grid if vortex segment is close to point of interest
  - Blade: prescribed strength distribution along the cord
  - Wake: linear varying strength distribution



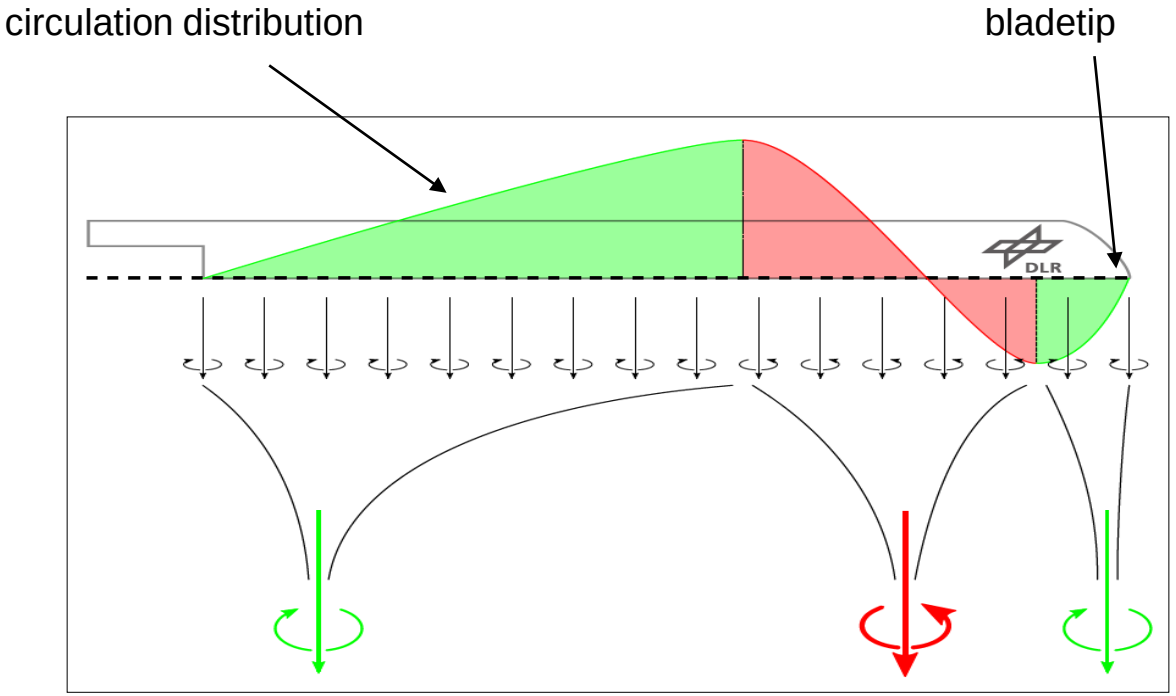
- Far range:
  - Neglected (distance > rotor radius)



Induction on nodes

Wake Rollup and Vortex Aging

- Rollup model:
  - Explicit redistribution of vorticity to the center of vorticity
  - Multiple rolled up vortices depending on sign change of bound circulation
- Energy conservation:
  - Keep the product of vorticity and segment length $(\Gamma \cdot l)$ constant
- Vortex strength:
  - Vorticity decay (empirical)
- Core radius depends on:
  - Core radii of initial shed vortices (prescribed)
  - Radial extension of bound circulation sampled into rolled-up vortex
  - Vortex age
  - Number and intensity of previous interactions with other blades
  - Parameters identified using wind tunnel data



Wake rollup (dual tip vortex system)

Weak coupling

Interface to rotor simulation:

- Circulation and blade motion from rotor simulation code
- Returns induced velocity on the blades

Weak coupling scheme:

First trim performed by rotor simulation using lower fidelity wake model

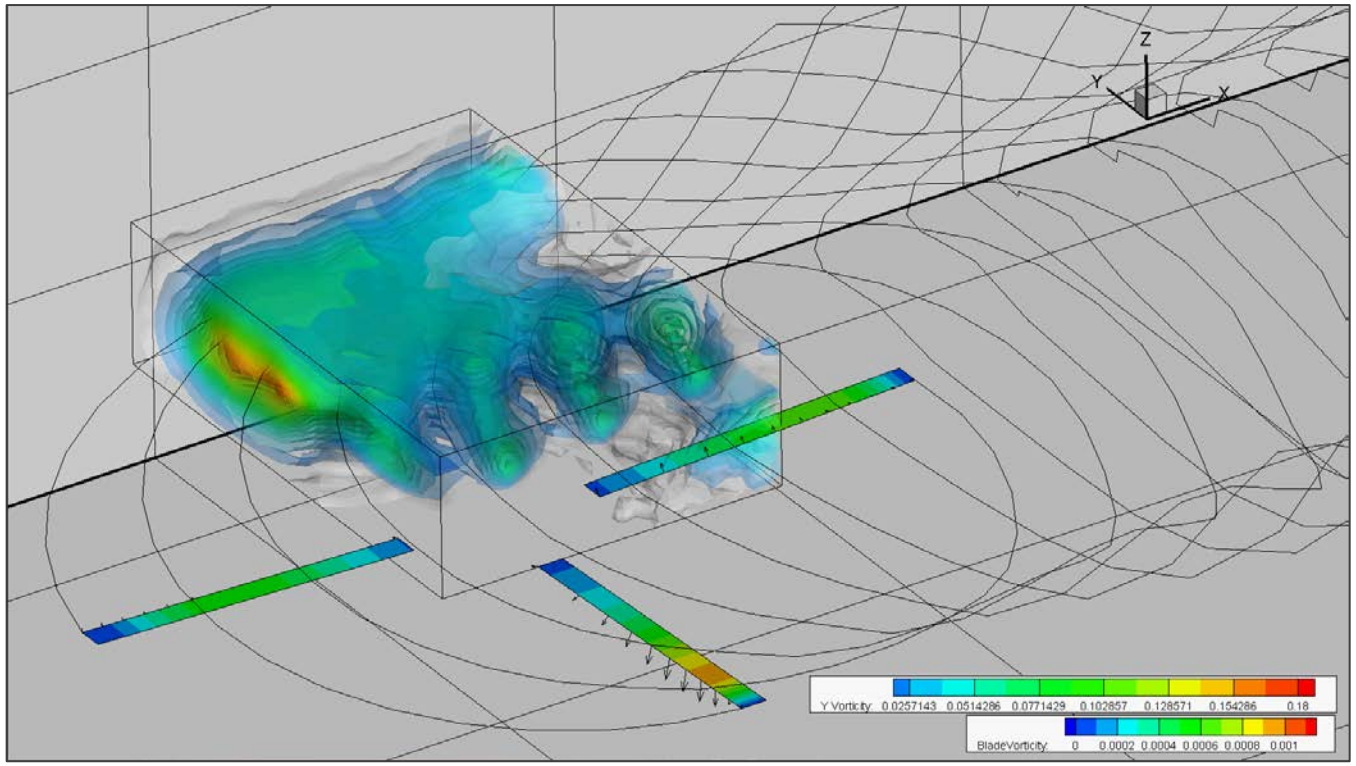
Circulation and blade motion over one revolution saved to file

Freewake calculates a number of revolutions

Induced velocities over one revolution saved to file

Rotor simulation re-trims using the pre-calculated induced velocities

Iteration until the wake does not change anymore (3-5 times)



Y-Vorticity on a helicopter rotor (HART II)

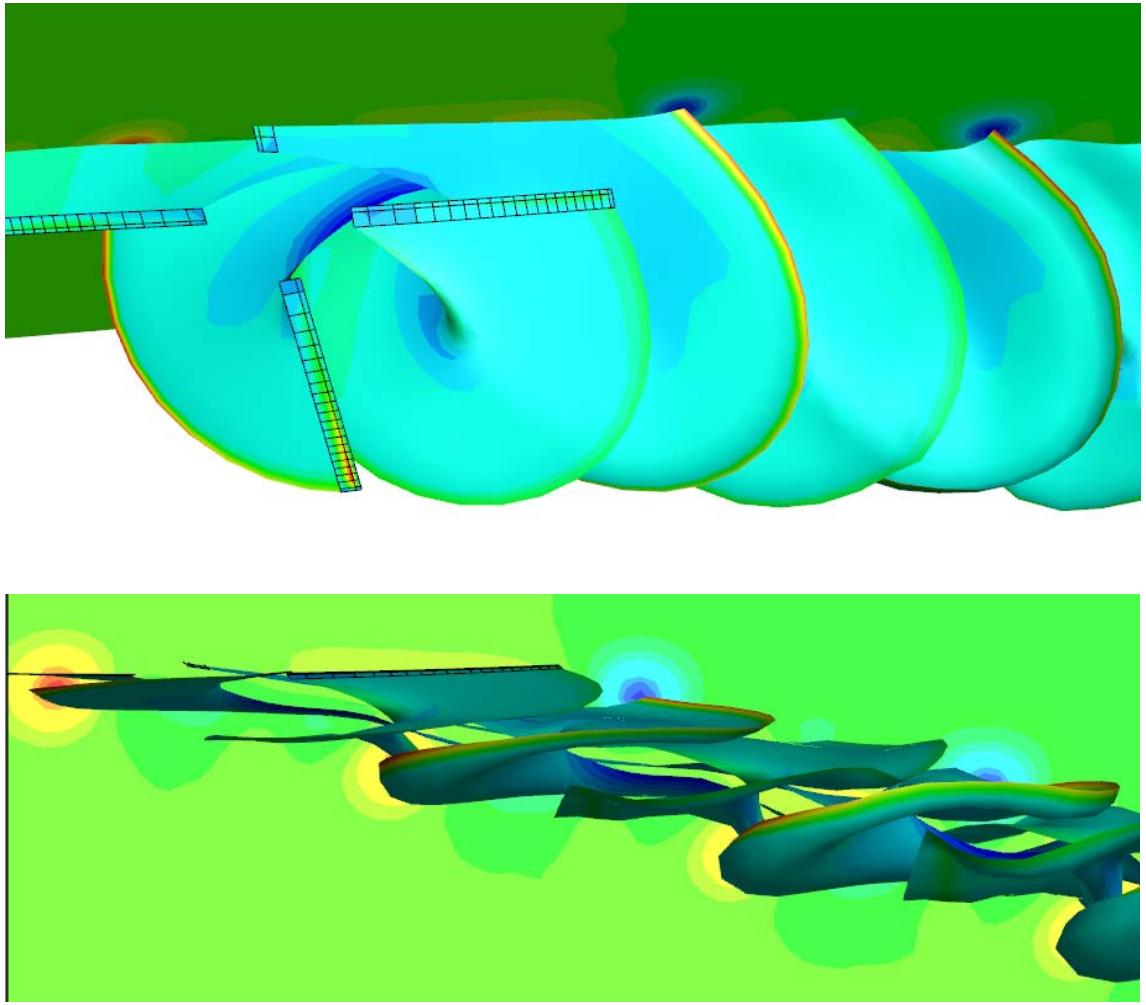
- Usually no more than 4 revolutions per iteration needed (forward flight)
- Converges fast due to good initial wake (prescribed wake model)

Strong coupling – what for?

- Non-harmonic blade motion
- Maneuvering flight
- Flight in the disturbed atmosphere
- Variable rotor speed

Problems

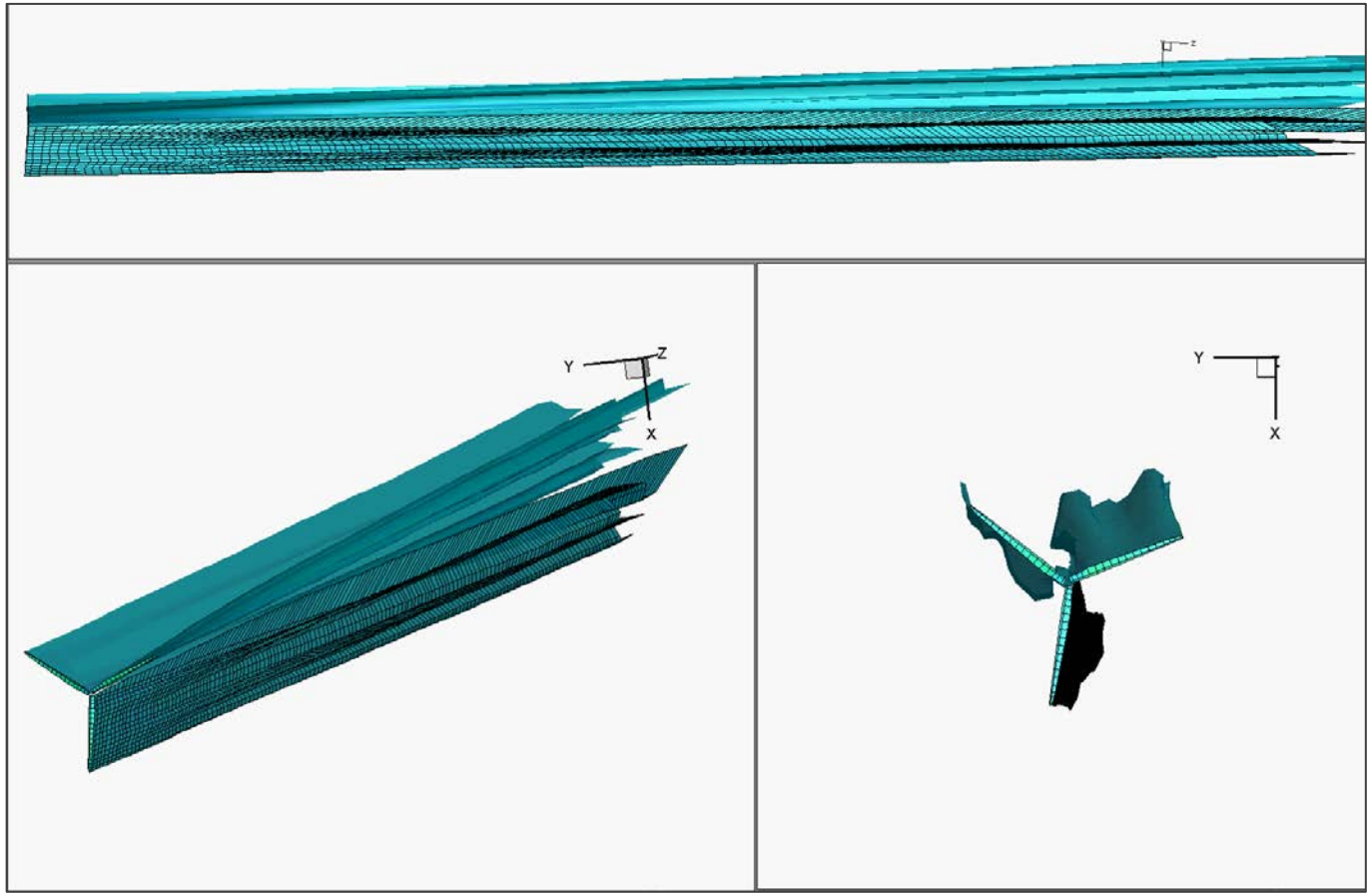
- circular dependencies
  - Freewake requires **circulation** to calculate induced velocities (**inflow**)
  - Aeromechanic model requires **inflow** to calculate **circulation** Idea: prediction + correction steps in Freewake (→ fixed-point iteration)
- New rollup model required
  - Initial model offline as preprocessing step
  - New model online



Strong coupling every second blade 2° pitch offset
(HART II Rotor in fast forward flight)

Variable rotational speed

- Constant RPM → Wake discretization based on azimuthal steps
  - Fixed azimuthal steps
  - New wake element released every 10 steps
- Variable RPM → wake discretization based on initial spacial length of wake element
  - Fixed time steps
  - New wake element released when it reaches a certain length (time steps per new wake element variable)



Turbine startup – NREL 5MW – 10m/s constant wind speed
(playback at 3x original speed)

Algorithmic Improvement: Multistep Time Integration

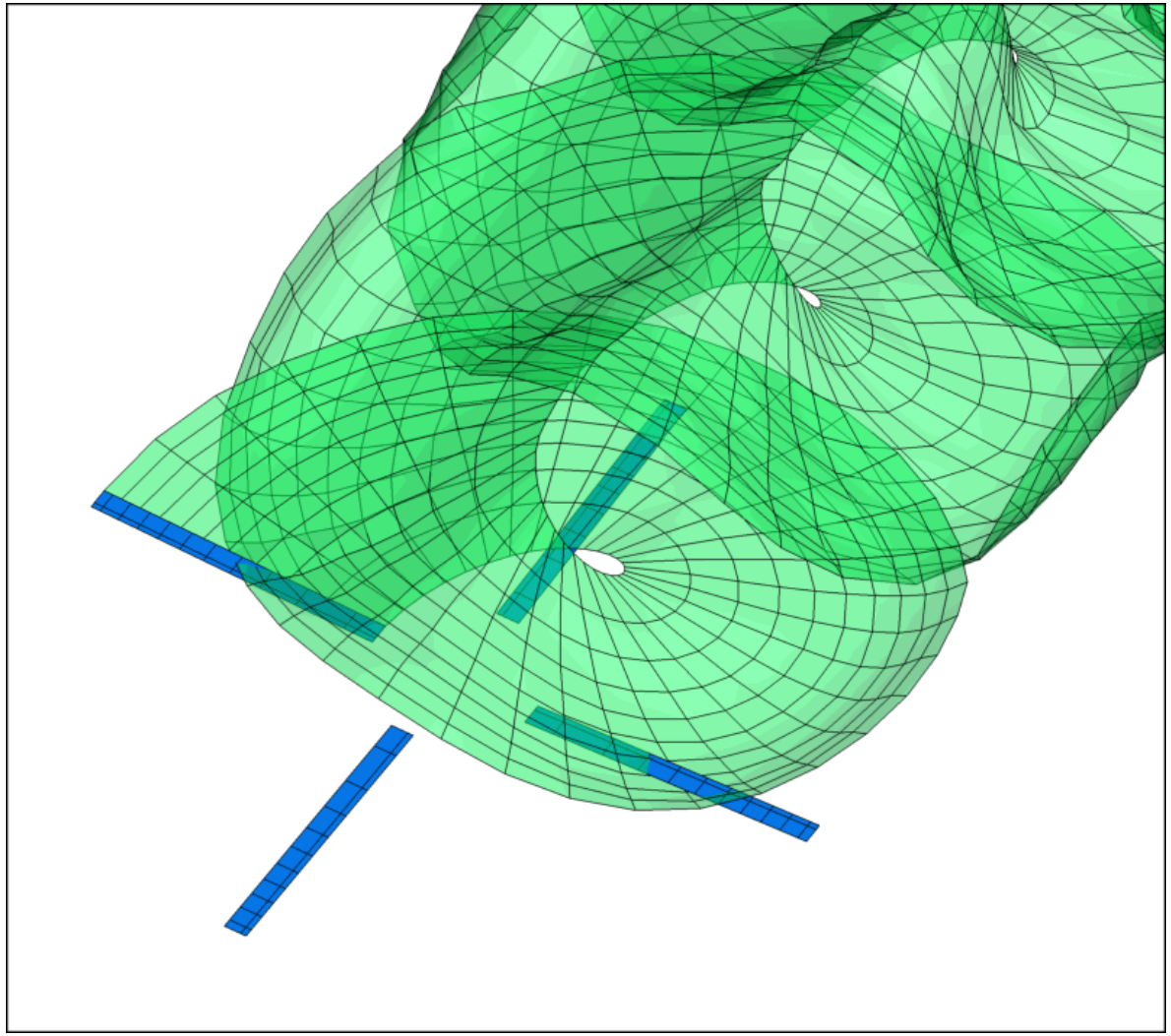
- **Wake moves slowly wrt. rotor blades**

- Split time integration:
  - fast part: explicit Euler for blades→wake
  - slow part: Adams-Bashforth 2 for wake→wake

$$x_{t+1} = x_t + \Delta t\,(b_0 v_t + b_1 v_{t-1})$$

  → reuse wake→wake induction

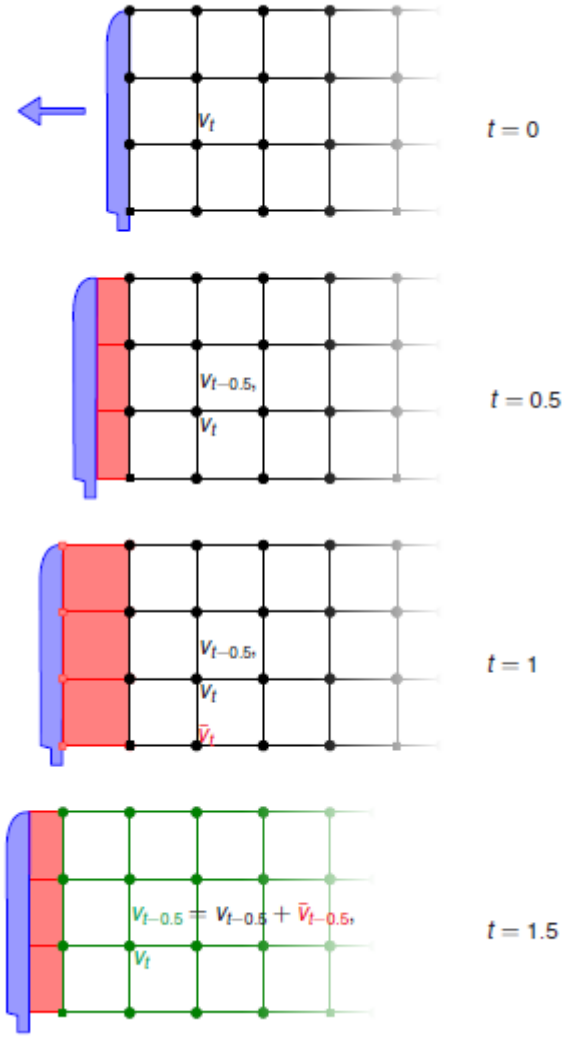
- Tricky implementation:
  - Special handling for first mesh element (no old data available)



First Element in the Multistep Scheme

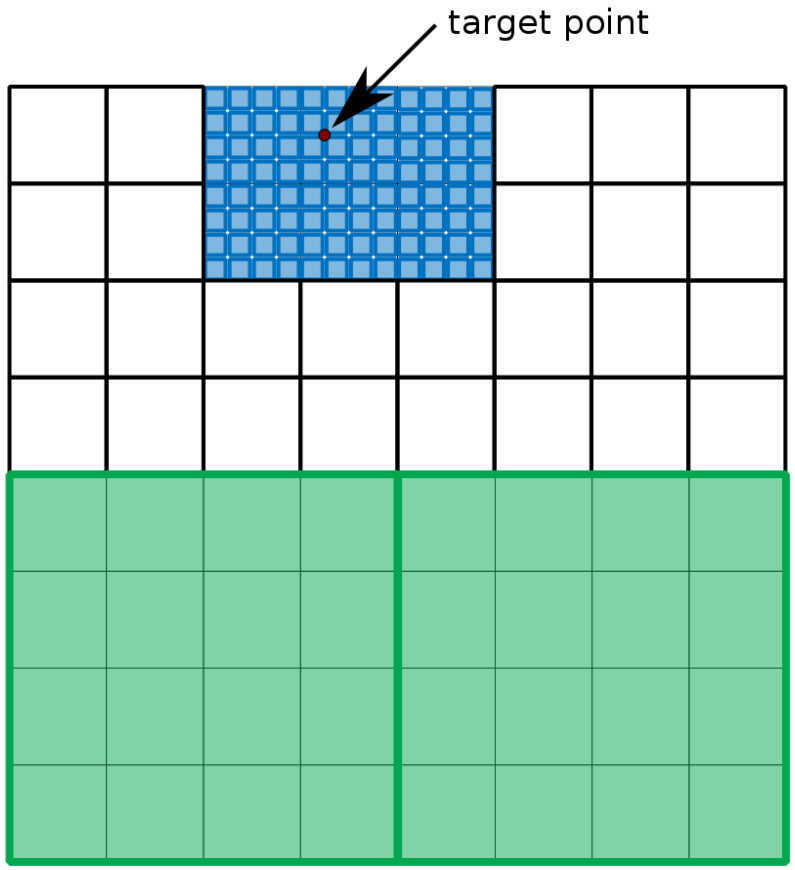
- Calculating/tracking velocities in the wake:
  - t = 0.0 : No old data available, start with explicit Euler.
  - t = 0.5 : Use the two-step method.
  - t = 1.0 : Use the two-step method for **black dots**, explicit Euler for **red dots**.
  - t = 1.5 : Combine **old data**, and use the two-step method.

    (Scheme more complex with variable step size and smaller explicit Euler steps!)



Algorithmic improvement: Multilevel wake induction

- Evaluate surface integral $v(y) = \int_{wake} \frac{(y-x) \times \omega(x)}{(\|y-x\|^2 + r_c^2)^{\frac{3}{2}}} dx$
- Wake discretized using a mesh
  - For small $\|y - x\|$: use fine quadrature rule
  - For medium $\|y - x\|$: use cell midpoint
  - For large $\|y - x\|$: use coarse mesh
- Coarse mesh: add up 4x4 Cells
- Idea of the algorithm:
  - Start with coarse mesh cells, go to finer level
    → avoids n^2 runtime



Parallelization: OpenMP

- Aim: multi core and multi-CPU per node
- Parallelization over coarse grid cells
- Store subset of points for finer level
- Uses OpenMP array reductions

```
real :: sum(m)
!$omp parallel do reduction(+:sum(1:m))
do i = 1, n
   sum(1:m) = sum(1:m) + a(1:m,i)
enddo
```

Parallelization: OpenACC

- Aim: calculations on GPUs

- CPU loop ordering not suitable here
  → dedicated OpenACC code
  Testable on CPUs

- **No array reductions available!**
  - Use atomicAdd operations on global memory
    → quite fast on NVIDIA, slow on AMD GPUs
    (requires further investigation)

- Distributing (coarse) grid cells lacks enough parallelism!
  → 3 individual kernels (coarse cells, original mesh cells, refined quadrature rule)
  - all kernels loop over all points skipping far/near points (theoretically n^2 runtime)
  - coarse mesh helps to reduce #operations and bandwidth

OpenACC / GPUs require multiple levels of parallelism

```
!$acc parallel
!$acc loop gang
do i = 1, n
  !$acc loop worker
  do j = 1, m
    !$acc loop vector
    do k = 1, l
...
!$acc end parallel
```

Proper distribution over these levels very important and not identical to CPU loop ordering

Performance results
(single revolution on 2x12 core Intel Xeon E-2670 and Nvidia Tesla K40m)

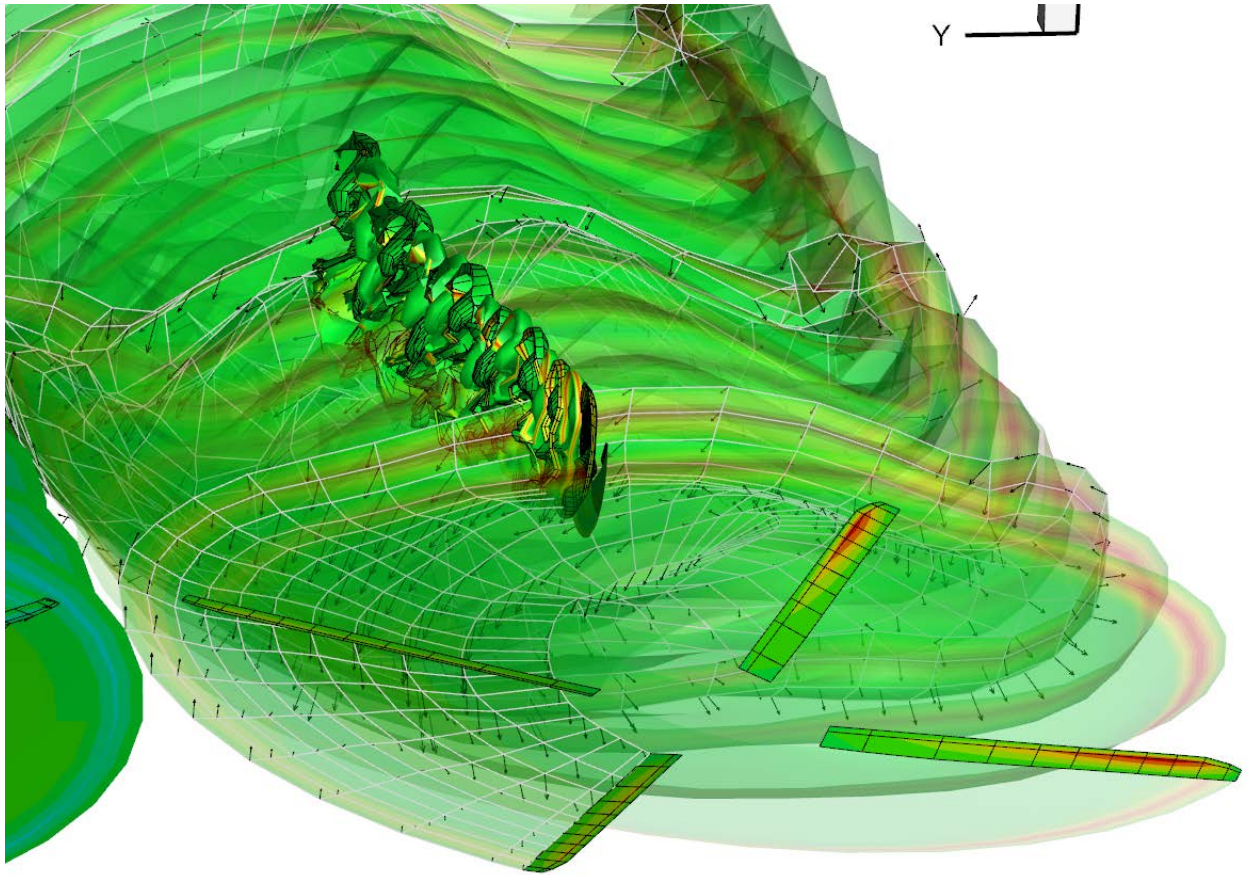
Timestepping	Parallelization	Time [s]
Adams-Bashforth 2	Single core	17.7
Adams-Bashforth 2	OpenMP (12 cores)	1.8
Adams-Bashforth 2	OpenMP (24 cores)	1.2
Adams-Bashforth 2	MPI (24 proc.)	4.0
Adams-Bashforth 2	MPI-OpenMP (2x12)	1.1
Adams-Bashforth 2	OpenACC (K40m)	3.7
Explicit Euler	OpenMP (12 cores)	6.8
Explicit Euler, no multilevel	OpenMP (12 cores)	10.0

Conclusion an future work

- New features:
  - Variable rpm
  - Strong coupling
- Algorithmic improvements:
  - Multilevel induction calculation
  - Multistep time integration
- Design cycle down to ~1 minute (for simple setups)
- Best performance with OpenMP on multicore CPUs for complex algorithms (and reasonable effort)

Next:

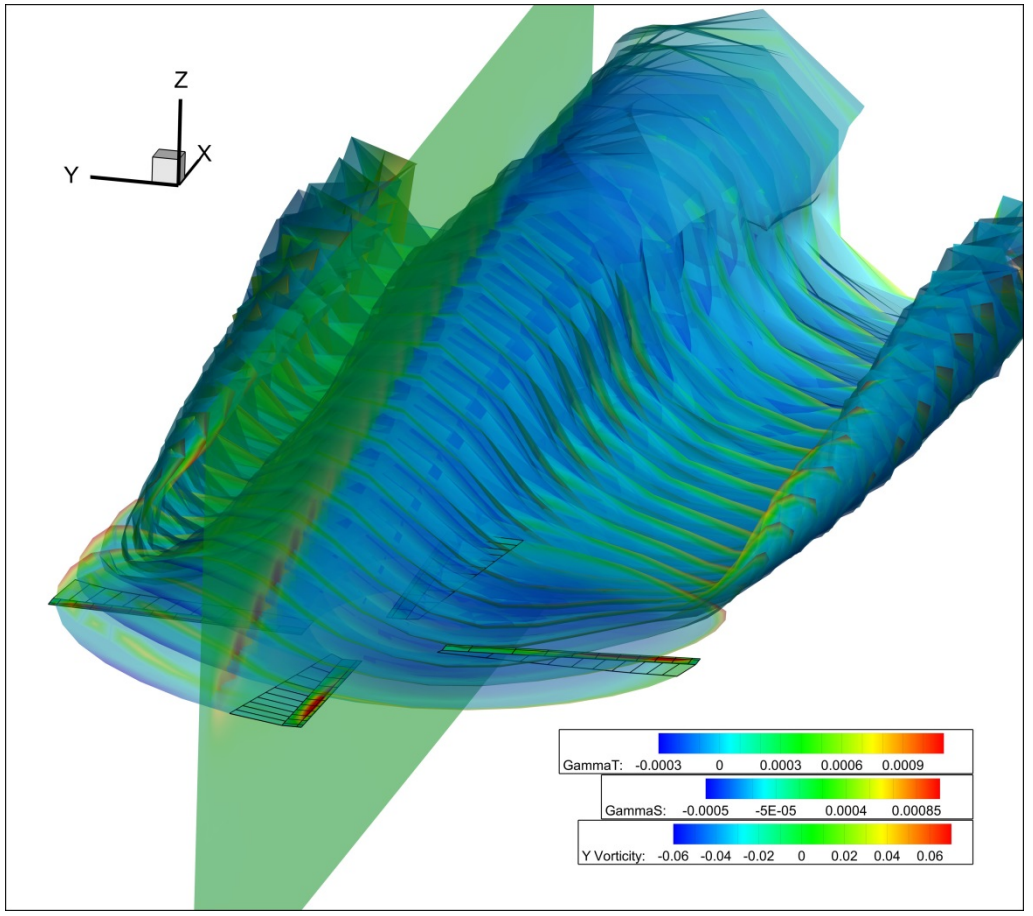
- Multiple rotor simulation
  - Better algorithms for multiple rotors with different RPM (in general: combining rotor flows with effects happening on different time-scales)
- Finer meshes (nearfield) to better resolve BVI events



Thank you for your attention

Parallelization: MPI

- Aim: distribute work between CPU and GPU
- parallelization of wake induction over target points
- Wake data stored redundantly
  - some redundant calculations
- Idea for the future: distribute different rotors on different MPI processes



Multiple Rotors

- Helicopters + Windturbine interactions
- Straight-forward as long as rotational speeds are identical (same step size)
- Large differences in RPM lead to very small convection steps on the slower rotors → large computational effort