Fluid-Solid Interaction Modelling in Fluidity

Frank Milthaler

March 6, 2013

Outline

Motivation

- Fluid-Solid Interaction Modelling
- Flow past turbines (wind/tidal), small scale and large scale

Motivation

- Fluid-Solid Interaction Modelling
- Flow past turbines (wind/tidal), small scale and large scale
- Moving bodies in fluids, e.g. turbines, vehicles

Motivation

- Fluid-Solid Interaction Modelling
- Flow past turbines (wind/tidal), small scale and large scale
- Moving bodies in fluids, e.g. turbines, vehicles
- Arbitrary unstructured mesh
- Mesh adaptivity in FSI
- Conservative projection between solid/fluid mesh

Branch on Launchpad

- Code is hosted as a branch of Fluidity on Launchpad
- Checkout: `bzr branch lp:~f-milthaler/fluidity/fsi-model-sidebranch fluidity-fsi-model`

Branch on Launchpad

- Code is hosted as a branch of Fluidity on Launchpad
- Checkout: `bzr branch lp:~f-milthaler/fluidity/fsi-model-sidebranch fluidity-fsi-model`
- `cd fluidity-fsi-model`
- `module load petsc-gcc4`
- `./configure --enable-2d-adaptivity && make clean && make && make fltools`

Branch on Launchpad

- Code is hosted as a branch of Fluidity on Launchpad
- Checkout: `bzr branch lp:~f-milthaler/fluidity/fsi-model-sidebranch fluidity-fsi-model`
- `cd fluidity-fsi-model`
- `module load petsc-gcc4`
- `./configure --enable-2d-adaptivity && make clean && make && make fltools`
- Code/Project is still under development
- Currently no documentation available
- Will be merged into the trunk in the future

Governing Equations

Momementum Equation and Continuity Equation

$$\rho_f \left(\frac{\partial u}{\partial t} + u \cdot \nabla u \right) = -\nabla p + \nabla \cdot \tau + B + F_s \tag{1}$$

$$\nabla \cdot u = 0 \tag{2}$$

with

$$F_s = \sigma_f \left(u_s - u \right) \tag{3}$$

in which $\sigma_f = \frac{\alpha_s \rho_f}{\Delta t}$ and α_s being the solid volume fraction on the fluid mesh

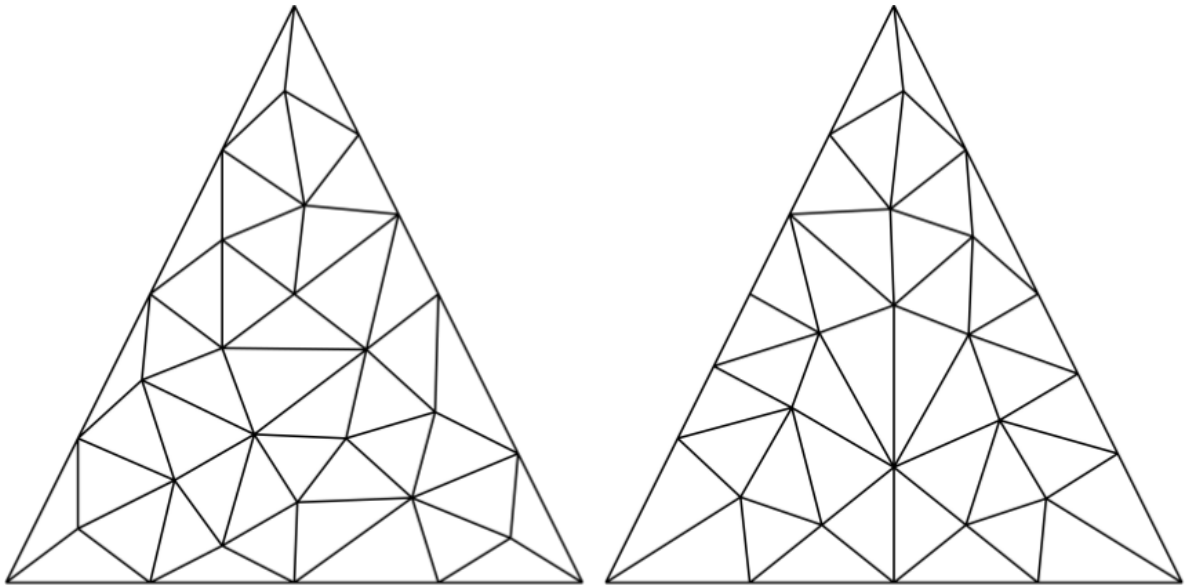
Projection

- Galerkin projection via supermeshing[1]

[1] Farrell, Piggott et al.: *Conservative interpolation between unstructured meshes via supermesh construction*;
and Farrell, Maddison: *Conservative interpolation between volume meshes by local Galerkin projection*

Projection

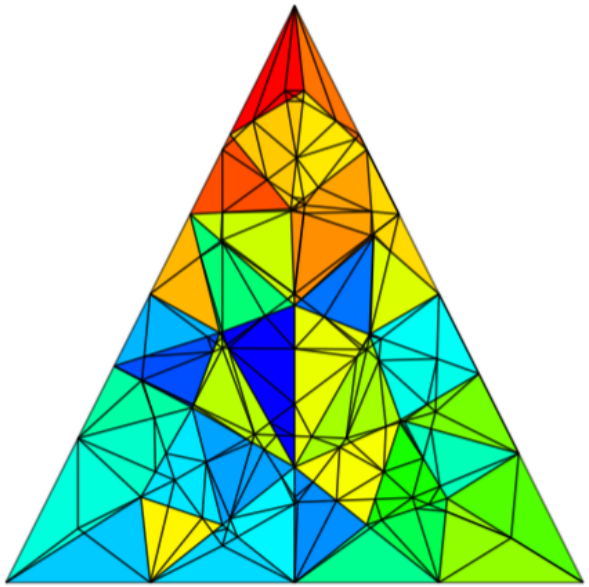
Donor and target mesh of same domain[2]



[2]Graphic taken from: Farrell, PhD Thesis, *Galerkin projection of discrete fields via supermesh construction*

Projection

Supermesh with the elements of the target mesh being coloured[2]



[2]Graphic taken from: Farrell, PhD Thesis, *Galerkin projection of discrete fields via supermesh construction*

Projection

- Galerkin projection via supermeshing[1]
- Conservative projection of fields between meshes
- Bounded (with minimal numerical diffusion) or not bounded

[1] Farrell, Piggott et al.: *Conservative interpolation between unstructured meshes via supermesh construction*;
and Farrell, Maddison: *Conservative interpolation between volume meshes by local Galerkin projection*

Projection

- Galerkin projection via supermeshing[1]
- Conservative projection of fields between meshes
- Bounded (with minimal numerical diffusion) or not bounded
- Local supermesh of intersecting elements

[1] Farrell, Piggott et al.: *Conservative interpolation between unstructured meshes via supermesh construction*;
and Farrell, Maddison: *Conservative interpolation between volume meshes by local Galerkin projection*

Projection

- Galerkin projection via supermeshing[1]
- Conservative projection of fields between meshes
- Bounded (with minimal numerical diffusion) or not bounded
- Local supermesh of intersecting elements
- Originally developed for mesh adaptivity methods: Projecting fields from old mesh to new (adapted) mesh

[1] Farrell, Piggott et al.: *Conservative interpolation between unstructured meshes via supermesh construction*;
and Farrell, Maddison: *Conservative interpolation between volume meshes by local Galerkin projection*

Projection: FSI

Usage of Galerkin projection via supermesh for FSI modelling:

- τ_F, τ_S: Fluid and Solid volume meshes of the domains Ω_F, Ω_S, and elements K_F and K_S respectively

Projection: FSI

Usage of Galerkin projection via supermesh for FSI modelling:

- τ_F, τ_S: Fluid and Solid volume meshes of the domains Ω_F, Ω_S, and elements K_F and K_S respectively
- Typically: $\Omega_F \supseteq \Omega_S$
- Thus, some $K_F \cap K_S$ are of measure zero

Projection: FSI

Usage of Galerkin projection via supermesh for FSI modelling:

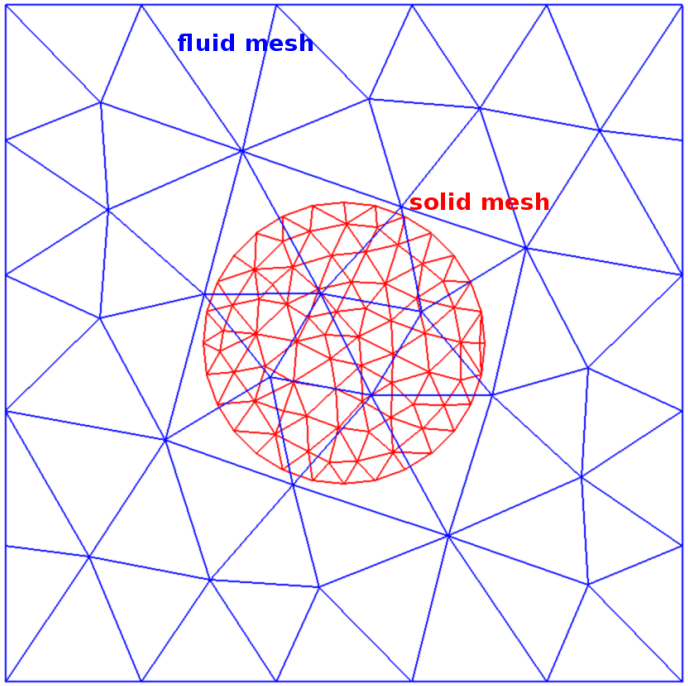
- τ_F, τ_S: Fluid and Solid volume meshes of the domains Ω_F, Ω_S, and elements K_F and K_S respectively
- Typically: $\Omega_F \supseteq \Omega_S$
- Thus, some $K_F \cap K_S$ are of measure zero
- Supermesh is constructed for any K_S for which $K_S \cap K_F$ of nonzero measure

Projection: FSI

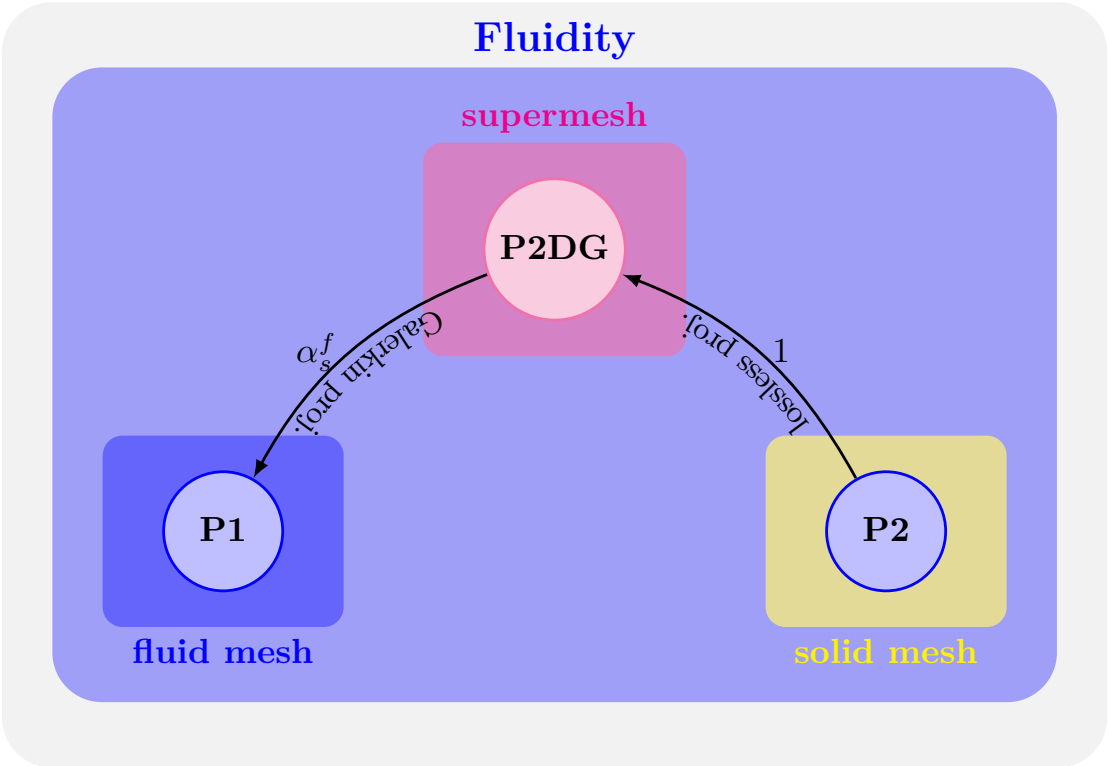
Usage of Galerkin projection via supermesh for FSI modelling:

- τ_F, τ_S: Fluid and Solid volume meshes of the domains Ω_F, Ω_S, and elements K_F and K_S respectively
- Typically: $\Omega_F \supseteq \Omega_S$
- Thus, some $K_F \cap K_S$ are of measure zero
- Supermesh is constructed for any K_S for which $K_S \cap K_F$ of nonzero measure
- Must be bounded!

Projection: FSI



Projection: FSI



Legend:

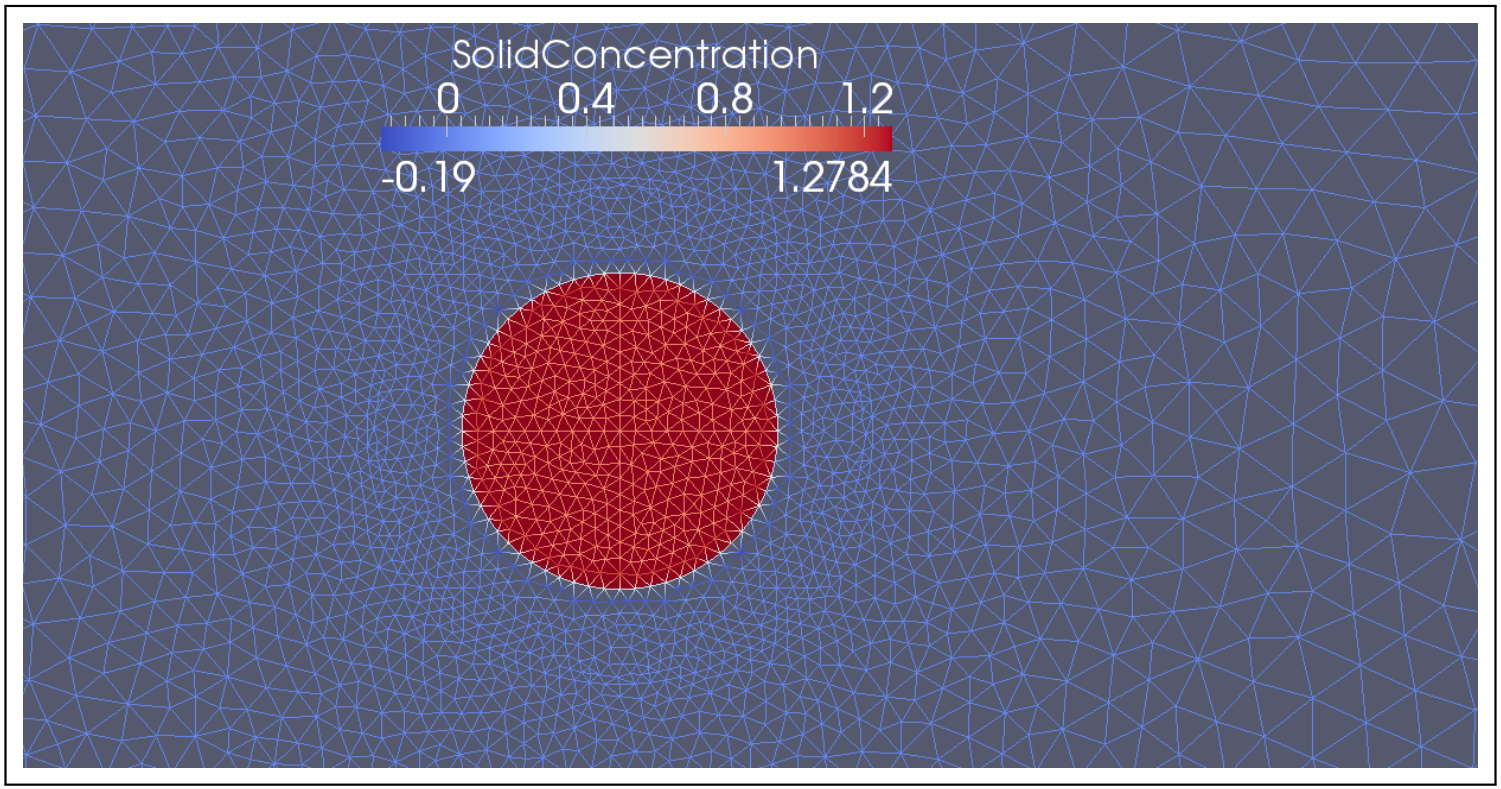
DG Discontinuous

P2 Quadratic mesh

P1 Linear mesh

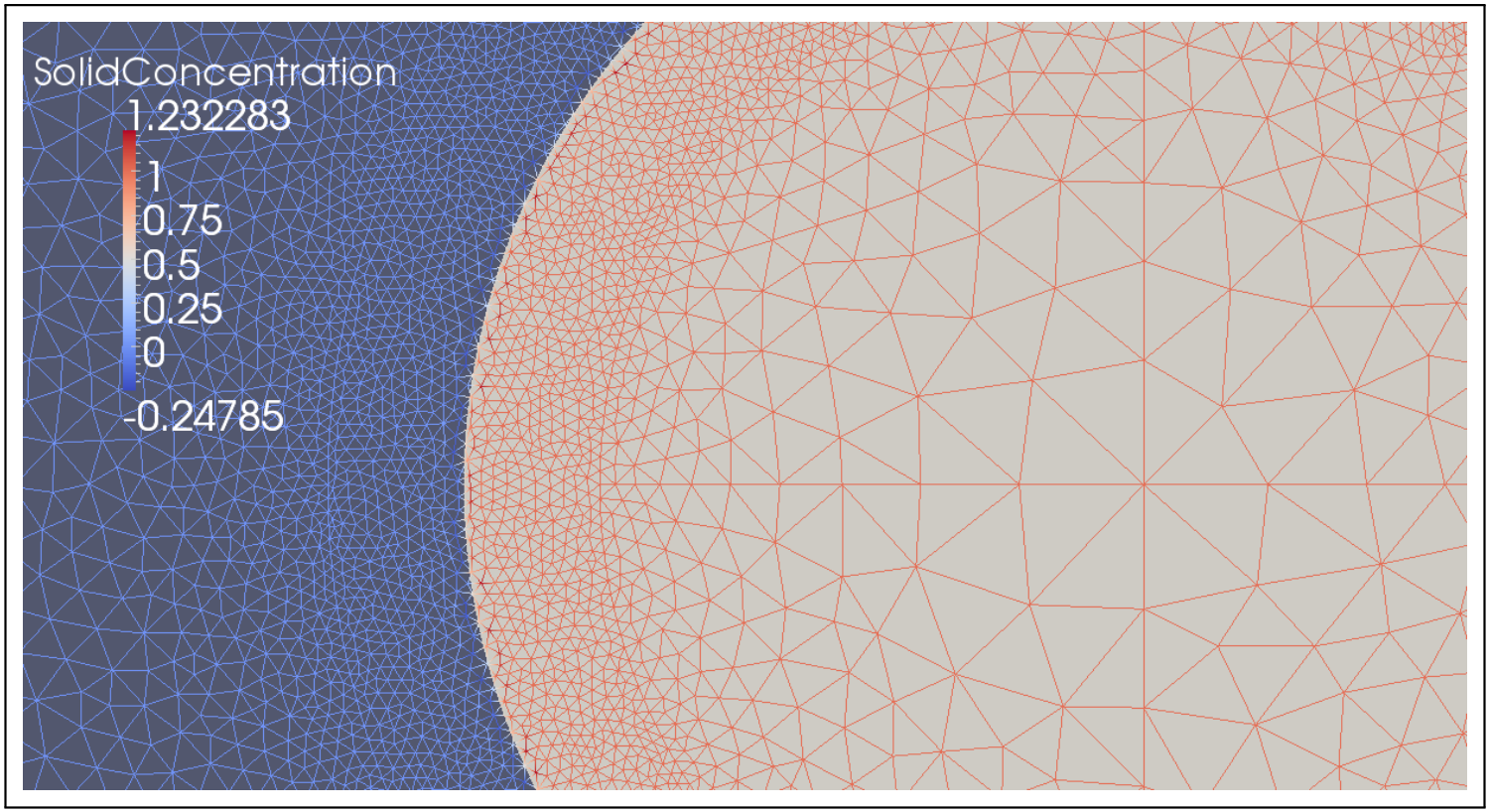
Projection: FSI

Unbounded projection:



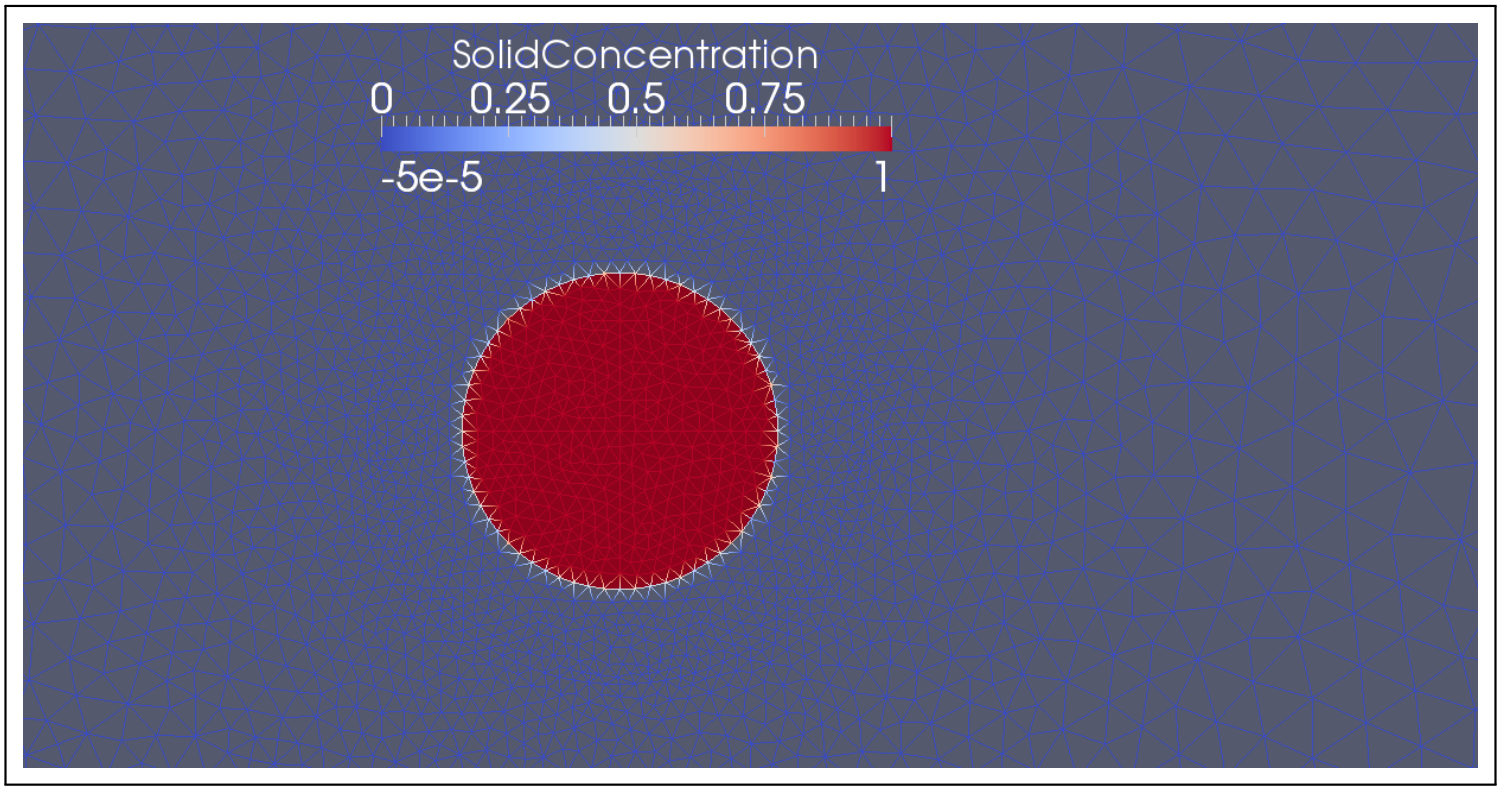
Projection: FSI

Unbounded projection:



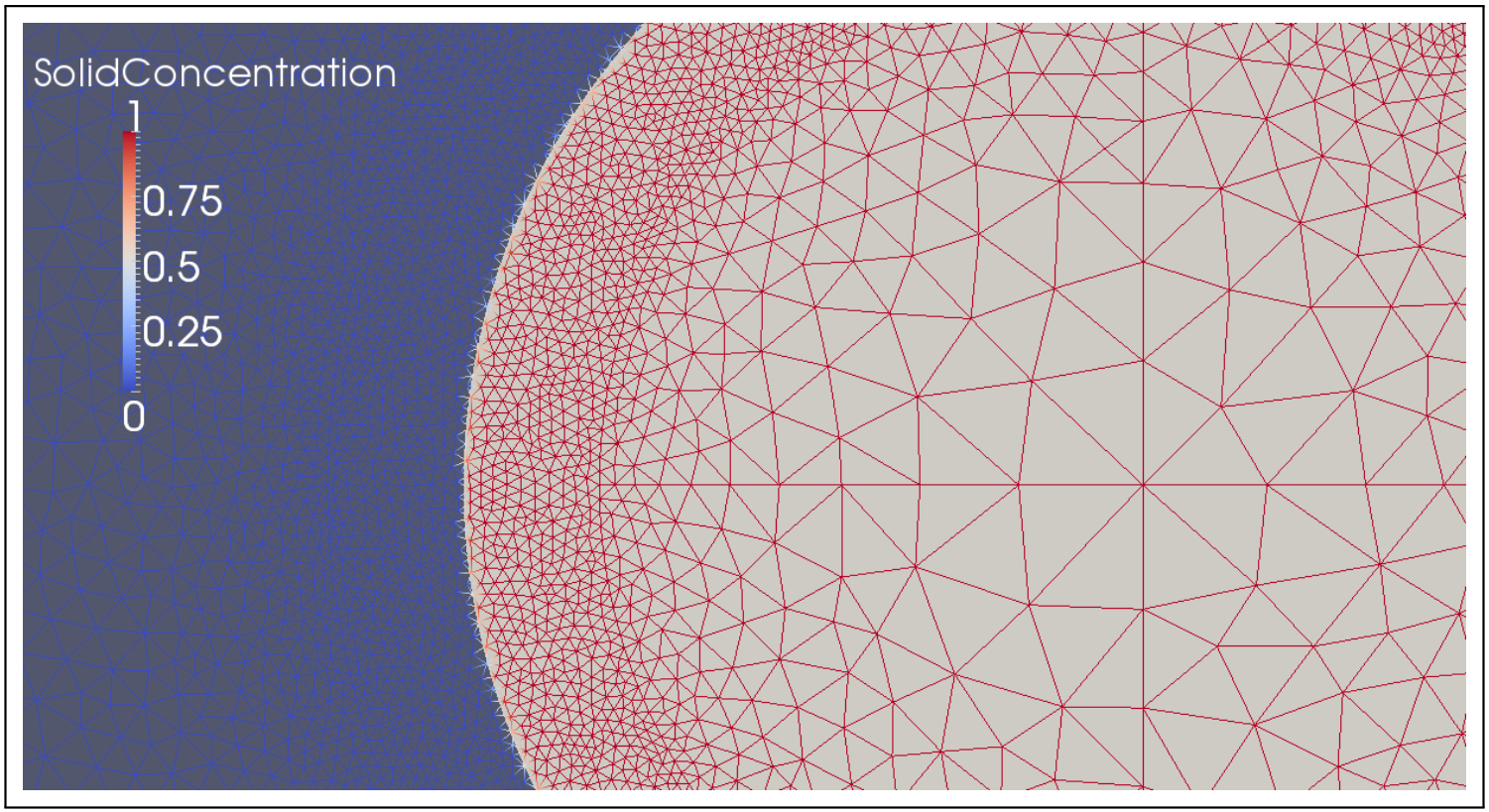
Projection: FSI

Bounded projection:



Projection: FSI

Bounded projection:



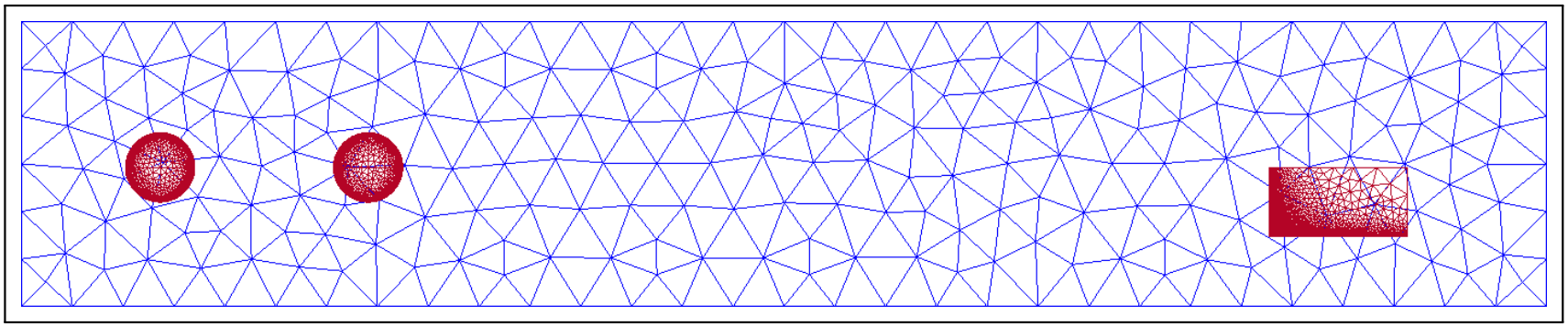
Mesh Adaptivity for FSI

Mesh adaptivity is very useful for FSI modelling:

- Refining the area of interest, e.g. space around the solid body
- Can be done as a preprocessing task to avoid manual local refinement
- Makes it simply to track a moving body
- Reduces computational effort

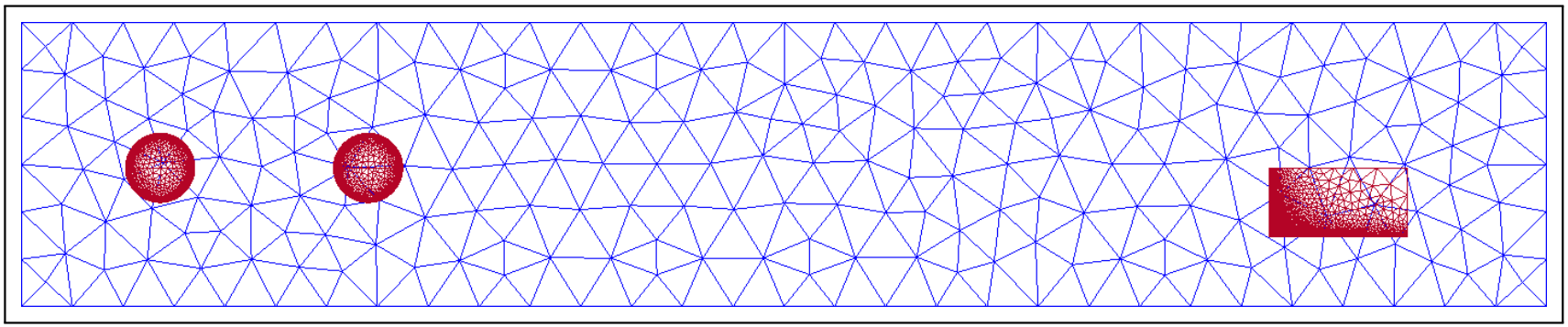
Mesh Adaptivity for FSI

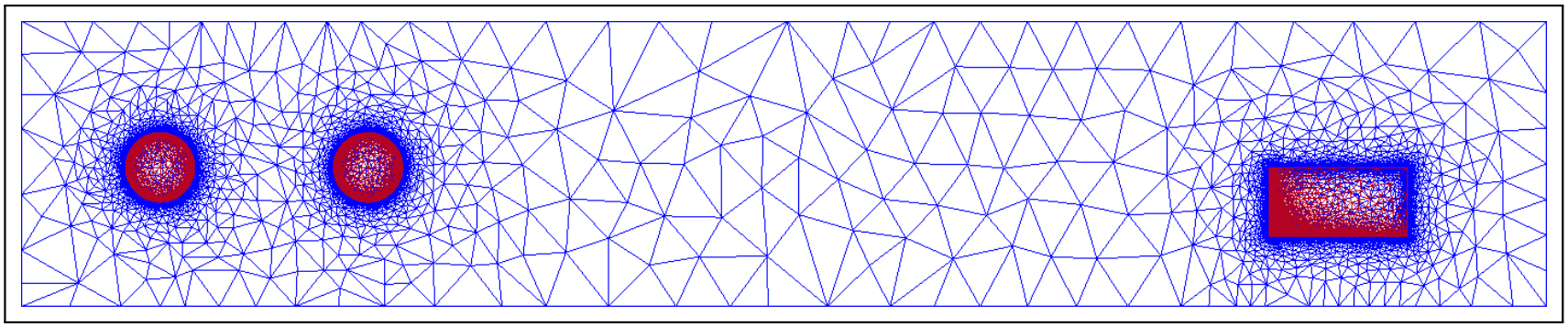
Fluid mesh and Solid meshes



Mesh Adaptivity for FSI

Fluid mesh and Solid meshes





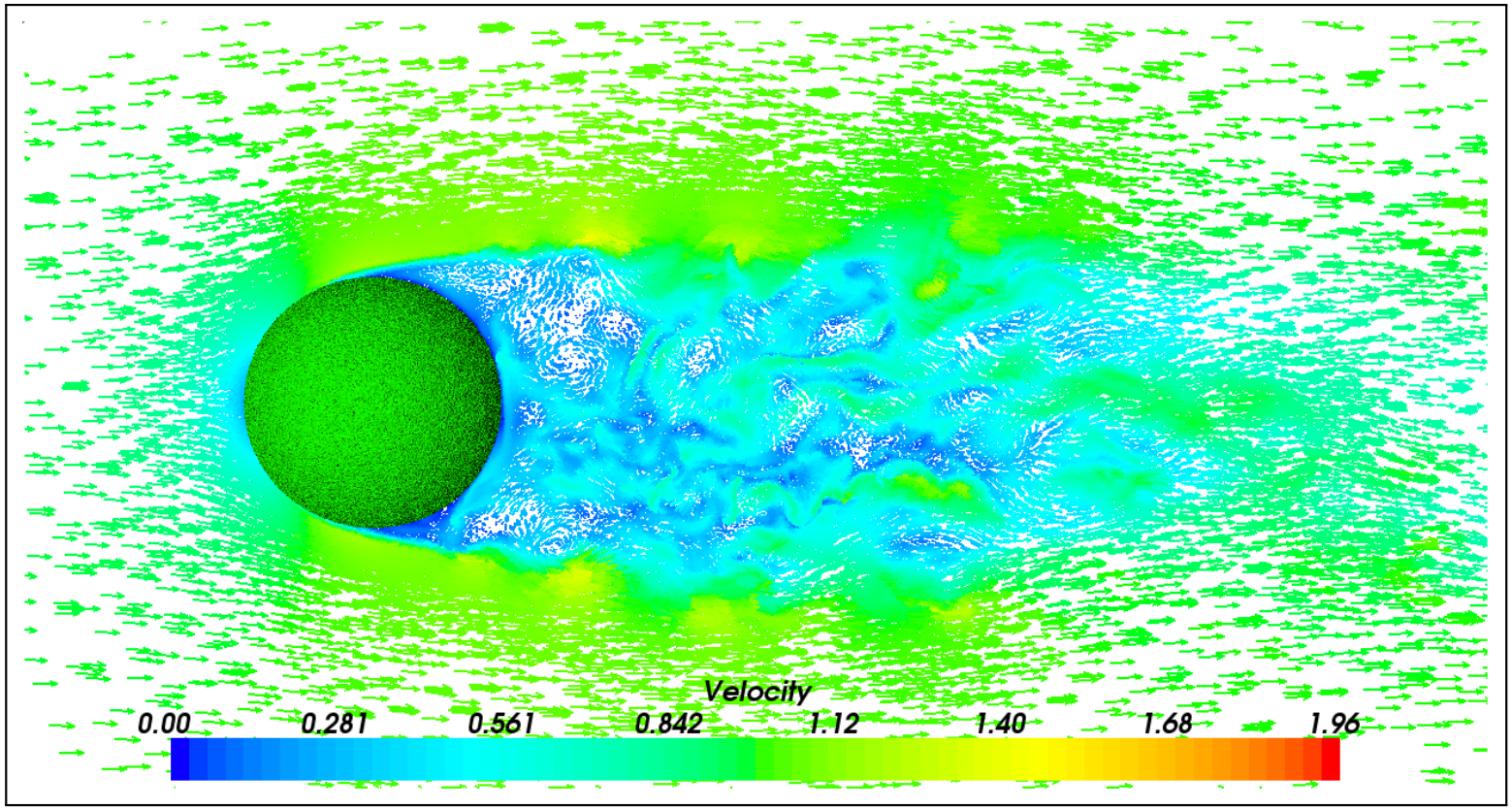
Applications

- Fixed solid
  - Flow past a cylinder (2D, 3D), $Re = 20$ and $Re = 100$
  - Flow past a sphere, $Re = [20, 5000]$
  - Von Karman Vortex Street
  - Flow past aerofoil

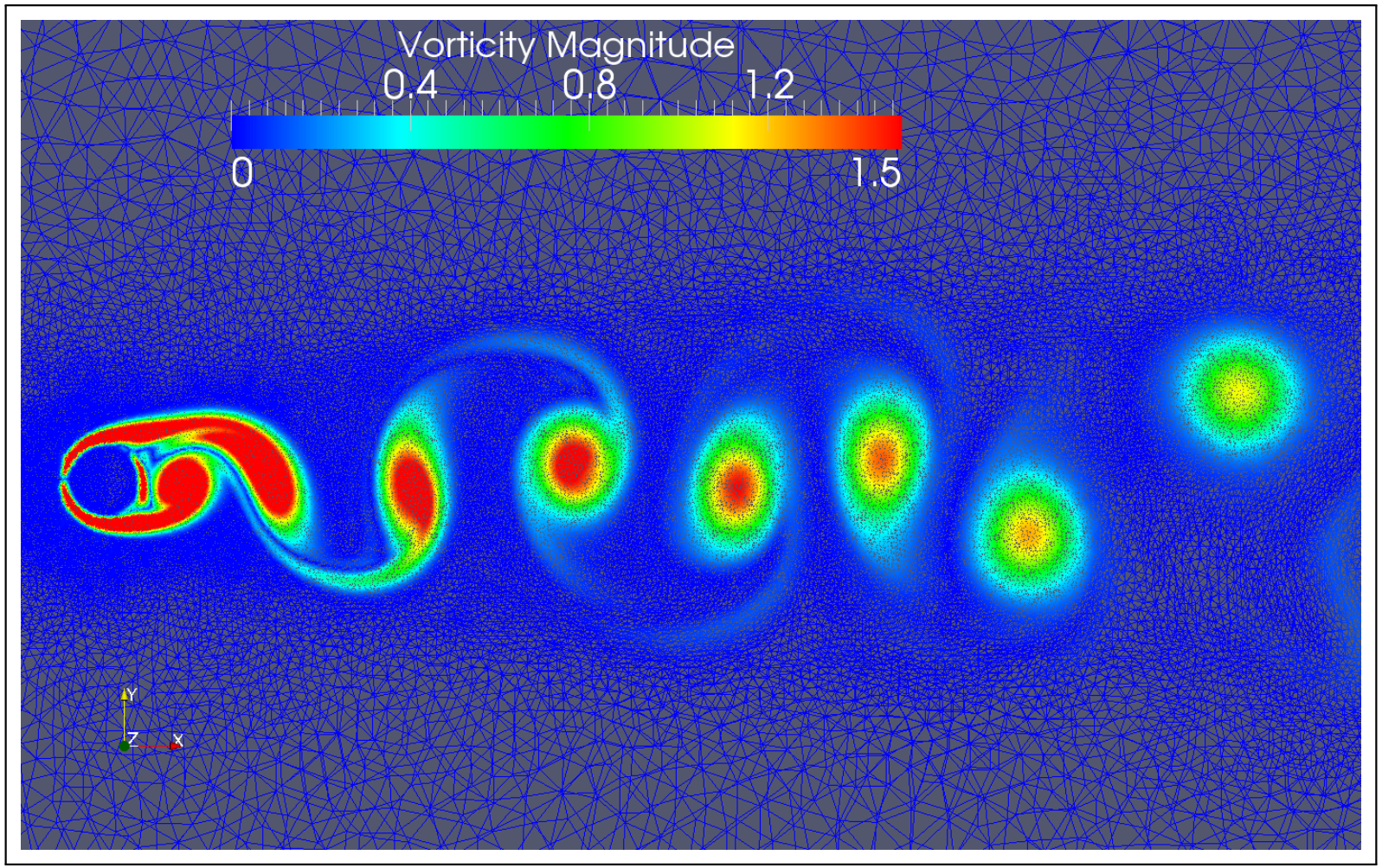
Applications

- Fixed solid
  - Flow past a cylinder (2D, 3D), $Re = 20$ and $Re = 100$
  - Flow past a sphere, $Re = [20, 5000]$
  - Von Karman Vortex Street
  - Flow past aerofoil
- Moving solid:
  - St Andrews Cross
  - Wind turbine, Tidal Turbine
  - Ellipse/Ellipsoid moving through stratified fluid

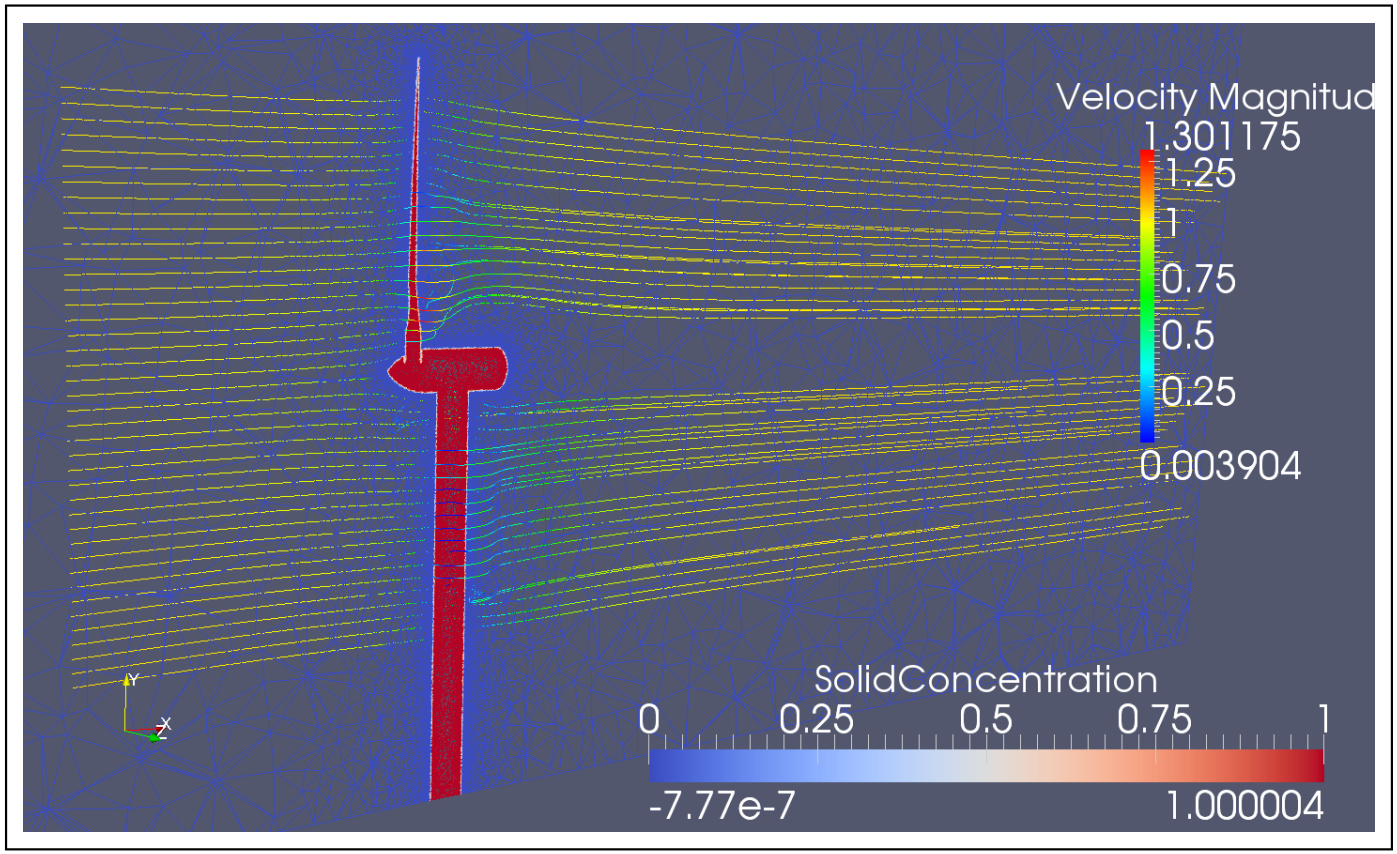
Fixed Solid



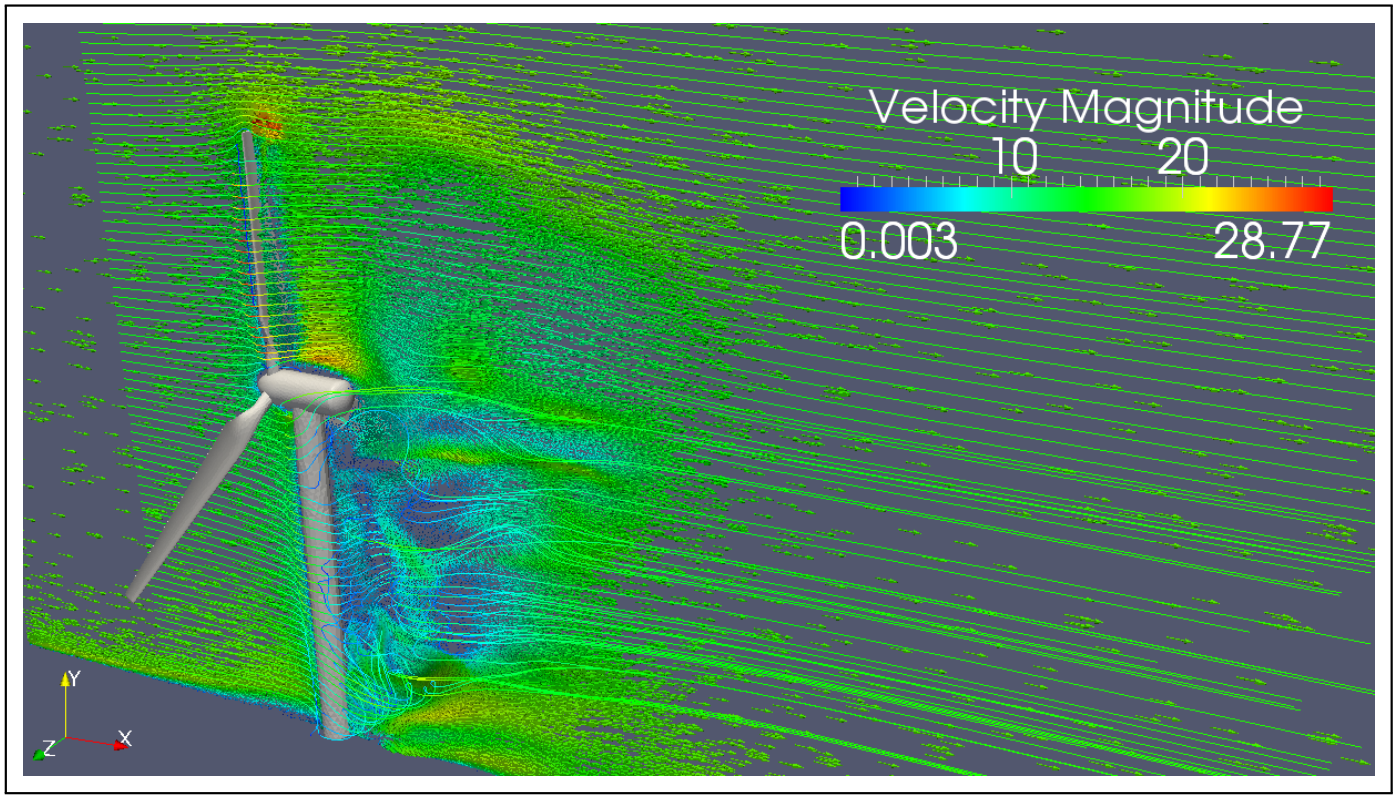
Fixed Solid



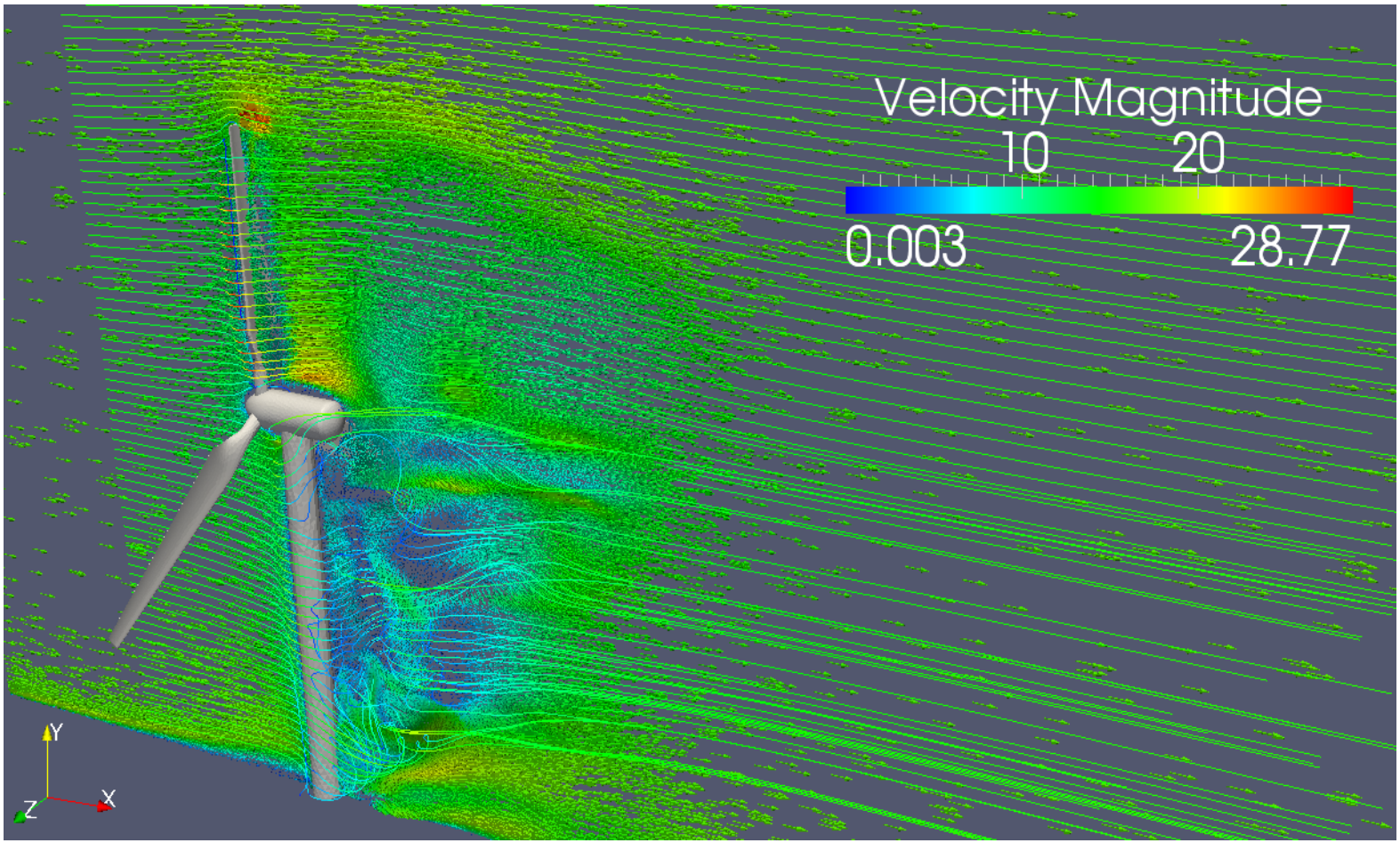
Rotating Wind Turbine



Rotating Wind Turbine



Rotating Wind Turbine



St Andrews Cross

- Momentum Source[3]
- Angle of internal wave propagation:

$$\sin\alpha = \frac{\omega}{N} \tag{4}$$

in which ω is the oscillation frequency, and $N = \left((g/\rho_0)\left|\frac{d\rho}{dy}\right|\right)^{1/2}$ the buoyancy frequency, with ρ, ρ_0, and g being the density, reference density, and gravitational acceleration respectively

- Validation for FSI with prescribed solid velocity

[3] Javam et al, *Numerical study of internal wave reflection from sloping boundaries*

St Andrews Cross

- Momentum Source[3]
- FSI model
- Both with:

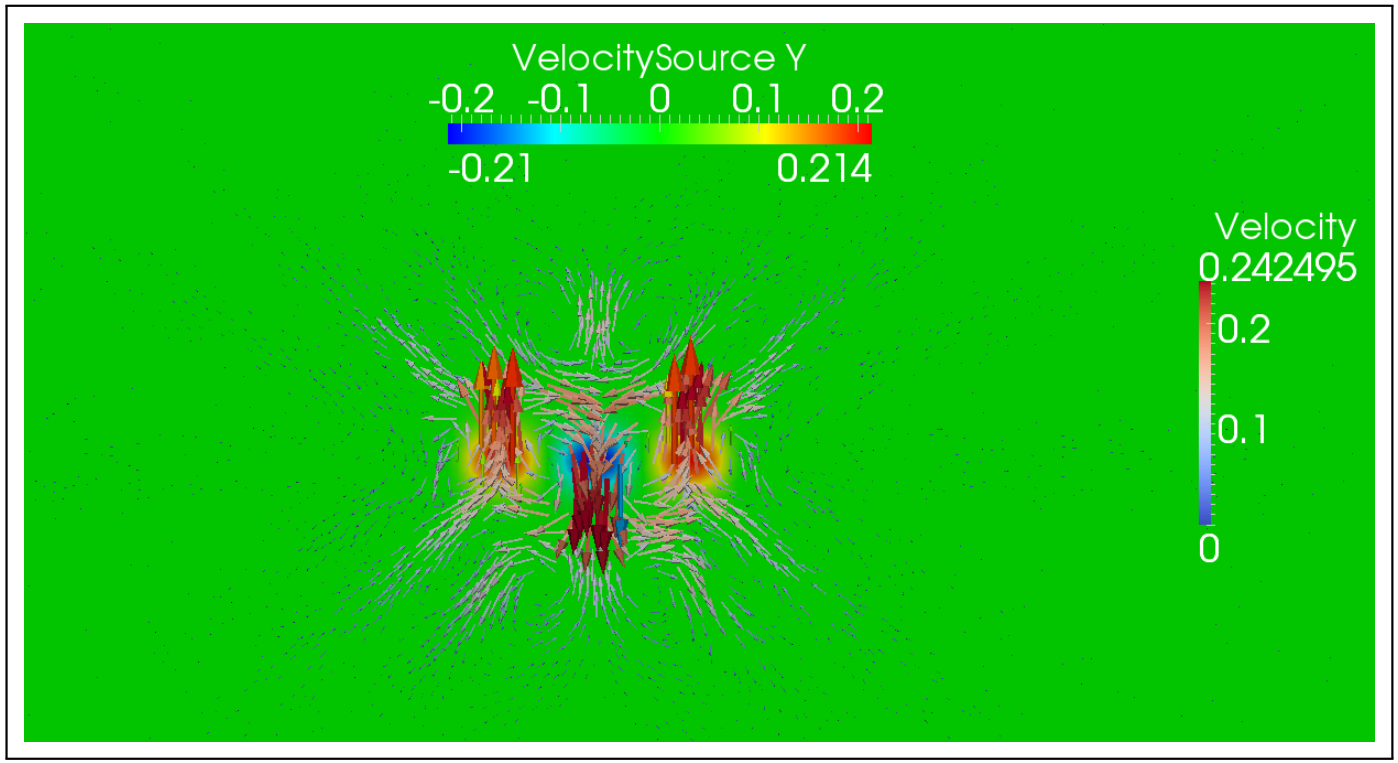
$$\omega = 0.699$$
$$N = 1$$

Which yields to

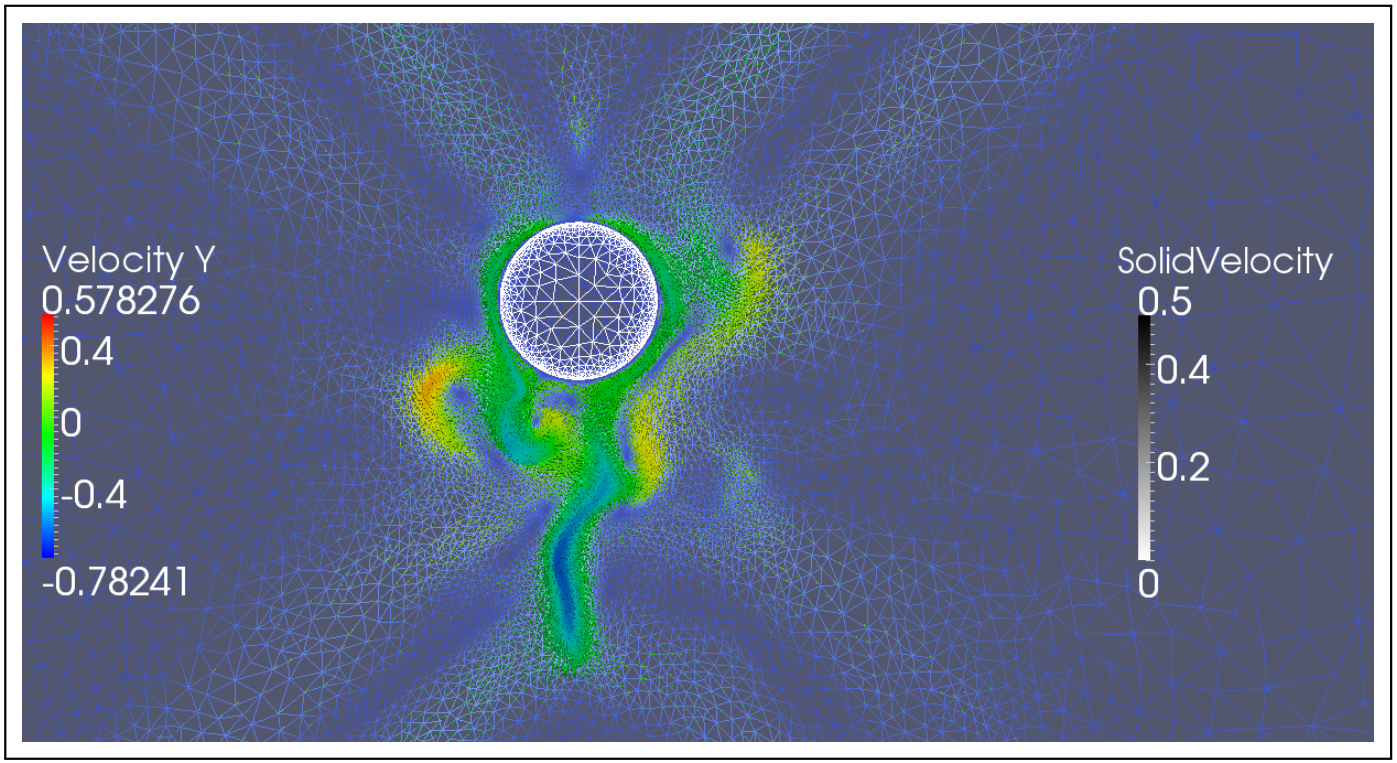
$$\alpha = 44.347$$

[3] Javam et al, *Numerical study of internal wave reflection from sloping boundaries*

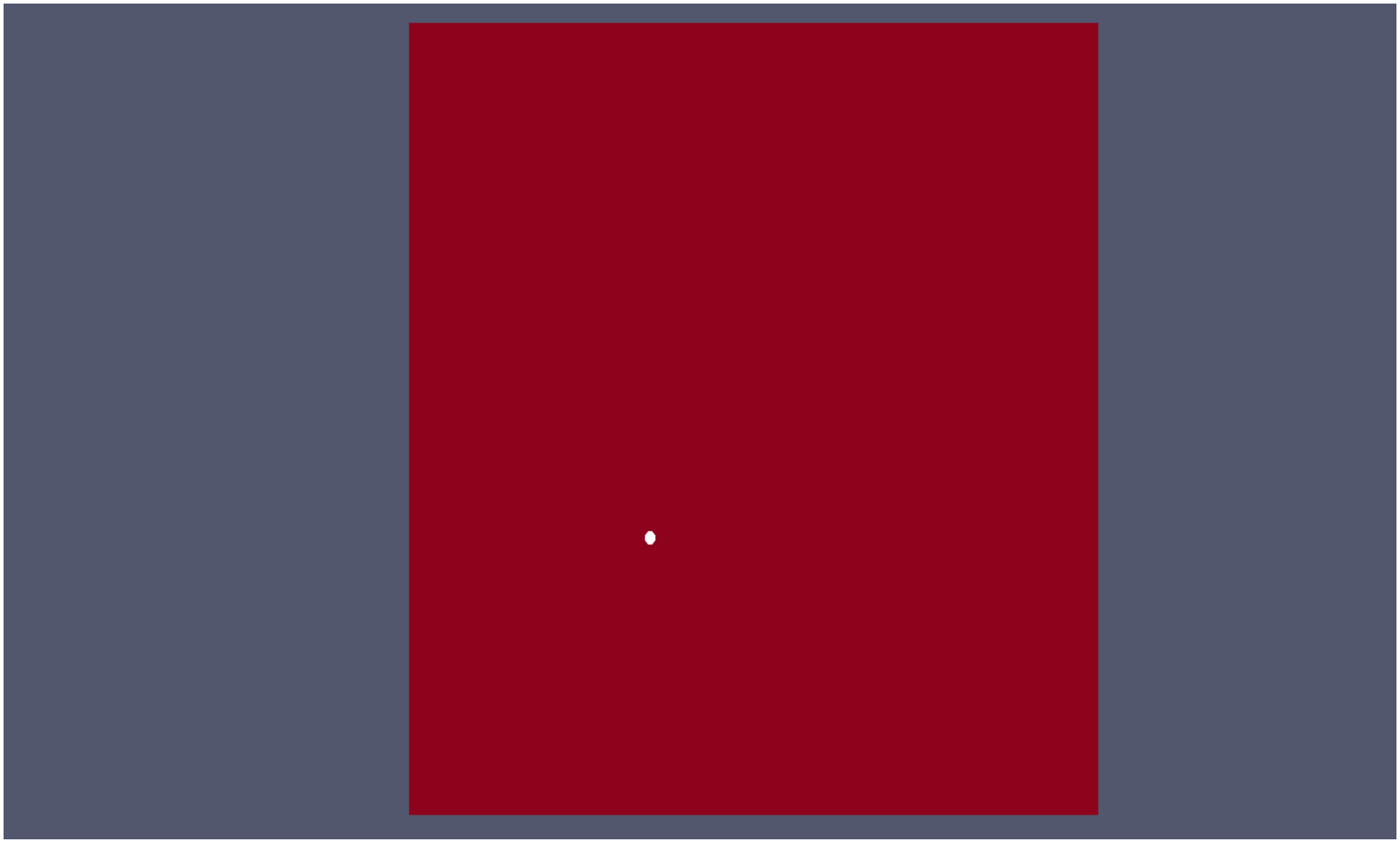
St Andrews Cross



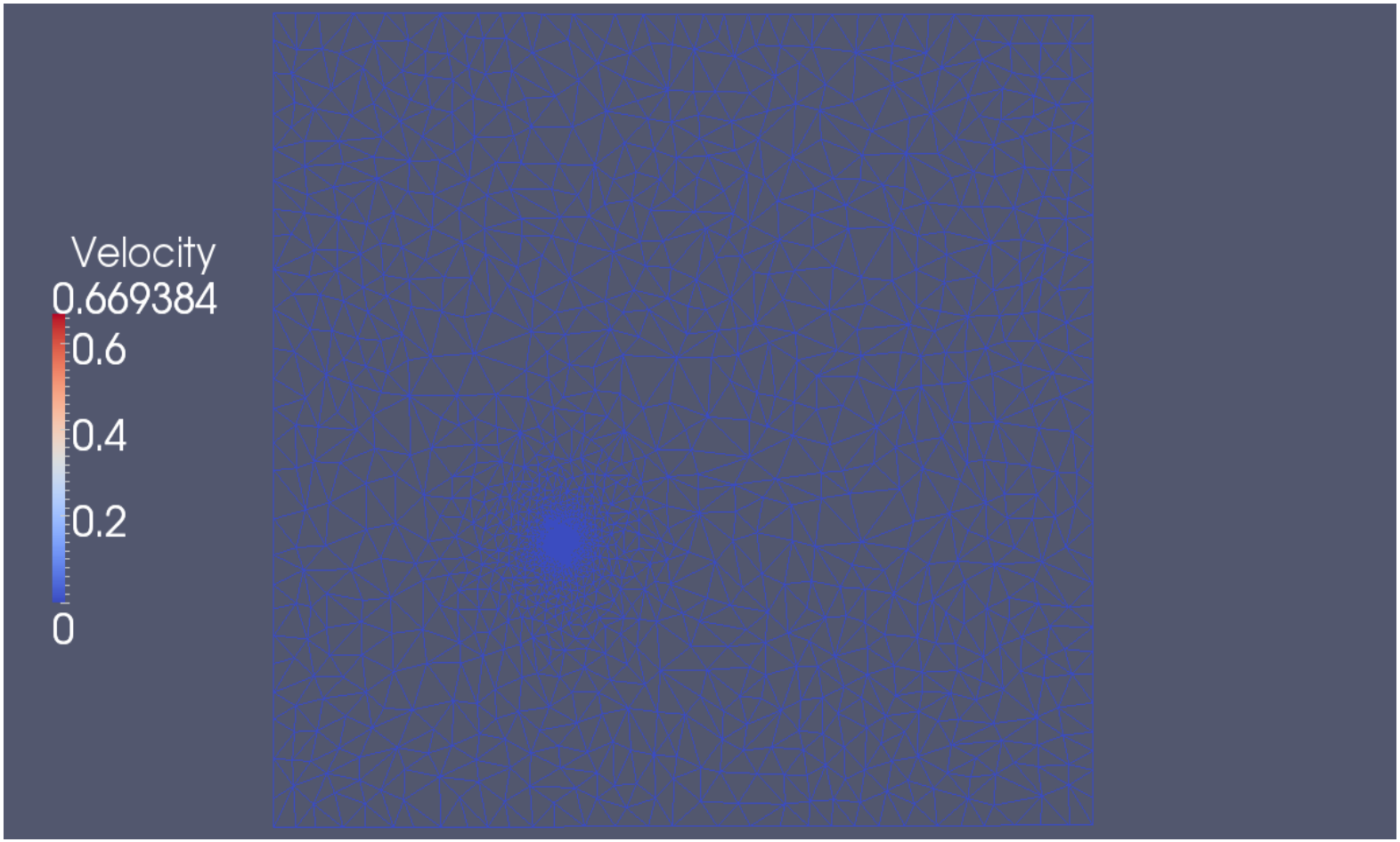
St Andrews Cross



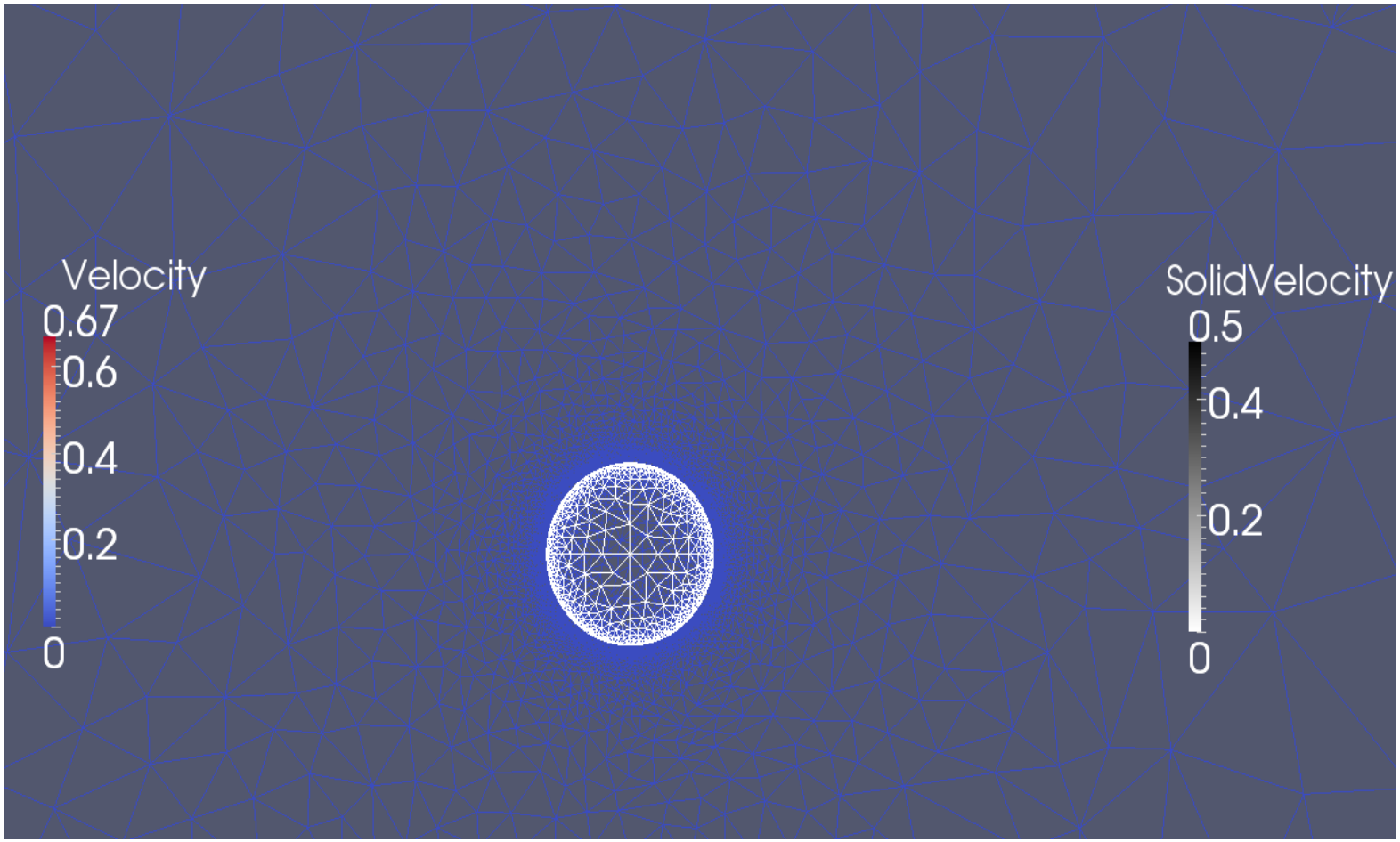
St Andrews Cross



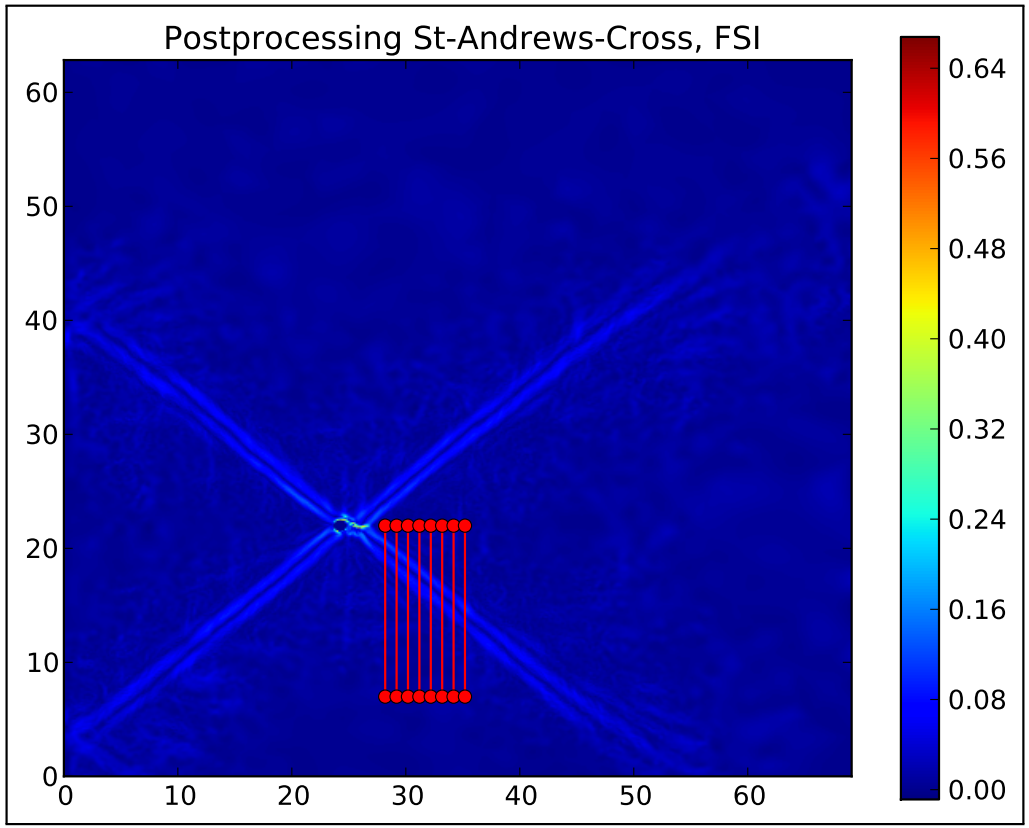
St Andrews Cross



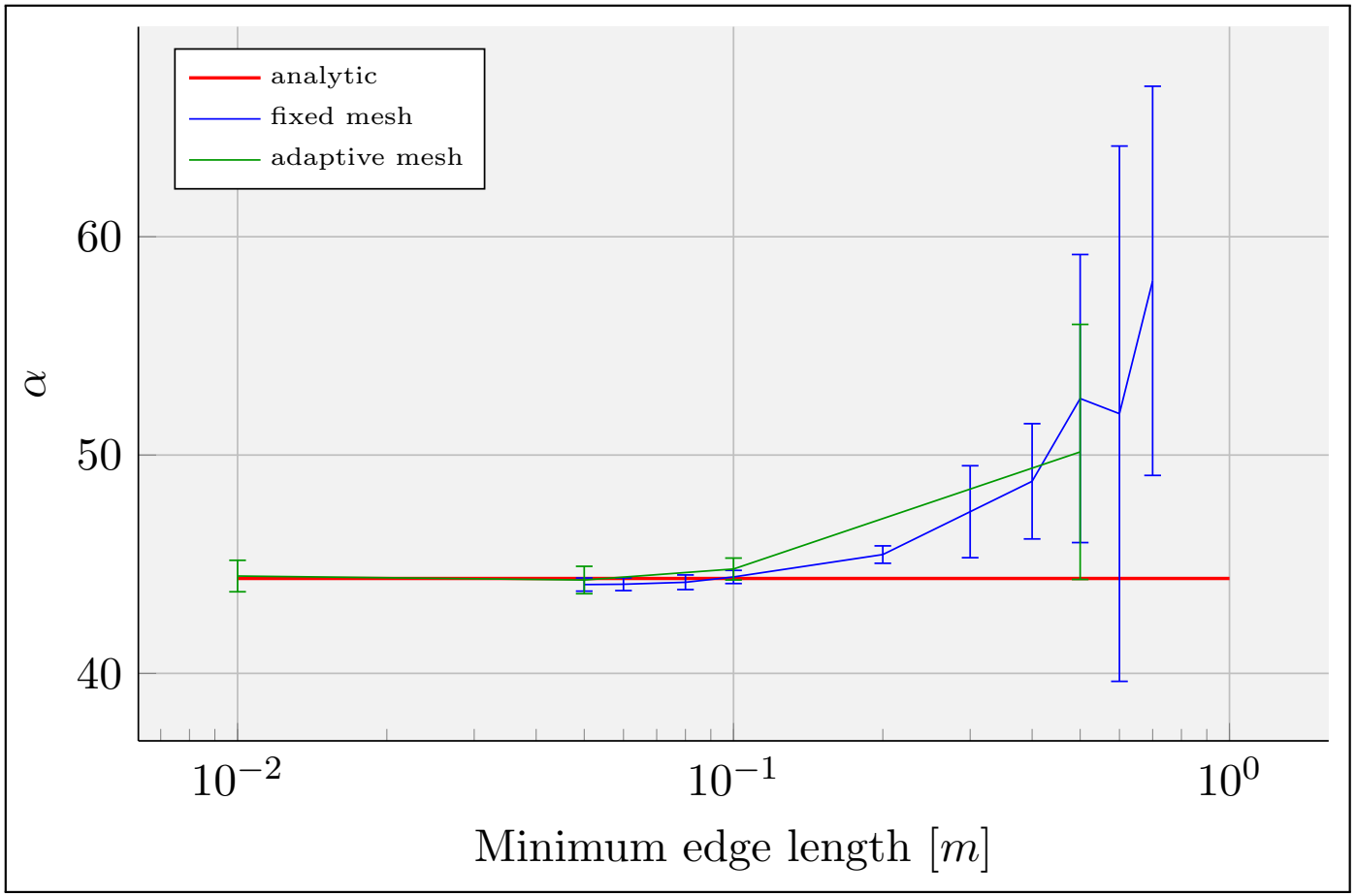
St Andrews Cross



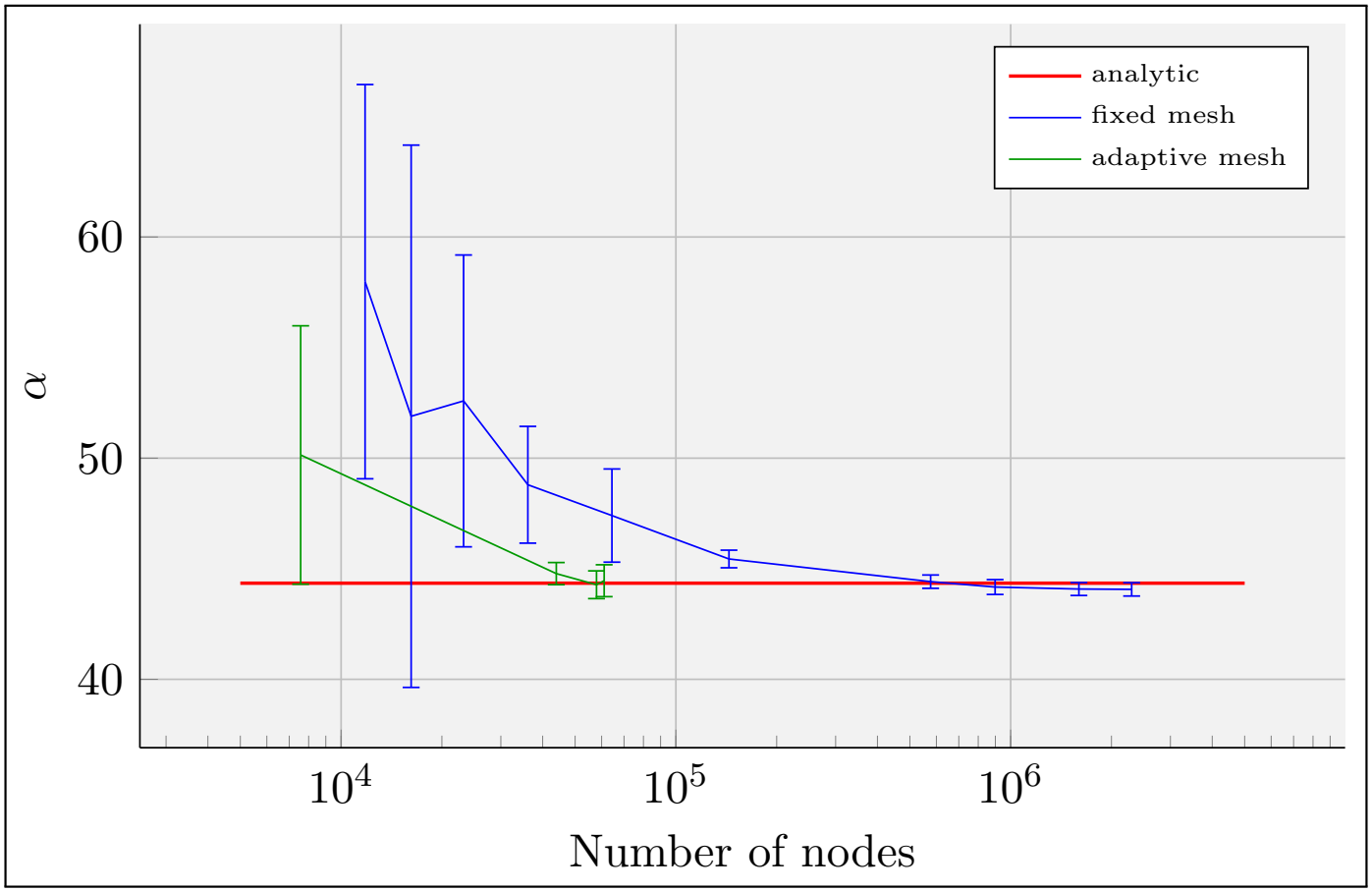
St Andrews Cross — Validation



St Andrews Cross — Validation



St Andrews Cross — Validation



St Andrews Cross — Validation

Min el	No Nodes	α_{min}	ε_{abs}	$\varepsilon_{\mu,\text{rel}}$	μ_{angle}	$\varepsilon_{\mu,\text{rel}}$	σ
0.05	2,298,494	44.343	−0.004	0.01 %	44.068	0.63 %	0.301
0.06	1,598,880	44.332	−0.015	0.033 %	44.083	0.596 %	0.291
0.08	898,531	44.336	−0.01	0.023 %	44.173	0.393 %	0.335
0.1	576,719	44.354	0.007	0.015 %	44.416	0.157 %	0.301
0.2	144,067	44.264	−0.083	0.187 %	45.442	2.469 %	0.399
0.3	64,416	44.676	0.329	0.742 %	47.404	6.895 %	2.107
0.4	36,131	44.948	0.601	1.355 %	48.798	10.036 %	2.64
0.5	23,220	44.934	0.587	1.324 %	52.588	18.584 %	6.598
0.6	16,191	44.425	0.079	0.177 %	51.895	17.02 %	12.26
0.7	11,793	42.844	−1.503	3.389 %	57.981	30.743 %	8.913

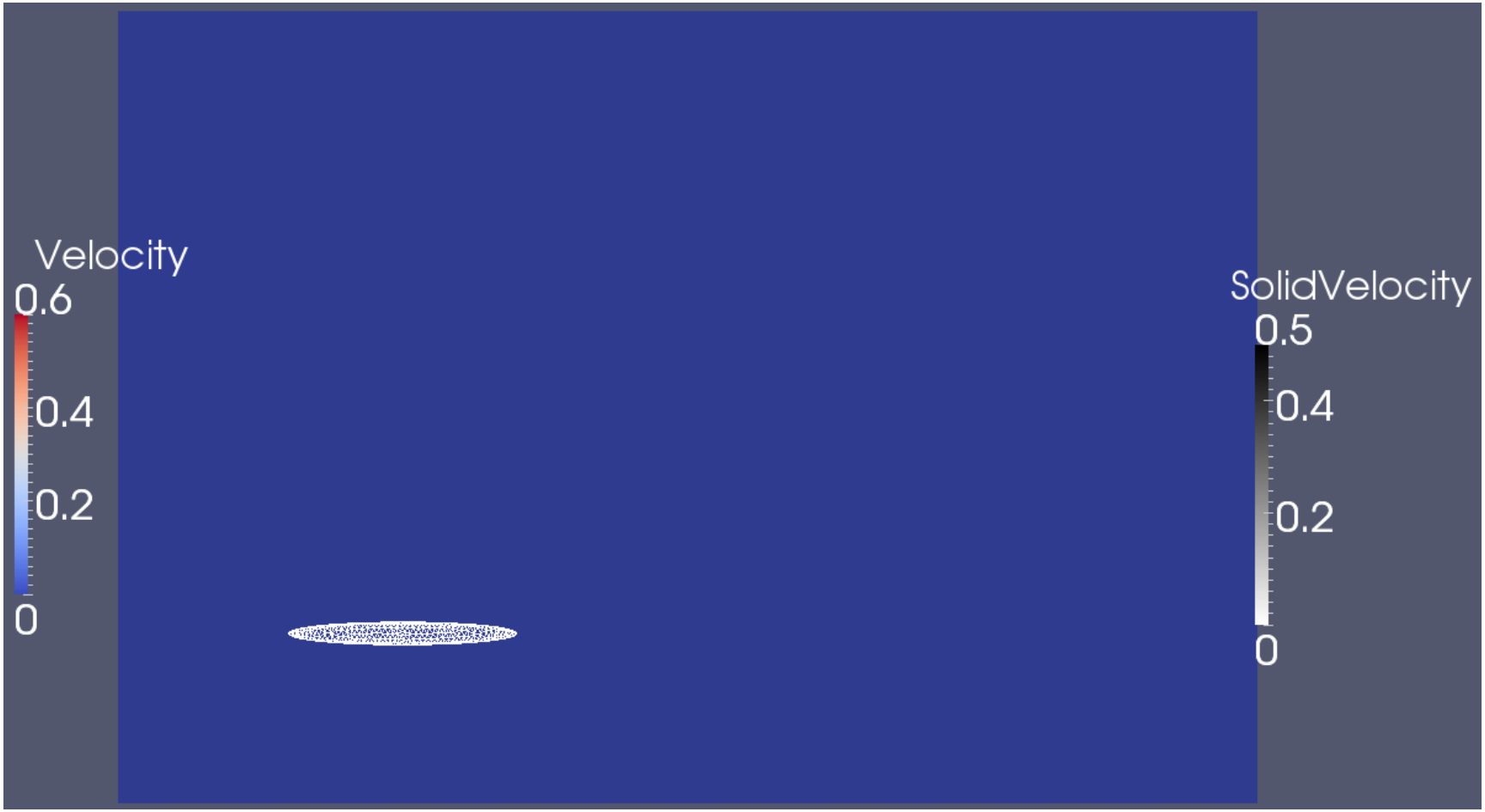
Table: St Andrews Cross validation, fixed mesh: $\omega = 0.699$, $N = 1$, $\alpha = 44.347$

St Andrews Cross — Validation

Min el	No Nodes	α_{min}	ε_{abs}	$\varepsilon_{\mu,\text{rel}}$	μ_{angle}	$\varepsilon_{\mu,\text{rel}}$	σ
0.01	61,070	44.364	0.017	0.039 %	44.459	0.252 %	0.721
0.05	57,932	44.316	−0.031	0.069 %	44.277	0.158 %	0.627
0.1	43,934	44.342	−0.005	0.01 %	44.78	0.977 %	0.501
0.5	7,573	44.352	0.006	0.013 %	50.143	13.071 %	5.841

Table: St Andrews Cross validation, adaptive mesh: $\omega = 0.699$, $N = 1$, $\alpha = 44.347$

Ellipse moving in stratified fluid



FSI in Diamond

- FSI Model can be found under *embedded_models*

FSI in Diamond

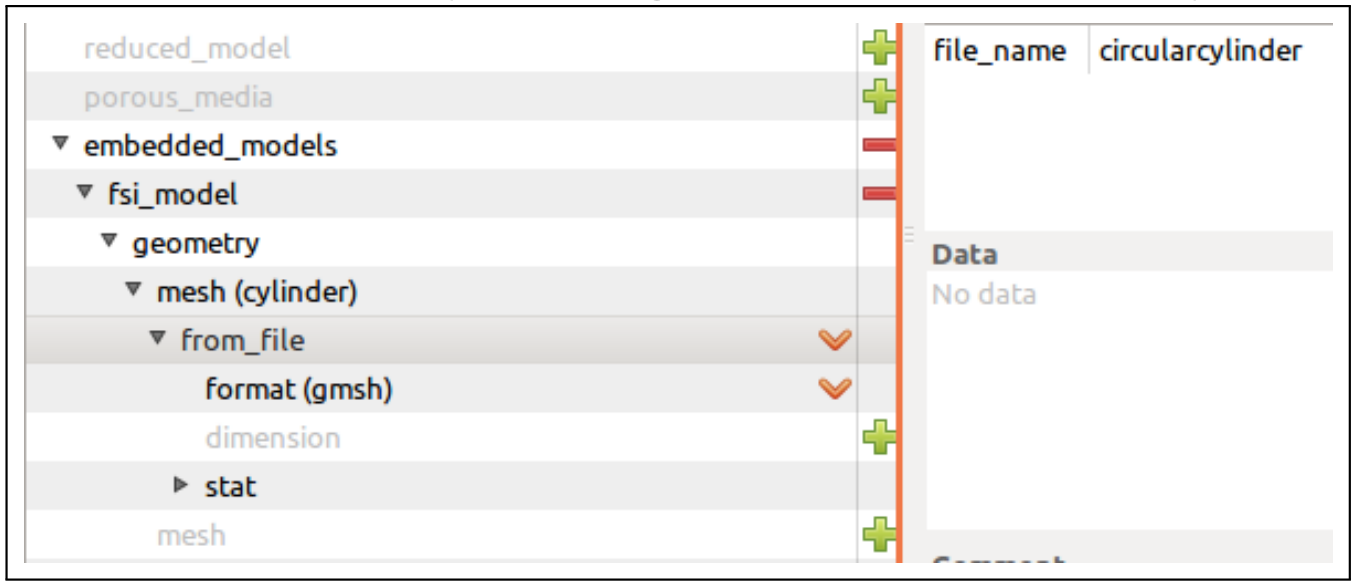
- FSI Model can be found under *embedded_models*
- Random name for mesh, but matters for solid phase

FSI in Diamond

- FSI Model can be found under *embedded_models*
- Random name for mesh, but matters for solid phase
- Multiple solids by adding another *mesh* entry

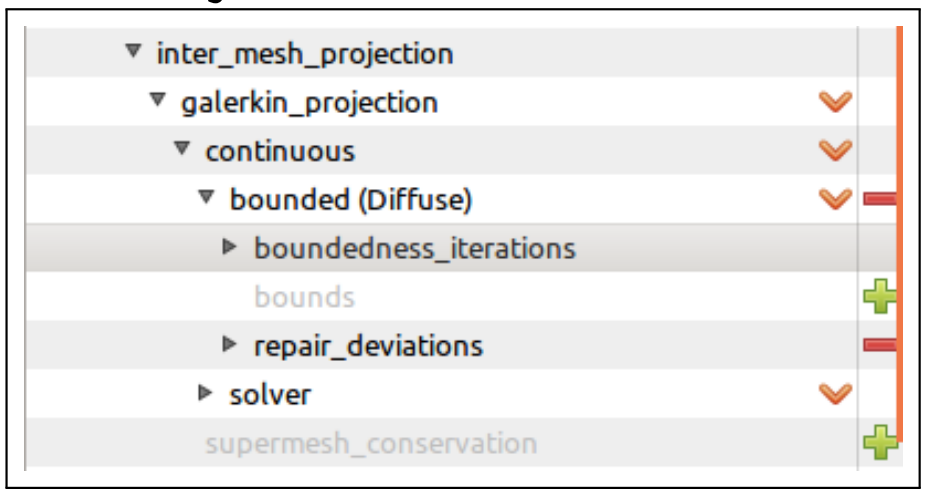
FSI in Diamond

- FSI Model can be found under *embedded_models*
- Random name for mesh, but matters for solid phase
- Multiple solids by adding another *mesh* entry



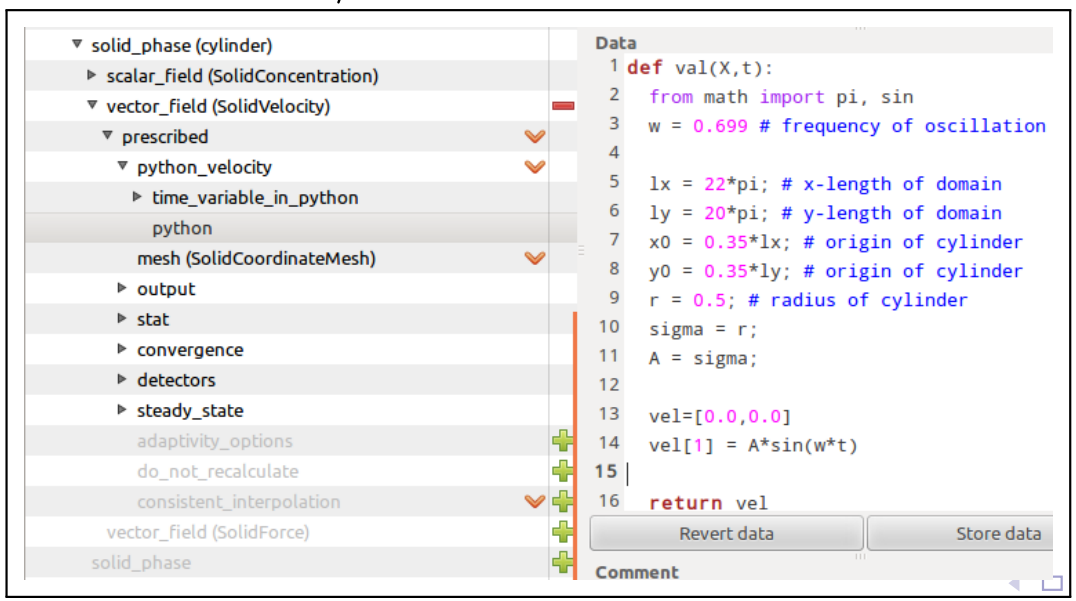
FSI in Diamond

- FSI Model can be found under *embedded_models*
- Random name for mesh, but matters for solid phase
- Multiple solids by adding another *mesh* entry
- FSI Projection



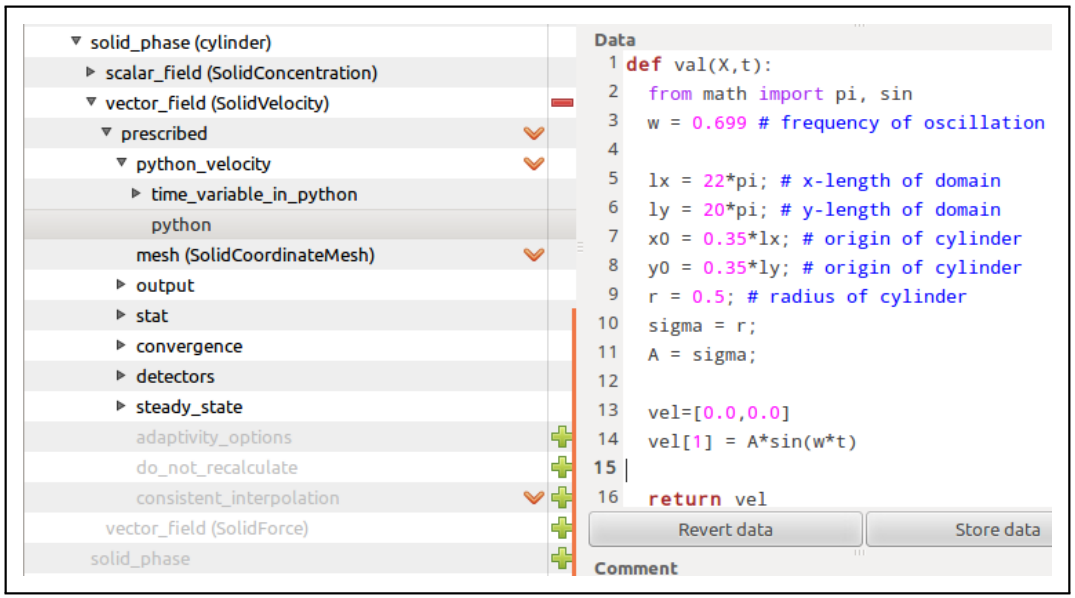
FSI in Diamond

- FSI Model can be found under *embedded_models*
- Random name for mesh, but matters for solid phase
- Multiple solids by adding another *mesh* entry
- FSI Projection
- Solid phases/fields



Prescribed Velocity

To give a solid mesh a prescribed velocity, we have to use the Python function under the corresponding solid phase in diamond. Here: Oscillating cylinder



Prescribed Velocity

Ellipse moving through a fluid, with positive and negative acceleration phases:

Data

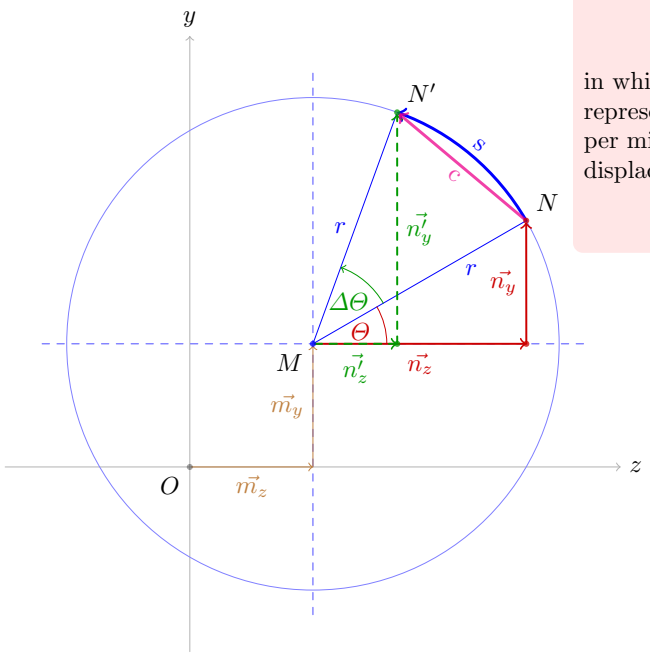
```python
def val(X,t):
    from math import cos, pi

    # parameters:
    maxvel = 0.5 #m/s maximal velocity
    rampuptime = 5.0 #s time to get to maximal velocity
    finishtime = 60.0 #time when vehicle should stand still

    # Constant maximum velocity:
    if (t > rampuptime and t < finishtime-rampuptime):
        v = maxvel
    # First part of acceleration:
    elif (t<=rampuptime/2.0):
        v = maxvel*(-cos(t*(pi/rampuptime))+1)*0.5
    # Second part of acceleration:
    elif (t>rampuptime/2.0 and t<=rampuptime):
        v = maxvel*(-cos(t*(pi/rampuptime)))*0.5 + maxvel*0.5
    # Deacceleration:
    elif (t >= finishtime-rampuptime and t <= finishtime):
            v = maxvel*(cos((finishtime-rampuptime-t)*(pi/rampuptime)))*0.5 + maxvel*0.5
    else: # no movement:
        v = 0.0

    # Return computed velocity:
    velocity = [v, 0.0] # vehicle is moving in x-direction only
    return velocity
```

Prescribed Velocity — Rotations



The known Cartesian coordinates of the points M, and N are

$$M = \begin{pmatrix} m_y \\ m_z \end{pmatrix}, \; N = \begin{pmatrix} n_1 \\ n_2 \end{pmatrix} = \begin{pmatrix} m_y + n_y \\ m_z + n_z \end{pmatrix}$$

in which M is the center of rotation and N a node of the mesh representing the turbine blades. Given a number of rotations per minute, e.g. rpm $= 10\,\mathrm{min}^{-1}$, the angular velocity ω and displacement $\Delta\Theta$ are computed by

$$\omega = 2\pi \cdot \mathrm{rpm}, \quad \Delta\Theta = \omega \Delta t$$

Within Δt seconds, a node N moves with an angular velocity ω along a circular arc s to it's position in N'. To compute the coordinates of N', we first compute the radius r and current angle Θ in the polar coordinate system.

$$r = \sqrt{{n_y}^2 + {n_z}^2}, \quad \Theta = \arcsin\left(\frac{n_y}{r}\right).$$

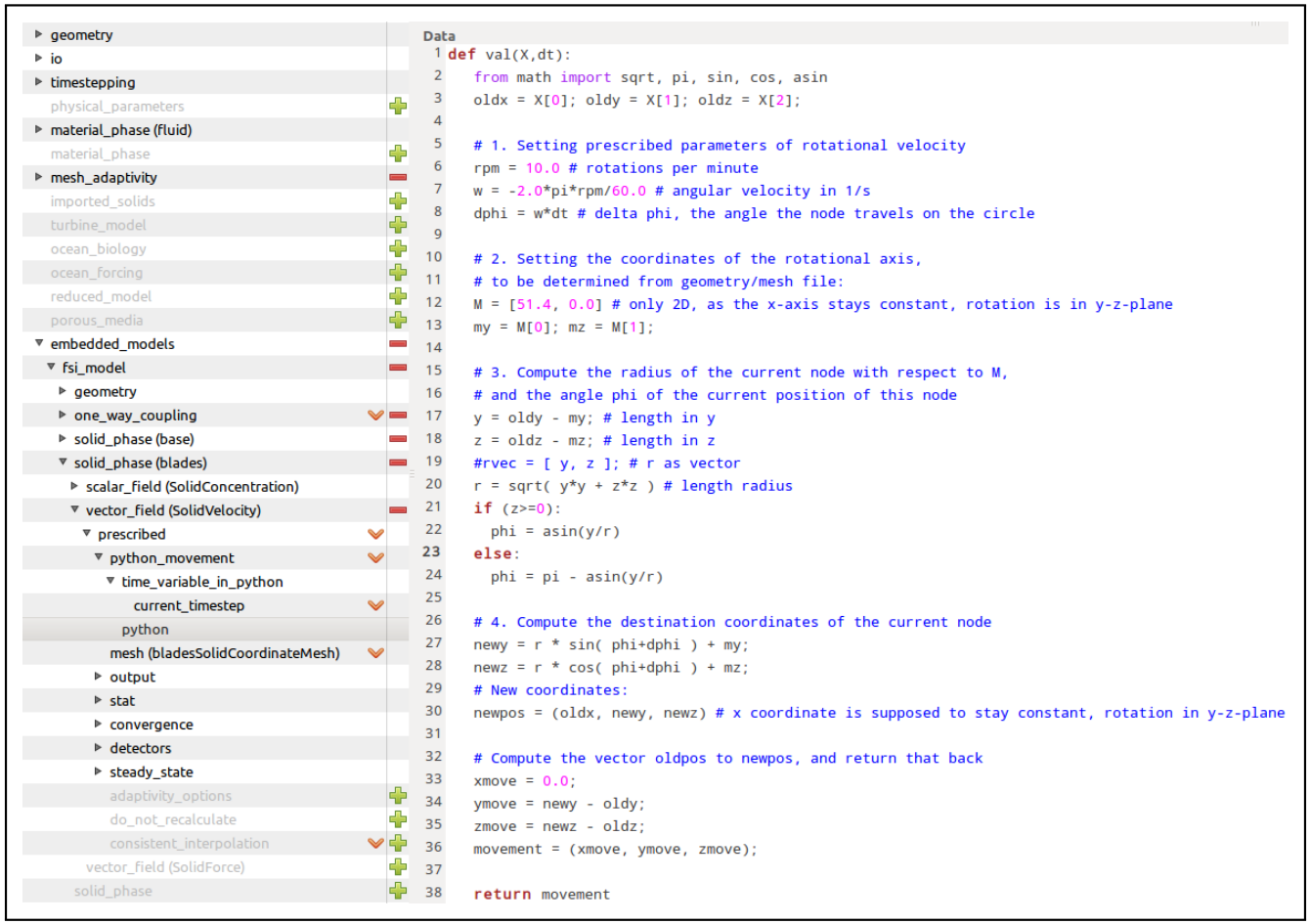
Via simple trigonometrical functions we can then derive the coordinates of the destination N'.

$$n'_y = r \cdot \sin\left(\Theta + \Delta\Theta\right), \quad n'_z = r \cdot \cos\left(\Theta + \Delta\Theta\right)$$

Thus the coordinates of N' in the Cartesian coordinate system are

$$N' = \begin{pmatrix} n'_1 \\ n'_2 \end{pmatrix} = \begin{pmatrix} m_y + n'_y \\ m_z + n'_z \end{pmatrix}$$

Prescribed Velocity — Rotations



FSI Output

- Solid vtu files (serial)

  Fluid: *simbasename*_i.(p)vtu[4]

  Solid: *simbasename*_solid_*solidmeshname*_i.vtu[5]

[4] i corresponds to the vtu dump number

[5] solidmeshname: name set in the flml for a solid mesh file

FSI Output

- Solid vtu files (serial)

  Fluid: *simbasename*_i.(p)vtu[4]

  Solid: *simbasename*_solid_*solidmeshname*_i.vtu[5]

- Solid checkpoints (serial)

  Fluid: *simbasename*_CoordinateMesh_i_checkpoint_j.msh[6]
  Solid:

  *simbasename*_solid_*solidmeshname*SolidCoordinateMesh_i_checkpoint.msh

[4] i corresponds to the vtu dump number
[5] solidmeshname: name set in the flml for a solid mesh file
[6] In a parallel run j is the corresponding MPI process number

FSI Output

- Solid vtu files (serial)

  Fluid: *simbasename*_i.(p)vtu[4]

  Solid: *simbasename*_solid_*solidmeshname*_i.vtu[5]

- Solid checkpoints (serial)

  Fluid: *simbasename*_CoordinateMesh_i_checkpoint_j.msh[6]
  Solid:

  *simbasename*_solid_*solidmeshname*SolidCoordinateMesh_i_checkpoint.msh

- Solid stat file (will be added in future)

[4] i corresponds to the vtu dump number
[5] solidmeshname: name set in the flml for a solid mesh file
[6] In a parallel run j is the corresponding MPI process number