

Fluid-Solid Interaction Modelling in Fluidity

Frank Milthaler

March 6, 2013

Outline

Motivation

How to get it

Governing Equations

Projection

Mesh Adaptivity and FSI

Applications

Diamond Interface — FSI

FSI Output

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

Momentum Equation and Continuity Equation

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

$$\nabla \cdot u = 0 \quad (2)$$

with

$$F_s = \sigma_f (u_s - u) \quad (3)$$

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

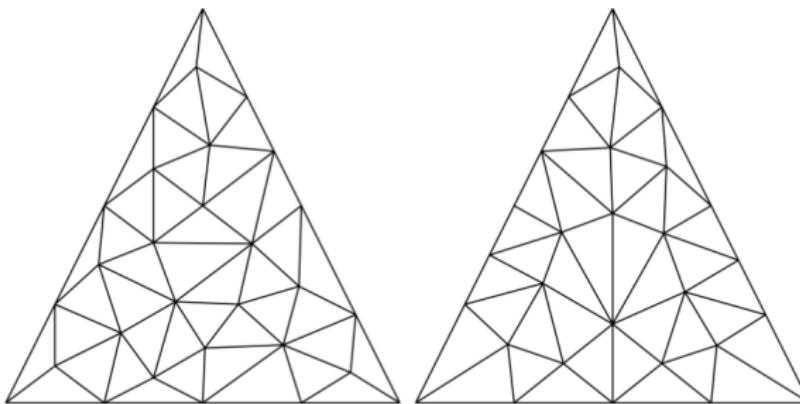
Projection

- ▶ Galerkin projection via supermeshing¹

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

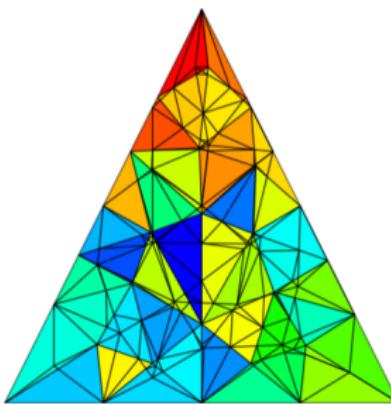
Donor and target mesh of same domain²



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

Projection

Supremesh with the elements of the target mesh being coloured²



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

Projection

- ▶ Galerkin projection via supermeshing¹
- ▶ 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 supermeshing¹
- ▶ 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 supermeshing¹
- ▶ 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:

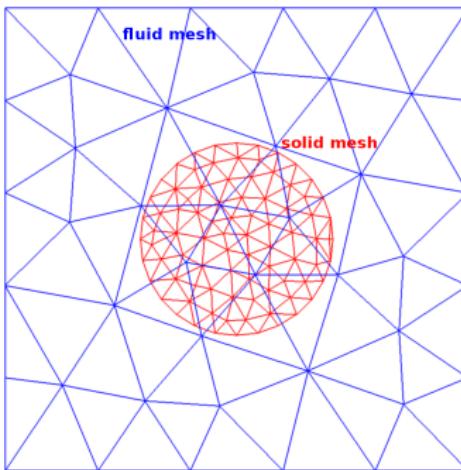
- ▶ τ_F, τ_S : Fluid and Solid volume meshes of the domains Ω_F, Ω_S , and elements K_F and K_S respectively
- ▶ Typically: $\Omega_F \supseteq \Omega_S$
- ▶ Thus, some $K_F \cap K_S$ are of measure zero
- ▶ Supermesh is constructed for any K_S for which $K_S \cap K_F$ of nonzero measure

Projection: FSI

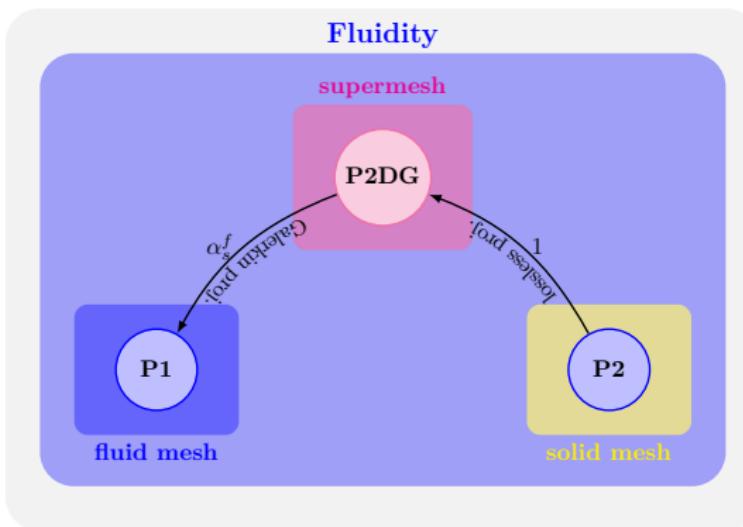
Usage of Galerkin projection via supermesh for FSI modelling:

- ▶ 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

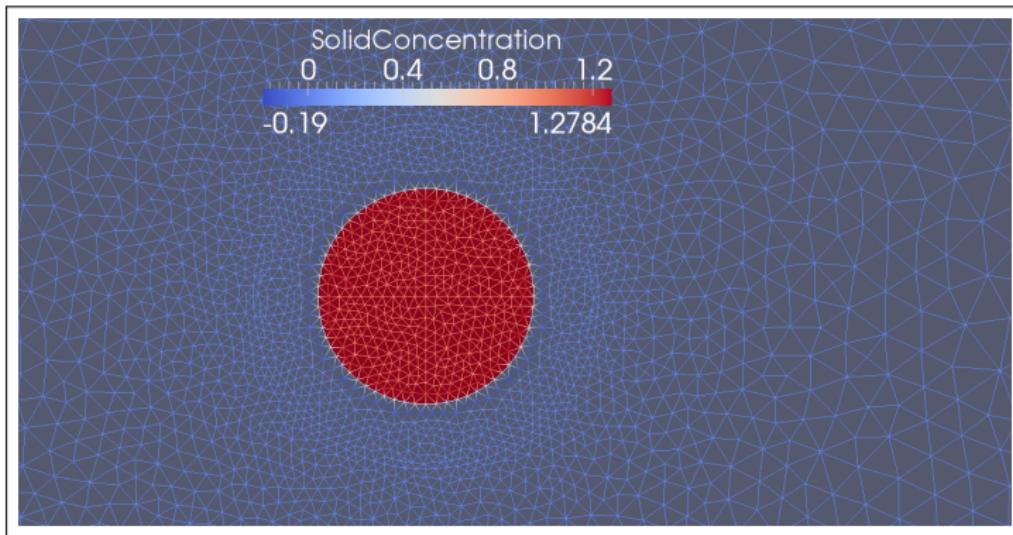


Legend:

- DG** Discontinuous
- P2** Quadratic mesh
- P1** Linear mesh

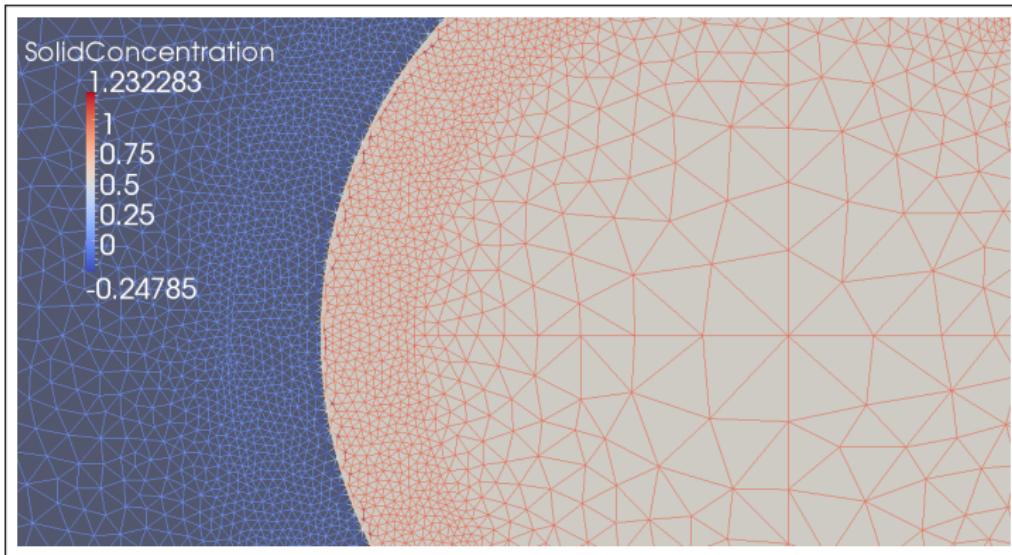
Projection: FSI

Unbounded projection:



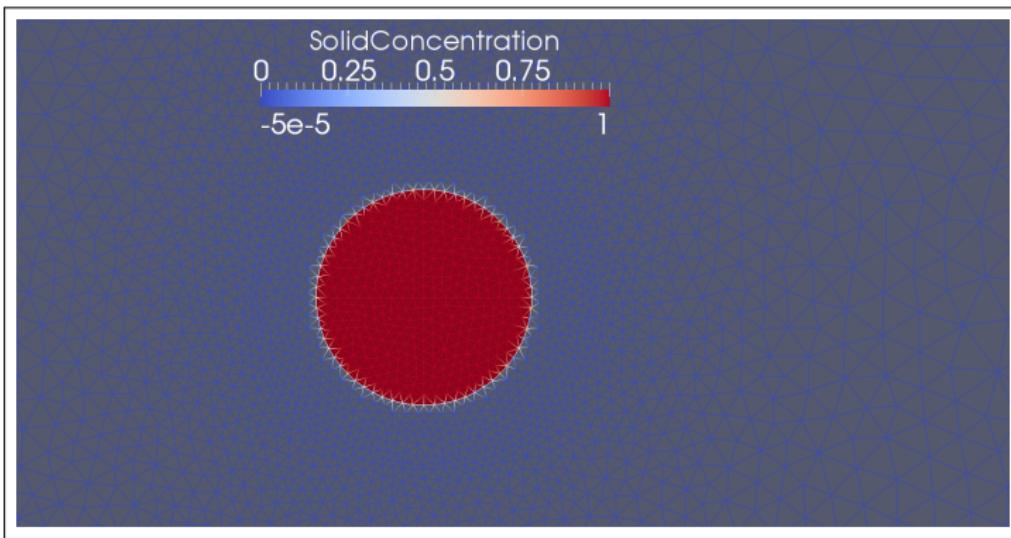
Projection: FSI

Unbounded projection:



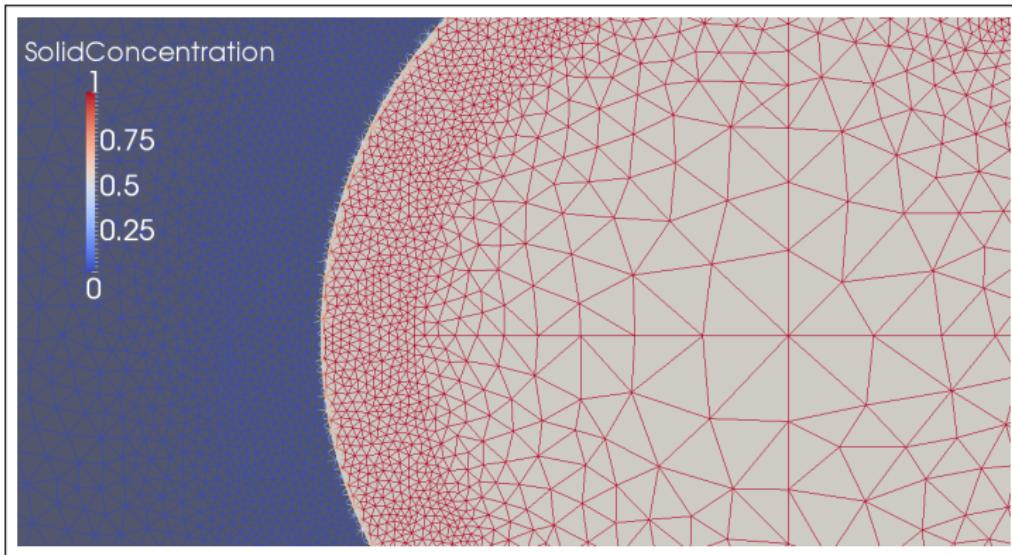
Projection: FSI

Bounded projection:



Projection: FSI

Bounded projection:



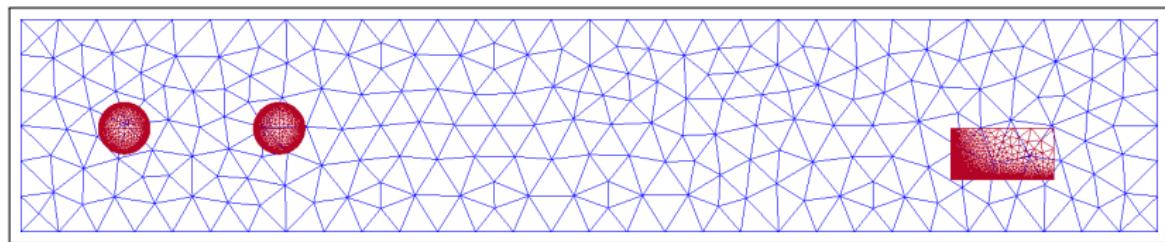
Mesh Adaptivity for FSI

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

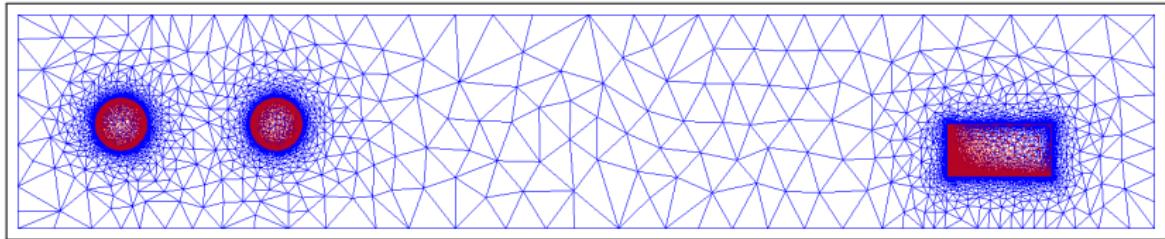
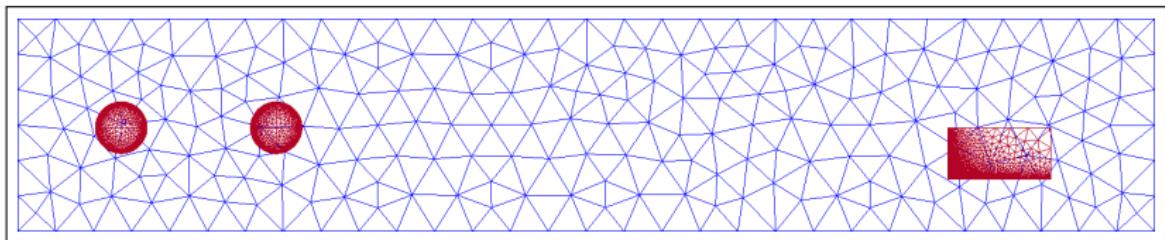
Mesh Adaptivity for FSI

Fluid mesh and Solid meshes



Mesh Adaptivity for FSI

Fluid mesh and Solid meshes



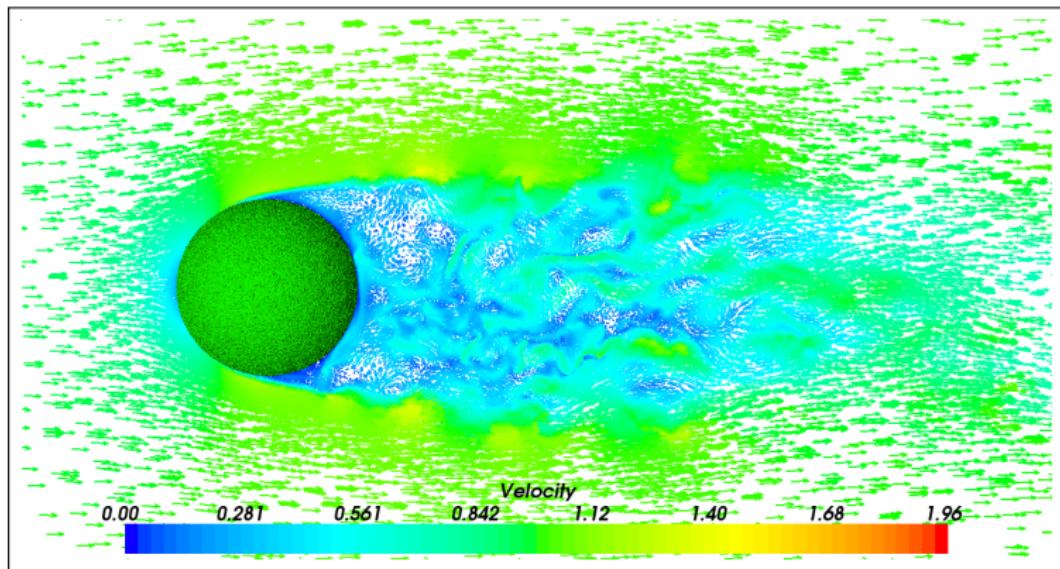
Applications

- ▶ 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

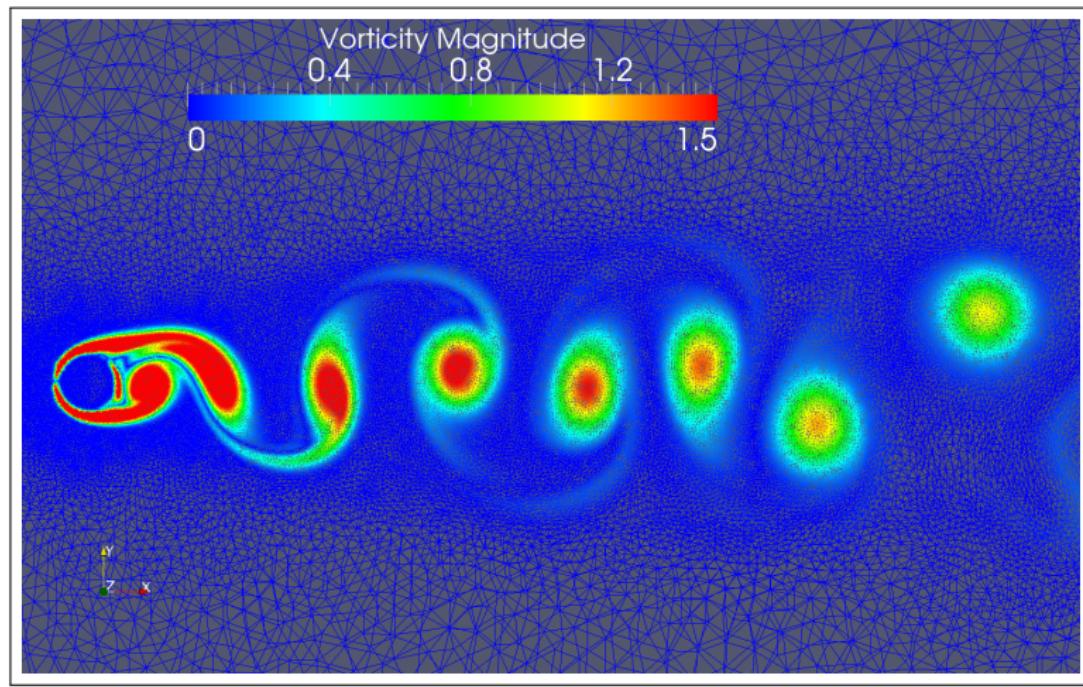
Applications

- ▶ 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

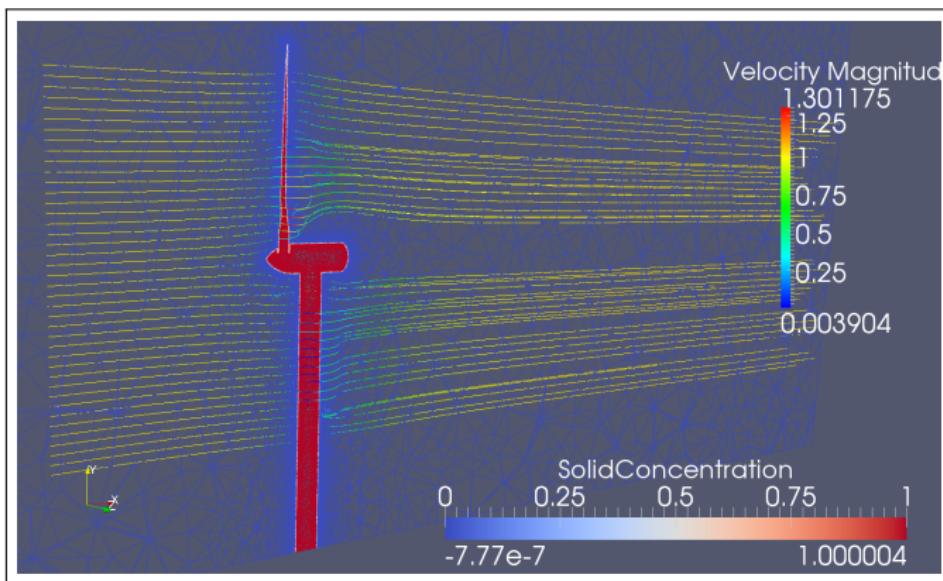
Fixed Solid



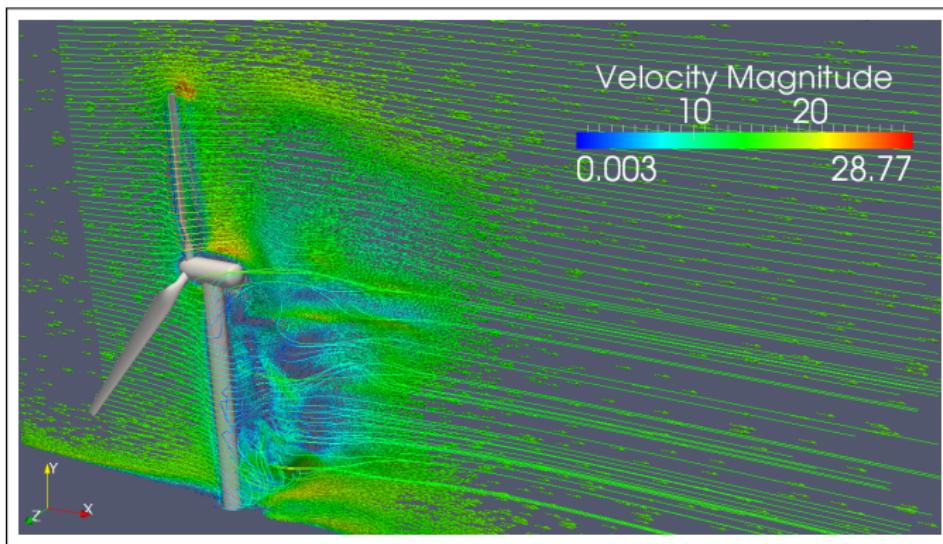
Fixed Solid



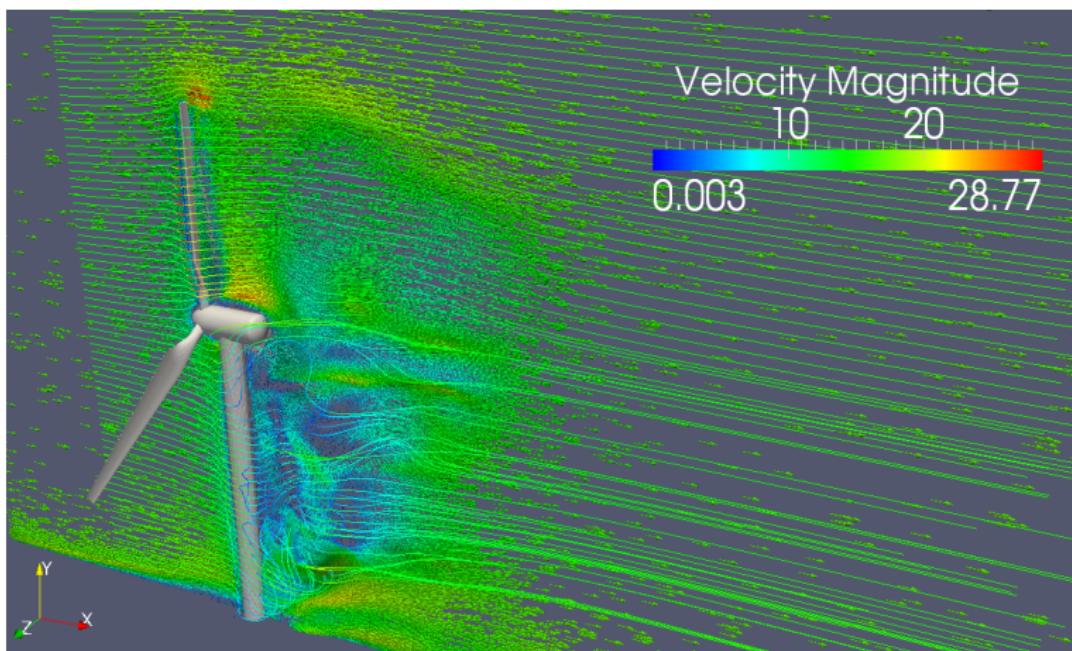
Rotating Wind Turbine



Rotating Wind Turbine



Rotating Wind Turbine



St Andrews Cross

- ▶ Momentum Source³
- ▶ Angle of internal wave propagation:

$$\sin \alpha = \frac{\omega}{N} \quad (4)$$

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

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

St Andrews Cross

- ▶ Momentum Source³
- ▶ FSI model
- ▶ Both with:

$$\omega = 0.699$$

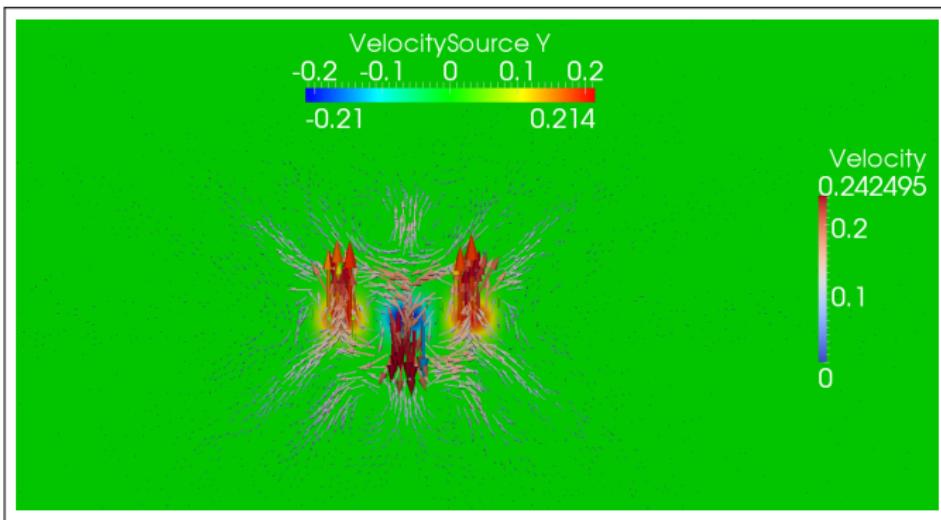
$$N = 1$$

Which yields to

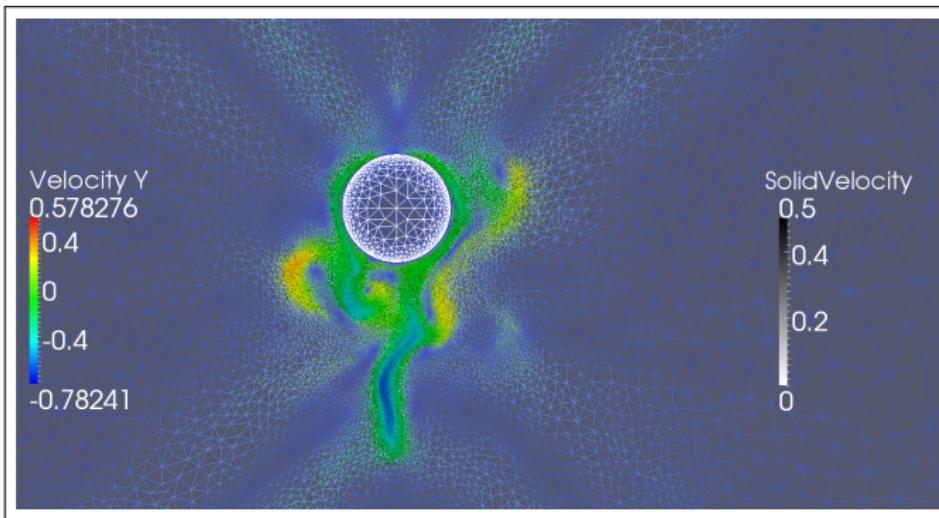
$$\alpha = 44.347$$

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

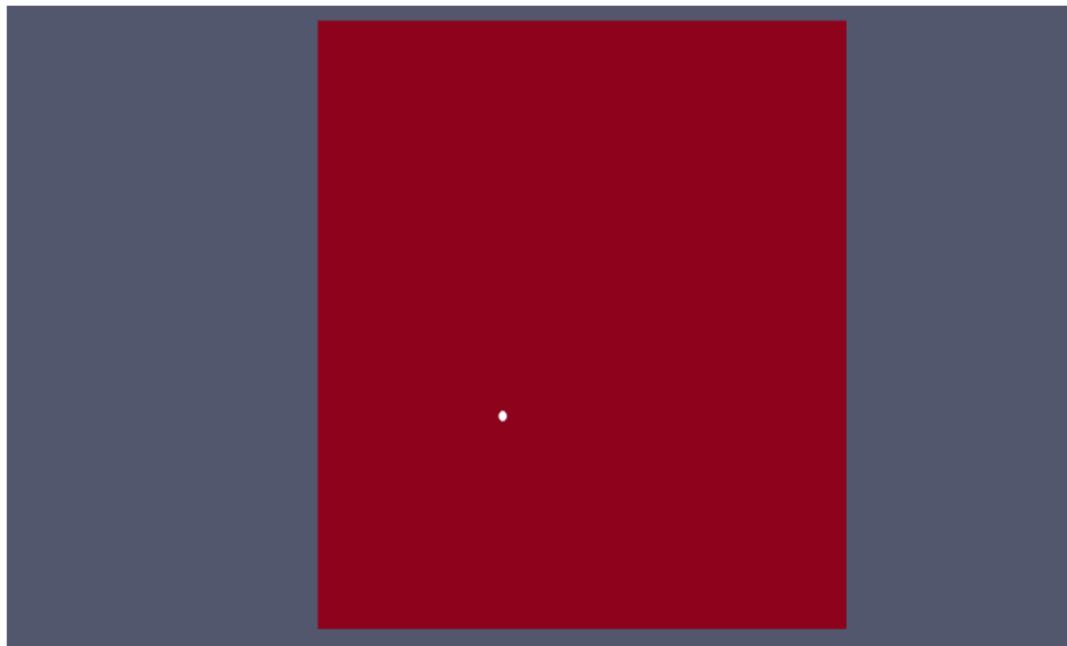
St Andrews Cross



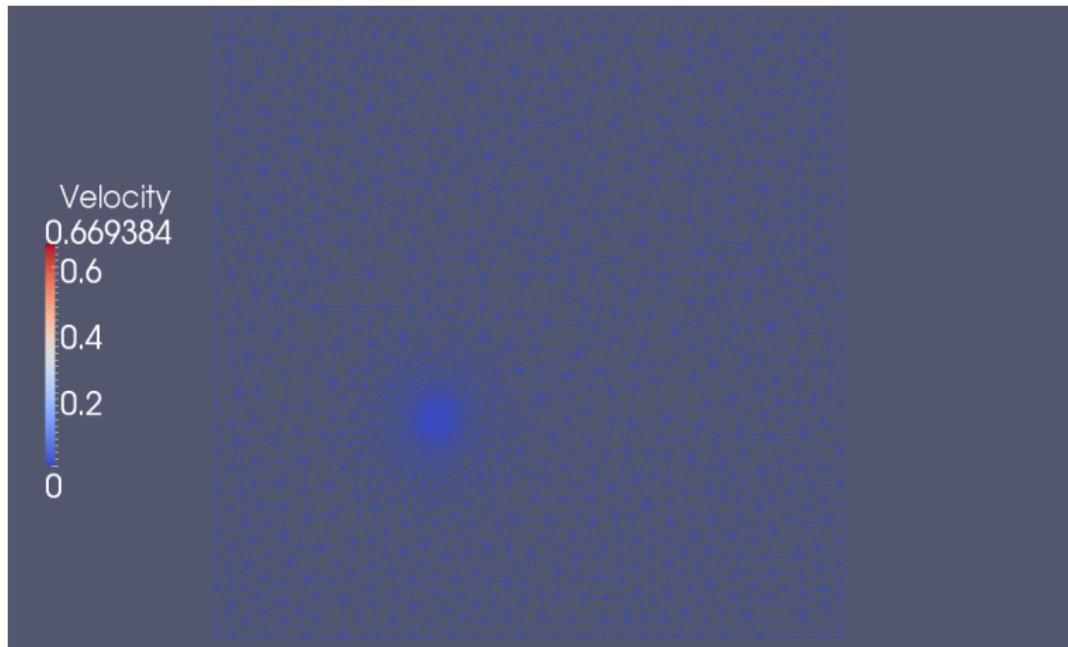
St Andrews Cross



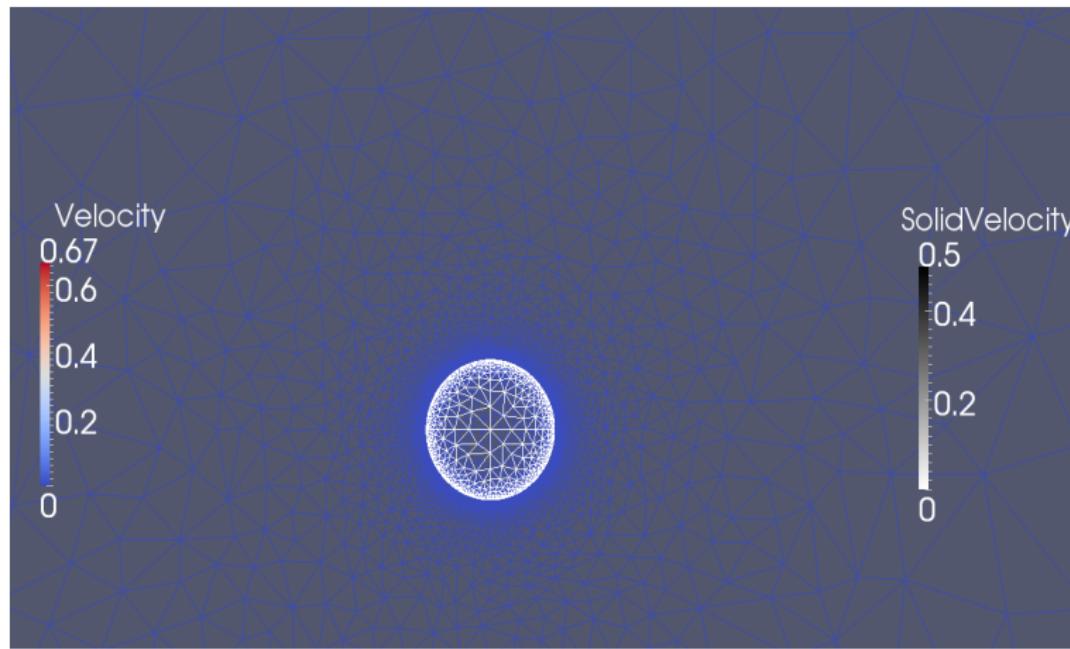
St Andrews Cross



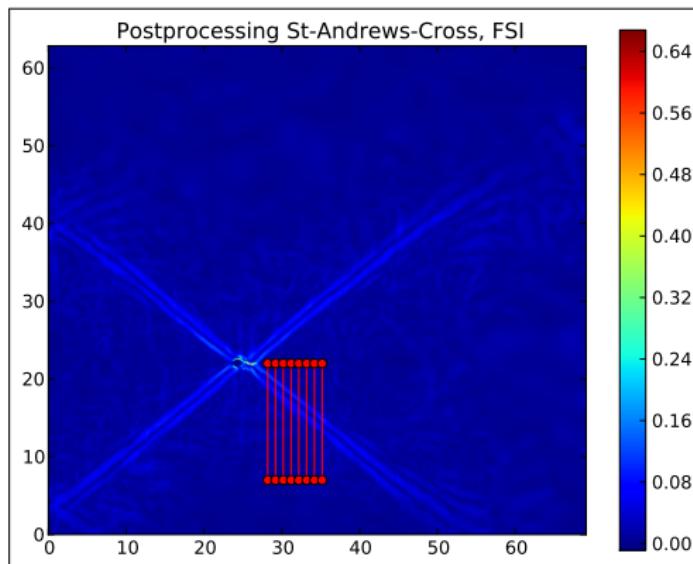
St Andrews Cross



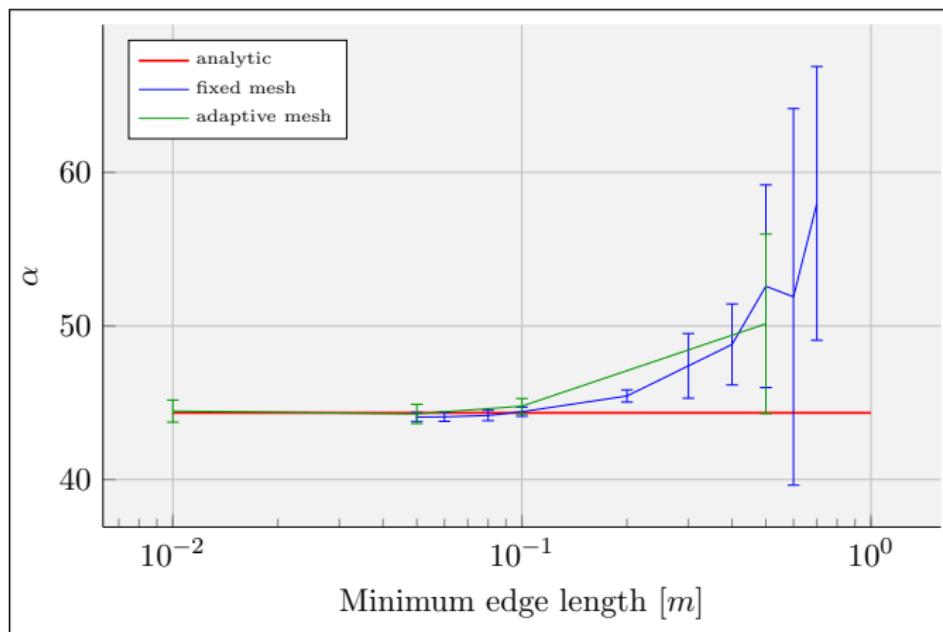
St Andrews Cross



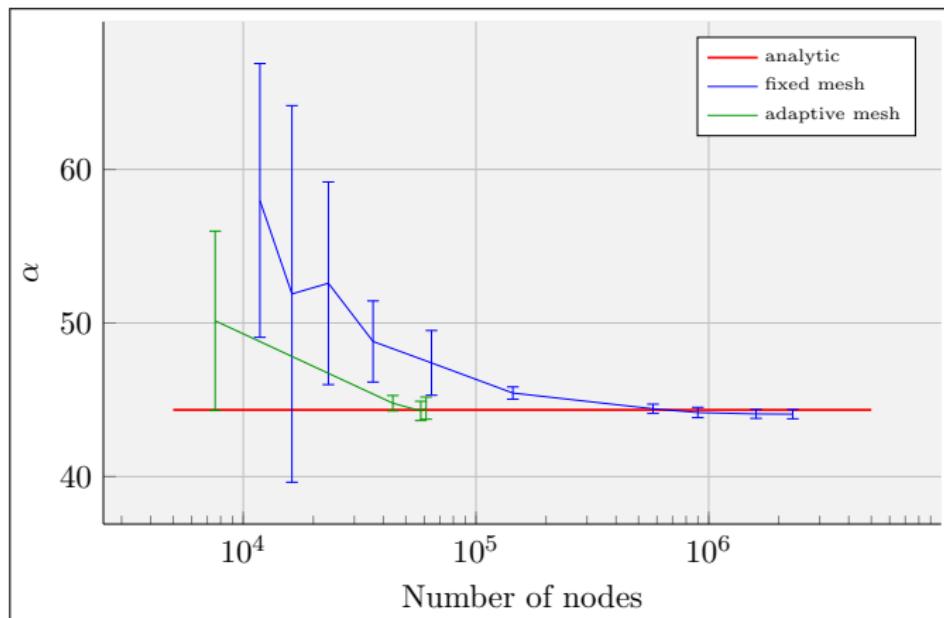
St Andrews Cross — Validation



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

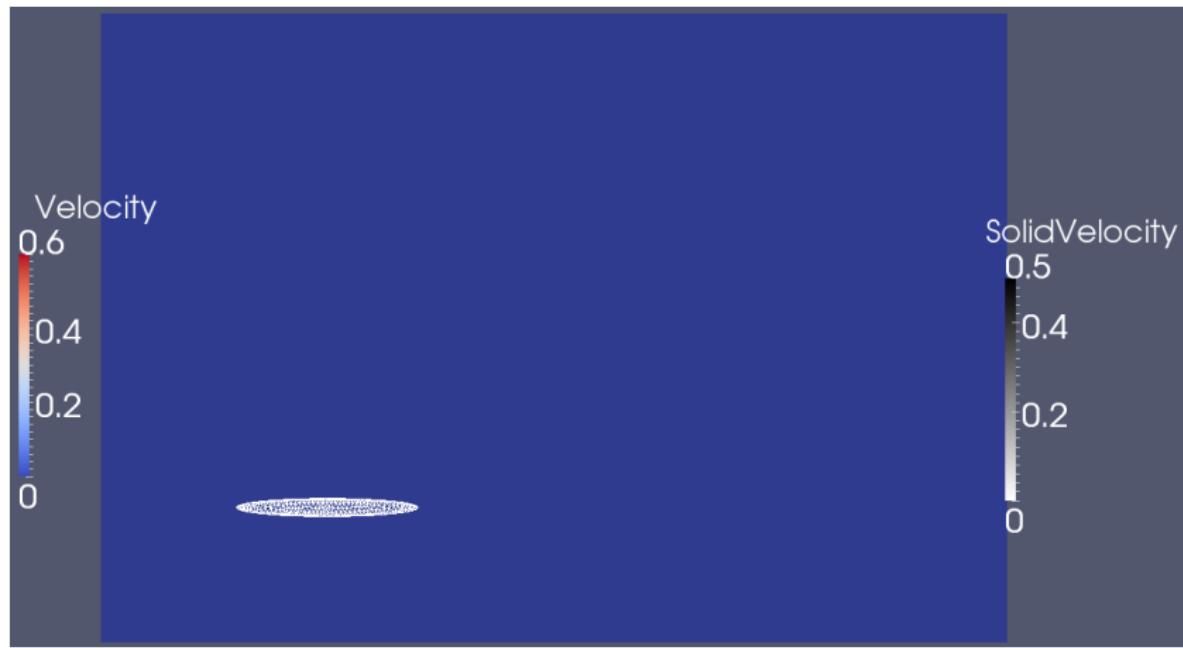
Table: St Andrews Cross validation, fixed mesh: $\omega = 0.699$, $N = 1$, $\alpha = 44.347$

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

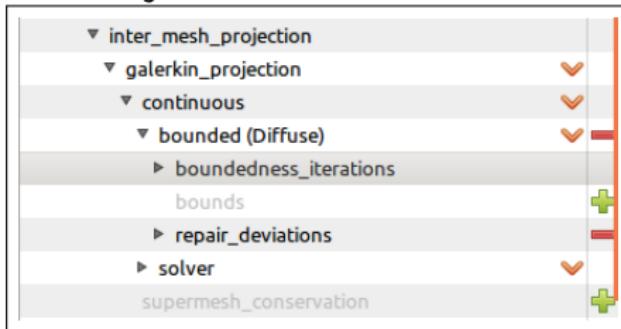
FSI in Diamond

- ▶ 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

The screenshot shows the Diamond software interface. On the left, there is a tree view of model components:

- solid_phase (cylinder)
 - scalar_field (SolidConcentration)
 - vector_field (SolidVelocity)
 - prescribed
 - python_velocity
 - time_variable_in_python
 - python
 - mesh (SolidCoordinateMesh)
 - output
 - stat
 - convergence
 - detectors
 - steady_state
 - adaptivity_options
 - do_not_recalculate
 - consistent_interpolation
 - vector_field (SolidForce)
 - solid_phase

```
Data
1 def val(X,t):
2     from math import pi, sin
3     w = 0.699 # frequency of oscillation
4
5     lx = 22*pi; # x-length of domain
6     ly = 20*pi; # y-length of domain
7     x0 = 0.35*lx; # origin of cylinder
8     y0 = 0.35*ly; # origin of cylinder
9     r = 0.5; # radius of cylinder
10    sigma = r;
11    A = sigma;
12
13    vel=[0.0,0.0]
14    vel[1] = A*sin(w*t)
15
16    return vel
```

Revert data

Store data

Comment

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

The screenshot shows the 'solid_phase (cylinder)' configuration panel in the Diamond software. On the left, a tree view lists various parameters: scalar_field (SolidConcentration), vector_field (SolidVelocity), prescribed, python_velocity, time_variable_in_python, python, mesh (SolidCoordinateMesh), output, stat, convergence, detectors, steady_state, adaptivity_options, do_not_recalculate, consistent_interpolation, vector_field (SolidForce), and solid_phase. The 'prescribed' and 'python_velocity' nodes are expanded. The 'python' node contains the following code:

```
def val(X,t):
    from math import pi, sin
    w = 0.699 # frequency of oscillation
    lx = 22*pi; # x-length of domain
    ly = 20*pi; # y-length of domain
    x0 = 0.35*lx; # origin of cylinder
    y0 = 0.35*ly; # origin of cylinder
    r = 0.5; # radius of cylinder
    sigma = r;
    A = sigma;
    vel=[0.0,0.0]
    vel[1] = A*sin(w*t)
    return vel
```

Below the code are 'Revert data' and 'Store data' buttons, and a 'Comment' field.

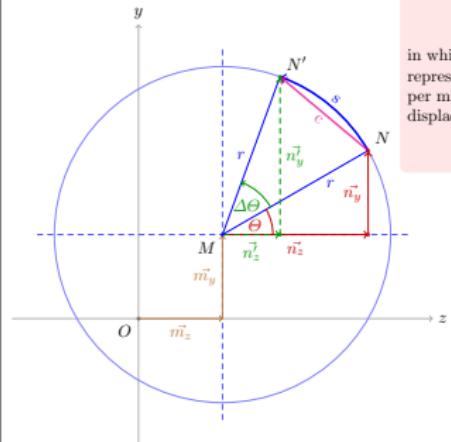


Prescribed Velocity

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

```
Data
1 def val(X,t):
2     from math import cos, pi
3
4     # parameters:
5     maxvel = 0.5 #m/s maximal velocity
6     rampertime = 5.0 #s time to get to maximal velocity
7     finishtime = 60.0 #time when vehicle should stand still
8
9     # Constant maximum velocity:
10    if (t > rampertime and t < finishtime-rampertime):
11        v = maxvel
12    # First part of acceleration:
13    elif (t<=rampertime/2.0):
14        v = maxvel*(-cos(t*(pi/rampertime))+1)*0.5
15    # Second part of acceleration:
16    elif ((rampertime/2.0 < t)&(t<=rampertime)):
17        v = maxvel*(-cos(t*(pi/rampertime)))*0.5 + maxvel*0.5
18    # Deacceleration:
19    elif (t >= finishtime-rampertime and t <= finishtime):
20        v = maxvel*(cos((finishtime-rampertime-t)*(pi/rampertime)))*0.5 + maxvel*0.5
21    else: # no movement:
22        v = 0.0
23
24    # Return computed velocity:
25    velocity = [v, 0.0] # vehicle is moving in x-direction only
26
27    return velocity
```

Prescribed Velocity — Rotations



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

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

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. $\text{rpm} = 10 \text{ min}^{-1}$, the angular velocity ω and displacement $\Delta\theta$ are computed by

$$\omega = 2\pi \cdot \text{rpm}, \quad \Delta\theta = \omega \Delta t$$

Within Δt seconds, a node N moves with an angular velocity ω along a circular arc s to its 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' .

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

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

```

geometry
io
timestepping
physical_parameters
material_phase (fluid)
material_phase
mesh_adaptivity
imported_solids
turbine_model
ocean_biology
ocean_forcing
reduced_model
porous_media
embedded_models
  fsi_model
    geometry
  one_way_coupling
  solid_phase (base)
  solid_phase (blades)
    scalar_field (SolidConcentration)
    vector_field (SolidVelocity)
      prescribed
        python_movement
          time_variable_in_python
            current_timestep
              python
            mesh (bladesSolidCoordinateMesh)
output
stat
convergence
detectors
steady_state
adaptive_options
do_not_recalculate
consistent_interpolation
vector_field (SolidForce)
solid_phase

```

```

Data
1 def val(X,dt):
2   from math import sqrt, pi, sin, cos, asin
3   oldx = X[0]; oldy = X[1]; oldz = X[2];
4
5   # 1. Setting prescribed parameters of rotational velocity
6   rpm = 10.0 # rotations per minute
7   w = -2.0*pi*rpm/60.0 # angular velocity in 1/s
8   dphi = w*dt # delta phi, the angle the node travels on the circle
9
10  # 2. Setting the coordinates of the rotational axis,
11  # to be determined from geometry/mesh file:
12  M = [51.4, 0.0] # only 2D, as the x-axis stays constant, rotation is in y-z-plane
13  my = M[0]; mz = M[1];
14
15  # 3. Compute the radius of the current node with respect to M,
16  # and the angle phi of the current position of this node
17  y = oldy - my; # length in y
18  z = oldz - mz; # length in z
19  rvec = [ y, z ]; # r as vector
20  r = sqrt( y*y + z*z ) # length radius
21  if (z>=0):
22    phi = asin(y/r)
23  else:
24    phi = pi - asin(y/r)
25
26  # 4. Compute the destination coordinates of the current node
27  newy = r * sin( phi+dphi ) + my;
28  newz = r * cos( phi+dphi ) + mz;
29
30  # New coordinates:
31  newpos = (oldx, newy, newz) # x coordinate is supposed to stay constant, rotation in y-z-plane
32
33  # Compute the vector oldpos to newpos, and return that back
34  xmove = 0.0;
35  ymove = newy - oldy;
36  zmove = newz - oldz;
37
38  movement = (xmove, ymove, zmove);

```



FSI Output

- ▶ Solid vtu files (serial)

Fluid: *simbasename_i.(p).vtu*⁴

Solid: *simbasename_solid_solidmeshname_i.vtu*⁵

⁴i corresponds to the vtu dump number

⁵solidmeshname: 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_solidmeshnameSolidCoordinateMesh_i_checkpoint.msh

⁴i corresponds to the vtu dump number

⁵solidmeshname: name set in the fml for a solid mesh file

⁶In 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_solidmeshnameSolidCoordinateMesh_i_checkpoint.msh

- ▶ Solid stat file (will be added in future)

⁴i corresponds to the vtu dump number

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

⁶In a parallel run j is the corresponding MPI process number