SPHinXsys

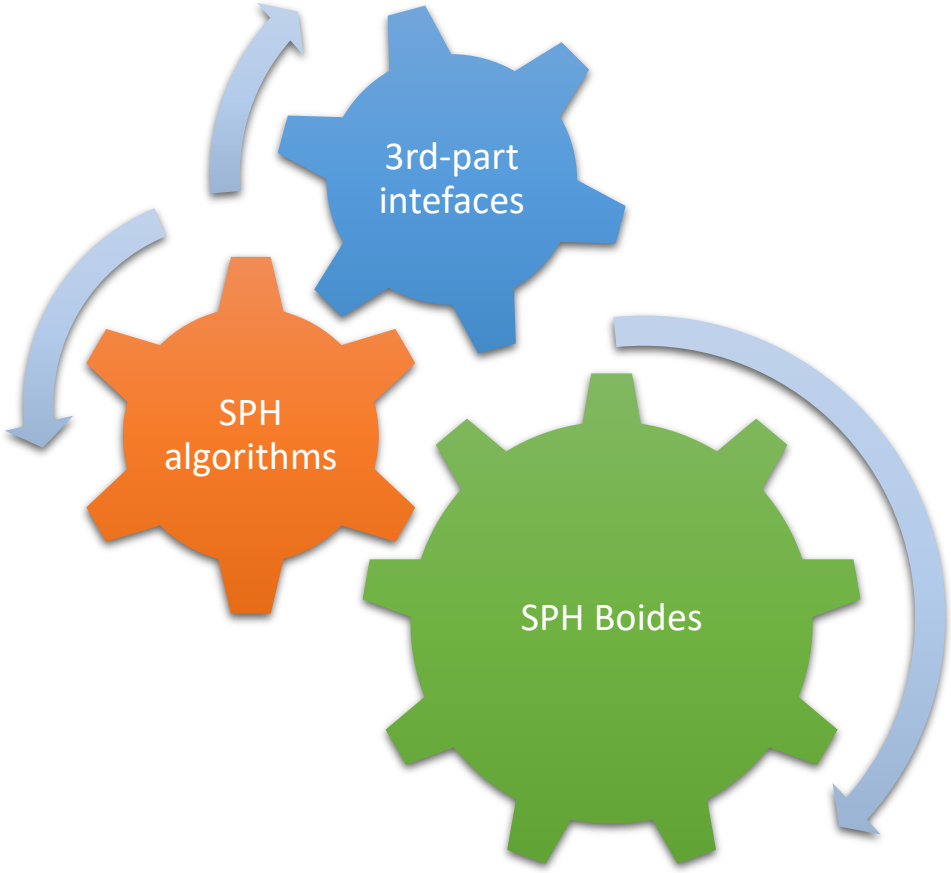
An API of SPH method for industrial complex systems

Xiangyu Hu
Department of Mechanical Engineering
Technical University of Munich

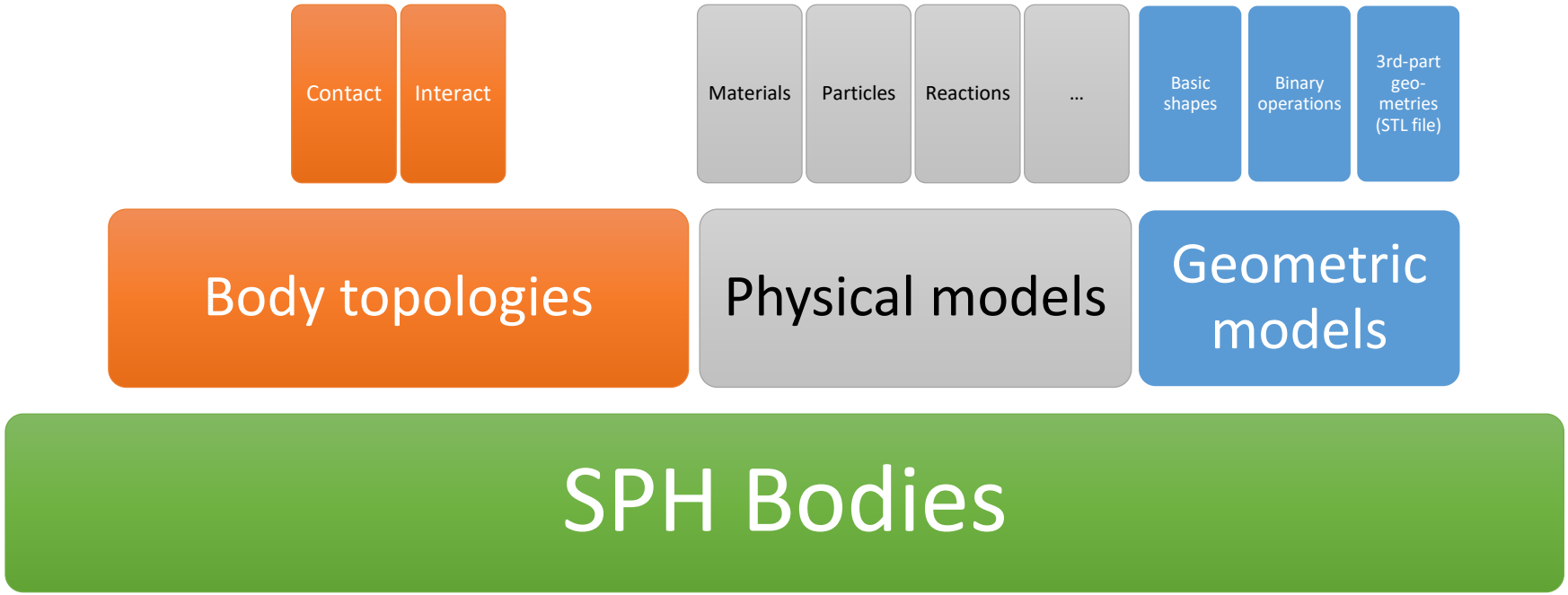
What is an API of SPH method?

- API: application programing interface
  - Libraries and framworks
  - Write APPs (applications) using APIs
- SPHinXsys as an API
  - C++ Libraries and framworks for SPH simulation
  - Write SPH APPs using SPHinXsys
    - Fast and easy coding
    - An example: Dambreak application with complex geometry and parallel computing in less than 200 lines of code
  - A multi-physics framework
    - Designed for industrial complex systems
    - Extensiblity and flexibility

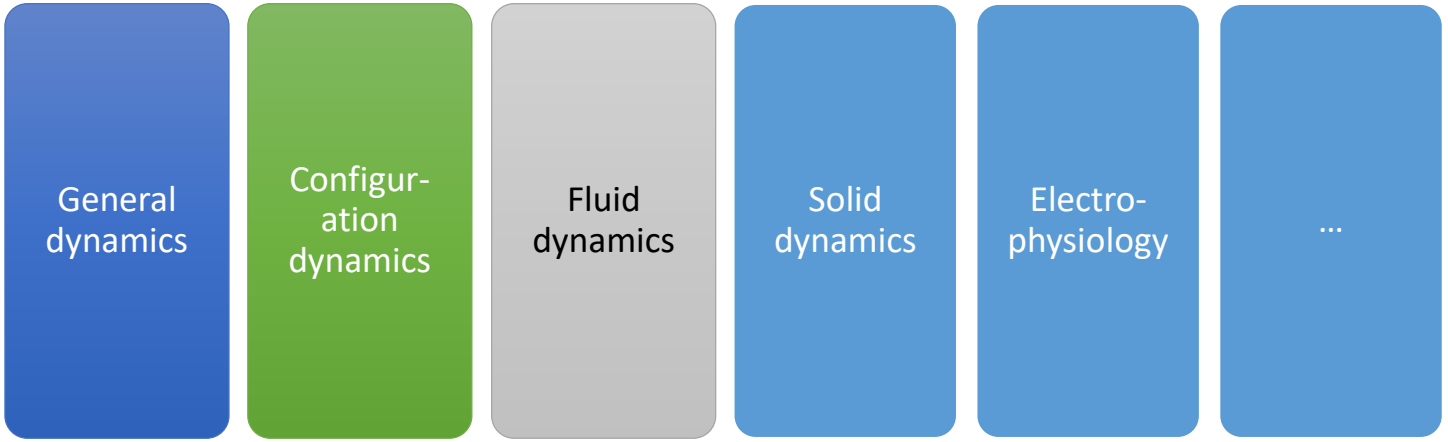
Core components of SPHinXsys



SPH Bodies

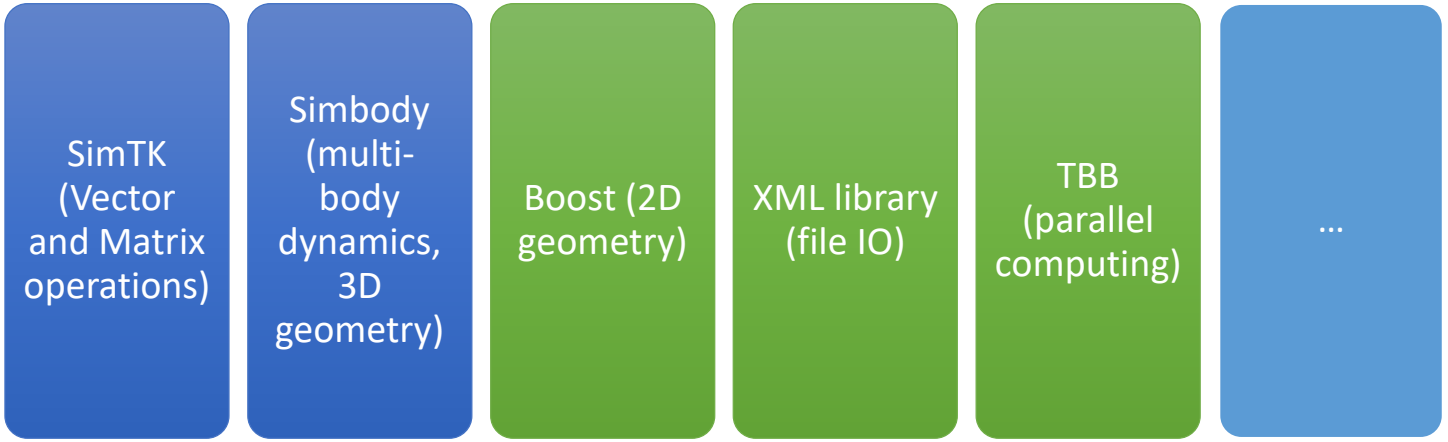

Contact
Interact
Materials
Particles
Reactions
...
Basic shapes
Binary operations
3rd-part geo-metries (STL file)
Body topologies
Physical models
Geometric models
SPH Bodies

SPH algorithms



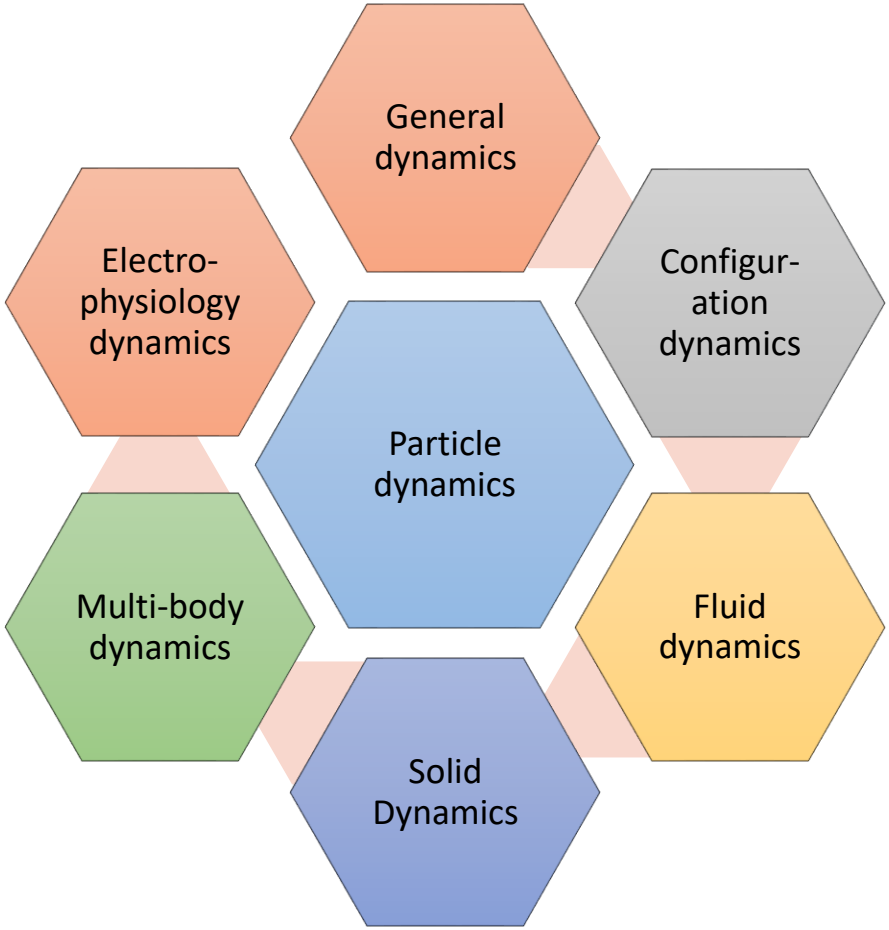
SPH alogirthms

3rd-part interfaces

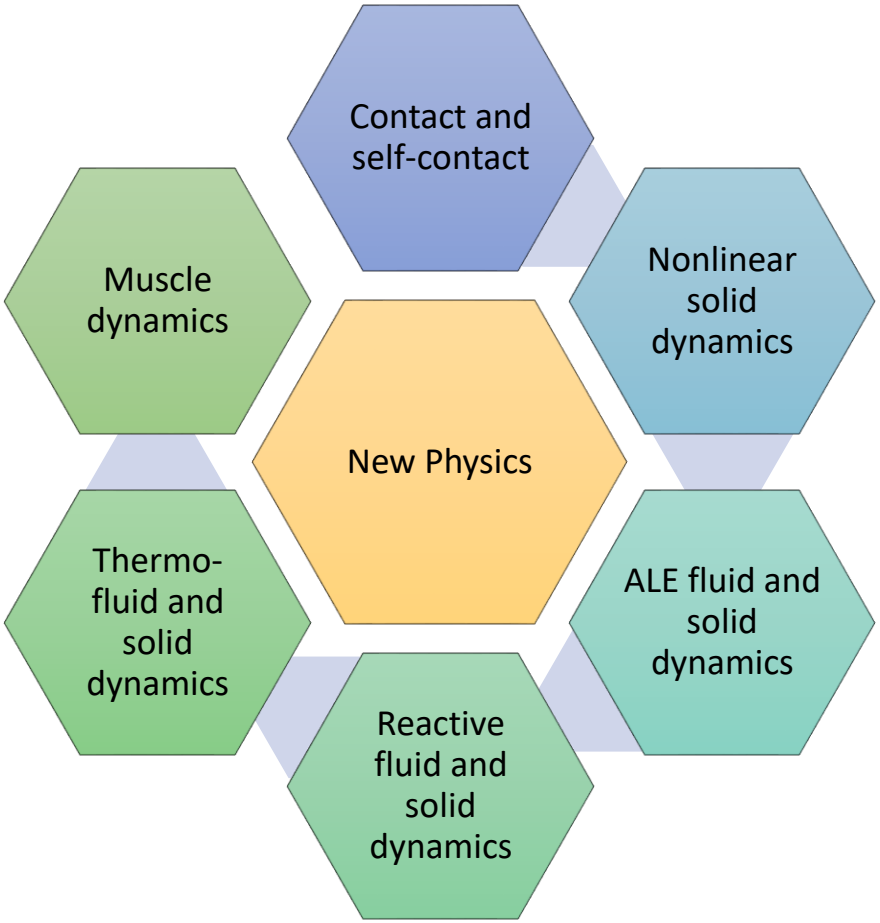


3rd-part interfaces

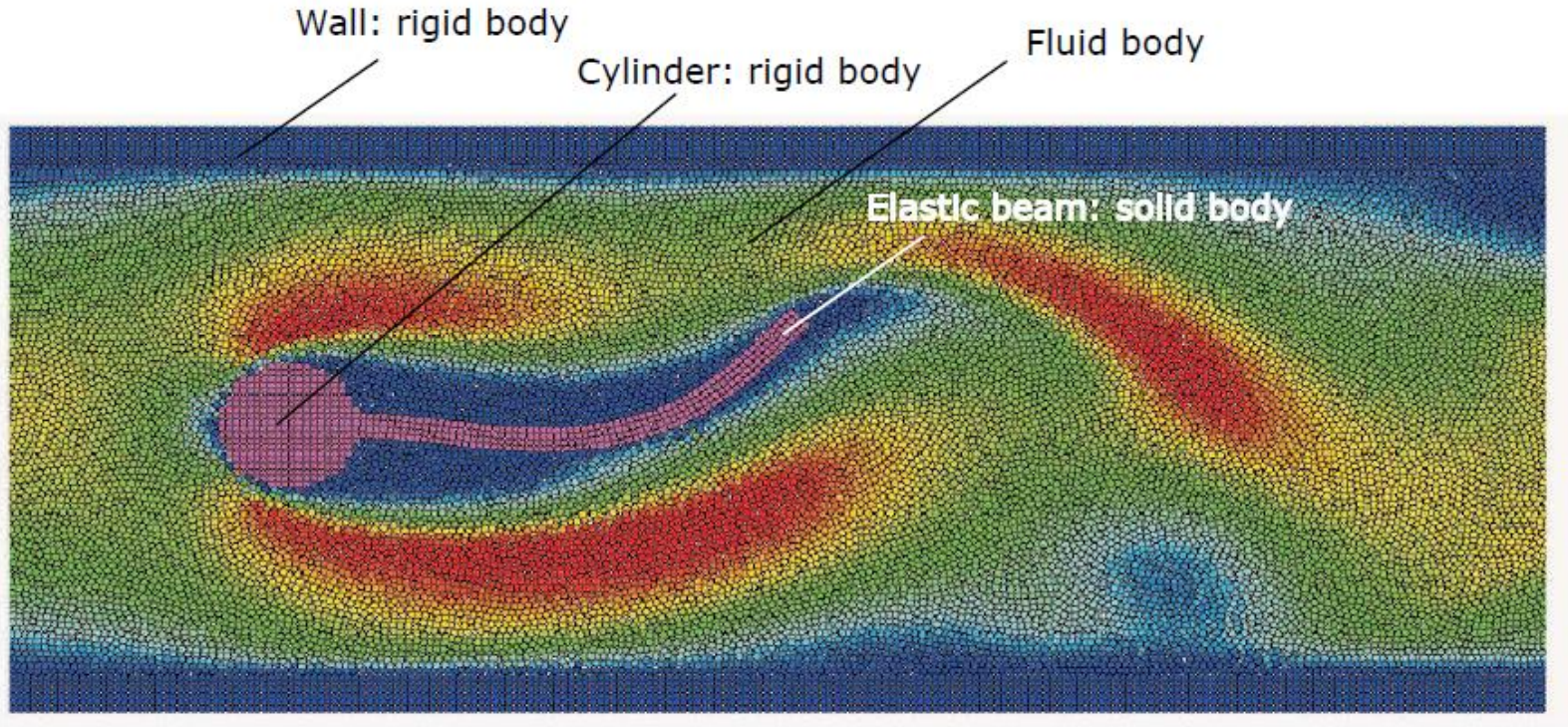
Multi-physics view of SPHinXsys



Future development on multi-physics

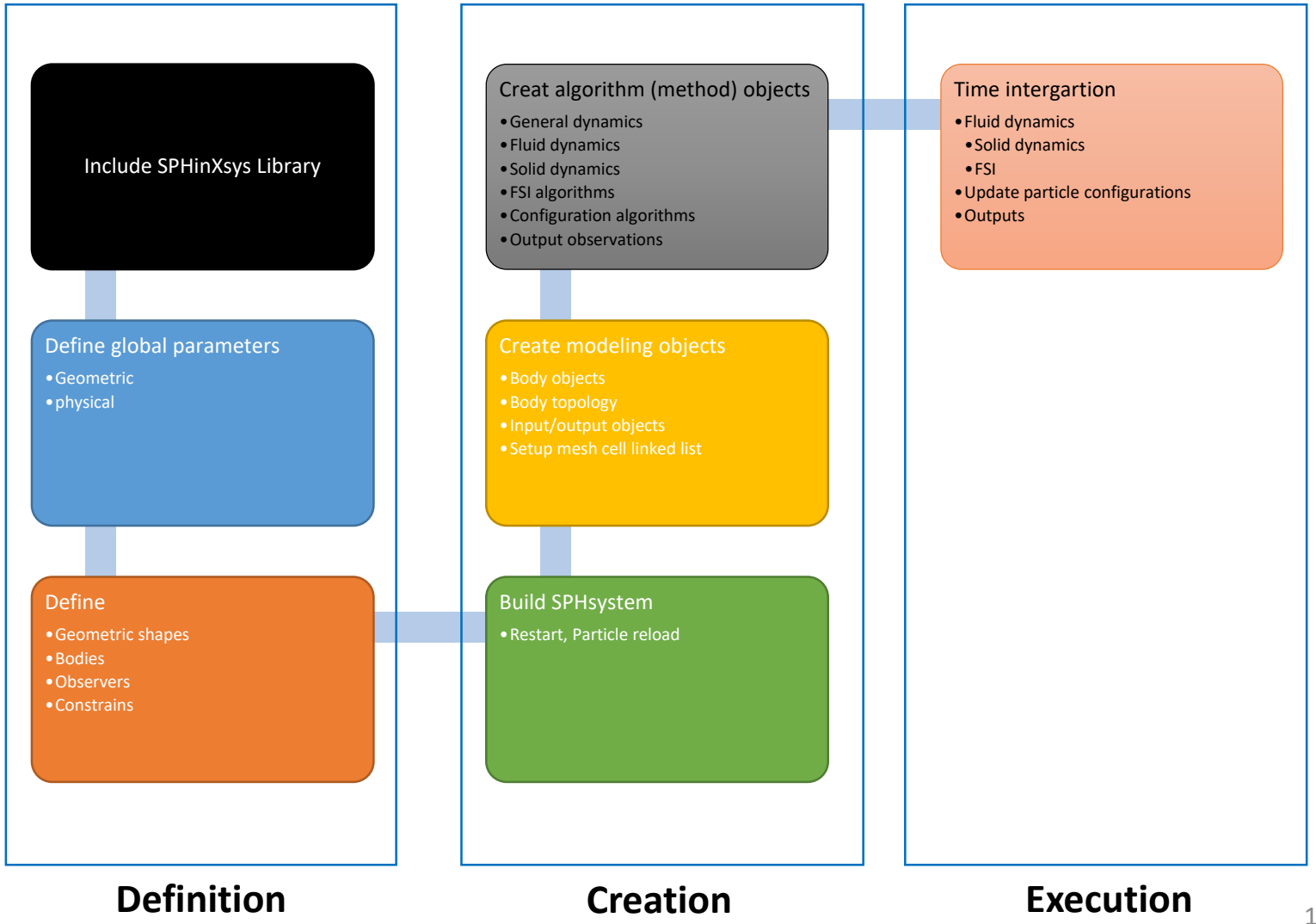


An application on FSI



A typical FSI problem involving a rigid solid (wall) body, a composite solid (insert) body and a fluid body. The wall body has two (upper and lower) components. The insert body is composed of a rigid (cylinder) and an elastic (beam) components.

Overview of the application code



Include SPHinXsys library

```
/**
 * @file fsi2.cpp
 * @brief This is the benchmark test of fliud-structure interaction.
 * @details We consider a flow-induced vibration of an elastic beam behind a cylinder in 2D.
 * @author Xiangyu Hu, Chi Zhang and Luhui Han
 * @version 0.1
 */
 /**
   * @brief SPHinXsys Library.
   */
#include "sphinxsys.h"
   /**
  * @brief Namespace cite here.
  */
using namespace SPH;
```

Global geometric parameters

```
Real DL = 11.0; /**< Channel length. */
Real DH = 4.1; /**< Channel height. */
Real particle_spacing_ref = 0.1; /**< Initial reference particle spacing. */
Real DLsponge = particle_spacing_ref *20.0;/**< Sponge region to impose inflow condition. */
Real BW = particle_spacing_ref * 4.0; /**< Boundary width, determined by specific layer of boundary particles. */
Vec2d insert_circle_center(2.0, 2.0);/**< Location of the cylinder center. */
Real insert_circle_radius = 0.5;/**< Radius of the cylinder. */
Real bh = 0.4*insert_circle_radius;/**< Height of the beam. */
Real bl = 7.0*insert_circle_radius;/**< Length of the beam. */
/** @brief Geometry of the beam. Defined through the 4 corners of a box. */
Real hbh = bh / 2.0;
Vec2d BLB(insert_circle_center[0], insert_circle_center[1] - hbh);
Vec2d BLT(insert_circle_center[0], insert_circle_center[1] + hbh);
Vec2d BRB(insert_circle_center[0] + insert_circle_radius + bl, insert_circle_center[1] - hbh);
Vec2d BRT(insert_circle_center[0] + insert_circle_radius + bl, insert_circle_center[1] + hbh);
```

Global physical parameters

```
/**
 * @brief Material properties of the fluid.
 */
Real rho0_f = 1.0;/**< Density. */
Real U_f = 1.0;/**< Cheractristic velocity. */
Real c_f = 10.0*U_f;/**< Speed of sound. */
Real Re = 100.0;/**< Reynolds number. */
Real mu_f = rho0_f * U_f * (2.0 * insert_circle_radius) / Re;/**< Dynamics visocisty. */
Real k_f = 0.0;/**< kinetic smoothness. */
/**
 * @brief Material properties of the solid,
 */
Real rho0_s = 10.0; /**< Reference density.*/
Real poisson = 0.4; /**< Poisson ratio.*/
Real Ae = 1.4e3; /**< Normalized Youngs Modulus. */
Real Youngs_modulus = Ae * rho0_f * U_f * U_f;
```

Define water block shape

```
/**
* @brief define geometry of SPH bodies
*/
std::vector<Point> CreatWaterBlockShape()
{
//geometry
std::vector<Point> water_block_shape;
water_block_shape.push_back(Point(-DLsponge, 0.0));
water_block_shape.push_back(Point(-DLsponge, DH));
water_block_shape.push_back(Point(DL, DH));
water_block_shape.push_back(Point(DL, 0.0));
water_block_shape.push_back(Point(-DLsponge, 0.0));
return water_block_shape;
}
```

Other goemtric shapes

- Inflow buffer
- Oscilating beam shape
- Wall boundary shape

Define fluid body

```
/**
 * @brief Fluid body definition.
 */
class WaterBlock : public FluidBody
{ public:
WaterBlock(SPHSystem &system, string body_name,
int refinement_level, ParticlesGeneratorOps op)
: FluidBody(system, body_name, refinement_level, op)
{ /** Geomerty definition. */
std::vector<Point> water_bock_shape = CreatWaterBlockShape();
body_region_.add_geometry(new Geometry(water_bock_shape), RegionBooleanOps::add);
/** Geomerty definition. */
body_region_.add_geometry(new Geometry(insert_circle_center, insert_circle_radius, 100),
RegionBooleanOps::sub);
std::vector<Point> beam_shape = CreatBeamShape();
body_region_.add_geometry(new Geometry(beam_shape), RegionBooleanOps::sub);
/** Finalize the geometry definition and correspoding opertation. */
body_region_.done_modeling();
} };
```

Modeling using pre-defined shapes and binary operations

Other Bodies

- Insert solid body
- Wall boundary

Insert body constrains

```
/**
* @brief constrain the beam base
*/
class BeamBase : public SolidBodyPart
{ public:
BeamBase(SolidBody *solid_body, string constrianed_region_name)
: SolidBodyPart(solid_body, constrianed_region_name)
{ std::vector<Point> beam_shape = CreatBeamShape();
Geometry *circle_geometry = new Geometry(insert_circle_center, insert_circle_radius,
100);
soild_body_part_region_.add_geometry(circle_geometry, RegionBooleanOps::add);
Geometry * beam_gemetry = new Geometry(beam_shape);
soild_body_part_region_.add_geometry(beam_gemetry, RegionBooleanOps::sub);
soild_body_part_region_.done_modeling();
/**  Tag the constrained particle. */
TagBodyPartParticles();
} };
```

Geometric region in which particles are constrained

Other constrains

- Inflow buffer
- Inflow velocity profile

Create observer for output measured beam tip position

```
/**
* @brief Definition of an observer body with one particle located at specific position
* of the insert beam.
*/
class BeamObserver : public ObserverLagrangianBody
{
public:
BeamObserver(SPHSystem &system, string body_name,
int refinement_level, ParticlesGeneratorOps op)
: ObserverLagrangianBody(system, body_name, refinement_level, op)
{
/** postion and volume. */
body_input_points_volumes_.push_back(make_pair(0.5 * (BRT + BRB), 0.0));
}
};
```

Measuring one-point displacement

Create observer for output measured inflow flow velocity profile

```
/**
* @brief Definition of an observer body with several particles located
* at the entrance of the flow channel.
*/
class FluidObserver : public ObserverEulerianBody
{ public:
FluidObserver(SPHSystem &system, string body_name,
int refinement_level, ParticlesGeneratorOps op)
: ObserverEulerianBody(system, body_name, refinement_level, op)
{ /** A line of measuring points at the entrance of the channel. */
size_t number_observation_pionts = 21;
Real range_of_measure = DH - particle_spacing_ref * 4.0;
Real start_of_measure = particle_spacing_ref*2.0;
for (size_t i = 0; i < number_observation_pionts; ++i) {
Vec2d point_coordinate(0.0, range_of_measure*Real(i) / Real(number_observation_pionts -
1) + start_of_measure);
body_input_points_volumes_.push_back(make_pair(point_coordinate, 0.0));
} } };
```

Measuring flow profile at a cross-section

Main and creating SPHSystem

```
int main()
{
/**
 * @brief Build up -- a SPHSystem --
 */
SPHSystem system(Vec2d(-DLsponge - BW, -BW), Vec2d(DL + BW, DH + BW),
particle_spacing_ref);
/** Set the starting time. */
GlobalStaticVariables::physical_time_ = 0.0;
/** Tag for computation from restart files. 0: not from restart files. */
system.restart_step_ = 0;
/** Tag for reload initially repaxed particles. */
system.reload_particle_ = false;
```

Entire domain lower and upper bounds

Create body objects

```
/**
 * @brief Material property, particles and body creation of fluid.
 */
WaterBlock *water_block
=new WaterBlock(system, "WaterBody", 0, ParticlesGeneratorOps::lattice);
SymmetricTaitFluid fluid("Water", water_block, rho0_f, c_f, mu_f, k_f);
FluidParticles fluid_particles(water_block);
/**
 * @brief Particle and body creation of wall boundary.
 */
WallBoundary *wall_boundary
= new WallBoundary(system, "Wall", 0, ParticlesGeneratorOps::lattice);
SolidParticles solid_particles(wall_boundary);
```

Body, material and particles

Body with default material, particles

Create other body objects

```
/**
 * @brief Material property, particle and body creation of elastic beam(inserted body).
 */
InsertedBody *inserted_body = new InsertedBody(system, "InsertedBody", 1, ParticlesGeneratorOps::lattice);
ElasticSolid insert_body_material("ElasticSolid", inserted_body, rho0_s, Youngs_modulus, poisson, 0.0);
ElasticSolidParticles inserted_body_particles(inserted_body);
/**
 * @brief Particle and body creation of beam and fluid observers.
 */
BeamObserver *beam_observer =new BeamObserver(system, "BeamObserver", 0, ParticlesGeneratorOps::direct);
ObserverParticles beam_observer_particles(beam_observer);


FluidObserver *fluid_observer = new FluidObserver(system, "FluidObserver", 0, ParticlesGeneratorOps::direct);
ObserverParticlesflow_observer_particles(fluid_observer);
```

Create input and output objects

```
/**
 * @brief simple input and outputs.
 */
In_Output in_output(system);
WriteBodyStatesToVtu write_real_body_states_to_vtu(in_output, system.real_bodies_);
WriteBodyStatesToPlt write_real_body_states_to_plt(in_output, system.real_bodies_);
WriteRestart write_restart_files(in_output, system.real_bodies_);
ReadRestart read_restart_files(in_output, system.real_bodies_);
ReadReloadParticle read_reload_particles(in_output, { inserted_body, water_block }, {
"InsertBody", "WaterBody" });
```

Body topology and setup mesh cell linked list

Body topology

```
/**
 * @brief Body contact map.
 * @details The contact map gives the data conntections between the bodies.
 * Basically the the rangE of bodies to build neighbor particle lists.
 */
SPHBodyTopology body_topology = { { water_block, { wall_boundary, inserted_body } },
  { wall_boundary, { } }, { inserted_body, { water_block } },
  { beam_observer, {inserted_body} }, { fluid_observer, { water_block } } };
system.SetBodyTopology(&body_topology);
/**
 * @brief Simulation data structure set up.
 */
/** check whether reload particles. */
if (system.reload_particle_) read_reload_particles.ReadFromFile();
system.InitializeSystemCellLinkedLists();
system.InitializeSystemConfigurations();
```

Define methods used once: initial conditions and corrective kernel

- `/** initial condition for fluid body */`
- `fluid_dynamics::WeaklyCompressibleFluidInitialCondition set_all_fluid_particles_at_rest(water_block);`
- `/** Obtain the initial number density of fluid. */`
- `fluid_dynamics::InitialNumberDensity fluid_initial_number_density(water_block, { wall_boundary, inserted_body });`
- `/** initial condition for the solid body */`
- `solid_dynamics::SolidDynamicsInitialCondition set_all_wall_particles_at_rest(wall_boundary);`
- `/** initial condition for the elastic solid bodies */`
- `solid_dynamics::ElasticSolidDynamicsInitialCondition set_all_insert_body_particles_at_rest(inserted_body);`

Reproducing kernel

- `/** Corrected strong configuration. */`
- `solid_dynamics::CorrectConfiguration inserted_body_corrected_configuration_in_strong_form(inserted_body, {});`

Reset acceleration, periodic and inflow boudary condition

```
/** Initialize particle acceleration. */
InitializeOtherAccelerations initialize_other_acceleration(water_block);
/** Periodic bounding. */
PeriodicBoundingInXDirection periodic_bounding(water_block);
/** Periodic BCs. */
PeriodicConditionInXDirection periodic_condition(water_block);
/** Inflow boundary condition. */
ParabolicInflow
parabolic_inflow(water_block, new InflowBuffer(water_block, "Buffer"));
```

In and out flow conditions achieved by the combination of periodic and inflow condition

Fluid dynamics

```
/** Evaluation of density by summation approach. */
fluid_dynamics::DensityBySummation update_fluid_desnity(water_block, { wall_boundary, inserted_body });
/** Time step size without considering sound wave speed. */
fluid_dynamics::GetAdvectionTimeStepSize get_fluid_adevction_time_step_size(water_block, U_f);
/** Time step size with considering sound wave speed. */
fluid_dynamics::GetAcousticTimeStepSizeget_fluid_time_step_size(water_block);
/** Pressure relaxation using verlet time stepping. */
fluid_dynamics::PressureRelaxationVerlet pressure_relaxation(water_block, { wall_boundary, inserted_body });
/** Computing viscous acceleration. */
fluid_dynamics::ComputingViscousAcceleration viscous_acceleration(water_block, { wall_boundary, inserted_body });
/** Impose transport velocity. */
fluid_dynamics::TransportVelocityCorrection transport_velocity_correction(water_block, { wall_boundary, inserted_body });
/** Computing vorticity in the flow. */
fluid_dynamics::ComputingVorticityInFluidField compute_vorticity(water_block);
```

Namespace for fluid dynamics

Solid dynamics

```
/**
 * @brief Algorithms of solid dynamics.
 */
/** Compute time step size of elastic solid. */
solid_dynamics::GetAcousticTimeStepSize
inserted_body_computing_time_step_size(inserted_body);


/** Stress relaxation for the inserted body. */
solid_dynamics::StressRelaxationFirstStep
inserted_body_stress_relaxation_first_step(inserted_body);
solid_dynamics::StressRelaxationSecondStep
inserted_body_stress_relaxation_second_step(inserted_body);


/** Constrain region of the inserted body. */
solid_dynamics::ConstrainSolidBodyRegion
constrain_beam_base(inserted_body, new BeamBase(inserted_body, "BeamBase"));
```

Namespace for solid dynamics

Fluid-Structure-Interaction (FSI) methods

```
/**
 * @brief Algorithms of FSI.
 */
/** Compute the force exerted on solid body due to fluid pressure and viscosity. */
solid_dynamics::FluidPressureForceOnSolid
fluid_pressure_force_on_insrted_body(inserted_body, { water_block });
solid_dynamics::FluidViscousForceOnSolid
fluid_viscous_force_on_insrted_body(inserted_body, { water_block });


/** Computing the average velocity. */
solid_dynamics::InitializeDisplacement
inserted_body_initialize_displacement(inserted_body);
solid_dynamics::UpdateAverageVelocity inserted_body_average_velocity(inserted_body);
```

Namespace for Solid dynamics

Update particle configurations and obsrevation methods

Only update when necessary

```
/**
 * @brief Methods used for updating data structure.
 */
/** Update the cell linked list of bodies when necessary. */
ParticleDynamicsCellLinkedList update_water_block_cell_linked_list(water_block);
/** Update the configuration of bodies when necessary. */
ParticleDynamicsConfiguration update_water_block_configuration(water_block);
/** Update the cell linked list of bodies when necessary. */
ParticleDynamicsCellLinkedList update_inserted_body_cell_linked_list(inserted_body);
/** Update the contact configuration for a given contact map. */
ParticleDynamicsContactConfiguration update_inserted_body_contact_configuration(inserted_body);
/** Update the contact configuration for the flow observer. */
ParticleDynamicsContactConfiguration update_fluid_observer_body_contact_configuration(fluid_observer);
/**
 * @brief observation outputs.
 */
WriteTotalViscousForceOnSolid write_total_viscous_force_on_inserted_body(in_output, inserted_body);
WriteObservedElasticDisplacement write_beam_tip_displacement(in_output, beam_observer, { inserted_body });
WriteObservedFluidVelocity write_fluid_velocity(in_output, fluid_observer, { water_block });
```

Pre-simulation step

Execute initial condition

```
/** Pre-simulation*/
set_all_fluid_particles_at_rest.exec();
set_all_wall_particles_at_rest.exec();
set_all_insert_body_particles_at_rest.exec();
periodic_condition.parallel_exec();
/** one need update configuration after periodic condition. */
update_water_block_configuration.parallel_exec();
fluid_initial_number_density.parallel_exec();
inserted_body_corrected_configuration_in_strong_form.parallel_exec();
```

Restart if necessary

```
/**
 * @brief The time stepping starts here.
 */
if(system.restart_step_ != 0)
{
GlobalStaticVariables::physical_time_ =
read_restart_files.ReadRestartFiles(system.restart_step_);
update_water_block_cell_linked_list.parallel_exec();
update_inserted_body_cell_linked_list.parallel_exec();
periodic_condition.parallel_exec();
/** one need update configuration after peroidic condition. */
update_water_block_configuration.parallel_exec();
update_inserted_body_contact_configuration.parallel_exec();
get_inserted_body_normal.parallel_exec();
}
write_real_body_states_to_plt.WriteToFile(GlobalStaticVariables::physical_time_);
write_beam_tip_displacement.WriteToFile(GlobalStaticVariables::physical_time_);
```

Time integration controls

```
int number_of_iterations = system.restart_step_;
int screen_output_interval = 100;
int restart_output_interval = screen_output_interval * 10;
Real End_Time = 200.0;/**< End time. */
Real D_Time = End_Time/200.0;/**< time stamps for output. */
Real Dt = 0.0;/**< Default advection time step sizes for fluid. */
Real dt = 0.0; /**< Default acoustic time step sizes for fluid. */
Real dt_s = 0.0;/**< Default acoustic time step sizes for solid. */
/** Statistics for computing time. */
tick_count t1 = tick_count::now();
tick_count::interval_t interval;
```

Measure computation time

Density, viscous force and transport velocity formulation

```
while (GlobalStaticVariables::physical_time_ < End_Time)
{ Real integeral_time = 0.0;
/** Integrate time (loop) until the next output time. */
while (integeral_time < D_Time) {
Dt = get_fluid_adevction_time_step_size.parallel_exec();
update_fluid_desnity.parallel_exec();
initialize_other_acceleration.parallel_exec();
viscous_acceleration.parallel_exec();
transport_velocity_correction.parallel_exec(Dt);
/** FSI for viscous force. */
fluid_viscous_force_on_insrted_body.parallel_exec();
/** Update normal direction on elastic body.*/
inserted_body_update_normal.parallel_exec();
```

Fluid advection time scale dynamics

Fluid pressure and solid stress relaxation, FSI

```
Real relaxation_time = 0.0;
while (relaxation_time < Dt) {
/** Fluid pressure relaxation. */
pressure_relaxation.parallel_exec(dt);
/** FSI for pressure force. */
fluid_pressure_force_on_insrted_body.parallel_exec();
/** Solid dynamics. */
Real dt_s_sum = 0.0;
inserted_body_initialize_displacement.parallel_exec();
while (dt_s_sum < dt) {

dt_s = inserted_body_computing_time_step_size.parallel_exec();
if (dt - dt_s_sum < dt_s) dt_s = dt - dt_s_sum;
inserted_body_stress_relaxation_first_step.parallel_exec(dt_s);
constrain_beam_base.parallel_exec();
inserted_body_stress_relaxation_second_step.parallel_exec(dt_s);
dt_s_sum += dt_s;
}
inserted_body_average_velocity.parallel_exec(dt);
```

Fluid and solid acoustic time scale interactions

Update particle configuration and carry out observations

```
dt = get_fluid_time_step_size.parallel_exec();
relaxation_time += dt;
integeral_time += dt;
GlobalStaticVariables::physical_time_ += dt;
parabolic_inflow.parallel_exec();
}
```

Boundary conditions and update particle configurations

```
/** Water block configuration and periodic condition. */
periodic_bounding.parallel_exec();
update_water_block_cell_linked_list.parallel_exec();
periodic_condition.parallel_exec();
update_water_block_configuration.parallel_exec();
/** Inserted body contact configuration. */
update_inserted_body_cell_linked_list.parallel_exec();
update_inserted_body_contact_configuration.parallel_exec();
write_beam_tip_displacement.WriteToFile(GlobalStaticVariables::physical_time_);
```

Write bodies states and computing time statistics

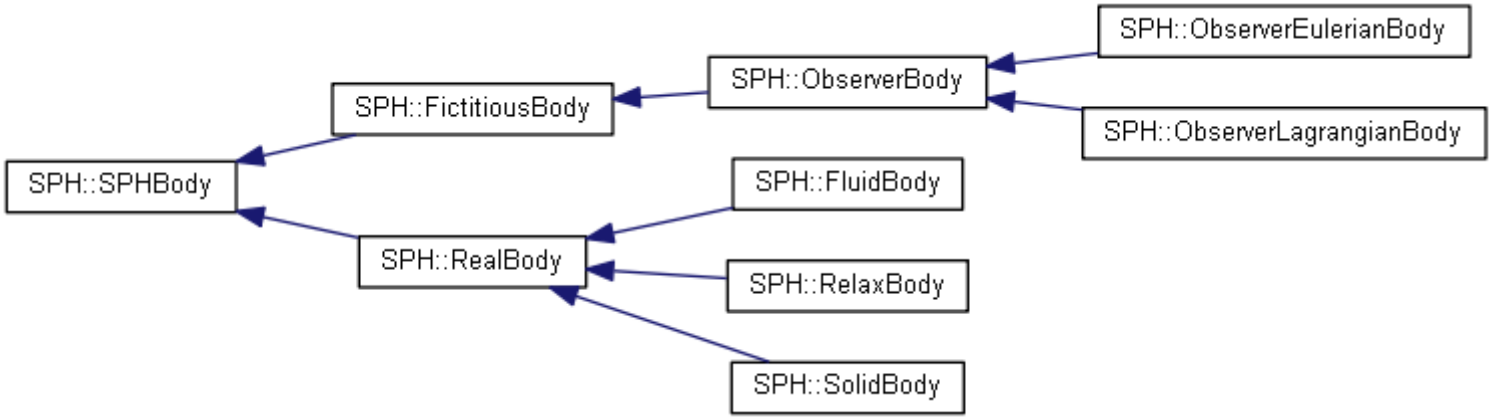
```
compute_vorticity.parallel_exec();
tick_count t2 = tick_count::now();
write_real_body_states_to_vtu.WriteToFile(GlobalStaticVariables::physical_time_);
write_total_viscous_force_on_inserted_body.WriteToFile(GlobalStaticVariables::physical_time_);
update_fluid_observer_body_contact_configuration.parallel_exec();
write_fluid_velocity.WriteToFile(GlobalStaticVariables::physical_time_);
tick_count t3 = tick_count::now();
interval += t3 - t2;
}
tick_count t4 = tick_count::now();

tick_count::interval_t tt;
tt = t4 - t1 - interval;
cout << "Total wall time for computation: " << tt.seconds() << " seconds." << endl;

return 0;
```

Measure total computation time

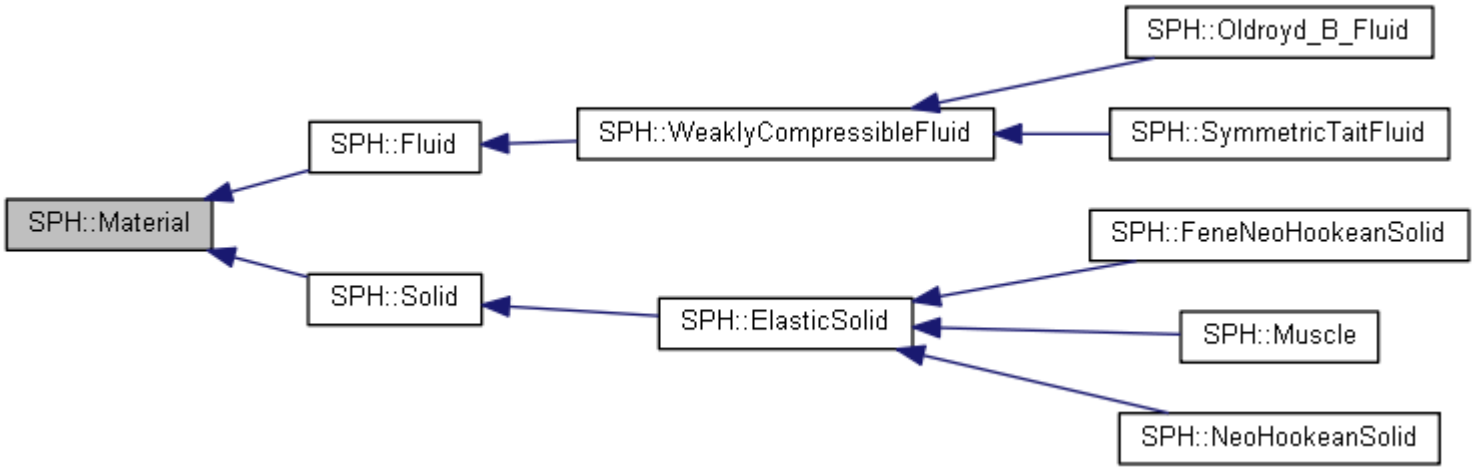
Develop to extend SPHinXsys

- Define new SPH bodies
  - New materials
  - New type of particles with new dynamical variables
- Define new SPH algorithms
  - New physics
- Main approch for extending
  - Override base classes in SPHinXsys
  - New namespace for the new physics
- Do not need worry about
  - Parallelization
  - Complex geometries

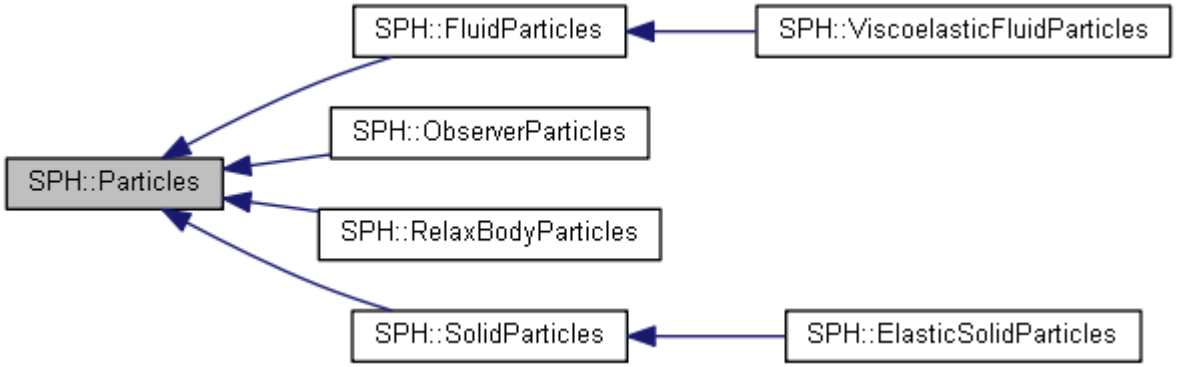
SPH Bodies


SPH::ObserverEulerianBody
SPH::ObserverBody
SPH::FictitiousBody
SPH::ObserverLagrangianBody
SPH::SPHBody
SPH::FluidBody
SPH::RealBody
SPH::RelaxBody
SPH::SolidBody

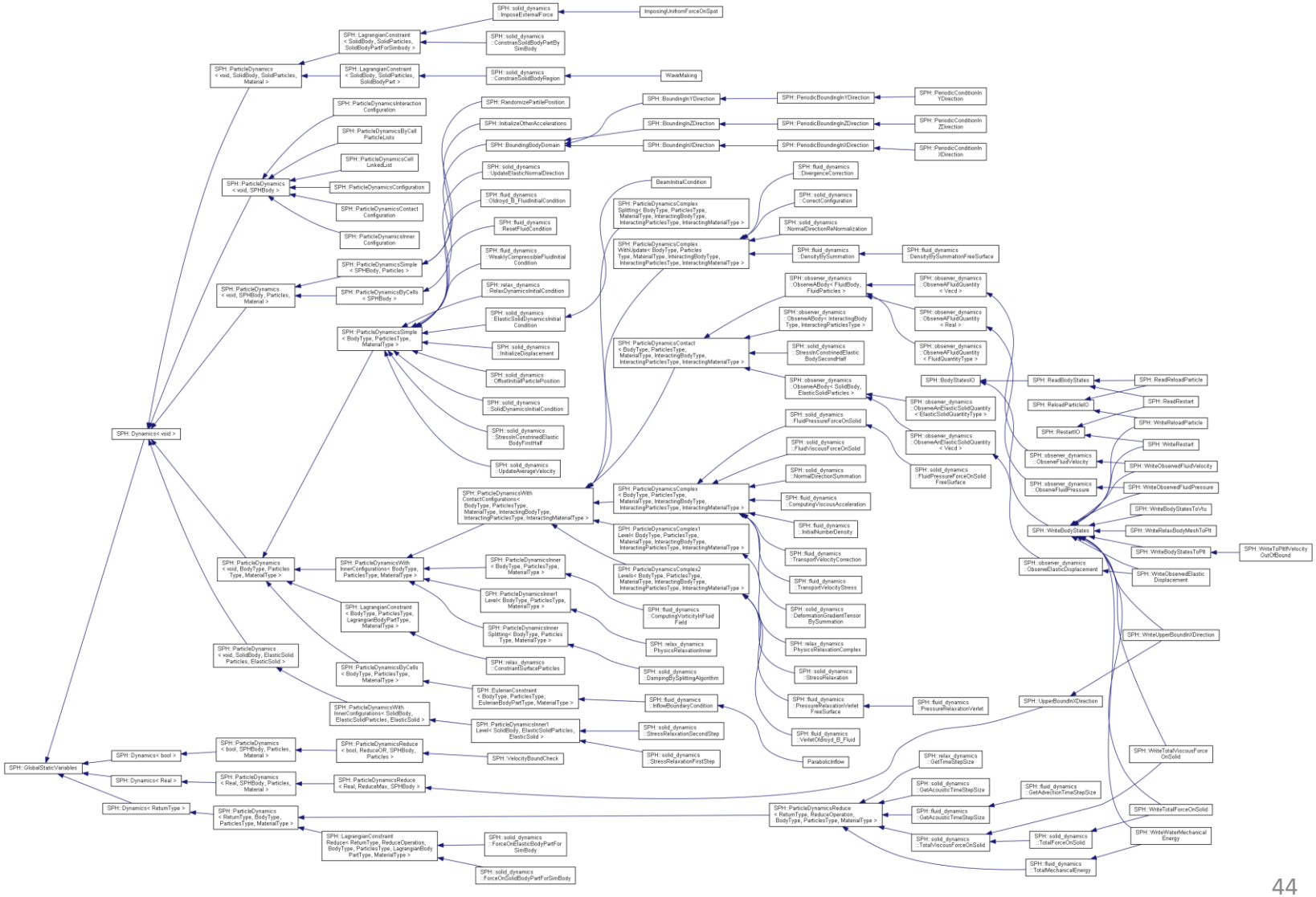
Materials


SPH::Oldroyd_B_Fluid
SPH::Fluid
SPH::WeaklyCompressibleFluid
SPH::SymmetricTaitFluid
SPH::Material
SPH::FeneNeoHookeanSolid
SPH::Solid
SPH::ElasticSolid
SPH::Muscle
SPH::NeoHookeanSolid

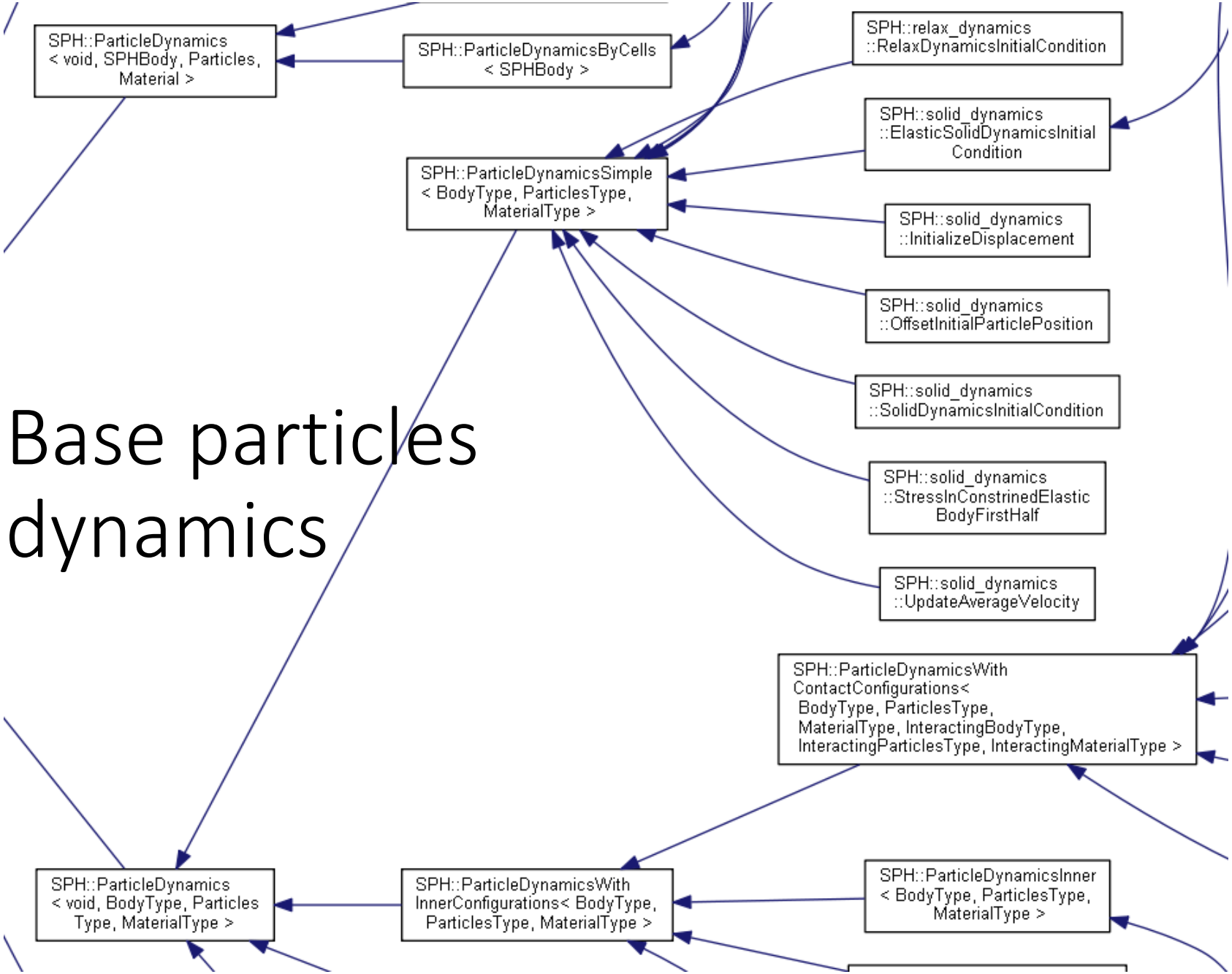
Particles


SPH::FluidParticles
SPH::ViscoelasticFluidParticles
SPH::ObserverParticles
SPH::Particles
SPH::RelaxBodyParticles
SPH::SolidParticles
SPH::ElasticSolidParticles

Particle dynamics



Base particles dynamics



Example of typical material

```
/**
* @class NeoHookeanSolid
* @brief Neo-Hookean solid
*/
class NeoHookeanSolid : public ElasticSolid
{
public:
            NeoHookeanSolid(string elastic_solid_name, SPHBody *body, Real rho_0 = 1.0,
            Real E_0 = 1.0, Real poisson = 0.3, Real eta_0 = 0.0);
            virtual ~NeoHookeanSolid() {};

            /** the interface for dynamical cast*/
            virtual NeoHookeanSolid* PointToThisObject() override { return this; };

            /** compute elastic stress */
            virtual Matd ConstitutiveRelation(Matd &deform_grad,
            size_t particle_index_i) override;
};
```

Example of typical particle dynamics

```
/**
 * @class ComputingViscousAcceleration
 * @brief  the viscosity force induecd accelerarion
 */
class ComputingViscousAcceleration : public WeaklyCompressibleFluidDynamicsComplex
{
protected:
          //viscousity
          Real mu_;
          Real smoothing_length_;

          virtual void InnerInteraction(size_t index_particle_i, Real dt = 0.0) override;
          virtual void ContactInteraction(size_t index_particle_i,
          size_t interacting_body_index, Real dt = 0.0) override;
public:
          ComputingViscousAcceleration(FluidBody *body, StdVec<SolidBody*> interacting_bodies)
          : WeaklyCompressibleFluidDynamicsComplex(body, interacting_bodies) {
                    mu_ = material_->mu_;
                    smoothing_length_ = body->kernel_->GetSmoothingLength();
          };
          virtual ~ComputingViscousAcceleration() {};
};
```

SPHinXsys repository

https://github.com/Xiangyu-Hu/SPHinXsys

SPHinXsys documents

https://xiangyu-hu.github.io/SPHinXsys/

SPHinXsys examples