Adding an Explicit Godunov-Type Solid Model to Solids4Foam

Khoder Alhamwi Alshaar

Aerospace Engineering Department,

Indian Institute of Technology Bombay,

Mumbai, India

January 2025

Content:

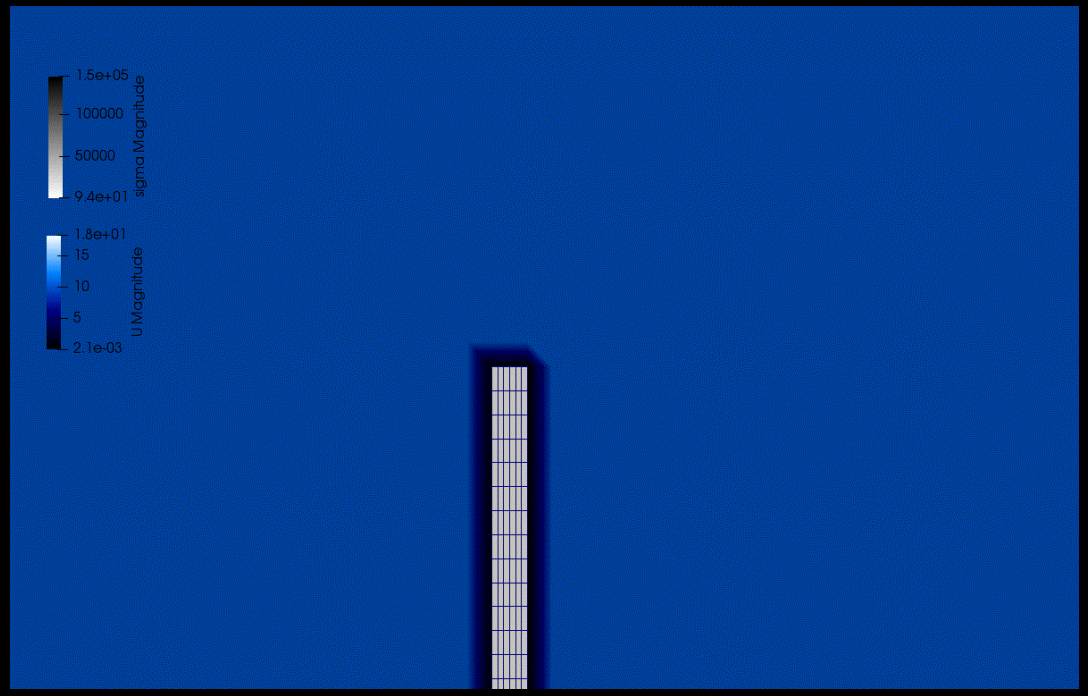
1. Theory
2. Structure of explicitSolidDynamics Toolkit
3. Structure of solids4foam Toolbox
4. Implementing explicitGodunovCC Solid Model
5. FSI Benchmark

Content:

- Theory
  - Intorduction
  - Fluid-Solid Interaction
  - Explicit Godunov Solid Solver
- Structure of explicitSolidDynamics Toolkit
- Structure of solids4foam Toolbox
- Implementing explicitGodunovCC Solid Model
- FSI Benchmark

1.1 Introduction

- What is Fluid-solid interaction (FSI) ?
  - Interaction between a deformable solid structure and a surrounding or internal fluid.



1.1 Introduction

- What is Fluid-solid interaction (FSI) ?
  - Interaction between a deformable solid structure and a surrounding or internal fluid.
- Why Finite Volume Method (FVM) for FSI?
  - FVM is a robust foundation for numerically simulating fluid and solid systems[1].
- Available software for FSI with FVM?
  - solids4foam toolbox: Built on OpenFOAM

[1] P. Cardiff and I. Demirˇzi´c, “Thirty years of the finite volume method for solid mechanics,” Archives of Computational Methods in Engineering, vol. 28, no. 5, pp. 3721–3780, 2021.
[2] P. Cardiff, A. Karaˇ c, P. De Jaeger, H. Jasak, J. Nagy, A. Ivankovi´ c, and ˇ Z. Tukovi´ c, “An opensource finite volume toolbox for solid mechanics and fluid-solid interaction simulations,” arXiv preprint arXiv:1808.10736, 2018.

1.1 Introduction

- Goal of this project?
  - Extend the solids4foam toolbox with a novel solid model.
  - Based on a momentum-deformation first-order system of equations[3].
  - Implemented in OpenFOAM as explicitSolidDynamics Toolkit[4].
  - First application of this formulation to FSI problems.

[3] J. A. Trangenstein and P. Colella, "A higher-order godunov method for modeling finite deformation in elastic-plastic solids," Communications on Pure and Applied mathematics, vol. 44, no. 1, pp. 41–100, 1991.

[4] J. Haider, "ExplicitSolidDynamics toolkit for OpenFOAM," 2019. [Online]. Available: https://github.com/jibranhaider/explicitSolidDynamics

1.1 Introduction

- Advantage of using a momentum-deformation-based formulation?
  - It is expressed as a first-order conservation laws.
  - The system is hyperbolic.
  - It enables the use of Riemann solvers.

Source: C. H. Lee, A. J. Gil, and J. Bonet, “Development of a cell centred upwind finite volume algorithm for a new conservation law formulation in structural dynamics,” Computers & Structures, vol. 118, pp. 13–38,

1.1 Introduction

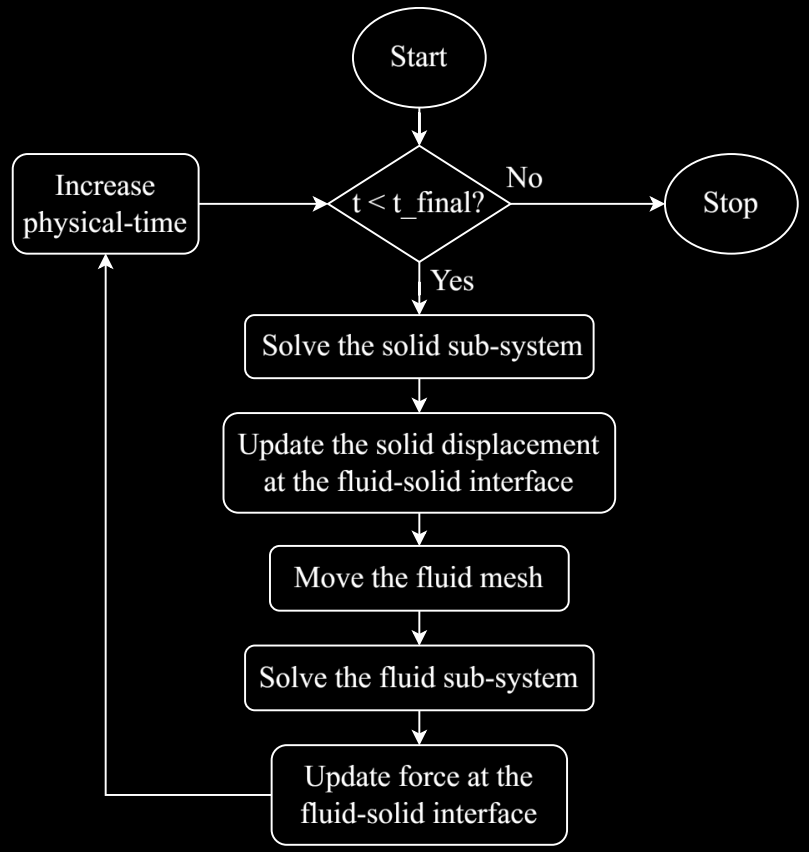
- General notes:
  - The code development for this project is based on **solids4foam v2.1** using **OpenFOAM v2012**.
  - The new solid model is derived from the explicitSolidDynamics **ported** to **OpenFOAM v2012.**

1.2 Fluid-Solid Interaction

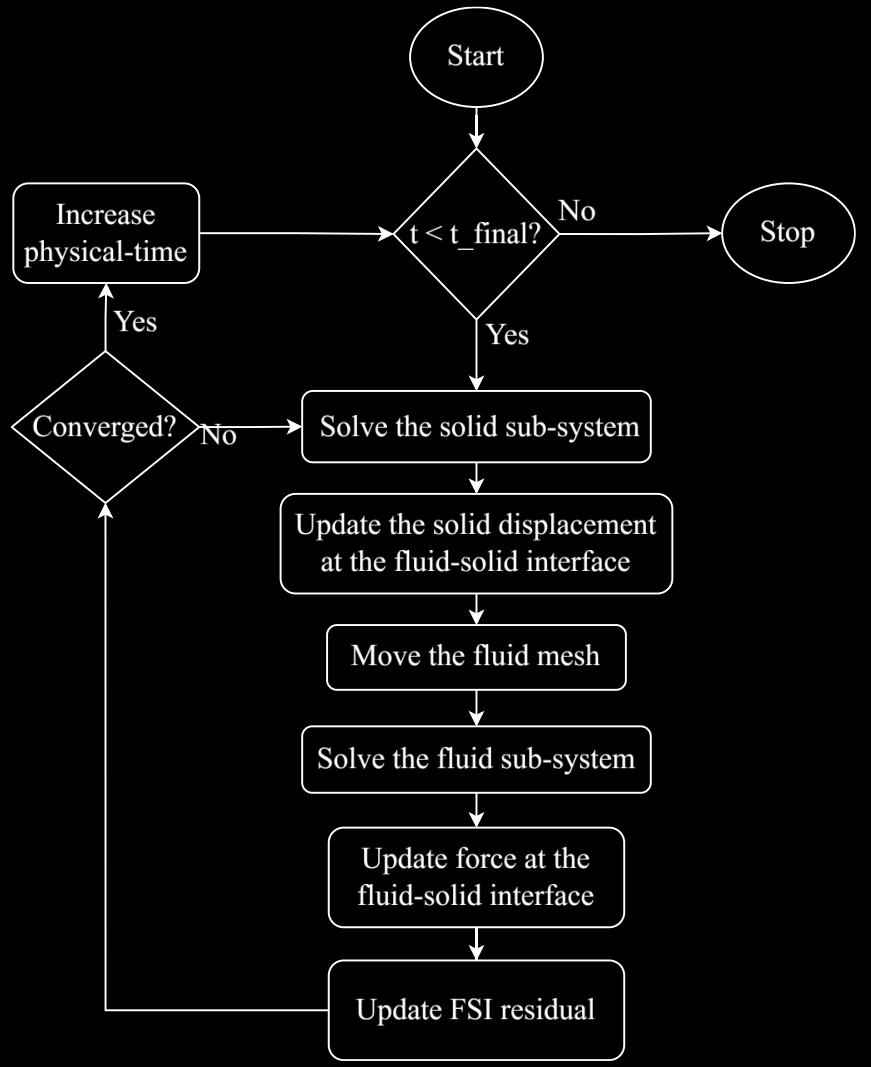
- Classified based on the coupling procedure:
  - Monolithic: Both domains are treated as a single coupled system.
  - Partitioned: Each domain solved independently with a coupling algorithm[6].
- Solids4foam toolbox incorporate a partitioned algorithm.

[6] Z. Tukovi´c, A. Kara˘c, P. Cardiff, H. Jasak, and A. Ivankovi´ c, “Openfoam finite volume solver for fluid-solid interaction,” Transactions of FAMENA, vol. 42, no. 3, pp. 1–31, 2018

1.2 Fluid-Solid Interaction



Weakly-coupled algorithm



Strongly-coupled algorithm

1.3 Explicit Godunov Solid Solver

The formulation has the linear momentum and deformation gradient as the conservative variables:

$$\begin{aligned} \frac{\partial \boldsymbol{p}}{\partial t} - \frac{\partial (\boldsymbol{P}\boldsymbol{E_I})}{\partial \boldsymbol{X_I}} &= \rho_0 \boldsymbol{b}, \\ \frac{\partial \boldsymbol{F}}{\partial t} - \frac{\partial (\frac{1}{\rho_0}\boldsymbol{p} \otimes \boldsymbol{E}_I)}{\partial \boldsymbol{X_I}} &= 0, \end{aligned} \qquad \forall\, I = 1,2,3$$

$$\frac{\partial}{\partial t}\begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{F} \end{bmatrix} + \frac{\partial}{\partial \boldsymbol{X}_I}\begin{bmatrix} -\boldsymbol{P}\boldsymbol{E}_I \\ -\frac{1}{\rho_0}\boldsymbol{p} \otimes \boldsymbol{E}_I \end{bmatrix} = \begin{bmatrix} \rho_0 \boldsymbol{b} \\ 0 \end{bmatrix}$$

$$\frac{\partial \boldsymbol{\mathcal{U}}}{\partial t} + \frac{\partial \boldsymbol{\mathcal{F}_I}}{\partial \boldsymbol{X}_I} = \boldsymbol{\mathcal{S}}$$

$\boldsymbol{p}$: Linear Momentum

$\boldsymbol{P}$: PK1 Stress Tensor

$\boldsymbol{F}$: Deformation gradient Tensor

$\boldsymbol{b}$: Body force

ρ : Density

$\boldsymbol{E}$: Cartesian material coordinate basis vector

$\boldsymbol{X}$: Cartesian material coordinates vector

Source:C. H. Lee, A. J. Gil, and J. Bonet, “Development of a cell centred upwind finite volume algorithm for a new conservation law formulation in structural dynamics,” Computers & Structures, vol. 118, pp. 13–38, 2013.

1.3.1 Finite volume discretization

$$\frac{\partial \boldsymbol{\mathcal{U}}}{\partial t} + \frac{\partial \boldsymbol{\mathcal{F}_I}}{\partial \boldsymbol{X}_I} = \boldsymbol{\mathcal{S}} \longrightarrow \int_{\Omega_0} \frac{\partial \boldsymbol{\mathcal{U}}}{\partial t}\, \mathrm{d}\Omega_0 + \int_{\partial\Omega_0} \boldsymbol{\mathcal{F}}_I N_I\, \mathrm{d}A_0 = \int_{\Omega_0} \boldsymbol{\mathcal{S}}\, \mathrm{d}\Omega_0$$

Ω_0 : An arbitrary volume

$\partial\Omega_0$: a surface enclosing the volume

N_I : The I^{th} component of the material unit normal vector

A_0 : Surface area

Source:C. H. Lee, A. J. Gil, and J. Bonet, "Development of a cell centred upwind finite volume algorithm for a new conservation law formulation in structural dynamics," Computers & Structures, vol. 118, pp. 13–38, 2013.

1.3.1 Finite volume discretization

$$\int_{\Omega_0} \frac{\partial \boldsymbol{\mathcal{U}}}{\partial t}\,\mathrm{d}\Omega_0 + \int_{\partial\Omega_0} \boldsymbol{\mathcal{F}}_I N_I\,\mathrm{d}A_0 = \int_{\Omega_0} \boldsymbol{\mathcal{S}}\,\mathrm{d}\Omega_0$$

The finite volume discretization of the above equation over an M-sided cell “e’’ leads to the following semi-discrete form:

$$\frac{\mathrm{d}\boldsymbol{\mathcal{U}}_e}{\mathrm{d}t} = \boldsymbol{R}(\boldsymbol{\mathcal{U}}_e),$$

where,

$$\boldsymbol{R}(\boldsymbol{\mathcal{U}}_e) = -\frac{1}{\Omega_e}\sum_{f=1}^{M}(\boldsymbol{\mathcal{F}}_I N_I\,\Delta A)_f + \boldsymbol{\mathcal{S}}_e.$$

→ The flux vector is evaluated via a Riemann solver

$$\boldsymbol{\mathcal{F}}_I N_I = \boldsymbol{\mathcal{F}}_N = \begin{bmatrix} \boldsymbol{t} \\ \frac{1}{\rho_0}\boldsymbol{p}\otimes\boldsymbol{N} \end{bmatrix}$$

Source:C. H. Lee, A. J. Gil, and J. Bonet, “Development of a cell centred upwind finite volume algorithm for a new conservation law formulation in structural dynamics,” Computers & Structures, vol. 118, pp. 13–38, 2013.

1.3.1 Finite volume discretization

$$\frac{\mathrm{d}\boldsymbol{\mathcal{U}}_e}{\mathrm{d}t} = \boldsymbol{R}(\boldsymbol{\mathcal{U}}_e),$$

The single-step, two-stage, second-order Total Variation Diminishing (TVD) Runge-Kutta (RK) scheme for discretizing the time derivative, reads:

$$\begin{aligned}
\boldsymbol{\mathcal{U}}^{\star} &= \boldsymbol{\mathcal{U}}^{n} + \Delta t\, \boldsymbol{R}\left(\boldsymbol{\mathcal{U}}^{n}\right), \\
\boldsymbol{\mathcal{U}}^{\star\star} &= \boldsymbol{\mathcal{U}}^{\star} + \Delta t \boldsymbol{R}\left(\boldsymbol{\mathcal{U}}^{\star}\right), \\
\boldsymbol{\mathcal{U}}^{n+1} &= \frac{1}{2}\left(\boldsymbol{\mathcal{U}}^{n} + \boldsymbol{\mathcal{U}}^{\star\star}\right),
\end{aligned}
\qquad \Delta t = \frac{h_{\min}}{U_{\max}}, \qquad U_{\max} = \sqrt{\frac{\lambda + 2\mu}{\rho_0}}.$$

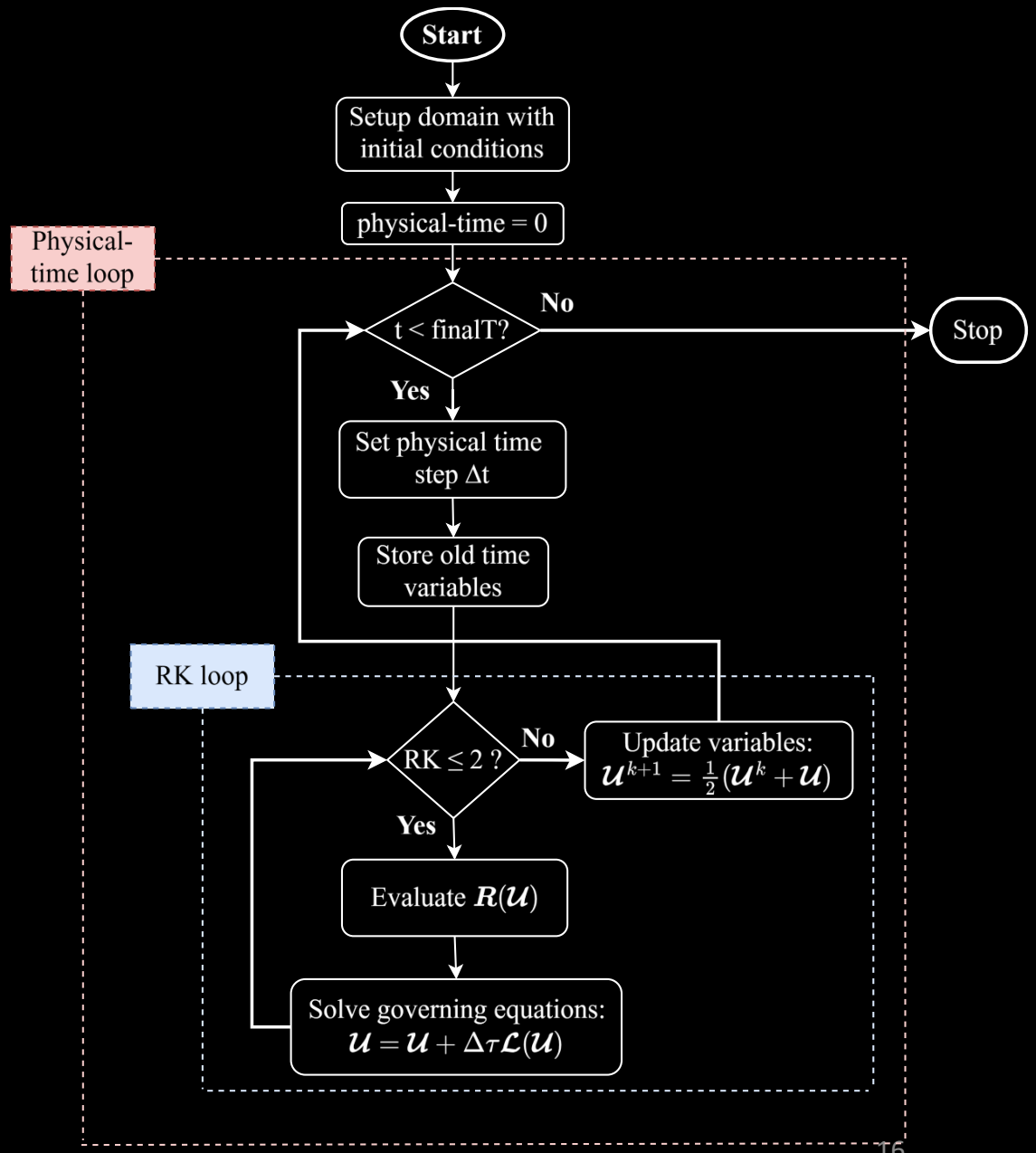
- **Explicit schemes** often require extremely **small** time steps for **stability**[8].
- This limitation can lead to an impractically **large** number of time steps[8].

Source:C. H. Lee, A. J. Gil, and J. Bonet, "Development of a cell centred upwind finite volume algorithm for a new conservation law formulation in structural dynamics," Computers & Structures, vol. 118, pp. 13–38, 2013.

[8] A. Jameson, "Time dependent calculations using multigrid, with applications to unsteady flows past airfoils and wings," in 10th Computational fluid dynamics conference, 1991, p. 1596.

1.3.1 Finite volume discretization

The single-step, two-stage, Runge-Kutta (RK) flow chart



1.3.2 Dual-time stepping

- The **dual-time stepping (DTS)** method transforms the **unsteady** problem in **physical time** into a **pseudo-steady** problem in a **fictitious** time domain.
- Adding a pseudo-time derivative gives:

$$\frac{\mathrm{d}\boldsymbol{\mathcal{U}}_e}{\mathrm{d}t} = \boldsymbol{R}(\boldsymbol{\mathcal{U}}_e) \longrightarrow \frac{\partial \boldsymbol{\mathcal{U}}_e}{\partial \tau} = -\frac{\partial \boldsymbol{\mathcal{U}}_e}{\partial t} + \boldsymbol{R}(\boldsymbol{\mathcal{U}}_e).$$

Source: A. Jameson, “Time dependent calculations using multigrid, with applications to unsteady flows past airfoils and wings,” in 10th Computational fluid dynamics conference, 1991, p. 1596.

1.3.2 Dual-time stepping

$$\frac{\partial \mathcal{U}_e}{\partial \tau} = -\frac{\partial \mathcal{U}_e}{\partial t} + \boldsymbol{R}(\mathcal{U}_e).$$

Discretize the physical time derivative using the implicit second-order backward differencing scheme[9]:

$$\frac{\partial \mathcal{U}}{\partial \tau} = -\frac{3\mathcal{U}^{n+1} - 4\mathcal{U}^{n} + \mathcal{U}^{n-1}}{2\Delta t} + \boldsymbol{R}\left(\mathcal{U}^{n+1}\right)$$

which can be rearranged as:

$$\frac{\partial \mathcal{U}_e}{\partial \tau} = \boldsymbol{\mathcal{L}}(\mathcal{U}_e)$$

Where, $$\boldsymbol{\mathcal{L}}(\mathcal{U}_e) = -\frac{3\mathcal{U}^{n+1} - 4\mathcal{U}^{n} + \mathcal{U}^{n-1}}{\Delta t} + \boldsymbol{R}\left(\mathcal{U}^{n+1}\right)$$

[9] S. Bhat and J. C. Mandal, "Contact preserving riemann solver for incompressible two-phase f lows," Journal of Computational Physics, vol. 379, pp. 173–191, 2019.

1.3.2 Dual-time stepping

$$\mathcal{L}(\mathcal{U}_e) = -\frac{3\mathcal{U}^{n+1} - 4\mathcal{U}^n + \mathcal{U}^{n-1}}{\Delta t} + \boldsymbol{R}\left(\mathcal{U}^{n+1}\right)$$

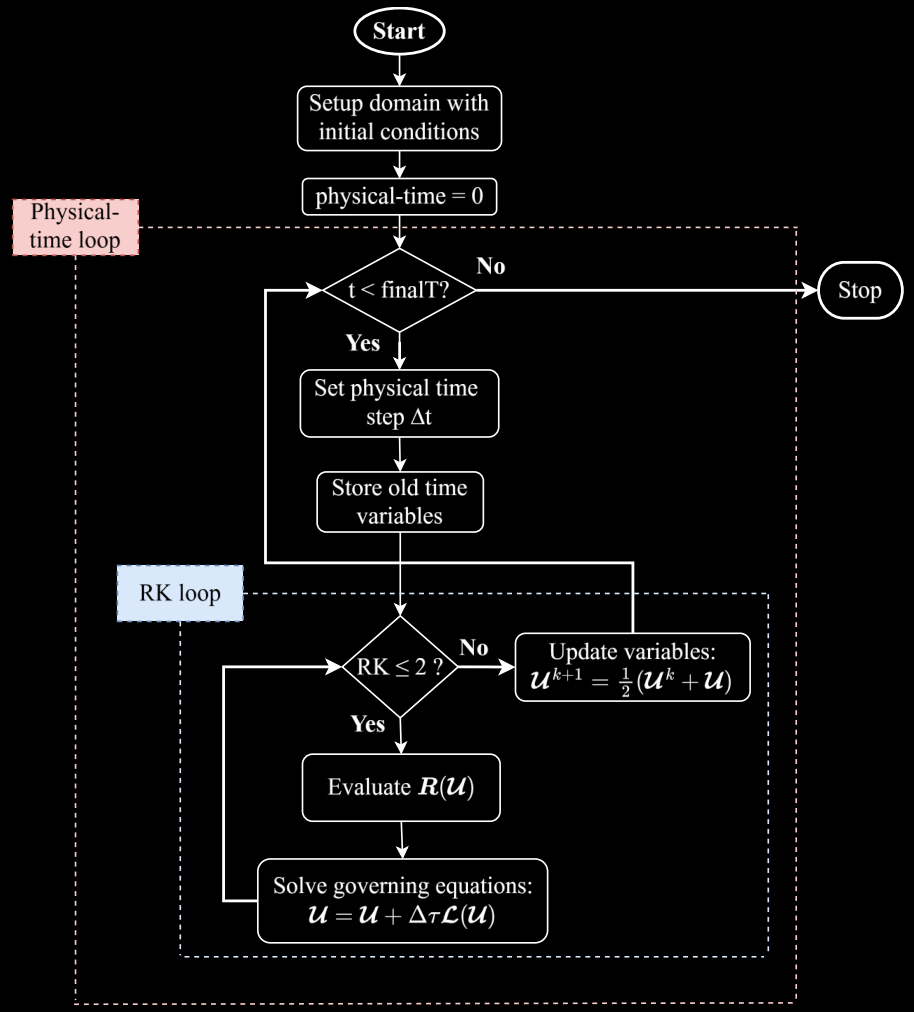
Then, the two-stage RK scheme is evaluated by:

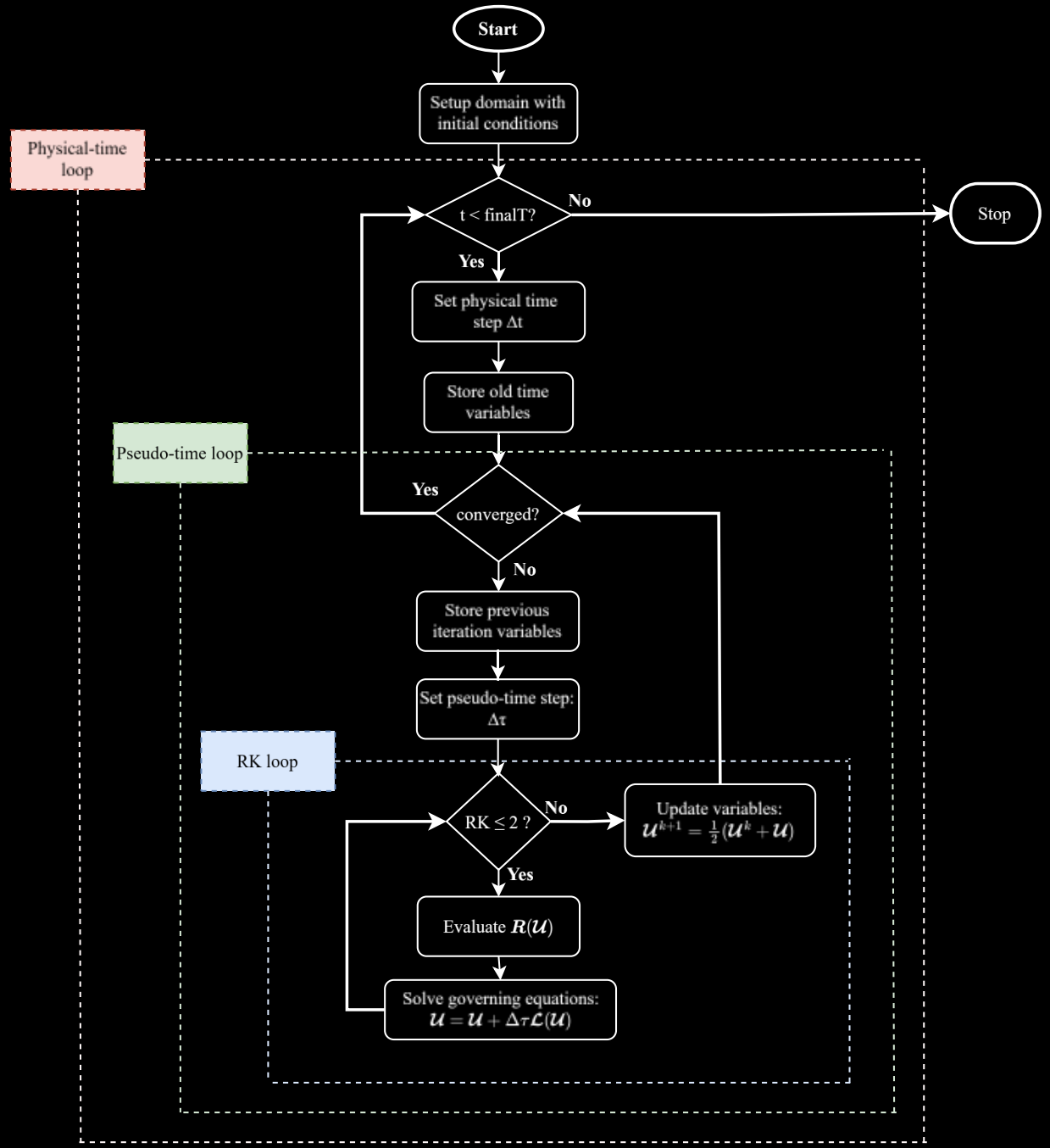
$$\begin{aligned}
\mathcal{U}^{\star} &= \mathcal{U}^k + \Delta\tau\,\mathcal{L}\left(\mathcal{U}^k\right), \\
\mathcal{U}^{\star\star} &= \mathcal{U}^{\star} + \Delta\tau\mathcal{L}\left(\mathcal{U}^{\star}\right), \\
\mathcal{U}^{k+1} &= \frac{1}{2}\left(\mathcal{U}^k + \mathcal{U}^{\star\star}\right),
\end{aligned}$$

$$\Delta\tau = \mathrm{CFL}\left[min\left(\Delta\hat{\tau}, \frac{2}{3}\Delta t\right)\right], \quad \Delta\hat{\tau} = \frac{h_{\min}}{U_{\max}}, \qquad U_{\max} = \sqrt{\frac{\lambda + 2\mu}{\rho_0}}.$$

Source:S. Bhat and J. C. Mandal, “Contact preserving riemann solver for incompressible two-phase f lows,” Journal of Computational Physics, vol. 379, pp. 173–191, 2019.

1.3.2 Dual-time stepping





Content:

- Theory
- Structure of explicitSolidDynamics Toolkit
- Structure of `solids4foam` Toolbox
- Implementing `explicitGodunovCC` Solid Model
- FSI Benchmark

Content:

1. Theory
2. Structure of explicitSolidDynamics Toolkit
    1. Directory of explicitSolidDynamics Toolkit
    2. Closer Look at `solidFoam.C`
    3. Closer Look at `gEqns.H`
3. Structure of `solids4foam` Toolbox
4. Implementing `explicitGodunovCC` Solid Model
5. FSI Benchmark

2.1 Directory of `explicitSolidDynamics` Toolkit

```
explicitSolidDynamics
|-- applications
|   |-- solvers
|       |--solidFoam
|       |--placticFoam
|   |-- utilities
|-- src
|   |-- boundaryConditions
|   |-- mathematics
|   |-- models
|   |-- schemes
|-- tutorials
|-- …
```

2.1 Directory of `explicitSolidDynamics` Toolkit

```
solidFoam
|-- Make
|-- compile
|-- createFields.H
|-- gEqns.H
|-- readControls.H
|-- riemannSolver.H
|-- solidFoam.C
|-- strongBCs.H
|-- updateVariables.H
```

2.2 Closer Look at `solidFoam.C`

```
#include "fvCFD.H"
#include "pointFields.H"
#include "operations.H"
#include "solidModel.H"
#include "mechanics.H"
#include "gradientSchemes.H"
#include "interpolationSchemes.H"
#include "angularMomentum.H"

int main(int argc, char *argv[])
{

… rest of the code
```

2.2 Closer Look at `solidFoam.C`

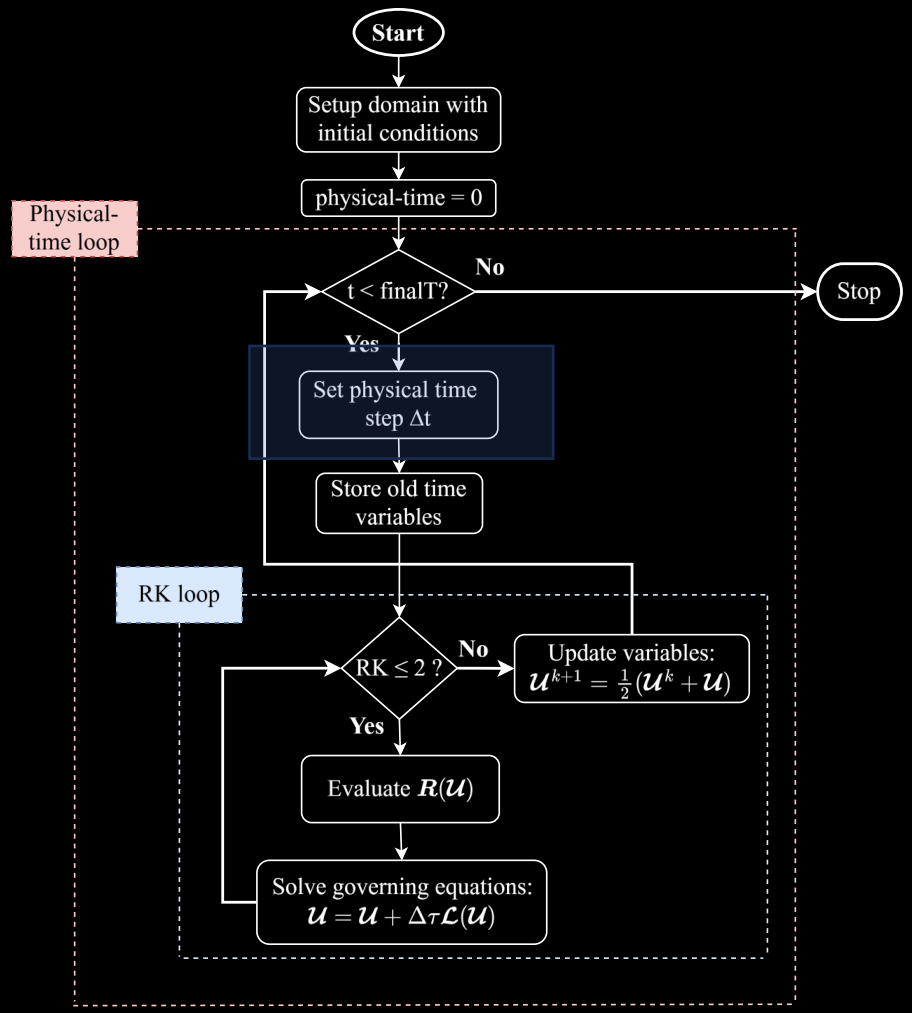
```
int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "readControls.H"
    #include "createFields.H"

    while (runTime.run())
    {
        mech.time(runTime, deltaT, max(Up_time));

… rest of the code
```

2.2 Closer Look at `solidFoam.C`

```
    while (runTime.run())
    {
        …

        forAll(RKstages, stage)
        {
            …
        }

         …
     }


… rest of the code
```

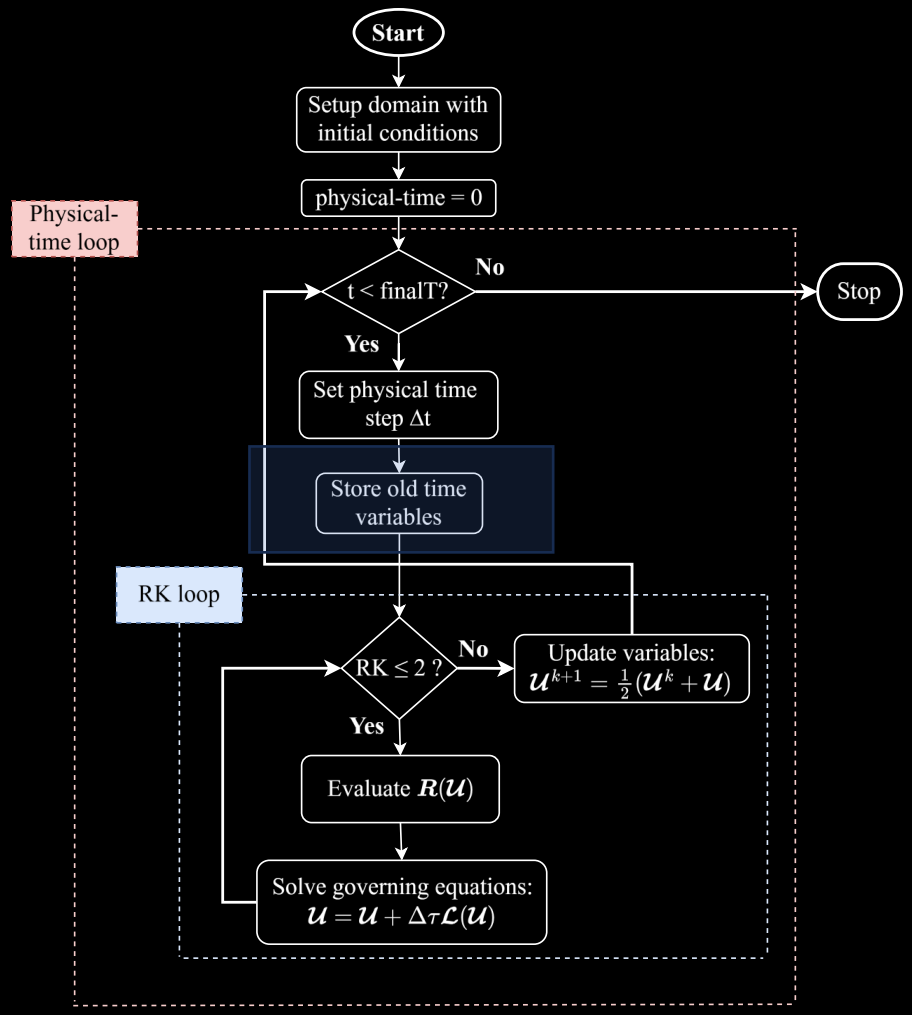
2.2 Closer Look at `solidFoam.C`

```
while (runTime.run())
{
    mech.time(runTime, deltaT, max(Up_time));

… rest of the code
```



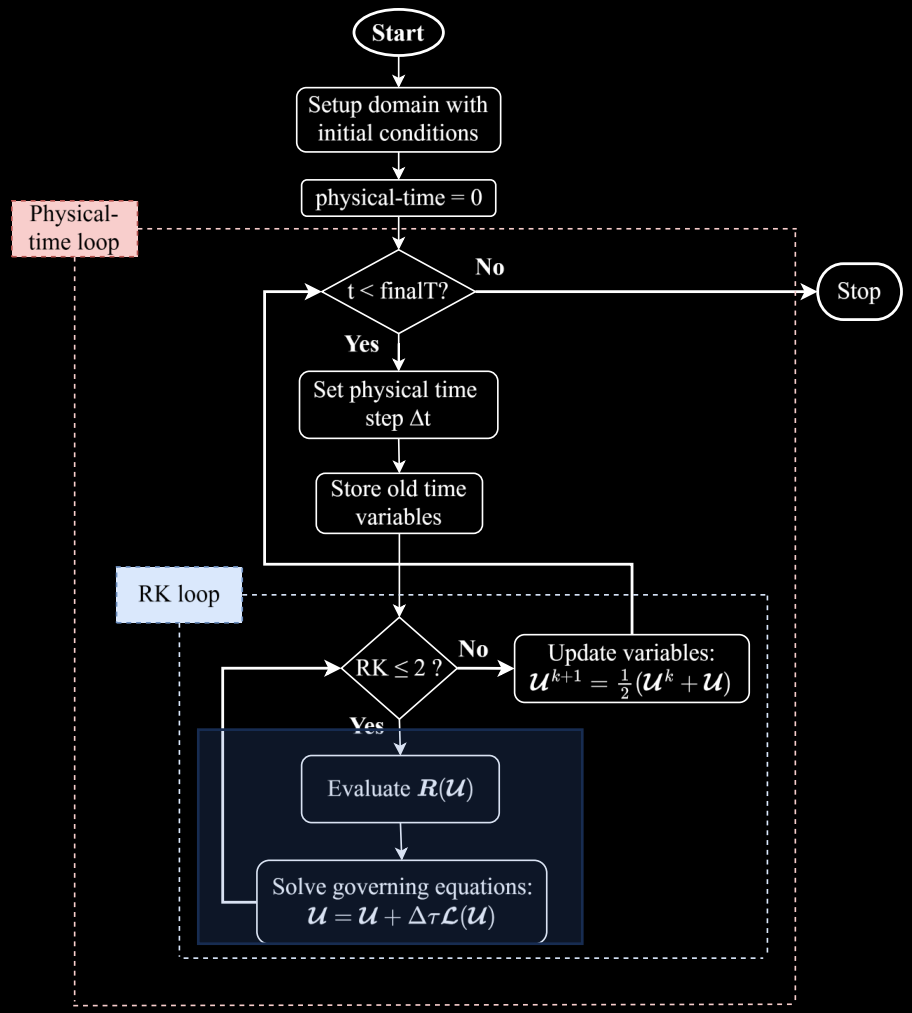
2.2 Closer Look at `solidFoam.C`

```
while (runTime.run())
{
    mech.time(runTime, deltaT, max(Up_time));

    lm.oldTime();
    F.oldTime();
    x.oldTime();
    xF.oldTime();
    xN.oldTime();


… rest of the code
```



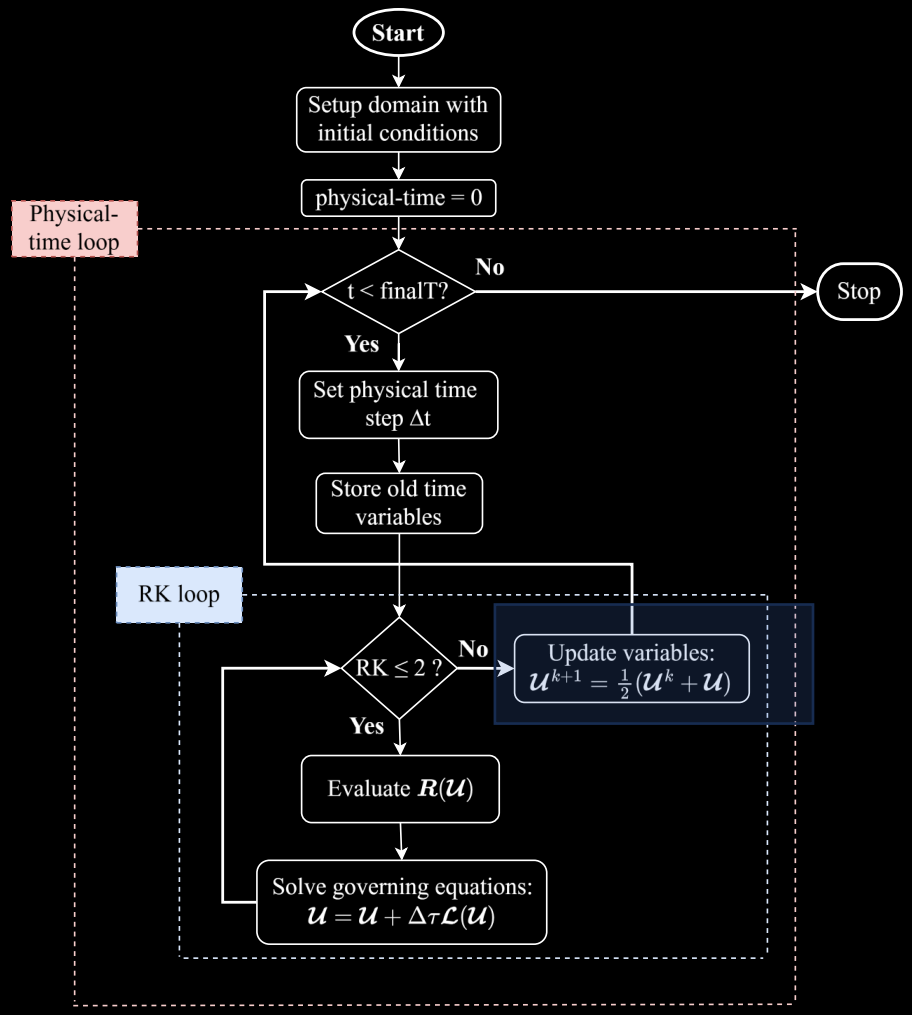
2.2 Closer Look at `solidFoam.C`

```
while (runTime.run())
{
    forAll(RKstages, stage)
    {
        #include "gEqns.H"

        if (RKstages[stage] == 0)
        {
            #include "updateVariables.H"
        }
    }

… rest of the code
```



2.2 Closer Look at `solidFoam.C`

```
while (runTime.run())
{

    lm = 0.5*(lm.oldTime() + lm);
    F = 0.5*(F.oldTime() + F);
    x = 0.5*(x.oldTime() + x);
    xF = 0.5*(xF.oldTime() + xF);
    xN = 0.5*(xN.oldTime() + xN);

    #include "updateVariables.H"


… rest of the code
```



Content:

1. Theory
2. Structure of explicitSolidDynamics Toolkit
3. Structure of `solids4foam` Toolbox
4. Implementing `explicitGodunovCC` Solid Model
5. FSI Benchmark

Content:

1. Theory
2. Structure of explicitSolidDynamics Toolkit
3. Structure of `solids4foam` Toolbox
    1. Directory of `solids4foam` Toolbox
    2. Closer Look at `solids4Foam.C`
    3. Closer Look at `solidModel Class`
    4. Closer Look at the `fluidSolidInterface` Class
4. Implementing `explicitGodunovCC` Solid Model
5. FSI Benchmark

3.1 Structure of `solids4foam` toolbox

```
solids4foam
|-- applications
|   |-- solvers
|       |--solid4Foam
|   |-- utilities
|   |-- ...
|-- src
|   |-- solids4FoamModels
|   |-- ...
|-- tutorials
|   |-- fluidSolidInteraction
|   |-- fluids
|   |-- solids
|   |-- ...
|-- …
```

3.1 Structure of `solids4foam` toolbox

```
solids4FoamModels
|-- physicsModel
|-- fluidModels
|-- solidModels
|-- fluidSolidInterfaces
|-- ...
```

3.3 Closer look at `solids4Foam.C`

```
#include "fvCFD.H"
#include "physicsModel.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
        #include "setRootCase.H"
        #include "createTime.H"
        #include "solids4FoamWriteHeader.H"

 … rest of the code
```

3.3 Closer look at `solids4Foam.C`

```
int main(int argc, char *argv[])
{
        … previous sild

    // Create the general physics class
    autoPtr<physicsModel> physics = physicsModel::New(runTime);

    while (runTime.run())
    {

 … rest of the code
```

3.3 Closer look at `solids4Foam.C`

```
    … previous sild

    while (runTime.run())
    {
        physics().setDeltaT(runTime);

        runTime++;

        if (physics().printInfo())
        {
            Info<< "Time = " << runTime.timeName() << nl << endl;
        }

        … rest of the code
    }

… rest of the code
```

3.3 Closer look at `solids4Foam.C`

```
    …previous sild

    while (runTime.run())
    {
        physics().setDeltaT(runTime);

        runTime++;

        if (physics().printInfo())
        {
            Info<< "Time = " << runTime.timeName() << nl << endl;
        }

        // Solve the mathematical model
        physics().evolve();

        … rest of the code
    }
```

3.4 Closer look at `solidModel` class

```
class solidModel
:
    public physicsModel,
    public regIOobject
{
    // Private data

        //- Solid properties dictionary
        mutable IOdictionary solidProperties_;

       //- Derived solidModel type
        const word type_;

       //- Point total displacement field
        pointVectorField pointD_;


 … rest of the code
```

3.4 Closer look at `solidModel` class

```
class solidModel
:
    public physicsModel,
    public regIOobject
{
    // Member Functions

        // Edit

            //- Update the size of the time-step
            virtual void setDeltaT(Time& runTime)
            {}

            //- Evolve the solid model
            virtual bool evolve() = 0;

 … rest of the code
```

3.4 Closer look at `solidModel` class

```
class solidModel
:
    public physicsModel,
    public regIOobject
{
    // Member Functions

        // Edit

            //- Set traction at specified patch
            virtual void setTraction
            (
                fvPatchVectorField& tractionPatch,
                const vectorField& traction
            );

 … rest of the code
```

3.4.1 Closer look at `linGeomTotalDispSolid` class

```
class linGeomTotalDispSolid
:
    public solidModel
{
    // Private data
        //- Implicit stiffness; coefficient of the Laplacian term
        const volScalarField impK_;

    // Private Member Functions

        //- Predict the fields for the next time-step based on the
        //  previous time-steps
        void predict();


 … rest of the code
```

3.4.1 Closer look at `linGeomTotalDispSolid` class

```
class linGeomTotalDispSolid
:
    public solidModel
{

public:

    // Member Functions

        //- Evolve the solid solver and solve the mathematical model
        virtual bool evolve();


 … rest of the code
```

3.5 Closer look at `fluidSolidInterface` class

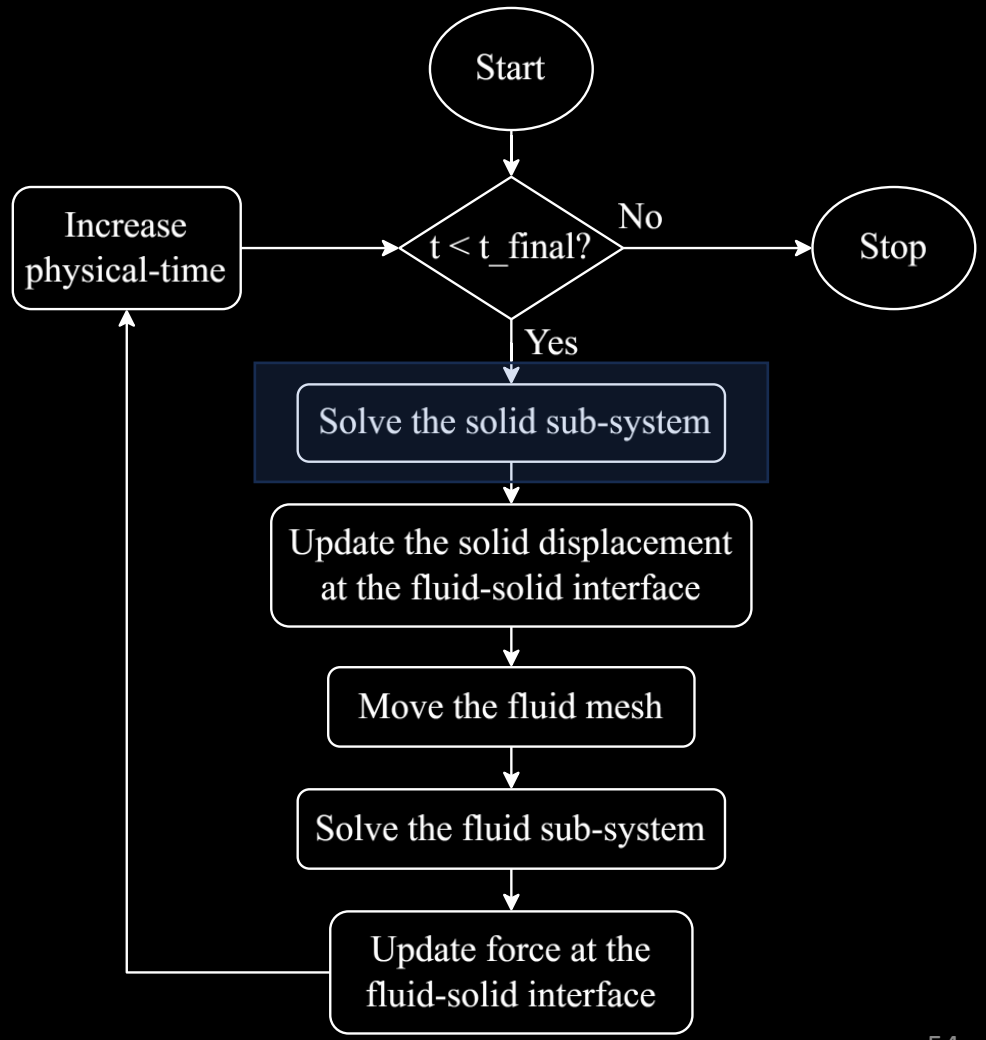
```
class fluidSolidInterface
:
    public physicsModel,
    public IOdictionary
{
    // Private data

        //- FSI properties dictionary
        dictionary& fsiProperties_;

        //- Flow solver
        autoPtr<fluidModel> fluid_;

        //- Solid solver
        autoPtr<solidModel> solid_;

 … rest of the code
```

3.5 Closer look at `fluidSolidInterface` class

```
class fluidSolidInterface
:
    public physicsModel,
    public IOdictionary
{
    // Member Functions

            //- Evolve the interface
            virtual bool evolve() = 0;

            //- Update interface force
            virtual void updateForce();

            //- Move fluid mesh
            virtual void moveFluidMesh();

 … rest of the code
```

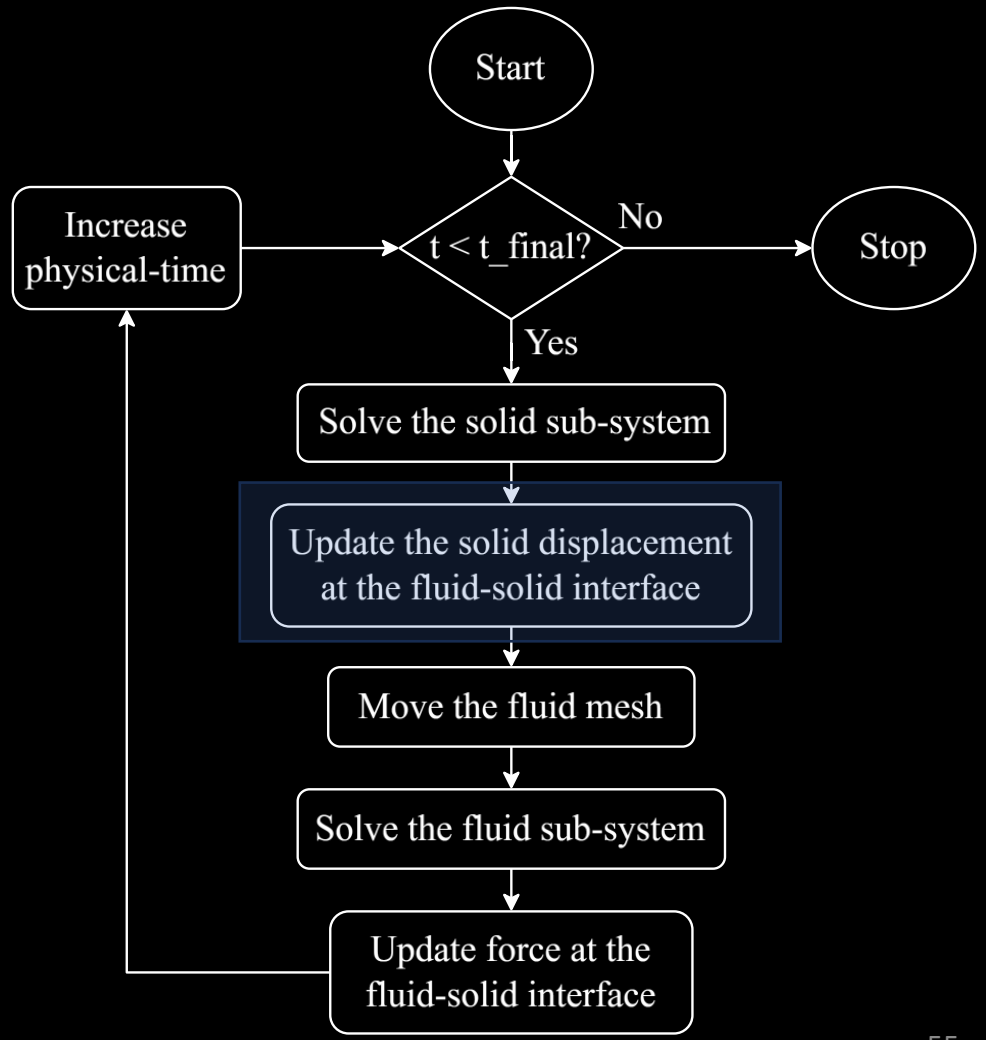
3.5 Closer look at `fluidSolidInterface` class

```
bool weakCouplingInterface::evolve()
{
    initializeFields();

    updateInterpolatorAndGlobalPatches();

    solid().evolve();

    updateWeakDisplacement();

    moveFluidMesh();

    fluid().evolve();

    updateForce();

    … rest of the code
```



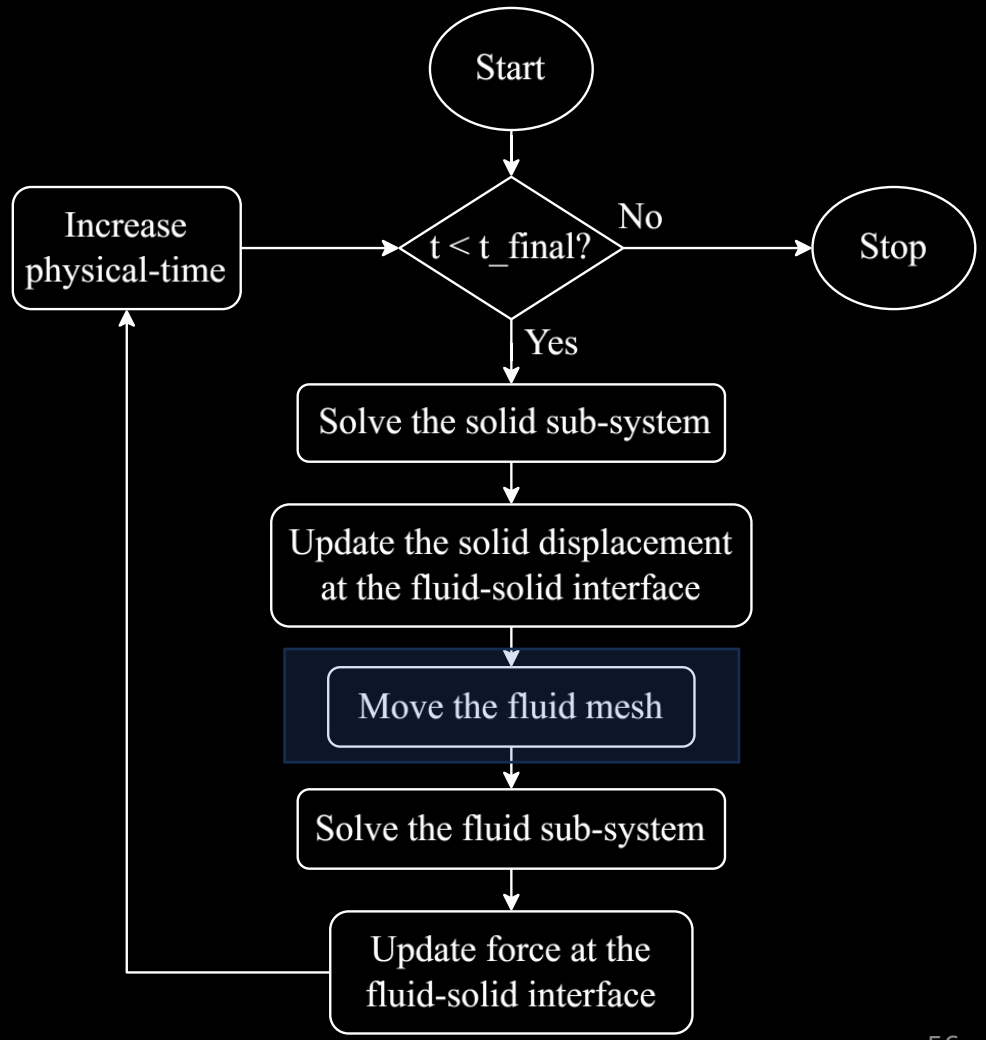
3.5 Closer look at `fluidSolidInterface` class

```
bool weakCouplingInterface::evolve()
{
    initializeFields();

    updateInterpolatorAndGlobalPatches();

    solid().evolve();

    updateWeakDisplacement();

    moveFluidMesh();

    fluid().evolve();

    updateForce();

    … rest of the code
```



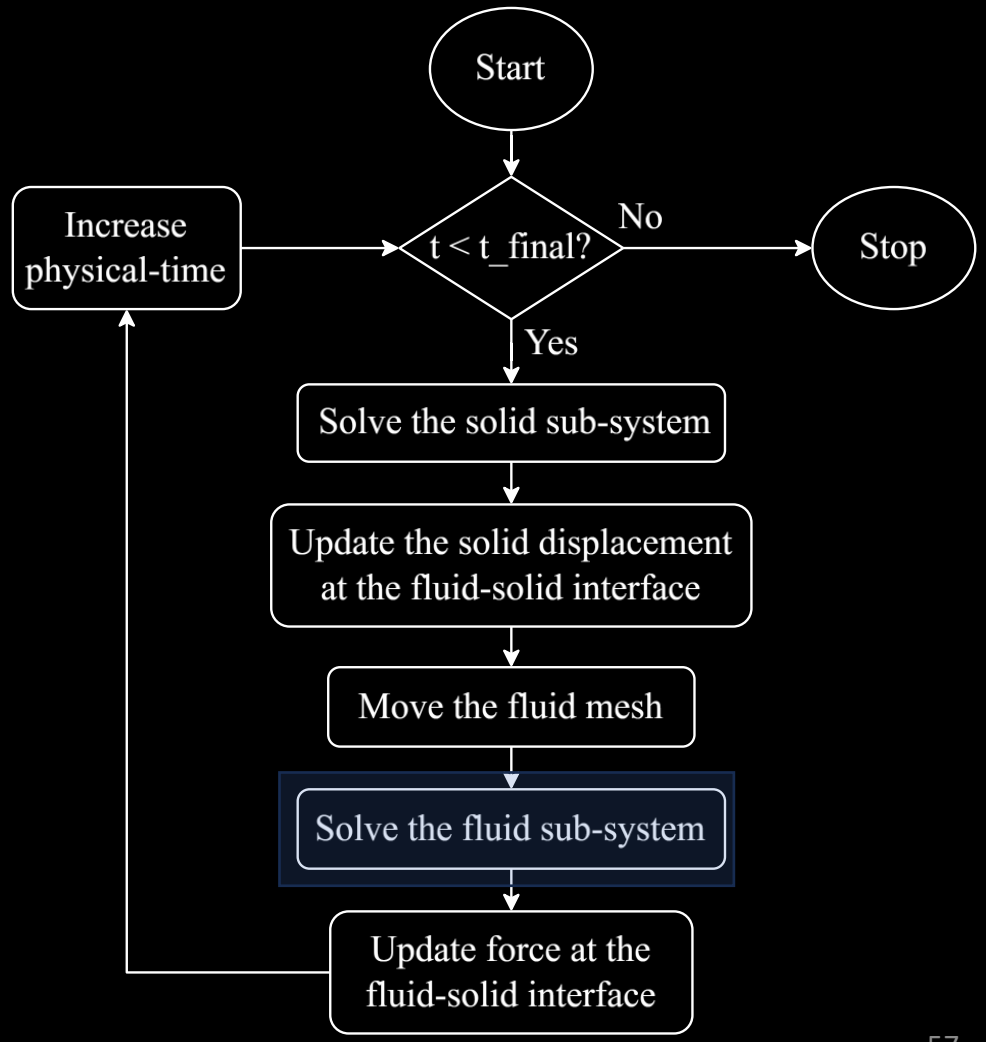
3.5 Closer look at `fluidSolidInterface` class

```
bool weakCouplingInterface::evolve()
{
    initializeFields();

    updateInterpolatorAndGlobalPatches();

    solid().evolve();

    updateWeakDisplacement();

    moveFluidMesh();

    fluid().evolve();

    updateForce();

    … rest of the code
```



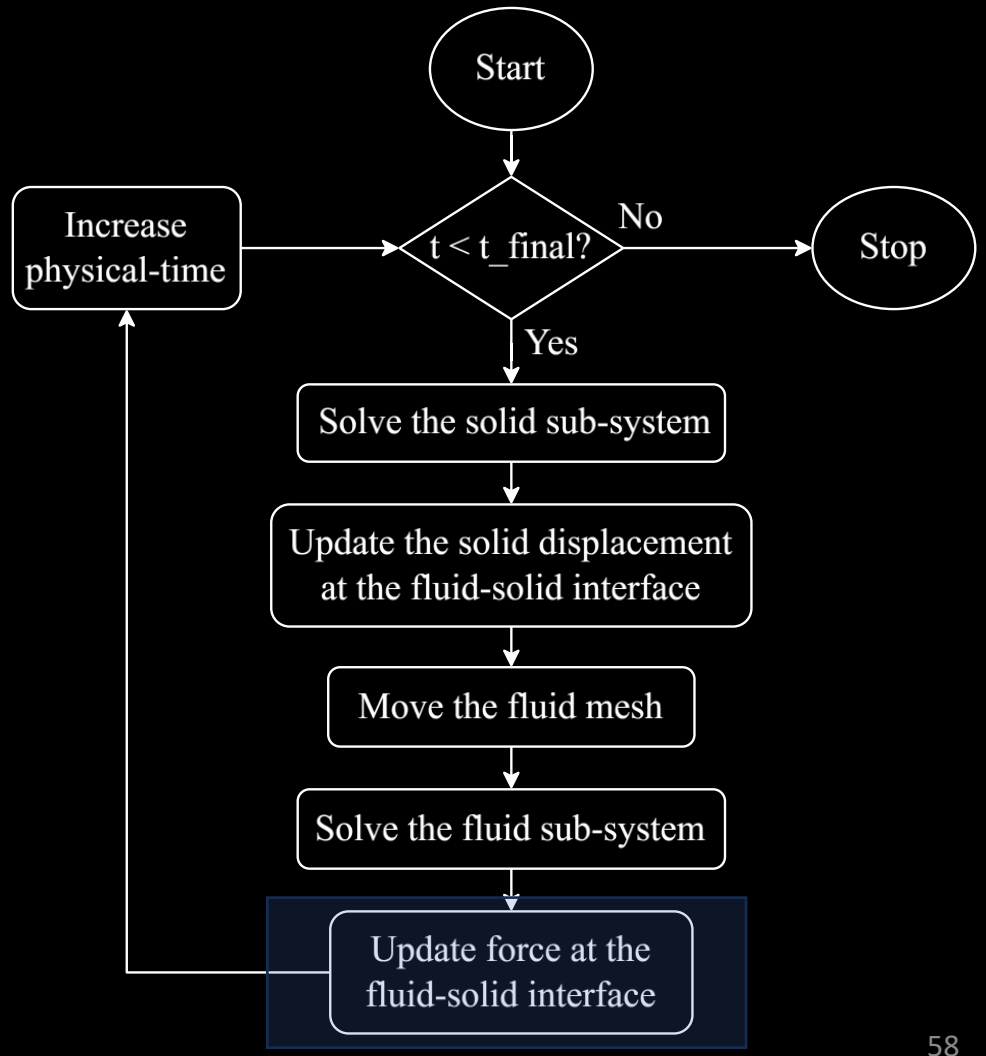
3.5 Closer look at `fluidSolidInterface` class

```
bool weakCouplingInterface::evolve()
{
    initializeFields();

    updateInterpolatorAndGlobalPatches();

    solid().evolve();

    updateWeakDisplacement();

    moveFluidMesh();

    fluid().evolve();

    updateForce();

    … rest of the code
```



3.5 Closer look at `fluidSolidInterface` class

```
bool weakCouplingInterface::evolve()
{
    initializeFields();

    updateInterpolatorAndGlobalPatches();

    solid().evolve();

    updateWeakDisplacement();

    moveFluidMesh();

    fluid().evolve();

    updateForce();

    … rest of the code
```



3.5.1 updateForce() function

```
void Foam::fluidSolidInterface::updateForce()
{
    for (label interfaceI = 0; interfaceI < nGlobalPatches_; interfaceI++)
    {
        .. Previous slide

        // Set traction on solid
        if (coupled())
        {
            solid().setTraction
            (
                interfaceI,
                solidPatchIndices()[interfaceI],
                solidZoneTotalTraction
            );
        }

… rest of the code
```

Content:

1. Theory
2. Structure of explicitSolidDynamics toolkit
3. Structure of `solids4foam` toolbox
4. Implementing `explicitGodunovCC` solid model
5. FSI benchmark

Content:

1. Theory
2. Structure of explicitSolidDynamics toolkit
3. Structure of `solids4foam` toolbox
4. Implementing `explicitGodunovCC` solid model
    1. Adding a new `solidModel` sub-class
    2. Wrapping `explicitSolidDynamics` in a class
    3. Enabling `explicitGodunovCC` with FSI
5. FSI benchmark

4 Implementing `explicitGodunovCC` solid model

Instructions to install solids4foam toolbox:

- `cd $HOME`
- `git clone--branch v2.1 https://github.com/solids4foam/solids4foam.git`
- `cd solids4foam && ./Allwmake-j && cd tutorials && ./Alltest`

Instructions to install explicitSoliddynamics toolbox:

- git clone--branch openfoam-v2012 https://github.com/philipcardiff/explicitSolidDynamics.git

4.1 Adding a new solidModel sub-class

We start by creating an empty testModelSolid:

- `cd ~/solids4foam/src/solids4FoamModels/solidModels`
- `mkdir testModelSolid`
- `cp -r linGeomTotalDispSolid/. testModelSolid`
- `cd testModelSolid/`
- `mv linGeomTotalDispSolid.C testModelSolid.C`
- `mv linGeomTotalDispSolid.H testModelSolid.H`
- `sed -i 's/linGeomTotalDispSolid/testModelSolid/g' testModelSolid.H`
- `sed -i 's/linearGeometryTotalDisplacement/testModel/g' testModelSolid.H`
- `sed -i 's/linGeomTotalDispSolid/testModelSolid/g' testModelSolid.C`

compile the new solid model:

- Add the following line to `solids4FoamModels/Make/files.openfoam` file:
  - `solidModels/testModelSolid/testModelSolid.C`
- Compile the library:
  - `cd ~/solids4foam/src/solids4FoamModels`
  - `./Allwmake`

4.1 Adding a new solidModel sub-class

Test the new solid model against one of the tutorial:

- Run the following commands:
  - `run`
  - `cp -r ~/solids4foam/tutorials/solids/linearElasticity/cantilever2d/segregatedCantilever2d/ .`
  - `cd segregatedCantilever2d`

- Set the `solidModel` to `testModel` in the `solidProperties` dictionary

- Add to the same file the solidModel sub-dictionary with default values:

```
testModelCoeffs
{
}
```

- Run the tutorial:
  - `./Allrun`

4.1.1Base `solidModel` sub-class

Clean up the testModelSolid.H file following these steps:

- Remove the following includes:

```
#include "volFields.H"
#include "surfaceFields.H"
#include "pointFields.H"
#include "uniformDimensionedFields.
```

- Remove all the Privet member data.

- Remove the flowing privet member functions:

```
void predict();
```

- Keep the rest of the member functions as they have a virtual prefix.

4.1.1 Base `solidModel` sub-class

Clean up the testModelSolid.C file following these steps:

- Remove the privet member function.
- Remove all member data definition in the constructor keeping only the following line:
  `solidModel(typeName, runTime, region)`
- Remove all the lines in the evolve() function
- Keep the `tractionBoundarySnGrad`, However, remove all the lines in the class implementation.
- Other steps are covered in the report.

Content:

1. Theory
2. Structure of explicitSolidDynamics Toolkit
3. Structure of `solids4foam` Toolbox
4. Implementing `explicitGodunovCC` Solid Model
    1. Adding a new `solidModel` Sub-Class
    2. Wrapping `explicitSolidDynamics` in a Class
    3. Enabling `explicitGodunovCC` with FSI
5. FSI Benchmark

Content:

1. Theory
2. Structure of explicitSolidDynamics Toolkit
3. Structure of `solids4foam` Toolbox
4. Implementing `explicitGodunovCC` Solid Model
    1. Adding a new `solidModel` sub-class
    2. Wrapping `explicitSolidDynamics` in a Class
        1. Adding explicitSolidDynamics libraries
        2. Adding member data
        3. Definition of evolve() function
    3. Enabling `explicitGodunovCC` with FSI
5. FSI Benchmark

4.2.1 Wrapping `explicitSolidDynamics` in a class

Transform the testModelSolid Class to the new solid model class we want (explicitGodunovCCSolid) by following these steps:

- `cd ~/solids4foam/src/solids4FoamModels/solidModels/`
- `mv testModelSolid/ explicitGodunovCCSolid`
- `cd explicitGodunovCCSolid`
- `mv testModelSolid.C explicitGodunovCCSolid.C`
- `mv testModelSolid.H explicitGodunovCCSolid.H`
- `sed-i 's/testModelSolid/explicitGodunovCCSolid/g' explicitGodunovCCSolid.H`
- `sed-i 's/testModel/explicitGodunovCC/g' explicitGodunovCCSolid.H`
- `sed-i 's/testModelSolid/explicitGodunovCCSolid/g' explicitGodunovCCSolid.C`

4.2.2 Adding `explicitSolidDynamics` Libraries

Start by copying the contents of the src directory into the directory of explicitGodunovCCSolid using the following commands:

- `cd ~/solids4foam/src/solids4FoamModels/solidModels/explicitGodunovCCSolid`
- `cp -r ~/explicitSolidDynamics/src/. .`

After that, we delete all the Make folders and compil scripts in each directory:

- `rm -r boundaryConditions/Make`
- `rm -r boundaryConditions/compile`
- `rm -r mathematics/Make`
- `rm -r mathematics/compile`
- `rm -r models/Make`
- `rm -r models/compile`
- `rm -r models/plasticityModel`
- `rm -r schemes/Make`
- `rm -r schemes/compile`

4.2.2 Adding `explicitSolidDynamics` Libraries

Finally, we include the libraries in explicitGodunovCCSolid.H :

```
#include "operations.H"
#include "solidMaterialModel.H"
#include "mechanics.H"
#include "gradientSchemes.H"
#include "interpolationSchemes.H"
#include "angularMomentum.H“
```

Now, we compile the code and check for errors

4.2.3 Adding Member Data

In the header file, add declarations for the member data within the private section of the class:

```
// Cell linear momentum
volVectorField lm_;
```

In the source file, define the fields and initialize them appropriately in the class constructor:

```
lm_
(
    IOobject
    (
        "lm",
        runTime.timeName(),
        mesh(),
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    mesh(),
    dimensionedVector("lm", dimensionSet(1,-2,-1,0,0,0,0), vector::zero)
),
```

4.2.3 Adding Member Data

All member data can be found in the accompanying files on the course web page.

Other necessary changes to member data to align with the `solids4foam` definitions can be found in the report.

4.2.3 Definition of evolve() function

- Recall that the main () function in solids4Foam and solidFoam include a while (runTime.run()) loop for the physical-time in both of them.
- The evolve() function in solids4Foam will take care of the model implementation.

```
while (runTime.run())
{
    physics().setDeltaT(runTime);

    …

    // Solve the mathematical model
    physics().evolve();

    … rest of the code
}
```

```
while (runTime.run())
{
    …

    forAll(RKstages, stage)
    {
        …
    }
     …
 }

… rest of the code
```

4.2.3 Definition of evolve() function

- Recall that the main () function in solids4Foam and solidFoam include a while (runTime.run()) loop for the physical-time in both of them.
- The evolve() function in solids4Foam will take care of the model implementation.

```
bool explicitGodunovCCSolid::evolve()
{

    return true;
}
```

```
while (runTime.run())
{
    …

    forAll(RKstages, stage)
    {
        …
    }
     …
 }

… rest of the code
```

4.2.3 Definition of evolve() function

- Recall that the main () function in solids4Foam and solidFoam include a while (runTime.run()) loop for the physical-time in both of them.
- The evolve() function in solids4Foam will take care of the model implementation.
- Move the content of while (runTime.run()) loop in the original main() function from solidFoam.C to the evolve() function.

```
bool explicitGodunovCCSolid::evolve()
{

    return true;
}
```

```
while (runTime.run())
{
    …

    forAll(RKstages, stage)
    {
        …
    }
     …
 }

… rest of the code
```

4.2.3 Definition of evolve() function

- Recall that the main () function in solids4Foam and solidFoam include a while (runTime.run()) loop for the physical-time in both of them.
- The evolve() function in solids4Foam will take care of the model implementation.
- Move the content of while (runTime.run()) loop in the original main() function from solidFoam.C to the evolve() function.

```
bool explicitGodunovCCSolid::evolve()
{
        …
        forAll(RKstages, stage)
        {
            …
        }
         …
    }

… rest of the code
    return true;
}
```

That's it!

Content:

1. Theory
2. Structure of explicitSolidDynamics Toolkit
3. Structure of `solids4foam` Toolbox
4. Implementing `explicitGodunovCC` Solid Model
    1. Adding a new `solidModel` Sub-Class
    2. Wrapping `explicitSolidDynamics` in a Class
    3. Enabling `explicitGodunovCC` with FSI
5. FSI Benchmark

Content:

4.3.1 Evaluating fields required by `solids4Foam`

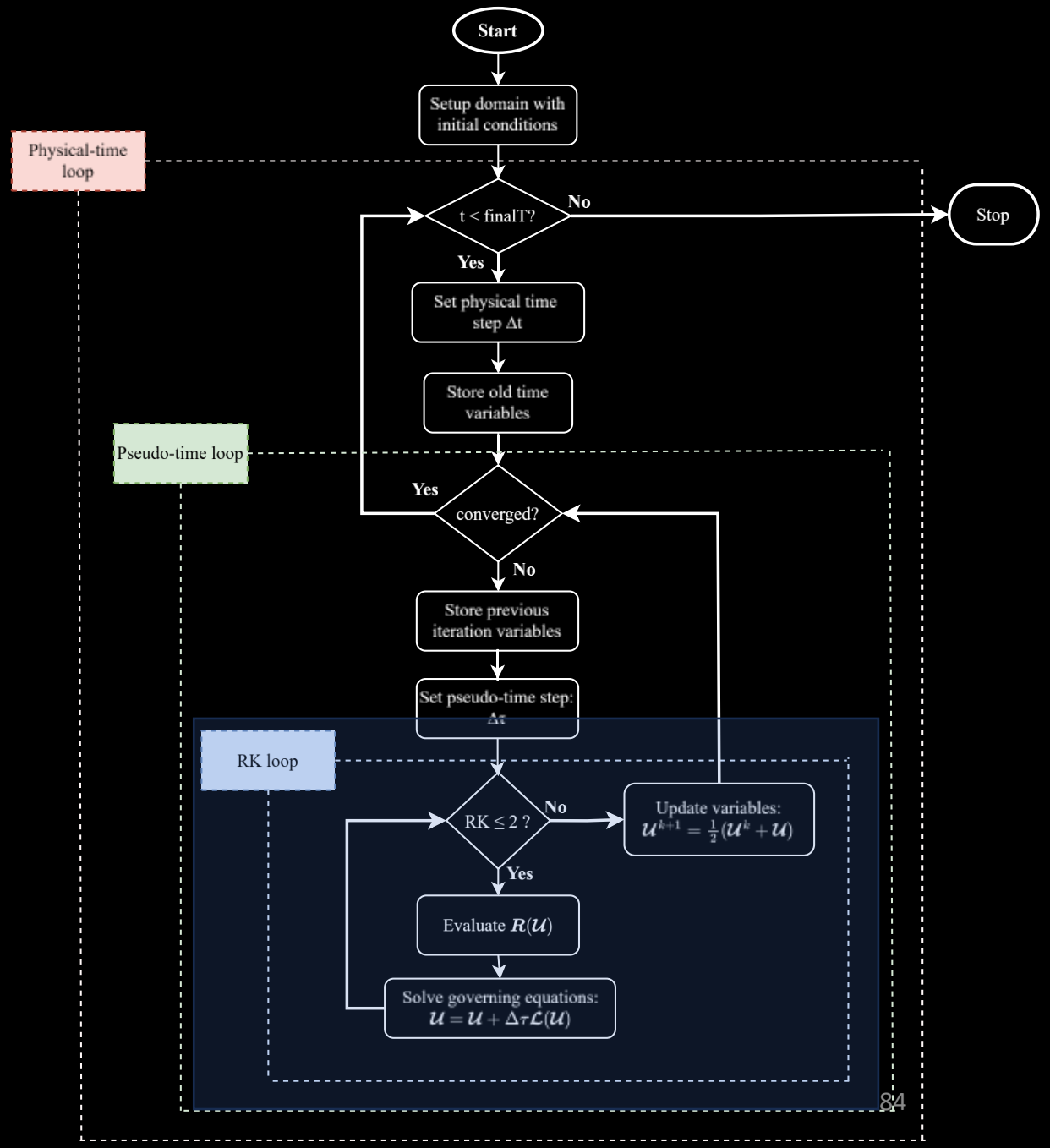
To integrate `explicitGodunovCC` fields into the broader `solids4foam` functionalities, the following fields are evaluated:

```
// Update the point displacement
pointD() = xN_ - XN_;

// Update the stress field
sigma() =  symm((1.0/J_)*(P_ & F_.T()));

// Increment of point displacement
pointDD() = pointD() - pointD().oldTime();
```

4.3.2 Dual-time stepping in explicitGodunovCC

Recall the evolve() function that we have updated earlier:

```
bool explicitGodunovCCSolid::evolve()
{
    …
    forAll(RKstages, stage)
    {
        …
    }
    …

… rest of the code
    return true;
}
```



4.3.2 Dual-time stepping in explicitGodunovCC

We introduce a do-while correction loop:

```
bool explicitGodunovCCSolid::evolve()
{
        …
        forAll(RKstages, stage)
        {
            …
        }
         …

… rest of the code
    return true;
}
```

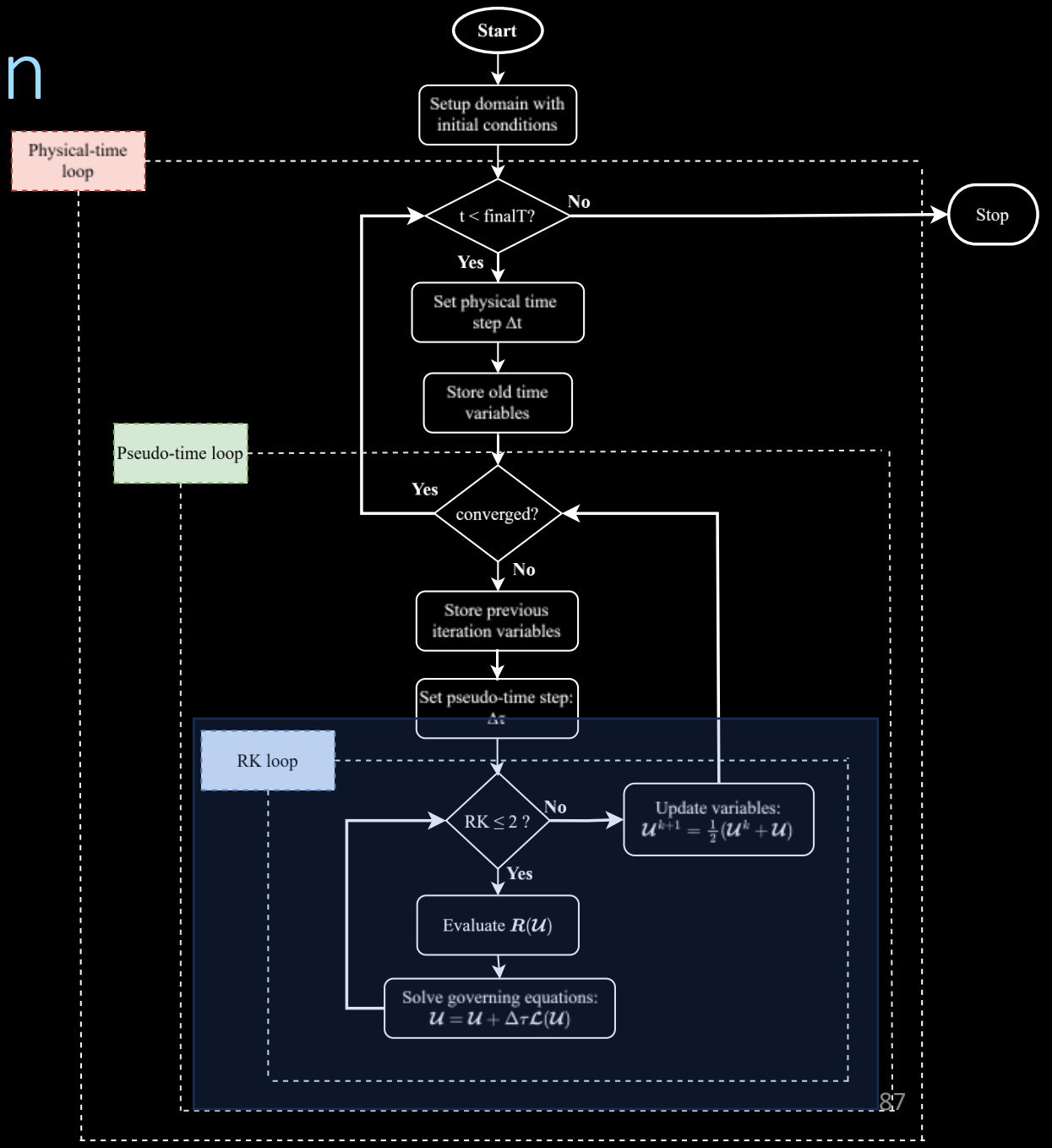
```
// Pseudo time loop (Correction loop)
do
{
        …
}
while
(
    !converged
        (
            iCorr,
            pDeltaT_,
            lm_
        )
 && ++iCorr < nCorr()
);
```

4.3.2 Dual-time stepping in explicitGodunovCC

```
bool explicitGodunovCCSolid::evolve()
{
    do
    {
        forAll(RKstages, stage)
        {
            …
        }
         …
    }
    while
    (
        !converged (iCorr, pDeltaT_, lm_ )

     && ++iCorr < nCorr()
    );

    return true;
}
```

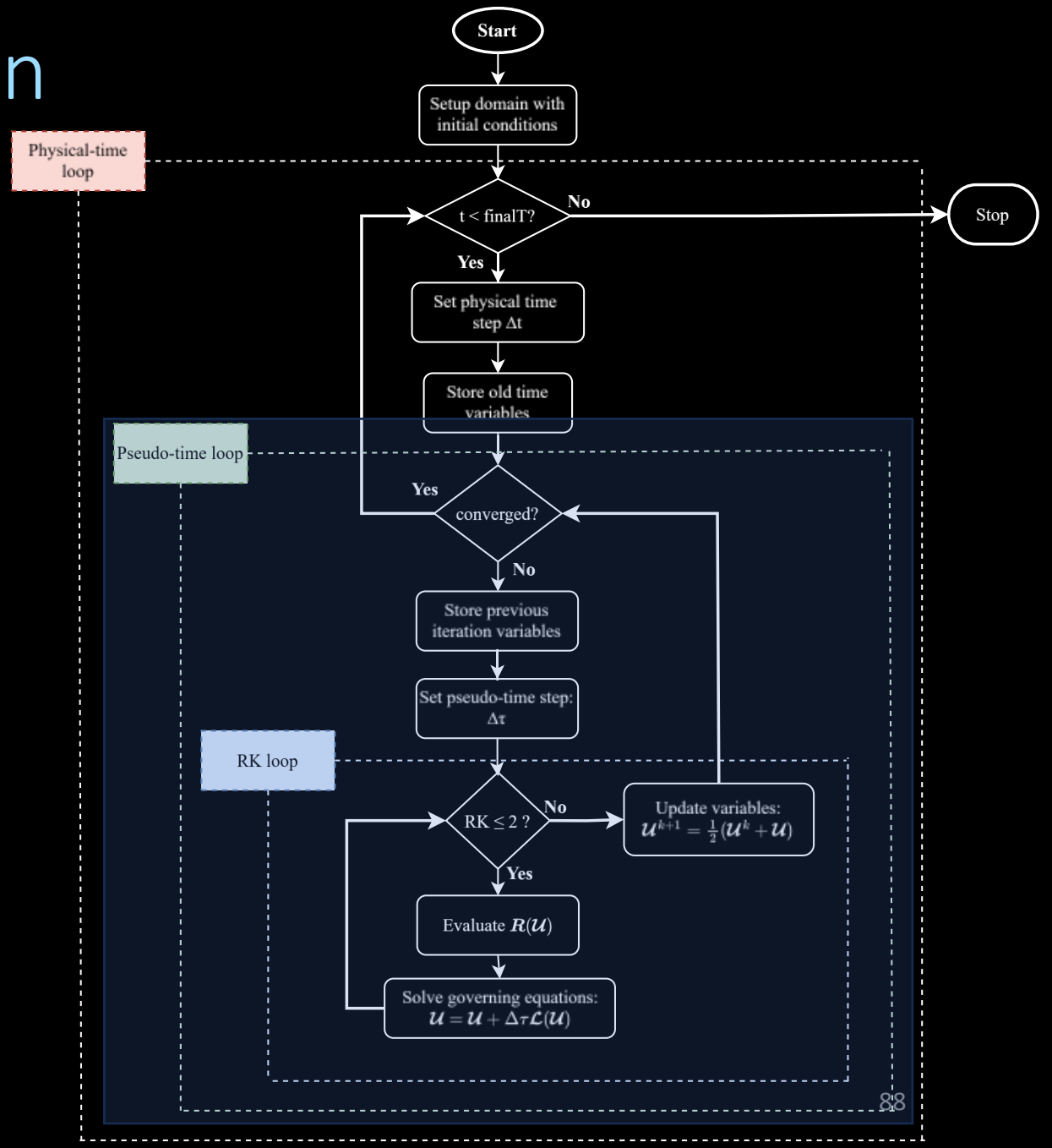
4.3.2 Dual-time stepping in explicitGodunovCC

```
bool explicitGodunovCCSolid::evolve()
{
    do
    {
        forAll(RKstages, stage)
        {
            …
        }
        …
    }
    while
    (
        !converged (iCorr, pDeltaT_, lm_ )

     && ++iCorr < nCorr()
    );

    return true;
}
```



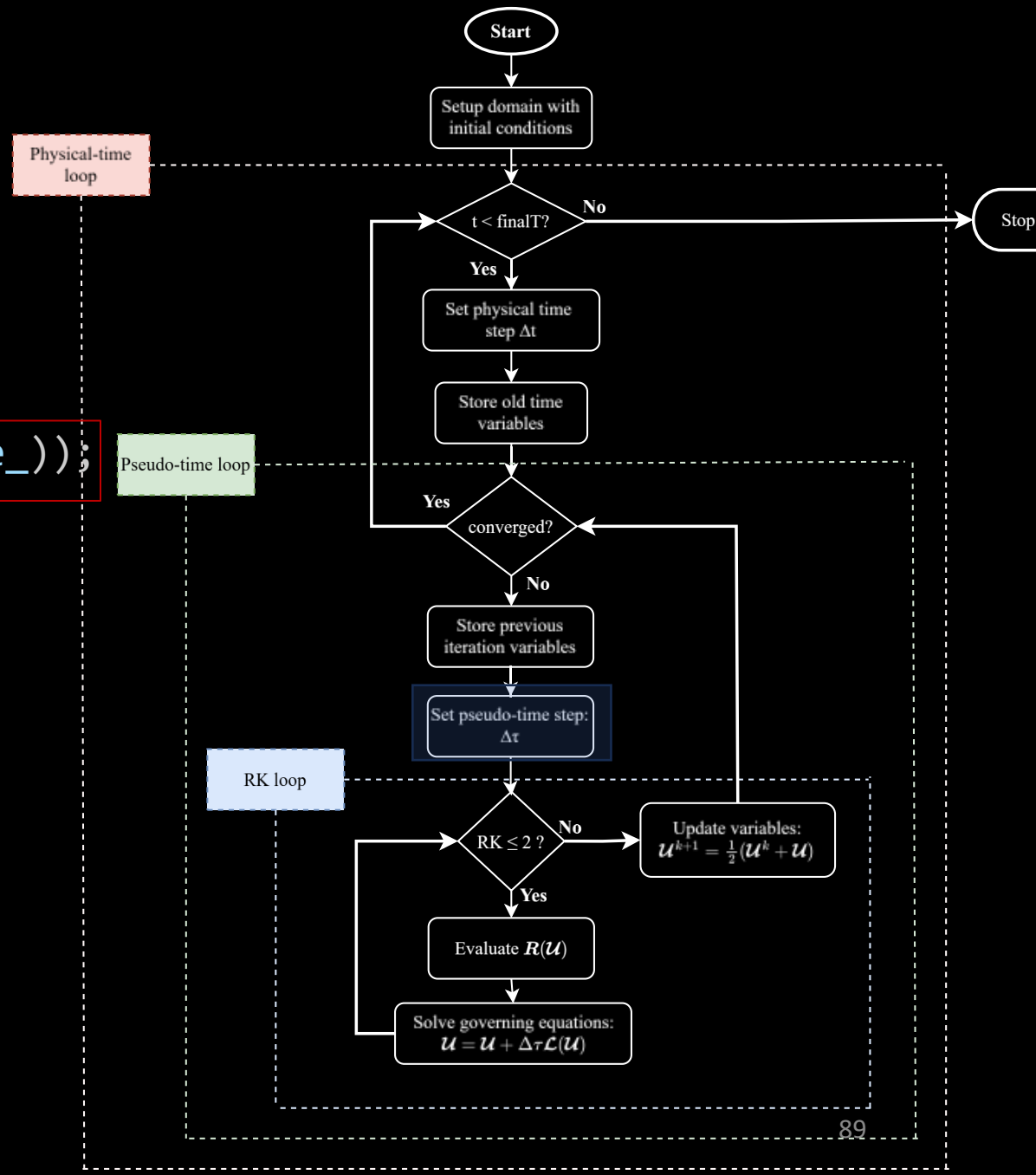
4.3.2 Dual-time stepping in explicitGodunovCC

```
bool explicitGodunovCCSolid::evolve()
{
    do
    {
        forAll(RKstages, stage)
        {
            …
        }
         …
    }
    while
    (
        !converged (iCorr, pDeltaT_, lm_ )

     && ++iCorr < nCorr()
    );

    return true;
}
```



4.3.2 Dual-time stepping in explicitGodunovCC

```
bool explicitGodunovCCSolid::evolve()
{
    do
    {
        mech_.time(runTime_, pDeltaT_, max(Up_time_));

        forAll(RKstages, stage)
        {
            …
        }
         …
    }
    while
    (
        !converged (iCorr, pDeltaT_, lm_ )

     && ++iCorr < nCorr()
    );
```



4.3.3 Interface coupling

Recall that the updateForce() function invoked by the fluidSolidInterface model which calls the setTraction() function

```
void Foam::fluidSolidInterface::updateForce()
{
    for (label interfaceI = 0; interfaceI < nGlobalPatches_; interfaceI++)
    {
        …
        // Set traction on solid
        if (coupled())
        {
            solid().setTraction
            (
                interfaceI,
                solidPatchIndices()[interfaceI],
                solidZoneTotalTraction
            );
        }
```

4.3.3 Interface coupling

The `setTraction()` function is defined `solidModel` as follows

```
void Foam::solidModel::setTraction
(
    const label interfaceI,
    const label patchID,
    const vectorField& faceZoneTraction
)
{
    const vectorField patchTraction
    (
        globalPatches()[interfaceI].globalFaceToPatch(faceZoneTraction)
    );

    setTraction(solutionD().boundaryFieldRef()[patchID], patchTraction);

}
```

It calls an overloaded `setTraction()` function that takes a reference to the displacement field at the fluid solid interface, and the interface traction.

4.3.3 Interface coupling

The second call of `setTraction()` function is defined in `solidModel` as follows

```
void Foam::solidModel::setTraction
(
    fvPatchVectorField& tractionPatch,
    const vectorField& traction
)
{
    if (tractionPatch.type() == solidTractionFvPatchVectorField::typeName)
    {
        solidTractionFvPatchVectorField& patchD =
            refCast<solidTractionFvPatchVectorField>(tractionPatch);

        patchD.traction() = traction;
    }
    else if

… rest of the code
```

It assigns a given traction vector field `traction` to a patch vector field `tractionPatch`.

Notice that `solidTractionFvPatchVectorField` Is a boundary field class

4.3.3 Interface coupling

Redefine the `setTraction()` function to include the linear momentum and traction fields.

First add the functions declaration to the `explicitGodunovCCSolid.H`:

```
//- Set traction at specified patch
virtual void setTraction
(
    fvPatchVectorField& tractionPatch,
    const vectorField& traction
);

//- Set traction at specified patch
virtual void setTraction
(
    const label interfaceI,
    const label patchID,
    const vectorField& faceZoneTraction
);
```

4.3.3 Interface coupling

The first call:

```
void Foam::solidModel::setTraction
(
    const label interfaceI,
    const label patchID,
    const vectorField& faceZoneTraction
)
{
    const vectorField patchTraction
    (
        globalPatches()[interfaceI].globalFaceToPatch(faceZoneTraction)
    );

    setTraction(solutionD().boundaryFieldRef()[patchID], patchTraction);

}
```

4.3.3 Interface coupling

Redefined first call:

```
void explicitGodunovCCSolid::setTraction
(
    const label interfaceI,
    const label patchID,
    const vectorField& faceZoneTraction
)
{
    const vectorField patchTraction
    (
        globalPatches()[interfaceI].globalFaceToPatch(faceZoneTraction)
    );

    volVectorField& lm_b = mesh().lookupObjectRef<volVectorField>("lm_b");
    volVectorField& t_b = mesh().lookupObjectRef<volVectorField>("t_b");

    setTraction(lm_b.boundaryFieldRef()[patchID], patchTraction);
    setTraction(t_b.boundaryFieldRef()[patchID], patchTraction);
```

4.3.3 Interface coupling

The second call:

```
void Foam::solidModel::setTraction
(
    fvPatchVectorField& tractionPatch,
    const vectorField& traction
)
{
    if (tractionPatch.type() == solidTractionFvPatchVectorField::typeName)
    {
        solidTractionFvPatchVectorField& patchD =
            refCast<solidTractionFvPatchVectorField>(tractionPatch);

        patchD.traction() = traction;
    }
    else if

… rest of the code
```

4.3.3 Interface coupling

Redefined second call:

```
void explicitGodunovCCSolid::setTraction
(
    fvPatchVectorField& tractionPatch,
    const vectorField& traction
)
{
    if (tractionPatch.type() == explicitSolidTractionTractionFvPatchVectorField::typeName)
    {
        explicitSolidTractionTractionFvPatchVectorField& patchTraction =
            refCast<explicitSolidTractionTractionFvPatchVectorField>(tractionPatch);

        patchTraction.traction() = traction;

    }
… rest of the code
```

Notice that created
`explicitSolidTractionTractionFvPatchVectorField`
Is a boundary field class.
Similarly, for the linear momentum.

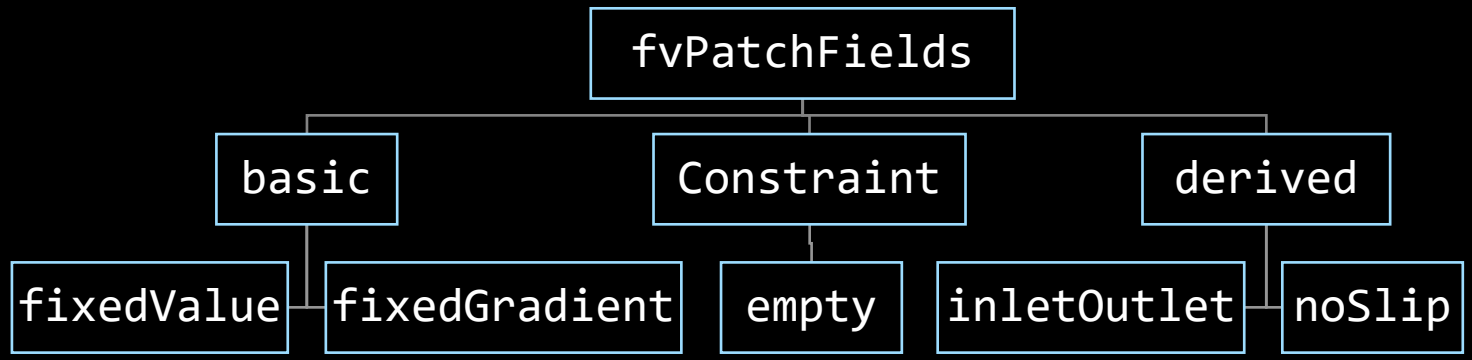
4.3.4 Coupling boundary conditions

We need to create new classes for the boundary condition for linear momentum and traction fields

`explicitSolidTractionTractionFvPatchVectorField`
`explicitSolidTractionLinearMomentumFvPatchVectorField`

4.3.4 Coupling boundary conditions

In OpenFOAM the boundary conditions are categorized as following:



- The solidTraction BC is a derived type of the basic class fixedGradient.
- This is because, in a displacement-based formulation, the displacement BC is of the Neumann type.
- Conversely, for the momentum-deformation-based formulation, the BCs are of the Dirichlet type, which requires setting a fixed value for the field at the boundary.
- The steps required for this implementation is found in the report.

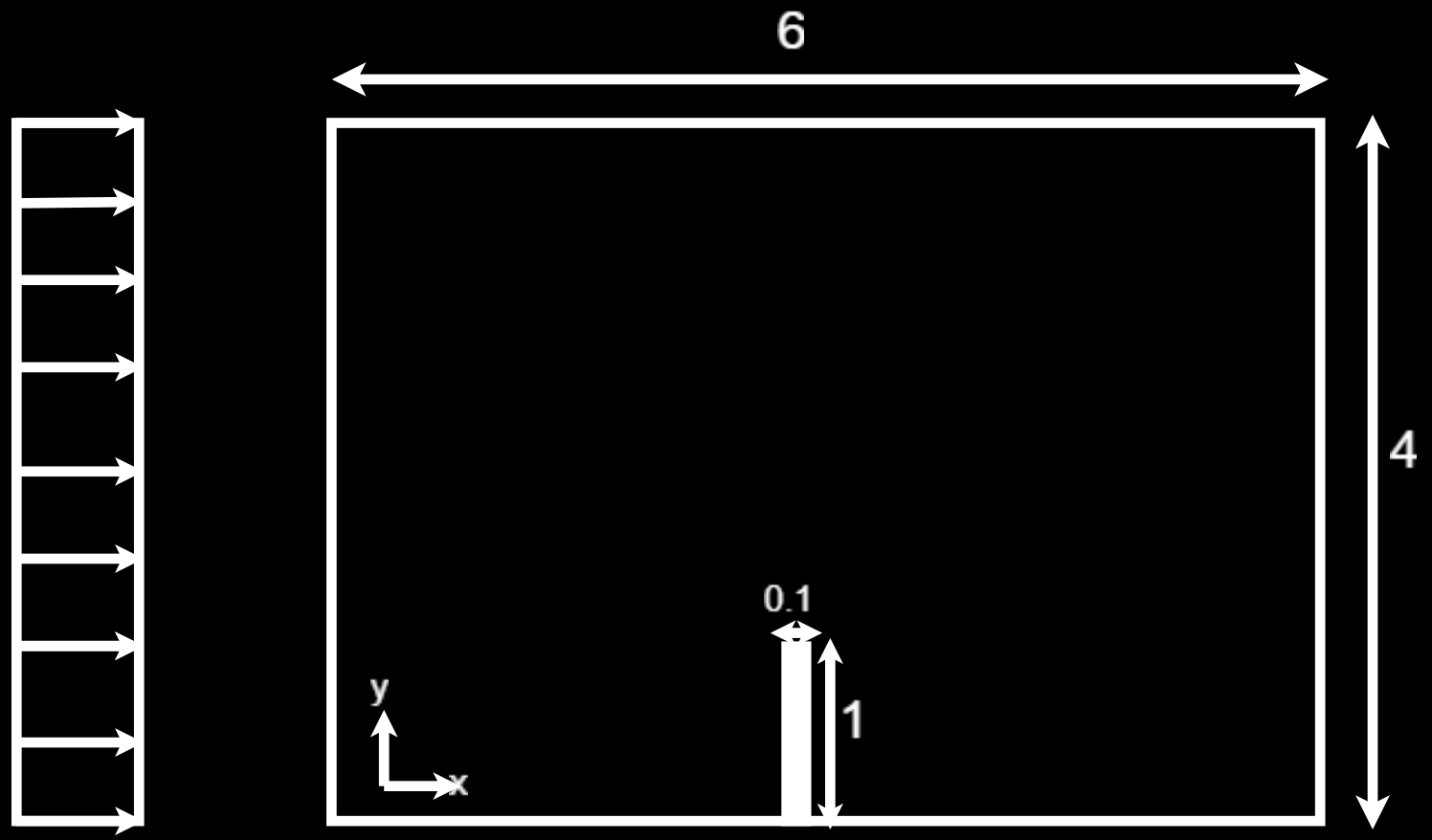
Content:

1. Theory
2. Structure of explicitSolidDynamics Toolkit
3. Structure of `solids4foam` Toolbox
4. Implementing `explicitGodunovCC` Solid Model
5. FSI Benchmark

Content:

1. Theory
2. Structure of explicitSolidDynamics Toolkit
3. Structure of `solids4foam` Toolbox
4. Implementing `explicitGodunovCC` Solid Model
5. FSI Benchmark
    1. Verification: perpendicular flap in cross flow

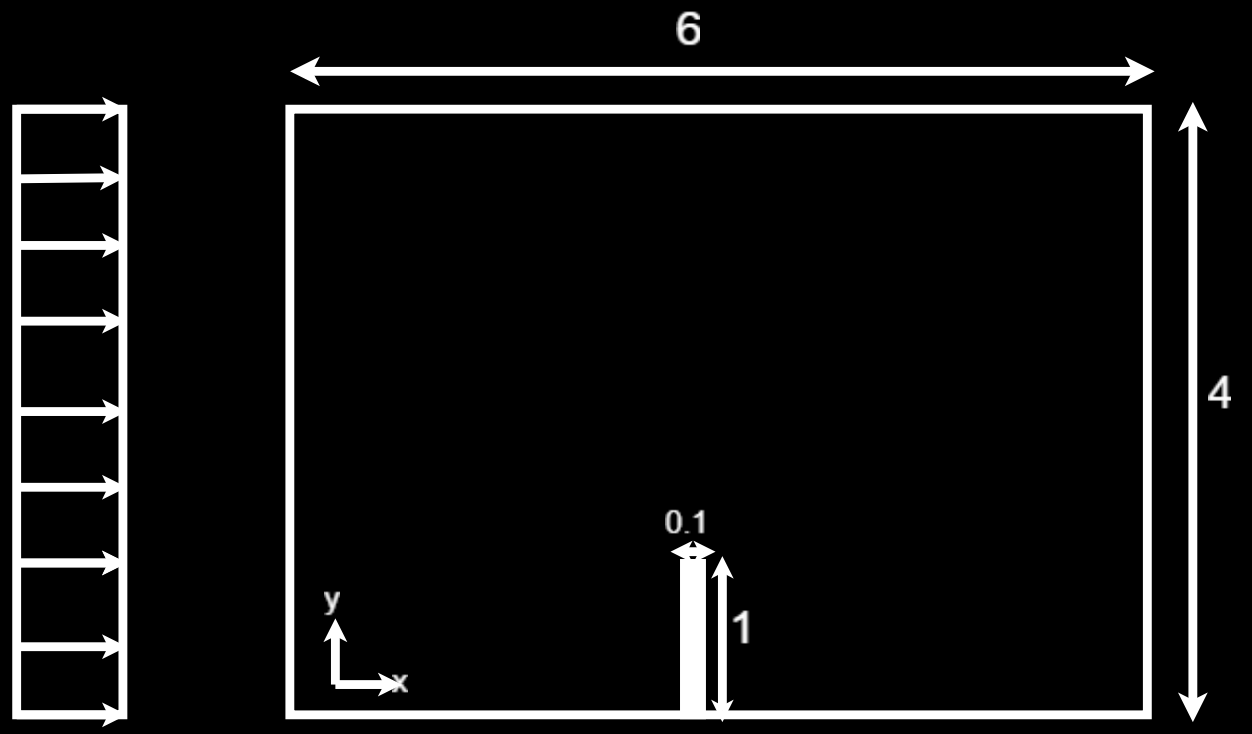
5 Verification: perpendicular flap in cross flow



Source:A. Zanella, L. Abergo, F. Caccia, M. Morelli, and A. Guardone, “Towards an open-source framework for fluid–structure interaction using su2, mbdyn and precice,” Journal of Computational and Applied Mathematics, vol. 429, p. 115211, 2023.

5 Verification: perpendicular flap in cross flow

Property	Value	Units
Fluid Properties	Laminar, incompressible	
Fluid Density	1	kg/m3
Kinematic Viscosity	1	m^2/s
Inlet Velocity	10	m/s
Solid Properties	Linear elastic	
Solid Density	3 X 10^3	kg/m^3
Young's Modulus	4 X 10^6	kg/ms^2
Poisson's Ratio	0.3	-



Source:A. Zanella, L. Abergo, F. Caccia, M. Morelli, and A. Guardone, "Towards an open-source framework for fluid–structure interaction using su2, mbdyn and precice," Journal of Computational and Applied Mathematics, vol. 429, p. 115211, 2023.

5 Verification: perpendicular flap in cross flow

`Solids4foam` FSI case directory structure:

5 Verification: perpendicular flap in cross flow

`Solids4foam` FSI case directory structure:

```
perpendicularFlap
├── 0
├── Allclean
├── Allrun
├── constant
└── system
```

5 Verification: perpendicular flap in cross flow

```
perpendicularFlap
├── 0
│   ├── fluid
│   └── solid
├── Allclean
├── Allrun
├── constant
└── system
```

5 Verification: perpendicular flap in cross flow

```
perpendicularFlap
├── 0
│   ├── fluid
│   └── solid
├── Allclean
├── Allrun
├── constant
│   ├── fluid
│   ├── fsiProperties
│   ├── g
│   ├── physicsProperties
│   └── solid
└── system
```

5 Verification: perpendicular flap in cross flow

```
perpendicularFlap
├── 0
│   ├── fluid
│   └── solid
├── Allclean
├── Allrun
├── constant
│   ├── fluid
│   ├── fsiProperties
│   ├── g
│   ├── physicsProperties
│   └── solid
└── system
    ├── controlDict
    ├── decomposeParDict
    ├── fluid
    ├── functions
    ├── functions.openfoam
    └── solid
```

5 Verification: perpendicular flap in cross flow

```
perpendicularFlap
├── 0
│   ├── fluid
│   └── solid
│       ├── D
│       └── pointD
├── Allclean
├── Allrun
├── constant
└── system
```

5 Verification: perpendicular flap in cross flow

```
perpendicularFlap
├── 0
│   ├── fluid
│   └── solid
│       ├── lmN
│       ├── lm_b
│       └── t_b
├── Allclean
├── Allrun
├── constant
└── system
```

```
boundaryField
{
    interface
    {
        type            explicitSolidTractionLinearMomentum;
        traction        uniform ( 0 0 0 );
        pressure        uniform 0;
        value           uniform (0 0 0);
    }
    bottom
    {
        type    fixedValue;
        value   uniform (0 0 0);
    }
    frontAndBack
    {
        type            empty;
    }

}
```

5 Verification: perpendicular flap in cross flow

```
perpendicularFlap
├── 0
│   ├── fluid
│   └── solid
│       ├── lmN
│       ├── lm_b
│       └── t_b
├── Allclean
├── Allrun
├── constant
└── system
```

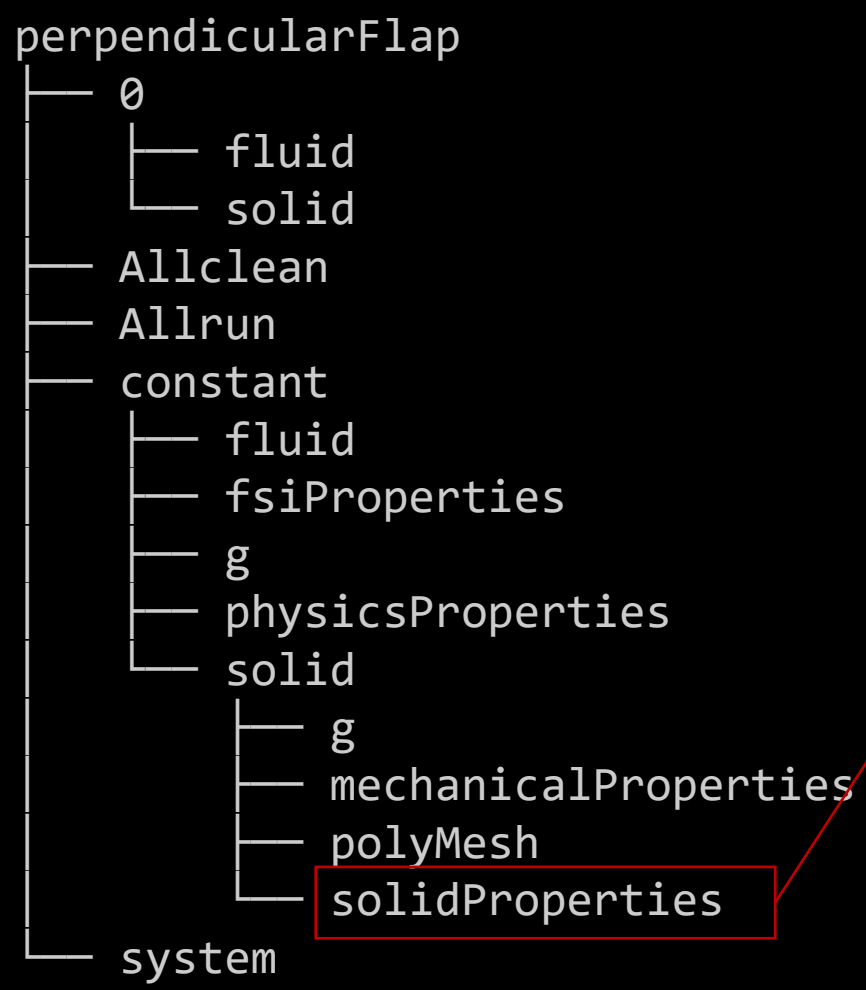
```
internalField   uniform (0 0 0);

boundaryField
{
    interface
    {
        type            explicitSolidTractionTraction;
        traction        uniform ( 0 0 0 );
        pressure        uniform 0;
        value           uniform (0 0 0);
    }
    bottom
    {
        type    movingTraction;
        value   uniform (0 0 0);
    }
    frontAndBack
    {
        type            empty;
    }
}
```

5 Verification: perpendicular flap in cross flow

```
perpendicularFlap
├── 0
│   ├── fluid
│   └── solid
├── Allclean
├── Allrun
├── constant
│   ├── fluid
│   ├── fsiProperties
│   ├── g
│   ├── physicsProperties
│   └── solid
└── system
```

```
// type    fluid;
// type    solid;
type    fluidSolidInteraction;
```

5 Verification: perpendicular flap in cross flow




```
solidModel    explicitGodunovCC;;

explicitGodunovCCCoeffs
{
   // Maximum number of momentum correctors
   nCorrectors           10000;

   // Solution tolerance for displacement
   solutionTolerance      1e-4;

   angularMomentumConservation     no;

   incompressiblilityCoefficient   1.0;

   dampingCoeff  dampingCoeff [ 0 0 -1 0 0 0 0 ] 0;
   // Write frequency for the residuals
   infoFrequency          100;
}
```

5 Verification: perpendicular flap in cross flow

```
perpendicularFlap
├── 0
├── Allclean
├── Allrun
├── constant
└── system
    ├── controlDict
    ├── decomposeParDict
    ├── fluid
    ├── functions
    ├── functions.openfoam
    └── solid
```

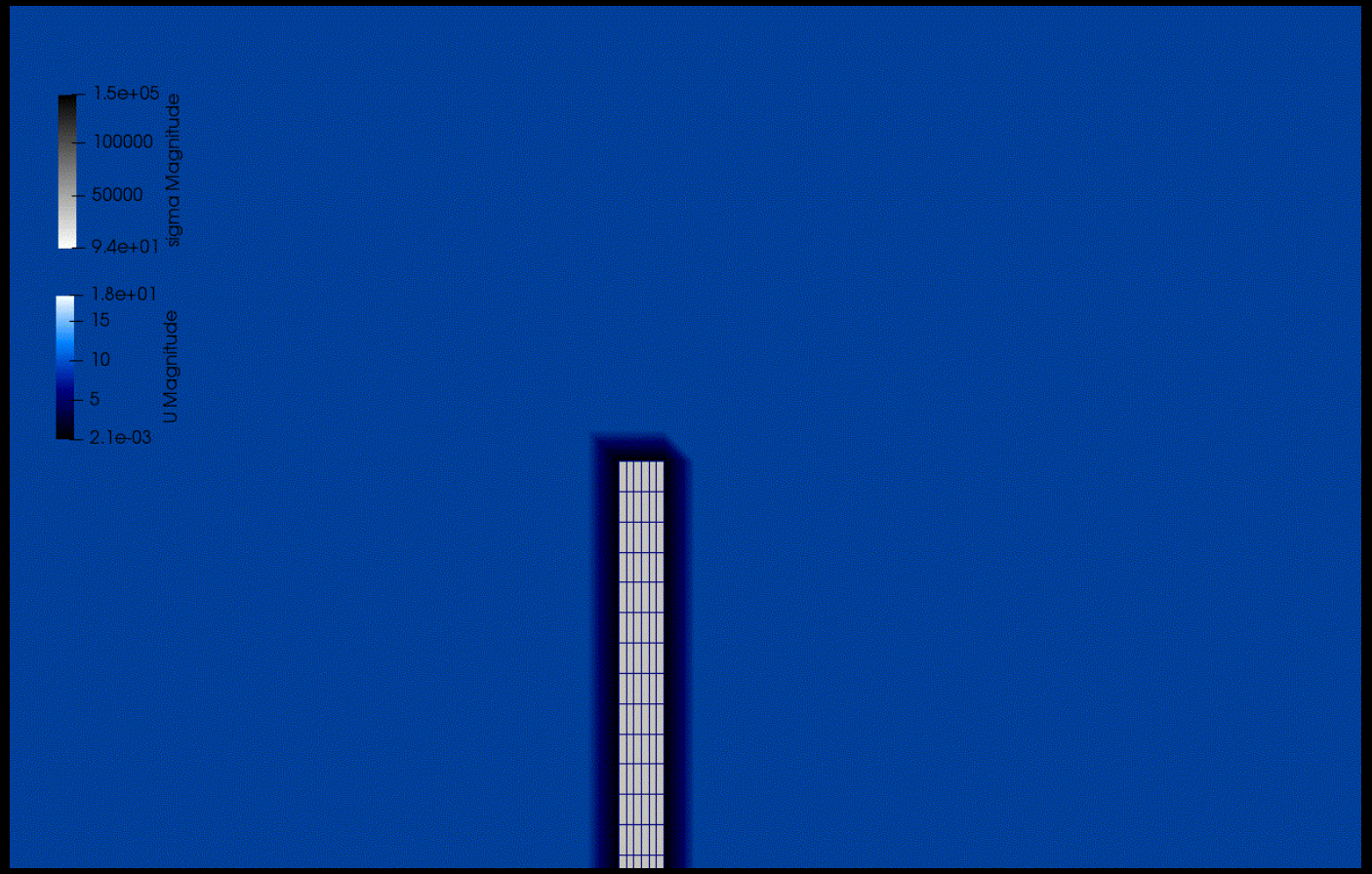
```
deltaT          0.01;//physical-time step
```

5 Verification: perpendicular flap in cross flow

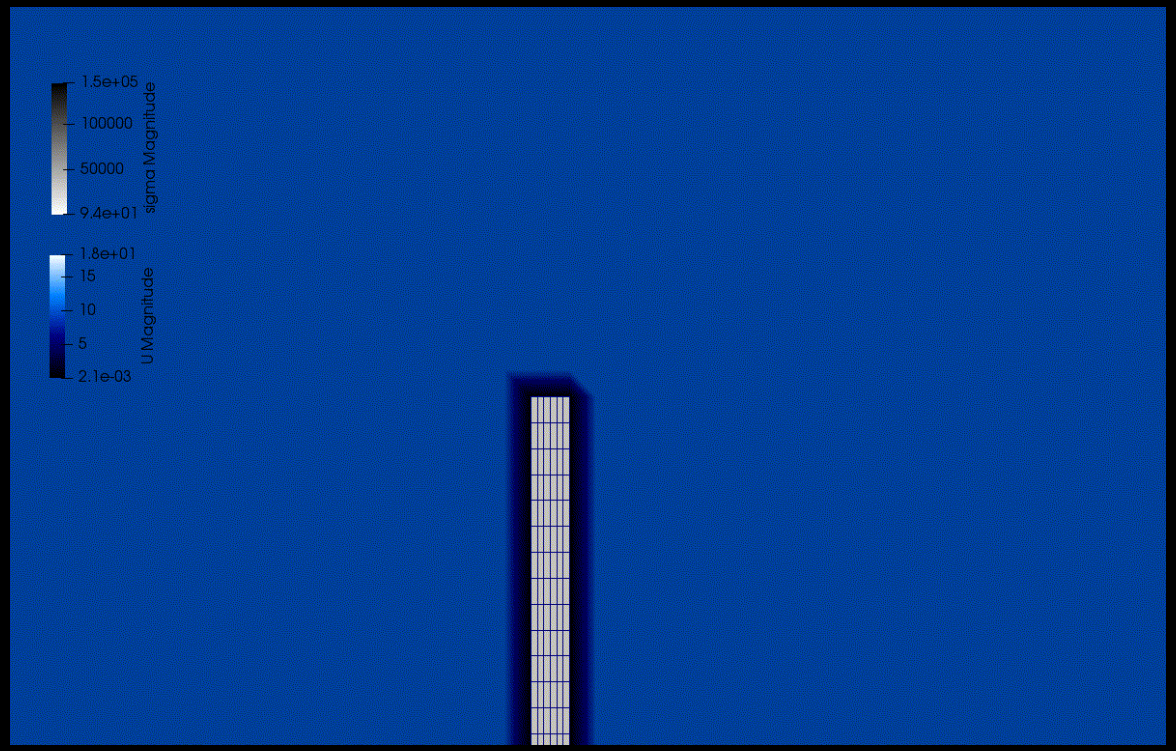
```
perpendicularFlap
├── 0
├── Allclean
├── Allrun
├── constant
└── system
    ├── controlDict
    ├── decomposeParDict
    ├── fluid
    ├── functions
    ├── functions.openfoam
    └── solid
        ├── controlDict
        ├── decomposeParDict
        ├── fvSchemes
        └── fvSolution
```

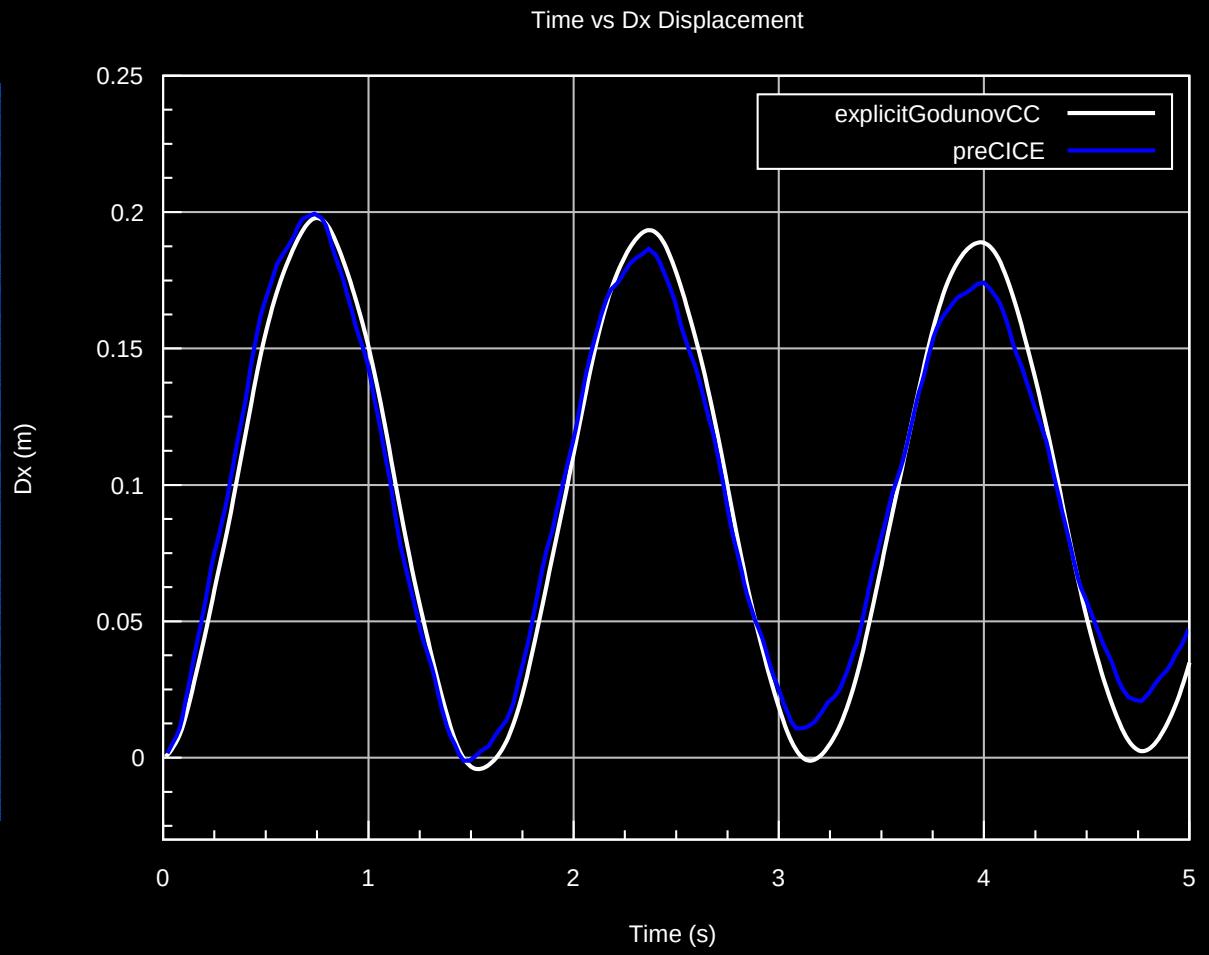
```
deltaT          0; //pseudo-time step

cfl              0.4;
```

5 Verification: perpendicular flap in cross flow



5 Verification: perpendicular flap in cross flow





Source: PreCICE, “Perpendicular flap tutorial using PreCICE,” 2019. [Online]. Available: https: //precice.org/tutorials-perpendicular-flap.html

Thank you!